DOTS
INTERNET-DRAFT
Intended Status: Standard Track
Expires: December 25, 2016

J. Francois
Inria
A. Lahmadi
University of Lorraine - LORIA
Marco Davids
SIDN Labs
Giovane C. M. Moura
SIDN Labs

June 23, 2016

IPv6 DOTS Signal Option
draft-francois-dots-ipv6-signal-option-00

Abstract

This document specifies an optional fall-back opportunistic method that employs the IPv6 Hop-by-Hop options extension header type. It allows a DOTS client to send a signaling message over a congested network due to a DDoS attack by ''tagging'' bypassing outgoing IPv6 packets to reach a DOTS server or relay.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Copyright and License Notice

Table of Contents

1 Introduction

1.1 Overview

A distributed denial-of-service (DDoS) attack aims at rendering machines or network resources unavailable. These attacks have grown in frequency, intensity and target diversity [I-D.draft-ietf-dots-requirements]. Moreover, several protocols have been utilized to amplify the intensity of the attacks [kuhrer2014exit], peaking at several hundred gigabits per second.

The DOTS aims at defining a common and open protocol to signal DDoS attacks to facilitate a coordinated response to these attacks. This document specifies a signalling mechanism that instead of designing a new application-layer protocol, it utilizes the IPv6 Hop-by-Hop header [RFC2460]. This header has the advantage to be fully inspected by all network devices and it is the first header in IPv6 extension headers [RFC7045].

The new option containing the attributes of the signalling message is included in an opportunistic way in available IPv6 packets leaving a network element until the message reaches a DOTS server. It thus constitutes an additional signalling channel but MUST NOT replace the original signalling channel used between DOTS client and servers as the one defined in [I-D.draft-reddy-dots-transport]. The DOTS client will thus embed the signalling attributes into outgoing IPv6 packets not necessarily going to the DOTS server. Intermediate routers receiving such a packet will examine it and embed the same information into other IPv6 packets. domain in this opportunistic way to increase the probability that such a packet will be finally forwarded to a DOTS Relay or Server, but also in controlled way to avoid that the mechanism is exploited for a malicious purposes.

Only the Hop-by-Hop options header allows such behavior and using Destination options header is not enough to make the DOTS signal going through the network in an opportunistic way. Each network element recognizing this new option will select the best fitted IPv6 packets to deliver the signal to the DOTS server or relay. For this reason the Hop-by-Hop header option is essential to make such behavior compared to other existing IPv6 extension headers [RFC6564].

1.2 Motivation

The traffic generated by a DDoS can be characterized according to various parameters, such as the layer (IP/ICMP or application), maximum and instant throughput, among others. Regardless its nature, we assume that for most cases, a DOTS client will be able to signal back one or few messages, during the attack, to the DOTS phase.

We have the same behavior in other DDoS attacks. For instance, on November 30th and December 1st, 2015, the Root DNS system was hit by an application layer (DNS) attack [rootops-ddos]. Each one of the 13 root server letters (A--M) was hit by attacks peaking at 5 million queries per second. By utilizing the RIPE Atlas DNSMON infrastructure, we can see that during the DDoS attacks, most of the root server letters remained reachable and able to respond to the DNS request sent by the probes employed by the DNSMON [ripe-dnsmon-ddos]. Few letters, however, had a packet loss rate of more than 99%. The DNSMON probes, however, experience mostly delays in their DNS requests instead.

Our signalling mechanism operates in an opportunistic way it is designed for DDoS as the ones on the Root DNS system.

1.3 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

The terms DOTS client, DOTS server, DOTS relay, DOTS agents, signal channel, DOTS signal and DOTS signal refers to the terminology introduced in [I-D.draft-ietf-dots-requirements].

The following terms are introduced:

Opportunistic DOTS signal:
an IPv6 packet containing the signalling attributes of an attack within the Hop-by-Hop extension header. The purpose is the same as the DOTS signal. It is used to request help for mitigating the attack.

DOTS opportunistic-capable router:
a router with the capacity to decode the opportunistic DOTS signal and re-embed such an information in other IPv6 packets.

All DOTS opportunistic-capable agents are defined as the DOTS agents supporting the opportunistic DOTS signal processing.

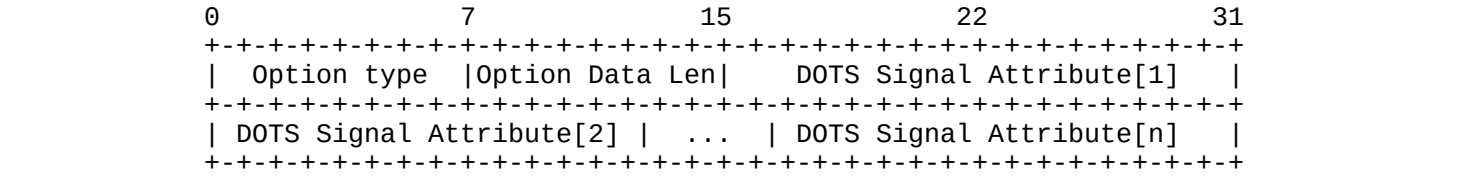
2. Opportunistic DOTS signal option

The goal is to provide an efficient mechanism where nodes in a IPv6 network facing a DDoS attack can deliver a DOTS signal message sent by a DOTS client to the DOTS server. The specified mechanism does not generate transport packets to carry the DOST signal message but it only relies on existing IPv6 packets in the

network to include inside them a hop-by-hop extension header which contains an encoded DOTS signal message. The solution defines a new IPv6 Hop-by-Hop header option with the semantic that the network node SHOULD include the option content within one or multiple outgoing IPv6 packets available in that network node.

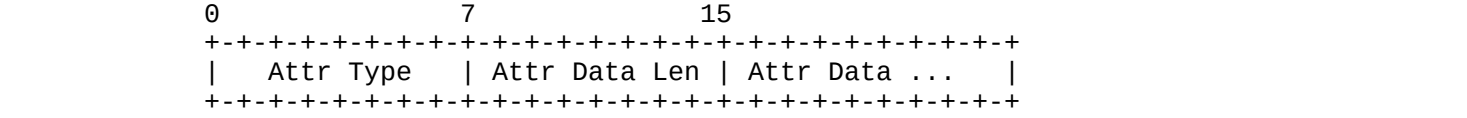
2.1 Hop-by-Hop option encoding

According to [RFC2460], options encoded into the IPv6 Hop-by-Hop header are formatted as Type-Length-Values (TLVs). The option for opportunistic DOTS signal is thus defined as follows:



The first byte defines the Hop-by-Hop Option type number allocated to the DOTS opportunistic signalling. This number is not yet fixed but the first three bits MUST be set to 0. The first two zero bits indicate that routers which cannot handle the DOTS signal option will continue to process other options. The third 0 bit means that the option processing will not change the packet’s final destination [RFC2460].

The second byte contains the length of the option content. The content of the DOTS Signal option is a variable-length field that contains one or more type-length-values (TLV) encoded DOTS signal attributes, and has the following format:



The Attr Type is 8-bit identifier of a DOTS signal attribute.

The Attr Data Len is 8-bit unsigned integer which is the length of Attr Data in bytes.

The Attr Data is variable-length field that contains the data of the attribute.

2.2 DOTS signal Option attributes

The first attribute embedded into the opportunistic DOTS signal is a TTL (Time-to-Live) field which indicates the maximum number of retransmission of the signal into another IPv6 packets until it MUST be discarded. Remaining attributes are similar to the header fields described in [I-D.draft-reddy-dots-transport] (section 5.1.1) used to convey a DOTS signal through a HTTP POST.

The sequence of attributes to be inserted within the header MUST start with fixed-length attributes which are defined in the following order:

TTL: Time-to-Live. This is a mandatory attribute encoded in one byte.

Flags: one byte is reserved for flags.
The first bit indicates the type of the IP address of the host: 0 for IPv4, 1 for IPv6. The second bit indicate if the protocol to use is TCP (1) or UDP (0). The third bit indicates if the message is signed The remaining bit are not used yet.

host: the IP address of the DOTS server where the signal option SHOULD be delivered. Depending on the flags, this field is encoded in 4 or 16 bytes.

port: the listening port of the DOTS server.
It is encoded in 2 bytes.

The remaining attributes MUST be TLV encoded, and they are defined in the following order:

policy-id: defined in [I-D.draft-reddy-dots-transport].

target-ip: defined in [I-D.draft-reddy-dots-transport].
However, each address or prefix is encoded in its own TLV element. The distinction between IPv4 and IPv6 is done over the length of the value.

target-port: defined in [I-D.draft-reddy-dots-transport].
However, each target port is encoded in its own TLV element.

target-protocol: defined in [I-D.draft-reddy-dots-transport].
However each target protocol is encoded in its own TLV element.

lifetime (lt): lifetime of the mitigation request defined in [I-D.draft-reddy-dots-transport].

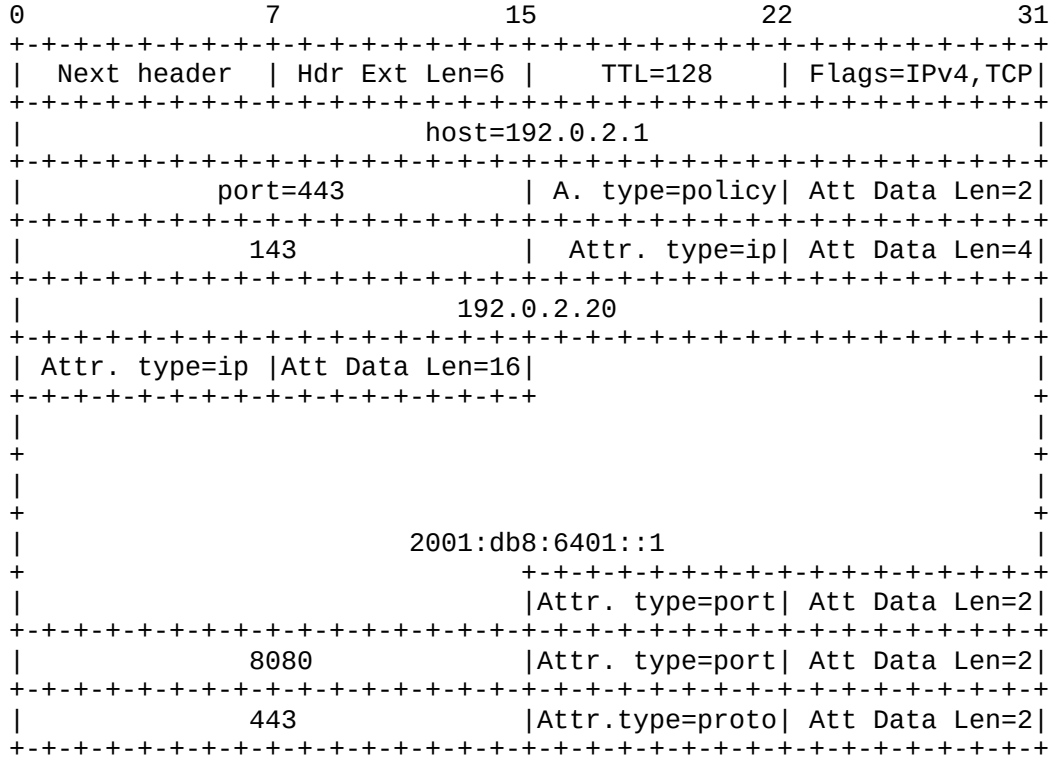
The encoded attributes MUST be included in the option header in the order defined above.

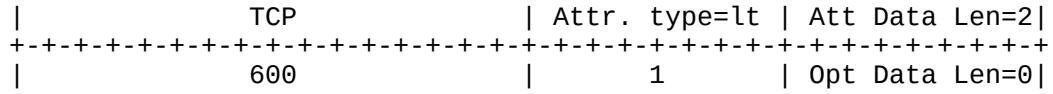
The following table provides the value of types that are used by the TLV encoded attributes.

Attribute type	value
policy-id	0
target-ip	1
target-port	2
target-protocol	3
lifetime	4

2.3 Example

Following is an example of an encoded Hop-by-Hop Option header to signal that a web service is under attack.





In the previous example, the message is not signed and terminates with padding. If it is the case, then the signature MUST BE added at the end such that the integrity and authenticity can be checked by the DOTS server or relay. The TTL attributes MUST be excluded from the signature calculation (see section 6).

3 Option Processing

3.2 Opportunistic DOTS signal initialization by a DOTS client

When a DOTS client needs to inform the DOTS server that it is under attack, it firstly makes a connection attempt and applies the mechanisms described in [I-D.draft-reddy-dots-transport].

In addition, it MAY activates an opportunistic mechanism to include the Hop-by-Hop header option specified in this document in one or multiple available IPv6 packets leaving the node.

The selection of packets has to be configured a priori. The configuration is composed of a sequence of rules defined in a hierarchical order such that they are triggered in a sequential manner.

Each rule is defined by:

- o a set of filters over the IPv6 packet headers. Only packets matching those filters are selected for opportunistic signalling. For instance, only packets heading to a given subnetwork or to specific address close to a DOTS server can be selected to increase the chance to reach the latter.

- o a ratio to select only a proportion of packets matching the filters in order to limit the induced overhead of the opportunistic signalling.

- o a timeout until the rule is active and selected IPv6 packets embed the DOTS opportunistic signal.

The objective is to apply each ordered rule after another according to their timeouts. The first rule is triggered immediately after the opportunistic signalling is activated.

Although the definition of rules MUST be configured by the user. It is RECOMMENDED to order them inversely related to the number of packets that would be selected. This can be approximated regarding the definition of filters. The core idea is to benefit from the first instants of the attack before losing connectivity by using a maximum number of outgoing packets to include the DOTS signalling option. It is thus RECOMMENDED to define the first as matching all IPv6 packets with a ratio equals one to rapidly disseminate the information but with a short timeout to limit the implied overhead.

Here is the an example of rules:

1: all outgoing IPv6 packets with a 10 second timeout
2: all outgoing IPv6 packets with a ratio of 10% and a 1 minute timeout
3: all outgoing multicast IPv6 packets with a ratio of 10% and a 1 minute timeout
4: all outgoing anycast IPv6 packets with a ratio of 10% and a 5 minute timeout
5: all outgoing IPv6 packets heading to the DOTS server with a ratio of 100% and a one hour timeout

3.2 Processing by a non DOTS opportunistic-capable router

When receiving an opportunistic DOTS signal encoded in a IPv6 packet, a non DOT opportunistic capable router simply skips the Hop-by-Hop option and continue the normal processing of the IPv6 packet because the option type MUST start with three zero bits.

3.3 Processing by a DOTS opportunistic-capable router

A DOTS opportunistic-capable router MUST store DOTS signalling information whose it is aware of. If a router processes an IPv6 DOTS opportunistic signal and supports this option, it first checks if it has already stored the associated information. In that case, the router simply skips the option and continues the normal processing otherwise it stores the encoded information in order to embed it again in other IPv6 packets similarly to the DOTS client. Hence, a set of rules are also defined in advance and are triggered upon the reception of a new opportunistic DOTS signal. Once all rule have been applied, signalling information MUST be discarded by the router. When embedding the information into other IPv6 packets, the router MUST decrease the TTL by one since opportunistic signalling does not prevent loops in the dissemination of signalling.

3.4 Processing by a DOTS opportunistic-capable relay

If a DOTS relay has DOTS capabilities, it will apply the same strategy as a DOTS client by making attempts of direct connections to the DOST server and in addition it inserts the Hop-by-Hop header DOTS signalling option in leaving IPv6 packets using the strategy specified above.

3.5 Processing by a DOTS opportunistic-capable server

When the IP layer of the host where the DOTS server is running receives an IPv6 packet carrying a Hop-by-Hop DOTS signal option header it MUST extracts the content of the option and provides the attributes data to the server program.

4 Deployment considerations

This mechanism will be potentially used by networks with IPv6 capable elements and requires that of IPv6 traffic exist in the network during the attack. The existing IPv6 traffic to be used could be of any type from management or user levels. It is also important to emphasize that while our mechanism utilizes an IPv6 header field, it can also be used to signal IPv4 attacks as well - given that the network devices are dual stacked.

IPv6 extension headers are often rate-limited or dropped entirely [HBH-HEADER]. To be able to use the mechanism specified in this document, network operators need to avoid discarding packets or ignoring the processing of the hop-by-hop option on their deployed network elements. However, instead of dropping or ignoring packets with hop-by-hop option carrying DOTS signal, they need to assign these packets to slow forwarding path, and be processed by the router’s CPU. This behavior will not affect the performance of the network devices since the network is already facing a DDoS attack and fast forwarding paths are saturated by the attacker traffic.

If the DOTS server, relay and the client are located in the same administrative domain, marking the IPv6 packets with the proposed hop-by-hop header option could be done in a straight forward way, while considering that an agreement exists inside the domain to avoid dropping or rate limiting of IPv6 extension headers as described above. The proposed mechanism becomes less practical and difficult to deploy when the DOST server is running on the Internet. In such scenario, the mechanism could be used in the intra-domain part to deliver the hop-by-hop option carrying the DOTS signal until it reaches a DOTS relay located in the same domain as the client, then the relay will apply mechanisms provided by the DOTS transport protocol [I-D.draft-reddy-dots-transport] to inform the server running on Internet about the attack. This deployment scenario

requires that at least one DOTS relay is deployed in the same domain than the DOTS client.

5 Impact on existing IP layer implementations

For this option to be applicable within an IP system, it requires modifications to existing IP layer implementation. At DOTS capable nodes (client, relay and server), it requires a service interface used by upper-layer protocols and application programs to ask the IP layer to insert and listen to the Hop-by-Hop header option in IPv6 packets with the content and strategies described in Section 3. A DOTS client invokes the service interface to insert the option, A DOTS relay invokes the service interface for listening and inserting the option, and finally a DOTS server only invokes the service interface to listen to the DOTS signalling option.

Intermediate nodes (routers or middle boxes) IP layer needs to be extended to perform processing of the new Hop-by-Hop header option as described in Section 3. They mainly parse the first host attribute of the option and make a selection of a leaving IPv6 packet where the option will be inserted.

Every node inserting the new proposed Hop-by-Hop option SHOULD only select IPv6 packets with enough left space to avoid fragmentation.

6 Security Considerations

Any IPv6 header option could be used by an attacker to create an attack on the routers and intermediate boxes that process packets containing the option. The proposed IPv6 option in this document MAY be abused by an attacker to create a covert channel at the IP layer where data is hidden inside the content of the option [RFC6564]. However, this attack is not specific to the proposed option and it is a known issue of IPv6 header extensions and options. The option MAY also be used by an attacker to forge or modify opportunistic DOTS signal leading to trigger additional processing on intermediate nodes and DOTS servers.

However the proposed option should be only initiated by a DOTS client and information embedded in new IPv6 messages by opportunistic DOTS capable routers. Defining proper policies to filter all messages with this option set and originated from other nodes would limit security issues since these DOTS opportunistic-capable agents SHOULD be trustworhy.

In addition, the message MAY be signed using techniques to enforce authenticity and integrity over the opportunistic DOTS signal

channel. The signalling message specification includes a flag to indicate if the message is signed by the choice of the signature algorithm is let to the users. This signature has to be computed by the DOTS opportunistic-capable client and checked by the DOTS opportunistic-capable relay or router. Hence, intermediate routers MUST NOT modify the message and its signature except the TTL, which so has not be considered during the signature computation.

Assuming a compromised router, the attacker could nevertheless replay the message or increase the TTL but thanks to the unique policy-id all intermediate-DOTS capable router will drop such messages and thus limiting their forwarding in the network.

Besides, an attacker can also listen opportunistic DOTS signals to monitor the impact of its own attack. These considerations are not specific to the proposed option and supposes that the attacker is able to compromise intermediate routers.

7 IANA Considerations

This draft defines a new IPv6 [RFC2460] hop-by-hop option. This requires an IANA RFC3692-style update of: http://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml and ultimately the assignment of a new hop-by-hop option according to the guidelines described in [RFC5237].

7 References

7.1 Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <http://www.rfc-editor.org/info/rfc2460>.

[RFC6564] Krishnan, S., Woodyatt, J., Kline, E., Hoagland, J., and M. Bhatia, "A Uniform Format for IPv6 Extension Headers", RFC 6564, DOI 10.17487/RFC6564, April 2012, <http://www.rfc-editor.org/info/rfc6564>.

[RFC7045] Carpenter, B. and S. Jiang, "Transmission and Processing of IPv6 Extension Headers", RFC 7045, DOI 10.17487/RFC7045, December 2013, <http://www.rfc-editor.org/info/rfc7045>.

7.2 Informative References

[I-D.draft-ietf-dots-requirements]
A. Mortensen., R. Moskowitz., and T. Reddy., "DDoS Open Threat Signaling Requirements", draft-ietf-dots-requirements-00 (work in progress), October 2015.

[kuhrer2014exit]
Kuhrer, Marc and Hupperich, Thomas and Rossow, Christian and Holz, Thorsten. Exit from Hell? Reducing the Impact of Amplification DDoS Attacks. In: 23rd USENIX Security Symposium (USENIX Security 14).

[I-D.draft-reddy-dots-transport]
T. Reddy., D. Wing., P. Patil., M. Geller., M. Boucadair.,and R. Moskowitz., "Co-operative DDoS Mitigation", draft-reddy-dots-transport-03 (work in progress), March 2016.

[rootops-ddos]
rootops.: Events of 2015-11-30. Online: http://root-servers.org/news/events-of-20151130.txt

[ripe-dnsmon-ddos]
RIPE NCC DNS Monitoring Service (DNSMON). Online: https://atlas.ripe.net/dnsmon/

[HBH-HEADER]
Baker, F., "IPv6 Hop-by-Hop Options Extension Header",Work in Progress, draft-ietf-6man-hbh-header-handling-03,March 2016.

Acknowledgements

This work is partly funded by FLAMINGO, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

Authors' Addresses

Jerome Francois
Inria
615 rue du Jardin Botanique
54600 Villers-les-Nancy
France

Phone: +33 3 83 59 30 66
EMail: jerome.francois@inria.fr

Abdelkader Lahmadi
University of Lorraine - LORIA
615 rue du Jardin Botanique
54600 Villers-les-Nancy
France

Phone: +33 3 83 59 30 00
Email: Abdelkader.Lahmadi@loria.fr

Marco Davids
SIDN Labs
Meander 501
6825 MD Arnhem
The Netherlands

Email: marco.davids@sidn.nl

Giovane C. M. Moura
SIDN Labs
Meander 501
6825 MD Arnhem
The Netherlands

Email: giovane.moura@sidn.nl

DOTS
Internet-Draft
Intended status: Standards Track
Expires: November 4, 2017

J. Francois
Inria
A. Lahmadi
University of Lorraine - LORIA
M. Davids
G. Moura
SIDN Labs

May 3, 2017

IPv6 DOTS Signal Option
draft-francois-dots-ipv6-signal-option-02

Abstract

DOTS client signal using original signal communication channel can expect service degradation and even service disruption as any other service over Internet but in more severe conditions because the signal may have to be transmitted over congested paths due to the denial-of-service attack.

This document specifies a fall-back asynchronous mechanism using an intermediate agent to store DOTS signal information during a limited period of time. This mechanism allows a DOTS server to request a signal information stored by a DOTS client when no heartbeat is received from the DOTS client. This intermediate agent called DOTS Signal Repository have to be connected to the DOTS client and server independently. The repository must be located and/or reached through one or multiple network paths, preferably as most as possible disjoint from regular signal channel, in order to increase its reachability. The document introduces a set of support protocols to build the asynchronous communication between the DOTS cient, server and the repository.

This Internet-Draft will expire on November 4, 2017.

Table of Contents

1. Introduction

1.1. Overview

A distributed denial-of-service (DDoS) attack aims at rendering machines or network resources unavailable. These attacks have grown in frequency, intensity and target diversity [I-D.ietf-dots-requirements]. Moreover, several protocols have been utilized to amplify the intensity of the attacks [kuhrer2014exit], peaking at several hundred gigabits per second.

DDoS Open Threat Signaling (DOTS) aims at defining a common and open protocol to signal DDoS attacks to facilitate a coordinated response to these attacks. This document specifies an asynchronous signaling method that MAY be used between a DOTS client and server instead of relying on purely synchronous communication as specified in [I-D.ietf-dots-signal-channel]. Indeed initial signaling should be done in real-time through connections between DOTS clients and severs such that a client can forward signal information as soon as the attack is detected. However, the signaling messages may have to be forwarded through paths impacted by the attack itself, i.e. highly congested. Asynchronous signaling in this document is an additional mechanism which MAY propagate signal information in a less reactive manner due to the use of an asynchronous communication channel but through alternative paths in the network. It increases the reachability of the DOTS server which will then in charge of requesting the mitigation.

The proposed mechanism constitutes an additional signaling channel but MUST NOT replace the original signaling channel used between DOTS client and servers as the one defined in [I-D.ietf-dots-signal-channel].

To perform asynchronous communication, this document introduces DOTS Signals Repository (DSR) which represents datastores where DOTS clients can send signal information. This information is then stored and the DOTS server can request it. In addition to provide a general process for achieving asynchronous signaling, this document introduces also a set of protocols which can support it.

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The terms DOTS client, DOTS server, DOTS gateway, DOTS agents refers to the terminology introduced in [I-D.ietf-dots-architecture].

The following terms are introduced:

- DSR (DOTS signal repository): intermediate agent able to store signal from clients during a limited period of time and which can be requested by DOTS servers.

- Opportunistic DOTS signal: an IPv6 packet containing the signaling attributes of an attack within the Hop-by-Hop extension header. The purpose is the same as the DOTS signal. It is used to request help for mitigating the attack.

- DOTS opportunistic-capable router: a router with the capacity to decode the opportunistic DOTS signal and re-embed such an information in other IPv6 packets.

- All DOTS opportunistic-capable agents are defined as the DOTS agents supporting the opportunistic DOTS signal processing.

2. Asynchronous DOTS signaling

2.1. Motivation

The traffic generated by a DDoS can be characterized according to various parameters, such as the layer (IP/ICMP or application), maximum and instant throughput, among others. Regardless its nature, we assume that for most cases, a DOTS client will be able to signal back one or few messages, during the attack, to the DOTS phase.

We have the same behavior in other DDoS attacks. For instance, on November 30th and December 1st, 2015, the Root DNS system was hit by an application layer (DNS) attack [rootops-ddos]. Each one of the 13 root server letters (A-M) was hit by attacks peaking at 5 million queries per second. By utilizing the RIPE Atlas DNSMON infrastructure, we can see that during the DDoS attacks, most of the root server letters remained reachable and able to respond to the DNS request sent by the probes employed by the DNSMON [ripe-dnsmon-ddos]. Few letters, however, had a packet loss rate of more than 99%. The DNSMON probes, however, experience mostly delays in their DNS requests instead.

As regular signaling from the DOTS client to the DOTS server or the DOTS gateway might be affected by the attack traffic, it is important to maximize the delivering success of the signals by using alternative packets and/or paths to deliver it. As a result, it forces to have intermediate agents, DSRs, able to catch DOTS signals delivered through those auxiliary mechanisms. However, those agents MUST not always forward DOTS signal to the server in order to limit the induced overhead. Only if the regular signal is not received by

the server, retrieving the signal from the DSRs is required, which has thus initiated by the DOTS server itself.

2.2. Architecture

DSR (DOTS Signal Repository) are additional REQUIRED datastores. They are integrated in the DOTS architecture [I-D.ietf-dots-architecture]as highlighted in Figure 1. (1) refers to the regular signaling. (2) and (3) refers to the proposed auxiliary mechanism. As shown in this figure, DSRs act as synchronizing agent where the DOTS client drops signal information (2) (attack details). On the contrary, the server will retrieve this information (3).

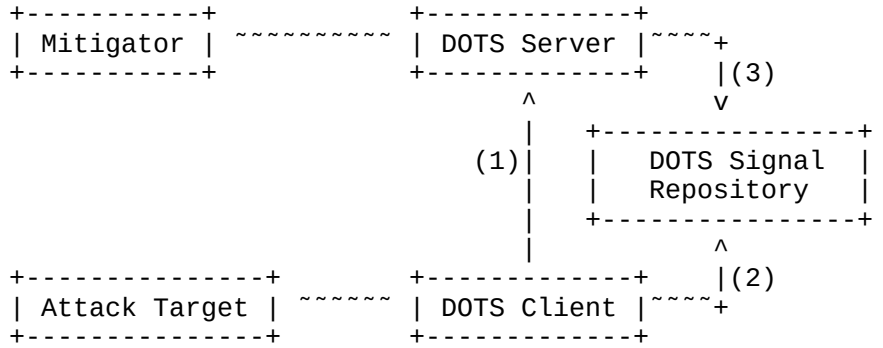


Figure 1: Asynchronous signaling

2.3. Asynchronous process

This section describes the process of asynchronous DOTS signal processing. In Figure Figure 1, there are two main communication channels which are not synchronized:

- Between DOTS client and DSR: the client sends DOTS signal information when a condition is satisfied. The condition MUST be configured by the user but SHOULD be linked to information provided by Attack target. For instance, when an attack is detected, the client connects to DOTS server and in parallel sends signal to one or more DSRs.

- Between DOTS server and DSR: the server will retrieve signal information when a specific condition occurs. Such a condition is linked with the probable occurrence of an attack. The server can infer this condition when the client is not responsive anymore. In [I-D.ietf-dots-signal-channel], an heartbeat mechanism is defined. Hence, when no heartbeat is received from the client, the server MUST try to get signal information by the asynchronous communication channel.

Each communication channel can implement its own protocol. They are NOT REQUIRED to be the same.

We have to note that the client condition to provide signals to the DSR can be weaker than regular synchronous signaling between DOTS client and server. Indeed, a client can signal to the DSR some suspect activities for which no mitigation is required yet. However, when the supposed attack is stronger provoking client disruption, the latter is not able to provide any type of signaling anymore and the server can thus retrieve information on prior stored signals.

2.4. Protocol requirements

DOTS signaling requirements are documented in [I-D.ietf-dots-requirements].

GEN-003 (Bidirectionality) requires that signal channel MUST enable asynchronous communications between DOTS agent by allowing unsolicited messages. Asynchronous signaling described in the current document allows DOTS client to provide signals, which can be retrieved later by DOTS server(s).

Because of this mechanism, there are requirements which are not supported: OP-002 (Session Health Monitoring), OP-003 (Session Redirection). Therefore, the fall-back asynchronous DOTS signaling is an additionnal mechanism and MUST NOT replace regular signaling as described in [I-D.ietf-dots-signal-channel].

It is particularly designed to fulfill GEN-002 (Resilience and Robustness) by increasing the signal delivery success even under the severely constrained network conditions imposed by particular attack traffic.

In addition, the fall-back asynchronous DOTS signal MUST specify a TTL (Time-to-Live) used by DSRs to store received signal in a limited period of time. It is different from mitigation lifetime but MUST be lower or equals.

3. DOTS Signal Repository

DSRs have to be deployed and distributed in order to enhance its reachability by the DOTS client. It is NOT REQUIRED that all DOTS agents use the same set of DSRs and a DOTS client and server SHOULD define their own set regarding their particular context, e.g. the network topology. The Data channel [I-D.ietf-dots-data-channel] between client and server MUST be used to configure it. As an example in the case of the inter-domain scenario, the DOTS server can inform the DOTS client to use DSRs scattered in multiple domains.

There is no restriction on the environment where DSRs can be deployed. Two types of DSRs are mainly considered:

- Routers: they have low capacity to process and store received signals but they are well distributed by nature in the network.
- Servers or stations: they provide higher computational power and storage but are less distributed

Selection of protocols to use for asynchronous signaling MUST take into account those specificities.

4. Protocol

4.1. Between DSR and DOTS server

To retrieve signals, the DOTS server MUST request the DSRs. Standard protocols can be used. Requests from the DOTS server MUST specify a client identifier and the DSRs returns all stored signal related to this client. The protocol MUST provide integrity and authenticity and SHOULD guarantee confidentiality. To limit entailed overhead lightweight protocol SHOULD be used. COAP [RFC7252] over DTLS [RFC6347] is RECOMMENDED when DSRs are servers. In the case of routers acting as DSR, network management protocol such as SNMP [RFC1157] or NETCONF [RFC6241] SHOULD be leveraged.

4.2. Between DSR and DOTS client

The signal sent by the DOTS client to the DSR is more prone to be affected by attack traffic due to its proximity to the attack victim. Similar protocol as between DSR and DOTS server can be used but it is RECOMMENDED to convey the signal over multiple paths to increase the reachability sucess

This document introduces two mechanisms to deliver the signal from the DOTS client to the DSR: IPv6 opportunistic signaling using Hop-by-Hop Option header; source routing using IPv6 Segment Routing Header (SRH).

4.2.1. Opportunistic DOTS signaling

This section specifies a signalling mechanism that instead of designing a new application-layer protocol, it utilizes the IPv6 Hop-by-Hop header [RFC2460]. This header SHOULD be processed by intermediate devices and it MUST be the first header in IPv6 extension headers [RFC7045].

In such a particular scenario, DSRs are intermediate routers capable of processing the option. DOTS server MAY also receive IPV6 packets with the Hop-by-Hop option and could thus process it directly. It is till considered as asynchronous since the server MAY NOT receive the initial packet emitted by the client but a copy of the signal in another packet (done by an intermediate DSR/router). Otherwise, it MUST connect to the DSR (section Section 4.1)

The new option containing the attributes of the signalling message is included in an opportunistic way in available IPv6 packets leaving a network element. The DOTS client will thus embed the signalling attributes into outgoing IPv6 packets not necessarily going to the DOTS server. Intermediate routers receiving such a packet will examine it and embed the same information into other IPv6 packets. domain in this opportunistic way to increase the probability that such a packet will be finally forwarded to a DOTS gateway or Server, but also in controlled way to avoid that the mechanism is exploited for a malicious purposes.

Only the Hop-by-Hop options header allows such behavior and using Destination options header is not enough to make the DOTS signal going through the network in an opportunistic way. Each network element recognizing this new option will select the best fitted IPv6 packets to deliver the signal to the DOTS DSRs. For this reason the Hop-by-Hop header option is essential to make such behavior compared to other existing IPv6 extension headers [RFC6564].

The goal is to provide an efficient mechanism where nodes in a IPv6 network facing a DDoS attack can deliver a DOTS signal message sent by a DOTS client to the DOTS server. The specified mechanism does not generate transport packets to carry the DOST signal message but it only relies on existing IPv6 packets in the network to include inside them a hop-by-hop extension header which contains an encoded DOTS signal message. The solution defines a new IPv6 Hop-by-Hop header option with the semantic that the network node SHOULD include

the option content within one or multiple outgoing IPv6 packets available in that network node.

4.2.1.1. Hop-by-Hop option encoding

According to [RFC2460], options encoded into the IPv6 Hop-by-Hop header are formatted as Type-Length-Values (TLVs). The option for opportunistic DOTS signal is thus defined as described in Figure 2

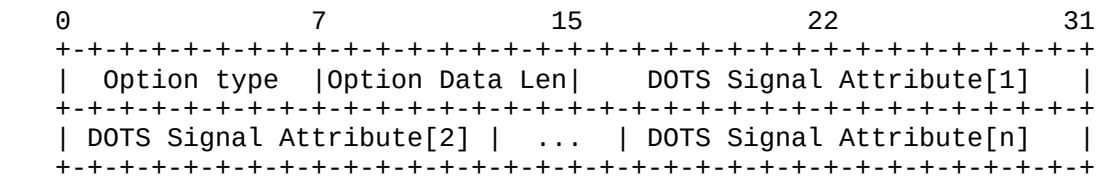


Figure 2: Hop-by-Hop option encoding

The first byte defines the Hop-by-Hop Option type number allocated to the DOTS opportunistic signalling. This number is not yet fixed but the first three bits MUST be set to 0. The first two zero bits indicate that routers which cannot handle the DOTS signal option will continue to process other options. The third 0 bit means that the option processing will not change the packet's final destination [RFC2460].

The second byte contains the length of the option content. The content of the DOTS Signal option is a variable-length field that contains one or more type-length-values (TLV) encoded DOTS signal attributes, and has the format described in Figure 3.

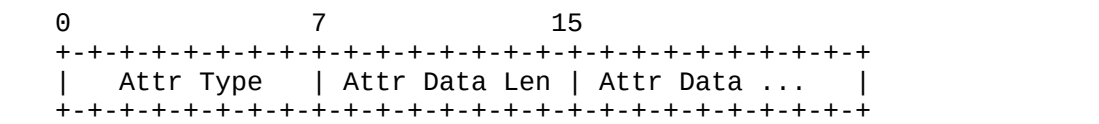


Figure 3: Hop-by-Hop option encoding

The Attr Type is 8-bit identifier of a DOTS signal attribute.

The Attr Data Len is 8-bit unsigned integer which is the length of Attr Data in bytes.

The Attr Data is variable-length field that contains the data of the attribute.

Since using TLVs in Hop-by-Hop options is known to be a factor of attacks [I-D.krishnan-ipv6-hopbyhop], DOTS attributes are encoded with fixed length when possible.

4.2.1.2. DOTS signal Option attributes

The first attribute embedded into the opportunistic DOTS signal is a TTL (Time-to-Live) field which indicates the maximum number of retransmission of the signal into another IPv6 packets until it MUST be discarded. Remaining attributes are similar to the header fields described in [I-D.ietf-dots-signal-channel] used to convey a DOTS signal through a HTTP POST.

The sequence of attributes to be inserted within the header MUST start with fixed-length attributes which are defined in the following order:

- TTL: Time-to-Live. This is a mandatory attribute encoded in one byte.
- Flags: one byte is reserved for flags.
  The first bit indicates the type of the IP address of the host: 0 for IPv4, 1 for IPv6. The second bit indicate if the protocol to use is TCP (1) or UDP (0). The third bit indicates if the message is signed. The remaining bit are not used yet.
- host: the IP address of the DOTS server where the signal option SHOULD be delivered. Depending on the flags, this field is encoded in 4 or 16 bytes.
- port: the listening port of the DOTS server. It is encoded in 2 bytes.

The remaining attributes MUST be TLV encoded, and they are defined in the following order:

- policy-id: defined in [I-D.ietf-dots-signal-channel].
- target-ip: defined in [I-D.ietf-dots-signal-channel].
  However, each address or prefix is encoded in its own TLV element. The distinction between IPv4 and IPv6 is done over the length of the value.
- target-port: defined in [I-D.ietf-dots-signal-channel].
  However, each target port is encoded in its own TLV element.
- target-protocol: defined in [I-D.ietf-dots-signal-channel]
  However each target protocol is encoded in its own TLV element.

o lifetime (lt): lifetime of the mitigation request defined in [I-D.ietf-dots-signal-channel]

The encoded attributes MUST be included in the option header in the order defined above.

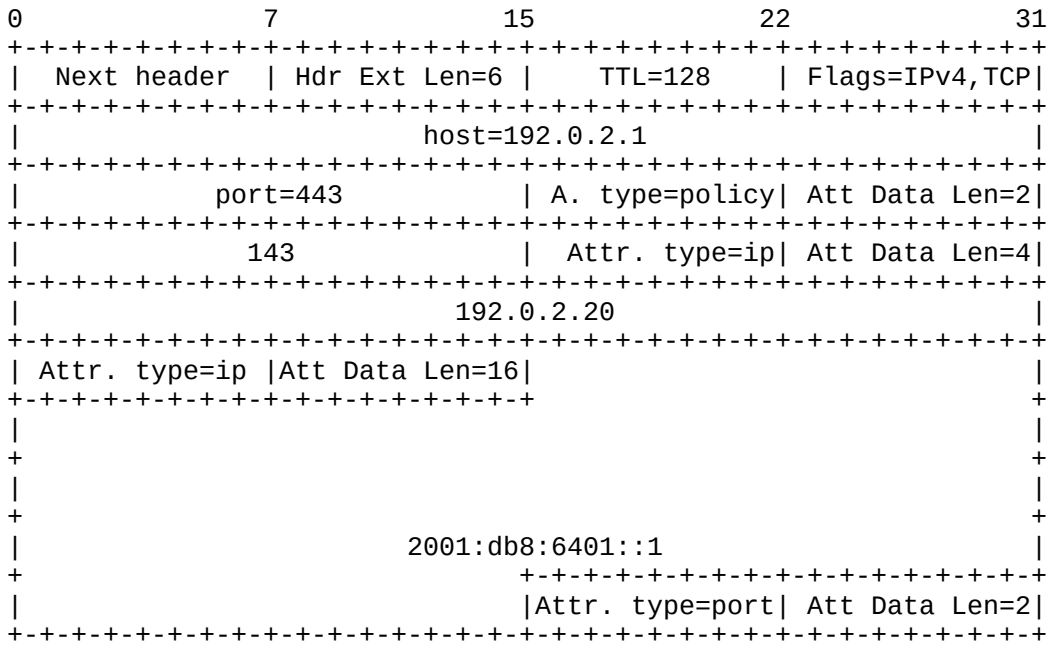
Table 1 provides the value of types that are used by the TLV encoded attributes.

Attribute type	Value
policy-id	0
target-ip	1
target-port	2
target-protocol	3
lifetime	4

Table 1: TLV encoded attributes types

4.2.1.3. Example

Following is an example of an encoded Hop-by-Hop Option header to signal that a web service is under attack.



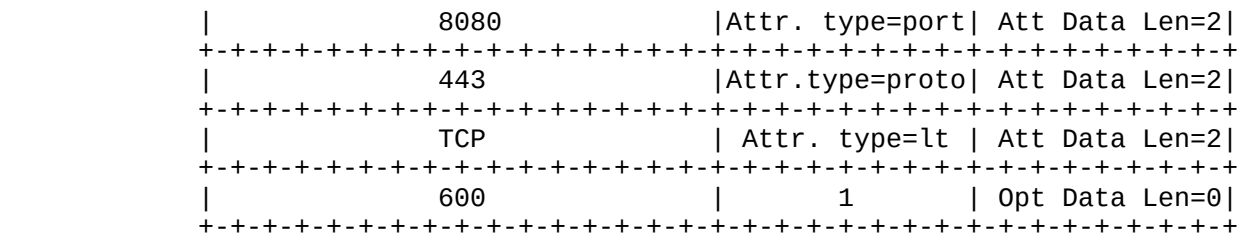


Figure 4: Example of Hop-by-Hop DOTS signal encoding

In the previous example, the message is not signed and terminates with padding. If it is the case, then the signature MUST BE added at the end such that the integrity and authenticity can be checked by the DOTS server or gateway. The TTL attributes MUST be excluded from the signature calculation.

4.2.1.4. Option Processing

4.2.1.4.1. Opportunistic DOTS signal initialization by a DOTS client

When a DOTS client needs to inform the DOTS server that it is under attack, it firstly makes a connection attempt and applies the mechanisms described in [I-D.ietf-dots-signal-channel].

In addition, it MAY activates an opportunistic mechanism to include the Hop-by-Hop header option specified in this document in one or multiple available IPv6 packets leaving the node. Because the DOTS client location is independent of the signalling, it can be positionned in a part of the network where there is no passing-by traffic which can serve for opportunistic signalling. DOTS client MAY also create and emit IPv6 datagrams without payload but with the signal encoded in the Hop-by-Hop option header.

Otherwise, the selection of packets has to be configured a priori. The configuration is composed of a sequence of rules defined in a hierarchical order such that they are triggered in a sequential manner.

The selection of packets has to be configured a priori. The configuration is composed of a sequence of rules defined in a hierarchical order such that they are triggered in a sequential manner.

Each rule is defined by:

- a set of filters over the IPv6 packet headers. Only packets matching those filters are selected for opportunistic signalling.

For instance, only packets heading to a given subnetwork or to specific address close to a DOTS server can be selected to increase the chance to reach the latter.

- o a ratio to select only a proportion of packets matching the filters in order to limit the induced overhead of the opportunistic signalling.

- o a timeout until the rule is active and selected IPv6 packets embed the DOTS opportunistic signal.

The objective is to apply each ordered rule after another according to their timeouts. The first rule is triggered immediately after the opportunistic signalling is activated.

In all cases (embedding information into an exsiting packet or creating an new pakket with no payload), the client MUST avoid fragmentation.

Although the definition of rules MUST be configured by the user. It is RECOMMENDED to order them inversely related to the number of packets that would be selected. This can be approximated regarding the definition of filters. The core idea is to benefit from the first instants of the attack before losing connectivity by using a maximum number of outgoing packets to include the DOTS signalling option. It is thus RECOMMENDED to define the first as matching all IPv6 packets with a ratio equals one to rapidly disseminate the information but with a short timeout to limit the implied overhead.

Here is the an example of rules:

1. all outgoing IPv6 packets with a 10 second timeout

2. all outgoing IPv6 packets with a ratio of 10% and a 1 minute timeout

3. all outgoing multicast IPv6 packets with a ratio of 10% and a 1 minute timeout

4. all outgoing IPv6 packets heading to the DOTS server with a ratio of 100% and a one hour timeout

4.2.1.4.2. Processing by a non DOTS opportunistic-capable router

When receiving an opportunistic DOTS signal encoded in a IPv6 packet, a non DOT opportunistic capable router simply skips the Hop-by-Hop option and continue the normal processing of the IPv6 packet because the option type MUST start with three zero bits.

4.2.1.4.3. Processing by a DOTS opportunistic-capable router

A DOTS opportunistic-capable router MUST store DOTS signalling information whose it is aware of. If a router processes an IPv6 DOTS opportunistic signal and supports this option, it first checks if it has already stored the associated information. In that case, the router simply skips the option and continues the normal processing otherwise it stores the encoded information in order to embed it again in other IPv6 packets similarly to the DOTS client. Hence, a set of rules are also defined in advance and are triggered upon the reception of a new opportunistic DOTS signal. Once all rule have been applied, signalling information MUST be discarded by the router. When embedding the information into other IPv6 packets, the router MUST decrease the TTL by one since opportunistic signalling does not prevent loops in the dissemination of signalling.

4.2.1.4.4. Processing by a DOTS opportunistic-capable gateway

If a DOTS gateway has DOTS capabilities, it will apply the same strategy as a DOTS client by making attempts of direct connections to the DOST server and in addition it inserts the Hop-by-Hop header DOTS signalling option in leaving IPv6 packets using the strategy specified above.

4.2.1.4.5. Processing by a DOTS opportunistic-capable server

When the IP layer of the host where the DOTS server is running receives an IPv6 packet carrying a Hop-by-Hop DOTS signal option header it MUST extracts the content of the option and provides the attributes data to the server program.

4.2.1.5. Deployment considerations

This mechanism will be potentially used by networks with IPv6 capable elements and requires that of IPv6 traffic exist in the network during the attack. The existing IPv6 traffic to be used could be of any type from management or user levels. It is also important to emphasize that while our mechanism utilizes an IPv6 header field, it can also be used to signal IPv4 attacks as well - given that the network devices are dual stacked.

IPv6 extension headers are often rate-limited or dropped entirely. To be able to use the mechanism specified in this document, network operators need to avoid discarding packets or ignoring the processing of the hop-by-hop option on their deployed network elements. However, instead of dropping or ignoring packets with hop-by-hop option carrying DOTS signal, they need to assign these packets to slow forwarding path, and be processed by the router’s CPU. This

behavior will not affect the performance of the network devices since the network is already facing a DDoS attack and fast forwarding paths are saturated by the attacker traffic.

If the DOTS server, gateway and the client are located in the same administrative domain, marking the IPv6 packets with the proposed hop-by-hop header option could be done in a straight forward way, while considering that an agreement exists inside the domain to avoid dropping or rate limiting of IPv6 extension headers as described above. The proposed mechanism becomes less practical and difficult to deploy when the DOST server is running on the Internet. In such scenario, the mechanism could be used in the intra-domain part to deliver the hop-by-hop option carrying the DOTS signal until it reaches a DOTS gateway located in the same domain as the client, then the gateway will apply mechanisms provided by the DOTS transport protocol [I-D.ietf-dots-signal-channel] to inform the server running on Internet about the attack. This deployment scenario requires that at least one DOTS gateway is deployed in the same domain than the DOTS client.

4.2.1.6. Impact on existing IP layer implementations

For this option to be applicable within an IP system, it requires modifications to existing IP layer implementation. At DOTS capable nodes (client, gateway and server), it requires a service interface used by upper-layer protocols and application programs to ask the IP layer to insert and listen to the Hop-by-Hop header option in IPv6 packets. A DOTS client invokes the service interface to insert the option, A DOTS gateway invokes the service interface for listening and inserting the option, and finally a DOTS server only invokes the service interface to listen to the DOTS signalling option.

Intermediate nodes (routers or middle boxes) IP layer needs to be extended to perform processing of the new Hop-by-Hop header option. They mainly parse the first host attribute of the option and make a selection of a leaving IPv6 packet where the option will be inserted.

Every node inserting the new proposed Hop-by-Hop option SHOULD only select IPv6 packets with enough left space to avoid fragmentation.

4.2.2. IPv6 SRH

DOTS signalling may be carried using IPv6 source routing header. Details will be provided in a later version of this document.

5. Security Considerations

Any IPv6 header option could be used by an attacker to create an attack on the routers and intermediate boxes that process packets containing the option. The proposed IPv6 option in this document MAY be abused by an attacker to create a covert channel at the IP layer where data is hidden inside the content of the option [RFC6564]. However, this attack is not specific to the proposed option and it is a known issue of IPv6 header extensions and options. The option MAY also be used by an attacker to forge or modify opportunistic DOTS signal leading to trigger additional processing on intermediate nodes and DOTS servers.

However the proposed option should be only initiated by a DOTS client and information embedded in new IPv6 messages by opportunistic DOTS capable routers. Defining proper policies to filter all messages with this option set and originated from other nodes would limit security issues since these DOTS opportunistic-capable agents SHOULD be trustworhy.

In addition, the message MAY be signed using techniques to enforce authenticity and integrity over the opportunistic DOTS signal channel. The signalling message specification includes a flag to indicate if the message is signed by the choice of the signature algorithm is let to the users. This signature has to be computed by the DOTS opportunistic-capable client and checked by the DOTS opportunistic-capable gateway or router. Hence, intermediate routers MUST NOT modify the message and its signature except the TTL, which so has not be considered during the signature computation.

Assuming a compromised router, the attacker could nevertheless replay the message or increase the TTL but thanks to the unique policy-id all intermediate-DOTS capable router will drop such messages and thus limiting their forwarding in the network.

Besides, an attacker can also listen opportunistic DOTS signals to monitor the impact of its own attack. These considerations are not specific to the proposed option and supposes that the attacker is able to compromise intermediate routers.

6. IANA Considerations

This draft defines a new IPv6 hop-by-hop option[RFC2460].This requires an IANA RFC3692-style update of:http://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml and ultimately the assignment of a new hop-by-hop option according to the guidelines described in [RFC5237].

7. Acknowledgements

This work is partly funded by FLAMINGO, a Network of Excellence Seventh Framework Programme.

8. References

8.1. Normative References

[I-D.ietf-dots-architecture]
Mortensen, A., Andreasen, F., Reddy, T., christopher_gray3@cable.comcast.com, c., Compton, R., and N. Teague, "Distributed-Denial-of-Service Open Threat Signaling (DOTS) Architecture", draft-ietf-dots-architecture-01 (work in progress), October 2016.

[I-D.ietf-dots-data-channel]
Reddy, T., Boucadair, M., Nishizuka, K., Xia, L., Patil, P., Mortensen, A., and N. Teague, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Data Channel", draft-ietf-dots-data-channel-00 (work in progress), April 2017.

[I-D.ietf-dots-requirements]
Mortensen, A., Moskowitz, R., and T. Reddy, "Distributed Denial of Service (DDoS) Open Threat Signaling Requirements", draft-ietf-dots-requirements-04 (work in progress), March 2017.

[I-D.ietf-dots-signal-channel]
Reddy, T., Boucadair, M., Patil, P., Mortensen, A., and N. Teague, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel", draft-ietf-dots-signal-channel-01 (work in progress), April 2017.

[I-D.krishnan-ipv6-hopbyhop]
Krishnan, S., "The case against Hop-by-Hop options", draft-krishnan-ipv6-hopbyhop-05 (work in progress), October 2010.

8.2. Informative References

[RFC1157] Case, J., Fedor, M., Schoffstall, M., and J. Davin, "Simple Network Management Protocol (SNMP)", RFC 1157, DOI 10.17487/RFC1157, May 1990, <http://www.rfc-editor.org/info/rfc1157>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <http://www.rfc-editor.org/info/rfc2460>.

[RFC5237] Arkko, J. and S. Bradner, "IANA Allocation Guidelines for the Protocol Field", BCP 37, RFC 5237, DOI 10.17487/RFC5237, February 2008, <http://www.rfc-editor.org/info/rfc5237>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC6564] Krishnan, S., Woodyatt, J., Kline, E., Hoagland, J., and M. Bhatia, "A Uniform Format for IPv6 Extension Headers", RFC 6564, DOI 10.17487/RFC6564, April 2012, <http://www.rfc-editor.org/info/rfc6564>.

[RFC7045] Carpenter, B. and S. Jiang, "Transmission and Processing of IPv6 Extension Headers", RFC 7045, DOI 10.17487/RFC7045, December 2013, <http://www.rfc-editor.org/info/rfc7045>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[kuhrer2014exit] M. Kuhrer, T. Hupperich, C. Rossow, T. Holz, "Exit from Hell? Reducing the Impact of Amplification DDoS Attacks", USENIX Security Symposium 23rd, 2014.

[ripe-dnsmon-ddos] RIPE, "NCC DNS Monitoring Service (DNSMON)", <https://atlas.ripe.net/dnsmon/>.

[rootops-ddos] rootops., "Events of 2015-11-30", 2015, <http://root-servers.org/news/events-of-20151130.txt>.

Appendix A. Additional Stuff

This becomes an Appendix.

Authors' Addresses

Jerome Francois
Inria
615 rue du jardin botanique
Villers-les-Nancy 54600
FR

Phone: +33 3 83 59 30 66
Email: jerome.francois@inria.fr

Abdelkader Lahmadi
University of Lorraine - LORIA
615 rue du jardin botanique
Villers-les-Nancy 54600
FR

Phone: +33 3 83 59 30 00
Email: Abdelkader.Lahmadi@loria.fr

Marco Davids
SIDN Labs
Meander 501
Arnhem 6825 MD
NL

Email: marco.davids@sidn.nl

Giovane Moura
SIDN Labs
Meander 501
Arnhem 6825 MD
NL

Email: marco.davids@sidn.nl

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 6, 2016

R. Barnes
Mozilla
J. Hoffman-Andrews
EFF
J. Kasten
University of Michigan

October 04, 2015

Automatic Certificate Management Environment (ACME)
draft-ietf-acme-acme-01

Abstract

Certificates in the Web's X.509 PKI (PKIX) are used for a number of purposes, the most significant of which is the authentication of domain names. Thus, certificate authorities in the Web PKI are trusted to verify that an applicant for a certificate legitimately represents the domain name(s) in the certificate. Today, this verification is done through a collection of ad hoc mechanisms. This document describes a protocol that a certificate authority (CA) and an applicant can use to automate the process of verification and certificate issuance. The protocol also provides facilities for other certificate management functions, such as certificate revocation.

DANGER: Do not implement this specification. It has a known signature reuse vulnerability. For details, see the following discussion:

https://mailarchive.ietf.org/arch/msg/acme/F71iz6qq1o_QPVhJCV4dqWf-4Yc

Status of This Memo

This Internet-Draft will expire on April 6, 2016.

Table of Contents

1. Introduction

Certificates in the Web PKI are most commonly used to authenticate domain names. Thus, certificate authorities in the Web PKI are trusted to verify that an applicant for a certificate legitimately represents the domain name(s) in the certificate.

Existing Web PKI certificate authorities tend to run on a set of ad hoc protocols for certificate issuance and identity verification. A typical user experience is something like:

- Generate a PKCS#10 [RFC2314] Certificate Signing Request (CSR).
- Cut-and-paste the CSR into a CA web page.
- Prove ownership of the domain by one of the following methods:
  - Put a CA-provided challenge at a specific place on the web server.
  - Put a CA-provided challenge at a DNS location corresponding to the target domain.
  - Receive CA challenge at a (hopefully) administrator-controlled e-mail address corresponding to the domain and then respond to it on the CA's web page.
- Download the issued certificate and install it on their Web Server.

With the exception of the CSR itself and the certificates that are issued, these are all completely ad hoc procedures and are accomplished by getting the human user to follow interactive natural-language instructions from the CA rather than by machine-implemented

published protocols. In many cases, the instructions are difficult to follow and cause significant confusion. Informal usability tests by the authors indicate that webmasters often need 1-3 hours to obtain and install a certificate for a domain. Even in the best case, the lack of published, standardized mechanisms presents an obstacle to the wide deployment of HTTPS and other PKIX-dependent systems because it inhibits mechanization of tasks related to certificate issuance, deployment, and revocation.

This document describes an extensible framework for automating the issuance and domain validation procedure, thereby allowing servers and infrastructural software to obtain certificates without user interaction. Use of this protocol should radically simplify the deployment of HTTPS and the practicality of PKIX authentication for other protocols based on TLS [RFC5246].

2. Deployment Model and Operator Experience

The major guiding use case for ACME is obtaining certificates for Web sites (HTTPS [RFC2818]). In that case, the server is intended to speak for one or more domains, and the process of certificate issuance is intended to verify that the server actually speaks for the domain.

Different types of certificates reflect different kinds of CA verification of information about the certificate subject. "Domain Validation" (DV) certificates are by far the most common type. For DV validation, the CA merely verifies that the requester has effective control of the web server and/or DNS server for the domain, but does not explicitly attempt to verify their real-world identity. (This is as opposed to "Organization Validation" (OV) and "Extended Validation" (EV) certificates, where the process is intended to also verify the real-world identity of the requester.)

DV certificate validation commonly checks claims about properties related to control of a domain name - properties that can be observed by the issuing authority in an interactive process that can be conducted purely online. That means that under typical circumstances, all steps in the request, verification, and issuance process can be represented and performed by Internet protocols with no out-of-band human intervention.

When an operator deploys a current HTTPS server, it generally prompts him to generate a self-signed certificate. When an operator deploys an ACME-compatible web server, the experience would be something like this:

- The ACME client prompts the operator for the intended domain name(s) that the web server is to stand for.
- The ACME client presents the operator with a list of CAs from which it could get a certificate. (This list will change over time based on the capabilities of CAs and updates to ACME configuration.) The ACME client might prompt the operator for payment information at this point.
- The operator selects a CA.
- In the background, the ACME client contacts the CA and requests that a certificate be issued for the intended domain name(s).
- Once the CA is satisfied, the certificate is issued and the ACME client automatically downloads and installs it, potentially notifying the operator via e-mail, SMS, etc.
- The ACME client periodically contacts the CA to get updated certificates, stapled OCSP responses, or whatever else would be required to keep the server functional and its credentials up-to-date.

The overall idea is that it’s nearly as easy to deploy with a CA-issued certificate as a self-signed certificate, and that once the operator has done so, the process is self-sustaining with minimal manual intervention. Close integration of ACME with HTTPS servers, for example, can allow the immediate and automated deployment of certificates as they are issued, optionally sparing the human administrator from additional configuration work.

3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

The two main roles in ACME are "client" and "server". The ACME client uses the protocol to request certificate management actions, such as issuance or revocation. An ACME client therefore typically runs on a web server, mail server, or some other server system which requires valid TLS certificates. The ACME server runs at a certificate authority, and responds to client requests, performing the requested actions if the client is authorized.

For simplicity, in all HTTPS transactions used by ACME, the ACME client is the HTTPS client and the ACME server is the HTTPS server.

In the discussion below, we will refer to three different types of keys / key pairs:

Subject Public Key: A public key to be included in a certificate.

Account Key Pair: A key pair for which the ACME server considers the holder of the private key authorized to manage certificates for a given identifier. The same key pair may be authorized for multiple identifiers.

Recovery Key: A MAC key that a client can use to demonstrate that it participated in a prior registration transaction.

ACME messaging is based on HTTPS [RFC2818] and JSON [RFC7159]. Since JSON is a text-based format, binary fields are Base64-encoded. For Base64 encoding, we use the variant defined in [RFC7515]. The important features of this encoding are (1) that it uses the URL-safe character set, and (2) that "=" padding characters are stripped.

Some HTTPS bodies in ACME are authenticated and integrity-protected by being encapsulated in a JSON Web Signature (JWS) object [RFC7515]. ACME uses a profile of JWS, with the following restrictions:

- The JWS MUST use the Flattened JSON Serialization
- The JWS MUST be encoded using UTF-8
- The JWS Header or Protected Header MUST include "alg" and "jwk" fields
- The JWS MUST NOT have the value "none" in its "alg" field

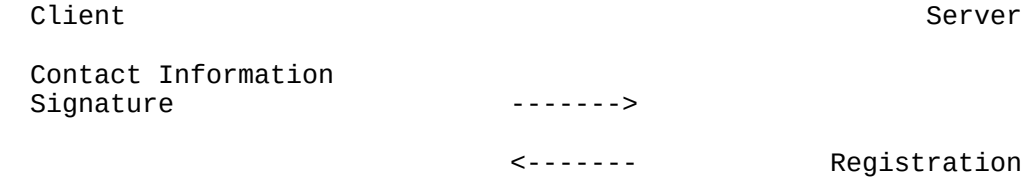
Additionally, JWS objects used in ACME MUST include the "nonce" header parameter, defined below.

4. Protocol Overview

ACME allows a client to request certificate management actions using a set of JSON messages carried over HTTPS. In some ways, ACME functions much like a traditional CA, in which a user creates an account, adds identifiers to that account (proving control of the domains), and requests certificate issuance for those domains while logged in to the account.

In ACME, the account is represented by an account key pair. The "add a domain" function is accomplished by authorizing the key pair for a given domain. Certificate issuance and revocation are authorized by a signature with the key pair.

The first phase of ACME is for the client to register with the ACME server. The client generates an asymmetric key pair and associates this key pair with a set of contact information by signing the contact information. The server acknowledges the registration by replying with a registration object echoing the client's input.

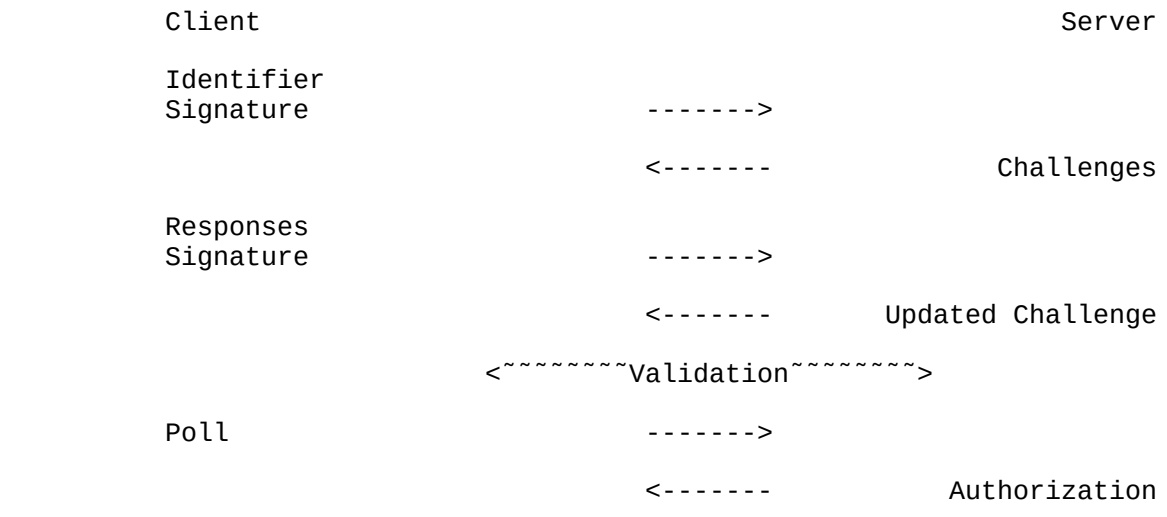


Before a client can issue certificates, it must establish an authorization with the server for an account key pair to act for the identifier(s) that it wishes to include in the certificate. To do this, the client must demonstrate to the server both (1) that it holds the private key of the account key pair, and (2) that it has authority over the identifier being claimed.

Proof of possession of the account key is built into the ACME protocol. All messages from the client to the server are signed by the client, and the server verifies them using the public key of the account key pair.

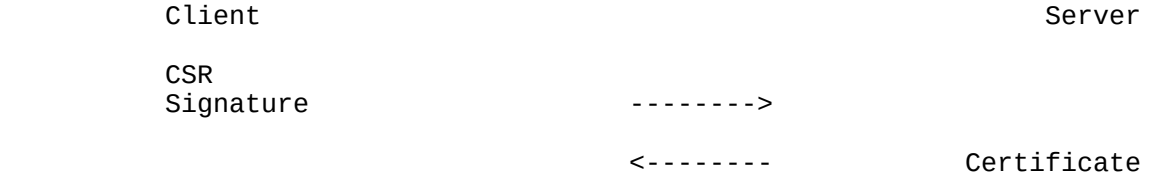
To verify that the client controls the identifier being claimed, the server issues the client a set of challenges. Because there are many different ways to validate possession of different types of identifiers, the server will choose from an extensible set of challenges that are appropriate for the identifier being claimed. The client responds with a set of responses that tell the server which challenges the client has completed. The server then validates the challenges to check that the client has accomplished the challenge.

For example, if the client requests a domain name, the server might challenge the client to provision a record in the DNS under that name, or to provision a file on a web server referenced by an A or AAAA record under that name. The server would then query the DNS for the record in question, or send an HTTP request for the file. If the client provisioned the DNS or the web server as expected, then the server considers the client authorized for the domain name.

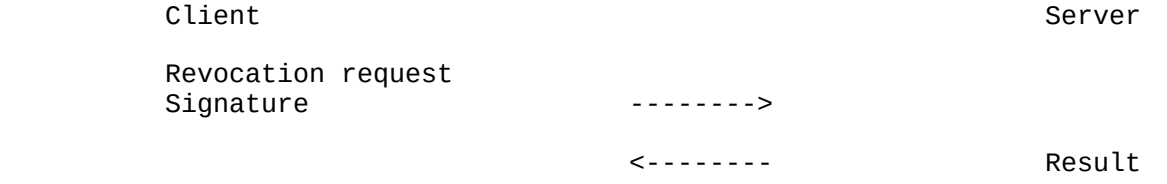


Once the client has authorized an account key pair for an identifier, it can use the key pair to authorize the issuance of certificates for the identifier. To do this, the client sends a PKCS#10 Certificate Signing Request (CSR) to the server (indicating the identifier(s) to be included in the issued certificate) and a signature over the CSR by the private key of the account key pair.

If the server agrees to issue the certificate, then it creates the certificate and provides it in its response. The certificate is assigned a URI, which the client can use to fetch updated versions of the certificate.



To revoke a certificate, the client simply sends a revocation request, signed with an authorized key pair, and the server indicates whether the request has succeeded.



Note that while ACME is defined with enough flexibility to handle different types of identifiers in principle, the primary use case addressed by this document is the case where domain names are used as identifiers. For example, all of the identifier validation challenges described in Section 7 below address validation of domain names. The use of ACME for other protocols will require further specification, in order to describe how these identifiers are encoded in the protocol, and what types of validation challenges the server might require.

5. Protocol Elements

This section describes several components that are used by ACME, and general rules that apply to ACME transactions.

5.1. HTTPS Requests

Each ACME function is accomplished by the client sending a sequence of HTTPS requests to the server, carrying JSON messages. Use of HTTPS is REQUIRED. Clients SHOULD support HTTP public key pinning [RFC7469], and servers SHOULD emit pinning headers. Each subsection of Section 6 below describes the message formats used by the function, and the order in which messages are sent.

All ACME requests with a non-empty body MUST encapsulate the body in a JWS object, signed using the account key pair. The server MUST verify the JWS before processing the request. (For readability, however, the examples below omit this encapsulation.) Encapsulating request bodies in JWS provides a simple authentication of requests by way of key continuity.

Note that this implies that GET requests are not authenticated. Servers MUST NOT respond to GET requests for resources that might be considered sensitive.

An ACME request carries a JSON dictionary that provides the details of the client’s request to the server. In order to avoid attacks that might arise from sending a request object to a resource of the wrong type, each request object MUST have a "resource" field that indicates what type of resource the request is addressed to, as defined in the below table:

Resource type	"resource" value
New registration	new-reg
Recover registration	recover-reg
New authorization	new-authz
New certificate	new-cert
Revoke certificate	revoke-cert
Registration	reg
Authorization	authz
Challenge	challenge
Certificate	cert

Other fields in ACME request bodies are described below.

ACME servers that are intended to be generally accessible need to use Cross-Origin Resource Sharing (CORS) in order to be accessible from browser-based clients [W3C.CR-cors-20130129]. Such servers SHOULD set the Access-Control-Allow-Origin header field to the value "*".

5.2. Registration Objects

An ACME registration resource represents a set of metadata associated to an account key pair. Registration resources have the following structure:

key (required, dictionary): The public key of the account key pair, encoded as a JSON Web Key object [RFC7517].

contact (optional, array of string): An array of URIs that the server can use to contact the client for issues related to this authorization. For example, the server may wish to notify the client about server-initiated revocation.

agreement (optional, string): A URI referring to a subscriber agreement or terms of service provided by the server (see below). Including this field indicates the client's agreement with the referenced terms.

authorizations (optional, string): A URI from which a list of authorizations granted to this account can be fetched via a GET request. The result of the GET request MUST be a JSON object whose "authorizations" field is an array of strings, where each string is the URI of an authorization belonging to this registration. The server SHOULD include pending authorizations, and SHOULD NOT include authorizations that are invalid or expired.

certificates (optional, string): A URI from which a list of certificates issued for this account can be fetched via a GET request. The result of the GET request MUST be a JSON object whose "certificates" field is an array of strings, where each string is the URI of a certificate. The server SHOULD NOT include expired certificates.

```
{
  "resource": "new-reg",
  "contact": [
    "mailto:cert-admin@example.com",
    "tel:+12025551212"
  ],
  "agreement": "https://example.com/acme/terms",
  "authorizations": "https://example.com/acme/reg/1/authz",
  "certificates": "https://example.com/acme/reg/1/cert",
}
```

5.3. Authorization Objects

An ACME authorization object represents server's authorization for an account to represent an identifier. In addition to the identifier, an authorization includes several metadata fields, such as the status of the authorization (e.g., "pending", "valid", or "revoked") and which challenges were used to validate possession of the identifier.

The structure of an ACME authorization resource is as follows:

identifier (required, dictionary of string): The identifier that the account is authorized to represent

type (required, string): The type of identifier.

value (required, string): The identifier itself.

status (optional, string): The status of this authorization. Possible values are: "unknown", "pending", "processing", "valid", "invalid" and "revoked". If this field is missing, then the default value is "pending".

expires (optional, string): The date after which the server will consider this authorization invalid, encoded in the format specified in RFC 3339 [RFC3339].

challenges (required, array): The challenges that the client needs to fulfill in order to prove possession of the identifier (for pending authorizations). For final authorizations, the challenges that were used. Each array entry is a dictionary with parameters required to validate the challenge, as specified in Section 7.

combinations (optional, array of arrays of integers): A collection of sets of challenges, each of which would be sufficient to prove possession of the identifier. Clients complete a set of challenges that that covers at least one set in this array. Challenges are identified by their indices in the challenges array. If no "combinations" element is included in an authorization object, the client completes all challenges.

The only type of identifier defined by this specification is a fully-qualified domain name (type: "dns"). The value of the identifier MUST be the ASCII representation of the domain name. Wildcard domain names (with "*" as the first label) MUST NOT be included in authorization requests. See Section 6.6 below for more information about wildcard domains.

```
{
  "status": "valid",
  "expires": "2015-03-01",

  "identifier": {
    "type": "dns",
    "value": "example.org"
  },

  "challenges": [
    {
      "type": "http-01",
      "status": "valid",
      "validated": "2014-12-01T12:05Z",
      "keyAuthorization": "SXQe-2XODaDxNR...vb29HhjjLPSggwiE"
    }
  ],
}
```

5.4. Errors

Errors can be reported in ACME both at the HTTP layer and within ACME payloads. ACME servers can return responses with an HTTP error response code (4XX or 5XX). For example: If the client submits a request using a method not allowed in this document, then the server MAY return status code 405 (Method Not Allowed).

When the server responds with an error status, it SHOULD provide additional information using problem document [I-D.ietf-appsawg-http-problem]. The "type" and "detail" fields MUST be populated. To facilitate automatic response to errors, this document defines the following standard tokens for use in the "type" field (within the "urn:acme:" namespace):

Code	Semantic
badCSR	The CSR is unacceptable (e.g., due to a short key)
badNonce	The client sent an unacceptable anti-replay nonce
connection	The server could not connect to the client for DV
dnssec	The server could not validate a DNSSEC signed domain
malformed	The request message was malformed
serverInternal	The server experienced an internal error
tls	The server experienced a TLS error during DV
unauthorized	The client lacks sufficient authorization
unknownHost	The server could not resolve a domain name

Authorization and challenge objects can also contain error information to indicate why the server was unable to validate authorization.

TODO: Flesh out errors and syntax for them

5.5. Replay protection

In order to protect ACME resources from any possible replay attacks, ACME requests have a mandatory anti-replay mechanism. This mechanism is based on the server maintaining a list of nonces that it has issued to clients, and requiring any signed request from the client to carry such a nonce.

An ACME server MUST include a Replay-Nonce header field in each successful response it provides to a client, with contents as specified below. In particular, the ACME server MUST provide a Replay-Nonce header field in response to a HEAD request for any valid resource. (This allows clients to easily obtain a fresh nonce.) It MAY also provide nonces in error responses.

Every JWS sent by an ACME client MUST include, in its protected header, the "nonce" header parameter, with contents as defined below. As part of JWS verification, the ACME server MUST verify that the value of the "nonce" header is a value that the server previously provided in a Replay-Nonce header field. Once a nonce value has appeared in an ACME request, the server MUST consider it invalid, in the same way as a value it had never issued.

When a server rejects a request because its nonce value was unacceptable (or not present), it SHOULD provide HTTP status code 400 (Bad Request), and indicate the ACME error code "urn:acme:badNonce".

The precise method used to generate and track nonces is up to the server. For example, the server could generate a random 128-bit value for each response, keep a list of issued nonces, and strike nonces from this list as they are used.

5.5.1. Replay-Nonce

The "Replay-Nonce" header field includes a server-generated value that the server can use to detect unauthorized replay in future client requests. The server should generate the value provided in Replay-Nonce in such a way that they are unique to each message, with high probability.

The value of the Replay-Nonce field MUST be an octet string encoded according to the base64url encoding described in Section 2 of [RFC7515]. Clients MUST ignore invalid Replay-Nonce values.

```
base64url = [A-Z] / [a-z] / [0-9] / "-" / "_"

Replay-Nonce = *base64url
```

The Replay-Nonce header field SHOULD NOT be included in HTTP request messages.

5.5.2. "nonce" (Nonce) JWS header parameter

The "nonce" header parameter provides a unique value that enables the verifier of a JWS to recognize when replay has occurred. The "nonce" header parameter MUST be carried in the protected header of the JWS.

The value of the "nonce" header parameter MUST be an octet string, encoded according to the base64url encoding described in Section 2 of [RFC7515]. If the value of a "nonce" header parameter is not valid according to this encoding, then the verifier MUST reject the JWS as malformed.

5.6. Key Agreement

Certain elements of the protocol will require the establishment of a shared secret between the client and the server, in such a way that an entity observing the ACME protocol cannot derive the secret. In these cases, we use a simple ECDH key exchange, based on the system used by CMS [RFC5753]:

- Inputs:
  - Client-generated key pair
  - Server-generated key pair
  - Length of the shared secret to be derived
  - Label
- Perform the ECDH primitive operation to obtain Z (Section 3.3.1 of [SEC1])
- Select a hash algorithm according to the curve being used:
  - For "P-256", use SHA-256
  - For "P-384", use SHA-384
  - For "P-521", use SHA-512
- Derive the shared secret value using the KDF in Section 3.6.1 of [SEC1] using Z and the selected hash algorithm, and with the UTF-8 encoding of the label as the SharedInfo value

In cases where the length of the derived secret is shorter than the output length of the chosen hash algorithm, the KDF referenced above reduces to a single hash invocation. The shared secret is equal to the leftmost octets of the following:

H(Z || 00000001 || label)

6. Certificate Management

In this section, we describe the certificate management functions that ACME enables:

- Account Key Registration
- Account Recovery
- Account Key Authorization
- Certificate Issuance
- Certificate Renewal
- Certificate Revocation

6.1. Resources

ACME is structured as a REST application with a few types of resources:

- Registration resources, representing information about an account
- Authorization resources, representing an account's authorization to act for an identifier
- Challenge resources, representing a challenge to prove control of an identifier
- Certificate resources, representing issued certificates
- A "directory" resource
- A "new-registration" resource
- A "new-authorization" resource
- A "new-certificate" resource
- A "revoke-certificate" resource

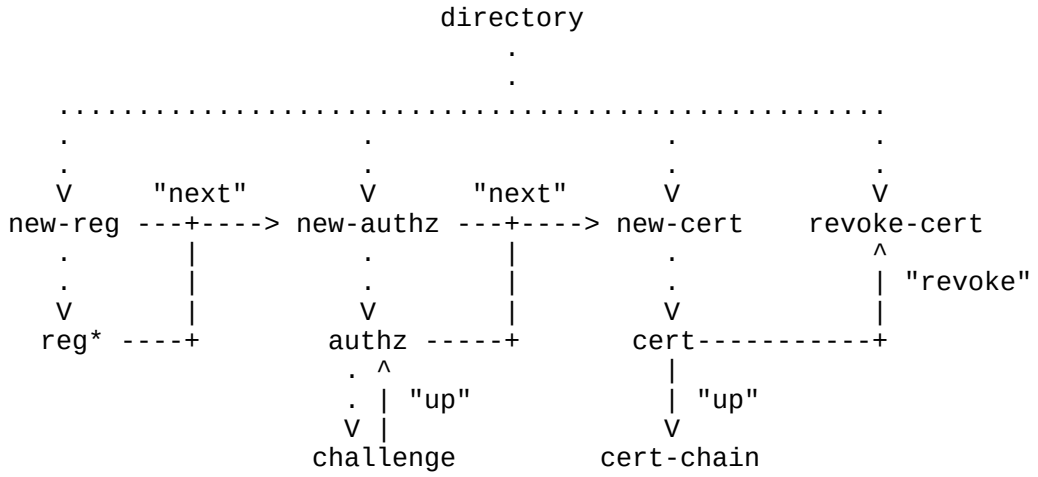
For the "new-X" resources above, the server MUST have exactly one resource for each function. This resource may be addressed by multiple URIs, but all must provide equivalent functionality.

In general, the intent is for authorization and certificate resources to contain only public information, so that CAs may publish these resources to document what certificates have been issued and how they were authorized. Non-public information, such as contact information, is stored in registration resources.

ACME uses different URIs for different management functions. Each function is listed in a directory along with its corresponding URI, so clients only need to be configured with the directory URI.

The "up" link relation is used with challenge resources to indicate the authorization resource to which a challenge belongs. It is also used from certificate resources to indicate a resource from which the client may fetch a chain of CA certificates that could be used to validate the certificate in the original resource.

The following diagram illustrates the relations between resources on an ACME server. The solid lines indicate link relations, and the dotted lines correspond to relationships expressed in other ways, e.g., the Location header in a 201 (Created) response.



The following table illustrates a typical sequence of requests required to establish a new account with the server, prove control of an identifier, issue a certificate, and fetch an updated certificate some time after issuance. The "->" is a mnemonic for a Location header pointing to a created resource.

Action	Request	Response
Register	POST new-reg	201 -> reg
Request challenges	POST new-authz	201 -> authz
Answer challenges	POST challenge	200
Poll for status	GET authz	200
Request issuance	POST new-cert	201 -> cert
Check for new cert	GET cert	200

The remainder of this section provides the details of how these resources are structured and how the ACME protocol makes use of them.

6.2. Directory

In order to help clients configure themselves with the right URIs for each ACME operation, ACME servers provide a directory object. This should be the root URL with which clients are configured. It is a JSON dictionary, whose keys are the "resource" values listed in Section 5.1, and whose values are the URIs used to accomplish the corresponding function.

Clients access the directory by sending a GET request to the directory URI.

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "new-reg": "https://example.com/acme/new-reg",
  "recover-reg": "https://example.com/acme/recover-reg",
  "new-authz": "https://example.com/acme/new-authz",
  "new-cert": "https://example.com/acme/new-cert",
  "revoke-cert": "https://example.com/acme/revoke-cert"
}
```

6.3. Registration

A client creates a new account with the server by sending a POST request to the server’s new-registration URI. The body of the request is a stub registration object containing only the "contact" field (along with the required "resource" field).

```
POST /acme/new-registration HTTP/1.1
Host: example.com

{
  "resource": "new-reg",
  "contact": [
    "mailto:cert-admin@example.com",
    "tel:+12025551212"
  ],
}
/* Signed as JWS */
```

The server MUST ignore any values provided in the "key", "authorizations", and "certificates" fields in registration bodies sent by the client, as well as any other fields that it does not recognize. If new fields are specified in the future, the specification of those fields MUST describe whether they may be provided by the client.

The server creates a registration object with the included contact information. The "key" element of the registration is set to the public key used to verify the JWS (i.e., the "jwk" element of the JWS header). The server returns this registration object in a 201 (Created) response, with the registration URI in a Location header field. The server MUST also indicate its new-authorization URI using the "next" link relation.

If the server already has a registration object with the provided account key, then it MUST return a 409 (Conflict) response and provide the URI of that registration in a Location header field. This allows a client that has an account key but not the corresponding registration URI to recover the registration URI.

If the server wishes to present the client with terms under which the ACME service is to be used, it MUST indicate the URI where such terms can be accessed in a Link header with link relation "terms-of-service". As noted above, the client may indicate its agreement with these terms by updating its registration to include the "agreement" field, with the terms URI as its value.

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: https://example.com/acme/reg/asdf
Link: <https://example.com/acme/new-authz>;rel="next"
Link: <https://example.com/acme/recover-reg>;rel="recover"
Link: <https://example.com/acme/terms>;rel="terms-of-service"

{
  "key": { /* JWK from JWS header */ },

  "contact": [
    "mailto:cert-admin@example.com",
    "tel:+12025551212"
  ]
}
```

If the client wishes to update this information in the future, it sends a POST request with updated information to the registration URI. The server MUST ignore any updates to the "key", "authorizations, or "certificates" fields, and MUST verify that the request is signed with the private key corresponding to the "key" field of the request before updating the registration.

Servers SHOULD NOT respond to GET requests for registration resources as these requests are not authenticated. If a client wishes to query the server for information about its account (e.g., to examine the "contact" or "certificates" fields), then it SHOULD do so by sending a POST request with an empty update. That is, it should send a JWS whose payload is trivial ({"resource":"reg"}). In this case the server reply MUST contain the same link headers sent for a new registration, to allow a client to retreive the "new-authorization" and "terms-of-service" URI

6.3.1. Recovery Keys

If the client wishes to establish a secret key with the server that it can use to recover this account later (a "recovery key"), then it must perform a simple key agreement protocol as part of the new-registration transaction. The client and server perform an ECDH exchange through the new-registration transaction (using the technique in Section 5.6), and the result is the recovery key.

To request a recovery key, the client includes a "recoveryKey" field in its new-registration request. The value of this field is a JSON object.

client (required, JWK): The client’s ECDH public key

length (required, number): The length of the derived secret, in octets.

In the client's request, this object contains a JWK for a random ECDH public key generated by the client and the client-selected length value. Clients need to choose length values that balance security and usability. On the one hand, a longer secret makes it more difficult for an attacker to recover the secret when it is used for recovery (see Section 6.4.1). On the other hand, clients may wish to make the recovery key short enough for a user to easily write it down.

```
POST /acme/new-registration HTTP/1.1
Host: example.com

{
  "resource": "new-reg",
  "contact": [
    "mailto:cert-admin@example.com",
    "tel:+12025551212"
  ],
  "recoveryKey": {
    "client": { "kty": "EC", ... },
    "length": 128
  }
}
/* Signed as JWS */
```

The server MUST validate that the elliptic curve ("crv") and length value chosen by the client are acceptable, and that it is otherwise willing to create a recovery key. If not, then it MUST reject the new-registration request.

If the server agrees to create a recovery key, then it generates its own random ECDH key pair and combines it with with the client's public key as described in Section 5.6 above, using the label "recovery". The derived secret value is the recovery key. The server then returns to the client the ECDH key that it generated. The server MUST generate a fresh key pair for every transaction.

server (required, JWK): The server's ECDH public key

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: https://example.com/acme/reg/asdf

{
  "key": { /* JWK from JWS header */ },

  "contact": [
    "mailto:cert-admin@example.com",
    "tel:+12025551212"
  ],

  "recoveryKey": {
    "server": { "kty": "EC", ... }
  }
}
```

On receiving the server's response, the client can compute the recovery key by combining the server's public key together with the private key corresponding to the public key that it sent to the server.

Clients may refresh the recovery key associated with a registration by sending a POST request with a new recoveryKey object. If the server agrees to refresh the recovery key, then it responds in the same way as to a new registration request that asks for a recovery key.

```
POST /acme/reg/asdf HTTP/1.1
Host: example.com

{
  "resource": "reg",
  "recoveryKey": {
    "client": { "kty": "EC", ... }
  }
}
/* Signed as JWS */
```

6.4. Account Recovery

Once a client has created an account with an ACME server, it is possible that the private key for the account will be lost. The recovery contacts included in the registration allows the client to recover from this situation, as long as it still has access to these contacts.

By "recovery", we mean that the information associated with an old account key is bound to a new account key. When a recovery process succeeds, the server provides the client with a new registration whose contents are the same as base registration object - except for the "key" field, which is set to the new account key. The server reassigns resources associated with the base registration to the new registration (e.g., authorizations and certificates). The server SHOULD delete the old registration resource after it has been used as a base for recovery.

In addition to the recovery mechanisms defined by ACME, individual client implementations may also offer implementation-specific recovery mechanisms. For example, if a client creates account keys deterministically from a seed value, then this seed could be used to recover the account key by re-generating it. Or an implementation could escrow an encrypted copy of the account key with a cloud storage provider, and give the encryption key to the user as a recovery value.

6.4.1. MAC-Based Recovery

With MAC-based recovery, the client proves to the server that it holds a secret value established in the initial registration transaction. The client requests MAC-based recovery by sending a MAC over the new account key, using the recovery key from the initial registration.

method (required, string): The string "mac"

base (required, string): The URI for the registration to be recovered.

mac (required, string): A JSON-formatted JWS object using an HMAC algorithm, whose payload is the JWK representation of the public key of the new account key pair.

```
POST /acme/recover-reg HTTP/1.1
Host: example.com

{
  "resource": "recover-reg",
  "method": "mac",
  "base": "https://example.com/acme/reg/asdf",
  "mac": {
    "header": { "alg": "HS256" },
    "payload": base64(JWK(newAccountKey)),
    "signature": "5wUrDI3eAaV4wl2Rfj3aC0Pp--XB3t4YYuNgacv_D3U"
  }
}
/* Signed as JWS, with new account key */
```

On receiving such a request the server MUST verify that:

- The base registration has a recovery key associated with it
- The "alg" value in the "mac" JWS represents a MAC algorithm
- The "mac" JWS is valid according to the validation rules in [RFC7515], using the recovery key as the MAC key
- The JWK in the payload represents the new account key (i.e. the key used to verify the ACME message)

If those conditions are met, and the recovery request is otherwise acceptable to the server, then the recovery process has succeeded. The server creates a new registration resource based on the base registration and the new account key, and returns it on a 201 (Created) response, together with a Location header indicating a URI for the new registration. If the recovery request is unsuccessful, the server returns an error response, such as 403 (Forbidden).

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: https://example.com/acme/reg/asdf
Link: <https://example.com/acme/new-authz>;rel="next"
Link: <https://example.com/acme/recover-reg>;rel="recover"
Link: <https://example.com/acme/terms>;rel="terms-of-service"

{
  "key": { /* JWK from JWS header */ },

  "contact": [
    "mailto:cert-admin@example.com",
    "tel:+12025551212"
  ],

  "authorizations": "...",
  "certificate": "..."
}
```

6.4.2. Contact-Based Recovery

In the contact-based recovery process, the client requests that the server send a message to one of the contact URIs registered for the account. That message indicates some action that the server requires the client's user to perform, e.g., clicking a link in an email. If the user successfully completes the server's required actions, then the server will bind the account to the new account key.

(Note that this process is almost entirely out of band with respect to ACME. ACME only allows the client to initiate the process, and the server to indicate the result.)

To initiate contact-based recovery, the client sends a POST request to the server's recover-registration URI, with a body specifying which registration is to be recovered. The body of the request MUST be signed by the client's new account key pair.

method (required, string): The string "contact"

base (required, string): The URI for the registration to be recovered.

```
POST /acme/recover-reg HTTP/1.1
Host: example.com

{
  "resource": "recover-reg",
  "method": "contact",
  "base": "https://example.com/acme/reg/asdf",
  "contact": [
    "mailto:forgetful@example.net"
  ]
}
/* Signed as JWS, with new account key */
```

If the server agrees to attempt contact-based recovery, then it creates a new registration resource containing a stub registration object. The stub registration has the client's new account key and contacts, but no authorizations or certificates associated. The server returns the stub contact in a 201 (Created) response, along with a Location header field indicating the URI for the new registration resource (which will be the registration URI if the recovery succeeds).

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: https://example.com/acme/reg/qwer

{
  "key": { /* new account key from JWS header */ },

  "contact": [
    "mailto:forgetful@example.net"
  ]
}
```

After recovery has been initiated, the server follows its chosen recovery process, out-of-band to ACME. While the recovery process is ongoing, the client may poll the registration resource's URI for status, by sending a POST request with a trivial body ({"resource":"reg"}). If the recovery process is still pending, the server sends a 202 (Accepted) status code, and a Retry-After header field. If the recovery process has failed, the server sends an error code (e.g., 404), and SHOULD delete the stub registration resource.

If the recovery process has succeeded, then the server will send a 200 (OK) response, containing the full registration object, with any necessary information copied from the old registration). The client may now use this in the same way as if he had gotten it from a new-registration transaction.

6.5. Identifier Authorization

The identifier authorization process establishes the authorization of an account to manage certificates for a given identifier. This process must assure the server of two things: First, that the client controls the private key of the account key pair, and second, that the client holds the identifier in question. This process may be repeated to associate multiple identifiers to a key pair (e.g., to request certificates with multiple identifiers), or to associate multiple accounts with an identifier (e.g., to allow multiple entities to manage certificates).

As illustrated by the figure in the overview section above, the authorization process proceeds in two phases. The client first requests a new authorization, and the server issues challenges, then the client responds to those challenges and the server validates the client's responses.

To begin the key authorization process, the client sends a POST request to the server's new-authorization resource. The body of the POST request MUST contain a JWS object, whose payload is a partial authorization object. This JWS object MUST contain only the "identifier" field, so that the server knows what identifier is being authorized. The server MUST ignore any other fields present in the client's request object.

The authorization object is implicitly tied to the account key used to sign the request. Once created, the authorization may only be updated by that account.

```
POST /acme/new-authorization HTTP/1.1
Host: example.com

{
  "resource": "new-authz",
  "identifier": {
    "type": "dns",
    "value": "example.org"
  }
}
/* Signed as JWS */
```

Before processing the authorization further, the server SHOULD determine whether it is willing to issue certificates for the identifier. For example, the server should check that the identifier is of a supported type. Servers might also check names against a blacklist of known high-value identifiers. If the server is unwilling to issue for the identifier, it SHOULD return a 403

(Forbidden) error, with a problem document describing the reason for the rejection.

If the server is willing to proceed, it builds a pending authorization object from the initial authorization object submitted by the client.

- "identifier" the identifier submitted by the client.
- "status": MUST be "pending"
- "challenges" and "combinations": As selected by the server's policy for this identifier
- The "expires" field MUST be absent.

The server allocates a new URI for this authorization, and returns a 201 (Created) response, with the authorization URI in a Location header field, and the JSON authorization object in the body.

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: https://example.com/authz/asdf
Link: <https://example.com/acme/new-cert>;rel="next"

{
  "status": "pending",

  "identifier": {
    "type": "dns",
    "value": "example.org"
  },

  "challenges": [
    {
      "type": "http-01",
      "uri": "https://example.com/authz/asdf/0",
      "token": "IlirfxKKXAsHtmzK29Pj8A"
    },
    {
      "type": "dns-01",
      "uri": "https://example.com/authz/asdf/1",
      "token": "DGyRejmCefe7v4NfDGDKfA"
    }
  },

  "combinations": [
    [0, 2],
    [1, 2]
  ]
}
```

The client needs to respond with information to complete the challenges. To do this, the client updates the authorization object received from the server by filling in any required information in the elements of the "challenges" dictionary. (This is also the stage where the client should perform any actions required by the challenge.)

The client sends these updates back to the server in the form of a JSON object with the response fields required by the challenge type, carried in a POST request to the challenge URI (not authorization URI or the new-authorization URI). This allows the client to send information only for challenges it is responding to.

For example, if the client were to respond to the "http-01" challenge in the above authorization, it would send the following request:

```
POST /acme/authz/asdf/0 HTTP/1.1
Host: example.com

{
  "resource": "challenge",
  "type": "http-01",
  "token": "evaGxfADs6pSRb2LAv9IZf17Dt3juxGJ-PCt92wr-oA"
}
/* Signed as JWS */
```

The server updates the authorization document by updating its representation of the challenge with the response fields provided by the client. The server MUST ignore any fields in the response object that are not specified as response fields for this type of challenge. The server provides a 200 (OK) response with the updated challenge object as its body.

If the client's response is invalid for some reason, or does not provide the server with appropriate information to validate the challenge, then the server MUST return an HTTP error. On receiving such an error, the client MUST undo any actions that have been taken to fulfil the challenge, e.g., removing files that have been provisioned to a web server.

Presumably, the client's responses provide the server with enough information to validate one or more challenges. The server is said to "finalize" the authorization when it has completed all the validations it is going to complete, and assigns the authorization a status of "valid" or "invalid", corresponding to whether it considers the account authorized for the identifier. If the final state is "valid", the server MUST add an "expires" field to the authorization. When finalizing an authorization, the server MAY remove the "combinations" field (if present) or remove any challenges still pending. The server SHOULD NOT remove challenges with status "invalid".

Usually, the validation process will take some time, so the client will need to poll the authorization resource to see when it is finalized. For challenges where the client can tell when the server has validated the challenge (e.g., by seeing an HTTP or DNS request from the server), the client SHOULD NOT begin polling until it has seen the validation request from the server.

To check on the status of an authorization, the client sends a GET request to the authorization URI, and the server responds with the current authorization object. In responding to poll requests while the validation is still in progress, the server MUST return a 202 (Accepted) response with a Retry-After header field.

```
GET /acme/authz/asdf HTTP/1.1
Host: example.com

HTTP/1.1 200 OK

{
  "status": "valid",
  "expires": "2015-03-01",

  "identifier": {
    "type": "dns",
    "value": "example.org"
  },

  "challenges": [
    {
      "type": "http-01"
      "status": "valid",
      "validated": "2014-12-01T12:05Z",
      "keyAuthorization": "SXQe-2XODaDxNR...vb29HhjjLPSggwiE"
    }
  ]
}
```

6.6. Certificate Issuance

The holder of an authorized key pair for an identifier may use ACME to request that a certificate be issued for that identifier. The client makes this request by sending a POST request to the server's new-certificate resource. The body of the POST is a JWS object whose JSON payload contains a Certificate Signing Request (CSR) [RFC2986]. The CSR encodes the parameters of the requested certificate; authority to issue is demonstrated by the JWS signature by an account key, from which the server can look up related authorizations.

csr (required, string): A CSR encoding the parameters for the certificate being requested. The CSR is sent in the Base64-encoded version of the DER format. (Note: This field uses the same modified Base64-encoding rules used elsewhere in this document, so it is different from PEM.)

```
POST /acme/new-cert HTTP/1.1
Host: example.com
Accept: application/pkix-cert

{
  "resource": "new-cert",
  "csr": "5jNudRx6Ye4HzKEqT5...FS6aKdZeGsysoCo4H9P",
}
/* Signed as JWS */
```

The CSR encodes the client's requests with regard to the content of the certificate to be issued. The CSR MUST indicate the requested identifiers, either in the commonName portion of the requested subject name, or in an extensionRequest attribute [RFC2985] requesting a subjectAltName extension.

The values provided in the CSR are only a request, and are not guaranteed. The server or CA may alter any fields in the certificate before issuance. For example, the CA may remove identifiers that are not authorized for the account key that signed the request.

It is up to the server's local policy to decide which names are acceptable in a certificate, given the authorizations that the server associates with the client's account key. A server MAY consider a client authorized for a wildcard domain if it is authorized for the underlying domain name (without the "*" label). Servers SHOULD NOT extend authorization across identifier types. For example, if a client is authorized for "example.com", then the server should not allow the client to issue a certificate with an iPAddress subjectAltName, even if it contains an IP address to which example.com resolves.

If the CA decides to issue a certificate, then the server creates a new certificate resource and returns a URI for it in the Location header field of a 201 (Created) response.

```
HTTP/1.1 201 Created
Location: https://example.com/acme/cert/asdf
```

If the certificate is available at the time of the response, it is provided in the body of the response. If the CA has not yet issued the certificate, the body of this response will be empty. The client should then send a GET request to the certificate URI to poll for the certificate. As long as the certificate is unavailable, the server MUST provide a 202 (Accepted) response and include a Retry-After header to indicate when the server believes the certificate will be issued (as in the example above).

```
GET /acme/cert/asdf HTTP/1.1
Host: example.com
Accept: application/pkix-cert

HTTP/1.1 202 Accepted
Retry-After: 120
```

The default format of the certificate is DER (application/pkix-cert). The client may request other formats by including an Accept header in its request.

The server provides metadata about the certificate in HTTP headers. In particular, the server MUST include a Link relation header field [RFC5988] with relation "up" to provide a certificate under which this certificate was issued, and one with relation "author" to indicate the registration under which this certicate was issued. The server MAY also include an Expires header as a hint to the client about when to renew the certificate. (Of course, the real expiration of the certificate is controlled by the notAfter time in the certificate itself.)

```
GET /acme/cert/asdf HTTP/1.1
Host: example.com
Accept: application/pkix-cert

HTTP/1.1 200 OK
Content-Type: application/pkix-cert
Link: <https://example.com/acme/ca-cert>;rel="up";title="issuer"
Link: <https://example.com/acme/revoke-cert>;rel="revoke"
Link: <https://example.com/acme/reg/asdf>;rel="author"
Location: https://example.com/acme/cert/asdf
Content-Location: https://example.com/acme/cert-seq/12345

[DER-encoded certificate]
```

A certificate resource always represents the most recent certificate issued for the name/key binding expressed in the CSR. If the CA allows a certificate to be renewed, then it publishes renewed versions of the certificate through the same certificate URI.

Clients retrieve renewed versions of the certificate using a GET query to the certificate URI, which the server should then return in a 200 (OK) response. The server SHOULD provide a stable URI for each specific certificate in the Content-Location header field, as shown above. Requests to stable certificate URIs MUST always result in the same certificate.

To avoid unnecessary renewals, the CA may choose not to issue a renewed certificate until it receives such a request (if it even allows renewal at all). In such cases, if the CA requires some time to generate the new certificate, the CA MUST return a 202 (Accepted) response, with a Retry-After header field that indicates when the new certificate will be available. The CA MAY include the current (non-renewed) certificate as the body of the response.

Likewise, in order to prevent unnecessary renewal due to queries by parties other than the account key holder, certificate URIs should be structured as capability URLs [W3C.WD-capability-urls-20140218].

From the client's perspective, there is no difference between a certificate URI that allows renewal and one that does not. If the client wishes to obtain a renewed certificate, and a GET request to the certificate URI does not yield one, then the client may initiate a new-certificate transaction to request one.

6.7. Certificate Revocation

To request that a certificate be revoked, the client sends a POST request to the ACME server's revoke-cert URI. The body of the POST is a JWS object whose JSON payload contains the certificate to be revoked:

certificate (required, string): The certificate to be revoked, in the Base64-encoded version of the DER format. (Note: This field uses the same modified Base64-encoding rules used elsewhere in this document, so it is different from PEM.)

```
POST /acme/revoke-cert HTTP/1.1
Host: example.com

{
  "resource": "revoke-cert",
  "certificate": "MIIEDTCCAvegAwIBAgIRAP8..."
}
/* Signed as JWS */
```

Revocation requests are different from other ACME request in that they can be signed either with an account key pair or the key pair in the certificate. Before revoking a certificate, the server MUST verify at least one of these conditions applies:

- the public key of the key pair signing the request matches the public key in the certificate.

- the key pair signing the request is an account key, and the corresponding account is authorized to act for all of the identifier(s) in the certificate.

If the revocation succeeds, the server responds with status code 200 (OK). If the revocation fails, the server returns an error.

```
HTTP/1.1 200 OK
Content-Length: 0

--- or ---

HTTP/1.1 403 Forbidden
Content-Type: application/problem+json
Content-Language: en

{
  "type": "urn:acme:error:unauthorized"
  "detail": "No authorization provided for name example.net"
  "instance": "http://example.com/doc/unauthorized"
}
```

7. Identifier Validation Challenges

There are few types of identifier in the world for which there is a standardized mechanism to prove possession of a given identifier. In all practical cases, CAs rely on a variety of means to test whether an entity applying for a certificate with a given identifier actually controls that identifier.

Challenges provide the server with assurance that an account key holder is also the entity that controls an identifier. For each type of challenge, it must be the case that in order for an entity to successfully complete the challenge the entity must both:

- Hold the private key of the account key pair used to respond to the challenge

- Control the identifier in question

Section 9 documents how the challenges defined in this document meet these requirements. New challenges will need to document how they do.

To accommodate this reality, ACME includes an extensible challenge/response framework for identifier validation. This section describes an initial set of Challenge types. Each challenge must describe:

- Content of Challenge payloads (in Challenge messages)
- Content of Response payloads (in authorizationRequest messages)
- How the server uses the Challenge and Response to verify control of an identifier

The general structure of Challenge and Response payloads is as follows:

type (required, string): The type of Challenge or Response encoded in the object.

uri (required, string): The URI to which a response can be posted.

status (optional, string): : The status of this authorization. Possible values are: "unknown", "pending", "processing", "valid", "invalid" and "revoked". If this field is missing, then the default value is "pending".

validated (optional, string): : The time at which this challenge was completed by the server, encoded in the format specified in RFC 3339 [RFC3339].

error (optional, dictionary of string): : The error that occurred while the server was validating the challenge, if any. This field is structured as a problem document [I-D.ietf-appsawg-http-problem].

All additional fields are specified by the Challenge type. The server MUST ignore any values provided in the "uri", "status", "validated", and "error" fields of a Response payload. If the server sets a Challenge's "status" to "invalid", it SHOULD also include the "error" field to help the client diagnose why they failed the challenge.

Different challenges allow the server to obtain proof of different aspects of control over an identifier. In some challenges, like HTTP and TLS SNI, the client directly proves its ability to do certain things related to the identifier. In the Proof of Possession challenge, the client proves historical control of the identifier, by reference to a prior authorization transaction or certificate.

The choice of which Challenges to offer to a client under which circumstances is a matter of server policy. A CA may choose different sets of challenges depending on whether it has interacted with a domain before, and how. For example:

- New domain with no known certificates: Domain Validation (HTTP or TLS SNI)
- Domain for which known certs exist from other CAs: DV + Proof of Possession of previous CA-signed key
- Domain with a cert from this CA, lost account key: DV + PoP of ACME-certified Subject key
- Domain with a cert from this CA, all keys and recovery mechanisms lost: Out of band proof of authority for the domain

The identifier validation challenges described in this section all relate to validation of domain names. If ACME is extended in the future to support other types of identifier, there will need to be new Challenge types, and they will need to specify which types of identifier they apply to.

[[Editor's Note: In pre-RFC versions of this specification, challenges are labeled by type, and with the version of the draft in which they were introduced. For example, if an HTTP challenge were introduced in version -03 and a breaking change made in version -05, then there would be a challenge labeled "http-03" and one labeled "http-05" - but not one labeled "http-04", since challenge in version -04 was compatible with one in version -04.]]

[[Editor's Note: Operators SHOULD NOT issue "combinations" arrays in authorization objects that require the client to perform multiple challenges over the same type, e.g., ["http-03", "http-05"]. Challenges within a type are testing the same capability of the domain owner, and it may not be possible to satisfy both at once.]]

7.1. Key Authorizations

Several of the challenges in this document makes use of a key authorization string. A key authorization expresses a domain holder's authorization for a specified key to satisfy a specified challenge, by concatenating the token for the challenge with a key fingerprint, separated by a "." character:

```
key-authz = token || '.' || base64(JWK_Thumbprint(accountKey))
```

The "JWK_Thumbprint" step indicates the computation specified in [RFC7638], using the SHA-256 digest. As specified in the individual challenges below, the token for a challenge is a JSON string comprised entirely of characters in the base64 alphabet. The "||" operator indicates concatenation of strings.

In computations involving key authorizations, such as the digest computations required for the DNS and TLS SNI challenges, the key authorization string MUST be represented in UTF-8 form (or, equivalently, ASCII).

7.2. HTTP

With Simple HTTP validation, the client in an ACME transaction proves its control over a domain name by proving that it can provision resources on an HTTP server that responds for that domain name. The ACME server challenges the client to provision a file with a specific JWS as its contents.

As a domain may resolve to multiple IPv4 and IPv6 addresses, the server will connect to at least one of the hosts found in A and AAAA records, at its discretion. Because many webservers allocate a default HTTPS virtual host to a particular low-privilege tenant user in a subtle and non-intuitive manner, the challenge must be completed over HTTP, not HTTPS.

type (required, string): The string "http-01"

token (required, string): A random value that uniquely identifies the challenge. This value MUST have at least 128 bits of entropy, in order to prevent an attacker from guessing it. It MUST NOT contain any characters outside the URL-safe Base64 alphabet.

```
{
  "type": "http-01",
  "token": "evaGxfADs6pSRb2LAv9IZf17Dt3juxGJ-PCt92wr-oA",
}
```

A client responds to this challenge by constructing a key authorization from the "token" value provided in the challenge and the client’s account key. The client then provisions the key authorization as a resource on the HTTP server for the domain in question.

```
evaGxfADs6pSRb2LAv9IZf17Dt3juxGJ-PCt92wr-oA
.nP1qzpXGymHBrUEepNY9HCsQk7K8KhOypzEt62jcerQ
```

The path at which the resource is provisioned is comprised of the fixed prefix ".well-known/acme-challenge/", followed by the "token" value in the challenge.

```
.well-known/acme-challenge/evaGxfADs6pSRb2LAv9IZf17Dt3juxGJ-PCt92wr-oA
```

The client's response to this challenge indicates its agreement to this challenge by sending the server the key authorization covering the challenge's token and the client's account key:

keyAuthorization (required, string): The key authorization for this challenge. This value MUST match the token from the challenge and the client's account key.

```
{
  "keyAuthorization": "evaGxfADs...62jcerQ"
}
/* Signed as JWS */
```

On receiving a response, the server MUST verify that the key authorization in the response matches the "token" value in the challenge and the client's account key. If they do not match, then the server MUST return an HTTP error in response to the POST request in which the client sent the challenge.

Given a Challenge/Response pair, the server verifies the client's control of the domain by verifying that the resource was provisioned as expected.

1. Form a URI by populating the URI template [RFC6570] "http://{domain}/.well-known/acme-challenge/{token}", where:

   * the domain field is set to the domain name being verified; and

   * the token field is set to the token in the challenge.

2. Verify that the resulting URI is well-formed.

3. Dereference the URI using an HTTP or HTTPS GET request. If using HTTPS, the ACME server MUST ignore the certificate provided by the HTTPS server.

4. Verify that the Content-Type header of the response is either absent, or has the value "text/plain".

5. Verify that the body of the response is well-formed key authorization.

6. Verify that key authorization provided by the server matches the token for this challenge and the client's account key.

If all of the above verifications succeed, then the validation is successful. If the request fails, or the body does not pass these checks, then it has failed.

7.3. TLS with Server Name Indication (TLS SNI)

The TLS with Server Name Indication (TLS SNI) validation method proves control over a domain name by requiring the client to configure a TLS server referenced by an A/AAAA record under the domain name to respond to specific connection attempts utilizing the Server Name Indication extension [RFC6066]. The server verifies the client's challenge by accessing the reconfigured server and verifying a particular challenge certificate is presented.

type (required, string): The string "tls-sni-01"

token (required, string): A random value that uniquely identifies the challenge. This value MUST have at least 128 bits of entropy, in order to prevent an attacker from guessing it. It MUST NOT contain any characters outside the URL-safe Base64 alphabet.

n (required, number): Number of tls-sni-01 iterations

```
{
  "type": "tls-sni-01",
  "token": "evaGxfADs6pSRb2LAv9IZf17Dt3juxGJ-PCt92wr-oA",
  "n": 25
}
```

A client responds to this challenge by constructing a key authorization from the "token" value provided in the challenge and the client's account key. The client first computes the SHA-256 digest Z0 of the UTF8-encoded key authorization, and encodes Z0 in UTF-8 lower-case hexadecimal form. The client then generates iterated hash values Z1...Z(n-1) as follows:

Z(i) = lowercase_hexadecimal(SHA256(Z(i-1))).

The client generates a self-signed certificate for each iteration of Zi with a single subjectAlternativeName extension dNSName that is "<Zi[0:32]>.<Zi[32:64]>.acme.invalid", where "Zi[0:32]" and "Zi[32:64]" represent the first 32 and last 32 characters of the hex-encoded value, respectively (following the notation used in Python). The client then configures the TLS server at the domain such that when a handshake is initiated with the Server Name Indication extension set to "<Zi[0:32]>.<Zi[32:64]>.acme.invalid", the corresponding generated certificate is presented.

The response to the TLS SNI challenge simply acknowledges that the client is ready to fulfill this challenge.

keyAuthorization (required, string): The key authorization for this challenge. This value MUST match the token from the challenge and the client's account key.

```
{
  "keyAuthorization": "evaGxfADs...62jcerQ",
}
/* Signed as JWS */
```

On receiving a response, the server MUST verify that the key authorization in the response matches the "token" value in the challenge and the client's account key. If they do not match, then the server MUST return an HTTP error in response to the POST request in which the client sent the challenge.

Given a Challenge/Response pair, the ACME server verifies the client's control of the domain by verifying that the TLS server was configured appropriately.

1. Choose a subset of the N iterations to check, according to local policy.

2. For each iteration, compute the Zi-value from the key authorization in the same way as the client.

3. Open a TLS connection to the domain name being validated on the requested port, presenting the value "<Zi[0:32]>.<Zi[32:64]>.acme.invalid" in the SNI field (where the comparison is case-insensitive).

4. Verify that the certificate contains a subjectAltName extension with the dNSName of "<Z[0:32]>.<Z[32:64]>.acme.invalid", and that no other dNSName entries of the form "*.acme.invalid" are present in the subjectAltName extension.

It is RECOMMENDED that the ACME server verify a random subset of the N iterations with an appropriate sized to ensure that an attacker who can provision certs for a default virtual host, but not for arbitrary simultaneous virtual hosts, cannot pass the challenge. For instance, testing a subset of 5 of N=25 domains ensures that such an attacker has only a one in 25/5 chance of success if they post certs Zn in random succession. (This probability is enforced by the requirement that each certificate have only one Zi value.)

It is RECOMMENDED that the ACME server validation TLS connections from multiple vantage points to reduce the risk of DNS hijacking attacks.

If all of the above verifications succeed, then the validation is successful. Otherwise, the validation fails.

7.4. Proof of Possession of a Prior Key

The Proof of Possession challenge verifies that a client possesses a private key corresponding to a server-specified public key, as demonstrated by its ability to sign with that key. This challenge is meant to be used when the server knows of a public key that is already associated with the identifier being claimed, and wishes for new authorizations to be authorized by the holder of the corresponding private key. For DNS identifiers, for example, this can help guard against domain hijacking.

This method is useful if a server policy calls for issuing a certificate only to an entity that already possesses the subject private key of a particular prior related certificate (perhaps issued by a different CA). It may also help enable other kinds of server policy that are related to authenticating a client's identity using digital signatures.

This challenge proceeds in much the same way as the proof of possession of the authorized key pair in the main ACME flow (challenge + authorizationRequest). The server provides a nonce and the client signs over the nonce. The main difference is that rather than signing with the private key of the key pair being authorized, the client signs with a private key specified by the server. The server can specify which key pair(s) are acceptable directly (by indicating a public key), or by asking for the key corresponding to a certificate.

The server provides the following fields as part of the challenge:

type (required, string): The string "proofOfPossession-01"

certs (optional, array of string): An array of certificates, in Base64-encoded DER format, that contain acceptable public keys.

```
{
  "type": "proofOfPossession-01",
  "certs": ["MIIF7z...bYVQLY"]
}
```

In response to this challenge, the client uses the private key corresponding to one of the acceptable public keys to sign a JWS object including data related to the challenge. The validation object covered by the signature has the following fields:

type (required, string): The string "proofOfPossession"

identifiers (required, identifier): A list of identifiers for which the holder of the prior key authorizes the new key

accountKey (required, JWK): The client's account public key

```
{
  "type": "proofOfPossession-01",
  "identifiers: [{"type": "dns", "value": "example.com"}],
  "accountKey": { "kty": "RSA", ... }
}
```

This JWS is NOT REQUIRED to have a "nonce" header parameter (as with the JWS objects that carry ACME request objects). This allows proof-of-possession response objects to be computed off-line. For example, as part of a domain transfer, the new domain owner might require the old domain owner to sign a proof-of-possession validation object, so that the new domain owner can present that in an ACME transaction later.

The validation JWS MUST contain a "jwk" header parameter indicating the public key under which the server should verify the JWS.

The client's response includes the server-provided nonce, together with a signature over that nonce by one of the private keys requested by the server.

type (required, string): The string "proofOfPossession"

authorization (required, JWS): The validation JWS

```
{
  "type": "proofOfPossession-01",
  "authorization": {
    "header": {
      "alg": "RS256",
      "jwk": {
        "kty": "RSA",
        "e": "AQAB",
        "n": "AMswMT...3aVtjE"
      }
    },
    "payload": "SfiR1...gSAl7A",
    "signature": "XcQLfL...cW5beg"
  }
}
```

To validate a proof-of-possession challenge, the server performs the following steps:

1. Verify that the public key in the "jwk" header of the "authorization" JWS corresponds to one of the certificates in the "certs" field of the challenge

2. Verify the "authorization" JWS using the key indicated in its "jwk" header

3. Decode the payload of the JWS as UTF-8 encoded JSON

4. Verify that there are exactly three fields in the decoded object, and that:

   * The "type" field is set to "proofOfPossession"

   * The "identifier" field contains the identifier for which authorization is being validated

   * The "accountKey" field matches the account key for which the challenge was issued

If all of the above verifications succeed, then the validation is successful. Otherwise, the validation fails.

7.5. DNS

When the identifier being validated is a domain name, the client can prove control of that domain by provisioning resource records under it. The DNS challenge requires the client to provision a TXT record containing a designated value under a specific validation domain name.

type (required, string): The string "dns-01"

token (required, string): A random value that uniquely identifies the challenge. This value MUST have at least 128 bits of entropy, in order to prevent an attacker from guessing it. It MUST NOT contain any characters outside the URL-safe Base64 alphabet.

```
{
  "type": "dns-01",
  "token": "evaGxfADs6pSRb2LAv9IZf17Dt3juxGJ-PCt92wr-oA"
}
```

A client responds to this challenge by constructing a key authorization from the "token" value provided in the challenge and

the client's account key. The client then computes the SHA-256 digest of the key authorization.

The record provisioned to the DNS is the base64 encoding of this digest. The client constructs the validation domain name by prepending the label "_acme-challenge" to the domain name being validated, then provisions a TXT record with the digest value under that name. For example, if the domain name being validated is "example.com", then the client would provision the following DNS record:

```
_acme-challenge.example.com. 300 IN TXT "gfj9Xq...Rg85nM"
```

The response to the DNS challenge simply acknowledges that the client is ready to fulfill this challenge.

keyAuthorization (required, string): The key authorization for this challenge. This value MUST match the token from the challenge and the client's account key.

```
{
  "keyAuthorization": "evaGxfADs...62jcerQ",
}
/* Signed as JWS */
```

On receiving a response, the server MUST verify that the key authorization in the response matches the "token" value in the challenge and the client's account key. If they do not match, then the server MUST return an HTTP error in response to the POST request in which the client sent the challenge.

To validate a DNS challenge, the server performs the following steps:

1. Compute the SHA-256 digest of the key authorization

2. Query for TXT records under the validation domain name

3. Verify that the contents of one of the TXT records matches the digest value

If all of the above verifications succeed, then the validation is successful. If no DNS record is found, or DNS record and response payload do not pass these checks, then the validation fails.

8. IANA Considerations

TODO

- Register .well-known path
- Register Replay-Nonce HTTP header
- Register "nonce" JWS header parameter
- Register "urn:acme" namespace
- Create identifier validation method registry
- Registries of syntax tokens, e.g., message types / error types?

9. Security Considerations

ACME is a protocol for managing certificates that attest to identifier/key bindings. Thus the foremost security goal of ACME is to ensure the integrity of this process, i.e., to ensure that the bindings attested by certificates are correct, and that only authorized entities can manage certificates. ACME identifies clients by their account keys, so this overall goal breaks down into two more precise goals:

1. Only an entity that controls a identifier can get an account key authorized for that identifier
2. Once authorized, an account key’s authorizations cannot be improperly transferred to another account key

In this section, we discuss the threat model that underlies ACME and the ways that ACME achieves these security goals within that threat model. We also discuss the denial-of-service risks that ACME servers face, and a few other miscellaneous considerations.

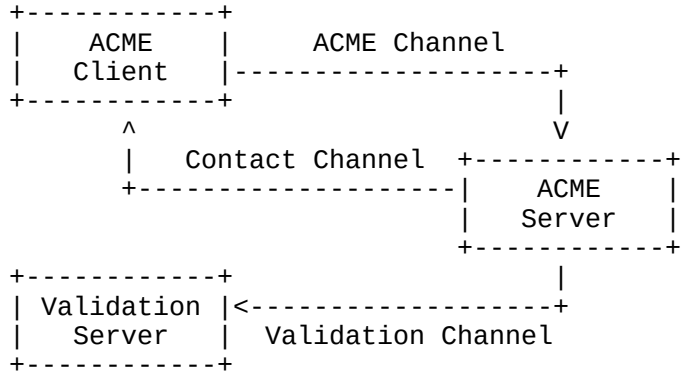
9.1. Threat model

As a service on the Internet, ACME broadly exists within the Internet threat model [RFC3552]. In analyzing ACME, it is useful to think of an ACME server interacting with other Internet hosts along three "channels":

- An ACME channel, over which the ACME HTTPS requests are exchanged

- A validation channel, over which the ACME server performs additional requests to validate a client's control of an identifier

- A contact channel, over which the ACME server sends messages to the registered contacts for ACME clients



In practice, the risks to these channels are not entirely separate, but they are different in most cases. Each of the three channels, for example, uses a different communications pattern: the ACME channel will comprise inbound HTTPS connections to the ACME server, the validation channel outbound HTTP or DNS requests, and the contact channel will use channels such as email and PSTN.

Broadly speaking, ACME aims to be secure against active and passive attackers on any individual channel. Some vulnerabilities arise (noted below), when an attacker can exploit both the ACME channel and one of the others.

On the ACME channel, in addition to network-layer attackers, we also need to account for application-layer man in the middle attacks, and for abusive use of the protocol itself. Protection against application-layer MitM addresses potential attackers such as Content Distribution Networks (CDNs) and middleboxes with a TLS MitM function. Preventing abusive use of ACME means ensuring that an attacker with access to the validation or contact channels can't obtain illegitimate authorization by acting as an ACME client (legitimately, in terms of the protocol).

9.2. Integrity of Authorizations

ACME allows anyone to request challenges for an identifier by registering an account key and sending a new-authorization request under that account key. The integrity of the authorization process

thus depends on the identifier validation challenges to ensure that the challenge can only be completed by someone who both (1) holds the private key of the account key pair, and (2) controls the identifier in question.

Validation responses need to be bound to an account key pair in order to avoid situations where an ACME MitM can switch out a legitimate domain holder's account key for one of his choosing, e.g.:

- Legitimate domain holder registers account key pair A
- MitM registers account key pair B
- Legitimate domain holder sends a new-authorization request signed under account key A
- MitM suppresses the legitimate request, but sends the same request signed under account key B
- ACME server issues challenges and MitM forwards them to the legitimate domain holder
- Legitimate domain holder provisions the validation response
- ACME server performs validation query and sees the response provisioned by the legitimate domain holder
- Because the challenges were issued in response to a message signed account key B, the ACME server grants authorization to account key B (the MitM) instead of account key A (the legitimate domain holder)

All of the challenges above that require an out-of-band query by the server have a binding to the account private key, such that the only the account private key holder can successfully respond to the validation query:

- HTTP: The value provided in the validation request is signed by the account private key.
- TLS SNI: The validation TLS request uses the account key pair as the server's key pair.
- DNS: The MAC covers the account key, and the MAC key is derived from an ECDH public key signed with the account private key.
- Proof of possession of a prior key: The signature by the prior key covers the account public key.

The association of challenges to identifiers is typically done by requiring the client to perform some action that only someone who effectively controls the identifier can perform. For the challenges in this document, the actions are:

- HTTP: Provision files under .well-known on a web server for the domain
- TLS SNI: Configure a TLS server for the domain
- DNS: Provision DNS resource records for the domain
- Proof of possession of a prior key: Sign using the private key specified by the server

There are several ways that these assumptions can be violated, both by misconfiguration and by attack. For example, on a web server that allows non-administrative users to write to .well-known, any user can claim to own the server's hostname by responding to a Simple HTTP challenge, and likewise for TLS configuration and TLS SNI.

The use of hosting providers is a particular risk for ACME validation. If the owner of the domain has outsourced operation of DNS or web services to a hosting provider, there is nothing that can be done against tampering by the hosting provider. As far as the outside world is concerned, the zone or web site provided by the hosting provider is the real thing.

More limited forms of delegation can also lead to an unintended party gaining the ability to successfully complete a validation transaction. For example, suppose an ACME server follows HTTP redirects in Simple HTTP validation and a web site operator provisions a catch-all redirect rule that redirects requests for unknown resources to different domain. Then the target of the redirect could use that to get a certificate through Simple HTTP validation, since the validation path will not be known to the primary server.

The DNS is a common point of vulnerability for all of these challenges. An entity that can provision false DNS records for a domain can attack the DNS challenge directly, and can provision false A/AAAA records to direct the ACME server to send its TLS SNI or HTTP validation query to a server of the attacker's choosing. There are a few different mitigations that ACME servers can apply:

- Checking the DNSSEC status of DNS records used in ACME validation (for zones that are DNSSEC-enabled)

- Querying the DNS from multiple vantage points to address local attackers
- Applying mitigations against DNS off-path attackers, e.g., adding entropy to requests [I-D.vixie-dnsext-dns0x20] or only using TCP

Given these considerations, the ACME validation process makes it impossible for any attacker on the ACME channel, or a passive attacker on the validation channel to hijack the authorization process to authorize a key of the attacker's choice.

An attacker that can only see the ACME channel would need to convince the validation server to provide a response that would authorize the attacker's account key, but this is prevented by binding the validation response to the account key used to request challenges. A passive attacker on the validation channel can observe the correct validation response and even replay it, but that response can only be used with the account key for which it was generated.

An active attacker on the validation channel can subvert the ACME process, by performing normal ACME transactions and providing a validation response for his own account key. The risks due to hosting providers noted above are a particular case. For identifiers where the server already has some credential associated with the domain this attack can be prevented by requiring the client to complete a proof-of-possession challenge.

9.3. Preventing Authorization Hijacking

The account recovery processes described in Section 6.4 allow authorization to be transferred from one account key to another, in case the former account key pair's private key is lost. ACME needs to prevent these processes from being exploited by an attacker to hijack the authorizations attached to one key and assign them to a key of the attacker's choosing.

Recovery takes place in two steps: 1. Provisioning recovery information (contact or recovery key) 2. Using recovery information to recover an account

The provisioning process needs to ensure that only the account key holder ends up with information that is useful for recovery. The recovery process needs to assure that only the (now former) account key holder can successfully execute recovery, i.e., that this entity is the only one that can choose the new account key that receives the capabilities held by the account being recovered.

MAC-based recovery can be performed if the attacker knows the account key and registration URI for the account being recovered. Both of these are difficult to obtain for a network attacker, because ACME uses HTTPS, though if the recovery key and registration URI are sufficiently predictable, the attacker might be able to guess them. An ACME MitM can see the registration URI, but still has to guess the recovery key, since neither the ECDH in the provisioning phase nor HMAC in the recovery phase will reveal it to him.

ACME clients can thus mitigate problems with MAC-based recovery by using long recovery keys. ACME servers should enforce a minimum recovery key length, and impose rate limits on recovery to limit an attacker’s ability to test different guesses about the recovery key.

Contact-based recovery uses both the ACME channel and the contact channel. The provisioning process is only visible to an ACME MitM, and even then, the MitM can only observe the contact information provided. If the ACME attacker does not also have access to the contact channel, there is no risk.

The security of the contact-based recovery process is entirely dependent on the security of the contact channel. The details of this will depend on the specific out-of-band technique used by the server. For example:

- If the server requires a user to click a link in a message sent to a contact address, then the contact channel will need to ensure that the message is only available to the legitimate owner of the contact address. Otherwise, a passive attacker could see the link and click it first, or an active attacker could redirect the message.

- If the server requires a user to respond to a message sent to a contact address containing a secret value, then the contact channel will need to ensure that an attacker cannot observe the secret value and spoof a message from the contact address.

In practice, many contact channels that can be used to reach many clients do not provide strong assurances of the types noted above. In designing and deploying contact-based recovery schemes, ACME servers operators will need to find an appropriate balance between using contact channels that can reach many clients and using contact-based recovery schemes that achieve an appropriate level of risk using those contact channels.

9.4. Denial-of-Service Considerations

As a protocol run over HTTPS, standard considerations for TCP-based and HTTP-based DoS mitigation also apply to ACME.

At the application layer, ACME requires the server to perform a few potentially expensive operations. Identifier validation transactions require the ACME server to make outbound connections to potentially attacker-controlled servers, and certificate issuance can require interactions with cryptographic hardware.

In addition, an attacker can also cause the ACME server to send validation requests to a domain of its choosing by submitting authorization requests for the victim domain.

All of these attacks can be mitigated by the application of appropriate rate limits. Issues closer to the front end, like POST body validation, can be addressed using HTTP request limiting. For validation and certificate requests, there are other identifiers on which rate limits can be keyed. For example, the server might limit the rate at which any individual account key can issue certificates, or the rate at which validation can be requested within a given subtree of the DNS.

9.5. CA Policy Considerations

The controls on issuance enabled by ACME are focused on validating that a certificate applicant controls the identifier he claims. Before issuing a certificate, however, there are many other checks that a CA might need to perform, for example:

- Has the client agreed to a subscriber agreement?
- Is the claimed identifier syntactically valid?
- For domain names:
  - If the leftmost label is a '*', then have the appropriate checks been applied?
  - Is the name on the Public Suffix List?
  - Is the name a high-value name?
  - Is the name a known phishing domain?
- Is the key in the CSR sufficiently strong?

- Is the CSR signed with an acceptable algorithm?

CAs that use ACME to automate issuance will need to ensure that their servers perform all necessary checks before issuing.

10. Acknowledgements

In addition to the editors listed on the front page, this document has benefited from contributions from a broad set of contributors, all the way back to its inception.

- Peter Eckersley, EFF
- Eric Rescorla, Mozilla
- Seth Schoen, EFF
- Alex Halderman, University of Michigan
- Martin Thomson, Mozilla
- Jakub Warmuz, University of Oxford

This document draws on many concepts established by Eric Rescorla's "Automated Certificate Issuance Protocol" draft. Martin Thomson provided helpful guidance in the use of HTTP.

11. References

11.1. Normative References

[I-D.ietf-appsawg-http-problem] mnot, m. and E. Wilde, "Problem Details for HTTP APIs", draft-ietf-appsawg-http-problem-01 (work in progress), September 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2314] Kaliski, B., "PKCS #10: Certification Request Syntax Version 1.5", RFC 2314, DOI 10.17487/RFC2314, March 1998, <http://www.rfc-editor.org/info/rfc2314>.

[RFC2985] Nystrom, M. and B. Kaliski, "PKCS #9: Selected Object Classes and Attribute Types Version 2.0", RFC 2985, DOI 10.17487/RFC2985, November 2000, <http://www.rfc-editor.org/info/rfc2985>.

[RFC2986] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <http://www.rfc-editor.org/info/rfc2986>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <http://www.rfc-editor.org/info/rfc3339>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC4514] Zeilenga, K., Ed., "Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names", RFC 4514, DOI 10.17487/RFC4514, June 2006, <http://www.rfc-editor.org/info/rfc4514>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <http://www.rfc-editor.org/info/rfc5280>.

[RFC5753] Turner, S. and D. Brown, "Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS)", RFC 5753, DOI 10.17487/RFC5753, January 2010, <http://www.rfc-editor.org/info/rfc5753>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <http://www.rfc-editor.org/info/rfc6066>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <http://www.rfc-editor.org/info/rfc6570>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7469] Evans, C., Palmer, C., and R. Sleevi, "Public Key Pinning Extension for HTTP", RFC 7469, DOI 10.17487/RFC7469, April 2015, <http://www.rfc-editor.org/info/rfc7469>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <http://www.rfc-editor.org/info/rfc7515>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <http://www.rfc-editor.org/info/rfc7517>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <http://www.rfc-editor.org/info/rfc7518>.

[RFC7638] Jones, M. and N. Sakimura, "JSON Web Key (JWK) Thumbprint", RFC 7638, DOI 10.17487/RFC7638, September 2015, <http://www.rfc-editor.org/info/rfc7638>.

[SEC1] Standards for Efficient Cryptography Group, "SEC 1: Elliptic Curve Cryptography", May 2009, <http://www.secg.org/sec1-v2.pdf>.

11.2. Informative References

[I-D.vixie-dnsext-dns0x20] Vixie, P. and D. Dagon, "Use of Bit 0x20 in DNS Labels to Improve Transaction Identity", draft-vixie-dnsext-dns0x20-00 (work in progress), March 2008.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <http://www.rfc-editor.org/info/rfc2818>.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <http://www.rfc-editor.org/info/rfc3552>.

[W3C.CR-cors-20130129] Kesteren, A., "Cross-Origin Resource Sharing", World Wide Web Consortium CR CR-cors-20130129, January 2013, <http://www.w3.org/TR/2013/CR-cors-20130129>.

[W3C.WD-capability-urls-20140218] Tennison, J., "Good Practices for Capability URLs", World Wide Web Consortium WD WD-capability-urls-20140218, February 2014, <http://www.w3.org/TR/2014/WD-capability-urls-20140218>.

Authors' Addresses

Richard Barnes
Mozilla

Email: rlb@ipv.sx

Jacob Hoffman-Andrews
EFF

Email: jsha@eff.org

James Kasten
University of Michigan

Email: jdkasten@umich.edu

ACME Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 28, 2018

R. Barnes
Cisco
J. Hoffman-Andrews
EFF
D. McCarney
Let's Encrypt
J. Kasten
University of Michigan

March 27, 2018

Automatic Certificate Management Environment (ACME)
draft-ietf-acme-acme-11

Abstract

Certificates in PKI using X.509 (PKIX) are used for a number of purposes, the most significant of which is the authentication of domain names. Thus, certificate authorities in the Web PKI are trusted to verify that an applicant for a certificate legitimately represents the domain name(s) in the certificate. Today, this verification is done through a collection of ad hoc mechanisms. This document describes a protocol that a certification authority (CA) and an applicant can use to automate the process of verification and certificate issuance. The protocol also provides facilities for other certificate management functions, such as certificate revocation.

RFC EDITOR: PLEASE REMOVE THE FOLLOWING PARAGRAPH: The source for this draft is maintained in GitHub. Suggested changes should be submitted as pull requests at https://github.com/ietf-wg-acme/acme [1]. Instructions are on that page as well. Editorial changes can be managed in GitHub, but any substantive change should be discussed on the ACME mailing list (acme@ietf.org).

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 28, 2018.

Table of Contents

1. Introduction

Certificates [RFC5280] in the Web PKI are most commonly used to authenticate domain names. Thus, certificate authorities in the Web PKI are trusted to verify that an applicant for a certificate legitimately represents the domain name(s) in the certificate.

Different types of certificates reflect different kinds of CA verification of information about the certificate subject. "Domain Validation" (DV) certificates are by far the most common type. For DV validation, the CA merely verifies that the requester has effective control of the web server and/or DNS server for the domain, but does not explicitly attempt to verify their real-world identity. (This is as opposed to "Organization Validation" (OV) and "Extended Validation" (EV) certificates, where the process is intended to also verify the real-world identity of the requester.)

Existing Web PKI certificate authorities tend to use a set of ad hoc protocols for certificate issuance and identity verification. In the case of DV certificates, a typical user experience is something like:

- Generate a PKCS#10 [RFC2986] Certificate Signing Request (CSR).
- Cut-and-paste the CSR into a CA web page.
- Prove ownership of the domain by one of the following methods:
  - Put a CA-provided challenge at a specific place on the web server.
  - Put a CA-provided challenge in a DNS record corresponding to the target domain.
  - Receive a CA-provided challenge at a (hopefully) administrator-controlled email address corresponding to the domain and then respond to it on the CA's web page.
- Download the issued certificate and install it on their Web Server.

With the exception of the CSR itself and the certificates that are issued, these are all completely ad hoc procedures and are accomplished by getting the human user to follow interactive natural-language instructions from the CA rather than by machine-implemented

published protocols. In many cases, the instructions are difficult to follow and cause significant confusion. Informal usability tests by the authors indicate that webmasters often need 1-3 hours to obtain and install a certificate for a domain. Even in the best case, the lack of published, standardized mechanisms presents an obstacle to the wide deployment of HTTPS and other PKIX-dependent systems because it inhibits mechanization of tasks related to certificate issuance, deployment, and revocation.

This document describes an extensible framework for automating the issuance and domain validation procedure, thereby allowing servers and infrastructural software to obtain certificates without user interaction. Use of this protocol should radically simplify the deployment of HTTPS and the practicality of PKIX authentication for other protocols based on Transport Layer Security (TLS) [RFC5246].

It should be noted that while the focus of this document is on validating domain names for purposes of issuing certificates in the Web PKI, ACME supports extensions for uses with other identifiers in other PKI contexts. For example, as of this writing, there is ongoing work to use ACME for issuance of WebPKI certificates attesting to IP addresses [I-D.ietf-acme-ip] and STIR certificates attesting to telephone numbers [I-D.ietf-acme-telephone].

ACME can also be used to automate some aspects of certificate management even where non-automated processes are still needed. For example, the external account binding feature (see Section 7.3.5) can allow an ACME account to use authorizations that have been granted to an external, non-ACME account. This allows ACME to address issuance scenarios that cannot yet be fully automated, such as the issuance of Extended Validation certificates.

2. Deployment Model and Operator Experience

The guiding use case for ACME is obtaining certificates for websites (HTTPS [RFC2818]). In this case, the user’s web server is intended to speak for one or more domains, and the process of certificate issuance is intended to verify that this web server actually speaks for the domain(s).

DV certificate validation commonly checks claims about properties related to control of a domain name - properties that can be observed by the certificate issuer in an interactive process that can be conducted purely online. That means that under typical circumstances, all steps in the request, verification, and issuance process can be represented and performed by Internet protocols with no out-of-band human intervention.

Prior to ACME, when deploying an HTTPS server, a server operator typically gets a prompt to generate a self-signed certificate. If the operator were instead deploying an HTTPS server using ACME, the experience would be something like this:

- The operator's ACME client prompts the operator for the intended domain name(s) that the web server is to stand for.
- The ACME client presents the operator with a list of CAs from which it could get a certificate. (This list will change over time based on the capabilities of CAs and updates to ACME configuration.) The ACME client might prompt the operator for payment information at this point.
- The operator selects a CA.
- In the background, the ACME client contacts the CA and requests that it issue a certificate for the intended domain name(s).
- The CA verifies that the client controls the requested domain name(s) by having the ACME client perform some action related to the domain name(s).
- Once the CA is satisfied, it issues the certificate and the ACME client automatically downloads and installs it, potentially notifying the operator via email, SMS, etc.
- The ACME client periodically contacts the CA to get updated certificates, stapled OCSP responses, or whatever else would be required to keep the web server functional and its credentials up-to-date.

In this way, it would be nearly as easy to deploy with a CA-issued certificate as with a self-signed certificate. Furthermore, the maintenance of that CA-issued certificate would require minimal manual intervention. Such close integration of ACME with HTTPS servers allows the immediate and automated deployment of certificates as they are issued, sparing the human administrator from much of the time-consuming work described in the previous section.

3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

The two main roles in ACME are "client" and "server". The ACME client uses the protocol to request certificate management actions,

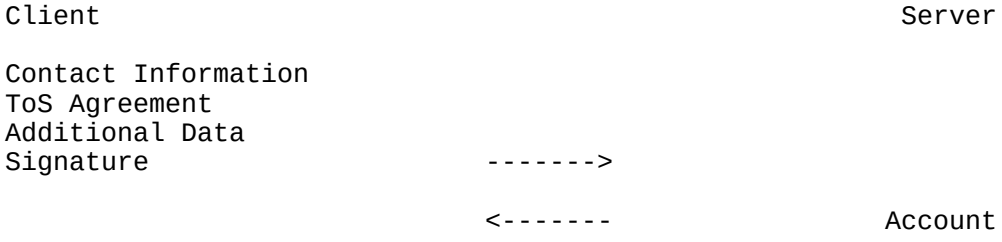
such as issuance or revocation. An ACME client may run on a web server, mail server, or some other server system which requires valid TLS certificates. Or, it may run on a separate server that does not consume the certificate, but is authorized to respond to a CA-provided challenge. The ACME server runs at a certification authority, and responds to client requests, performing the requested actions if the client is authorized.

An ACME client is represented by an "account key pair". The client uses the private key of this key pair to sign all messages sent to the server. The server uses the public key to verify the authenticity and integrity of messages from the client.

4. Protocol Overview

ACME allows a client to request certificate management actions using a set of JavaScript Object Notation (JSON) messages carried over HTTPS. Issuance using ACME resembles a traditional CA's issuance process, in which a user creates an account, requests a certificate, and proves control of the domain(s) in that certificate in order for the CA to sign the requested certificate.

The first phase of ACME is for the client to request an account with the ACME server. The client generates an asymmetric key pair and requests a new account, optionally providing contact information, agreeing to terms of service, and/or associating the account with an existing account in another system. The creation request is signed with the generated private key to prove that the client controls it.



Once an account is registered, there are four major steps the client needs to take to get a certificate:

1. Submit an order for a certificate to be issued

2. Prove control of any identifiers requested in the certificate

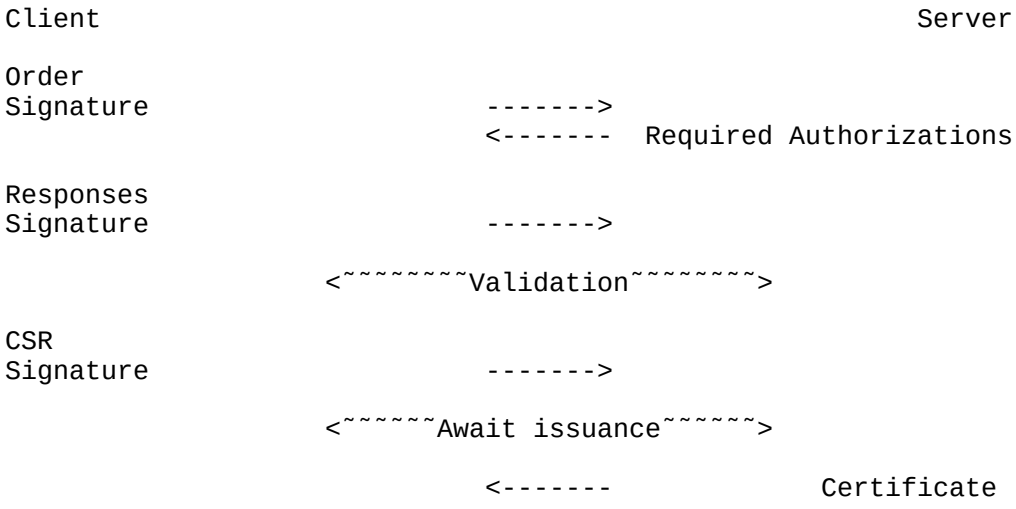
3. Finalize the order by submitting a CSR

4. Await issuance and download the issued certificate

The client's order for a certificate describes the desired identifiers plus a few additional fields that capture semantics that are not supported in the CSR format. If the server is willing to consider issuing such a certificate, it responds with a list of requirements that the client must satisfy before the certificate will be issued.

For example, in most cases, the server will require the client to demonstrate that it controls the identifiers in the requested certificate. Because there are many different ways to validate possession of different types of identifiers, the server will choose from an extensible set of challenges that are appropriate for the identifier being claimed. The client responds with a set of responses that tell the server which challenges the client has completed. The server then validates that the client has completed the challenges.

Once the validation process is complete and the server is satisfied that the client has met its requirements, the client finalizes the order by submitting a PKCS#10 Certificate Signing Request (CSR). The server will issue the requested certificate and make it available to the client.



To revoke a certificate, the client sends a signed revocation request indicating the certificate to be revoked:

Note that while ACME is defined with enough flexibility to handle different types of identifiers in principle, the primary use case addressed by this document is the case where domain names are used as identifiers. For example, all of the identifier validation challenges described in Section 8 below address validation of domain names. The use of ACME for other identifiers will require further specification in order to describe how these identifiers are encoded in the protocol and what types of validation challenges the server might require.

5. Character Encoding

All requests and responses sent via HTTP by ACME clients, ACME servers, and validation servers as well as any inputs for digest computations MUST be encoded using the UTF-8 [RFC3629] character set.

6. Message Transport

Communications between an ACME client and an ACME server are done over HTTPS, using JSON Web Signature (JWS) [RFC7515] to provide some additional security properties for messages sent from the client to the server. HTTPS provides server authentication and confidentiality. With some ACME-specific extensions, JWS provides authentication of the client's request payloads, anti-replay protection, and integrity for the HTTPS request URL.

6.1. HTTPS Requests

Each ACME function is accomplished by the client sending a sequence of HTTPS requests to the server, carrying JSON messages [RFC2818][RFC7159]. Use of HTTPS is REQUIRED. Each subsection of Section 7 below describes the message formats used by the function and the order in which messages are sent.

In most HTTPS transactions used by ACME, the ACME client is the HTTPS client and the ACME server is the HTTPS server. The ACME server acts as an HTTP and HTTPS client when validating challenges via HTTP.

ACME servers SHOULD follow the recommendations of [RFC7525] when configuring their TLS implementations. ACME servers that support TLS 1.3 MAY allow clients to send early data (0-RTT). This is safe

because the ACME protocol itself includes anti-replay protections (see Section 6.4).

ACME clients SHOULD send a User-Agent header in accordance with [RFC7231], including the name and version of the ACME software in addition to the name and version of the underlying HTTP client software.

ACME clients SHOULD send an Accept-Language header in accordance with [RFC7231] to enable localization of error messages.

ACME servers that are intended to be generally accessible need to use Cross-Origin Resource Sharing (CORS) in order to be accessible from browser-based clients [W3C.CR-cors-20130129]. Such servers SHOULD set the Access-Control-Allow-Origin header field to the value "*".

Binary fields in the JSON objects used by ACME are encoded using base64url encoding described in [RFC4648] Section 5, according to the profile specified in JSON Web Signature [RFC7515] Section 2. This encoding uses a URL safe character set. Trailing '=' characters MUST be stripped.

6.2. Request Authentication

All ACME requests with a non-empty body MUST encapsulate their payload in a JSON Web Signature (JWS) [RFC7515] object, signed using the account's private key unless otherwise specified. The server MUST verify the JWS before processing the request. Encapsulating request bodies in JWS provides authentication of requests.

JWS objects sent in ACME requests MUST meet the following additional criteria:

- The JWS MUST be in the Flattened JSON Serialization [RFC7515]
- The JWS MUST NOT have the value "none" in its "alg" field
- The JWS MUST NOT have multiple signatures
- The JWS Unencoded Payload Option [RFC7797] MUST NOT be used
- The JWS Unprotected Header [RFC7515] MUST NOT be used
- The JWS MUST NOT have a Message Authentication Code (MAC)-based algorithm in its "alg" field
- The JWS Payload MUST NOT be detached

- The JWS Protected Header MUST include the following fields:
  - "alg" (Algorithm)
  - "jwk" (JSON Web Key, for all requests not signed using an existing account, e.g. newAccount)
  - "kid" (Key ID, for all requests signed using an existing account)
  - "nonce" (defined in Section 6.4 below)
  - "url" (defined in Section 6.3 below)

The "jwk" and "kid" fields are mutually exclusive. Servers MUST reject requests that contain both.

For newAccount requests, and for revokeCert requests authenticated by certificate key, there MUST be a "jwk" field. This field MUST contain the public key corresponding to the private key used to sign the JWS.

For all other requests, the request is signed using an existing account and there MUST be a "kid" field. This field MUST contain the account URL received by POSTing to the newAccount resource.

Note that authentication via signed JWS request bodies implies that GET requests are not authenticated. Servers MUST NOT respond to GET requests for resources that might be considered sensitive. Account resources are the only sensitive resources defined in this specification.

If the client sends a JWS signed with an algorithm that the server does not support, then the server MUST return an error with status code 400 (Bad Request) and type "urn:ietf:params:acme:error:badSignatureAlgorithm". The problem document returned with the error MUST include an "algorithms" field with an array of supported "alg" values.

Because client requests in ACME carry JWS objects in the Flattened JSON Serialization, they must have the "Content-Type" header field set to "application/jose+json". If a request does not meet this requirement, then the server MUST return a response with status code 415 (Unsupported Media Type).

6.3. Request URL Integrity

It is common in deployment for the entity terminating TLS for HTTPS to be different from the entity operating the logical HTTPS server, with a "request routing" layer in the middle. For example, an ACME CA might have a content delivery network terminate TLS connections from clients so that it can inspect client requests for denial-of-service protection.

These intermediaries can also change values in the request that are not signed in the HTTPS request, e.g., the request URL and headers. ACME uses JWS to provide an integrity mechanism, which protects against an intermediary changing the request URL to another ACME URL.

As noted in Section 6.2 above, all ACME request objects carry a "url" header parameter in their protected header. This header parameter encodes the URL to which the client is directing the request. On receiving such an object in an HTTP request, the server MUST compare the "url" header parameter to the request URL. If the two do not match, then the server MUST reject the request as unauthorized.

Except for the directory resource, all ACME resources are addressed with URLs provided to the client by the server. In requests sent to these resources, the client MUST set the "url" header parameter to the exact string provided by the server (rather than performing any re-encoding on the URL). The server SHOULD perform the corresponding string equality check, configuring each resource with the URL string provided to clients and having the resource check that requests have the same string in their "url" header parameter.

6.3.1. "url" (URL) JWS Header Parameter

The "url" header parameter specifies the URL [RFC3986] to which this JWS object is directed. The "url" header parameter MUST be carried in the protected header of the JWS. The value of the "url" header parameter MUST be a string representing the URL.

6.4. Replay protection

In order to protect ACME resources from any possible replay attacks, ACME requests have a mandatory anti-replay mechanism. This mechanism is based on the server maintaining a list of nonces that it has issued to clients, and requiring any signed request from the client to carry such a nonce.

An ACME server provides nonces to clients using the Replay-Nonce header field, as specified in Section 6.4.1 below. The server MUST

include a Replay-Nonce header field in every successful response to a POST request and SHOULD provide it in error responses as well.

Every JWS sent by an ACME client MUST include, in its protected header, the "nonce" header parameter, with contents as defined in Section 6.4.2 below. As part of JWS verification, the ACME server MUST verify that the value of the "nonce" header is a value that the server previously provided in a Replay-Nonce header field. Once a nonce value has appeared in an ACME request, the server MUST consider it invalid, in the same way as a value it had never issued.

When a server rejects a request because its nonce value was unacceptable (or not present), it MUST provide HTTP status code 400 (Bad Request), and indicate the ACME error type "urn:ietf:params:acme:error:badNonce". An error response with the "badNonce" error type MUST include a Replay-Nonce header with a fresh nonce. On receiving such a response, a client SHOULD retry the request using the new nonce.

The precise method used to generate and track nonces is up to the server. For example, the server could generate a random 128-bit value for each response, keep a list of issued nonces, and strike nonces from this list as they are used.

6.4.1. Replay-Nonce

The "Replay-Nonce" header field includes a server-generated value that the server can use to detect unauthorized replay in future client requests. The server MUST generate the value provided in Replay-Nonce in such a way that they are unique to each message, with high probability. For instance, it is acceptable to generate Replay-Nonces randomly.

The value of the Replay-Nonce field MUST be an octet string encoded according to the base64url encoding described in Section 2 of [RFC7515]. Clients MUST ignore invalid Replay-Nonce values.

```
base64url = [A-Z] / [a-z] / [0-9] / "-" / "_"

Replay-Nonce = *base64url
```

The Replay-Nonce header field SHOULD NOT be included in HTTP request messages.

6.4.2. "nonce" (Nonce) JWS Header Parameter

The "nonce" header parameter provides a unique value that enables the verifier of a JWS to recognize when replay has occurred. The "nonce" header parameter MUST be carried in the protected header of the JWS.

The value of the "nonce" header parameter MUST be an octet string, encoded according to the base64url encoding described in Section 2 of [RFC7515]. If the value of a "nonce" header parameter is not valid according to this encoding, then the verifier MUST reject the JWS as malformed.

6.5. Rate Limits

Creation of resources can be rate limited by ACME servers to ensure fair usage and prevent abuse. Once the rate limit is exceeded, the server MUST respond with an error with the type "urn:ietf:params:acme:error:rateLimited". Additionally, the server SHOULD send a "Retry-After" header indicating when the current request may succeed again. If multiple rate limits are in place, that is the time where all rate limits allow access again for the current request with exactly the same parameters.

In addition to the human-readable "detail" field of the error response, the server MAY send one or multiple link relations in the "Link" header pointing to documentation about the specific rate limit that was hit, using the "help" link relation type.

6.6. Errors

Errors can be reported in ACME both at the HTTP layer and within challenge objects as defined in Section 8. ACME servers can return responses with an HTTP error response code (4XX or 5XX). For example: If the client submits a request using a method not allowed in this document, then the server MAY return status code 405 (Method Not Allowed).

When the server responds with an error status, it SHOULD provide additional information using a problem document [RFC7807]. To facilitate automatic response to errors, this document defines the following standard tokens for use in the "type" field (within the "urn:ietf:params:acme:error:" namespace):

Type	Description
badCSR	The CSR is unacceptable (e.g., due to a short key)

badNonce	The client sent an unacceptable anti-replay nonce
badSignatureAlgorithm	The JWS was signed with an algorithm the server does not support
invalidContact	A contact URL for an account was invalid
unsupportedContact	A contact URL for an account used an unsupported protocol scheme
externalAccountRequired	The request must include a value for the "externalAccountBinding" field
accountDoesNotExist	The request specified an account that does not exist
malformed	The request message was malformed
rateLimited	The request exceeds a rate limit
rejectedIdentifier	The server will not issue for the identifier
serverInternal	The server experienced an internal error
unauthorized	The client lacks sufficient authorization
unsupportedIdentifier	Identifier is not supported, but may be in future
userActionRequired	Visit the "instance" URL and take actions specified there
badRevocationReason	The revocation reason provided is not allowed by the server
caa	Certification Authority Authorization (CAA) records forbid the CA from issuing
dns	There was a problem with a DNS query
connection	The server could not connect to

	validation target	
tls	The server received a TLS error during validation	
incorrectResponse	Response received didn't match the challenge's requirements	

This list is not exhaustive. The server MAY return errors whose "type" field is set to a URI other than those defined above. Servers MUST NOT use the ACME URN [RFC3553] namespace for errors other than the standard types. Clients SHOULD display the "detail" field of all errors.

In the remainder of this document, we use the tokens in the table above to refer to error types, rather than the full URNs. For example, an "error of type 'badCSR'" refers to an error document with "type" value "urn:ietf:params:acme:error:badCSR".

6.6.1. Subproblems

Sometimes a CA may need to return multiple errors in response to a request. Additionally, the CA may need to attribute errors to specific identifiers. For instance, a new-order request may contain multiple identifiers for which the CA cannot issue. In this situation, an ACME problem document MAY contain the "subproblems" field, containing a JSON array of problem documents, each of which MAY contain an "identifier" field. If present, the "identifier" field MUST contain an ACME identifier (Section 9.7.7). The "identifier" field MUST NOT be present at the top level in ACME problem documents. It can only be present in subproblems. Subproblems need not all have the same type, and do not need to match the top level type.

ACME clients may choose to use the "identifier" field of a subproblem as a hint that an operation would succeed if that identifier were omitted. For instance, if an order contains ten DNS identifiers, and the new-order request returns a problem document with two subproblems, referencing two of those identifiers, the ACME client may choose to submit another order containing only the eight identifiers not listed in the problem document.

```
HTTP/1.1 403 Forbidden
Content-Type: application/problem+json

{
    "type": "urn:ietf:params:acme:error:malformed",
    "detail": "Some of the identifiers requested were rejected",
    "subproblems": [
        {
            "type": "urn:ietf:params:acme:error:malformed",
            "detail": "Invalid underscore in DNS name \"_example.com\"",
            "identifier": {
                "type": "dns",
                "value": "_example.com"
            }
        },
        {
            "type": "urn:ietf:params:acme:error:rejectedIdentifier",
            "detail": "This CA will not issue for \"example.net\"",
            "identifier": {
                "type": "dns",
                "value": "example.net"
            }
        }
    ]
}
```

7. Certificate Management

In this section, we describe the certificate management functions that ACME enables:

- Account Creation
- Ordering a Certificate
- Identifier Authorization
- Certificate Issuance
- Certificate Revocation

7.1. Resources

ACME is structured as a REST application with the following types of resources:

- Account resources, representing information about an account (Section 7.1.2, Section 7.3)

- Order resources, representing an account's requests to issue certificates (Section 7.1.3)
- Authorization resources, representing an account's authorization to act for an identifier (Section 7.1.4)
- Challenge resources, representing a challenge to prove control of an identifier (Section 7.5, Section 8)
- Certificate resources, representing issued certificates (Section 7.4.2)
- A "directory" resource (Section 7.1.1)
- A "newNonce" resource (Section 7.2)
- A "newAccount" resource (Section 7.3)
- A "newOrder" resource (Section 7.4)
- A "revokeCert" resource (Section 7.6)
- A "keyChange" resource (Section 7.3.6)

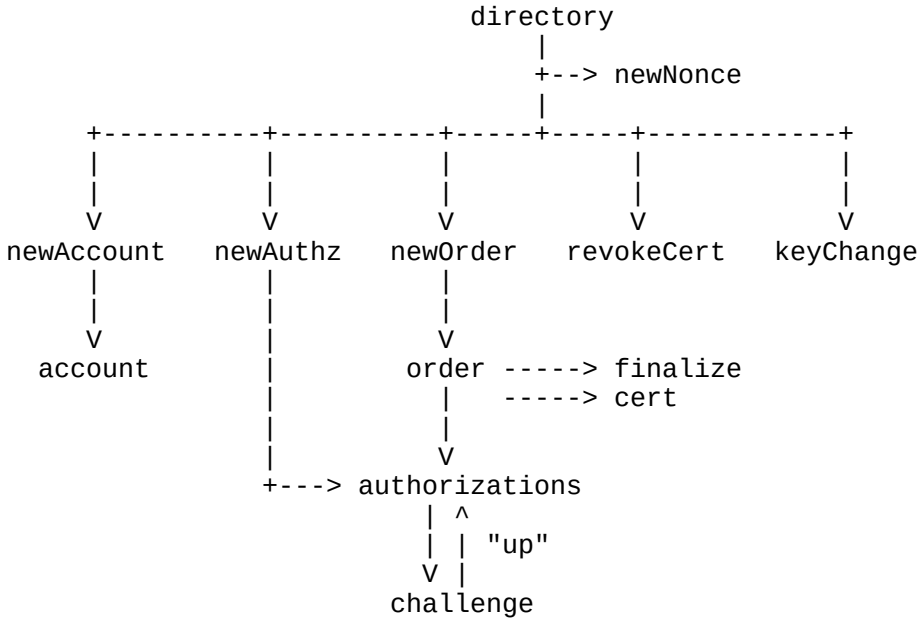
The server MUST provide "directory" and "newNonce" resources.

ACME uses different URLs for different management functions. Each function is listed in a directory along with its corresponding URL, so clients only need to be configured with the directory URL. These URLs are connected by a few different link relations [RFC5988].

The "up" link relation is used with challenge resources to indicate the authorization resource to which a challenge belongs. It is also used from certificate resources to indicate a resource from which the client may fetch a chain of CA certificates that could be used to validate the certificate in the original resource.

The "index" link relation is present on all resources other than the directory and indicates the URL of the directory.

The following diagram illustrates the relations between resources on an ACME server. For the most part, these relations are expressed by URLs provided as strings in the resources' JSON representations. Lines with labels in quotes indicate HTTP link relations.



The following table illustrates a typical sequence of requests required to establish a new account with the server, prove control of an identifier, issue a certificate, and fetch an updated certificate some time after issuance. The "->" is a mnemonic for a Location header pointing to a created resource.

Action	Request	Response
Get directory	GET directory	200
Get nonce	HEAD newNonce	200
Create account	POST newAccount	201 -> account
Submit order	POST newOrder	201 -> order
Fetch challenges	GET order authorizations	200
Respond to challenges	POST challenge urls	200
Poll for status	GET order	200
Finalize order	POST order finalize	200
Poll for status	GET order	200
Download certificate	GET order certificate	200

The remainder of this section provides the details of how these resources are structured and how the ACME protocol makes use of them.

7.1.1. Directory

In order to help clients configure themselves with the right URLs for each ACME operation, ACME servers provide a directory object. This should be the only URL needed to configure clients. It is a JSON object, whose field names are drawn from the following table and whose values are the corresponding URLs.

Field	URL in value
newNonce	New nonce
newAccount	New account
newOrder	New order
newAuthz	New authorization
revokeCert	Revoke certificate
keyChange	Key change

There is no constraint on the URL of the directory except that it should be different from the other ACME server resources' URLs, and that it should not clash with other services. For instance:

- a host which functions as both an ACME and a Web server may want to keep the root path "/" for an HTML "front page", and place the ACME directory under the path "/acme".
- a host which only functions as an ACME server could place the directory under the path "/".

If the ACME server does not implement pre-authorization (Section 7.4.1) it MUST omit the "newAuthz" field of the directory.

The object MAY additionally contain a field "meta". If present, it MUST be a JSON object; each field in the object is an item of metadata relating to the service provided by the ACME server.

The following metadata items are defined, all of which are OPTIONAL:

termsOfService (optional, string): A URL identifying the current terms of service.

website (optional, string): An HTTP or HTTPS URL locating a website providing more information about the ACME server.

caaIdentities (optional, array of string): Each string MUST be a lowercase hostname which the ACME server recognizes as referring to itself for the purposes of CAA record validation as defined in [RFC6844]. This allows clients to determine the correct issuer domain name to use when configuring CAA records.

externalAccountRequired (optional, boolean): If this field is present and set to "true", then the CA requires that all new-account requests include an "externalAccountBinding" field associating the new account with an external account.

Clients access the directory by sending a GET request to the directory URL.

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "newNonce": "https://example.com/acme/new-nonce",
  "newAccount": "https://example.com/acme/new-account",
  "newOrder": "https://example.com/acme/new-order",
  "newAuthz": "https://example.com/acme/new-authz",
  "revokeCert": "https://example.com/acme/revoke-cert",
  "keyChange": "https://example.com/acme/key-change",
  "meta": {
    "termsOfService": "https://example.com/acme/terms/2017-5-30",
    "website": "https://www.example.com/",
    "caaIdentities": ["example.com"],
    "externalAccountRequired": false
  }
}
```

7.1.2. Account Objects

An ACME account resource represents a set of metadata associated with an account. Account resources have the following structure:

status (required, string): The status of this account. Possible values are: "valid", "deactivated", and "revoked". The value "deactivated" should be used to indicate client-initiated deactivation whereas "revoked" should be used to indicate server-initiated deactivation.

contact (optional, array of string): An array of URLs that the server can use to contact the client for issues related to this account. For example, the server may wish to notify the client about server-initiated revocation or certificate expiration.

termsOfServiceAgreed (optional, boolean): Including this field in a new-account request, with a value of true, indicates the client’s agreement with the terms of service. This field is not updateable by the client.

orders (required, string): A URL from which a list of orders submitted by this account can be fetched via a GET request, as described in Section 7.1.2.1.

```
{
  "status": "valid",
  "contact": [
    "mailto:cert-admin@example.com",
    "mailto:admin@example.com"
  ],
  "termsOfServiceAgreed": true,
  "orders": "https://example.com/acme/acct/1/orders"
}
```

7.1.2.1. Orders List

Each account object includes an "orders" URL from which a list of orders created by the account can be fetched via GET request. The result of the GET request MUST be a JSON object whose "orders" field is an array of URLs, each identifying an order belonging to the account. The server SHOULD include pending orders, and SHOULD NOT include orders that are invalid in the array of URLs. The server MAY return an incomplete list, along with a Link header with a "next" link relation indicating where further entries can be acquired.

```
HTTP/1.1 200 OK
Content-Type: application/json
Link: <https://example.com/acme/acct/1/orders?cursor=2>;rel="next"

{
  "orders": [
    "https://example.com/acme/acct/1/order/1",
    "https://example.com/acme/acct/1/order/2",
    /* 47 more URLs not shown for example brevity */
    "https://example.com/acme/acct/1/order/50"
  ]
}
```

7.1.3. Order Objects

An ACME order object represents a client’s request for a certificate and is used to track the progress of that order through to issuance. Thus, the object contains information about the requested certificate, the authorizations that the server requires the client to complete, and any certificates that have resulted from this order.

status (required, string): The status of this order. Possible values are: "pending", "ready", "processing", "valid", and "invalid".

expires (optional, string): The timestamp after which the server will consider this order invalid, encoded in the format specified in RFC 3339 [RFC3339]. This field is REQUIRED for objects with "pending" or "valid" in the status field.

identifiers (required, array of object): An array of identifier objects that the order pertains to.

type (required, string): The type of identifier.

value (required, string): The identifier itself.

notBefore (optional, string): The requested value of the notBefore field in the certificate, in the date format defined in [RFC3339].

notAfter (optional, string): The requested value of the notAfter field in the certificate, in the date format defined in [RFC3339].

error (optional, object): The error that occurred while processing the order, if any. This field is structured as a problem document [RFC7807].

authorizations (required, array of string): For pending orders, the authorizations that the client needs to complete before the requested certificate can be issued (see Section 7.5). For final orders (in the "valid" or "invalid" state), the authorizations that were completed. Each entry is a URL from which an authorization can be fetched with a GET request.

finalize (required, string): A URL that a CSR must be POSTed to once all of the order’s authorizations are satisfied to finalize the order. The result of a successful finalization will be the population of the certificate URL for the order.

certificate (optional, string): A URL for the certificate that has been issued in response to this order.

```
{
  "status": "valid",
  "expires": "2015-03-01T14:09:00Z",

  "identifiers": [
    { "type": "dns", "value": "example.com" },
    { "type": "dns", "value": "www.example.com" }
  ],

  "notBefore": "2016-01-01T00:00:00Z",
  "notAfter": "2016-01-08T00:00:00Z",

  "authorizations": [
    "https://example.com/acme/authz/1234",
    "https://example.com/acme/authz/2345"
  ],

  "finalize": "https://example.com/acme/acct/1/order/1/finalize",

  "certificate": "https://example.com/acme/cert/1234"
}
```

Any identifier of type "dns" in a new-order request MAY have a wildcard domain name as its value. A wildcard domain name consists of a single asterisk character followed by a single full stop character ("*.") followed by a domain name as defined for use in the Subject Alternate Name Extension by RFC 5280 [RFC5280]. An authorization returned by the server for a wildcard domain name identifier MUST NOT include the asterisk and full stop ("*.") prefix in the authorization identifier value. The returned authorization MUST include the optional "wildcard" field, with a value of true.

The elements of the "authorizations" and "identifiers" array are immutable once set. The server MUST NOT change the contents of either array after they are created. If a client observes a change in the contents of either array, then it SHOULD consider the order invalid.

The "authorizations" array of the order SHOULD reflect all authorizations that the CA takes into account in deciding to issue, even if some authorizations were fulfilled in earlier orders or in pre-authorization transactions. For example, if a CA allows multiple orders to be fulfilled based on a single authorization transaction, then it SHOULD reflect that authorization in all of the orders.

7.1.4. Authorization Objects

An ACME authorization object represents a server's authorization for an account to represent an identifier. In addition to the identifier, an authorization includes several metadata fields, such as the status of the authorization (e.g., "pending", "valid", or "revoked") and which challenges were used to validate possession of the identifier.

The structure of an ACME authorization resource is as follows:

identifier (required, object): The identifier that the account is authorized to represent

type (required, string): The type of identifier.

value (required, string): The identifier itself.

status (required, string): The status of this authorization. Possible values are: "pending", "valid", "invalid", "deactivated", "expired", and "revoked".

expires (optional, string): The timestamp after which the server will consider this authorization invalid, encoded in the format specified in RFC 3339 [RFC3339]. This field is REQUIRED for objects with "valid" in the "status" field.

challenges (required, array of objects): For pending authorizations, the challenges that the client can fulfill in order to prove possession of the identifier. For final authorizations (in the "valid" or "invalid" state), the challenges that were used. Each array entry is an object with parameters required to validate the challenge. A client should attempt to fulfill one of these challenges, and a server should consider any one of the challenges sufficient to make the authorization valid. For final authorizations, it contains the challenges that were successfully completed.

wildcard (optional, boolean): For authorizations created as a result of a newOrder request containing a DNS identifier with a value that contained a wildcard prefix this field MUST be present, and true.

The only type of identifier defined by this specification is a fully-qualified domain name (type: "dns"). If a domain name contains non-ASCII Unicode characters it MUST be encoded using the rules defined in [RFC3492]. Servers MUST verify any identifier values that begin with the ASCII Compatible Encoding prefix "xn--" as defined in

[RFC5890] are properly encoded. Wildcard domain names (with "*" as the first label) MUST NOT be included in authorization objects. If an authorization object conveys authorization for the base domain of a newOrder DNS type identifier with a wildcard prefix then the optional authorizations "wildcard" field MUST be present with a value of true.

Section 8 describes a set of challenges for domain name validation.

```
{
  "status": "valid",
  "expires": "2015-03-01T14:09:00Z",

  "identifier": {
    "type": "dns",
    "value": "example.org"
  },

  "challenges": [
    {
      "url": "https://example.com/acme/authz/1234/0",
      "type": "http-01",
      "status": "valid",
      "token": "DGyRejmCefe7v4NfDGDKfA",
      "validated": "2014-12-01T12:05:00Z"
    }
  ],

  "wildcard": false
}
```

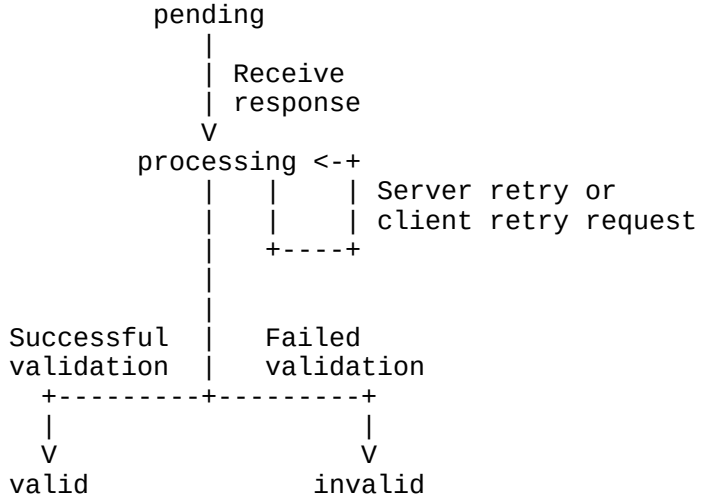
7.1.5. Challenge Objects

An ACME challenge object represents a server's offer to validate a client's possession of an identifier in a specific way. Unlike the other objects listed above, there is not a single standard structure for a challenge object. The contents of a challenge object depend on the validation method being used. The general structure of challenge objects and an initial set of validation methods are described in Section 8.

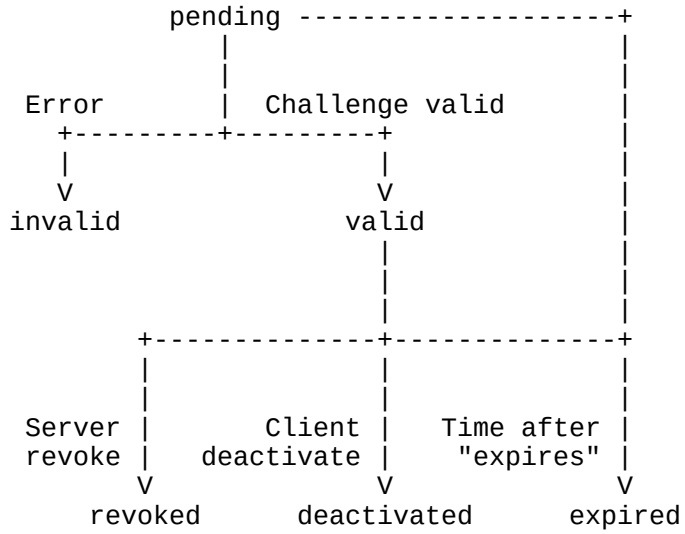
7.1.6. Status Changes

Each ACME object type goes through a simple state machine over its lifetime. The "status" field of the object indicates which state the object is currently in.

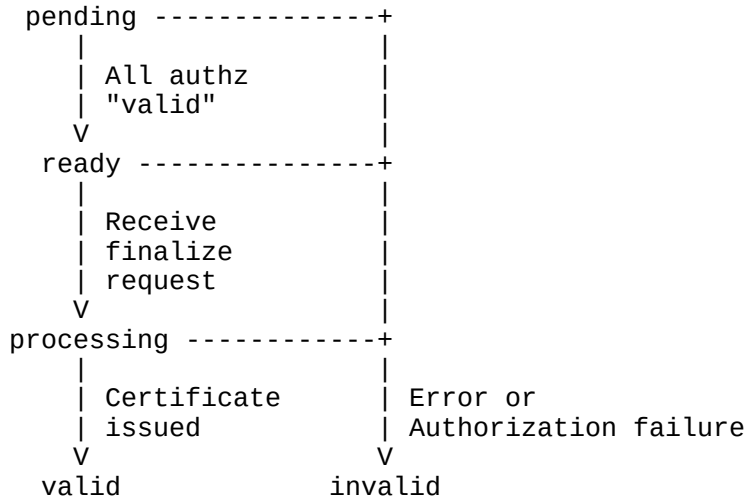
Challenge objects are created in the "pending" state. They transition to the "processing" state when the client responds to the challenge (see Section 7.5.1) and the server begins attempting to validate that the client has completed the challenge. Note that within the "processing" state, the server may attempt to validate the challenge multiple times (see Section 8.2). Likewise, client requests for retries do not cause a state change. If validation is successful, the challenge moves to the "valid" state; if there is an error, the challenge moves to the "invalid" state.



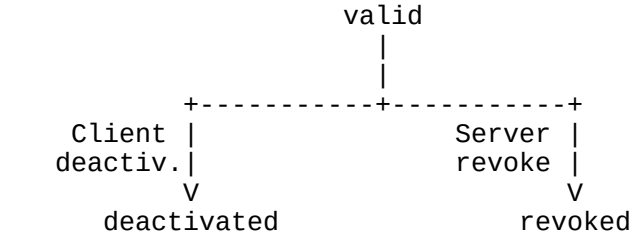
Authorization objects are created in the "pending" state. If one of the challenges listed in the authorization transitions to the "valid" state, then the authorization also changes to the "valid" state. If there is an error while the authorization is still pending, then the authorization transitions to the "invalid" state. Once the authorization is in the valid state, it can expire ("expired"), be deactivated by the client ("deactivated", see Section 7.5.2), or revoked by the server ("revoked").



Order objects are created in the "pending" state. Once all of the authorizations listed in the order object are in the "valid" state, the order transitions to the "ready" state. The order moves to the "processing" state after the client submits a request to the order's "finalize" URL and the CA begins the issuance process for the certificate. Once the certificate is issued, the order enters the "valid" state. If an error occurs at any of these stages, the order moves to the "invalid" state. The order also moves to the "invalid" state if it expires, or one of its authorizations enters a final state other than "valid" ("expired", "revoked", "deactivated").



Account objects are created in the "valid" state, since no further action is required to create an account after a successful newAccount request. If the account is deactivated by the client or revoked by the server, it moves to the corresponding state.



Note that some of these states may not ever appear in a "status" field, depending on server behavior. For example, a server that issues synchronously will never show an order in the "processing" state. A server that deletes expired authorizations immediately will never show an authorization in the "expired" state.

7.2. Getting a Nonce

Before sending a POST request to the server, an ACME client needs to have a fresh anti-replay nonce to put in the "nonce" header of the JWS. In most cases, the client will have gotten a nonce from a previous request. However, the client might sometimes need to get a new nonce, e.g., on its first request to the server or if an existing nonce is no longer valid.

To get a fresh nonce, the client sends a HEAD request to the new-nonce resource on the server. The server's response MUST include a Replay-Nonce header field containing a fresh nonce, and SHOULD have status code 200 (OK). The server SHOULD also respond to GET requests for this resource, returning an empty body (while still providing a Replay-Nonce header) with a 204 (No Content) status.

```
HEAD /acme/new-nonce HTTP/1.1
Host: example.com

HTTP/1.1 200 OK
Replay-Nonce: oFvnlFP1wIhRlYS2jTaXbA
Cache-Control: no-store
```

Proxy caching of responses from the new-nonce resource can cause clients receive the same nonce repeatedly, leading to badNonce errors. The server MUST include a Cache-Control header field with the "no-store" directive in responses for the new-nonce resource, in order to prevent caching of this resource.

7.3. Account Creation

A client creates a new account with the server by sending a POST request to the server's new-account URL. The body of the request is a stub account object optionally containing the "contact" and "termsOfServiceAgreed" fields.

contact (optional, array of string): Same meaning as the corresponding server field defined in Section 7.1.2

termsOfServiceAgreed (optional, boolean): Same meaning as the corresponding server field defined in Section 7.1.2

onlyReturnExisting (optional, boolean): If this field is present with the value "true", then the server MUST NOT create a new account if one does not already exist. This allows a client to look up an account URL based on an account key (see Section 7.3.1).

externalAccountBinding (optional, object): An optional field for binding the new account with an existing non-ACME account (see Section 7.3.5).

```
POST /acme/new-account HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url({
    "alg": "ES256",
    "jwk": {...},
    "nonce": "6S8IqOGY7eL2lsGoTZYifg",
    "url": "https://example.com/acme/new-account"
  }),
  "payload": base64url({
    "termsOfServiceAgreed": true,
    "contact": [
      "mailto:cert-admin@example.com",
      "mailto:admin@example.com"
    ]
  }),
  "signature": "RZPOnYoPs1PhjszF...-nh6X1qtOFPB519I"
}
```

The server MUST ignore any values provided in the "orders" fields in account bodies sent by the client, as well as any other fields that it does not recognize. If new fields are specified in the future,

the specification of those fields MUST describe whether they can be provided by the client.

In general, the server MUST ignore any fields in the request object that it does not recognize. In particular, it MUST NOT reflect unrecognized fields in the resulting account object. This allows clients to detect when servers do not support an extension field.

The server SHOULD validate that the contact URLs in the "contact" field are valid and supported by the server. If the server validates contact URLs it MUST support the "mailto" scheme. Clients MUST NOT provide a "mailto" URL in the "contact" field that contains "hfields" [RFC6068], or more than one "addr-spec" in the "to" component. If a server encounters a "mailto" contact URL that does not meet these criteria, then it SHOULD reject it as invalid.

If the server rejects a contact URL for using an unsupported scheme it MUST return an error of type "unsupportedContact", with a description describing the error and what types of contact URLs the server considers acceptable. If the server rejects a contact URL for using a supported scheme but an invalid value then the server MUST return an error of type "invalidContact".

If the server wishes to present the client with terms under which the ACME service is to be used, it MUST indicate the URL where such terms can be accessed in the "termsOfService" subfield of the "meta" field in the directory object, and the server MUST reject new-account requests that do not have the "termsOfServiceAgreed" field set to "true". Clients SHOULD NOT automatically agree to terms by default. Rather, they SHOULD require some user interaction for agreement to terms.

The server creates an account and stores the public key used to verify the JWS (i.e., the "jwk" element of the JWS header) to authenticate future requests from the account. The server returns this account object in a 201 (Created) response, with the account URL in a Location header field. The account URL is used as the "kid" value in the JWS authenticating subsequent requests by this account (See Section 6.2).

```
HTTP/1.1 201 Created
Content-Type: application/json
Replay-Nonce: D8s4D2mLs8Vn-goWuPQeKA
Location: https://example.com/acme/acct/1
Link: <https://example.com/acme/some-directory>;rel="index"

{
  "status": "valid",

  "contact": [
    "mailto:cert-admin@example.com",
    "mailto:admin@example.com"
  ],

  "orders": "https://example.com/acme/acct/1/orders"
}
```

7.3.1. Finding an Account URL Given a Key

If the server already has an account registered with the provided account key, then it MUST return a response with a 200 (OK) status code and provide the URL of that account in the Location header field. This allows a client that has an account key but not the corresponding account URL to recover the account URL.

If a client wishes to find the URL for an existing account and does not want an account to be created if one does not already exist, then it SHOULD do so by sending a POST request to the new-account URL with a JWS whose payload has an "onlyReturnExisting" field set to "true" ({"onlyReturnExisting": true}). If a client sends such a request and an account does not exist, then the server MUST return an error response with status code 400 (Bad Request) and type "urn:ietf:params:acme:error:accountDoesNotExist".

7.3.2. Account Update

If the client wishes to update this information in the future, it sends a POST request with updated information to the account URL. The server MUST ignore any updates to the "orders" field or any other fields it does not recognize. If the server accepts the update, it MUST return a response with a 200 (OK) status code and the resulting account object.

For example, to update the contact information in the above account, the client could send the following request:

```
POST /acme/acct/1 HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url({
    "alg": "ES256",
    "kid": "https://example.com/acme/acct/1",
    "nonce": "ax5RnthDqp_Yf4_HZnFLmA",
    "url": "https://example.com/acme/acct/1"
  }),
  "payload": base64url({
    "contact": [
      "mailto:certificates@example.com",
      "mailto:admin@example.com"
    ]
  }),
  "signature": "hDXzvcj8T6fbFbmn...rDzXzzvzpRy64N0o"
}
```

7.3.3. Account Information

Servers MUST NOT respond to GET requests for account resources as these requests are not authenticated. If a client wishes to query the server for information about its account (e.g., to examine the "contact" or "orders" fields), then it SHOULD do so by sending a POST request with an empty update. That is, it should send a JWS whose payload is an empty object ({}).

7.3.4. Changes of Terms of Service

As described above, a client can indicate its agreement with the CA's terms of service by setting the "termsOfServiceAgreed" field in its account object to "true".

If the server has changed its terms of service since a client initially agreed, and the server is unwilling to process a request without explicit agreement to the new terms, then it MUST return an error response with status code 403 (Forbidden) and type "urn:ietf:params:acme:error:userActionRequired". This response MUST include a Link header with link relation "terms-of-service" and the latest terms-of-service URL.

The problem document returned with the error MUST also include an "instance" field, indicating a URL that the client should direct a human user to visit in order for instructions on how to agree to the terms.

```
HTTP/1.1 403 Forbidden
Replay-Nonce: IXVHDyxIRGcTE0VSblhPzw
Link: <https://example.com/acme/terms/2017-6-02>;rel="terms-of-service"
Content-Type: application/problem+json
Content-Language: en

{
  "type": "urn:ietf:params:acme:error:userActionRequired",
  "detail": "Terms of service have changed",
  "instance": "https://example.com/acme/agreement/?token=W8Ih3PswD-8"
}
```

7.3.5. External Account Binding

The server MAY require a value for the "externalAccountBinding" field to be present in "newAccount" requests. This can be used to associate an ACME account with an existing account in a non-ACME system, such as a CA customer database.

To enable ACME account binding, a CA needs to provide the ACME client with a MAC key and a key identifier, using some mechanism outside of ACME. The key identifier MUST be an ASCII string. The MAC key SHOULD be provided in base64url-encoded form, to maximize compatibility between non-ACME provisioning systems and ACME clients.

The ACME client then computes a binding JWS to indicate the external account holder's approval of the ACME account key. The payload of this JWS is the account key being registered, in JWK form. The protected header of the JWS MUST meet the following criteria:

- The "alg" field MUST indicate a MAC-based algorithm
- The "kid" field MUST contain the key identifier provided by the CA
- The "nonce" field MUST NOT be present
- The "url" field MUST be set to the same value as the outer JWS

The "signature" field of the JWS will contain the MAC value computed with the MAC key provided by the CA.

```
POST /acme/new-account HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url({
    "alg": "ES256",
    "jwk": /* account key */,
    "nonce": "K60BWPrMQG9SDxBDS_xtSw",
    "url": "https://example.com/acme/new-account"
  }),
  "payload": base64url({
    "contact": ["mailto:example@anonymous.invalid"],
    "termsOfServiceAgreed": true,

    "externalAccountBinding": {
      "protected": base64url({
        "alg": "HS256",
        "kid": /* key identifier from CA */,
        "url": "https://example.com/acme/new-account"
      }),
      "payload": base64url(/* same as in "jwk" above */),
      "signature": /* MAC using MAC key from CA */
    }
  }),
  "signature": "5TWiqIYQfIDfALQv...x9C2mg8JGPxl5bI4"
}
```

If a CA requires that new-account requests contain an "externalAccountBinding" field, then it MUST provide the value "true" in the "externalAccountRequired" subfield of the "meta" field in the directory object. If the CA receives a new-account request without an "externalAccountBinding" field, then it SHOULD reply with an error of type "externalAccountRequired".

When a CA receives a new-account request containing an "externalAccountBinding" field, it decides whether or not to verify the binding. If the CA does not verify the binding, then it MUST NOT reflect the "externalAccountBinding" field in the resulting account object (if any). To verify the account binding, the CA MUST take the following steps:

1. Verify that the value of the field is a well-formed JWS

2. Verify that the JWS protected field meets the above criteria

3. Retrieve the MAC key corresponding to the key identifier in the "kid" field

4. Verify that the MAC on the JWS verifies using that MAC key

5. Verify that the payload of the JWS represents the same key as was used to verify the outer JWS (i.e., the "jwk" field of the outer JWS)

If all of these checks pass and the CA creates a new account, then the CA may consider the new account associated with the external account corresponding to the MAC key and MUST reflect the value of the "externalAccountBinding" field in the resulting account object. If any of these checks fail, then the CA MUST reject the new-account request.

7.3.6. Account Key Roll-over

A client may wish to change the public key that is associated with an account in order to recover from a key compromise or proactively mitigate the impact of an unnoticed key compromise.

To change the key associated with an account, the client first constructs a key-change object describing the change that it would like the server to make:

account (required, string): The URL for the account being modified. The content of this field MUST be the exact string provided in the Location header field in response to the new-account request that created the account.

newKey (required, JWK): The JWK representation of the new key

The client then encapsulates the key-change object in an "inner" JWS, signed with the requested new account key (i.e., the key matching the "newKey" value). This JWS then becomes the payload for the "outer" JWS that is the body of the ACME request.

The outer JWS MUST meet the normal requirements for an ACME JWS (see Section 6.2). The inner JWS MUST meet the normal requirements, with the following differences:

- The inner JWS MUST have a "jwk" header parameter, containing the public key of the new key pair (i.e., the same value as the "newKey" field).
- The inner JWS MUST have the same "url" header parameter as the outer JWS.

- The inner JWS is NOT REQUIRED to have a "nonce" header parameter. The server MUST ignore any value provided for the "nonce" header parameter.

This transaction has signatures from both the old and new keys so that the server can verify that the holders of the two keys both agree to the change. The signatures are nested to preserve the property that all signatures on POST messages are signed by exactly one key.

```
POST /acme/key-change HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url({
    "alg": "ES256",
    "kid": "https://example.com/acme/acct/1",
    "nonce": "K60BWPrMQG9SDxBDS_xtSw",
    "url": "https://example.com/acme/key-change"
  }),
  "payload": base64url({
    "protected": base64url({
      "alg": "ES256",
      "jwk": /* new key */,
      "url": "https://example.com/acme/key-change"
    }),
    "payload": base64url({
      "account": "https://example.com/acme/acct/1",
      "newKey": /* new key */
    }),
    "signature": "Xe8B94RD30Azj2ea...8BmZIRtcSKPSd8gU"
  }),
  "signature": "5TWiqIYQfIDfALQv...x9C2mg8JGPxl5bI4"
}
```

On receiving key-change request, the server MUST perform the following steps in addition to the typical JWS validation:

1. Validate the POST request belongs to a currently active account, as described in Section 6.

2. Check that the payload of the JWS is a well-formed JWS object (the "inner JWS").

3. Check that the JWS protected header of the inner JWS has a "jwk" field.

4. Check that the inner JWS verifies using the key in its "jwk" field.

5. Check that the payload of the inner JWS is a well-formed key-change object (as described above).

6. Check that the "url" parameters of the inner and outer JWSs are the same.

7. Check that the "account" field of the key-change object contains the URL for the account matching the old key.

8. Check that the "newKey" field of the key-change object also verifies the inner JWS.

9. Check that no account exists whose account key is the same as the key in the "newKey" field.

If all of these checks pass, then the server updates the corresponding account by replacing the old account key with the new public key and returns status code 200 (OK). Otherwise, the server responds with an error status code and a problem document describing the error. If there is an existing account with the new key provided, then the server SHOULD use status code 409 (Conflict) and provide the URL of that account in the Location header field.

Note that changing the account key for an account SHOULD NOT have any other impact on the account. For example, the server MUST NOT invalidate pending orders or authorization transactions based on a change of account key.

7.3.7. Account Deactivation

A client can deactivate an account by posting a signed update to the server with a status field of "deactivated." Clients may wish to do this when the account key is compromised or decommissioned.

```
POST /acme/acct/1 HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url({
    "alg": "ES256",
    "kid": "https://example.com/acme/acct/1",
    "nonce": "ntuJWWSic4WVNSqeUmshgg",
    "url": "https://example.com/acme/acct/1"
  }),
  "payload": base64url({
    "status": "deactivated"
  }),
  "signature": "earzVLd3m5M4xJzR...bVTqn7R08AKOVf3Y"
}
```

The server MUST verify that the request is signed by the account key. If the server accepts the deactivation request, it replies with a 200 (OK) status code and the current contents of the account object.

Once an account is deactivated, the server MUST NOT accept further requests authorized by that account's key. The server SHOULD cancel any pending operations authorized by the account's key, such as certificate orders. A server may take a variety of actions in response to an account deactivation, e.g., deleting data related to that account or sending mail to the account's contacts. Servers SHOULD NOT revoke certificates issued by the deactivated account, since this could cause operational disruption for servers using these certificates. ACME does not provide a way to reactivate a deactivated account.

7.4. Applying for Certificate Issuance

The client requests certificate issuance by sending a POST request to the server's new-order resource. The body of the POST is a JWS object whose JSON payload is a subset of the order object defined in Section 7.1.3, containing the fields that describe the certificate to be issued:

identifiers (required, array of object): An array of identifier objects that the client wishes to submit an order for.

type (required, string): The type of identifier.

value (required, string): The identifier itself.

notBefore (optional, string): The requested value of the notBefore field in the certificate, in the date format defined in [RFC3339].

notAfter (optional, string): The requested value of the notAfter field in the certificate, in the date format defined in [RFC3339].

```
POST /acme/new-order HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url({
    "alg": "ES256",
    "kid": "https://example.com/acme/acct/1",
    "nonce": "5XJ1L3lEkMG7tR6pA00clA",
    "url": "https://example.com/acme/new-order"
  }),
  "payload": base64url({
    "identifiers": [
      { "type": "dns", "value": "example.com" }
    ],
    "notBefore": "2016-01-01T00:00:00Z",
    "notAfter": "2016-01-08T00:00:00Z"
  }),
  "signature": "H6ZXtGjTZyUnPeKn...wEA4TklBdh3e454g"
}
```

The server MUST return an error if it cannot fulfill the request as specified, and MUST NOT issue a certificate with contents other than those requested. If the server requires the request to be modified in a certain way, it should indicate the required changes using an appropriate error type and description.

If the server is willing to issue the requested certificate, it responds with a 201 (Created) response. The body of this response is an order object reflecting the client’s request and any authorizations the client must complete before the certificate will be issued.

```
HTTP/1.1 201 Created
Replay-Nonce: MYAuvOpaoIiywTezizk5vw
Location: https://example.com/acme/order/asdf

{
  "status": "pending",
  "expires": "2016-01-01T00:00:00Z",

  "notBefore": "2016-01-01T00:00:00Z",
  "notAfter": "2016-01-08T00:00:00Z",

  "identifiers": [
    { "type": "dns", "value": "example.com" },
    { "type": "dns", "value": "www.example.com" }
  ],

  "authorizations": [
    "https://example.com/acme/authz/1234",
    "https://example.com/acme/authz/2345"
  ],

  "finalize": "https://example.com/acme/order/asdf/finalize"
}
```

The order object returned by the server represents a promise that if the client fulfills the server's requirements before the "expires" time, then the server will be willing to finalize the order upon request and issue the requested certificate. In the order object, any authorization referenced in the "authorizations" array whose status is "pending" represents an authorization transaction that the client must complete before the server will issue the certificate (see Section 7.5). If the client fails to complete the required actions before the "expires" time, then the server SHOULD change the status of the order to "invalid" and MAY delete the order resource.

Once the client believes it has fulfilled the server's requirements, it should send a POST request to the order resource's finalize URL. The POST body MUST include a CSR:

csr (required, string): A CSR encoding the parameters for the certificate being requested [RFC2986]. The CSR is sent in the base64url-encoded version of the DER format. (Note: Because this field uses base64url, and does not include headers, it is different from PEM.).

```
POST /acme/order/asdf/finalize HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url({
    "alg": "ES256",
    "kid": "https://example.com/acme/acct/1",
    "nonce": "MSF2j2nawWHPxxkE3ZJtKQ",
    "url": "https://example.com/acme/order/asdf/finalize"
  }),
  "payload": base64url({
    "csr": "MIIBPTCBxAIBADBFMQ...FS6aKdZeGsysoCo4H9P",
  }),
  "signature": "uOrUfIIk5RyQ...nw62Ay1cl6AB"
}
```

The CSR encodes the client's requests with regard to the content of the certificate to be issued. The CSR MUST indicate the exact same set of requested identifiers as the initial new-order request, either in the commonName portion of the requested subject name, or in an extensionRequest attribute [RFC2985] requesting a subjectAltName extension.

A request to finalize an order will result in error if the order indicated does not have status "ready", if the CSR and order identifiers differ, or if the account is not authorized for the identifiers indicated in the CSR.

A valid request to finalize an order will return the order to be finalized. The client should begin polling the order by sending a GET request to the order resource to obtain its current state. The status of the order will indicate what action the client should take:

- "invalid": The certificate will not be issued. Consider this order process abandoned.
- "pending": The server does not believe that the client has fulfilled the requirements. Check the "authorizations" array for entries that are still pending.
- "ready": The server agrees that the requirements have been fulfilled, and is awaiting finalization. Submit a finalization request.
- "processing": The certificate is being issued. Send a GET request after the time given in the "Retry-After" header field of the response, if any.

- "valid": The server has issued the certificate and provisioned its URL to the "certificate" field of the order. Download the certificate.

```
HTTP/1.1 200 OK
Replay-Nonce: CGf81JWBsq8QyIgPCi9Q9X
Location: https://example.com/acme/order/asdf

{
  "status": "valid",
  "expires": "2016-01-01T00:00:00Z",

  "notBefore": "2016-01-01T00:00:00Z",
  "notAfter": "2016-01-08T00:00:00Z",

  "identifiers": [
    { "type": "dns", "value": "example.com" },
    { "type": "dns", "value": "www.example.com" }
  ],

  "authorizations": [
    "https://example.com/acme/authz/1234",
    "https://example.com/acme/authz/2345"
  ],

  "finalize": "https://example.com/acme/order/asdf/finalize",

  "certificate": "https://example.com/acme/cert/asdf"
}
```

7.4.1. Pre-Authorization

The order process described above presumes that authorization objects are created reactively, in response to a certificate order. Some servers may also wish to enable clients to obtain authorization for an identifier proactively, outside of the context of a specific issuance. For example, a client hosting virtual servers for a collection of names might wish to obtain authorization before any virtual servers are created and only create a certificate when a virtual server starts up.

In some cases, a CA running an ACME server might have a completely external, non-ACME process for authorizing a client to issue certificates for an identifier. In these cases, the CA should provision its ACME server with authorization objects corresponding to these authorizations and reflect them as already valid in any orders submitted by the client.

If a CA wishes to allow pre-authorization within ACME, it can offer a "new authorization" resource in its directory by adding the field "newAuthz" with a URL for the new authorization resource.

To request authorization for an identifier, the client sends a POST request to the new-authorization resource specifying the identifier for which authorization is being requested.

identifier (required, object): The identifier to appear in the resulting authorization object (see Section 7.1.4)

```
POST /acme/new-authz HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url({
    "alg": "ES256",
    "kid": "https://example.com/acme/acct/1",
    "nonce": "uQpSjlRb4vQVCjVYAyyUWg",
    "url": "https://example.com/acme/new-authz"
  }),
  "payload": base64url({
    "identifier": {
      "type": "dns",
      "value": "example.net"
    }
  }),
  "signature": "nuSDISbWG8mMgE7H...QyVUL68yzf3Zawps"
}
```

Note that because the identifier in a pre-authorization request is the exact identifier to be included in the authorization object, pre-authorization cannot be used to authorize issuance with wildcard DNS identifiers.

Before processing the authorization request, the server SHOULD determine whether it is willing to issue certificates for the identifier. For example, the server should check that the identifier is of a supported type. Servers might also check names against a blacklist of known high-value identifiers. If the server is unwilling to issue for the identifier, it SHOULD return a 403 (Forbidden) error, with a problem document describing the reason for the rejection.

If the server is willing to proceed, it builds a pending authorization object from the inputs submitted by the client:

- "identifier" the identifier submitted by the client
- "status" MUST be "pending" unless the server has out-of-band information about the client's authorization status
- "challenges" as selected by the server's policy for this identifier

The server allocates a new URL for this authorization, and returns a 201 (Created) response, with the authorization URL in the Location header field, and the JSON authorization object in the body. The client then follows the process described in Section 7.5 to complete the authorization process.

7.4.2. Downloading the Certificate

To download the issued certificate, the client simply sends a GET request to the certificate URL.

The default format of the certificate is application/pem-certificate-chain (see IANA Considerations).

The server MAY provide one or more link relation header fields [RFC5988] with relation "alternate". Each such field SHOULD express an alternative certificate chain starting with the same end-entity certificate. This can be used to express paths to various trust anchors. Clients can fetch these alternates and use their own heuristics to decide which is optimal.

```
GET /acme/cert/asdf HTTP/1.1
Host: example.com
Accept: application/pkix-cert

HTTP/1.1 200 OK
Content-Type: application/pem-certificate-chain
Link: <https://example.com/acme/some-directory>;rel="index"

-----BEGIN CERTIFICATE-----
[End-entity certificate contents]
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
[Issuer certificate contents]
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
[Other certificate contents]
-----END CERTIFICATE-----
```

A certificate resource represents a single, immutable certificate. If the client wishes to obtain a renewed certificate, the client initiates a new order process to request one.

Because certificate resources are immutable once issuance is complete, the server MAY enable the caching of the resource by adding Expires and Cache-Control headers specifying a point in time in the distant future. These headers have no relation to the certificate's period of validity.

The ACME client MAY request other formats by including an Accept header in its request. For example, the client could use the media type "application/pkix-cert" [RFC2585] to request the end-entity certificate in DER format. Server support for alternate formats is OPTIONAL. For formats that can only express a single certificate, the server SHOULD provide one or more "Link: rel="up"" headers pointing to an issuer or issuers so that ACME clients can build a certificate chain as defined in TLS.

7.5. Identifier Authorization

The identifier authorization process establishes the authorization of an account to manage certificates for a given identifier. This process assures the server of two things:

1. That the client controls the private key of the account key pair, and

2. That the client controls the identifier in question.

This process may be repeated to associate multiple identifiers to a key pair (e.g., to request certificates with multiple identifiers), or to associate multiple accounts with an identifier (e.g., to allow multiple entities to manage certificates).

Authorization resources are created by the server in response to certificate orders or authorization requests submitted by an account key holder; their URLs are provided to the client in the responses to these requests. The authorization object is implicitly tied to the account key used to sign the request.

When a client receives an order from the server it downloads the authorization resources by sending GET requests to the indicated URLs. If the client initiates authorization using a request to the new authorization resource, it will have already received the pending authorization object in the response to that request.

```
GET /acme/authz/1234 HTTP/1.1
Host: example.com

HTTP/1.1 200 OK
Content-Type: application/json
Link: <https://example.com/acme/some-directory>;rel="index"

{
  "status": "pending",
  "expires": "2018-03-03T14:09:00Z",

  "identifier": {
    "type": "dns",
    "value": "example.org"
  },

  "challenges": [
    {
      "type": "http-01",
      "url": "https://example.com/acme/authz/1234/0",
      "token": "DGyRejmCefe7v4NfDGDKfA"
    },
    {
      "type": "dns-01",
      "url": "https://example.com/acme/authz/1234/2",
      "token": "DGyRejmCefe7v4NfDGDKfA"
    }
  ],

  "wildcard": false
}
```

7.5.1. Responding to Challenges

To prove control of the identifier and receive authorization, the client needs to respond with information to complete the challenges. To do this, the client updates the authorization object received from the server by filling in any required information in the elements of the "challenges" dictionary.

The client sends these updates back to the server in the form of a JSON object with contents as specified by the challenge type, carried in a POST request to the challenge URL (not authorization URL) once it is ready for the server to attempt validation.

For example, if the client were to respond to the "http-01" challenge in the above authorization, it would send the following request:

```
POST /acme/authz/1234/0 HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url({
    "alg": "ES256",
    "kid": "https://example.com/acme/acct/1",
    "nonce": "Q_s3MWoqT05TrdkM2MTDcw",
    "url": "https://example.com/acme/authz/1234/0"
  }),
  "payload": base64url({}),
  "signature": "9cbg5JO1Gf5YLjjz...SpkUfcdPai9uVYYQ"
}
```

The server updates the authorization document by updating its representation of the challenge with the response object provided by the client. The server MUST ignore any fields in the response object that are not specified as response fields for this type of challenge. The server provides a 200 (OK) response with the updated challenge object as its body.

If the client's response is invalid for any reason or does not provide the server with appropriate information to validate the challenge, then the server MUST return an HTTP error. On receiving such an error, the client SHOULD undo any actions that have been taken to fulfill the challenge, e.g., removing files that have been provisioned to a web server.

The server is said to "finalize" the authorization when it has completed one of the validations, by assigning the authorization a status of "valid" or "invalid", corresponding to whether it considers the account authorized for the identifier. If the final state is "valid", then the server MUST include an "expires" field. When finalizing an authorization, the server MAY remove challenges other than the one that was completed, and may modify the "expires" field. The server SHOULD NOT remove challenges with status "invalid".

Usually, the validation process will take some time, so the client will need to poll the authorization resource to see when it is finalized. For challenges where the client can tell when the server has validated the challenge (e.g., by seeing an HTTP or DNS request from the server), the client SHOULD NOT begin polling until it has seen the validation request from the server.

To check on the status of an authorization, the client sends a GET request to the authorization URL, and the server responds with the current authorization object. In responding to poll requests while

the validation is still in progress, the server MUST return a 200 (OK) response and MAY include a Retry-After header field to suggest a polling interval to the client.

```
GET /acme/authz/1234 HTTP/1.1
Host: example.com

HTTP/1.1 200 OK
Content-Type: application/json

{
  "status": "valid",
  "expires": "2018-09-09T14:09:00Z",

  "identifier": {
    "type": "dns",
    "value": "example.org"
  },

  "challenges": [
    {
      "type": "http-01",
      "url": "https://example.com/acme/authz/1234/0",
      "status": "valid",
      "validated": "2014-12-01T12:05:00Z",
      "token": "IlirfxKKXAsHtmzK29Pj8A"
    }
  ],

  "wildcard": false
}
```

7.5.2. Deactivating an Authorization

If a client wishes to relinquish its authorization to issue certificates for an identifier, then it may request that the server deactivates each authorization associated with it by sending POST requests with the static object {"status": "deactivated"} to each authorization URL.

```
POST /acme/authz/1234 HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url({
    "alg": "ES256",
    "kid": "https://example.com/acme/acct/1",
    "nonce": "xWCM9lGbIyCgue8di6ueWQ",
    "url": "https://example.com/acme/authz/1234"
  }),
  "payload": base64url({
    "status": "deactivated"
  }),
  "signature": "srX9Ji7Le9bjszhu...WTFdtujObzMtZcx4"
}
```

The server MUST verify that the request is signed by the account key corresponding to the account that owns the authorization. If the server accepts the deactivation, it should reply with a 200 (OK) status code and the updated contents of the authorization object.

The server MUST NOT treat deactivated authorization objects as sufficient for issuing certificates.

7.6. Certificate Revocation

To request that a certificate be revoked, the client sends a POST request to the ACME server's revokeCert URL. The body of the POST is a JWS object whose JSON payload contains the certificate to be revoked:

certificate (required, string): The certificate to be revoked, in the base64url-encoded version of the DER format. (Note: Because this field uses base64url, and does not include headers, it is different from PEM.)

reason (optional, int): One of the revocation reasonCodes defined in Section 5.3.1 of [RFC5280] to be used when generating OCSP responses and CRLs. If this field is not set the server SHOULD omit the reasonCode CRL entry extension when generating OCSP responses and CRLs. The server MAY disallow a subset of reasonCodes from being used by the user. If a request contains a disallowed reasonCode the server MUST reject it with the error type "urn:ietf:params:acme:error:badRevocationReason". The problem document detail SHOULD indicate which reasonCodes are allowed.

Revocation requests are different from other ACME requests in that they can be signed either with an account key pair or the key pair in the certificate.

Example using an account key pair for the signature:

```
POST /acme/revoke-cert HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url({
    "alg": "ES256",
    "kid": "https://example.com/acme/acct/1",
    "nonce": "JHb54aT_KTXBWQOzGYkt9A",
    "url": "https://example.com/acme/revoke-cert"
  }),
  "payload": base64url({
    "certificate": "MIIEDTCCAvegAwIBAgIRAP8...",
    "reason": 4
  }),
  "signature": "Q1bURgJoEslbD1c5...3pYdSMLio57mQNN4"
}
```

Example using the certificate key pair for the signature:

```
POST /acme/revoke-cert HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url({
    "alg": "RS256",
    "jwk": /* certificate's public key */,
    "nonce": "JHb54aT_KTXBWQOzGYkt9A",
    "url": "https://example.com/acme/revoke-cert"
  }),
  "payload": base64url({
    "certificate": "MIIEDTCCAvegAwIBAgIRAP8...",
    "reason": 1
  }),
  "signature": "Q1bURgJoEslbD1c5...3pYdSMLio57mQNN4"
}
```

Before revoking a certificate, the server MUST verify that the key used to sign the request is authorized to revoke the certificate. The server MUST consider at least the following accounts authorized for a given certificate:

- the account that issued the certificate.

- an account that holds authorizations for all of the identifiers in the certificate.

The server MUST also consider a revocation request valid if it is signed with the private key corresponding to the public key in the certificate.

If the revocation succeeds, the server responds with status code 200 (OK). If the revocation fails, the server returns an error.

```
HTTP/1.1 200 OK
Replay-Nonce: IXVHDyxIRGcTE0VSblhPzw
Content-Length: 0

--- or ---

HTTP/1.1 403 Forbidden
Replay-Nonce: IXVHDyxIRGcTE0VSblhPzw
Content-Type: application/problem+json
Content-Language: en

{
  "type": "urn:ietf:params:acme:error:unauthorized",
  "detail": "No authorization provided for name example.net"
}
```

8. Identifier Validation Challenges

There are few types of identifiers in the world for which there is a standardized mechanism to prove possession of a given identifier. In all practical cases, CAs rely on a variety of means to test whether an entity applying for a certificate with a given identifier actually controls that identifier.

Challenges provide the server with assurance that an account holder is also the entity that controls an identifier. For each type of challenge, it must be the case that in order for an entity to successfully complete the challenge the entity must both:

- Hold the private key of the account key pair used to respond to the challenge

- Control the identifier in question

Section 10 documents how the challenges defined in this document meet these requirements. New challenges will need to document how they do.

ACME uses an extensible challenge/response framework for identifier validation. The server presents a set of challenges in the authorization object it sends to a client (as objects in the "challenges" array), and the client responds by sending a response object in a POST request to a challenge URL.

This section describes an initial set of challenge types. The definition of a challenge type includes:

1. Content of challenge objects

2. Content of response objects

3. How the server uses the challenge and response to verify control of an identifier

Challenge objects all contain the following basic fields:

type (required, string): The type of challenge encoded in the object.

url (required, string): The URL to which a response can be posted.

status (required, string): The status of this challenge. Possible values are: "pending", "processing", "valid", and "invalid".

validated (optional, string): The time at which the server validated this challenge, encoded in the format specified in RFC 3339 [RFC3339]. This field is REQUIRED if the "status" field is "valid".

error (optional, object): Error that occurred while the server was validating the challenge, if any, structured as a problem document [RFC7807]. Multiple errors can be indicated by using subproblems Section 6.6.1.

All additional fields are specified by the challenge type. If the server sets a challenge's "status" to "invalid", it SHOULD also include the "error" field to help the client diagnose why the challenge failed.

Different challenges allow the server to obtain proof of different aspects of control over an identifier. In some challenges, like HTTP and DNS, the client directly proves its ability to do certain things

related to the identifier. The choice of which challenges to offer to a client under which circumstances is a matter of server policy.

The identifier validation challenges described in this section all relate to validation of domain names. If ACME is extended in the future to support other types of identifiers, there will need to be new challenge types, and they will need to specify which types of identifier they apply to.

8.1. Key Authorizations

All challenges defined in this document make use of a key authorization string. A key authorization is a string that expresses a domain holder's authorization for a specified key to satisfy a specified challenge, by concatenating the token for the challenge with a key fingerprint, separated by a "." character:

```
keyAuthorization = token || '.' || base64url(JWK_Thumbprint(accountKey))
```

The "JWK_Thumbprint" step indicates the computation specified in [RFC7638], using the SHA-256 digest [FIPS180-4]. As noted in [RFC7518] any prepended zero octets in the fields of a JWK object MUST be stripped before doing the computation.

As specified in the individual challenges below, the token for a challenge is a string comprised entirely of characters in the URL-safe base64 alphabet. The "||" operator indicates concatenation of strings.

8.2. Retrying Challenges

ACME challenges typically require the client to set up some network-accessible resource that the server can query in order to validate that the client controls an identifier. In practice it is not uncommon for the server's queries to fail while a resource is being set up, e.g., due to information propagating across a cluster or firewall rules not being in place.

Clients SHOULD NOT respond to challenges until they believe that the server's queries will succeed. If a server's initial validation query fails, the server SHOULD retry the query after some time, in order to account for delay in setting up responses such as DNS records or HTTP resources. The precise retry schedule is up to the server, but server operators should keep in mind the operational scenarios that the schedule is trying to accommodate. Given that retries are intended to address things like propagation delays in HTTP or DNS provisioning, there should not usually be any reason to retry more often than every 5 or 10 seconds. While the server is

still trying, the status of the challenge remains "processing"; it is only marked "invalid" once the server has given up.

The server MUST provide information about its retry state to the client via the "error" field in the challenge and the Retry-After HTTP header field in response to requests to the challenge resource. The server MUST add an entry to the "error" field in the challenge after each failed validation query. The server SHOULD set the Retry-After header field to a time after the server's next validation query, since the status of the challenge will not change until that time.

Clients can explicitly request a retry by re-sending their response to a challenge in a new POST request (with a new nonce, etc.). This allows clients to request a retry when the state has changed (e.g., after firewall rules have been updated). Servers SHOULD retry a request immediately on receiving such a POST request. In order to avoid denial-of-service attacks via client-initiated retries, servers SHOULD rate-limit such requests.

8.3. HTTP Challenge

With HTTP validation, the client in an ACME transaction proves its control over a domain name by proving that it can provision HTTP resources on a server accessible under that domain name. The ACME server challenges the client to provision a file at a specific path, with a specific string as its content.

As a domain may resolve to multiple IPv4 and IPv6 addresses, the server will connect to at least one of the hosts found in the DNS A and AAAA records, at its discretion. Because many web servers allocate a default HTTPS virtual host to a particular low-privilege tenant user in a subtle and non-intuitive manner, the challenge must be completed over HTTP, not HTTPS.

type (required, string): The string "http-01"

token (required, string): A random value that uniquely identifies the challenge. This value MUST have at least 128 bits of entropy. It MUST NOT contain any characters outside the base64url alphabet, and MUST NOT include base64 padding characters ("=").

```
   GET /acme/authz/1234/0 HTTP/1.1
   Host: example.com

   HTTP/1.1 200 OK
   Content-Type: application/json

   {
     "type": "http-01",
     "url": "https://example.com/acme/authz/0",
     "status": "pending",
     "token": "LoqXcYV8q5ONbJQxbmR7SCTNo3tiAXDfowyjxAjEuX0"
   }
```

A client fulfills this challenge by constructing a key authorization from the "token" value provided in the challenge and the client's account key. The client then provisions the key authorization as a resource on the HTTP server for the domain in question.

The path at which the resource is provisioned is comprised of the fixed prefix "/.well-known/acme-challenge/", followed by the "token" value in the challenge. The value of the resource MUST be the ASCII representation of the key authorization.

```
GET /.well-known/acme-challenge/LoqXcYV8q5ONbJQxbmR7SCTNo3tiAXDfowyjxAjEuX0
Host: example.org

HTTP/1.1 200 OK
Content-Type: application/octet-stream

LoqXcYV8q5ONbJQxbmR7SCTNo3tiAXDfowyjxAjEuX0.9jg46WB3rR_AHD-EBXdN7cBkH1WOu0tA3M9f
m21mqTI
```

A client responds with an empty object ({}) to acknowledge that the challenge can be validated by the server.

```
   POST /acme/authz/1234/0
   Host: example.com
   Content-Type: application/jose+json

   {
     "protected": base64url({
       "alg": "ES256",
       "kid": "https://example.com/acme/acct/1",
       "nonce": "JHb54aT_KTXBWQOzGYkt9A",
       "url": "https://example.com/acme/authz/1234/0"
     }),
     "payload": base64url({}),
     "signature": "Q1bURgJoEslbD1c5...3pYdSMLio57mQNN4"
   }
```

On receiving a response, the server constructs and stores the key authorization from the challenge "token" value and the current client account key.

Given a challenge/response pair, the server verifies the client's control of the domain by verifying that the resource was provisioned as expected.

1. Construct a URL by populating the URL template [RFC6570] "http://{domain}/.well-known/acme-challenge/{token}", where:

   * the domain field is set to the domain name being verified; and

   * the token field is set to the token in the challenge.

2. Verify that the resulting URL is well-formed.

3. Dereference the URL using an HTTP GET request. This request MUST be sent to TCP port 80 on the HTTP server.

4. Verify that the body of the response is well-formed key authorization. The server SHOULD ignore whitespace characters at the end of the body.

5. Verify that key authorization provided by the HTTP server matches the key authorization stored by the server.

The server SHOULD follow redirects when dereferencing the URL.

If all of the above verifications succeed, then the validation is successful. If the request fails, or the body does not pass these checks, then it has failed.

8.4. DNS Challenge

When the identifier being validated is a domain name, the client can prove control of that domain by provisioning a TXT resource record containing a designated value for a specific validation domain name.

type (required, string): The string "dns-01"

token (required, string): A random value that uniquely identifies the challenge. This value MUST have at least 128 bits of entropy. It MUST NOT contain any characters outside the base64url alphabet, including padding characters ("=").

```
GET /acme/authz/1234/2 HTTP/1.1
Host: example.com

HTTP/1.1 200 OK
Content-Type: application/json

{
  "type": "dns-01",
  "url": "https://example.com/acme/authz/1234/2",
  "status": "pending",
  "token": "evaGxfADs6pSRb2LAv9IZf17Dt3juxGJ-PCt92wr-oA"
}
```

A client fulfills this challenge by constructing a key authorization from the "token" value provided in the challenge and the client's account key. The client then computes the SHA-256 digest [FIPS180-4] of the key authorization.

The record provisioned to the DNS contains the base64url encoding of this digest. The client constructs the validation domain name by prepending the label "_acme-challenge" to the domain name being validated, then provisions a TXT record with the digest value under that name. For example, if the domain name being validated is "example.org", then the client would provision the following DNS record:

```
_acme-challenge.example.org. 300 IN TXT "gfj9Xq...Rg85nM"
```

A client responds with an empty object ({}) to acknowledge that the challenge can be validated by the server.

```
POST /acme/authz/1234/2
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url({
    "alg": "ES256",
    "kid": "https://example.com/acme/acct/1",
    "nonce": "JHb54aT_KTXBWQOzGYkt9A",
    "url": "https://example.com/acme/authz/1234/2"
  }),
  "payload": base64url({}),
  "signature": "Q1bURgJoEslbD1c5...3pYdSMLio57mQNN4"
}
```

On receiving a response, the server constructs and stores the key authorization from the challenge "token" value and the current client account key.

To validate a DNS challenge, the server performs the following steps:

1. Compute the SHA-256 digest [FIPS180-4] of the stored key authorization

2. Query for TXT records for the validation domain name

3. Verify that the contents of one of the TXT records match the digest value

If all of the above verifications succeed, then the validation is successful. If no DNS record is found, or DNS record and response payload do not pass these checks, then the validation fails.

9. IANA Considerations

9.1. MIME Type: application/pem-certificate-chain

The "Media Types" registry should be updated with the following additional value:

MIME media type name: application

MIME subtype name: pem-certificate-chain

Required parameters: None

Optional parameters: None

Encoding considerations: None

Security considerations: Carries a cryptographic certificate and its associated certificate chain

Interoperability considerations: None

Published specification: draft-ietf-acme-acme [[RFC EDITOR: Please replace draft-ietf-acme-acme above with the RFC number assigned to this]]

Applications which use this media type: Any MIME-compliant transport

Additional information:

File contains one or more certificates encoded with the PEM textual encoding, according to RFC 7468 [RFC7468]. In order to provide easy interoperation with TLS, the first certificate MUST be an end-entity certificate. Each following certificate SHOULD directly certify one preceding it. Because certificate validation requires that trust anchors be distributed independently, a certificate that specifies a trust anchor MAY be omitted from the chain, provided that supported peers are known to possess any omitted certificates.

9.2. Well-Known URI for the HTTP Challenge

The "Well-Known URIs" registry should be updated with the following additional value (using the template from [RFC5785]):

URI suffix: acme-challenge

Change controller: IETF

Specification document(s): This document, Section Section 8.3

Related information: N/A

9.3. Replay-Nonce HTTP Header

The "Message Headers" registry should be updated with the following additional value:

Header Field Name	Protocol	Status	Reference
Replay-Nonce	http	standard	Section 6.4.1

9.4. "url" JWS Header Parameter

The "JSON Web Signature and Encryption Header Parameters" registry should be updated with the following additional value:

- Header Parameter Name: "url"
- Header Parameter Description: URL
- Header Parameter Usage Location(s): JWE, JWS
- Change Controller: IESG
- Specification Document(s): Section 6.3.1 of RFC XXXX

[[RFC EDITOR: Please replace XXXX above with the RFC number assigned to this document]]

9.5. "nonce" JWS Header Parameter

The "JSON Web Signature and Encryption Header Parameters" registry should be updated with the following additional value:

- Header Parameter Name: "nonce"
- Header Parameter Description: Nonce
- Header Parameter Usage Location(s): JWE, JWS
- Change Controller: IESG
- Specification Document(s): Section 6.4.2 of RFC XXXX

[[RFC EDITOR: Please replace XXXX above with the RFC number assigned to this document]]

9.6. URN Sub-namespace for ACME (urn:ietf:params:acme)

The "IETF URN Sub-namespace for Registered Protocol Parameter Identifiers" registry should be updated with the following additional value, following the template in [RFC3553]:

Registry name: acme

Specification: RFC XXXX

Repository: URL-TBD

Index value: No transformation needed.

[[RFC EDITOR: Please replace XXXX above with the RFC number assigned to this document, and replace URL-TBD with the URL assigned by IANA for registries of ACME parameters.]]

9.7. New Registries

This document requests that IANA create the following new registries:

1. ACME Account Object Fields (Section 9.7.1)
2. ACME Order Object Fields (Section 9.7.2)
3. ACME Error Types (Section 9.7.4)

4. ACME Resource Types (Section 9.7.5)

5. ACME Directory Metadata Fields (Section 9.7.6)

6. ACME Identifier Types (Section 9.7.7)

7. ACME Validation Methods (Section 9.7.8)

All of these registries are under a heading of "Automated Certificate Management Environment (ACME) Protocol" and are administered under a Specification Required policy [RFC8126].

9.7.1. Fields in Account Objects

This registry lists field names that are defined for use in ACME account objects. Fields marked as "configurable" may be included in a new-account request.

Template:

- Field name: The string to be used as a field name in the JSON object
- Field type: The type of value to be provided, e.g., string, boolean, array of string
- Client configurable: Boolean indicating whether the server should accept values provided by the client
- Reference: Where this field is defined

Initial contents: The fields and descriptions defined in Section 7.1.2.

Field Name	Field Type	Configurable	Reference
status	string	false	RFC XXXX
contact	array of string	true	RFC XXXX
externalAccountBinding	object	true	RFC XXXX
termsOfServiceAgreed	boolean	true	RFC XXXX
orders	string	false	RFC XXXX

[[RFC EDITOR: Please replace XXXX above with the RFC number assigned to this document]]

9.7.2. Fields in Order Objects

This registry lists field names that are defined for use in ACME order objects. Fields marked as "configurable" may be included in a new-order request.

Template:

- Field name: The string to be used as a field name in the JSON object
- Field type: The type of value to be provided, e.g., string, boolean, array of string
- Client configurable: Boolean indicating whether the server should accept values provided by the client
- Reference: Where this field is defined

Initial contents: The fields and descriptions defined in Section 7.1.3.

Field Name	Field Type	Configurable	Reference
status	string	false	RFC XXXX
expires	string	false	RFC XXXX
identifiers	array of object	true	RFC XXXX
notBefore	string	true	RFC XXXX
notAfter	string	true	RFC XXXX
authorizations	array of string	false	RFC XXXX
finalize	string	false	RFC XXXX
certificate	string	false	RFC XXXX

[[RFC EDITOR: Please replace XXXX above with the RFC number assigned to this document]]

9.7.3. Fields in Authorization Objects

This registry lists field names that are defined for use in ACME authorization objects. Fields marked as "configurable" may be included in a new-authorization request.

Template:

- Field name: The string to be used as a field name in the JSON object
- Field type: The type of value to be provided, e.g., string, boolean, array of string
- Client configurable: Boolean indicating whether the server should accept values provided by the client
- Reference: Where this field is defined

Initial contents: The fields and descriptions defined in Section 7.1.4.

Field Name	Field Type	Configurable	Reference
identifier	object	true	RFC XXXX
status	string	false	RFC XXXX
expires	string	false	RFC XXXX
challenges	array of object	false	RFC XXXX
wildcard	boolean	false	RFC XXXX

[[RFC EDITOR: Please replace XXXX above with the RFC number assigned to this document]]

9.7.4. Error Types

This registry lists values that are used within URN values that are provided in the "type" field of problem documents in ACME.

Template:

- Type: The label to be included in the URN for this error, following "urn:ietf:params:acme:error:"

- Description: A human-readable description of the error

- Reference: Where the error is defined

Initial contents: The types and descriptions in the table in Section 6.6 above, with the Reference field set to point to this specification.

9.7.5. Resource Types

This registry lists the types of resources that ACME servers may list in their directory objects.

Template:

- Field name: The value to be used as a field name in the directory object

- Resource type: The type of resource labeled by the field

- Reference: Where the resource type is defined

Initial contents:

Field Name	Resource Type	Reference
newNonce	New nonce	RFC XXXX
newAccount	New account	RFC XXXX
newOrder	New order	RFC XXXX
newAuthz	New authorization	RFC XXXX
revokeCert	Revoke certificate	RFC XXXX
keyChange	Key change	RFC XXXX
meta	Metadata object	RFC XXXX

[[RFC EDITOR: Please replace XXXX above with the RFC number assigned to this document]]

9.7.6. Fields in the "meta" Object within a Directory Object

This registry lists field names that are defined for use in the JSON object included in the "meta" field of an ACME directory object.

Template:

- Field name: The string to be used as a field name in the JSON object
- Field type: The type of value to be provided, e.g., string, boolean, array of string
- Reference: Where this field is defined

Initial contents: The fields and descriptions defined in Section 7.1.2.

Field Name	Field Type	Reference
termsOfService	string	RFC XXXX
website	string	RFC XXXX
caaIdentities	array of string	RFC XXXX
externalAccountRequired	boolean	RFC XXXX

[[RFC EDITOR: Please replace XXXX above with the RFC number assigned to this document]]

9.7.7. Identifier Types

This registry lists the types of identifiers that can be present in ACME authorization objects.

Template:

- Label: The value to be put in the "type" field of the identifier object
- Reference: Where the identifier type is defined

Initial contents:

Label	Reference
dns	RFC XXXX

[[RFC EDITOR: Please replace XXXX above with the RFC number assigned to this document]]

9.7.8. Validation Methods

This registry lists identifiers for the ways that CAs can validate control of identifiers. Each method's entry must specify whether it corresponds to an ACME challenge type. The "Identifier Type" field must be contained in the Label column of the ACME Identifier Types registry.

Template:

- Label: The identifier for this validation method
- Identifier Type: The type of identifier that this method applies to
- ACME: "Y" if the validation method corresponds to an ACME challenge type; "N" otherwise.
- Reference: Where the validation method is defined

Initial Contents

Label	Identifier Type	ACME	Reference
http-01	dns	Y	RFC XXXX
dns-01	dns	Y	RFC XXXX
tls-sni-01	RESERVED	N	N/A
tls-sni-02	RESERVED	N	N/A

When evaluating a request for an assignment in this registry, the designated expert should ensure that the method being registered has a clear, interoperable definition and does not overlap with existing validation methods. That is, it should not be possible for a client

and server to follow the same set of actions to fulfill two different validation methods.

Validation methods do not have to be compatible with ACME in order to be registered. For example, a CA might wish to register a validation method in order to support its use with the ACME extensions to CAA [I-D.ietf-acme-caa].

[[RFC EDITOR: Please replace XXXX above with the RFC number assigned to this document]]

10. Security Considerations

ACME is a protocol for managing certificates that attest to identifier/key bindings. Thus the foremost security goal of ACME is to ensure the integrity of this process, i.e., to ensure that the bindings attested by certificates are correct and that only authorized entities can manage certificates. ACME identifies clients by their account keys, so this overall goal breaks down into two more precise goals:

1. Only an entity that controls an identifier can get an authorization for that identifier

2. Once authorized, an account key’s authorizations cannot be improperly used by another account

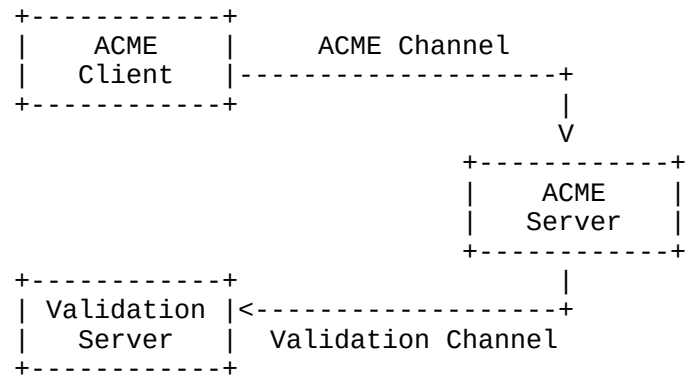
In this section, we discuss the threat model that underlies ACME and the ways that ACME achieves these security goals within that threat model. We also discuss the denial-of-service risks that ACME servers face, and a few other miscellaneous considerations.

10.1. Threat Model

As a service on the Internet, ACME broadly exists within the Internet threat model [RFC3552]. In analyzing ACME, it is useful to think of an ACME server interacting with other Internet hosts along two "channels":

- An ACME channel, over which the ACME HTTPS requests are exchanged

- A validation channel, over which the ACME server performs additional requests to validate a client’s control of an identifier



In practice, the risks to these channels are not entirely separate, but they are different in most cases. Each channel, for example, uses a different communications pattern: the ACME channel will comprise inbound HTTPS connections to the ACME server and the validation channel outbound HTTP or DNS requests.

Broadly speaking, ACME aims to be secure against active and passive attackers on any individual channel. Some vulnerabilities arise (noted below) when an attacker can exploit both the ACME channel and one of the others.

On the ACME channel, in addition to network layer attackers, we also need to account for man-in-the-middle (MitM) attacks at the application layer, and for abusive use of the protocol itself. Protection against application layer MitM addresses potential attackers such as Content Distribution Networks (CDNs) and middleboxes with a TLS MitM function. Preventing abusive use of ACME means ensuring that an attacker with access to the validation channel can't obtain illegitimate authorization by acting as an ACME client (legitimately, in terms of the protocol).

10.2. Integrity of Authorizations

ACME allows anyone to request challenges for an identifier by registering an account key and sending a new-order request using that account key. The integrity of the authorization process thus depends on the identifier validation challenges to ensure that the challenge can only be completed by someone who both (1) holds the private key of the account key pair, and (2) controls the identifier in question.

Validation responses need to be bound to an account key pair in order to avoid situations where an ACME MitM can switch out a legitimate domain holder's account key for one of his choosing, e.g.:

- Legitimate domain holder registers account key pair A
- MitM registers account key pair B
- Legitimate domain holder sends a new-order request signed using account key A
- MitM suppresses the legitimate request but sends the same request signed using account key B
- ACME server issues challenges and MitM forwards them to the legitimate domain holder
- Legitimate domain holder provisions the validation response
- ACME server performs validation query and sees the response provisioned by the legitimate domain holder
- Because the challenges were issued in response to a message signed account key B, the ACME server grants authorization to account key B (the MitM) instead of account key A (the legitimate domain holder)

All of the challenges above have a binding between the account private key and the validation query made by the server, via the key authorization. The key authorization reflects the account public key, is provided to the server in the validation response over the validation channel and signed afterwards by the corresponding private key in the challenge response over the ACME channel.

The association of challenges to identifiers is typically done by requiring the client to perform some action that only someone who effectively controls the identifier can perform. For the challenges in this document, the actions are:

- HTTP: Provision files under .well-known on a web server for the domain
- DNS: Provision DNS resource records for the domain

There are several ways that these assumptions can be violated, both by misconfiguration and by attacks. For example, on a web server that allows non-administrative users to write to .well-known, any user can claim to own the web server’s hostname by responding to an HTTP challenge. Similarly, if a server that can be used for ACME validation is compromised by a malicious actor, then that malicious actor can use that access to obtain certificates via ACME.

The use of hosting providers is a particular risk for ACME validation. If the owner of the domain has outsourced operation of DNS or web services to a hosting provider, there is nothing that can be done against tampering by the hosting provider. As far as the outside world is concerned, the zone or website provided by the hosting provider is the real thing.

More limited forms of delegation can also lead to an unintended party gaining the ability to successfully complete a validation transaction. For example, suppose an ACME server follows HTTP redirects in HTTP validation and a website operator provisions a catch-all redirect rule that redirects requests for unknown resources to a different domain. Then the target of the redirect could use that to get a certificate through HTTP validation since the validation path will not be known to the primary server.

The DNS is a common point of vulnerability for all of these challenges. An entity that can provision false DNS records for a domain can attack the DNS challenge directly and can provision false A/AAAA records to direct the ACME server to send its HTTP validation query to a remote server of the attacker's choosing. There are a few different mitigations that ACME servers can apply:

- Always querying the DNS using a DNSSEC-validating resolver (enhancing security for zones that are DNSSEC-enabled)
- Querying the DNS from multiple vantage points to address local attackers
- Applying mitigations against DNS off-path attackers, e.g., adding entropy to requests [I-D.vixie-dnsext-dns0x20] or only using TCP

Given these considerations, the ACME validation process makes it impossible for any attacker on the ACME channel or a passive attacker on the validation channel to hijack the authorization process to authorize a key of the attacker's choice.

An attacker that can only see the ACME channel would need to convince the validation server to provide a response that would authorize the attacker's account key, but this is prevented by binding the validation response to the account key used to request challenges. A passive attacker on the validation channel can observe the correct validation response and even replay it, but that response can only be used with the account key for which it was generated.

An active attacker on the validation channel can subvert the ACME process, by performing normal ACME transactions and providing a

validation response for his own account key. The risks due to hosting providers noted above are a particular case.

It is RECOMMENDED that the server perform DNS queries and make HTTP connections from various network perspectives, in order to make MitM attacks harder.

10.3. Denial-of-Service Considerations

As a protocol run over HTTPS, standard considerations for TCP-based and HTTP-based DoS mitigation also apply to ACME.

At the application layer, ACME requires the server to perform a few potentially expensive operations. Identifier validation transactions require the ACME server to make outbound connections to potentially attacker-controlled servers, and certificate issuance can require interactions with cryptographic hardware.

In addition, an attacker can also cause the ACME server to send validation requests to a domain of its choosing by submitting authorization requests for the victim domain.

All of these attacks can be mitigated by the application of appropriate rate limits. Issues closer to the front end, like POST body validation, can be addressed using HTTP request limiting. For validation and certificate requests, there are other identifiers on which rate limits can be keyed. For example, the server might limit the rate at which any individual account key can issue certificates or the rate at which validation can be requested within a given subtree of the DNS. And in order to prevent attackers from circumventing these limits simply by minting new accounts, servers would need to limit the rate at which accounts can be registered.

10.4. Server-Side Request Forgery

Server-Side Request Forgery (SSRF) attacks can arise when an attacker can cause a server to perform HTTP requests to an attacker-chosen URL. In the ACME HTTP challenge validation process, the ACME server performs an HTTP GET request to a URL in which the attacker can choose the domain. This request is made before the server has verified that the client controls the domain, so any client can cause a query to any domain.

Some server implementations include information from the validation server’s response (in order to facilitate debugging). Such implementations enable an attacker to extract this information from any web server that is accessible to the ACME server, even if it is not accessible to the ACME client.

It might seem that the risk of SSRF through this channel is limited by the fact that the attacker can only control the domain of the URL, not the path. However, if the attacker first sets the domain to one they control, then they can send the server an HTTP redirect (e.g., a 302 response) which will cause the server to query an arbitrary URL.

In order to further limit the SSRF risk, ACME server operators should ensure that validation queries can only be sent to servers on the public Internet, and not, say, web services within the server operator's internal network. Since the attacker could make requests to these public servers himself, he can't gain anything extra through an SSRF attack on ACME aside from a layer of anonymization.

10.5. CA Policy Considerations

The controls on issuance enabled by ACME are focused on validating that a certificate applicant controls the identifier he claims. Before issuing a certificate, however, there are many other checks that a CA might need to perform, for example:

- Has the client agreed to a subscriber agreement?
- Is the claimed identifier syntactically valid?
- For domain names:
  - If the leftmost label is a '*', then have the appropriate checks been applied?
  - Is the name on the Public Suffix List?
  - Is the name a high-value name?
  - Is the name a known phishing domain?
- Is the key in the CSR sufficiently strong?
- Is the CSR signed with an acceptable algorithm?
- Has issuance been authorized or forbidden by a Certificate Authority Authorization (CAA) record? [RFC6844]

CAs that use ACME to automate issuance will need to ensure that their servers perform all necessary checks before issuing.

CAs using ACME to allow clients to agree to terms of service should keep in mind that ACME clients can automate this agreement, possibly not involving a human user.

11. Operational Considerations

There are certain factors that arise in operational reality that operators of ACME-based CAs will need to keep in mind when configuring their services. For example:

11.1. DNS security

As noted above, DNS forgery attacks against the ACME server can result in the server making incorrect decisions about domain control and thus mis-issuing certificates. Servers SHOULD perform DNS queries over TCP, which provides better resistance to some forgery attacks than DNS over UDP.

An ACME-based CA will often need to make DNS queries, e.g., to validate control of DNS names. Because the security of such validations ultimately depends on the authenticity of DNS data, every possible precaution should be taken to secure DNS queries done by the CA. It is therefore RECOMMENDED that ACME-based CAs make all DNS queries via DNSSEC-validating stub or recursive resolvers. This provides additional protection to domains which choose to make use of DNSSEC.

An ACME-based CA must use only a resolver if it trusts the resolver and every component of the network route by which it is accessed. It is therefore RECOMMENDED that ACME-based CAs operate their own DNSSEC-validating resolvers within their trusted network and use these resolvers both for both CAA record lookups and all record lookups in furtherance of a challenge scheme (A, AAAA, TXT, etc.).

11.2. Token Entropy

The http-01, and dns-01 validation methods mandate the usage of a random token value to uniquely identify the challenge. The value of the token is required to contain at least 128 bits of entropy for the following security properties. First, the ACME client should not be able to influence the ACME server’s choice of token as this may allow an attacker to reuse a domain owner’s previous challenge responses for a new validation request. Secondly, the entropy requirement prevents ACME clients from implementing a "naive" validation server that automatically replies to challenges without participating in the creation of the initial authorization request.

11.3. Malformed Certificate Chains

ACME provides certificate chains in the widely-used format known colloquially as PEM (though it may diverge from the actual Privacy Enhanced Mail specifications [RFC1421], as noted in [RFC7468]). Some

current software will allow the configuration of a private key and a certificate in one PEM file, by concatenating the textual encodings of the two objects. In the context of ACME, such software might be vulnerable to "key replacement" attacks. A malicious ACME server could cause a client to use a private key of its choosing by including the key in the PEM file returned in response to a query for a certificate URL.

When processing an file of type "application/pem-certificate-chain", a client SHOULD verify that the file contains only encoded certificates. If anything other than a certificate is found (i.e., if the string "-----BEGIN" is ever followed by anything other than "CERTIFICATE"), then the client MUST reject the file as invalid.

12. Acknowledgements

In addition to the editors listed on the front page, this document has benefited from contributions from a broad set of contributors, all the way back to its inception.

- Peter Eckersley, EFF
- Eric Rescorla, Mozilla
- Seth Schoen, EFF
- Alex Halderman, University of Michigan
- Martin Thomson, Mozilla
- Jakub Warmuz, University of Oxford
- Sophie Herold, Hemio

This document draws on many concepts established by Eric Rescorla's "Automated Certificate Issuance Protocol" draft. Martin Thomson provided helpful guidance in the use of HTTP.

13. References

13.1. Normative References

[FIPS180-4]
Department of Commerce, National., "NIST FIPS 180-4, Secure Hash Standard", March 2012, <http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2585] Housley, R. and P. Hoffman, "Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP", RFC 2585, DOI 10.17487/RFC2585, May 1999, <https://www.rfc-editor.org/info/rfc2585>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

[RFC2985] Nystrom, M. and B. Kaliski, "PKCS #9: Selected Object Classes and Attribute Types Version 2.0", RFC 2985, DOI 10.17487/RFC2985, November 2000, <https://www.rfc-editor.org/info/rfc2985>.

[RFC2986] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <https://www.rfc-editor.org/info/rfc2986>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

[RFC3492] Costello, A., "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)", RFC 3492, DOI 10.17487/RFC3492, March 2003, <https://www.rfc-editor.org/info/rfc3492>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <https://www.rfc-editor.org/info/rfc5890>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <https://www.rfc-editor.org/info/rfc5988>.

[RFC6068] Duerst, M., Masinter, L., and J. Zawinski, "The 'mailto' URI Scheme", RFC 6068, DOI 10.17487/RFC6068, October 2010, <https://www.rfc-editor.org/info/rfc6068>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/info/rfc6570>.

[RFC6844] Hallam-Baker, P. and R. Stradling, "DNS Certification Authority Authorization (CAA) Resource Record", RFC 6844, DOI 10.17487/RFC6844, January 2013, <https://www.rfc-editor.org/info/rfc6844>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7468] Josefsson, S. and S. Leonard, "Textual Encodings of PKIX, PKCS, and CMS Structures", RFC 7468, DOI 10.17487/RFC7468, April 2015, <https://www.rfc-editor.org/info/rfc7468>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/info/rfc7518>.

[RFC7638] Jones, M. and N. Sakimura, "JSON Web Key (JWK) Thumbprint", RFC 7638, DOI 10.17487/RFC7638, September 2015, <https://www.rfc-editor.org/info/rfc7638>.

[RFC7797] Jones, M., "JSON Web Signature (JWS) Unencoded Payload Option", RFC 7797, DOI 10.17487/RFC7797, February 2016, <https://www.rfc-editor.org/info/rfc7797>.

[RFC7807] Nottingham, M. and E. Wilde, "Problem Details for HTTP APIs", RFC 7807, DOI 10.17487/RFC7807, March 2016, <https://www.rfc-editor.org/info/rfc7807>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

13.2. Informative References

[I-D.ietf-acme-caa]
Landau, H., "CAA Record Extensions for Account URI and ACME Method Binding", draft-ietf-acme-caa-03 (work in progress), August 2017.

[I-D.ietf-acme-ip]
Shoemaker, R., "ACME IP Identifier Validation Extension", draft-ietf-acme-ip-01 (work in progress), September 2017.

[I-D.ietf-acme-telephone]
Peterson, J. and R. Barnes, "ACME Identifiers and Challenges for Telephone Numbers", draft-ietf-acme-telephone-01 (work in progress), October 2017.

[I-D.vixie-dnsext-dns0x20]
Vixie, P. and D. Dagon, "Use of Bit 0x20 in DNS Labels to Improve Transaction Identity", draft-vixie-dnsext-dns0x20-00 (work in progress), March 2008.

[RFC1421] Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures", RFC 1421, DOI 10.17487/RFC1421, February 1993, <https://www.rfc-editor.org/info/rfc1421>.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/info/rfc3552>.

[RFC3553] Mealling, M., Masinter, L., Hardie, T., and G. Klyne, "An IETF URN Sub-namespace for Registered Protocol Parameters", BCP 73, RFC 3553, DOI 10.17487/RFC3553, June 2003, <https://www.rfc-editor.org/info/rfc3553>.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <https://www.rfc-editor.org/info/rfc5785>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <https://www.rfc-editor.org/info/rfc7525>.

[W3C.CR-cors-20130129] Kesteren, A., "Cross-Origin Resource Sharing", World Wide Web Consortium CR CR-cors-20130129, January 2013, <http://www.w3.org/TR/2013/CR-cors-20130129>.

13.3. URIs

[1] https://github.com/ietf-wg-acme/acme

Authors' Addresses

Richard Barnes
Cisco

Email: rlb@ipv.sx

Jacob Hoffman-Andrews
EFF

Email: jsha@eff.org

Daniel McCarney
Let’s Encrypt

Email: cpu@letsencrypt.org

James Kasten
University of Michigan

Email: jdkasten@umich.edu

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 11, 2017

B. Volz
Cisco Systems
Y. Pal
Cisco Systems, Inc.

February 7, 2017

Security of Messages Exchanged Between Servers and Relay Agents
draft-ietf-dhc-relay-server-security-03.txt

Abstract

The Dynamic Host Configuration Protocol for IPv4 (DHCPv4) has no guidance for how to secure messages exchanged between servers and relay agents. The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) states that IPsec should be used to secure messages exchanged between servers and relay agents, but does not require encryption. And, with recent concerns about pervasive monitoring and other attacks, it is appropriate to require securing relay to relay and relay to server communication for DHCPv6 and relay to server communication for DHCPv4.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 11, 2017.

Copyright Notice

Table of Contents

1. Introduction

The Dynamic Host Configuration Protocol for IPv4 (DHCPv4) [RFC2131] and [RFC1542] has no guidance for how to secure messages exchanged between servers and relay agents. The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) [RFC3315] states that IPsec should be used to secure messages exchanged between servers and relay agents, but does not recommend encryption. And, with recent concerns about pervasive monitoring [RFC7258], it is appropriate to require use of IPsec with encryption for relay to server communication for DHCPv4 and require use of IPsec with encryption for relay to relay and relay to server communication for DHCPv6.

2. Requirements Language and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when they appear in ALL CAPS. When these words are not in ALL CAPS (such as "should" or "Should"), they have their usual English meanings, and are not to be interpreted as [RFC2119] key words.

This document uses terminology from [RFC1542], [RFC2131], and [RFC3315].

3. Security of Messages Exchanged Between Servers and Relay Agents

For DHCPv6 [RFC3315], this specification REQUIRES IPsec encryption of relay to relay and relay to server communication and replaces the text in RFC3315 Section 21.1.

For DHCPv4 [RFC2131], this specification REQUIRES IPsec encryption of relay to server communication.

By using IPsec with encryption for this communication, the potentially sensitive client message and relay included information, such as the DHCPv4 relay-agent information option (82) [RFC3046], vendor-specific information (for example, [CableLabs-DHCP]), and Access-Network-Identifier Option(s) [RFC7839], are protected from pervasive monitoring and other attacks.

Relay agents and servers MUST exchange messages securely using the IPsec mechanisms described in [RFC4301]. If a client message is relayed through multiple relay agents, each of the relay agents MUST have an established independent, pairwise trust relationships. That is, if messages from client C will be relayed by relay agent A to relay agent B and then to the server, relay agents A and B MUST be configured to use IPsec for the messages they exchange, and relay agent B and the server MUST be configured to use IPsec for the messages they exchange.

Selectors
: Relay agents are manually configured with the addresses of the relay agent or server to which DHCP messages are to be forwarded. Each relay agent and server that will be using IPsec for securing DHCP messages MUST also be configured with a list of the relay agents to which messages will be returned. The selectors for the relay agents and servers will be the pairs of addresses defining relay agents and servers and the

	direction of DHCP message exchange on DHCPv4 UDP port 67 or DHCPv6 UDP port 547.
Mode	Relay agents and servers MUST use IPsec in transport mode and Encapsulating Security Payload (ESP).
Encryption and authentication algorithms	This document REQUIRES combined mode algorithms for ESP authenticated encryption, ESP encryption algorithms, and ESP authentication algorithms as per Sections 2.1, 2.2, and 2.3 of [RFC7321] respectively. Encryption is required as relay agents may forward unencrypted client messages as well as include additional sensitive information, such as vendor-specific information (for example, [CableLabs-DHCP]) and [RFC7839].
Key management	Because both relay agents and servers tend to be managed by a single organizational entity, public key schemes MAY be optional. Manually configured key management MAY suffice, but does not provide defense against replayed messages. Accordingly, IKEv2 [RFC7296] with preshared secrets SHOULD be supported. IKEv2 with public keys MAY be supported. Additional information on manual vs automated key management and when one should be used over the other can be found in [RFC4107].
Security policy	DHCP messages between relay agents and servers MUST only be accepted from DHCP peers as identified in the local configuration.
Authentication	Shared keys, indexed to the source IP address of the received DHCP message, are adequate in this application.

4. Security Considerations

The security model specified in this document is hop-by-hop. For DHCPv6, there could be multiple relay agents between a client and server and each of these hops needs to be secured. For DHCPv4, there is no support for multiple relays.

As this document only mandates securing messages exchanged between relay agents and servers, the message exchanges between clients and

the first hop relay agent or server are not secured. Clients may follow the recommendations in [RFC7844] to minimize what information they expose or make use of [I-D.ietf-dhc-sedhcpv6] to secure communication between the client and server.

As mentioned in [RFC4552] Section 14, the following are known limitations of the usage of manual keys:

- As the sequence numbers cannot be negotiated, replay protection cannot be provided. This leaves DHCP insecure against all the attacks that can be performed by replaying DHCP packets.

- Manual keys are usually long lived (changing them often is a tedious task). This gives an attacker enough time to discover the keys.

It should be noted if the requirements in this document are followed, while the DHCP traffic on the wire between relays and servers is encrypted, the unencrypted data may still be available through other attacks on the DHCP servers, relays, and related systems. Securing these systems and the data in databases and logs also needs to be considered - on the systems themselves and if transferred over a network (i.e., to network attached storage, for backups, or to operational support systems).

Use of IPsec as described herein is also applicable to Lightweight DHCPv6 Relay Agents [RFC6221], as they have a link-local address which can be used to secure communication with their next hop relay(s).

5. IANA Considerations

This document has no requests of the fantastic IANA team.

6. Acknowledgments

The motivation for this document was several IESG discusses on recent DHCP relay agent options.

Thanks to Kim Kinnear, Jinmei Tatuya, and Tomek Mrugalski for reviewing drafts and helping to improve the document. And, thanks to the authors of [RFC3315] for the original Section 21.1 text.

7. References

7.1. Normative References

[RFC1542] Wimer, W., "Clarifications and Extensions for the Bootstrap Protocol", RFC 1542, DOI 10.17487/RFC1542, October 1993, <http://www.rfc-editor.org/info/rfc1542>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, DOI 10.17487/RFC2131, March 1997, <http://www.rfc-editor.org/info/rfc2131>.

[RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, DOI 10.17487/RFC3315, July 2003, <http://www.rfc-editor.org/info/rfc3315>.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <http://www.rfc-editor.org/info/rfc4301>.

[RFC7321] McGrew, D. and P. Hoffman, "Cryptographic Algorithm Implementation Requirements and Usage Guidance for Encapsulating Security Payload (ESP) and Authentication Header (AH)", RFC 7321, DOI 10.17487/RFC7321, August 2014, <http://www.rfc-editor.org/info/rfc7321>.

7.2. Informative References

[CableLabs-DHCP] "CableLabs' DHCP Options Registry", <http://www.cablelabs.com/specification/cablelabs-dhcp-options-registry-2/>.

[I-D.ietf-dhc-sedhcpv6] Jiang, S., Li, L., Cui, Y., Jinmei, T., Lemon, T., and D. Zhang, "Secure DHCPv6", draft-ietf-dhc-sedhcpv6-20 (work in progress), January 2017.

[RFC3046] Patrick, M., "DHCP Relay Agent Information Option", RFC 3046, DOI 10.17487/RFC3046, January 2001, <http://www.rfc-editor.org/info/rfc3046>.

[RFC4107] Bellovin, S. and R. Housley, "Guidelines for Cryptographic Key Management", BCP 107, RFC 4107, DOI 10.17487/RFC4107, June 2005, <http://www.rfc-editor.org/info/rfc4107>.

[RFC4552] Gupta, M. and N. Melam, "Authentication/Confidentiality for OSPFv3", RFC 4552, DOI 10.17487/RFC4552, June 2006, <http://www.rfc-editor.org/info/rfc4552>.

[RFC6221] Miles, D., Ed., Ooghe, S., Dec, W., Krishnan, S., and A. Kavanagh, "Lightweight DHCPv6 Relay Agent", RFC 6221, DOI 10.17487/RFC6221, May 2011, <http://www.rfc-editor.org/info/rfc6221>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <http://www.rfc-editor.org/info/rfc7258>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <http://www.rfc-editor.org/info/rfc7296>.

[RFC7839] Bhandari, S., Gundavelli, S., Grayson, M., Volz, B., and J. Korhonen, "Access-Network-Identifier Option in DHCP", RFC 7839, DOI 10.17487/RFC7839, June 2016, <http://www.rfc-editor.org/info/rfc7839>.

[RFC7844] Huitema, C., Mrugalski, T., and S. Krishnan, "Anonymity Profiles for DHCP Clients", RFC 7844, DOI 10.17487/RFC7844, May 2016, <http://www.rfc-editor.org/info/rfc7844>.

Authors' Addresses

Bernie Volz
Cisco Systems, Inc.
1414 Massachusetts Ave
Boxborough, MA 01719
USA

Email: volz@cisco.com

Yogendra Pal
Cisco Systems, Inc.
Cessna Business Park,
Varthur Hobli, Outer Ring Road,
Bangalore, Karnataka 560103
India

Email: yogpal@cisco.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 21, 2017

B. Volz
Cisco Systems
Y. Pal
Cisco Systems, Inc.
April 19, 2017

Security of Messages Exchanged Between Servers and Relay Agents
draft-ietf-dhc-relay-server-security-05.txt

Abstract

The Dynamic Host Configuration Protocol for IPv4 (DHCPv4) has no guidance for how to secure messages exchanged between servers and relay agents. The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) states that IPsec should be used to secure messages exchanged between servers and relay agents, but does not require encryption. And, with recent concerns about pervasive monitoring and other attacks, it is appropriate to require securing relay to relay and relay to server communication for DHCPv6 and relay to server communication for DHCPv4.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 21, 2017.

Copyright Notice

Table of Contents

1. Introduction

The Dynamic Host Configuration Protocol for IPv4 (DHCPv4) [RFC2131] and [RFC1542] has no guidance for how to secure messages exchanged between servers and relay agents. The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) [RFC3315] states that IPsec should be used to secure messages exchanged between servers and relay agents, but does not recommend encryption. And, with recent concerns about pervasive monitoring [RFC7258], it is appropriate to require use of IPsec with encryption for relay to server communication for DHCPv4 and require use of IPsec with encryption for relay to relay and relay to server communication for DHCPv6.

This document specifies the optional requirements for relay agent and server implementations to support IPsec authentication and encryption and recommends operators enable this IPsec support.

2. Requirements Language and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when they appear in ALL CAPS. When these words are not in ALL CAPS (such as "should" or "Should"), they have their usual English meanings, and are not to be interpreted as [RFC2119] key words.

This document uses terminology from [RFC1542], [RFC2131], and [RFC3315].

3. Security of Messages Exchanged Between Servers and Relay Agents

For DHCPv6 [RFC3315], this specification REQUIRES relay and server implementations to support IPsec encryption of relay to relay and relay to server communication as documented below (this replaces the text in RFC3315 Section 21.1).

For DHCPv4 [RFC2131], this specification REQUIRES relay and server implementations to support IPsec encryption of relay to server communication as documented below.

This specification RECOMMENDS that operators enable IPsec for this communication.

By using IPsec with encryption for this communication, the potentially sensitive client message and relay included information, such as the DHCPv4 relay-agent information option (82) [RFC3046], vendor-specific information (for example, [CableLabs-DHCP]), and Access-Network-Identifier Option(s) [RFC7839], are protected from pervasive monitoring and other attacks.

Relay agents and servers MUST be able to exchange messages using the IPsec mechanisms described in [RFC4301] and with the conditions below. If a client message is relayed through multiple relay agents (relay chain), each of the relay agents MUST have an established independent, pairwise trust relationships. That is, if messages from client C will be relayed by relay agent A to relay agent B and then to the server, relay agents A and B MUST be configured to use IPsec for the messages they exchange, and relay agent B and the server MUST be configured to use IPsec for the messages they exchange.

Relay agents and servers use IPsec with the following conditions:

Selectors	Relay agents are manually configured with the addresses of the relay agent or server to which DHCP messages are to be forwarded.

	Each relay agent and server that will be using IPsec for securing DHCP messages MUST also be configured with a list of the relay agents to which messages will be returned. The selectors for the relay agents and servers will be the pairs of addresses defining relay agents and servers and the direction of DHCP message exchange on DHCPv4 UDP port 67 or DHCPv6 UDP port 547.
Mode	Relay agents and servers MUST use IPsec in transport mode and Encapsulating Security Payload (ESP).
Encryption and authentication algorithms	This document REQUIRES combined mode algorithms for ESP authenticated encryption, ESP encryption algorithms, and ESP authentication algorithms as per Sections 2.1, 2.2, and 2.3 of [RFC7321] respectively. Encryption is required as relay agents may forward unencrypted client messages as well as include additional sensitive information, such as vendor-specific information (for example, [CableLabs-DHCP]) and [RFC7839].
Key management	Because both relay agents and servers tend to be managed by a single organizational entity, public key schemes MAY be optional. Manually configured key management MAY suffice, but does not provide defense against replayed messages. Accordingly, IKEv2 [RFC7296] with pre-shared secrets SHOULD be supported. IKEv2 with public keys MAY be supported. Additional information on manual vs automated key management and when one should be used over the other can be found in [RFC4107].
Security policy	DHCP messages between relay agents and servers MUST only be accepted from DHCP peers as identified in the local configuration.
Authentication	Shared keys, indexed to the source IP address of the received DHCP message, are adequate in this application.

Note: As using IPsec with multicast has additional complexities (see [RFC5374]), relay agents SHOULD be configured to forward DHCP messages to unicast addresses.

4. Security Considerations

The security model specified in this document is hop-by-hop. For DHCPv6, there could be multiple relay agents between a client and server and each of these hops needs to be secured. For DHCPv4, there is no support for multiple relays.

As this document only mandates securing messages exchanged between relay agents and servers, the message exchanges between clients and the first hop relay agent or server are not secured. Clients may follow the recommendations in [RFC7844] to minimize what information they expose or make use of [I-D.ietf-dhc-sedhcpv6] to secure communication between the client and server.

As mentioned in [RFC4552] Section 14, the following are known limitations of the usage of manual keys:

- As the sequence numbers cannot be negotiated, replay protection cannot be provided. This leaves DHCP insecure against all the attacks that can be performed by replaying DHCP packets.

- Manual keys are usually long lived (changing them often is a tedious task). This gives an attacker enough time to discover the keys.

It should be noted if the requirements in this document are followed, while the DHCP traffic on the wire between relays and servers is encrypted, the unencrypted data may still be available through other attacks on the DHCP servers, relays, and related systems. Securing these systems and the data in databases and logs also needs to be considered - on the systems themselves and if transferred over a network (i.e., to network attached storage, for backups, or to operational support systems).

Use of IPsec as described herein is also applicable to Lightweight DHCPv6 Relay Agents [RFC6221], as they have a link-local address which can be used to secure communication with their next hop relay(s).

5. IANA Considerations

This document has no requests of the fantastic IANA team.

6. Acknowledgments

The motivation for this document was several IESG discusses on recent DHCP relay agent options.

Thanks to Kim Kinnear, Jinmei Tatuya, Francis Dupont, and Tomek Mrugalski for reviewing drafts and helping to improve the document. And, thanks to the authors of [RFC3315] for the original Section 21.1 text.

7. References

7.1. Normative References

[RFC1542] Wimer, W., "Clarifications and Extensions for the Bootstrap Protocol", RFC 1542, DOI 10.17487/RFC1542, October 1993, <http://www.rfc-editor.org/info/rfc1542>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, DOI 10.17487/RFC2131, March 1997, <http://www.rfc-editor.org/info/rfc2131>.

[RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, DOI 10.17487/RFC3315, July 2003, <http://www.rfc-editor.org/info/rfc3315>.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <http://www.rfc-editor.org/info/rfc4301>.

[RFC7321] McGrew, D. and P. Hoffman, "Cryptographic Algorithm Implementation Requirements and Usage Guidance for Encapsulating Security Payload (ESP) and Authentication Header (AH)", RFC 7321, DOI 10.17487/RFC7321, August 2014, <http://www.rfc-editor.org/info/rfc7321>.

7.2. Informative References

[CableLabs-DHCP] "CableLabs' DHCP Options Registry", <http://www.cablelabs.com/specification/cablelabs-dhcp-options-registry-2/>.

[I-D.ietf-dhc-sedhcpv6]
Li, L., Jiang, S., Cui, Y., Jinmei, T., Lemon, T., and D. Zhang, "Secure DHCPv6", draft-ietf-dhc-sedhcpv6-21 (work in progress), February 2017.

[RFC3046] Patrick, M., "DHCP Relay Agent Information Option", RFC 3046, DOI 10.17487/RFC3046, January 2001, <http://www.rfc-editor.org/info/rfc3046>.

[RFC4107] Bellovin, S. and R. Housley, "Guidelines for Cryptographic Key Management", BCP 107, RFC 4107, DOI 10.17487/RFC4107, June 2005, <http://www.rfc-editor.org/info/rfc4107>.

[RFC4552] Gupta, M. and N. Melam, "Authentication/Confidentiality for OSPFv3", RFC 4552, DOI 10.17487/RFC4552, June 2006, <http://www.rfc-editor.org/info/rfc4552>.

[RFC5374] Weis, B., Gross, G., and D. Ignjatic, "Multicast Extensions to the Security Architecture for the Internet Protocol", RFC 5374, DOI 10.17487/RFC5374, November 2008, <http://www.rfc-editor.org/info/rfc5374>.

[RFC6221] Miles, D., Ed., Ooghe, S., Dec, W., Krishnan, S., and A. Kavanagh, "Lightweight DHCPv6 Relay Agent", RFC 6221, DOI 10.17487/RFC6221, May 2011, <http://www.rfc-editor.org/info/rfc6221>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <http://www.rfc-editor.org/info/rfc7258>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <http://www.rfc-editor.org/info/rfc7296>.

[RFC7839] Bhandari, S., Gundavelli, S., Grayson, M., Volz, B., and J. Korhonen, "Access-Network-Identifier Option in DHCP", RFC 7839, DOI 10.17487/RFC7839, June 2016, <http://www.rfc-editor.org/info/rfc7839>.

[RFC7844] Huitema, C., Mrugalski, T., and S. Krishnan, "Anonymity Profiles for DHCP Clients", RFC 7844, DOI 10.17487/RFC7844, May 2016, <http://www.rfc-editor.org/info/rfc7844>.

Authors' Addresses

Bernie Volz
Cisco Systems, Inc.
1414 Massachusetts Ave
Boxborough, MA 01719
USA

Email: volz@cisco.com

Yogendra Pal
Cisco Systems, Inc.
Cessna Business Park,
Varthur Hobli, Outer Ring Road,
Bangalore, Karnataka 560103
India

Email: yogpal@cisco.com

I2RS working group
Internet-Draft
Intended status: Informational
Expires: April 2, 2017

S. Hares
Huawei
D. Migault
J. Halpern
Ericsson

September 29, 2016

I2RS Security Related Requirements
draft-ietf-i2rs-protocol-security-requirements-17

Abstract

This presents security-related requirements for the I2RS protocol which provides a new interface to the routing system described in the I2RS architecture document (RFC7921). The I2RS protocol is a re-use protocol implemented by re-using portions of existing IETF protocols and adding new features to these protocols. The I2RS protocol re-uses security features of a secure transport (E.g. TLS, SSH, DTLS) such as encryption, message integrity, mutual peer authentication, and replay protection. The new I2RS features to consider from a security perspective are: a priority mechanism to handle multi-headed write transactions, an opaque secondary identifier which identifies an application using the I2RS client, and an extremely constrained read-only non-secure transport. This document provides the detailed requirements for these security features.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 2, 2017.

Table of Contents

1. Introduction

The Interface to the Routing System (I2RS) provides read and write access to information and state within the routing system. An I2RS client interacts with one or more I2RS agents to collect information from network routing systems. [RFC7921] describes the architecture of this interface, and this documents assumes the reader is familiar with this architecture and its definitions. Section 2 highlights some of the references the reader is required to be familiar with.

The I2RS interface is instantiated by the I2RS protocol connecting an I2RS client and an I2RS agent associated with a routing system. The I2RS protocol is a re-use protocol implemented by re-using portions of existing IETF protocols, and adding new features to these protocols. As a re-use protocol, it can be considered a higher-level protocol since it can be instantiated in multiple management protocols (e.g. NETCONF [RFC6241] or RESTCONF [I-D.ietf-netconf-restconf]) operating over a secure transport. The security for the I2RS protocol comes from the management protocols operating over a a secure transport.

This document is part of the requirements for I2RS protocol which also include:

- I2RS architecture [RFC7921],
- I2RS ephemeral state requirements [I-D.ietf-i2rs-ephemeral-state],
- publication/subscription requirements [RFC7922], and
- traceability [RFC7923].

Since the I2RS "higher-level" protocol changes the interface to the routing systems, it is important that implementers understand the new security requirements for the environment the I2RS protocol operates in. These security requirements for the I2RS environment are specified in [I-D.ietf-i2rs-security-environment-reqs]; and the summary of the I2RS protocol security environment is found in the I2RS Architecture [RFC7920].

I2RS reuses the secure transport protocols (TLS, SSH, DTLS) which support encryption, message integrity, peer authentication, and key distribution protocols. Optionally, implementers may utilize AAA protocols (Radius over TLS or Diameter over TLS) to securely distribute identity information.

Section 3 provides an overview of security features and protocols being re-used (section 3.1) and the new security features being

required (section 3.2). Section 3 also explores how existing and new security features and protocols would be paired with existing IETF management protocols (section 3.3).

The new features I2RS extends to these protocols are a priority mechanism to handle multi-headed writes, an opaque secondary identifier to allow traceability of an application utilizing a specific I2RS client to communicate with an I2RS agent, and insecure transport constrained to be utilized only for read-only data, which may include publically available data (e.g. public BGP Events, public telemetry information, web service availability) and some legacy data.

Section 4 provides the I2RS protocol security requirements by the following security features:

- peer identity authentication (section 4.1),
- peer identity validation before role-based message actions (section 4.2)
- peer identity and client redundancy (section 4.3),
- multi-channel transport requirements: Secure transport and insecure Transport (section 4.4),
- management protocol security requirements (section 4.5),
- role-based security (section 4.6),
- security environment (section 4.7)

Protocols designed to be I2RS higher-layer protocols need to fulfill these security requirements.

2. Definitions

2.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2.2. Security Definitions

This document utilizes the definitions found in the following documents: [RFC4949] and [RFC7921]

Specifically, this document utilizes the following definitions from [RFC4949]:

- access control,
- authentication,
- data confidentiality,
- data integrity,
- data privacy,
- identity,
- identifier,
- mutual authentication,
- role,
- role-based access control,
- security audit trail, and
- trust.

[RFC7922] describes traceability for I2RS interface and the I2RS protocol. Traceability is not equivalent to a security audit trail or simple logging of information. A security audit trail may utilize traceability information.

This document also requires that the user is familiar with the pervasive security requirements in [RFC7258].

2.3. I2RS Specific Definitions

The document utilizes the following concepts from the I2RS architecture: [RFC7921]:

- I2RS client, I2RS agent, and I2RS protocol (section 2),
- I2RS higher-layer protocol (section 7.2)
- scope: read scope, notification scope, and write scope (section 2),
- identity and scope of the identity (section 2),

- roles or security rules (section 2),
- identity and scope, and secondary identity (section 2),
- routing system/subsytem (section 2),
- I2RS assumed security environment (section 4),
- I2RS identity and authorization (section 4.1),
- I2RS authorization, scope of Authorization in I2RS client and agent (section 4.2),
- client redundancy with a single client identity (section 4.3),
- restrictions on I2RS in personal devices (section 4.4),
- communication channels and I2RS high-layer protocol (section 7.2),
- active communication versus connectivity (section 7.5),
- multi-headed control (section 7.8), and
- transaction, message, multi-message atomicity (section 7.9).

This document assumes the reader is familar with these terms.

This document discusses the security of the multiple I2RS communication channels which operate over the higher-layer I2RS protocol. The higher-layer I2RS protocol combines a secure transport and I2RS contextual information, and re-uses IETF protocols and data models to create the secure transport and the I2RS data-model driven contextual information. To describe how the I2RS high-layer protocol combines other protocols into the I2RS higher-layer protocol, the following terms are used:

I2RS component protocols

: Protocols which are re-used and combined to create the I2RS protocol.

I2RS secure-transport component protocols

: The I2RS secure transport protocols that support the I2RS higher-layer protocol.

I2RS management component protocols

The I2RS management protocol which provide the management information context.

I2RS AAA component protocols

The I2RS AAA protocols supporting the I2RS higher-layer protocol.

The I2RS higher-layer protocol requires implementation of a I2RS secure-transport component protocol and the I2RS management component protocol. The I2RS AAA component protocol is optional.

3. Security Features and Protocols: Re-used and New

3.1. Security Protocols Re-Used by the I2RS Protocol

I2RS requires a secure transport protocol and key distribution protocols. The secure transport features required by I2RS are peer authentication, confidentiality, data integrity, and replay protection for I2RS messages. According to [I-D.ietf-taps-transports], the secure transport protocols which support peer authentication, confidentiality, data integrity, and replay protection are the following:

1. TLS [RFC5246] over TCP or SCTP,

2. DTLS over UDP with replay detection and anti-DoS stateless cookie mechanism required for the I2RS protocol, and the I2RS protocol allow DTLS options of record size negotiation and and conveyance of "don't" fragment bits to be optional in deployments.

3. HTTP over TLS (over TCP or SCTP), and

4. HTTP over DTLS (with the requirements and optional features specified above in item 2).

The following protocols would need to be extended to provide confidentiality, data integrity, peer authentication, and key distribution protocols: IPFIX (over SCTP, TCP or UDP) and ForCES TML layer (over SCTP). These protocols will need extensions to run over a secure transport (TLS or DTLS) (see section 3.3 for details).

The specific type of key management protocols an I2RS secure transport uses depends on the transport. Key management protocols utilized for the I2RS protocols SHOULD support automatic rotation.

An I2RS implementer may use AAA protocols over secure transport to distribute the identities for I2RS client and I2RS agent and role authorization information. Two AAA protocols are: Diameter [RFC6733]

and Radius [RFC2865]. To provide the best security I2RS peer identities, the AAA protocols MUST be run over a secure transport (Diameter over secure transport (TLS over TCP) [RFC6733]), Radius over a secure transport (TLS) [RFC6614]).

3.2. New Features Related to Security

The new features are priority, an opaque secondary identifier, and an insecure protocol for read-only data constrained to specific standard usages. The I2RS protocol allows multi-headed control by several I2RS clients. This multi-headed control is based on the assumption that the operator deploying the I2RS clients, I2RS agents, and the I2rs protocol will coordinate the read, write, and notification scope so the I2RS clients will not contend for the same write scope. However, just in case there is an unforseen overlap of I2RS clients attempting to write a particular piece of data, the I2RS architecture [RFC7921] provides the concept of each I2RS client having a priority. The I2RS client with the highest priority will have its write succeed. This document specifies requirements for this new concept of priority.

The opaque secondary identifier identifies an application which is using the I2RS client to I2RS agent communication to manage the routing system. The secondary identifier is opaque to the I2RS protocol. In order to protect personal privacy, the secondary identifier should not contain personal identifiable information.

The last new feature related to I2RS security is the ability to allow non-confidential data to be transferred over a non-secure transport. It is expected that most I2RS data models will describe information that will be transferred with confidentiality. Therefore, any model which transfers data over a non-secure transport is marked. The use of a non-secure transport is optional, and an implementer SHOULD create knobs that allow data marked as non-confidential to be sent over a secure transport.

Non-confidential data can only be read or notification scope transmission of events. Non-confidential data cannot be write scope or notification scope configuration. An example of non-confidential data is the telemetry information that is publically known (e.g. BGP route-views data or web site status data) or some legacy data (e.g. interface) which cannot be transported in secure transport. The IETF I2RS Data models MUST indicate in the data model the specific data which is non-confidential.

Most I2RS data models will expect that the information described in the model will be transferred with confidentiality.

3.3. I2RS Protocol Security Requirements vs. IETF Management Protocols

Table 1 below provides a partial list of the candidate management protocols and the secure transports each one of the support. One column in the table indicates the transport protocol will need I2RS security extensions.

Mangement Protocol	Transport Protocol	I2RS Extensions
NETCONF	TLS over TCP (*1)	None required (*2)
RESTCONF	HTTP over TLS with X.509v3 certificates, certificate validation, mutual authentication: 1) authenticated server identity, 2) authenticated client identity (*1)	None required (*2)
FORCES	TML over SCTP (*1)	Needs extension to TML to run TML over TLS over SCTP, or DTLS with options for replay protection and anti-DoS stateless cookie mechanism. (DTLS record size negotiation and conveyance of "don't" fragment bits are optional). The IPSEC mechanism is not sufficient for I2RS traveling over multiple hops (router + link) (*2)
IPFIX	SCTP, TCP, UDP TLS or DTLS for secure client (*1)	Needs to extension to support TLS or DTLS with options for replay protection and anti-DoS stateless cookie mechanism. (DTLS record size negotiation and conveyance of "don't" fragment

bits are optional).

*1 - Key management protocols
MUST support appropriate key rotation.

*2 - Identity and Role authorization distributed
by Diameter or Radius MUST use Diameter over TLS
or Radius over TLS.

4. Security-Related Requirements

This section discusses security requirements based on the following security functions:

- peer identity authentication (section 4.1),
- Peer Identity validation before Role-based Message Actions (section 4.2)
- peer identity and client redundancy (section 4.3),
- multi-channel transport requirements: Secure transport and insecure Transport (section 4.4),
- management protocol security requirements (section 4.5),
- role-based security (section 4.6),
- security environment (section 4.7)

The I2RS Protocol depends upon a secure transport layer for peer authentication, data integrity, confidentiality, and replay protection. The optional insecure transport can only be used restricted set of publically data available (events or information) or a select set of legacy data. Data passed over the insecure transport channel MUST NOT contain any data which identifies a person.

4.1. I2RS Peers(agent and client) Identity Authentication

The following requirements specify the security requirements for Peer Identity Authentication for the I2RS protocol:

- SEC-REQ-01: All I2RS clients and I2RS agents MUST have an identity, and at least one unique identifier that uniquely identifies each party in the I2RS protocol context.

- SEC-REQ-02: The I2RS protocol MUST utilize these identifiers for mutual identification of the I2RS client and I2RS agent.

- SEC-REQ-03: Identifier distribution and the loading of these identifiers into I2RS agent and I2RS client SHOULD occur outside the I2RS protocol prior to the I2RS protocol establishing a connection between I2RS client and I2RS agent. AAA protocols MAY be used to distribute these identifiers, but other mechanism can be used.

Explanation:

These requirements specify the requirements for I2RS peer (I2RS agent and I2RS client) authentication. A secure transport (E.g. TLS) will authenticate based on these identities, but these identities are identities for the I2RS management layer. An AAA protocol distributing I2RS identity information SHOULD transport its information over a secure transport.

4.2. Identity Validation Before Role-Based Message Actions

The requirements for I2RS clients with Secure Connections are the following:

SEC-REQ-04: An I2RS agent receiving a request from an I2RS client MUST confirm that the I2RS client has a valid identity.

SEC-REQ-05: An I2RS client receiving an I2RS message over a secure transport MUST confirm that the I2RS agent has a valid identifier.

SEC-REQ-06: An I2RS agent receiving an I2RS message over an insecure transport MUST confirm that the content is suitable for transfer over such a transport.

Explanation:

Each I2RS client has a scope based on its identity and the security roles (read, write, or events) associated with that identity, and that scope must be considered in processing an I2RS messages sent on a communication channel. An I2RS communication channel may utilize multiple transport sessions, or establish a transport session and then close the transport session. Therefore, it is important that the I2RS peers are operating utilizing valid peer identities when a message is processed rather than checking if a transport session exists.

During the time period when a secure transport session is active, the I2RS agent SHOULD assume that the I2RS client’s identity remains

valid. Similarly, while a secure connection exists that included validating the I2RS agent's identity and a message is received via that connection, the I2RS client SHOULD assume that the I2RS agent's identity remains valid.

The definition of what constitutes a valid identity or a valid identifier MUST be defined by the I2RS protocol.

4.3. Peer Identity, Priority, and Client Redundancy

Requirements:

SEC-REQ-07: Each I2RS Identifier MUST be associated with just one priority.

SEC-REQ-08: Each Identifier is associated with one secondary identifier during a particular I2RS transaction (e.g. read/write sequence), but the secondary identifier may vary during the time a connection between the I2RS client and I2RS agent is active.

Explanation:

The I2RS architecture also allows multiple I2RS clients with unique identities to connect to an I2RS agent (section 7.8). The I2RS deployment using multiple clients SHOULD coordinate this multi-headed control of I2RS agents by I2RS clients so no conflict occurs in the write scope. However, in the case of conflict on a write scope variable, the error resolution mechanisms defined by the I2RS architecture multi-headed control ([RFC7921], section 7.8) allow the I2RS agent to deterministically choose one I2RS client. The I2RS client with highest priority is given permission to write the variable, and the second client receives an error message.

A single I2RS client may be associated with multiple applications with different tasks (e.g. weekly configurations or emergency configurations). The secondary identity is an opaque value that the I2RS client passes to the I2RS agent so that this opaque value can be placed in the tracing file or event stream to identify the application using the I2RS client to I2RS agent communication. The I2RS client is trusted to simply assert the secondary identifier.

One example of the use of the secondary identity is the situation where an operator of a network has two applications that use an I2RS client. The first application is a weekly configuration application that uses the I2RS protocol to change configurations. The second application is an application that allows operators to makes emergency changes to routers in the network. Both of these applications use the same I2RS client to write to an I2RS agent. In

order for traceability to determine which application (weekly configuration or emergency) wrote some configuration changes to a router, the I2RS client sends a different opaque value for each of the applications. The weekly configuration secondary opaque value could be "xzzy-splot" and the emergency secondary opaque value could be "splish-splash".

A second example is if the I2RS client is used for monitoring of critical infrastructure. The operator of a network using the I2RS client may desire I2RS client redundancy where the monitoring application wth the I2RS client is deployed on two different boxes with the same I2RS client identity (see [RFC7921] section 4.3) These two monitoring applications pass to the I2RS client whether the application is the primary or back up application, and the I2RS client passes this information in the I2RS secondary identitifier as the figure below shows. The primary applications secondary identifier is "primary-monitoring", and the backup application secondary identifier is "backup-monitoring". The I2RS tracing information will include the secondary identifier information along with the transport information in the tracing file in the agent.

Example 2: Primary and Backup Application for Monitoring Identification sent to agent

Application A--I2RS client--Secure transport(#1)
[I2RS identity 1, secondary identifier: "primary-monitoring"]-->

Application B--I2RS client--Secure transport(#2)
[I2RS identity 1, secondary identifier: "backup-monitoring"]-->

Figure 1

4.4. Multi-Channel Transport: Secure Transport and Insecure Transport

Requirements:

SEC-REQ-09: The I2RS protocol MUST be able to transfer data over a secure transport and optionally MAY be able to transfer data over a non-secure transport. The default transport is a secure transport, and this secure transport is mandatory to implement (MTI) in all I2RS agents, and in any I2RS client which: a) performs a Write scope transaction which is sent to the I2RS agent or b): configures an Event Scope transaction. This secure transport is mandatory to use (MTU) on any I2RS client’s Write transaction or the configuration of an Event Scope transaction.

SEC-REQ-10: The secure transport MUST provide data confidentiality, data integrity, and practical replay prevention.

SEC-REQ-11: The I2RS client and I2RS agent protocol SHOULD implement mechanisms that mitigate DoS attacks. For the secure transport, this means the secure transport must support DoS prevention. For the insecure transport protocol, the I2RS higher-layer protocol MUST contain a transport management layer that considers the detection of DoS attacks and provides a warning over a secure-transport channel.

SEC-REQ-12: A secure transport MUST be associated with a key management solution that can guarantee that only the entities having sufficient privileges can get the keys to encrypt/decrypt the sensitive data.

SEC-REQ-13: A machine-readable mechanism to indicate that a data-model contains non-confidential data MUST be provided. A non-secure transport MAY be used to publish only read scope or notification scope data if the associated data model indicates that that data is non-confidential.

SEC-REQ-14: The I2RS protocol MUST be able to support multiple secure transport sessions providing protocol and data communication between an I2RS agent and an I2RS client. However, a single I2RS agent to I2RS client connection MAY elect to use a single secure transport session or a single non-secure transport session conforming the requirements above.

SEC-REQ-15: Deployment configuration knobs SHOULD be created to allow operators to send "non-confidential" Read scope (data or Event streams) over a secure transport.

SEC-REQ-16: The I2RS protocol makes use of both secure and insecure transports, but this use MUST NOT be done in any way that weakens the secure transport protocol used in the I2RS protocol or other contexts that do not have this requirement for mixing secure and insecure modes of operation.

Explanation:

The I2RS architecture defines three scopes: read, write, and notification scope. Insecure data can only be used for read scope and notification scope of "non-confidential data". The configuration of ephemeral data in the I2RS agent uses either write scope for data or write scope for configuration of event notification streams. The requirement to use secure transport for configuration prevents

accidental or malevolent entities from altering the I2RS routing system through the I2RS agent.

It is anticipated that the passing of most I2RS ephemeral state operational status SHOULD be done over a secure transport.

In most circumstances the secure transport protocol will be associated with a key management system. Most deployments of the I2RS protocol will allow for automatic key management systems. Since the data models for the I2RS protocol will control key routing functions, it is important that deployments of I2RS use automatic key management systems.

Per BCP107 [RFC4107] while key management system SHOULD be automatic, the systems MAY be manual in the following scenarios:

a) The environment has limited bandwidth or high round-trip times.

b) The information being protected has low value.

c) The total volume of traffic over the entire lifetime of the long-term session key will be very low.

d) The scale of the deployment is limited.

Operators deploying the I2RS protocol selecting manual key management SHOULD consider both short and medium term plans. Deploying automatic systems initially may save effort over the long-term.

4.5. Management Protocol Security

Requirements:

SEC-REQ-17: In a critical infrastructure, certain data within routing elements is sensitive and read/write operations on such data SHOULD be controlled in order to protect its confidentiality. To achieve this, higher-layer protocols MUST utilize a secure transport, and SHOULD provide access control functions to protect confidentiality of the data.

SEC-REQ-18: An integrity protection mechanism for I2RS MUST be provided that will be able to ensure the following:

1) the data being protected is not modified without detection during its transportation,

2) the data is actually from where it is expected to come from, and

3) the data is not repeated from some earlier interaction the higher layer protocol (best effort).

The I2RS higher-layer protocol operating over a secure transport provides this integrity. The I2RS higher-layer protocol operating over an insecure transport SHOULD provide some way for the client receiving non-confidential read-scoped or event-scoped data over the insecure connection to detect when the data integrity is questionable; and in the event of a questionable data integrity the I2RS client should disconnect the insecure transport connection.

SEC-REQ-19: The I2RS higher-layer protocol MUST provide a mechanism for message traceability (requirements in [RFC7922]) that supports the tracking higher-layer functions run across secure connection or a non-secure transport.

Explanation:

Most carriers do not want a router’s configuration and data flow statistics known by hackers or their competitors. While carriers may share peering information, most carriers do not share configuration and traffic statistics. To achieve this, the I2RS higher-layer protocol (e.g NETCONF) requires access control (NACM [RFC6536]) for sensitive data needs to be provided; and the confidentiality protection on such data during transportation needs to be enforced.

Integrity of data is important even if the I2RS protocol is sending non-confidential data over an insecure connection. The ability to trace I2RS protocol messages that enact I2RS transactions provides a minimal aid to helping operators check how messages enact transactions on a secure or insecure transport. Contextual checks on specific non-confidential data sent over a insecure connection may indicate the data has been modified.

4.6. Role-Based Data Model Security

The I2RS Architecture [RFC7921] specifies access control by "role" where role is a method of making access control more manageable by creating a grouping of users so that access control can be specified for a role rather than for each of the individuals. Therefore, I2RS role specifies the access control for a group as being read, write, or notification.

SEC-REQ-20: The rules around what I2RS security role is permitted to access and manipulate what information over a secure transport (which protects the data in transit) SHOULD ensure that data of any level of sensitivity is reasonably protected from being

observed by those without permission to view it, so that privacy requirements are met.

SEC-REQ-21: Role security MUST work when multiple transport connections are being used between the I2RS client and I2RS agent as the I2RS architecture [RFC7921] describes.

Sec-REQ-22: If an I2RS agents or an I2RS client is tightly correlated with a person, then the I2RS protocol and data models SHOULD provide additional security that protects the person's privacy.

Explanation:

I2RS higher-layer uses management protocol E.g. NETCONF, RESTCONF) to pass messages in order to enact I2RS transactions. Role Security must secure data (sensitivity and normal data) in a router even when it is operating over multiple connections at the same time. NETCONF can run over TLS (over TCP or SCTP) or SSH. RESTCONF runs over HTTP over a secure transport (TLS). SCTP [RFC4960] provides security for multiple streams plus end-to-end transport of data. Some I2RS functions may wish to operate over DTLS which runs over UDP ([RFC6347]), DDCP ([RFC6238]), and SCTP ([RFC5764]).

Please note the security of the application to I2RS client connection is outside of the I2RS protocol or I2RS interface.

While I2RS clients are expected to be related to network devices and not individual people, if an I2RS client ran on a person's phone, then privacy protection to anonymize any data relating to a person's identity or location would be needed.

A variety of forms of managemen may set policy on roles: "operator-applied knobs", roles that restrict personal access, data-models with specific "privacy roles", and access filters.

4.7. Security of the environment

The security for the implementation of a protocol also considers the protocol environment. The environmental security requirements are found in: [I-D.ietf-i2rs-security-environment-reqs].

5. Security Considerations

This is a document about security requirements for the I2RS protocol and data modules. Security considerations for the I2RS protocol include both the protocol and the security environment.

6. IANA Considerations

This draft is requirements, and does not request anything of IANA.

7. Acknowledgement

The authors would like to thank Wes George, Ahmed Abro, Qin Wu, Eric Yu, Joel Halpern, Scott Brim, Nancy Cam-Winget, DaCheng Zhang, Alia Atlas, and Jeff Haas for their contributions to the I2RS security requirements discussion and this document. The authors would like to thank Bob Moskowitz, Kathleen Moriarty, Stephen Farrell, Radia Perlman, Alvaro Retana, Ben Campbell, and Alissa Cooper for their review of these requirements.

8. References

8.1. Normative References

[I-D.ietf-i2rs-security-environment-reqs] Migault, D., Halpern, J., and S. Hares, "I2RS Environment Security Requirements", draft-ietf-i2rs-security-environment-reqs-01 (work in progress), April 2016.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC4107] Bellovin, S. and R. Housley, "Guidelines for Cryptographic Key Management", BCP 107, RFC 4107, DOI 10.17487/RFC4107, June 2005, <http://www.rfc-editor.org/info/rfc4107>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <http://www.rfc-editor.org/info/rfc4949>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <http://www.rfc-editor.org/info/rfc7258>.

[RFC7921] Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", RFC 7921, DOI 10.17487/RFC7921, June 2016, <http://www.rfc-editor.org/info/rfc7921>.

[RFC7922] Clarke, J., Salgueiro, G., and C. Pignataro, "Interface to the Routing System (I2RS) Traceability: Framework and Information Model", RFC 7922, DOI 10.17487/RFC7922, June 2016, <http://www.rfc-editor.org/info/rfc7922>.

[RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, DOI 10.17487/RFC7923, June 2016, <http://www.rfc-editor.org/info/rfc7923>.

8.2. Informative References

[I-D.ietf-i2rs-ephemeral-state]
Haas, J. and S. Hares, "I2RS Ephemeral State Requirements", draft-ietf-i2rs-ephemeral-state-18 (work in progress), September 2016.

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-17 (work in progress), September 2016.

[I-D.ietf-taps-transports]
Fairhurst, G., Trammell, B., and M. Kuehlewind, "Services provided by IETF transport protocols and congestion control mechanisms", draft-ietf-taps-transports-11 (work in progress), July 2016.

[RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, DOI 10.17487/RFC2865, June 2000, <http://www.rfc-editor.org/info/rfc2865>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <http://www.rfc-editor.org/info/rfc4960>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, DOI 10.17487/RFC5764, May 2010, <http://www.rfc-editor.org/info/rfc5764>.

[RFC6238] M'Raihi, D., Machani, S., Pei, M., and J. Rydell, "TOTP: Time-Based One-Time Password Algorithm", RFC 6238, DOI 10.17487/RFC6238, May 2011, <http://www.rfc-editor.org/info/rfc6238>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <http://www.rfc-editor.org/info/rfc6536>.

[RFC6614] Winter, S., McCauley, M., Venaas, S., and K. Wierenga, "Transport Layer Security (TLS) Encryption for RADIUS", RFC 6614, DOI 10.17487/RFC6614, May 2012, <http://www.rfc-editor.org/info/rfc6614>.

[RFC6733] Fajardo, V., Ed., Arkko, J., Loughney, J., and G. Zorn, Ed., "Diameter Base Protocol", RFC 6733, DOI 10.17487/RFC6733, October 2012, <http://www.rfc-editor.org/info/rfc6733>.

[RFC7920] Atlas, A., Ed., Nadeau, T., Ed., and D. Ward, "Problem Statement for the Interface to the Routing System", RFC 7920, DOI 10.17487/RFC7920, June 2016, <http://www.rfc-editor.org/info/rfc7920>.

Authors' Addresses

Susan Hares
Huawei
7453 Hickory Hill
Saline, MI 48176
USA

Email: shares@ndzh.com

Daniel Migault
Ericsson
8400 boulevard Decarie
Montreal, QC HAP 2N2
Canada

Email: daniel.migault@ericsson.com

Joel Halpern
Ericsson
US

Email: joel.halpern@ericsson.com

I2RS WG
Internet-Draft
Intended status: Informational
Expires: September 29, 2017

D. Migault
J. Halpern
Ericsson
S. Hares
Huawei

March 28, 2017

I2RS Environment Security Requirements
draft-ietf-i2rs-security-environment-reqs-05

Abstract

This document provides environment security requirements for the I2RS architecture. Environment security requirements are independent of the protocol used for I2RS. The security environment requirements are the good security practices to be used during implementation and deployment of the code related to the new interface to routing system (I2RS) so that I2RS implementations can be securely deployed and operated.

Environmental security requirements do not specify the I2RS protocol seecurity requirements. This is done in another document (draft-ietf-i2rs-protocol-security-requirements).

Table of Contents

1. Introduction

This document provides environment security requirements for the I2RS architecture. Environment security requirements are independent of the protocol used for I2RS. The I2RS protocol security requirements [I-D.ietf-i2rs-protocol-security-requirements] define the security for the communication between I2RS client and agent. The security environment requirements are good security practices to be used during implementation and deployment of the I2RS protocol so that I2RS protocol implementations can be securely deployed and operated. These environment security requirements address the security considerations described in the I2RS Architecture [RFC7921] required to provide a stable and secure environment in which the dynamic programmatic interface to the routing system (I2RS) should operates.

Even though the I2RS protocol is mostly concerned with the interface between the I2RS client and the I2RS agent, the environmental security requirements must consider the entire I2RS architecture and specify where security functions may be hosted and what criteria should be met in order to address any new attack vectors exposed by deploying this architecture. Environment security for I2RS has to be considered for the complete I2RS architecture and not only on the protocol interface.

This document is structured as follows:

- Section 2 describes the terminology used in this document,
- Section 3 describes how the I2RS plane can be securely isolated from the management plane, control plane, and forwarding plane.

The subsequent sections of the document focus on the security within the I2RS plane.

- Section 4 analyses how the I2RS access control policies can be deployed throughout the I2RS plane in order to limit access to the routing system resources to authorized components with the authorized privileges. This analysis examines how providing a robust communication system between the components aids the access control.
- Section 5 details how I2RS keeps applications isolated from another and without affecting the I2RS components. Applications may be independent, with different scopes, owned by different tenants. In addition, the applications may modify the routing system in an automatic way.

Motivations are described before the requirements are given.

The reader is expected to be familiar with the I2RS problem statement [RFC7920], I2RS architecture, [RFC7921], traceability requirements [RFC7922], I2RS Pub/Sub requirements [RFC7923], I2RS ephemeral state requirements [I-D.ietf-i2rs-ephemeral-state], I2RS protocol security requirements [I-D.ietf-i2rs-protocol-security-requirements].

2. Terminology and Acronyms

- Environment Security Requirements : Security requirements specifying how the environment a protocol operates in needs to be secured. These requirements do not specify the protocol security requirements.

- I2RS plane: The environment the I2RS process is running on. It includes the applications, the I2RS client, and the I2RS agent.

- I2RS user: The user of the I2RS client software or system.

- I2RS access control policies: The policies controlling access of the routing resources by applications. These policies are divided into policies applied by the I2RS client regarding applications and policies applied by the I2RS agent regarding I2RS clients.

- I2RS client access control policies: The access control policies processed by the I2RS client.

- I2RS agent access control policies: The access control policies processed by the I2RS agent.

2.1. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. I2RS Plane Isolation

Isolating the I2RS plane from other network planes (the management, forwarding, and control planes) is fundamental to the security of the I2RS environment. Clearly differentiating the I2RS components from the rest of the network device does the following:

1. protects the I2RS components from vulnerabilities in other parts of the network,

2. protects other systems vital to the health of the network from vulnerabilities in the I2RS plane.

Separating the I2RS plane from other network control and forwarding planes is similar to the best common practice of placing software into software containers within modules with clear interfaces to exterior modules. In a similar way, although the I2RS plane cannot be completely isolated from other planes, it can be carefully designed so the interactions between the I2RS plane and other planes can be identified and controlled. The following is a brief description of how the I2RS plane positions itself in regard to the other planes.

3.1. I2RS Plane and Management plane

The purpose of the I2RS plane is to provide a standard programmatic interface to the routing system resources to network oriented applications. Routing protocols often run in a control plane and provide entries for the forwarding plane as shown in figure 1. The I2RS plane contains the I2RS applications, the I2RS client, the north bound interface between the I2RS client and I2RS applications, the I2RS protocol, the I2RS agent, and the south bound API (SB API) to the routing system. The communication interfaces in the I2RS plane are shown on the the left hand side of figure 1.

The management plane contains the mangement application, the management client, the north bound API between the management client and management application, the mangement server, the management protocol (E.g. RESTCONF) between mangement client and management server, and the south bound API between the management server and the control plane. The communication interfaces associated with the management plane are shown on the right hand side of figure 2.

The I2RS plane and the management plane both interact with the control plane on which the routing systems operate. [RFC7921] describes several of these interaction points such as the local configuration, the static system state, routing, and signaling. A routing resource may be accessed by I2RS plane, the mangement plane, or routing protocol(s) which creates the potential for overlapping access. The southbound APIs can limit the scope of the management plane’s and the I2RS plane’s interaction with the routing resources.

Security focus:

Data can be read by I2RS plane from configuration as copy of configuration data, or by management plane as copies of the I2RS plane. The problem is when the I2RS plane installs the routing plane as its new configuration or the management plane installs the I2RS plane information as management plane configuration. In this circumstance, we define "infecting" as interfering with and leading into a incoherent state. Planned interactions such as interactions

denoted in in two cooperating Yang data modules is not an incoherent state.

The primary protection in this space is going to need to be validation rules on:

- the data being sent/received by the I2RS agent (including notification of changes that the I2RS agent sends the I2RS client),
- any data transferred between management datastores (configuration or operational state) and I2RS ephemeral control plane data stores;
- data transferred between I2RS Agent and Routing system,
- data transferred between a management server and the I2RS routing system,
- data transferred between I2RS agent and system (e.g. interfaces ephemeral configuration),
- data transferred between management server and the system (e.g. interface configuration).

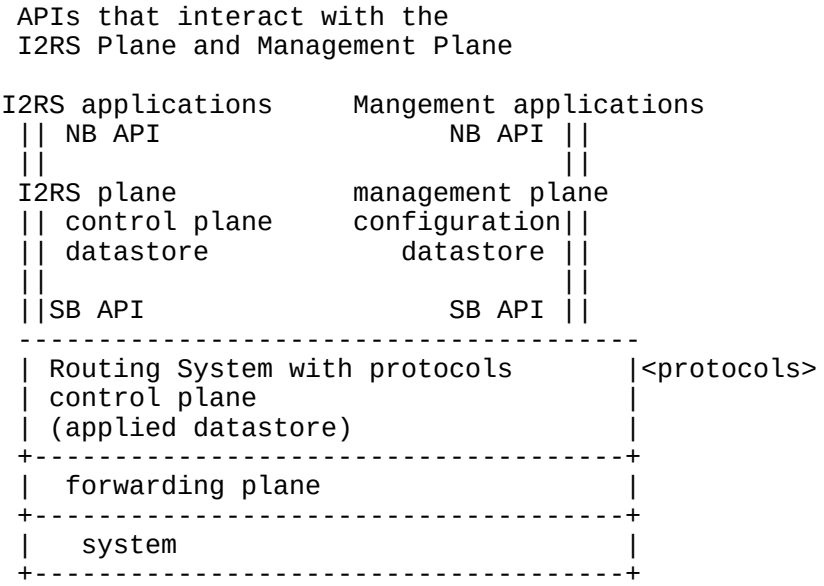


Figure 1 - North Bound (NB) APIs and South Bound (SB) APIs

3.2. I2RS Plane and Forwarding Plane

Applications hosted by the I2RS client belong to the I2RS plane. It is difficult to constrain these applications to the I2RS plane, or even to limit their scope within the I2RS plane. Applications using I2RS may also interact with components outside the I2RS plane. For example an application may use a management client to configure the network and monitored events via an I2RS agent as figure 4 shows.

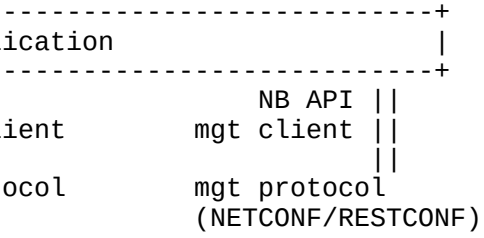


figure 2

Applications may also communicate with multiple I2RS clients in order to have a broader view of the current and potential states of the network and the I2RS plane itself. These varied remote communication

relationships between applications using the I2RS protocol to change the forwarding plane make it possible for an individual application to be an effective attack vector against the operation of the network, a router's I2RS plane, the forwarding plane of the routing system, and other planes (management and control planes).

Prevention measures:

Systems should consider the following prevention errors:

application validation - There is little the I2RS plane can do to validate applications with which it interacts. The I2RS client passes the I2RS agent an unique identifier for the application so that an application's actions can be traced back to the application.

Validation against common misconfigurations or errors - One way of securing the interfaces between application, the I2RS plane, and the forwarding plane is to limit the information accepted and to limit the rate information is accepted between these three software planes. Another method is to perform rudimentary checks on the results of any updates to the forwarding plane.

3.3. I2RS Plane and Control Plane

The network control plane consists of the processes and protocols that discover topology, advertise reachability, and determine the shortest path between any location on the network and any destination. I2RS client configures, monitors or receives events via the I2RS agent's interaction with the routing system including the process that handles the control plane signalling protocols (BGP, ISIS, OSPF, etc.), route information databases (RIBs), and interface databases. In some situations, to manage an network outage or to control traffic, the I2RS protocol may modify information in the route database or the configuration of routing process. While this is not a part of normal processing, such action allows the network operator to bypass temporary outages or DoS attacks.

This capability to modify the routing process information carries with it the risk that the I2RS agent may alter the normal properties of the routing protocols which provide normal loop free routing in the control plane. For example, information configured by the I2RS agent into routing process or RIBs could cause forwarding problems or routing loops. As a second example, state which is inserted or deleted from routing processes (control traffic, counters, etc.) could cause the routing protocols to fail to converge or loop).

Prevention measures:

The I2RS implementation can provide internal checks after a routing system protocol change that it is still operating correctly. These checks would be specific to the routing protocol the I2RS Agent would change. For example, if a BGP maximum prefix limit for a BGP peer is lowered then the BGP peer should not allow the number prefixes received from that peer to exceed this number.

3.4. Requirements

To isolate I2RS transactions from other planes, it is required that:

SEC-ENV-REQ 1: Application-to-routing system resources communications should use an isolated communication channel. Various levels of isolation can be considered. The highest level of isolation may be provided by using a physically isolated network. Alternatives may also consider logical isolation (e.g. using vLAN). In a virtual environment that shares a common infrastructure, encryption may also be used as a way to enforce isolation. Encryption can be added by using a secure transport required by the I2RS protocol security [I-D.ietf-i2rs-protocol-security-requirements], and sending the non-confidential I2RS data (designed for a non-secure transport) over a secure transport.

SEC-ENV-REQ 2: The interface used by the routing element to receive I2RS transactions via the I2RS protocol (e.g. the IP address) SHOULD be a dedicated physical or logical interface. As previously mentioned, a dedicated physical interface may contribute to a higher isolation. Isolation may also be achieved by using a dedicated IP address or a dedicated port.

SEC-ENV-REQ 3: An I2RS agent SHOULD have specific permissions for interaction with each routing element and access to the routing element should governed by policy specific to the I2RS agent’s interfaces (network, routing system, system, or cross-datastore).

Explanation:

When the I2RS agent performs an action on a routing element, the action is performed in a process (or processes) associated with a routing process. For example, in a typical UNIX system, the user is designated with a user id (uid) and belongs to groups designated by group ids (gid). If such a user id (uid) and group id (gid) is the identifier for the routing processes peforming routing tasks in the

control plane, then the I2RS Agent process would communicate with these routing processes. It is important that the I2RS agent has its own unique identifier and the routing processes have their own identifier so that access control can uniquely filter data between I2RS Agent and routing processes.

The specific policy that the I2RS agent uses to filter data from the network or from different processes on a system (routing, system or cross-datastore) should be specific to the I2RS agent. For example, the network access filter policy that the I2RS agent uses should be uniquely identifiable from the configuration datastore updated by a management protocol.

SEC-ENV-REQ 4: The I2RS plane should be informed when a routing system resource is modified by a user outside the I2RS plane access. Notifications from the control plane SHOULD not be allowed to flood the I2RS plane, and rate limiting (or summarization) is expected to be applied. These routing system notifications MAY translated to the appropriate I2RS agent notifications, and passed to various I2RS clients via notification relays.

Explanation:

This requirements is also described in section 7.6 of [RFC7921] for the I2RS client, and this section extends it to the entire I2RS plane (I2RS agent, client, and application).

A routing system resource may be accessed by management plane or control plane protocols so a change to a routing system resource may remain unnoticed unless and until the routing system resource notifies the I2RS plane by notifying the I2RS agent. Such notification is expected to trigger synchronization of the I2RS resource state between the I2RS agent and I2RS client - signalled by the I2RS agent sending a notification to an I2RS client.

The updated resource should be available in the operational state if there is a yang module referencing that operational state, but this is not always the case. In the cases where operational state is not updated, the I2RS SB (southbound) API should include the ability to send a notification.

SEC-ENV-REQ 5: I2RS plane should define an "I2RS plane overwrite policy". Such policy defines how an I2RS is able to update and overwrite a resource set by a user outside the I2RS plane. Such hierarchy has been described in section 6.3 and 7.8 of [RFC7921]

Explanation:

A key part of the I2RS architecture is notification regarding routing system changes across the I2RS plane (I2RS client to/from I2RS agent). The security environment requirements above (SEC-ENV-REQ-03 to SEC-ENV-REQ-05) provide the assurance that the I2RS plane and the routing systems the I2RS plane attaches to remains untouched by the other planes or the I2RS plane is notificed of such changes. Section 6.3 of [RFC7921] describes how the I2RS agent within the I2RS plane interacts with forwarding plane's local configuration, and provides the example of an overwrite policy between the I2RS plane and local configuration (instantiated in 2 Policy Knobs) that operators may wish to set. The prompt notification of any outside overwrite is key to the architecture and proper interworking of the I2RS Plane.

4. I2RS Access Control for Routing System Resources

This section provides recommendations on how the I2RS plane's access control should be designed to protect the routing system resources. These security policies for access control only apply within the I2RS plane. More especially, the policies are associated to the applications, I2RS clients and I2RS agents, with their associated identity and roles.

The I2RS deployment of I2RS applications, I2RS clients, and I2RS agents can be located locally in a closed environment or distributed over open networks. The normal case for routing system management is over an open environment. Even in a closed environment, access control policies should be carefully defined to be able to, in the future, carefully extend the I2RS plane to remote applications or remote I2RS clients.

[I-D.ietf-i2rs-protocol-security-requirements] defines the security requirements of the I2RS protocol between the I2RS client and the I2RS agent over a secure transport. This section focuses on I2RS access control architecture (section 4.1), access control policies of the I2RS agent (section 4.2), the I2RS client (section 4.3), and the application (section 4.4).

4.1. I2RS Access Control Architecture

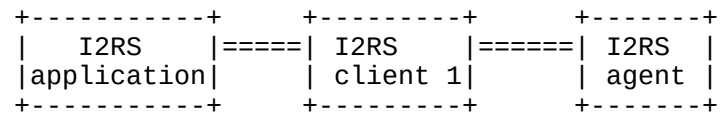
Overview:

Applications access the routing system resource via numerous intermediate nodes. The application communicates with an I2RS client. In some cases, the I2RS client is only associated with a single application attached to one or more agents (case a and case b

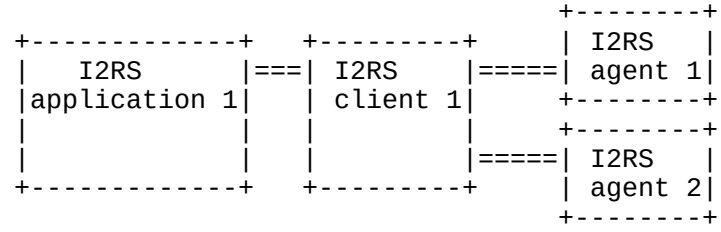
in figure 4 below). In other cases, the I2RS client may be connected to two applications (case c in figure 4 below), or the I2RS may act as a broker (agent/client device shown in case d in figure 4 below). The I2RS client broker approach provides scalability to the I2RS architecture as it avoids each application being registered to the I2RS agent. Similarly, the I2RS access control should be able to scale to numerous applications.

The goal of the security environment requirements in this section are to control the interactions between the applications and the I2RS client, and the interactions between the I2RS client and the I2RS agent. The key challenge is that the I2RS architecture puts the I2RS Client in control of the stream of communication (application to I2RS client and I2RS client to the I2RS agent). The I2RS agent must trust the I2RS client’s actions without having an ability to verify the I2RS client’s actions.

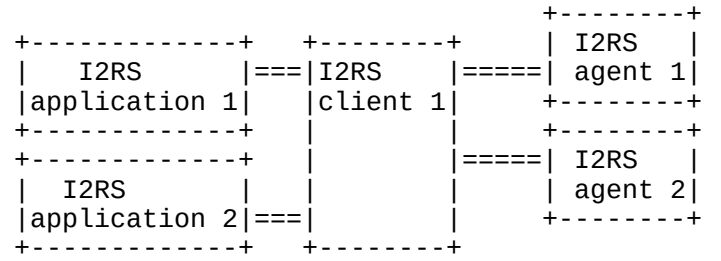
a) I2RS application-client pair talking
   to one I2RS agent



b) I2RS application client pair talking to
   two i2RS agents



c) two applications talk to 1 client



d) I2RS Broker (agent/client

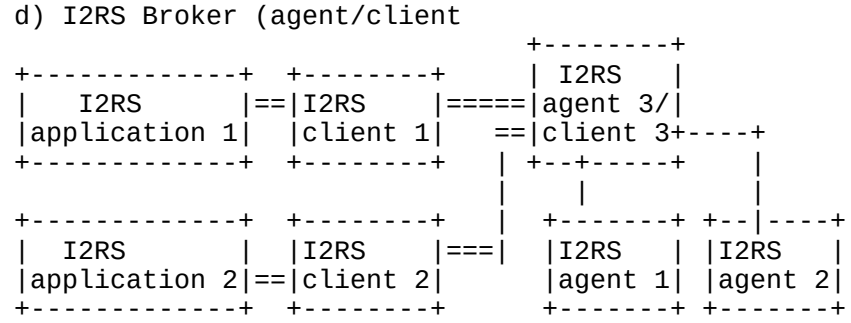


figure 3

4.1.1. Access control Enforcement Scope

SEC-ENV-REQ 6: I2RS access control should be performed through the whole I2RS plane. It should not be enforced by the I2RS agent only within the routing element. Instead, the I2RS client should enforce the I2RS client access control against applications and the I2RS agent should enforce the I2RS agent access control against the I2RS clients. The mechanisms for the I2RS client access control are not in scope of the I2RS architecture [RFC7921], which exclusively focuses on the I2RS agent access control provided by the I2RS protocol.

Explanation:

This architecture results in a layered and hierarchical or multi-party I2RS access control. An application will be able to access a routing system resource only if both the I2RS client is granted access by the I2RS agent and the application is granted access by the I2RS client.

4.1.2. Notification Requirements

SEC-ENV-REQ 7: When an access request to a routing resource is refused by one party (the I2RS client or the I2RS agent), the requester (e.g the application) as well as all intermediaries should indicate the reason the access has not been granted, and which entity rejected the request.

Explanation:

In case the I2RS client access control or the I2RS agent access control does not grant access to a routing system resource, the application should be able to determine whether its request has been rejected by the I2RS client or the I2RS agent as well as the reason that caused the reject.

SEC-REQ-07 indicates the I2RS agent may reject the request because the I2RS client is not an authorized I2RS client or lacks the privileges to perform the requested transaction (read, write, start notifications or logging). The I2RS client should be notified of the reason the I2RS agent rejected the transaction due to a lack of authorization or privileges, and the I2RS client should return a message to the application indicating the I2RS agent reject the transaction with an indication of this reason. Similarly, if the I2RS client does not grant the access to the application, the I2RS

client should also inform the application. An example of an error message could be, "Read failure: you do not have read permission", "Write failure: you do not have write permission", or "Write failure: resource accessed by someone else".

This requirement has been written in a generic manner as it relates to the following interactions:

- interactions between the application and the I2RS client,
- interactions between the I2RS client and the I2RS agent at a content level (Protocol security requirements are described by [I-D.ietf-i2rs-protocol-security-requirements]), and
- interactions between the I2RS agent and the I2RS routing system (forwarding plane, control plane, and routing plane).

4.1.3. Trust

SEC-ENV-REQ 8: In order to provide coherent access control policies enforced by multiple parties (e.g. the I2RS client or the I2RS agent), theses parties should trust each other, and communication between them should also be trusted (e.g. TLS) in order to reduce additional vectors of attacks.

SEC-ENV-REQ 9: I2RS client or I2RS agent SHOULD also be able to refuse a communication with an application or an I2RS client when the communication channel does not fulfill enough security requirements.

Explanation:

The participants in the I2RS Plane (I2RS client, I2RS agent, and I2RS application) exchange critical information, and to be effective the communication should be trusted and free from security attacks.

The I2RS client or the I2RS agent should be able to reject messages over a communication channel that can be easily hijacked, like a clear text UDP channel.

4.1.4. Sharing access control Information

For the I2RS client:

SEC-ENV-REQ 10: The I2RS client MAY request information on its I2RS access control subset policies from the I2RS agent or cache requests that have been rejected by the I2RS

agent to limit forwarding unnecessary queries to the I2RS agent.

SEC-ENV-REQ 11: The I2RS client MAY support receiving notifications when its I2RS access control subset policies have been updated by the I2RS agent.

Similarly, for the applications:

SEC-ENV-REQ 12: The applications MAY request information on its I2RS access control subset policies in order to limit forwarding unnecessary queries to the I2RS client.

SEC-ENV-REQ 13: The applications MAY subscribe to a service that provides notification when its I2RS access control subset policies have been updated.

For both the application and the client:

SEC-ENV-REQ 14: The I2RS access control should explicitly specify accesses that are granted. More specifically, anything not explicitly granted should be denied (default rule).

Explanation:

In order to limit the number of access requests that result in an error, each application or I2RS client can retrieve the I2RS access control policies that apply to it. This subset of rules is designated as the "Individual I2RS access control policies". As these policies are subject to changes, a dynamic synchronization mechanism should be provided. However, such mechanisms may be implemented with different levels of completeness and dynamicity of the individual I2RS access control policies. One example may be caching transaction requests that have been rejected.

I2RS access control should be appropriately balanced between the I2RS client and the I2RS agent. It remains relatively easy to avoid the complete disclosure of the access control policies of the I2RS agent. Relative disclosure of access control policies may allow leaking confidential information in case of misconfiguration. It is important to balance the level of trust of the I2RS client and the necessity of distributing the enforcement of the access control policies.

I2RS access control should not solely rely only on the I2RS client or the I2RS agent as illustrated below:

- 1) I2RS clients are dedicated to a single application: In this case, it is likely that I2RS access control is enforced only by the I2RS agent, as the I2RS client is likely to accept all access requests of the application. It is recommended that even in this case, I2RS client access control is not based on an "Allow anything from application" policy, but instead the I2RS client specifies accesses that are enabled. In addition, the I2RS client may sync its associated I2RS access control policies with the I2RS agent to limit the number of refused access requests being sent to the I2RS agent. The I2RS client is expected to balance benefits and problems with synchronizing its access control policies with the I2RS agent to proxy request validation versus simply passing the access request to the I2RS agent.

- 2) A single I2RS client connects to multiple applications or acts as a broker for many applications:
In this case the I2RS agent has a single I2RS client attached, so the I2RS client could be configured to enforce access control policies instead of the I2RS Agent. In this circumstance, it is possible that the I2RS agent may grant an I2RS client high priviledges and blindly trust the I2RS client without enforcing access control policies on what the I2RS client can do. Such a situation must be avoided as it could be used by malicious applications for a privilege escalation by compromising the I2RS client, causing the I2RS client to perform some action on behalf of the application that it normally does not have the privileges to perform. In order to mitigate such attacks, the I2RS client that connects to multiple applications or operates as a broker is expected to host application with an equivalent level of privileges.

4.1.5. Sharing Access Control in Groups of I2RS Clients and Agents

Overview:

To distribute the I2RS access control policies between I2RS clients and I2RS agents, I2RS access control policies can also be distributed within a set of I2RS clients or a set of I2RS agents.

Requirements:

SEC-ENV-REQ 15: I2RS clients should be distributed and act as brokers for applications that share roughly similar permissions.

SEC-ENV-REQ 16: I2RS agents should avoid granting extra privileges to their authorized I2RS client. I2RS agents should be shared by I2RS clients with roughly similar permissions. More explicitly, an I2RS agent shared between I2RS clients that are only provided read access to the routing system resources do not need to perform any write access, so the I2RS client should not be provided these accesses.

SEC-ENV-REQ 17: I2RS clients and I2RS agents should be able to trace [RFC7922] the various transactions they perform as well as suspicious activities. These logs should be collected regularly and analysed by functions that may be out of the I2RS plane.

Explanation:

This restriction for distributed I2RS clients to act as brokers only for applications with roughly the same privileges avoids the I2RS client having extra privileges compared to hosted applications, and discourages applications from performing privilege escalation within an I2RS client. For example, suppose an I2RS client requires write access to the resources. It is not recommended to grant the I2RS agent the write access in order to satisfy a unique I2RS client. Instead, the I2RS client that requires write access should be connected to an I2RS agent that is already shared by an I2RS client that requires write access.

Access control policies enforcement should be monitored in order to detect violation of the policies or detect an attack. Access control policies enforcement may not be performed by the I2RS client or the I2RS agent as violation may require a more global view of the I2RS access control policies. As a result, consistency check and mitigation may instead be performed by the management plane. However, I2RS clients and I2RS agents play a central role.

The I2RS agent can trace transactions that an I2RS client requests it to perform, and to link this to the application via the secondary opaque identifier to the application. This information is placed in a tracing log which is retrieved by management processes. If a particular application is granted a level of privileges it should not have, then this tracing mechanism may detect this security intrusion after the intrusion has occurred.

4.1.6. Managing Access Control Policy

Access control policies should be implemented so that the policies remain manageable in short and longer term deployments of the I2RS protocol and the I2RS plane.

Requirements:

SEC-ENV-REQ 18: access control should be managed in an automated way, that is granting or revoking an application should not involve manual configuration over the I2RS plane (I2RS client, I2RS agent, and application).

Explanation:

Granting or configuring an application with new policy should not require manual configuration of I2RS clients, I2RS agents, or other applications.

SEC-ENV-REQ 19: Access control should be scalable when the number of application grows as well as when the number of I2RS clients increases.

Explanation:

A typical implementation of a local I2RS client access control policies may result in creating manually a system user associated with each application. Such an approach is not likely to scale when the number of applications increases into the hundreds.

SEC-ENV-REQ 20: Access control should be dynamically managed and easily updated.

Explanation:

Although the number of I2RS clients is expected to be lower than the number of applications, as I2RS agents provide access to the routing resource, it is of primary importance that an access can be granted or revoke in an efficient way.

SEC-ENV-REQ 21: I2RS clients and I2RS agents should be uniquely identified in the network to enable centralized management of the I2RS access control policies.

Explanation:

Centralized management of the access control policies of an I2RS plane with network that hosts several I2RS applications, clients and agents requires that each devices can be identified.

4.2. I2RS Agent Access Control Policies

Overview:

The I2RS agent access control restricts routing system resource access to authorized identities - possible access policies may be none, read or write. The initiator of an access request to a routing resource is always an application. However, it remains challenging for the I2RS agent to establish its access control policies based on the application that initiates the request.

First, when an I2RS client acts as a broker, the I2RS agent may not be able to authenticate the application. In that sense, the I2RS agent relies on the capability of the I2RS client to authenticate the applications and apply the appropriated I2RS client access control.

Second, an I2RS agent may not uniquely identify a piece of software implementing an I2RS client. In fact, an I2RS client may be provided multiple identities which can be associated to different roles or privileges. The I2RS client is left responsible for using them appropriately according to the application.

Third, each I2RS client may contact various I2RS agent with different privileges and access control policies.

4.2.1. I2RS Agent Access Control

This section provides recommendations on the I2RS agent access control policies to keep I2RS access control coherent within the I2RS plane.

Requirements:

SEC-ENV-REQ 22: I2RS agent access control policies should be primarily based on the I2RS clients as described in [RFC7921].

SEC-ENV-REQ 23: I2RS agent access control policies MAY be based on the application if the application identity has been authenticated by the I2RS client and passed via the secondary identity to the I2RS agent.

SEC-ENV-REQ 24: The I2RS agent should know which identity (E.g. system user) performed the latest update of the

routing resource. This is true for an identity inside and outside the I2RS plane, so the I2RS agent can appropriately perform an update according to the priorities associated to the requesting identity and the identity that last updated the resource.

SEC-ENV-REQ 25: the I2RS agent should have an "I2RS agent overwrite Policy" that indicates how identities can be prioritized. This requirement is also described in section 7.6 of [RFC7921]. Similar requirements exist for components within the I2RS plane, but this is within the scope of the I2RS protocol security requirements [I-D.ietf-i2rs-protocol-security-requirements].

Explanation:

If the I2RS application is authenticated to the I2RS client, and the I2RS client is authenticated to the I2RS agent, and the I2RS client uses the opaque secondary identifier to pass an authenticated identifier to the I2RS agent, then this identifier may be used for access control. However, caution should be taken when using this chain of authentication since the secondary identifier is intended in the I2RS protocol only to aid traceability.

From the environment perspective the I2RS agent MUST be aware when the resource has been modified outside the I2RS plane by another plane (management, control, or forwarding). The prioritization between the different planes should set a deterministic policy that allows the collision of two planes (I2RS plane and another plane) to be resolved via an overwrite policy in the I2RS agent.

Similar requirements exist for knowledge about identities within the I2RS plane which modify things in the routing system, but this is within the scope of the I2RS protocol’s requirements for ephemeral state [I-D.ietf-i2rs-ephemeral-state] and security requirements [I-D.ietf-i2rs-protocol-security-requirements].

4.2.2. I2RS Client Access Control Policies

Overview:

The I2RS client access control policies are responsible for authenticating the application managing the privileges for the applications, and enforcing access control to resources by the applications.

Requirements:

REQ 26: I2RS client should authenticate its applications. If the I2RS client acts as a broker and supports multiple applications, it should authenticate each application.

REQ 27: I2RS client should define access control policies associated to each applications. An access to a routing resource by an application should not be forwarded immediately and transparently by the I2RS client based on the I2RS agent access control policies. The I2RS client should first check whether the application has sufficient privileges, and if so send an access request to the I2RS agent.

Explanation:

If no authentication mechanisms have being provided between the I2RS client and the application, then the I2RS client must be dedicated to a single application. By doing so, application authentication relies on the I2RS authentication mechanisms between the I2RS client and the I2RS agent.

If an I2RS client has multiple identities that are associated with different privileges for accessing an I2RS agent(s), the I2RS client access control policies should specify the I2RS client identity with the access control policy.

4.2.3. Application and Access Control Policies

Overview

Applications do not enforce access control policies. Instead these are enforced by the I2RS clients and the I2RS agents. This section provides recommendations for applications in order to ease I2RS access control by the I2RS client and the I2RS agent.

Requirements:

SEC-ENV-REQ 28: Applications SHOULD be uniquely identified by their associated I2RS clients

Explanation:

Different application may use different methods (or multiple methods) to communicate with its associated I2RS client, and each application may not use the same form of an application identifier. However, the I2RS client must obtain an identifier for each application. One method for this identification can be a system user id.

SEC-ENV-REQ 29: Each application SHOULD be associated to a restricted number of I2RS clients.

Explanation:

The I2RS client provides access to resource on its behalf and this access should only be granted for trusted applications, or applications with an similar level of trust. This does not prevent an I2RS client to host a large number of applications with the same levels of trust.

SEC-ENV-REQ 30: An application SHOULD be provided means and methods to contact their associated I2RS client.

Explanation:

It is obvious when an I2RS client belongs to the application as part of a module or a library that the application can communicate with a I2RS client. Similarly, if the application runs into a dedicated system with a I2RS client, it is obvious which I2RS client the application should contact. If the application connects to the I2RS client remotely, the application needs some means to retrieve the necessary information to contact its associated I2RS client (e.g. an IP address or a FQDN).

5. I2RS Application Isolation

A key aspect of the I2RS architecture is the network oriented application that uses the I2RS high bandwidth programmatic interface to monitor or change one or more routing systems. I2RS applications could be control by a single entity or serve various tenants of the network. If multiple entities use an I2RS application to monitor or change the network, security policies must preserve the isolation of each entity’s control and not let malicious entities controlling one I2RS application interfere with other I2RS applications.

This section discusses both security aspects related to programmability as well as application isolation in the I2RS architecture.

5.1. Robustness Toward Programmability

Overview

I2RS provides a programmatic interface in and out of the Internet routing system which provides the following advantages for security:

- the use of automation reduces configuration errors;

- the programmatic interface enables fast network reconfiguration and agility in adapting to network attacks; and
- monitoring facilities to detect a network attack, and configuration changes which can help mitigate the network attack.

Programmability allows applications to flexible control which may cause problems due to:

- applications which belong to different tenants with different objectives,
- applications which lack coordination resulting in unstable routing configurations such as oscillations between network configurations, and creation of loops. For example, one application may monitor a state and change to positive, and a second application performs the reverse operation (turns it negative). This fluctuation can cause a routing system to become unstable.

The I2RS plane requires data and application isolation to prevent such situations from happening. However, to guarantee the network stability constant monitoring and error detection are recommended.

Requirement:

SEC-ENV-REQ 31: The I2RS agents should monitor constantly parts of the system for which I2RS clients or applications have provided requests. It should also be able to detect any I2RS clients or applications causing problems that may leave the routing system in an unstable state.

Explanation:

In the least, monitoring consists of logging events and receiving streams of data. I2RS Plane implementations should monitor the I2RS applications and I2RS clients for potential problems. The cause for the I2RS clients or applications providing problematic requests can be failures in the implementation code or malicious intent.]

5.2. Application Isolation

5.2.1. DoS

Overview:

Requirements for robustness to DoS attacks have been addressed in the communication channel section [RFC7921]. This section focuses on requirements for application isolation that help prevent DoS.

Requirements:

SEC-ENV-REQ 32: In order to prevent DoS, it is recommended the I2RS agent control the resources allocated to each I2RS client. I2RS clients that act as broker may not be protected efficiently against these attacks unless the broker performs resource controls for the hosted applications.

SEC-ENV-REQ 33: I2RS agent SHOULD not make a response redirection unless the redirection is previously validated and agreed by the destination.

SEC-ENV-REQ 34: I2RS Applications should avoid the use of underlying protocols that are not robust enough to protect against reflection attacks.

Explanation:

The I2RS interface is used by applications to interact with the routing states. If the I2RS client is shared between multiple applications, one application can use the I2RS client to perform DoS or DDoS attacks on the I2RS agent(s) and through the I2RS agents attack the network. DoS attack targeting the I2RS agent would consist in providing requests that keep the I2RS agent busy for a long time. These attacks on the I2RS agent may involve an application (requesting through an I2RS Client) heavy computation by the I2RS agent in order to block operations like disk access.

Some DoS attacks may attack the I2RS Client’s reception of notification and monitoring data streams over the network. Other DoS attacks may focus on the application directly by performing reflection attacks to reflect traffic. Such an attack could be performed by first detecting an application is related to monitoring the RIB or changing the RIB. Reflection-based DoS may also attack at various levels in the stack utilizing UDP at the service to redirect data to a specific repository

I2RS implementation should consider how to protect I2RS against such attacks.

5.2.2. Application Logic Control

Overview

This section examines how application logic must be designed to ensure application isolation.

Requirements:

SEC-ENV-REQ 35: Application logic should remain opaque to external listeners. Application logic may be partly hidden by encrypting the communication between the I2RS client and the I2RS agent. Additional ways to obfuscate the communications may involve sending random messages of various sizes. Such strategies have to be balanced with network load. Note that I2RS client broker are more likely to hide the application logic compared to I2RS client associated to a single application.

Explanation:

Applications use the I2RS interface in order to update the routing system. These updates may be driven by behaviour on the forwarding plane or any external behaviours. In this case, correlating observation with the I2RS traffic may enable the derivation the application logic. Once the application logic has been derived, a malicious application may generate traffic or any event in the network in order to activate the alternate application.

6. Security Considerations

This whole document is about security requirements for the I2RS environment. To protect personal privacy, any identifier (I2RS application identifier, I2RS client identifier, or I2RS agent identifier) should not contain personal identifiable information.

7. IANA Considerations

No IANA considerations for this requirements.

8. Acknowledgments

A number of people provided a significant amount of helpful comments and reviews. Among them the authors would like to thank Russ White, Russ Housley, Thomas Nadeau, Juergen Schoenwaelder, Jeffrey Haas, Alia Atlas, and Linda Dunbar.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC7920] Atlas, A., Ed., Nadeau, T., Ed., and D. Ward, "Problem Statement for the Interface to the Routing System", RFC 7920, DOI 10.17487/RFC7920, June 2016, <http://www.rfc-editor.org/info/rfc7920>.

[RFC7921] Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", RFC 7921, DOI 10.17487/RFC7921, June 2016, <http://www.rfc-editor.org/info/rfc7921>.

[RFC7922] Clarke, J., Salgueiro, G., and C. Pignataro, "Interface to the Routing System (I2RS) Traceability: Framework and Information Model", RFC 7922, DOI 10.17487/RFC7922, June 2016, <http://www.rfc-editor.org/info/rfc7922>.

[RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, DOI 10.17487/RFC7923, June 2016, <http://www.rfc-editor.org/info/rfc7923>.

[I-D.ietf-i2rs-protocol-security-requirements] Hares, S., Migault, D., and J. Halpern, "I2RS Security Related Requirements", draft-ietf-i2rs-protocol-security-requirements-17 (work in progress), September 2016.

9.2. Informative References

[I-D.ietf-i2rs-ephemeral-state] Haas, J. and S. Hares, "I2RS Ephemeral State Requirements", draft-ietf-i2rs-ephemeral-state-23 (work in progress), November 2016.

[I-D.ietf-netconf-rfc6536bis] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", draft-ietf-netconf-rfc6536bis-01 (work in progress), March 2017.

[I-D.ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture", draft-ietf-netmod-revised-datastores-01 (work in progress), March 2017.

Authors' Addresses

Daniel Migault
Ericsson
8400 boulevard Decarie
Montreal, QC H4P 2N2
Canada

Phone: +1 514-452-2160
Email: daniel.migault@ericsson.com

Joel Halpern
Ericsson

Email: Joel.Halpern@ericsson.com

Susan Hares
Huawei
7453 Hickory Hill
Saline, MI 48176
USA

Email: shares@ndzh.com

OPSEC
Internet-Draft
Intended status: Informational
Expires: September 14, 2017

K. Chittimaneni
Dropbox Inc.
M. Kaeo
Double Shot Security
E. Vyncke, Ed.
Cisco

March 13, 2017

Operational Security Considerations for IPv6 Networks
draft-ietf-opsec-v6-10

Abstract

Knowledge and experience on how to operate IPv4 securely is available: whether it is the Internet or an enterprise internal network. However, IPv6 presents some new security challenges. RFC 4942 describes the security issues in the protocol but network managers also need a more practical, operations-minded document to enumerate advantages and/or disadvantages of certain choices.

This document analyzes the operational security issues in all places of a network (enterprises, service providers and residential users) and proposes technical and procedural mitigations techniques.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 14, 2017.

Copyright Notice

Table of Contents

1. Introduction

Running an IPv6 network is new for most operators not only because they are not yet used to large scale IPv6 networks but also because there are subtle differences between IPv4 and IPv6 especially with respect to security. For example, all layer-2 interactions are now done using Neighbor Discovery Protocol [RFC4861] rather than using Address Resolution Protocol [RFC0826]. Also, there are subtle differences between NAT44 [RFC2993] and NPTv6 [RFC6296] which are explicitly pointed out in the latter's security considerations section.

IPv6 networks are deployed using a variety of techniques, each of which have their own specific security concerns.

This document complements [RFC4942] by listing all security issues when operating a network utilizing varying transition technologies and updating with ones that have been standardized since 2007. It also provides more recent operational deployment experiences where warranted.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when they appear in ALL CAPS. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

2. Generic Security Considerations

2.1. Addressing Architecture

IPv6 address allocations and overall architecture are an important part of securing IPv6. Initial designs, even if intended to be temporary, tend to last much longer than expected. Although initially IPv6 was thought to make renumbering easy, in practice, it may be extremely difficult to renumber without a good IP Addresses Management (IPAM) system.

Once an address allocation has been assigned, there should be some thought given to an overall address allocation plan. With the abundance of address space available, an address allocation may be structured around services along with geographic locations, which then can be a basis for more structured security policies to permit or deny services between geographic regions.

A common question is whether companies should use PI vs PA space [RFC7381], but from a security perspective there is little difference. However, one aspect to keep in mind is who has administrative ownership of the address space and who is technically responsible if/when there is a need to enforce restrictions on routability of the space due to malicious criminal activity.

2.1.1. Statically Configured Addresses

When considering how to assign statically configured addresses it is necessary to take into consideration the effectiveness of perimeter security in a given environment. There is a trade-off between ease of operational deployment where some portions of the IPv6 address could be easily recognizable for operational debugging and troubleshooting versus the risk of scanning; [SCANNING] shows that there are scientifically based mechanisms that make scanning for IPv6 reachable nodes more realizable than expected; see also [RFC7707]. The use of common multicast groups which are defined for important networked devices and the use of commonly repeated addresses could make it easy to figure out which devices are name servers, routers or other critical devices.

While in some environments the security is so poor that obfuscating addresses is considered a benefit; it is a better practice to ensure that perimeter rules are actively checked and enforced and that statically configured addresses follow some logical allocation scheme for ease of operation.

2.1.2. Use of ULAs

ULAs are intended for scenarios where IP addresses will not have global scope so they should not appear in the global BGP routing table. The implicit expectation from the RFC is that all ULAs will be randomly created as /48s. Any use of ULAs that are not created as a /48 violates RFC4193 [RFC4193].

ULAs could be useful for infrastructure hiding as described in RFC4864 [RFC4864]. Alternatively Link-Local addresses RFC7404 [RFC7404] could also be used. Although ULAs are supposed to be used in conjunction with global addresses for hosts that desire external connectivity, a few operators chose to use ULAs in conjunction with some sort of address translation at the border in order to maintain a perception of parity between their IPv4 and IPv6 setup. Some operators believe that stateful IPv6 Network Address and Port Translation (NAPT) provides some security not provided by NPTv6 (the authors of this document do not share this point of view). The use of stateful IPv6 NAPT would be problematic in trying to track specific machines that may source malware although this is less of an issue if appropriate logging is done which includes utilizing accurate timestamps and logging a node’s source ports RFC6302 [RFC6302]. Another typical argument in favor of ULA is that there are too many mistakes made with ACL filters at the edge and the use of ULAs could make things easier to set filters.

The use of ULA does not isolate ’by magic’ the part of the network using ULA from other parts of the network (including the Internet). Although section 4.1 of RFC4193 [RFC4193] explicitly states "If BGP is being used at the site border with an ISP, the default BGP configuration must filter out any Local IPv6 address prefixes, both incoming and outgoing.", the operational reality is that this guideline is not always followed. As written, RFC4193 makes no changes to default routing behavior of exterior protocols. Therefore, routers will happily forward packets whose source or destination address is ULA as long as they have a route to the destination and there is no ACL blocking those packets. This means that using ULA does not prevent route and packet filters having to be implemented and monitored. This also means that all Internet transit networks should consider ULA as source or destination as bogons packets and drop them.

It is important to carefully weigh the benefits of using ULAs versus utilizing a section of the global allocation and creating a more effective filtering strategy. It is also important to note that the IETF does not recommend the use of ULA and NPTv6.

2.1.3. Point-to-Point Links

RFC6164 [RFC6164] recommends the use of /127 for inter-router point-to-point links. A /127 prevents the ping-pong attack between routers. However, it should be noted that at the time of this writing, there are still many networks out there that follow the advice provided by RFC3627 [RFC3627] (obsoleted and marked Historic by RFC6547 [RFC6547]) and therefore continue to use /64's and/or /112's. We recommend that the guidance provided by RFC6164 be followed.

Some environments are also using link-local addressing for point-to-point links. While this practice could further reduce the attack surface against infrastructure devices, the operational disadvantages need also to be carefully considered RFC7404 [RFC7404].

2.1.4. Temporary Addresses - Privacy Extensions for SLAAC

Normal stateless address autoconfiguration (SLAAC) relies on the automatically generated EUI-64 address, which together with the /64 prefix makes up the global unique IPv6 address. The EUI-64 address is generated from the MAC address. Randomly generating an interface ID, as described in [RFC4941], is part of SLAAC with so-called privacy extension addresses and used to address some privacy concerns. Privacy extension addresses a.k.a. temporary addresses may help to mitigate the correlation of activities of a node within the same network, and may also reduce the attack exposure window.

As privacy extension addresses could also be used to obfuscate some malevolent activities (whether on purpose or not), it is advised in scenarios where user attribution is important to rely on a layer-2 authentication mechanism such as IEEE 802.1X [IEEE-802.1X] with the appropriate RADIUS accounting (Section 2.6.1.6) or to disable SLAAC and rely only on DHCPv6. However, in scenarios where anonymity is a strong desire (protecting user privacy is more important than user attribution), privacy extension addresses should be used.

Using privacy extension addresses prevents the operator from building a priori host specific access control lists (ACLs). It must be noted that recent versions of Windows do not use the MAC address anymore to build the stable address but use a mechanism similar to the one described in [RFC7217], this also means that such an ACL cannot be configured based solely on the MAC address of the nodes, diminishing the value of such ACL. On the other hand, different VLANs are often used to segregate users, in this case ACL can rely on a /64 prefix per VLAN rather than a per host ACL entry.

The decision to utilize privacy extension addresses can come down to whether the network is managed versus unmanaged. In some environments full visibility into the network is required at all times which requires that all traffic be attributable to where it is sourced or where it is destined to within a specific network. This situation is dependent on what level of logging is performed. If logging considerations include utilizing accurate timestamps and logging a node's source ports [RFC6302] then there should always exist appropriate user attribution needed to get to the source of any malware originator or source of criminal activity.

Disabling SLAAC and privacy extensions addresses can be done by sending Router Advertisement with a hint to get addresses via DHCPv6 by setting the M-bit but also disabling SLAAC by resetting all A-bits in all prefix information options sent in the Router Advertisement message.

2.1.5. Privacy consideration of Addresses

However, there are several privacy issues still present with [RFC4941] such as host tracking, and address scanning attacks are still possible. More details are provided in Appendix A. of [RFC7217] and in [RFC7721].

2.1.6. DHCP/DNS Considerations

Many environments use DHCPv6 to allocate addresses to ensure audit-ability and traceability (but see Section 2.6.1.5). A main security concern is the ability to detect and counteract against rogue DHCP servers (Section 2.3.2).

DNS is often used for malware activities and while there are no fundamental differences with IPv4 and IPv6 security concerns, there are specific consideration in DNS64 RFC6147 [RFC6147] environments that need to be understood. Specifically the interactions and potential to interference with DNSsec implementation need to be understood - these are pointed out in detail in Section 2.7.3.2.

2.2. Extension Headers

The extension headers are one of the most critical differentiator between IPv4 and IPv6. Obvioulsy, the IPSec [RFC4301]IPsec extension headers (AH [RFC4302] and ESP [RFC4303]) are the most used IPv6 extension headers as they were back-ported to IPv4. But, they are other ones and there are even sub-type within each defined extension headers. The IANA has closed the the existing empty "Next Header Types" registry to new entries and is redirecting its users to a new "IPv6 Extension Header Types" registry.

A good read about extension headers is RFC7045 [RFC7045]. RFC7872 [RFC7872] seems to indicate that packets with some extension headers may not traverse safely the Internet.

2.2.1. Order and Repetition of Extension Headers

While RFC2460 defines the order and the maximum repetition of extension headers, there are still IPv6 implementations at the time of writing this document which do not support a wrong order of headers (such as ESP before routing) or an illegal repetition of headers (such as twice routing header). The same applies for options contained in the extension headers (see [I-D.kampanakis-6man-ipv6-eh-parsing]). In some case, it has lead to node crash when receiving or forwarding wrongly formated packets.

2.2.2. Hop-by-Hop Extension Header

The hop-by-hop extension header when present in an IPv6 packet forces all nodes in the path to inspect this header. This is of course a large avenue for a denial of service as most if not all routers cannot process this kind of packets in hardware but have to 'punt' this packet for software processing. See also [I-D.ietf-6man-hbh-header-handling].

2.2.3. Fragmentation Extension Header

The fragmentation extension header is used by the source when it has to fragment packets. RFC7112 [RFC7112] explains why it is important to:

- firewall and security devices should drop first fragment not containing enough of the layer-4 header;
- destiantion node should ignore first fragment not containing the entire IPv6 header chain.

Else, stateless access control list could be bypassed by an hostile party. RFC6980 [RFC6980] applies the same rule to NDP and the RA-guard function.

2.3. Link-Layer Security

IPv6 relies heavily on the Neighbor Discovery protocol (NDP) RFC4861 [RFC4861] to perform a variety of link operations such as discovering other nodes on the link, resolving their link-layer addresses, and finding routers on the link. If not secured, NDP is vulnerable to various attacks such as router/neighbor message spoofing, redirect attacks, Duplicate Address Detection (DAD) DoS attacks, etc. many of

these security threats to NDP have been documented in IPv6 ND Trust Models and Threats RFC3756 [RFC3756] and in RFC6583 [RFC6583].

2.3.1. SeND and CGA

SEcure Neighbor Discovery (SeND), as described in RFC3971 [RFC3971], is a mechanism that was designed to secure ND messages. This approach involves the use of new NDP options to carry public key based signatures. Cryptographically Generated Addresses (CGA), as described in RFC3972 [RFC3972], are used to ensure that the sender of a Neighbor Discovery message is the actual "owner" of the claimed IPv6 address. A new NDP option, the CGA option, was introduced and is used to carry the public key and associated parameters. Another NDP option, the RSA Signature option, is used to protect all messages relating to neighbor and Router discovery.

SeND protects against:

- Neighbor Solicitation/Advertisement Spoofing
- Neighbor Unreachability Detection Failure
- Duplicate Address Detection DoS Attack
- Router Solicitation and Advertisement Attacks
- Replay Attacks
- Neighbor Discovery DoS Attacks

SeND does NOT:

- Protect statically configured addresses
- Protect addresses configured using fixed identifiers (i.e. EUI-64)
- Provide confidentiality for NDP communications
- Compensate for an unsecured link - SEND does not require that the addresses on the link and Neighbor Advertisements correspond

However, at this time and after many years after their specifications, CGA and SeND do not have wide support from generic operating systems; hence, their usefulness is limited.

2.3.2. Securing DHCP

Dynamic Host Configuration Protocol for IPv6 (DHCPv6), as detailed in RFC3315 [RFC3315], enables DHCP servers to pass configuration parameters such as IPv6 network addresses and other configuration information to IPv6 nodes. DHCP plays an important role in any large network by providing robust stateful configuration and autoregistration of DNS Host Names.

The two most common threats to DHCP clients come from malicious (a.k.a. rogue) or unintentionally misconfigured DHCP servers. A malicious DHCP server is established with the intent of providing incorrect configuration information to the client to cause a denial of service attack or mount a man in the middle attack. While unintentionall, a misconfigured DHCP server can have the same impact. Additional threats against DHCP are discussed in the security considerations section of RFC3315 [RFC3315]DHCP-shield

RFC7610 [RFC7610] specifies a mechanism for protecting connected DHCPv6 clients against rogue DHCPv6 servers. This mechanism is based on DHCPv6 packet-filtering at the layer-2 device; the administrator specifies the interfaces connected to DHCPv6 servers.

It is recommended to use DHCP-shield.

2.3.3. ND/RA Rate Limiting

Neighbor Discovery (ND) can be vulnerable to denial of service (DoS) attacks in which a router is forced to perform address resolution for a large number of unassigned addresses. Possible side effects of this attack preclude new devices from joining the network or even worse rendering the last hop router ineffective due to high CPU usage. Easy mitigative steps include rate limiting Neighbor Solicitations, restricting the amount of state reserved for unresolved solicitations, and clever cache/timer management.

RFC6583 [RFC6583] discusses the potential for DoS in detail and suggests implementation improvements and operational mitigation techniques that may be used to mitigate or alleviate the impact of such attacks. Here are some feasible mitigation options that can be employed by network operators today:

- Ingress filtering of unused addresses by ACL, route filtering, longer than /64 prefix; These require static configuration of the addresses.

- Tuning of NDP process (where supported).

Additionally, IPv6 ND uses multicast extensively for signaling messages on the local link to avoid broadcast messages for on-the-wire efficiency. However, this has some side effects on wifi networks, especially a negative impact on battery life of smartphones and other battery operated devices that are connected to such networks. The following drafts are actively discussing methods to rate limit RAs and other ND messages on wifi networks in order to address this issue:

- [I-D.thubert-savi-ra-throttler]

- [I-D.chakrabarti-nordmark-6man-efficient-nd]

2.3.4. ND/RA Filtering

Router Advertisement spoofing is a well-known attack vector and has been extensively documented. The presence of rogue RAs, either intentional or malicious, can cause partial or complete failure of operation of hosts on an IPv6 link. For example, a host can select an incorrect router address which can be used as a man-in-the-middle (MITM) attack or can assume wrong prefixes to be used for stateless address configuration (SLAAC). RFC6104 [RFC6104] summarizes the scenarios in which rogue RAs may be observed and presents a list of possible solutions to the problem. RFC6105 [RFC6105] (RA-Guard) describes a solution framework for the rogue RA problem where network segments are designed around switching devices that are capable of identifying invalid RAs and blocking them before the attack packets actually reach the target nodes.

However, several evasion techniques that circumvent the protection provided by RA-Guard have surfaced. A key challenge to this mitigation technique is introduced by IPv6 fragmentation. An attacker can conceal the attack by fragmenting his packets into multiple fragments such that the switching device that is responsible for blocking invalid RAs cannot find all the necessary information to perform packet filtering in the same packet. RFC7113 [RFC7113] describes such evasion techniques, and provides advice to RA-Guard implementers such that the aforementioned evasion vectors can be eliminated.

Given that the IPv6 Fragmentation Header can be leveraged to circumvent current implementations of RA-Guard, RFC6980 [RFC6980] updates RFC4861 [RFC4861] such that use of the IPv6 Fragmentation Header is forbidden in all Neighbor Discovery messages except "Certification Path Advertisement", thus allowing for simple and effective measures to counter Neighbor Discovery attacks.

The Source Address Validation Improvements (SAVI) working group has worked on other ways to mitigate the effects of such attacks. RFC7513 [RFC7513] would help in creating bindings between a DHCPv4 RFC2131 [RFC2131] /DHCPv6 RFC3315 [RFC3315] assigned source IP address and a binding anchor RFC7039 [RFC7039] on a SAVI device. Also, RFC6620 [RFC6620] describes how to glean similar bindings when DHCP is not used. The bindings can be used to filter packets generated on the local link with forged source IP address.

It is still recommended that RA-Guard be be employed as a first line of defense against common attack vectors including misconfigured hosts.

2.3.5. 3GPP Link-Layer Security

The 3GPP link is a point-to-point like link that has no link-layer address. This implies there can only be an end host (the mobile hand-set) and the first-hop router (i.e., a GPRS Gateway Support Node (GGSN) or a Packet Gateway (PGW)) on that link. The GGSN/PGW never configures a non link-local address on the link using the advertised /64 prefix on it. The advertised prefix must not be used for on-link determination. There is no need for an address resolution on the 3GPP link, since there are no link-layer addresses. Furthermore, the GGSN/PGW assigns a prefix that is unique within each 3GPP link that uses IPv6 stateless address autoconfiguration. This avoids the necessity to perform DAD at the network level for every address built by the mobile host. The GGSN/PGW always provides an IID to the cellular host for the purpose of configuring the link-local address and ensures the uniqueness of the IID on the link (i.e., no collisions between its own link-local address and the mobile host’s one).

The 3GPP link model itself mitigates most of the known NDP-related Denial-of-Service attacks. In practice, the GGSN/PGW only needs to route all traffic to the mobile host that falls under the prefix assigned to it. As there is also a single host on the 3GPP link, there is no need to defend that IPv6 address.

See Section 5 of RFC6459 [RFC6459] for a more detailed discussion on the 3GPP link model, NDP on it and the address configuration detail.

2.4. Control Plane Security

RFC6192 [RFC6192] defines the router control plane. This definition is repeated here for the reader’s convenience.

Modern router architecture design maintains a strict separation of forwarding and router control plane hardware and software. The

router control plane supports routing and management functions. It is generally described as the router architecture hardware and software components for handling packets destined to the device itself as well as building and sending packets originated locally on the device. The forwarding plane is typically described as the router architecture hardware and software components responsible for receiving a packet on an incoming interface, performing a lookup to identify the packet's IP next hop and determine the best outgoing interface towards the destination, and forwarding the packet out through the appropriate outgoing interface.

While the forwarding plane is usually implemented in high-speed hardware, the control plane is implemented by a generic processor (named router processor RP) and cannot process packets at a high rate. Hence, this processor can be attacked by flooding its input queue with more packets than it can process. The control plane processor is then unable to process valid control packets and the router can lose OSPF or BGP adjacencies which can cause a severe network disruption.

The mitigation technique is:

- To drop non-legit control packet before they are queued to the RP (this can be done by a forwarding plane ACL) and

- To rate limit the remaining packets to a rate that the RP can sustain. Protocol specific protection should also be done (for example, a spoofed OSPFv3 packet could trigger the execution of the Dijkstra algorithm, therefore the number of Dijsktra execution should be also rate limited).

This section will consider several classes of control packets:

- Control protocols: routing protocols: such as OSPFv3, BGP and by extension Neighbor Discovery and ICMP

- Management protocols: SSH, SNMP, IPfix, etc

- Packet exceptions: which are normal data packets which requires a specific processing such as generating a packet-too-big ICMP message or having the hop-by-hop extension header.

2.4.1. Control Protocols

This class includes OSPFv3, BGP, NDP, ICMP.

An ingress ACL to be applied on all the router interfaces SHOULD be configured such as:

- drop OSPFv3 (identified by Next-Header being 89) and RIPng (identified by UDP port 521) packets from a non link-local address

- allow BGP (identified by TCP port 179) packets from all BGP neighbors and drop the others

- allow all ICMP packets (transit and to the router interfaces)

Note: dropping OSPFv3 packets which are authenticated by IPsec could be impossible on some routers whose ACL are unable to parse the IPsec ESP or AH extension headers.

Rate limiting of the valid packets SHOULD be done. The exact configuration obviously depends on the power of the Route Processor.

2.4.2. Management Protocols

This class includes: SSH, SNMP, syslog, NTP, etc

An ingress ACL to be applied on all the router interfaces SHOULD be configured such as:

- Drop packets destined to the routers except those belonging to protocols which are used (for example, permit TCP 22 and drop all when only SSH is used);

- Drop packets where the source does not match the security policy, for example if SSH connections should only be originated from the NOC, then the ACL should permit TCP port 22 packets only from the NOC prefix.

Rate limiting of the valid packets SHOULD be done. The exact configuration obviously depends on the power of the Route Processor.

2.4.3. Packet Exceptions

This class covers multiple cases where a data plane packet is punted to the route processor because it requires specific processing:

- generation of an ICMP packet-too-big message when a data plane packet cannot be forwarded because it is too large;

- generation of an ICMP hop-limit-expired message when a data plane packet cannot be forwarded because its hop-limit field has reached 0;

- generation of an ICMP destination-unreachable message when a data plane packet cannot be forwarded for any reason;

- o processing of the hop-by-hop extension header (see also [I-D.ietf-6man-hbh-header-handling]);

- o or more specific to some router implementation: an oversized extension header chain which cannot be processed by the hardware and force the packet to be punted to the generic router CPU.

On some routers, not everything can be done by the specialized data plane hardware which requires some packets to be 'punted' to the generic RP. This could include for example the processing of a long extension header chain in order to apply an ACL based on layer 4 information. RFC6980 [RFC6980] and more generally RFC7112 [RFC7112] highlights the security implications of oversized extension header chains on routers and updates RFC2460 [RFC2460] such that the first fragment of a packet is required to contain the entire IPv6 header chain.

An ingress ACL cannot help to mitigate a control plane attack using those packet exceptions. The only protection for the RP is to limit the rate of those packet exceptions forwarded to the RP, this means that some data plane packets will be dropped without any ICMP messages back to the source which will cause Path MTU holes. But, there is no other solution.

In addition to limiting the rate of data plane packets queued to the RP, it is also important to limit the generation rate of ICMP messages both the save the RP but also to prevent an amplification attack using the router as a reflector.

2.5. Routing Security

Routing security in general can be broadly divided into three sections:

1. Authenticating neighbors/peers

2. Securing routing updates between peers

3. Route filtering

[RFC7454] covers these sections specifically for BGP in detail.

2.5.1. Authenticating Neighbors/Peers

A basic element of routing is the process of forming adjacencies, neighbor, or peering relationships with other routers. From a security perspective, it is very important to establish such relationships only with routers and/or administrative domains that

one trusts. A traditional approach has been to use MD5 HMAC, which allows routers to authenticate each other prior to establishing a routing relationship.

OSPFv3 can rely on IPsec to fulfill the authentication function. However, it should be noted that IPsec support is not standard on all routing platforms. In some cases, this requires specialized hardware that offloads crypto over to dedicated ASICs or enhanced software images (both of which often come with added financial cost) to provide such functionality. An added detail is to determine whether OSPFv3 IPsec implementations use AH or ESP-Null for integrity protection. In early implementations all OSPFv3 IPsec configurations relied on AH since the details weren't specified in RFC5340 [RFC5340] or RFC2740 [RFC2740] that was obsoleted by the former. However, the document which specifically describes how IPsec should be implemented for OSPFv3 RFC4552 [RFC4552] specifically states that ESP-Null MUST and AH MAY be implemented since it follows the overall IPsec standards wordings. OSPFv3 can also use normal ESP to encrypt the OSPFv3 payload to hide the routing information.

RFC7166 [RFC7166] (which obsoletes RFC6506 [RFC6506] changes OSPFv3's reliance on IPsec by appending an authentication trailer to the end of the OSPFv3 packets. This document does not specifically provide for a mechanism that will authenticate the specific originator of a packet. Rather, it will allow a router to confirm that the packet has indeed been issued by a router that had access to the shared authentication key.

With all authentication mechanisms, operators should confirm that implementations can support re-keying mechanisms that do not cause outages. There have been instances where any re-keying cause outages and therefore the tradeoff between utilizing this functionality needs to be weighed against the protection it provides.

2.5.2. Securing Routing Updates Between Peers

IPv6 initially mandated the provisioning of IPsec capability in all nodes. However, in the updated IPv6 Nodes Requirement standard RFC6434 [RFC6434] is now a SHOULD and not MUST implement. Theoretically it is possible, and recommended, that communication between two IPv6 nodes, including routers exchanging routing information be encrypted using IPsec. In practice however, deploying IPsec is not always feasible given hardware and software limitations of various platforms deployed, as described in the earlier section. Additionally, in a protocol such as OSPFv3 where adjacencies are formed on a one-to-many basis, IPsec key management becomes difficult to maintain and is not often utilized.

2.5.3. Route Filtering

Route filtering policies will be different depending on whether they pertain to edge route filtering vs internal route filtering. At a minimum, IPv6 routing policy as it pertains to routing between different administrative domains should aim to maintain parity with IPv4 from a policy perspective e.g.,

- Filter internal-use, non-globally routable IPv6 addresses at the perimeter
- Discard packets from and to bogon and reserved space
- Configure ingress route filters that validate route origin, prefix ownership, etc. through the use of various routing databases, e.g., RADB. There is additional work being done in this area to formally validate the origin ASs of BGP announcements in RFC6810 [RFC6810]

Some good recommendations for filtering can be found from Team CYMRU at [CYMRU].

2.6. Logging/Monitoring

In order to perform forensic research in case of any security incident or to detect abnormal behaviors, network operator should log multiple pieces of information.

This includes:

- logs of all applications when available (for example web servers);
- use of IP Flow Information Export [RFC7011] also known as IPfix;
- use of SNMP MIB [RFC4293];
- use of the Neighbor cache;
- use of stateful DHCPv6 [RFC3315] lease cache, especially when a relay agent [RFC6221] in layer-2 switches is used;
- use of RADIUS [RFC2866] for accounting records.

Please note that there are privacy issues related to how those logs are collected, kept and safely discarded. Operators are urged to check their country legislation.

All those pieces of information will be used for:

- forensic (Section 2.6.2.1) research to answer questions such as who did what and when?
- correlation (Section 2.6.2.3): which IP addresses were used by a specific node (assuming the use of privacy extensions addresses [RFC4941])
- inventory (Section 2.6.2.2): which IPv6 nodes are on my network?
- abnormal behavior detection (Section 2.6.2.4): unusual traffic patterns are often the symptoms of a abnormal behavior which is in turn a potential attack (denial of services, network scan, a node being part of a botnet, ...)

2.6.1. Data Sources

This section lists the most important sources of data that are useful for operational security.

2.6.1.1. Logs of Applications

Those logs are usually text files where the remote IPv6 address is stored in all characters (not binary). This can complicate the processing since one IPv6 address, 2001:db8::1 can be written in multiple ways such as:

- 2001:DB8::1 (in uppercase)
- 2001:0db8::0001 (with leading 0)
- and many other ways.

RFC 5952 [RFC5952] explains this problem in detail and recommends the use of a single canonical format (in short use lower case and suppress leading 0). This memo recommends the use of canonical format [RFC5952] for IPv6 addresses in all possible cases. If the existing application cannot log under the canonical format, then this memo recommends the use an external program in order to canonicalize all IPv6 addresses.

For example, this perl script can be used:

```
#!/usr/bin/perl -w
use strict ;
use warnings ;
use Socket ;
use Socket6 ;

my (@words, $word, $binary_address) ;

## go through the file one line at a time
while (my $line = <STDIN>) {
  chomp $line;
  foreach my $word (split /[\s+]/, $line) {
    $binary_address = inet_pton AF_INET6, $word ;
    if ($binary_address) {
      print inet_ntop AF_INET6, $binary_address ;
    } else {
      print $word ;
    }
    print " " ;
  }
  print "\n" ;
}
```

2.6.1.2. IP Flow Information Export by IPv6 Routers

IPfix [RFC7012] defines some data elements that are useful for security:

- in section 5.4 (IP Header fields): nextHeaderIPv6 and sourceIPv6Address;
- in section 5.6 (Sub-IP fields) sourceMacAddress.

Moreover, IPfix is very efficient in terms of data handling and transport. It can also aggregate flows by a key such as sourceMacAddress in order to have aggregated data associated with a specific sourceMacAddress. This memo recommends the use of IPfix and aggregation on nextHeaderIPv6, sourceIPv6Address and sourceMacAddress.

2.6.1.3. SNMP MIB by IPv6 Routers

RFC 4293 [RFC4293] defines a Management Information Base (MIB) for the two address families of IP. This memo recommends the use of:

- ipIfStatsTable table which collects traffic counters per interface;

- ipNetToPhysicalTable table which is the content of the Neighbor cache, i.e. the mapping between IPv6 and data-link layer addresses.

2.6.1.4. Neighbor Cache of IPv6 Routers

The neighbor cache of routers contains all mappings between IPv6 addresses and data-link layer addresses. It is usually available by two means:

- the SNMP MIB (Section 2.6.1.3) as explained above;
- also by connecting over a secure management channel (such as SSH or HTTPS) and explicitely requesting a neighbor cache dump.

The neighbor cache is highly dynamic as mappings are added when a new IPv6 address appears on the network (could be quite often with privacy extension addresses [RFC4941] or when they are removed when the state goes from UNREACH to removed (the default time for a removal per Neighbor Unreachability Detection [RFC4861] algorithm is 38 seconds for a typical host such as Windows 7). This means that the content of the neighbor cache must periodically be fetched every 30 seconds (to be on the safe side) and stored for later use.

This is an important source of information because it is trivial (on a switch not using the SAVI [RFC7039] algorithm) to defeat the mapping between data-link layer address and IPv6 address. Let us rephrase the previous statement: having access to the current and past content of the neighbor cache has a paramount value for forensic and audit trail.

2.6.1.5. Stateful DHCPv6 Lease

In some networks, IPv6 addresses are managed by stateful DHCPv6 server [RFC3315] that leases IPv6 addresses to clients. It is indeed quite similar to DHCP for IPv4 so it can be tempting to use this DHCP lease file to discover the mapping between IPv6 addresses and data-link layer addresses as it was usually done in the IPv4 era.

It is not so easy in the IPv6 era because not all nodes will use DHCPv6 (there are nodes which can only do stateless autoconfiguration) but also because DHCPv6 clients are identified not by their hardware-client address as in IPv4 but by a DHCP Unique ID (DUID) which can have several formats: some being the data-link layer address, some being data-link layer address prepended with time information or even an opaque number which is useless for operation security. Moreover, when the DUID is based on the data-link address, this address can be of any interface of the client (such as the

wireless interface while the client actually uses its wired interface to connect to the network).

If a lightweight DHCP relay agent [RFC6221] is used in the layer-2 switches, then the DHCP server also receives the Interface-ID information which could be save in order to identifity the interface of the switches which received a specific leased IPv6 address.

In short, the DHCPv6 lease file is less interesting than in the IPv4 era. DHCPv6 servers that keeps the relayed data-link layer address in addition to the DUID in the lease file do not suffer from this limitation. On a managed network where all hosts support DHCPv6, special care must be taken to prevent stateless autoconfiguration anyway (and if applicable) by sending RA with all announced prefixes without the A-bit set.

The mapping between data-link layer address and the IPv6 address can be secured by using switches implementing the SAVI [RFC7513] algorithms. Of course, this also requires that data-link layer address is protected by using layer-2 mechanism such as [IEEE-802.1X].

2.6.1.6. RADIUS Accounting Log

For interfaces where the user is authenticated via a RADIUS [RFC2866] server, and if RADIUS accounting is enabled, then the RADIUS server receives accounting Acct-Status-Type records at the start and at the end of the connection which include all IPv6 (and IPv4) addresses used by the user. This technique can be used notably for Wi-Fi networks with Wi-Fi Protected Address (WPA) or any other IEEE 802.1X [IEEE-802.1X]wired interface on an Ethernet switch.

2.6.1.7. Other Data Sources

There are other data sources that must be kept exactly as in the IPv4 network:

- historical mapping of IPv6 addresses to users of remote access VPN;
- historical mapping of MAC address to switch interface in a wired network.

2.6.2. Use of Collected Data

This section leverages the data collected as described before (Section 2.6.1) in order to achieve several security benefits.

2.6.2.1. Forensic

The forensic use case is when the network operator must locate an IPv6 address that was present in the network at a certain time or is still currently in the network.

The source of information can be, in decreasing order, neighbor cache, DHCP lease file. Then, the procedure is:

1. based on the IPv6 prefix of the IPv6 address find the router(s) which are used to reach this prefix;

2. based on this limited set of routers, on the incident time and on IPv6 address to retrieve the data-link address from live neighbor cache, from the historical data of the neighbor cache, or from the DHCP lease file;

3. based on the data-link layer address, look-up on which switch interface was this data-link layer address. In the case of wireless LAN, the RADIUS log should have the mapping between user identification and the MAC address.

At the end of the process, the interface where the malicious user was connected or the username that was used by the malicious user is found.

2.6.2.2. Inventory

RFC 7707 [RFC7707] (which obsoletes RFC 5157 [RFC5157]) is about the difficulties to scan an IPv6 network due to the vast number of IPv6 addresses per link. This has the side effect of making the inventory task difficult in an IPv6 network while it was trivial to do in an IPv4 network (a simple enumeration of all IPv4 addresses, followed by a ping and a TCP/UDP port scan). Getting an inventory of all connected devices is of prime importance for a secure operation of a network.

There are many ways to do an inventory of an IPv6 network.

The first technique is to use the IPfix information and extract the list of all IPv6 source addresses to find all IPv6 nodes that sent packets through a router. This is very efficient but alas will not discover silent node that never transmitted such packets... Also, it must be noted that link-local addresses will never be discovered by this means.

The second way is again to use the collected neighbor cache content to find all IPv6 addresses in the cache. This process will also discover all link-local addresses. See Section 2.6.1.4.

Another way works only for local network, it consists in sending a ICMP ECHO_REQUEST to the link-local multicast address ff02::1 which is all IPv6 nodes on the network. All nodes should reply to this ECHO_REQUEST per [RFC4443].

Other techniques involve enumerating the DNS zones, parsing log files, leveraging service discovery such as mDNS RFC6762 [RFC6762] and RFC6763 [RFC6763].

2.6.2.3. Correlation

In an IPv4 network, it is easy to correlate multiple logs, for example to find events related to a specific IPv4 address. A simple Unix grep command was enough to scan through multiple text-based files and extract all lines relevant to a specific IPv4 address.

In an IPv6 network, this is slightly more difficult because different character strings can express the same IPv6 address. Therefore, the simple Unix grep command cannot be used. Moreover, an IPv6 node can have multiple IPv6 addresses...

In order to do correlation in IPv6-related logs, it is advised to have all logs with canonical IPv6 addresses. Then, the neighbor cache current (or historical) data set must be searched to find the data-link layer address of the IPv6 address. Then, the current and historical neighbor cache data sets must be searched for all IPv6 addresses associated to this data-link layer address: this is the search set. The last step is to search in all log files (containing only IPv6 address in canonical format) for any IPv6 addresses in the search set.

2.6.2.4. Abnormal Behavior Detection

Abnormal behaviors (such as network scanning, spamming, denial of service) can be detected in the same way as in an IPv4 network

- sudden increase of traffic detected by interface counter (SNMP) or by aggregated traffic from IPfix records [RFC7012];
- change of traffic pattern (number of connection per second, number of connection per host...) with the use of IPfix [RFC7012]

2.6.3. Summary

While some data sources (IPfix, MIB, switch CAM tables, logs, ...) used in IPv4 are also used in the secure operation of an IPv6 network, the DHCPv6 lease file is less reliable and the neighbor cache is of prime importance.

The fact that there are multiple ways to express in a character string the same IPv6 address renders the use of filters mandatory when correlation must be done.

2.7. Transition/Coexistence Technologies

Some text

2.7.1. Dual Stack

Dual stack has established itself as the preferred deployment choice for most network operators without an MPLS core where 6PE RFC4798 [RFC4798] is quite common. Dual stacking the network offers many advantages over other transition mechanisms. Firstly, it is easy to turn on without impacting normal IPv4 operations. Secondly, perhaps more importantly, it is easier to troubleshoot when things break. Dual stack allows you to gradually turn IPv4 operations down when your IPv6 network is ready for prime time.

From an operational security perspective, this now means that you have twice the exposure. One needs to think about protecting both protocols now. At a minimum, the IPv6 portion of a dual stacked network should maintain parity with IPv4 from a security policy point of view. Typically, the following methods are employed to protect IPv4 networks at the edge:

- ACLs to permit or deny traffic
- Firewalls with stateful packet inspection

It is recommended that these ACLs and/or firewalls be additionally configured to protect IPv6 communications. Also, given the end-to-end connectivity that IPv6 provides, it is also recommended that hosts be fortified against threats. General device hardening guidelines are provided in Section 2.8

2.7.2. Transition Mechanisms

There are many tunnels used for specific use cases. Except when protected by IPsec [RFC4301], all those tunnels have a couple of security issues (most of them being described in RFC 6169 [RFC6169]);

- tunnel injection: a malevolent person knowing a few pieces of information (for example the tunnel endpoints and the used protocol) can forge a packet which looks like a legit and valid encapsulated packet that will gladly be accepted by the destination tunnel endpoint, this is a specific case of spoofing;

- traffic interception: no confidentiality is provided by the tunnel protocols (without the use of IPsec), therefore anybody on the tunnel path can intercept the traffic and have access to the clear-text IPv6 packet;

- service theft: as there is no authorization, even a non authorized user can use a tunnel relay for free (this is a specific case of tunnel injection);

- reflection attack: another specific use case of tunnel injection where the attacker injects packets with an IPv4 destination address not matching the IPv6 address causing the first tunnel endpoint to re-encapsulate the packet to the destination... Hence, the final IPv4 destination will not see the original IPv4 address but only one IPv4 address of the relay router.

- bypassing security policy: if a firewall or an IPS is on the path of the tunnel, then it will probably neither inspect not detect an malevolent IPv6 traffic contained in the tunnel.

To mitigate the bypassing of security policies, it could be helpful to block all default configuration tunnels by denying all IPv4 traffic matching:

- IP protocol 41: this will block ISATAP (Section 2.7.2.2), 6to4 (Section 2.7.2.4), 6rd (Section 2.7.2.5) as well as 6in4 (Section 2.7.2.1) tunnels;

- IP protocol 47: this will block GRE (Section 2.7.2.1) tunnels;

- UDP protocol 3544: this will block the default encapsulation of Teredo (Section 2.7.2.3) tunnels.

Ingress filtering [RFC2827] should also be applied on all tunnel endpoints if applicable to prevent IPv6 address spoofing.

As several of the tunnel techniques share the same encapsulation (i.e. IPv4 protocol 41) and embed the IPv4 address in the IPv6 address, there are a set of well-known looping attacks described in RFC 6324 [RFC6324], this RFC also proposes mitigation techniques.

2.7.2.1. Site-to-Site Static Tunnels

Site-to-site static tunnels are described in RFC 2529 [RFC2529] and in GRE [RFC2784]. As the IPv4 endpoints are statically configured and are not dynamic they are slightly more secure (bi-directional service theft is mostly impossible) but traffic interception ad tunnel injection are still possible. Therefore, the use of IPsec [RFC4301] in transport mode and protecting the encapsulated IPv4 packets is recommended for those tunnels. Alternatively, IPsec in tunnel mode can be used to transport IPv6 traffic over a non-trusted IPv4 network.

2.7.2.2. ISATAP

ISATAP tunnels [RFC5214] are mainly used within a single administrative domain and to connect a single IPv6 host to the IPv6 network. This means that endpoints and and the tunnel endpoint are usually managed by a single entity; therefore, audit trail and strict anti-spoofing are usually possible and this raises the overall security.

Special care must be taken to avoid looping attack by implementing the measures of RFC 6324 [RFC6324] and of RFC6964 [RFC6964].

IPsec [RFC4301] in transport or tunnel mode can be used to secure the IPv4 ISATAP traffic to provide IPv6 traffic confidentiality and prevent service theft.

2.7.2.3. Teredo

Teredo tunnels [RFC4380] are mainly used in a residential environment because that can easily traverse an IPv4 NAT-PT device thanks to its UDP encapsulation and they connect a single host to the IPv6 Internet. Teredo shares the same issues as other tunnels: no authentication, no confidentiality, possible spoofing and reflection attacks.

IPsec [RFC4301] for the transported IPv6 traffic is recommended.

The biggest threat to Teredo is probably for IPv4-only network as Teredo has been designed to easily traverse IPV4 NAT-PT devices which are quite often co-located with a stateful firewall. Therefore, if the stateful IPv4 firewall allows unrestricted UDP outbound and accept the return UDP traffic, then Teredo actually punches a hole in this firewall for all IPv6 traffic to the Internet and from the Internet. While host policies can be deployed to block Teredo in an IPv4-only network in order to avoid this firewall bypass, it would be more efficient to block all UDP outbound traffic at the IPv4 firewall

if deemed possible (of course, at least port 53 should be left open for DNS traffic).

2.7.2.4. 6to4

6to4 tunnels [RFC3056] require a public routable IPv4 address in order to work correctly. They can be used to provide either one IPv6 host connectivity to the IPv6 Internet or multiple IPv6 networks connectivity to the IPv6 Internet. The 6to4 relay is usually the anycast address defined in RFC3068 [RFC3068] which has been deprecated by RFC7526 [RFC7526], and is no more used by recent Operating Systems. Some security considerations are explained in RFC3694 [RFC3964].

RFC6343 [RFC6343] points out that if an operator provides well-managed servers and relays for 6to4, non-encapsulated IPv6 packets will pass through well- defined points (the native IPv6 interfaces of those servers and relays) at which security mechanisms may be applied. Client usage of 6to4 by default is now discouraged, and significant precautions are needed to avoid operational problems.

2.7.2.5. 6rd

While 6rd tunnels share the same encapsulation as 6to4 tunnels (Section 2.7.2.4), they are designed to be used within a single SP domain, in other words they are deployed in a more constrained environment than 6to4 tunnels and have little security issues except lack of confidentiality. The security considerations (Section 12) of RFC5969 [RFC5969] describes how to secure the 6rd tunnels.

IPsec [RFC4301] for the transported IPv6 traffic can be used if confidentiality is important.

2.7.2.6. 6PE and 6VPE

Organizations using MPLS in their core can also use 6PE [RFC4798] and 6VPE RFC4659 [RFC4659] to enable IPv6 access over MPLS. As 6PE and 6VPE are really similar to BGP/MPLS IP VPN described in RFC4364 [RFC4364], the security of these networks is also similar to the one described in RFC4381 [RFC4381]. It relies on:

- Address space, routing and traffic seperation with the help of VRF (only applicable to 6VPE);
- Hiding the IPv4 core, hence removing all attacks against P-routers;

- o Securing the routing protocol between CE and PE, in the case of 6PE and 6VPE, link-local addresses (see [RFC7404]) can be used and as these addresses cannot be reached from outside of the link, the security of 6PE and 6VPE is even higher than the IPv4 BGP/MPLS IP VPN.

2.7.2.7. DS-Lite

DS-lite is more a translation mechanism and is therefore analyzed further (Section 2.7.3.3) in this document.

2.7.2.8. Mapping of Address and Port

With the tunnel and encapsulation versions of mapping of Address and Port (MAP-E [RFC7597] and MAP-T [RFC7599]), the access network is purely an IPv6 network and MAP protocols are used to give IPv4 hosts on the subscriber network, access to IPv4 hosts on the Internet. The subscriber router does stateful operations in order to map all internal IPv4 addresses and layer-4 ports to the IPv4 address and the set of layer-4 ports received through MAP configuration process. The SP equipment always does stateless operations (either decapsulation or stateless translation). Therefore, as opposed to Section 2.7.3.3 there is no state-exhaustion DoS attack against the SP equipment because there is no state and there is no operation caused by a new layer-4 connection (no logging operation).

The SP MAP equipment MUST implement all the security considerations of [RFC7597]; notably, ensuring that the mapping of the IPv4 address and port are consistent with the configuration. As MAP has a predictable IPv4 address and port mapping, the audit logs are easier to manager.

2.7.3. Translation Mechanisms

Translation mechanisms between IPv4 and IPv6 networks are alternative coexistence strategies while networks transition to IPv6. While a framework is described in [RFC6144] the specific security considerations are documented in each individual mechanism. For the most part they specifically mention interference with IPsec or DNSSEC deployments, how to mitigate spoofed traffic and what some effective filtering strategies may be.

2.7.3.1. Carrier-Grade Nat (CGN)

Carrier-Grade NAT (CGN), also called NAT444 CGN or Large Scale NAT (LSN) or SP NAT is described in [RFC6264] and is utilized as an interim measure to prolong the use of IPv4 in a large service provider network until the provider can deploy and effective IPv6

solution. [RFC6598] requested a specific IANA allocated /10 IPv4 address block to be used as address space shared by all access networks using CGN. This has been allocated as 100.64.0.0/10.

Section 13 of [RFC6269] lists some specific security-related issues caused by large scale address sharing. The Security Considerations section of [RFC6598] also lists some specific mitigation techniques for potential misuse of shared address space.

RFC7422 [RFC7422] suggests the use of deterministic address mapping in order to reduce logging requirements for CGN. The idea is to have an algorithm mapping back and forth the internal subscriber to public ports.

2.7.3.2. NAT64/DNS64

Stateful NAT64 translation [RFC6146] allows IPv6-only clients to contact IPv4 servers using unicast UDP, TCP, or ICMP. It can be used in conjunction with DNS64 [RFC6147], a mechanism which synthesizes AAAA records from existing A records. There is also a stateless NAT64 [RFC6145] which is similar for the security aspects with the added benefit of being stateless, so, less prone to a state exhaustion attack.

The Security Consideration sections of [RFC6146] and [RFC6147] list the comprehensive issues. A specific issue with the use of NAT64 is that it will interfere with most IPsec deployments unless UDP encapsulation is used. DNS64 has an incidence on DNSSEC see section 3.1 of [RFC7050].

2.7.3.3. DS-Lite

Dual-Stack Lite (DS-Lite) [RFC6333] is a transition technique that enables a service provider to share IPv4 addresses among customers by combining two well-known technologies: IP in IP (IPv4-in-IPv6) and Network Address and Port Translation (NAPT).

Security considerations with respect to DS-Lite mainly revolve around logging data, preventing DoS attacks from rogue devices (as the AFTR function is stateful) and restricting service offered by the AFTR only to registered customers.

Section 11 of [RFC6333] describes important security issues associated with this technology.

2.8. General Device Hardening

There are many environments which rely too much on the network infrastructure to disallow malicious traffic to get access to critical hosts. In new IPv6 deployments it has been common to see IPv6 traffic enabled but none of the typical access control mechanisms enabled for IPv6 device access. With the possibility of network device configuration mistakes and the growth of IPv6 in the overall Internet it is important to ensure that all individual devices are hardened agains miscreant behavior.

The following guidelines should be used to ensure appropriate hardening of the host, be it an individual computer or router, firewall, load-balancer,server, etc device.

- Restrict access to the device to authorized individuals
- Monitor and audit access to the device
- Turn off any unused services on the end node
- Understand which IPv6 addresses are being used to source traffic and change defaults if necessary
- Use cryptographically protected protocols for device management if possible (SCP, SNMPv3, SSH, TLS, etc)
- Use host firewall capabilities to control traffic that gets processed by upper layer protocols
- Use virus scanners to detect malicious programs

3. Enterprises Specific Security Considerations

Enterprises generally have robust network security policies in place to protect existing IPv4 networks. These policies have been distilled from years of experiential knowledge of securing IPv4 networks. At the very least, it is recommended that enterprise networks have parity between their security policies for both protocol versions.

Security considerations in the enterprise can be broadly categorized into two sections - External and Internal.

3.1. External Security Considerations:

The external aspect deals with providing security at the edge or perimeter of the enterprise network where it meets the service providers network. This is commonly achieved by enforcing a security policy either by implementing dedicated firewalls with stateful packet inspection or a router with ACLs. A common default IPv4 policy on firewalls that could easily be ported to IPv6 is to allow all traffic outbound while only allowing specific traffic, such as established sessions, inbound (see also [RFC6092]). Here are a few more things that could enhance the default policy:

- Filter internal-use IPv6 addresses at the perimeter
- Discard packets from and to bogon and reserved space, see also [CYMRU]
- Accept certain ICMPv6 messages to allow proper operation of ND and PMTUD, see also [RFC4890]
- Filter specific extension headers by accepting only the required ones (white list approach) such as ESP, AH (not forgetting the required transport layers: ICMP, TCP, UDP, ...) , where possible at the edge and possibly inside the perimeter; see also [I-D.gont-opsec-ipv6-eh-filtering]
- Filter packets having an illegal IPv6 headers chain at the perimeter (and possible inside as well), see Section 2.2
- Filter unneeded services at the perimeter
- Implement anti-spoofing
- Implement appropriate rate-limiters and control-plane policers

3.2. Internal Security Considerations:

The internal aspect deals with providing security inside the perimeter of the network, including the end host. The most significant concerns here are related to Neighbor Discovery. At the network level, it is recommended that all security considerations discussed in Section 2.3 be reviewed carefully and the recommendations be considered in-depth as well.

As mentioned in Section 2.6.2, care must be taken when running automated IPv6-in-IP4 tunnels.

Hosts need to be hardened directly through security policy to protect against security threats. The host firewall default capabilities have to be clearly understood, especially 3rd party ones which can have different settings for IPv4 or IPv6 default permit/deny behavior. In some cases, 3rd party firewalls have no IPv6 support whereas the native firewall installed by default has it. General device hardening guidelines are provided in Section 2.8

It should also be noted that many hosts still use IPv4 for transport for things like RADIUS, TACACS+, SYSLOG, etc. This will require some extra level of due diligence on the part of the operator.

4. Service Providers Security Considerations

4.1. BGP

The threats and mitigation techniques are identical between IPv4 and IPv6. Broadly speaking they are:

- Authenticating the TCP session;
- TTL security (which becomes hop-limit security in IPv6);
- Prefix Filtering.

These are explained in more detail in section Section 2.5.

4.1.1. Remote Triggered Black Hole Filtering

RTBH [RFC5635] works identically in IPv4 and IPv6. IANA has allocated 100::/64 as discard prefix RFC6666 [RFC6666].

4.2. Transition Mechanism

SP will typically use transition mechanisms such as 6rd, 6PE, MAP, DS-Lite which have been analyzed in the transition Section 2.7.2 section.

4.3. Lawful Intercept

The Lawful Intercept requirements are similar for IPv6 and IPv4 architectures and will be subject to the laws enforced in varying geographic regions. The local issues with each jurisdiction can make this challenging and both corporate legal and privacy personnel should be involved in discussions pertaining to what information gets logged and what the logging retention policies will be.

The target of interception will usually be a residential subscriber (e.g. his/her PPP session or physical line or CPE MAC address). With the absence of NAT on the CPE, IPv6 has the provision to allow for intercepting the traffic from a single host (a /128 target) rather than the whole set of hosts of a subscriber (which could be a /48, a /60 or /64).

In contrast, in mobile environments, since the 3GPP specifications allocate a /64 per device, it may be sufficient to intercept traffic from the /64 rather than specific /128's (since each time the device powers up it gets a new IID).

A sample architecture which was written for informational purposes is found in [RFC3924].

5. Residential Users Security Considerations

The IETF Homenet working group is working on how IPv6 residential network should be done; this obviously includes operational security considerations; but, this is still work in progress.

Residential users have usually less experience and knowledge about security or networking. As most of the recent hosts, smartphones, tablets have all IPv6 enabled by default, IPv6 security is important for those users. Even with an IPv4-only ISP, those users can get IPv6 Internet access with the help of Teredo tunnels. Several peer-to-peer programs (notably Bittorrent) support IPv6 and those programs can initiate a Teredo tunnel through the IPv4 residential gateway, with the consequence of making the internal host reachable from any IPv6 host on the Internet. It is therefore recommended that all host security products (personal firewall, ...) are configured with a dual-stack security policy.

If the Residential Gateway has IPv6 connectivity, [RFC7084] (which obsoletes [RFC6204]) defines the requirements of an IPv6 CPE and does not take position on the debate of default IPv6 security policy as defined in [RFC6092]:

- outbound only: allowing all internally initiated connections and block all externally initiated ones, which is a common default security policy enforced by IPv4 Residential Gateway doing NAT-PT but it also breaks the end-to-end reachability promise of IPv6. [RFC6092] lists several recommendations to design such a CPE;

- open/transparent: allowing all internally and externally initiated connections, therefore restoring the end-to-end nature of the Internet for the IPv6 traffic but having a different security policy for IPv6 than for IPv4.

[RFC6092] REC-49 states that a choice must be given to the user to select one of those two policies.

There is also an alternate solution which has been deployed notably by Swisscom ([I-D.ietf-v6ops-balanced-ipv6-security]: open to all outbound and inbound connections at the exception of an handful of TCP and UDP ports known as vulnerable.

6. Further Reading

There are several documents that describe in more details the security of an IPv6 network; these documents are not written by the IETF but are listed here for your convenience:

1. Guidelines for the Secure Deployment of IPv6 [NIST]

2. North American IPv6 Task Force Technology Report - IPv6 Security Technology Paper [NAv6TF_Security]

3. IPv6 Security [IPv6_Security_Book]

7. Acknowledgements

The authors would like to thank the following people for their useful comments: Mikael Abrahamsson, Fred Baker, Brian Carpenter, Tim Chown, Markus deBruen, Fernando Gont, Jeffry Handal, Panos Kampanakis, Erik Kline, Jouni Korhonen, Mark Lentczner, Bob Sleigh,Tarko Tikan (by alphabetical order).

8. IANA Considerations

This memo includes no request to IANA.

9. Security Considerations

This memo attempts to give an overview of security considerations of operating an IPv6 network both in an IPv6-only network and in utilizing the most widely deployed IPv4/IPv6 coexistence strategies.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <http://www.rfc-editor.org/info/rfc2460>.

[RFC6104] Chown, T. and S. Venaas, "Rogue IPv6 Router Advertisement Problem Statement", RFC 6104, DOI 10.17487/RFC6104, February 2011, <http://www.rfc-editor.org/info/rfc6104>.

[RFC6105] Levy-Abegnoli, E., Van de Velde, G., Popoviciu, C., and J. Mohacsi, "IPv6 Router Advertisement Guard", RFC 6105, DOI 10.17487/RFC6105, February 2011, <http://www.rfc-editor.org/info/rfc6105>.

10.2. Informative References

[CYMRU] "Packet Filter and Route Filter Recommendation for IPv6 at xSP routers", <http://www.team-cymru.org/ReadingRoom/Templates/IPv6Routers/xsp-recommendations.html>.

[I-D.chakrabarti-nordmark-6man-efficient-nd] Chakrabarti, S., Nordmark, E., Thubert, P., and M. Wasserman, "IPv6 Neighbor Discovery Optimizations for Wired and Wireless Networks", draft-chakrabarti-nordmark-6man-efficient-nd-07 (work in progress), February 2015.

[I-D.gont-opsec-ipv6-eh-filtering] Gont, F., Will, W., and R. Bonica, "Recommendations on Filtering of IPv6 Packets Containing IPv6 Extension Headers", draft-gont-opsec-ipv6-eh-filtering-02 (work in progress), August 2014.

[I-D.ietf-6man-hbh-header-handling] Baker, F. and R. Bonica, "IPv6 Hop-by-Hop Options Extension Header", draft-ietf-6man-hbh-header-handling-03 (work in progress), March 2016.

[I-D.ietf-v6ops-balanced-ipv6-security] Gysi, M., Leclanche, G., Vyncke, E., and R. Anfinsen, "Balanced Security for IPv6 Residential CPE", draft-ietf-v6ops-balanced-ipv6-security-01 (work in progress), December 2013.

[I-D.kampanakis-6man-ipv6-eh-parsing] Kampanakis, P., "Implementation Guidelines for parsing IPv6 Extension Headers", draft-kampanakis-6man-ipv6-eh-parsing-01 (work in progress), August 2014.

[I-D.thubert-savi-ra-throttler]
Thubert, P., "Throttling RAs on constrained interfaces", draft-thubert-savi-ra-throttler-01 (work in progress), June 2012.

[IEEE-802.1X]
IEEE, , "IEEE Standard for Local and metropolitan area networks - Port-Based Network Access Control", IEEE Std 802.1X-2010, February 2010.

[IPv6_Security_Book]
Hogg, and Vyncke, "IPv6 Security", ISBN 1-58705-594-5, Publisher CiscoPress, December 2008.

[NAv6TF_Security]
Kaeo, , Green, , Bound, , and Pouffary, "North American IPv6 Task Force Technology Report - IPv6 Security Technology Paper", 2006, <http://www.ipv6forum.com/dl/white/NAv6TF_Security_Report.pdf>.

[NIST] Frankel, , Graveman, , Pearce, , and Rooks, "Guidelines for the Secure Deployment of IPv6", 2010, <http://csrc.nist.gov/publications/nistpubs/800-119/sp800-119.pdf>.

[RFC0826] Plummer, D., "Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware", STD 37, RFC 826, DOI 10.17487/RFC0826, November 1982, <http://www.rfc-editor.org/info/rfc826>.

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, DOI 10.17487/RFC2131, March 1997, <http://www.rfc-editor.org/info/rfc2131>.

[RFC2529] Carpenter, B. and C. Jung, "Transmission of IPv6 over IPv4 Domains without Explicit Tunnels", RFC 2529, DOI 10.17487/RFC2529, March 1999, <http://www.rfc-editor.org/info/rfc2529>.

[RFC2740] Coltun, R., Ferguson, D., and J. Moy, "OSPF for IPv6", RFC 2740, DOI 10.17487/RFC2740, December 1999, <http://www.rfc-editor.org/info/rfc2740>.

[RFC2784] Farinacci, D., Li, T., Hanks, S., Meyer, D., and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 2784, DOI 10.17487/RFC2784, March 2000, <http://www.rfc-editor.org/info/rfc2784>.

[RFC2827] Ferguson, P. and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", BCP 38, RFC 2827, DOI 10.17487/RFC2827, May 2000, <http://www.rfc-editor.org/info/rfc2827>.

[RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, DOI 10.17487/RFC2866, June 2000, <http://www.rfc-editor.org/info/rfc2866>.

[RFC2993] Hain, T., "Architectural Implications of NAT", RFC 2993, DOI 10.17487/RFC2993, November 2000, <http://www.rfc-editor.org/info/rfc2993>.

[RFC3056] Carpenter, B. and K. Moore, "Connection of IPv6 Domains via IPv4 Clouds", RFC 3056, DOI 10.17487/RFC3056, February 2001, <http://www.rfc-editor.org/info/rfc3056>.

[RFC3068] Huitema, C., "An Anycast Prefix for 6to4 Relay Routers", RFC 3068, DOI 10.17487/RFC3068, June 2001, <http://www.rfc-editor.org/info/rfc3068>.

[RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, DOI 10.17487/RFC3315, July 2003, <http://www.rfc-editor.org/info/rfc3315>.

[RFC3627] Savola, P., "Use of /127 Prefix Length Between Routers Considered Harmful", RFC 3627, DOI 10.17487/RFC3627, September 2003, <http://www.rfc-editor.org/info/rfc3627>.

[RFC3756] Nikander, P., Ed., Kempf, J., and E. Nordmark, "IPv6 Neighbor Discovery (ND) Trust Models and Threats", RFC 3756, DOI 10.17487/RFC3756, May 2004, <http://www.rfc-editor.org/info/rfc3756>.

[RFC3924] Baker, F., Foster, B., and C. Sharp, "Cisco Architecture for Lawful Intercept in IP Networks", RFC 3924, DOI 10.17487/RFC3924, October 2004, <http://www.rfc-editor.org/info/rfc3924>.

[RFC3964] Savola, P. and C. Patel, "Security Considerations for 6to4", RFC 3964, DOI 10.17487/RFC3964, December 2004, <http://www.rfc-editor.org/info/rfc3964>.

[RFC3971] Arkko, J., Ed., Kempf, J., Zill, B., and P. Nikander, "SEcure Neighbor Discovery (SEND)", RFC 3971, DOI 10.17487/RFC3971, March 2005, <http://www.rfc-editor.org/info/rfc3971>.

[RFC3972] Aura, T., "Cryptographically Generated Addresses (CGA)", RFC 3972, DOI 10.17487/RFC3972, March 2005, <http://www.rfc-editor.org/info/rfc3972>.

[RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, DOI 10.17487/RFC4193, October 2005, <http://www.rfc-editor.org/info/rfc4193>.

[RFC4293] Routhier, S., Ed., "Management Information Base for the Internet Protocol (IP)", RFC 4293, DOI 10.17487/RFC4293, April 2006, <http://www.rfc-editor.org/info/rfc4293>.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <http://www.rfc-editor.org/info/rfc4301>.

[RFC4302] Kent, S., "IP Authentication Header", RFC 4302, DOI 10.17487/RFC4302, December 2005, <http://www.rfc-editor.org/info/rfc4302>.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <http://www.rfc-editor.org/info/rfc4303>.

[RFC4364] Rosen, E. and Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4364, DOI 10.17487/RFC4364, February 2006, <http://www.rfc-editor.org/info/rfc4364>.

[RFC4380] Huitema, C., "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)", RFC 4380, DOI 10.17487/RFC4380, February 2006, <http://www.rfc-editor.org/info/rfc4380>.

[RFC4381] Behringer, M., "Analysis of the Security of BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4381, DOI 10.17487/RFC4381, February 2006, <http://www.rfc-editor.org/info/rfc4381>.

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, DOI 10.17487/RFC4443, March 2006, <http://www.rfc-editor.org/info/rfc4443>.

[RFC4552] Gupta, M. and N. Melam, "Authentication/Confidentiality for OSPFv3", RFC 4552, DOI 10.17487/RFC4552, June 2006, <http://www.rfc-editor.org/info/rfc4552>.

[RFC4659] De Clercq, J., Ooms, D., Carugi, M., and F. Le Faucheur, "BGP-MPLS IP Virtual Private Network (VPN) Extension for IPv6 VPN", RFC 4659, DOI 10.17487/RFC4659, September 2006, <http://www.rfc-editor.org/info/rfc4659>.

[RFC4798] De Clercq, J., Ooms, D., Prevost, S., and F. Le Faucheur, "Connecting IPv6 Islands over IPv4 MPLS Using IPv6 Provider Edge Routers (6PE)", RFC 4798, DOI 10.17487/RFC4798, February 2007, <http://www.rfc-editor.org/info/rfc4798>.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, DOI 10.17487/RFC4861, September 2007, <http://www.rfc-editor.org/info/rfc4861>.

[RFC4864] Van de Velde, G., Hain, T., Droms, R., Carpenter, B., and E. Klein, "Local Network Protection for IPv6", RFC 4864, DOI 10.17487/RFC4864, May 2007, <http://www.rfc-editor.org/info/rfc4864>.

[RFC4890] Davies, E. and J. Mohacsi, "Recommendations for Filtering ICMPv6 Messages in Firewalls", RFC 4890, DOI 10.17487/RFC4890, May 2007, <http://www.rfc-editor.org/info/rfc4890>.

[RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, DOI 10.17487/RFC4941, September 2007, <http://www.rfc-editor.org/info/rfc4941>.

[RFC4942] Davies, E., Krishnan, S., and P. Savola, "IPv6 Transition/ Co-existence Security Considerations", RFC 4942, DOI 10.17487/RFC4942, September 2007, <http://www.rfc-editor.org/info/rfc4942>.

[RFC5157] Chown, T., "IPv6 Implications for Network Scanning", RFC 5157, DOI 10.17487/RFC5157, March 2008, <http://www.rfc-editor.org/info/rfc5157>.

[RFC5214] Templin, F., Gleeson, T., and D. Thaler, "Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)", RFC 5214, DOI 10.17487/RFC5214, March 2008, <http://www.rfc-editor.org/info/rfc5214>.

[RFC5340] Coltun, R., Ferguson, D., Moy, J., and A. Lindem, "OSPF for IPv6", RFC 5340, DOI 10.17487/RFC5340, July 2008, <http://www.rfc-editor.org/info/rfc5340>.

[RFC5635] Kumari, W. and D. McPherson, "Remote Triggered Black Hole Filtering with Unicast Reverse Path Forwarding (uRPF)", RFC 5635, DOI 10.17487/RFC5635, August 2009, <http://www.rfc-editor.org/info/rfc5635>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <http://www.rfc-editor.org/info/rfc5952>.

[RFC5969] Townsley, W. and O. Troan, "IPv6 Rapid Deployment on IPv4 Infrastructures (6rd) -- Protocol Specification", RFC 5969, DOI 10.17487/RFC5969, August 2010, <http://www.rfc-editor.org/info/rfc5969>.

[RFC6092] Woodyatt, J., Ed., "Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for Providing Residential IPv6 Internet Service", RFC 6092, DOI 10.17487/RFC6092, January 2011, <http://www.rfc-editor.org/info/rfc6092>.

[RFC6144] Baker, F., Li, X., Bao, C., and K. Yin, "Framework for IPv4/IPv6 Translation", RFC 6144, DOI 10.17487/RFC6144, April 2011, <http://www.rfc-editor.org/info/rfc6144>.

[RFC6145] Li, X., Bao, C., and F. Baker, "IP/ICMP Translation Algorithm", RFC 6145, DOI 10.17487/RFC6145, April 2011, <http://www.rfc-editor.org/info/rfc6145>.

[RFC6146] Bagnulo, M., Matthews, P., and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", RFC 6146, DOI 10.17487/RFC6146, April 2011, <http://www.rfc-editor.org/info/rfc6146>.

[RFC6147] Bagnulo, M., Sullivan, A., Matthews, P., and I. van Beijnum, "DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers", RFC 6147, DOI 10.17487/RFC6147, April 2011, <http://www.rfc-editor.org/info/rfc6147>.

[RFC6164] Kohno, M., Nitzan, B., Bush, R., Matsuzaki, Y., Colitti, L., and T. Narten, "Using 127-Bit IPv6 Prefixes on Inter-Router Links", RFC 6164, DOI 10.17487/RFC6164, April 2011, <http://www.rfc-editor.org/info/rfc6164>.

[RFC6169] Krishnan, S., Thaler, D., and J. Hoagland, "Security Concerns with IP Tunneling", RFC 6169, DOI 10.17487/RFC6169, April 2011, <http://www.rfc-editor.org/info/rfc6169>.

[RFC6192] Dugal, D., Pignataro, C., and R. Dunn, "Protecting the Router Control Plane", RFC 6192, DOI 10.17487/RFC6192, March 2011, <http://www.rfc-editor.org/info/rfc6192>.

[RFC6204] Singh, H., Beebee, W., Donley, C., Stark, B., and O. Troan, Ed., "Basic Requirements for IPv6 Customer Edge Routers", RFC 6204, DOI 10.17487/RFC6204, April 2011, <http://www.rfc-editor.org/info/rfc6204>.

[RFC6221] Miles, D., Ed., Ooghe, S., Dec, W., Krishnan, S., and A. Kavanagh, "Lightweight DHCPv6 Relay Agent", RFC 6221, DOI 10.17487/RFC6221, May 2011, <http://www.rfc-editor.org/info/rfc6221>.

[RFC6264] Jiang, S., Guo, D., and B. Carpenter, "An Incremental Carrier-Grade NAT (CGN) for IPv6 Transition", RFC 6264, DOI 10.17487/RFC6264, June 2011, <http://www.rfc-editor.org/info/rfc6264>.

[RFC6269] Ford, M., Ed., Boucadair, M., Durand, A., Levis, P., and P. Roberts, "Issues with IP Address Sharing", RFC 6269, DOI 10.17487/RFC6269, June 2011, <http://www.rfc-editor.org/info/rfc6269>.

[RFC6296] Wasserman, M. and F. Baker, "IPv6-to-IPv6 Network Prefix Translation", RFC 6296, DOI 10.17487/RFC6296, June 2011, <http://www.rfc-editor.org/info/rfc6296>.

[RFC6302] Durand, A., Gashinsky, I., Lee, D., and S. Sheppard, "Logging Recommendations for Internet-Facing Servers", BCP 162, RFC 6302, DOI 10.17487/RFC6302, June 2011, <http://www.rfc-editor.org/info/rfc6302>.

[RFC6324] Nakibly, G. and F. Templin, "Routing Loop Attack Using IPv6 Automatic Tunnels: Problem Statement and Proposed Mitigations", RFC 6324, DOI 10.17487/RFC6324, August 2011, <http://www.rfc-editor.org/info/rfc6324>.

[RFC6333] Durand, A., Droms, R., Woodyatt, J., and Y. Lee, "Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion", RFC 6333, DOI 10.17487/RFC6333, August 2011, <http://www.rfc-editor.org/info/rfc6333>.

[RFC6343] Carpenter, B., "Advisory Guidelines for 6to4 Deployment", RFC 6343, DOI 10.17487/RFC6343, August 2011, <http://www.rfc-editor.org/info/rfc6343>.

[RFC6434] Jankiewicz, E., Loughney, J., and T. Narten, "IPv6 Node Requirements", RFC 6434, DOI 10.17487/RFC6434, December 2011, <http://www.rfc-editor.org/info/rfc6434>.

[RFC6459] Korhonen, J., Ed., Soininen, J., Patil, B., Savolainen, T., Bajko, G., and K. Iisakkila, "IPv6 in 3rd Generation Partnership Project (3GPP) Evolved Packet System (EPS)", RFC 6459, DOI 10.17487/RFC6459, January 2012, <http://www.rfc-editor.org/info/rfc6459>.

[RFC6506] Bhatia, M., Manral, V., and A. Lindem, "Supporting Authentication Trailer for OSPFv3", RFC 6506, DOI 10.17487/RFC6506, February 2012, <http://www.rfc-editor.org/info/rfc6506>.

[RFC6547] George, W., "RFC 3627 to Historic Status", RFC 6547, DOI 10.17487/RFC6547, February 2012, <http://www.rfc-editor.org/info/rfc6547>.

[RFC6583] Gashinsky, I., Jaeggli, J., and W. Kumari, "Operational Neighbor Discovery Problems", RFC 6583, DOI 10.17487/RFC6583, March 2012, <http://www.rfc-editor.org/info/rfc6583>.

[RFC6598] Weil, J., Kuarsingh, V., Donley, C., Liljenstolpe, C., and M. Azinger, "IANA-Reserved IPv4 Prefix for Shared Address Space", BCP 153, RFC 6598, DOI 10.17487/RFC6598, April 2012, <http://www.rfc-editor.org/info/rfc6598>.

[RFC6620] Nordmark, E., Bagnulo, M., and E. Levy-Abegnoli, "FCFS SAVI: First-Come, First-Served Source Address Validation Improvement for Locally Assigned IPv6 Addresses", RFC 6620, DOI 10.17487/RFC6620, May 2012, <http://www.rfc-editor.org/info/rfc6620>.

[RFC6666] Hilliard, N. and D. Freedman, "A Discard Prefix for IPv6", RFC 6666, DOI 10.17487/RFC6666, August 2012, <http://www.rfc-editor.org/info/rfc6666>.

[RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <http://www.rfc-editor.org/info/rfc6762>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <http://www.rfc-editor.org/info/rfc6763>.

[RFC6810] Bush, R. and R. Austein, "The Resource Public Key Infrastructure (RPKI) to Router Protocol", RFC 6810, DOI 10.17487/RFC6810, January 2013, <http://www.rfc-editor.org/info/rfc6810>.

[RFC6964] Templin, F., "Operational Guidance for IPv6 Deployment in IPv4 Sites Using the Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)", RFC 6964, DOI 10.17487/RFC6964, May 2013, <http://www.rfc-editor.org/info/rfc6964>.

[RFC6980] Gont, F., "Security Implications of IPv6 Fragmentation with IPv6 Neighbor Discovery", RFC 6980, DOI 10.17487/RFC6980, August 2013, <http://www.rfc-editor.org/info/rfc6980>.

[RFC7011] Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, DOI 10.17487/RFC7011, September 2013, <http://www.rfc-editor.org/info/rfc7011>.

[RFC7012] Claise, B., Ed. and B. Trammell, Ed., "Information Model for IP Flow Information Export (IPFIX)", RFC 7012, DOI 10.17487/RFC7012, September 2013, <http://www.rfc-editor.org/info/rfc7012>.

[RFC7039] Wu, J., Bi, J., Bagnulo, M., Baker, F., and C. Vogt, Ed., "Source Address Validation Improvement (SAVI) Framework", RFC 7039, DOI 10.17487/RFC7039, October 2013, <http://www.rfc-editor.org/info/rfc7039>.

[RFC7045] Carpenter, B. and S. Jiang, "Transmission and Processing of IPv6 Extension Headers", RFC 7045, DOI 10.17487/RFC7045, December 2013, <http://www.rfc-editor.org/info/rfc7045>.

[RFC7050] Savolainen, T., Korhonen, J., and D. Wing, "Discovery of the IPv6 Prefix Used for IPv6 Address Synthesis", RFC 7050, DOI 10.17487/RFC7050, November 2013, <http://www.rfc-editor.org/info/rfc7050>.

[RFC7084] Singh, H., Beebee, W., Donley, C., and B. Stark, "Basic Requirements for IPv6 Customer Edge Routers", RFC 7084, DOI 10.17487/RFC7084, November 2013, <http://www.rfc-editor.org/info/rfc7084>.

[RFC7112] Gont, F., Manral, V., and R. Bonica, "Implications of Oversized IPv6 Header Chains", RFC 7112, DOI 10.17487/RFC7112, January 2014, <http://www.rfc-editor.org/info/rfc7112>.

[RFC7113] Gont, F., "Implementation Advice for IPv6 Router Advertisement Guard (RA-Guard)", RFC 7113, DOI 10.17487/RFC7113, February 2014, <http://www.rfc-editor.org/info/rfc7113>.

[RFC7166] Bhatia, M., Manral, V., and A. Lindem, "Supporting Authentication Trailer for OSPFv3", RFC 7166, DOI 10.17487/RFC7166, March 2014, <http://www.rfc-editor.org/info/rfc7166>.

[RFC7217] Gont, F., "A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC)", RFC 7217, DOI 10.17487/RFC7217, April 2014, <http://www.rfc-editor.org/info/rfc7217>.

[RFC7381] Chittimaneni, K., Chown, T., Howard, L., Kuarsingh, V., Pouffary, Y., and E. Vyncke, "Enterprise IPv6 Deployment Guidelines", RFC 7381, DOI 10.17487/RFC7381, October 2014, <http://www.rfc-editor.org/info/rfc7381>.

[RFC7404] Behringer, M. and E. Vyncke, "Using Only Link-Local Addressing inside an IPv6 Network", RFC 7404, DOI 10.17487/RFC7404, November 2014, <http://www.rfc-editor.org/info/rfc7404>.

[RFC7422] Donley, C., Grundemann, C., Sarawat, V., Sundaresan, K., and O. Vautrin, "Deterministic Address Mapping to Reduce Logging in Carrier-Grade NAT Deployments", RFC 7422, DOI 10.17487/RFC7422, December 2014, <http://www.rfc-editor.org/info/rfc7422>.

[RFC7454] Durand, J., Pepelnjak, I., and G. Doering, "BGP Operations and Security", BCP 194, RFC 7454, DOI 10.17487/RFC7454, February 2015, <http://www.rfc-editor.org/info/rfc7454>.

[RFC7513] Bi, J., Wu, J., Yao, G., and F. Baker, "Source Address Validation Improvement (SAVI) Solution for DHCP", RFC 7513, DOI 10.17487/RFC7513, May 2015, <http://www.rfc-editor.org/info/rfc7513>.

[RFC7526] Troan, O. and B. Carpenter, Ed., "Deprecating the Anycast Prefix for 6to4 Relay Routers", BCP 196, RFC 7526, DOI 10.17487/RFC7526, May 2015, <http://www.rfc-editor.org/info/rfc7526>.

[RFC7597] Troan, O., Ed., Dec, W., Li, X., Bao, C., Matsushima, S., Murakami, T., and T. Taylor, Ed., "Mapping of Address and Port with Encapsulation (MAP-E)", RFC 7597, DOI 10.17487/RFC7597, July 2015, <http://www.rfc-editor.org/info/rfc7597>.

[RFC7599] Li, X., Bao, C., Dec, W., Ed., Troan, O., Matsushima, S., and T. Murakami, "Mapping of Address and Port using Translation (MAP-T)", RFC 7599, DOI 10.17487/RFC7599, July 2015, <http://www.rfc-editor.org/info/rfc7599>.

[RFC7610] Gont, F., Liu, W., and G. Van de Velde, "DHCPv6-Shield: Protecting against Rogue DHCPv6 Servers", BCP 199, RFC 7610, DOI 10.17487/RFC7610, August 2015, <http://www.rfc-editor.org/info/rfc7610>.

[RFC7707] Gont, F. and T. Chown, "Network Reconnaissance in IPv6 Networks", RFC 7707, DOI 10.17487/RFC7707, March 2016, <http://www.rfc-editor.org/info/rfc7707>.

[RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", RFC 7721, DOI 10.17487/RFC7721, March 2016, <http://www.rfc-editor.org/info/rfc7721>.

[RFC7872] Gont, F., Linkova, J., Chown, T., and W. Liu, "Observations on the Dropping of Packets with IPv6 Extension Headers in the Real World", RFC 7872, DOI 10.17487/RFC7872, June 2016, <http://www.rfc-editor.org/info/rfc7872>.

[SCANNING] "Mapping the Great Void - Smarter scanning for IPv6", <http://www.caida.org/workshops/isma/1202/slides/aims1202_rbarnes.pdf>.

Authors' Addresses

Kiran K. Chittimaneni
Dropbox Inc.
185 Berry Street, Suite 400
San Francisco, CA 94107
USA

Email: kk@dropbox.com

Merike Kaeo
Double Shot Security
3518 Fremont Ave N 363
Seattle 98103
USA

Phone: +12066696394
Email: merike@doubleshotsecurity.com

Eric Vyncke (editor)
Cisco
De Kleetlaan 6a
Diegem 1831
Belgium

Phone: +32 2 778 4677
Email: evyncke@cisco.com

OPSEC
Internet-Draft
Intended status: Informational
Expires: September 1, 2018

E. Vyncke, Ed.
Cisco
K. Chittimaneni
Dropbox Inc.
M. Kaeo
Double Shot Security
E. Rey
ERNW

February 28, 2018

Operational Security Considerations for IPv6 Networks
draft-ietf-opsec-v6-13

Abstract

Knowledge and experience on how to operate IPv4 securely is available: whether it is the Internet or an enterprise internal network. However, IPv6 presents some new security challenges. RFC 4942 describes the security issues in the protocol but network managers also need a more practical, operations-minded document to enumerate advantages and/or disadvantages of certain choices.

This document analyzes the operational security issues in several places of a network (enterprises, service providers and residential users) and proposes technical and procedural mitigations techniques.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 1, 2018.

Table of Contents

1. Introduction

Running an IPv6 network is new for most operators not only because they are not yet used to large scale IPv6 networks but also because there are subtle differences between IPv4 and IPv6 especially with respect to security. For example, all layer-2 interactions are now done using Neighbor Discovery Protocol [RFC4861] rather than using Address Resolution Protocol [RFC0826]. Also, there are subtle differences between NAT44 [RFC2993] and NPTv6 [RFC6296] which are explicitly pointed out in the latter's security considerations section.

IPv6 networks are deployed using a variety of techniques, each of which have their own specific security concerns.

This document complements [RFC4942] by listing all security issues when operating a network utilizing varying transition technologies and updating with ones that have been standardized since 2007. It also provides more recent operational deployment experiences where warranted.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when they appear in ALL CAPS. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

2. Generic Security Considerations

2.1. Addressing Architecture

IPv6 address allocations and overall architecture are an important part of securing IPv6. Initial designs, even if intended to be temporary, tend to last much longer than expected. Although initially IPv6 was thought to make renumbering easy, in practice, it may be extremely difficult to renumber without a good IP Addresses Management (IPAM) system.

Once an address allocation has been assigned, there should be some thought given to an overall address allocation plan. With the abundance of address space available, an address allocation may be structured around services along with geographic locations, which then can be a basis for more structured security policies to permit or deny services between geographic regions.

A common question is whether companies should use PI vs PA space [RFC7381], but from a security perspective there is little difference. However, one aspect to keep in mind is who has administrative ownership of the address space and who is technically responsible if/when there is a need to enforce restrictions on routability of the space due to malicious criminal activity. Using PA space exposes the organization to a renumbering of the complete network including security policies (based on ACL), audit system, ... in short a complex task which could lead to some temporary security risk if done for a large network and without automation; hence, for large network, PI space should be preferred even if it comes with additional complexities (for example BGP routing) and duties (adding a route6 object in the Regional Internet Registry database).

2.1.1. Statically Configured Addresses

When considering how to assign statically configured addresses it is necessary to take into consideration the effectiveness of perimeter security in a given environment. There is a trade-off between ease of operation (where some portions of the IPv6 address could be easily recognizable for operational debugging and troubleshooting) versus the risk of trivial scanning used for reconnaissance. [SCANNING]

shows that there are scientifically based mechanisms that make scanning for IPv6 reachable nodes more realizable than expected; see also [RFC7707]. The use of common multicast groups which are defined for important networked devices and the use of commonly repeated addresses could make it easy to figure out which devices are name servers, routers or other critical devices; even a simple traceroute will expose most of the routers on a path. There are many scanning techniques and more to come possible, hence, operators should never relly on the 'impossible to find because my address is random' paradigm.

While in some unmanaged environments obfuscating addresses could be considered a benefit; it is a better practice to ensure that perimeter rules are actively checked and enforced and that statically configured addresses follow some logical allocation scheme for ease of operation (as simplicity always helps security).

2.1.2. Use of ULAs

Unique Local Addresses (ULAs) RFC4193 [RFC4193] are intended for scenarios where IP addresses are not globally reachable, despite formally having global scope. They must not appear in the routing system outside the administrative domain where they are considered valid. Therefore, packets with ULA source and/or destination addresses MUST be filtered at the domain boundary.

ULAs are assigned within pseudo-random /48 prefixes created as specified in RFC4193 [RFC4193]. They could be useful for infrastructure hiding as described in RFC4864 [RFC4864].

ULAs may be used for internal communication, in conjunction with globally reachable unicast addresses (GUAs) for hosts that also require external connectivity through a firewall. For this reason, no form of address translation is required in conjunction with ULAs.

Using ULAs as described here might simplify the filtering rules needed at the domain boundary, by allowing a regime in which only hosts that require external connectivity possess a globally reachable address. However, this does not remove the need for careful design of the filtering rules. Routers with ULA on their interfaces may also leak their address to the Internet when generating ICMP messages or ICMP error messages can also include ULA address as they contain a copy of the offending packet.

2.1.3. Point-to-Point Links

RFC6164 [RFC6164] in section 5.1 documents the reasons why to use a /127 for inter-router point-to-point links; notably, a /127 prevents the ping-pong attack between routers not implementing correctly RFC4443 [RFC4443]. The previous recommendation of RFC3627 [RFC3627] has been obsoleted and marked Historic by RFC6547 [RFC6547]).

Some environments are also using link-local addressing for point-to-point links. While this practice could further reduce the attack surface against infrastructure devices, the operational disadvantages need also to be carefully considered; see also RFC7404 [RFC7404].

2.1.4. Temporary Addresses - Privacy Extensions for SLAAC

Normal stateless address autoconfiguration (SLAAC) relies on the automatically generated EUI-64 address, which together with the /64 prefix makes up the global unique IPv6 address. The EUI-64 address is generated from the MAC address. Randomly generating an interface ID, as described in [RFC4941], is part of SLAAC with so-called privacy extension addresses and used to address some privacy concerns. Privacy extension addresses a.k.a. temporary addresses may help to mitigate the correlation of activities of a node within the same network, and may also reduce the attack exposure window.

As privacy extension addresses could also be used to obfuscate some malevolent activities (whether on purpose or not), it is advised in scenarios where user attribution is important to rely on a layer-2 authentication mechanism such as IEEE 802.1X [IEEE-802.1X] with the appropriate RADIUS accounting (Section 2.6.1.6) or to disable SLAAC and rely only on DHCPv6. However, in scenarios where anonymity is a strong desire (protecting user privacy is more important than user attribution), privacy extension addresses should be used. When [RFC8064] is available, the stable temporary address are probably a good balance between privacy (among multiple networks) and security/ user attribution (within a network).

Using privacy extension addresses prevents the operator from building a priori host specific access control lists (ACLs). It must be noted that recent versions of Windows do not use the MAC address anymore to build the stable address but use a mechanism similar to the one described in [RFC7217], this also means that such an ACL cannot be configured based solely on the MAC address of the nodes, diminishing the value of such ACL. On the other hand, different VLANs are often used to segregate users, in this case ACL can rely on a /64 prefix per VLAN rather than a per host ACL entry.

The decision to utilize privacy extension addresses can come down to whether the network is managed versus unmanaged. In some environments full visibility into the network is required at all times which requires that all traffic be attributable to where it is sourced or where it is destined to within a specific network. This situation is dependent on what level of logging is performed. If logging considerations include utilizing accurate timestamps and logging a node's source ports [RFC6302] then there should always exist appropriate user attribution needed to get to the source of any malware originator or source of criminal activity.

Disabling SLAAC and privacy extensions addresses can be done for most OS and for non-hacker users by sending RA messages with a hint to get addresses via DHCPv6 by setting the M-bit but also disabling SLAAC by resetting all A-bits in all prefix information options. Hackers will find a way to bypass this mechanism if not enforced at the switch/ router level.

2.1.5. Privacy consideration of Addresses

The reader can learn more about privacy considerations for IPv6 addresses in RFC7721 [RFC7721].

2.1.6. DHCP/DNS Considerations

Many environments use DHCPv6 to allocate addresses to ensure audit-ability and traceability (but see Section 2.6.1.5). A main security concern is the ability to detect and counteract against rogue DHCP servers (Section 2.3.1).

While there are no fundamental differences with IPv4 and IPv6 security concerns about DNS, there are specific consideration in DNS64 RFC6147 [RFC6147] environments that need to be understood. Specifically the interactions and potential to interference with DNSSEC implementation need to be understood - these are pointed out in detail in Section 2.7.3.2.

2.1.7. Using a /64 per host

An interesting approach is using a /64 per host as proposed in RFC8273 [RFC8273]. This allows an easier user attribution (typically based on the host MAC address) as its /64 prefix is stable even if applications, containers within the host can change of IPv6 address within this /64.

2.2. Extension Headers

The extension headers are an important difference between IPv4 and IPv6. The packet structure does make a big difference. For instance, it's trivial to find (in IPv4-based packets) the upper layer protocol type and protocol header, while in IPv6 it actually isn't as the extension header chain must be parsed completely. The IANA has closed the existing empty "Next Header Types" registry to new entries and is redirecting its users to a new "IPv6 Extension Header Types" registry per RFC7045 [RFC7045].

They have also become a very controversial topic since forwarding nodes that discard packets containing extension headers are known to cause connectivity failures and deployment problems RFC7872 [RFC7872]. Understanding the role of varying extension headers is important and this section enumerates the ones that need careful consideration.

A clarification on how intermediate nodes should handle existing packets with extension headers and any extension headers that are defined in the future is found in RFC7045 [RFC7045]. The uniform TLV format to be used for defining future extension headers is described in RFC6564 [RFC6564].

It must also be noted that there is no indication in the packet whether the Next Protocol field points to an extension header or to a transport header. This may confuse some filtering rules.

There is work in progress at the IETF about filtering rules for those extension headers: [I-D.ietf-opsec-ipv6-eh-filtering] for transit routers.

2.2.1. Order and Repetition of Extension Headers

While RFC8200 [RFC8200] recommends the order and the maximum repetition of extension headers, there are still IPv6 implementations at the time of writing this document which support a non-recommended order of headers (such as ESP before routing) or an illegal repetition of headers (such as multiple routing headers). The same applies for options contained in the extension headers (see [I-D.kampanakis-6man-ipv6-eh-parsing]). In some cases, it has lead to nodes crashing when receiving or forwarding wrongly formated packets.

A firewall or any edge device able to enforce the recommended order and number of occurences of extension headers is recommended.

2.2.2. Hop-by-Hop Options Header

The hop-by-hop options header, when present in an IPv6 packet, forces all nodes in the path to inspect this header in the original IPv6 specification RFC2460 [RFC2460]. This was of course a large avenue for a denial of service as most if not all routers cannot process this kind of packets in hardware but have to 'punt' this packet for software processing. Section 4.3 of the current Internet Standard for IPv6, RFC8200 [RFC8200], is more sensible to this respect as the processing of hop-by-hop options header is optional.

2.2.3. Fragment Header

The fragment header is used by the source when it has to fragment packets. RFC7112 [RFC7112] and section 4.5 of RFC8200 [RFC8200] explain why it is important to:

- firewall and security devices should drop first fragment not containing an upper-layer header;
- destination nodes should discard first fragments not containing an upper-layer header.

Else, stateless filtering could be bypassed by an hostile party. RFC6980 [RFC6980] applies the same rule to NDP and the RA-guard function described in RFC6105 [RFC6105].

2.2.4. IP Security Extension Header

The IPsec [RFC4301] [RFC4301] extension headers (AH [RFC4302] and ESP [RFC4303]) are required if IPsec is to be utilized for network level security functionality.

2.3. Link-Layer Security

IPv6 relies heavily on the Neighbor Discovery protocol (NDP) RFC4861 [RFC4861] to perform a variety of link operations such as discovering other nodes on the link, resolving their link-layer addresses, and finding routers on the link. If not secured, NDP is vulnerable to various attacks such as router/neighbor message spoofing, redirect attacks, Duplicate Address Detection (DAD) DoS attacks, etc. many of these security threats to NDP have been documented in IPv6 ND Trust Models and Threats RFC3756 [RFC3756] and in RFC6583 [RFC6583].

2.3.1. Securing DHCP

Dynamic Host Configuration Protocol for IPv6 (DHCPv6), as detailed in RFC3315 [RFC3315], enables DHCP servers to pass configuration parameters such as IPv6 network addresses and other configuration information to IPv6 nodes. DHCP plays an important role in any large network by providing robust stateful configuration and autoregistration of DNS Host Names.

The two most common threats to DHCP clients come from malicious (a.k.a. rogue) or unintentionally misconfigured DHCP servers. A malicious DHCP server is established with the intent of providing incorrect configuration information to the client to cause a denial of service attack or mount a man in the middle attack. While unintentionall, a misconfigured DHCP server can have the same impact. Additional threats against DHCP are discussed in the security considerations section of RFC3315 [RFC3315]DHCP-shield.

RFC7610 [RFC7610], DHCPv6-Shield, specifies a mechanism for protecting connected DHCPv6 clients against rogue DHCPv6 servers. This mechanism is based on DHCPv6 packet-filtering at the layer-2 device; the administrator specifies the interfaces connected to DHCPv6 servers. Of course, extension headers could be leveraged to bypass DHCPv6-Shield unless RFC7112 [RFC7112] is enforced. Another way to secure DHCPv6 would be to use the secure DHCPv6 protocol which is currently work in progress per [I-D.ietf-dhc-sedhcpv6] , but, with no real deployment known by the authors of this document.

It is recommended to use DHCP-shield and to analyze the log generated by this security feature.

2.3.2. ND/RA Rate Limiting

Neighbor Discovery (ND) can be vulnerable to denial of service (DoS) attacks in which a router is forced to perform address resolution for a large number of unassigned addresses. Possible side effects of this attack preclude new devices from joining the network or even worse rendering the last hop router ineffective due to high CPU usage. Easy mitigative steps include rate limiting Neighbor Solicitations, restricting the amount of state reserved for unresolved solicitations, and clever cache/timer management.

RFC6583 [RFC6583] discusses the potential for DoS in detail and suggests implementation improvements and operational mitigation techniques that may be used to mitigate or alleviate the impact of such attacks. Here are some feasible mitigation options that can be employed by network operators today:

- Ingress filtering of unused addresses by ACL, route filtering, longer than /64 prefix; These require static configuration of the addresses.

- Tuning of NDP process (where supported).

- Using /127 on point-to-point link per RFC6164 [RFC6164].

Additionally, IPv6 ND uses multicast extensively for signaling messages on the local link to avoid broadcast messages for on-the-wire efficiency. However, this has some side effects on wifi networks, especially a negative impact on battery life of smartphones and other battery operated devices that are connected to such networks. The following drafts are actively discussing methods to rate limit RAs and other ND messages on wifi networks in order to address this issue:

- [I-D.thubert-savi-ra-throttler]

- [I-D.chakrabarti-nordmark-6man-efficient-nd]

2.3.3. ND/RA Filtering

Router Advertisement spoofing is a well-known attack vector and has been extensively documented. The presence of rogue RAs, either intentional or malicious, can cause partial or complete failure of operation of hosts on an IPv6 link. For example, a host can select an incorrect router address which can be used as a man-in-the-middle (MITM) attack or can assume wrong prefixes to be used for stateless address configuration (SLAAC). RFC6104 [RFC6104] summarizes the scenarios in which rogue RAs may be observed and presents a list of possible solutions to the problem. RFC6105 [RFC6105] (RA-Guard) describes a solution framework for the rogue RA problem where network segments are designed around switching devices that are capable of identifying invalid RAs and blocking them before the attack packets actually reach the target nodes.

However, several evasion techniques that circumvent the protection provided by RA-Guard have surfaced. A key challenge to this mitigation technique is introduced by IPv6 fragmentation. An attacker can conceal the attack by fragmenting his packets into multiple fragments such that the switching device that is responsible for blocking invalid RAs cannot find all the necessary information to perform packet filtering in the same packet. RFC7113 [RFC7113] describes such evasion techniques, and provides advice to RA-Guard implementers such that the aforementioned evasion vectors can be eliminated.

Given that the IPv6 Fragmentation Header can be leveraged to circumvent current implementations of RA-Guard, RFC6980 [RFC6980] updates RFC4861 [RFC4861] such that use of the IPv6 Fragmentation Header is forbidden in all Neighbor Discovery messages except "Certification Path Advertisement", thus allowing for simple and effective measures to counter Neighbor Discovery attacks.

The Source Address Validation Improvements (SAVI) working group has worked on other ways to mitigate the effects of such attacks. RFC7513 [RFC7513] would help in creating bindings between a DHCPv4 RFC2131 [RFC2131] /DHCPv6 RFC3315 [RFC3315] assigned source IP address and a binding anchor RFC7039 [RFC7039] on a SAVI device. Also, RFC6620 [RFC6620] describes how to glean similar bindings when DHCP is not used. The bindings can be used to filter packets generated on the local link with forged source IP address.

It is still recommended that RA-Guard be be employed as a first line of defense against common attack vectors including misconfigured hosts. The generated log should also be analyzed to act on violations.

2.3.4. 3GPP Link-Layer Security

The 3GPP link is a point-to-point like link that has no link-layer address. This implies there can only be an end host (the mobile hand-set) and the first-hop router (i.e., a GPRS Gateway Support Node (GGSN) or a Packet Gateway (PGW)) on that link. The GGSN/PGW never configures a non link-local address on the link using the advertised /64 prefix on it. The advertised prefix must not be used for on-link determination. There is no need for an address resolution on the 3GPP link, since there are no link-layer addresses. Furthermore, the GGSN/PGW assigns a prefix that is unique within each 3GPP link that uses IPv6 stateless address autoconfiguration. This avoids the necessity to perform DAD at the network level for every address built by the mobile host. The GGSN/PGW always provides an IID to the cellular host for the purpose of configuring the link-local address and ensures the uniqueness of the IID on the link (i.e., no collisions between its own link-local address and the mobile host’s one).

The 3GPP link model itself mitigates most of the known NDP-related Denial-of-Service attacks. In practice, the GGSN/PGW only needs to route all traffic to the mobile host that falls under the prefix assigned to it. As there is also a single host on the 3GPP link, there is no need to defend that IPv6 address.

See Section 5 of RFC6459 [RFC6459] for a more detailed discussion on the 3GPP link model, NDP on it and the address configuration detail.

2.3.5. SeND and CGA

SEcure Neighbor Discovery (SeND), as described in RFC3971 [RFC3971], is a mechanism that was designed to secure ND messages. This approach involves the use of new NDP options to carry public key based signatures. Cryptographically Generated Addresses (CGA), as described in RFC3972 [RFC3972], are used to ensure that the sender of a Neighbor Discovery message is the actual "owner" of the claimed IPv6 address. A new NDP option, the CGA option, was introduced and is used to carry the public key and associated parameters. Another NDP option, the RSA Signature option, is used to protect all messages relating to neighbor and Router discovery.

SeND protects against:

- Neighbor Solicitation/Advertisement Spoofing
- Neighbor Unreachability Detection Failure
- Duplicate Address Detection DoS Attack
- Router Solicitation and Advertisement Attacks
- Replay Attacks
- Neighbor Discovery DoS Attacks

SeND does NOT:

- Protect statically configured addresses
- Protect addresses configured using fixed identifiers (i.e. EUI-64)
- Provide confidentiality for NDP communications
- Compensate for an unsecured link - SEND does not require that the addresses on the link and Neighbor Advertisements correspond

However, at this time and after many years after their specifications, CGA and SeND do not have wide support from generic operating systems; hence, their usefulness is limited.

2.4. Control Plane Security

RFC6192 [RFC6192] defines the router control plane. This definition is repeated here for the reader’s convenience.

Modern router architecture design maintains a strict separation of forwarding and router control plane hardware and software. The router control plane supports routing and management functions. It is generally described as the router architecture hardware and software components for handling packets destined to the device itself as well as building and sending packets originated locally on the device. The forwarding plane is typically described as the router architecture hardware and software components responsible for receiving a packet on an incoming interface, performing a lookup to identify the packet's IP next hop and determine the best outgoing interface towards the destination, and forwarding the packet out through the appropriate outgoing interface.

While the forwarding plane is usually implemented in high-speed hardware, the control plane is implemented by a generic processor (named router processor RP) and cannot process packets at a high rate. Hence, this processor can be attacked by flooding its input queue with more packets than it can process. The control plane processor is then unable to process valid control packets and the router can lose OSPF or BGP adjacencies which can cause a severe network disruption.

The mitigation technique is:

- To drop non-legit control packet before they are queued to the RP (this can be done by a forwarding plane ACL) and

- To rate limit the remaining packets to a rate that the RP can sustain. Protocol specific protection should also be done (for example, a spoofed OSPFv3 packet could trigger the execution of the Dijkstra algorithm, therefore the number of Dijsktra execution should be also rate limited).

This section will consider several classes of control packets:

- Control protocols: routing protocols: such as OSPFv3, BGP and by extension Neighbor Discovery and ICMP

- Management protocols: SSH, SNMP, IPfix, etc

- Packet exceptions: which are normal data packets which requires a specific processing such as generating a packet-too-big ICMP message or having the hop-by-hop options header.

2.4.1. Control Protocols

This class includes OSPFv3, BGP, NDP, ICMP.

An ingress ACL to be applied on all the router interfaces SHOULD be configured such as:

- drop OSPFv3 (identified by Next-Header being 89) and RIPng (identified by UDP port 521) packets from a non link-local address
- allow BGP (identified by TCP port 179) packets from all BGP neighbors and drop the others
- allow all ICMP packets (transit and to the router interfaces)

Note: dropping OSPFv3 packets which are authenticated by IPsec could be impossible on some routers whose ACL are unable to parse the IPsec ESP or AH extension headers.

Rate limiting of the valid packets SHOULD be done. The exact configuration obviously depends on the power of the Route Processor.

2.4.2. Management Protocols

This class includes: SSH, SNMP, syslog, NTP, etc

An ingress ACL to be applied on all the router interfaces SHOULD be configured such as:

- Drop packets destined to the routers except those belonging to protocols which are used (for example, permit TCP 22 and drop all when only SSH is used);
- Drop packets where the source does not match the security policy, for example if SSH connections should only be originated from the NOC, then the ACL should permit TCP port 22 packets only from the NOC prefix.

Rate limiting of the valid packets SHOULD be done. The exact configuration obviously depends on the power of the Route Processor.

2.4.3. Packet Exceptions

This class covers multiple cases where a data plane packet is punted to the route processor because it requires specific processing:

- generation of an ICMP packet-too-big message when a data plane packet cannot be forwarded because it is too large;

- o generation of an ICMP hop-limit-expired message when a data plane packet cannot be forwarded because its hop-limit field has reached 0;

- o generation of an ICMP destination-unreachable message when a data plane packet cannot be forwarded for any reason;

- o processing of the hop-by-hop options header, new implementations follow section 4.3 of RFC8200 [RFC8200] where this processing is optional;

- o or more specific to some router implementation: an oversized extension header chain which cannot be processed by the hardware and force the packet to be punted to the generic router CPU.

On some routers, not everything can be done by the specialized data plane hardware which requires some packets to be 'punted' to the generic RP. This could include for example the processing of a long extension header chain in order to apply an ACL based on layer 4 information. RFC6980 [RFC6980] and more generally RFC7112 [RFC7112] highlights the security implications of oversized extension header chains on routers and updates RFC2460 [RFC2460] such that the first fragment of a packet is required to contain the entire IPv6 header chain.

An ingress ACL cannot help to mitigate a control plane attack using those packet exceptions. The only protection for the RP is to limit the rate of those packet exceptions forwarded to the RP, this means that some data plane packets will be dropped without any ICMP messages back to the source which may cause Path MTU holes.

In addition to limiting the rate of data plane packets queued to the RP, it is also important to limit the generation rate of ICMP messages both the save the RP but also to prevent an amplification attack using the router as a reflector.

2.5. Routing Security

Routing security in general can be broadly divided into three sections:

1. Authenticating neighbors/peers

2. Securing routing updates between peers

3. Route filtering

[RFC7454] covers these sections specifically for BGP in detail.

2.5.1. Authenticating Neighbors/Peers

A basic element of routing is the process of forming adjacencies, neighbor, or peering relationships with other routers. From a security perspective, it is very important to establish such relationships only with routers and/or administrative domains that one trusts. A traditional approach has been to use MD5 HMAC, which allows routers to authenticate each other prior to establishing a routing relationship.

OSPFv3 can rely on IPsec to fulfill the authentication function. However, it should be noted that IPsec support is not standard on all routing platforms. In some cases, this requires specialized hardware that offloads crypto over to dedicated ASICs or enhanced software images (both of which often come with added financial cost) to provide such functionality. An added detail is to determine whether OSPFv3 IPsec implementations use AH or ESP-Null for integrity protection. In early implementations all OSPFv3 IPsec configurations relied on AH since the details weren't specified in RFC5340 [RFC5340] or RFC2740 [RFC2740] that was obsoleted by the former. However, the document which specifically describes how IPsec should be implemented for OSPFv3 RFC4552 [RFC4552] specifically states that ESP-Null MUST and AH MAY be implemented since it follows the overall IPsec standards wordings. OSPFv3 can also use normal ESP to encrypt the OSPFv3 payload to hide the routing information.

RFC7166 [RFC7166] (which obsoletes RFC6506 [RFC6506] changes OSPFv3's reliance on IPsec by appending an authentication trailer to the end of the OSPFv3 packets; it does not specifically authenticate the specific originator of an OSPFv3 packet; rather, it allows a router to confirm that the packet has indeed been issued by a router that had access to the shared authentication key.

With all authentication mechanisms, operators should confirm that implementations can support re-keying mechanisms that do not cause outages. There have been instances where any re-keying cause outages and therefore the tradeoff between utilizing this functionality needs to be weighed against the protection it provides.

2.5.2. Securing Routing Updates Between Peers

IPv6 initially mandated the provisioning of IPsec capability in all nodes. However, in the updated IPv6 Nodes Requirement standard RFC6434 [RFC6434] is now a 'SHOULD' and no more a 'MUST' implement. Theoretically it is possible, and recommended, that communication between two IPv6 nodes, including routers exchanging routing information be encrypted using IPsec. In practice however, deploying

IPsec is not always feasible given hardware and software limitations of various platforms deployed, as described in the earlier section.

2.5.3. Route Filtering

Route filtering policies will be different depending on whether they pertain to edge route filtering vs internal route filtering. At a minimum, IPv6 routing policy as it pertains to routing between different administrative domains should aim to maintain parity with IPv4 from a policy perspective e.g.,

- Filter internal-use, non-globally routable IPv6 addresses at the perimeter
- Discard packets from and to bogon and reserved space (see RFC8190 [RFC8190])
- Configure ingress route filters that validate route origin, prefix ownership, etc. through the use of various routing databases, e.g., RADB. There is additional work being done in this area to formally validate the origin ASs of BGP announcements in RFC6810 [RFC6810]

Some good recommendations for filtering can be found from Team CYMRU at [CYMRU].

2.6. Logging/Monitoring

In order to perform forensic research in case of any security incident or to detect abnormal behaviors, network operators should log multiple pieces of information.

This includes:

- logs of all applications when available (for example web servers);
- use of IP Flow Information Export [RFC7011] also known as IPfix;
- use of SNMP MIB [RFC4293];
- use of the Neighbor cache;
- use of stateful DHCPv6 [RFC3315] lease cache, especially when a relay agent [RFC6221] in layer-2 switches is used;
- use of RADIUS [RFC2866] for accounting records.

Please note that there are privacy issues related to how those logs are collected, kept and safely discarded. Operators are urged to check their country legislation.

All those pieces of information will be used for:

- forensic (Section 2.6.2.1) investigations such as who did what and when?
- correlation (Section 2.6.2.3): which IP addresses were used by a specific node (assuming the use of privacy extensions addresses [RFC4941])
- inventory (Section 2.6.2.2): which IPv6 nodes are on my network?
- abnormal behavior detection (Section 2.6.2.4): unusual traffic patterns are often the symptoms of a abnormal behavior which is in turn a potential attack (denial of services, network scan, a node being part of a botnet, ...)

2.6.1. Data Sources

This section lists the most important sources of data that are useful for operational security.

2.6.1.1. Logs of Applications

Those logs are usually text files where the remote IPv6 address is stored in all characters (not binary). This can complicate the processing since one IPv6 address, 2001:db8::1 can be written in multiple ways such as:

- 2001:DB8::1 (in uppercase)
- 2001:0db8::0001 (with leading 0)
- and many other ways including the reverse DNS mapping into a FQDN (which should not be trusted).

RFC 5952 [RFC5952] explains this problem in detail and recommends the use of a single canonical format (in short use lower case and suppress leading 0). This memo recommends the use of canonical format [RFC5952] for IPv6 addresses in all possible cases. If the existing application cannot log under the canonical format, then this memo recommends the use an external program in order to canonicalize all IPv6 addresses.

For example, this perl script can be used:

```
#!/usr/bin/perl -w
use strict ;
use warnings ;
use Socket ;
use Socket6 ;

my (@words, $word, $binary_address) ;

## go through the file one line at a time
while (my $line = <STDIN>) {
  chomp $line;
  foreach my $word (split /[\s+]/, $line) {
    $binary_address = inet_pton AF_INET6, $word ;
    if ($binary_address) {
      print inet_ntop AF_INET6, $binary_address ;
    } else {
      print $word ;
    }
    print " " ;
  }
  print "\n" ;
}
```

2.6.1.2. IP Flow Information Export by IPv6 Routers

IPfix [RFC7012] defines some data elements that are useful for security:

- in section 5.4 (IP Header fields): nextHeaderIPv6 and sourceIPv6Address;
- in section 5.6 (Sub-IP fields) sourceMacAddress.

Moreover, IPfix is very efficient in terms of data handling and transport. It can also aggregate flows by a key such as sourceMacAddress in order to have aggregated data associated with a specific sourceMacAddress. This memo recommends the use of IPfix and aggregation on nextHeaderIPv6, sourceIPv6Address and sourceMacAddress.

2.6.1.3. SNMP MIB by IPv6 Routers

RFC 4293 [RFC4293] defines a Management Information Base (MIB) for the two address families of IP. This memo recommends the use of:

- ipIfStatsTable table which collects traffic counters per interface;

- o ipNetToPhysicalTable table which is the content of the Neighbor cache, i.e. the mapping between IPv6 and data-link layer addresses.

2.6.1.4. Neighbor Cache of IPv6 Routers

The neighbor cache of routers contains all mappings between IPv6 addresses and data-link layer addresses. It is usually available by two means:

- o the SNMP MIB (Section 2.6.1.3) as explained above;
- o using NETCONF RFC6241 [RFC6241] to collect the state of the neighbor cache;
- o also by connecting over a secure management channel (such as SSH) and explicitely requesting a neighbor cache dump via the Command Line Interface or any other monitoring mechanism.

The neighbor cache is highly dynamic as mappings are added when a new IPv6 address appears on the network (could be quite often with privacy extension addresses [RFC4941] or when they are removed when the state goes from UNREACH to removed (the default time for a removal per Neighbor Unreachability Detection [RFC4861] algorithm is 38 seconds for a typical host such as Windows 7). This means that the content of the neighbor cache must periodically be fetched every 30 seconds (to be on the safe side) and stored for later use.

This is an important source of information because it is trivial (on a switch not using the SAVI [RFC7039] algorithm) to defeat the mapping between data-link layer address and IPv6 address. Let us rephrase the previous statement: having access to the current and past content of the neighbor cache has a paramount value for forensic and audit trail.

Using the approach of one /64 per host (Section 2.1.7) replaces the neighbor cache dumps by a mere caching of the allocated /64 prefix when combined with strict enforcement rule on the router and switches to prevent IPv6 spoofing.

2.6.1.5. Stateful DHCPv6 Lease

In some networks, IPv6 addresses are managed by stateful DHCPv6 server [RFC3315] that leases IPv6 addresses to clients. It is indeed quite similar to DHCP for IPv4 so it can be tempting to use this DHCP lease file to discover the mapping between IPv6 addresses and data-link layer addresses as it was usually done in the IPv4 era.

It is not so easy in the IPv6 era because not all nodes will use DHCPv6 (there are nodes which can only do stateless autoconfiguration) but also because DHCPv6 clients are identified not by their hardware-client address as in IPv4 but by a DHCP Unique ID (DUID) which can have several formats: some being the data-link layer address, some being data-link layer address prepended with time information or even an opaque number which is useless for operation security. Moreover, when the DUID is based on the data-link address, this address can be of any interface of the client (such as the wireless interface while the client actually uses its wired interface to connect to the network).

If a lightweight DHCP relay agent [RFC6221] is used in the layer-2 switches, then the DHCP server also receives the Interface-ID information which could be save in order to identifity the interface of the switches which received a specific leased IPv6 address. Also, if a 'normal' (not lightweight) relay agent adds the data-link layer address in the option for Relay Agent Remote-ID [RFC4649] or RFC6939 [RFC6939], then the DHCPv6 server can keep track of the data-link and leased IPv6 addresses.

In short, the DHCPv6 lease file is less interesting than in the IPv4 era. DHCPv6 servers that keep the relayed data-link layer address in addition to the DUID in the lease file do not suffer from this limitation.

The mapping between data-link layer address and the IPv6 address can be secured by using switches implementing the SAVI [RFC7513] algorithms. Of course, this also requires that data-link layer address is protected by using layer-2 mechanism such as [IEEE-802.1X].

2.6.1.6. RADIUS Accounting Log

For interfaces where the user is authenticated via a RADIUS [RFC2866] server, and if RADIUS accounting is enabled, then the RADIUS server receives accounting Acct-Status-Type records at the start and at the end of the connection which include all IPv6 (and IPv4) addresses used by the user. This technique can be used notably for Wi-Fi networks with Wi-Fi Protected Address (WPA) or any other IEEE 802.1X [IEEE-802.1X]wired interface on an Ethernet switch.

2.6.1.7. Other Data Sources

There are other data sources that must be kept exactly as in the IPv4 network:

- o historical mapping of IPv6 addresses to users of remote access VPN;
- o historical mapping of MAC address to switch interface in a wired network.

2.6.2. Use of Collected Data

This section leverages the data collected as described before (Section 2.6.1) in order to achieve several security benefits.

2.6.2.1. Forensic

The forensic use case is when the network operator must locate an IPv6 address that was present in the network at a certain time or is still currently in the network.

The source of information can be, in decreasing order, neighbor cache, DHCP lease file. Then, the procedure is:

1. based on the IPv6 prefix of the IPv6 address find the router(s) which are used to reach this prefix (assuming that anti-spoofing mechanisms are used);
2. based on this limited set of routers, on the incident time and on IPv6 address to retrieve the data-link address from live neighbor cache, from the historical data of the neighbor cache,
3. based on the incident time and on the IPv6 address, retrieve the data-link address from the DHCP lease file (Section 2.6.1.5);
4. based on the data-link layer address, look-up on which switch interface was this data-link layer address. In the case of wireless LAN, the RADIUS log should have the mapping between user identification and the MAC address. If a Configuration Management Data Base (CMDB) is used, the mapping between the data-link layer address and a switch port.

At the end of the process, the interface the host originating malicious activity or the username which was abused for malicious activity has been determined.

2.6.2.2. Inventory

RFC 7707 [RFC7707] (which obsoletes RFC 5157 [RFC5157]) is about the difficulties for an attacker to scan an IPv6 network due to the vast number of IPv6 addresses per link (and why in some case it can stil be done). While the huge addressing space can sometime be perceived

as a 'protection', it also make the inventory task difficult in an IPv6 network while it was trivial to do in an IPv4 network (a simple enumeration of all IPv4 addresses, followed by a ping and a TCP/UDP port scan). Getting an inventory of all connected devices is of prime importance for a secure operation of a network.

There are many ways to do an inventory of an IPv6 network.

The first technique is to use the IPfix information and extract the list of all IPv6 source addresses to find all IPv6 nodes that sent packets through a router. This is very efficient but alas will not discover silent node that never transmitted such packets... Also, it must be noted that link-local addresses will never be discovered by this means.

The second way is again to use the collected neighbor cache content to find all IPv6 addresses in the cache. This process will also discover all link-local addresses. See Section 2.6.1.4.

Another way works only for local network, it consists in sending a ICMP ECHO_REQUEST to the link-local multicast address ff02::1 which is all IPv6 nodes on the network. All nodes should reply to this ECHO_REQUEST per [RFC4443].

Other techniques involve obtaining data from DNS, parsing log files, leveraging service discovery such as mDNS RFC6761 [RFC6762] and RFC6763 [RFC6763].

Enumerating DNS zones, especially looking at reverse DNS records and CNAMES, is another common method employed by various tools. As already metioned in RFC7707 [RFC7707], this allows an attacker to prune the IPv6 reverse DNS tree, and hence enumerate it in a feasible time. Furthermore, authoritative servers that allow zone transfers (AXFR) may be a further information source.

2.6.2.3. Correlation

In an IPv4 network, it is easy to correlate multiple logs, for example to find events related to a specific IPv4 address. A simple Unix grep command was enough to scan through multiple text-based files and extract all lines relevant to a specific IPv4 address.

In an IPv6 network, this is slightly more difficult because different character strings can express the same IPv6 address. Therefore, the simple Unix grep command cannot be used. Moreover, an IPv6 node can have multiple IPv6 addresses.

In order to do correlation in IPv6-related logs, it is advised to have all logs with canonical IPv6 addresses. Then, the neighbor cache current (or historical) data set must be searched to find the data-link layer address of the IPv6 address. Then, the current and historical neighbor cache data sets must be searched for all IPv6 addresses associated to this data-link layer address: this is the search set. The last step is to search in all log files (containing only IPv6 address in canonical format) for any IPv6 addresses in the search set.

2.6.2.4. Abnormal Behavior Detection

Abnormal behaviors (such as network scanning, spamming, denial of service) can be detected in the same way as in an IPv4 network

- sudden increase of traffic detected by interface counter (SNMP) or by aggregated traffic from IPfix records [RFC7012];
- change of traffic pattern (number of connection per second, number of connection per host...) with the use of IPfix [RFC7012]

2.6.3. Summary

While some data sources (IPfix, MIB, switch CAM tables, logs, ...) used in IPv4 are also used in the secure operation of an IPv6 network, the DHCPv6 lease file is less reliable and the neighbor cache is of prime importance.

The fact that there are multiple ways to express in a character string the same IPv6 address renders the use of filters mandatory when correlation must be done.

2.7. Transition/Coexistence Technologies

As it is expected that network will not run in a pure IPv6-only way, the different transition mechanisms must be deployed and operated in a secure way. This section proposes operational guidelines for the most known and deployed transition techniques.

2.7.1. Dual Stack

Dual stack is often the first deployment choice for most existing network operators without an MPLS core where 6PE RFC4798 [RFC4798] is quite common. Dual stacking the network offers some advantages over other transition mechanisms. Firstly, the impact on existing IPv4 operations is reduced. Secondly, in the absence of tunnels or address translation, the IPv4 and IPv6 traffics are native (easier to observe) and should have the same network processing (path, quality

of service, ...). Dual stack allows you to gradually turn IPv4 operations down when your IPv6 network is ready for prime time. On the other hand, the operators have to manage two networks with the added complexities.

From an operational security perspective, this now means that you have twice the exposure. One needs to think about protecting both protocols now. At a minimum, the IPv6 portion of a dual stacked network should maintain parity with IPv4 from a security policy point of view. Typically, the following methods are employed to protect IPv4 networks at the edge:

- ACLs to permit or deny traffic
- Firewalls with stateful packet inspection

It is recommended that these ACLs and/or firewalls be additionally configured to protect IPv6 communications. Also, given the end-to-end connectivity that IPv6 provides, it is also recommended that hosts be fortified against threats. General device hardening guidelines are provided in Section 2.8

For many years, all host operating systems have IPv6 enabled by default, so, it is possible even in an 'IPv4-only' network to attack layer-2 adjacent victims over IPv6 link-local address or over a global IPv6 address is rogue RA or rogue DHCPv6 addresses are provided by an attacker.

2.7.2. Transition Mechanisms

There are many tunnels used for specific use cases. Except when protected by IPsec [RFC4301], all those tunnels have a couple of security issues (most of them being described in RFC 6169 [RFC6169]);

- tunnel injection: a malevolent person knowing a few pieces of information (for example the tunnel endpoints and the used protocol) can forge a packet which looks like a legit and valid encapsulated packet that will gladly be accepted by the destination tunnel endpoint, this is a specific case of spoofing;
- traffic interception: no confidentiality is provided by the tunnel protocols (without the use of IPsec), therefore anybody on the tunnel path can intercept the traffic and have access to the clear-text IPv6 packet; combined with the absence of authentication, a man in the middle attack can also be mounted;

- o service theft: as there is no authorization, even a non authorized user can use a tunnel relay for free (this is a specific case of tunnel injection);

- o reflection attack: another specific use case of tunnel injection where the attacker injects packets with an IPv4 destination address not matching the IPv6 address causing the first tunnel endpoint to re-encapsulate the packet to the destination... Hence, the final IPv4 destination will not see the original IPv4 address but only one IPv4 address of the relay router.

- o bypassing security policy: if a firewall or an IPS is on the path of the tunnel, then it will probably neither inspect not detect an malevolent IPv6 traffic contained in the tunnel.

To mitigate the bypassing of security policies, it is recomended to block all default configuration tunnels by denying all IPv4 traffic matching:

- o IP protocol 41: this will block ISATAP (Section 2.7.2.2), 6to4 (Section 2.7.2.7), 6rd (Section 2.7.2.3) as well as 6in4 (Section 2.7.2.1) tunnels;

- o IP protocol 47: this will block GRE (Section 2.7.2.1) tunnels;

- o UDP protocol 3544: this will block the default encapsulation of Teredo (Section 2.7.2.6) tunnels.

Ingress filtering [RFC2827] should also be applied on all tunnel endpoints if applicable to prevent IPv6 address spoofing.

As several of the tunnel techniques share the same encapsulation (i.e. IPv4 protocol 41) and embed the IPv4 address in the IPv6 address, there are a set of well-known looping attacks described in RFC 6324 [RFC6324], this RFC also proposes mitigation techniques.

2.7.2.1. Site-to-Site Static Tunnels

Site-to-site static tunnels are described in RFC 2529 [RFC2529] and in GRE [RFC2784]. As the IPv4 endpoints are statically configured and are not dynamic they are slightly more secure (bi-directional service theft is mostly impossible) but traffic interception and tunnel injection are still possible. Therefore, the use of IPsec [RFC4301] in transport mode and protecting the encapsulated IPv4 packets is recommended for those tunnels. Alternatively, IPsec in tunnel mode can be used to transport IPv6 traffic over a non-trusted IPv4 network.

2.7.2.2. ISATAP

ISATAP tunnels [RFC5214] are mainly used within a single administrative domain and to connect a single IPv6 host to the IPv6 network. This means that endpoints and and the tunnel endpoint are usually managed by a single entity; therefore, audit trail and strict anti-spoofing are usually possible and this raises the overall security.

Special care must be taken to avoid looping attack by implementing the measures of RFC 6324 [RFC6324] and of RFC6964 [RFC6964].

IPsec [RFC4301] in transport or tunnel mode can be used to secure the IPv4 ISATAP traffic to provide IPv6 traffic confidentiality and prevent service theft.

2.7.2.3. 6rd

While 6rd tunnels share the same encapsulation as 6to4 tunnels (Section 2.7.2.7), they are designed to be used within a single SP domain, in other words they are deployed in a more constrained environment than 6to4 tunnels and have little security issues except lack of confidentiality. The security considerations (Section 12) of RFC5969 [RFC5969] describes how to secure the 6rd tunnels.

IPsec [RFC4301] for the transported IPv6 traffic can be used if confidentiality is important.

2.7.2.4. 6PE and 6VPE

Organizations using MPLS in their core can also use 6PE [RFC4798] and 6VPE RFC4659 [RFC4659] to enable IPv6 access over MPLS. As 6PE and 6VPE are really similar to BGP/MPLS IP VPN described in RFC4364 [RFC4364], the security of these networks is also similar to the one described in RFC4381 [RFC4381]. It relies on:

- Address space, routing and traffic seperation with the help of VRF (only applicable to 6VPE);
- Hiding the IPv4 core, hence removing all attacks against P-routers;
- Securing the routing protocol between CE and PE, in the case of 6PE and 6VPE, link-local addresses (see [RFC7404]) can be used and as these addresses cannot be reached from outside of the link, the security of 6PE and 6VPE is even higher than the IPv4 BGP/MPLS IP VPN.

2.7.2.5. DS-Lite

DS-lite is more a translation mechanism and is therefore analyzed further (Section 2.7.3.3) in this document.

2.7.2.6. Teredo

Teredo tunnels [RFC4380] are mainly used in a residential environment because that can easily traverse an IPv4 NAT-PT device thanks to its UDP encapsulation and they connect a single host to the IPv6 Internet. Teredo shares the same issues as other tunnels: no authentication, no confidentiality, possible spoofing and reflection attacks.

IPsec [RFC4301] for the transported IPv6 traffic is recommended.

The biggest threat to Teredo is probably for IPv4-only network as Teredo has been designed to easily traverse IPV4 NAT-PT devices which are quite often co-located with a stateful firewall. Therefore, if the stateful IPv4 firewall allows unrestricted UDP outbound and accept the return UDP traffic, then Teredo actually punches a hole in this firewall for all IPv6 traffic to the Internet and from the Internet. While host policies can be deployed to block Teredo in an IPv4-only network in order to avoid this firewall bypass, it would be more efficient to block all UDP outbound traffic at the IPv4 firewall if deemed possible (of course, at least port 53 should be left open for DNS traffic).

Teredo is now mostly never used and it is no more automated in most environment, so, it is less of a threat.

2.7.2.7. 6to4

6to4 tunnels [RFC3056] require a public routable IPv4 address in order to work correctly. They can be used to provide either one IPv6 host connectivity to the IPv6 Internet or multiple IPv6 networks connectivity to the IPv6 Internet. The 6to4 relay is usually the anycast address defined in RFC3068 [RFC3068] which has been deprecated by RFC7526 [RFC7526], and is no more used by recent Operating Systems. Some security considerations are explained in RFC3694 [RFC3964].

RFC6343 [RFC6343] points out that if an operator provides well-managed servers and relays for 6to4, non-encapsulated IPv6 packets will pass through well- defined points (the native IPv6 interfaces of those servers and relays) at which security mechanisms may be applied. Client usage of 6to4 by default is now discouraged, and significant precautions are needed to avoid operational problems.

2.7.2.8. Mapping of Address and Port

With the encapsulation and translation versions of mapping of Address and Port (MAP-E [RFC7597] and MAP-T [RFC7599]), the access network is purely an IPv6 network and MAP protocols are used to give IPv4 hosts on the subscriber network, access to IPv4 hosts on the Internet. The subscriber router does stateful operations in order to map all internal IPv4 addresses and layer-4 ports to the IPv4 address and the set of layer-4 ports received through MAP configuration process. The SP equipment always does stateless operations (either decapsulation or stateless translation). Therefore, as opposed to Section 2.7.3.3 there is no state-exhaustion DoS attack against the SP equipment because there is no state and there is no operation caused by a new layer-4 connection (no logging operation).

The SP MAP equipment MUST implement all the security considerations of [RFC7597]; notably, ensuring that the mapping of the IPv4 address and port are consistent with the configuration. As MAP has a predictable IPv4 address and port mapping, the audit logs are easier to manager.

2.7.3. Translation Mechanisms

Translation mechanisms between IPv4 and IPv6 networks are alternative coexistence strategies while networks transition to IPv6. While a framework is described in [RFC6144] the specific security considerations are documented in each individual mechanism. For the most part they specifically mention interference with IPsec or DNSSEC deployments, how to mitigate spoofed traffic and what some effective filtering strategies may be.

2.7.3.1. Carrier-Grade Nat (CGN)

Carrier-Grade NAT (CGN), also called NAT444 CGN or Large Scale NAT (LSN) or SP NAT is described in [RFC6264] and is utilized as an interim measure to prolong the use of IPv4 in a large service provider network until the provider can deploy and effective IPv6 solution. [RFC6598] requested a specific IANA allocated /10 IPv4 address block to be used as address space shared by all access networks using CGN. This has been allocated as 100.64.0.0/10.

Section 13 of [RFC6269] lists some specific security-related issues caused by large scale address sharing. The Security Considerations section of [RFC6598] also lists some specific mitigation techniques for potential misuse of shared address space. Some Law Enforcement Agencies have identified CGN as impeding their cyber-crime investigations (for example Europol press release on CGN [europol-cgn]).

RFC7422 [RFC7422] suggests the use of deterministic address mapping in order to reduce logging requirements for CGN. The idea is to have an algorithm mapping back and forth the internal subscriber to public ports.

2.7.3.2. NAT64/DNS64

Stateful NAT64 translation [RFC6146] allows IPv6-only clients to contact IPv4 servers using unicast UDP, TCP, or ICMP. It can be used in conjunction with DNS64 [RFC6147], a mechanism which synthesizes AAAA records from existing A records. There is also a stateless NAT64 [RFC6145] which is similar for the security aspects with the added benefit of being stateless, so, less prone to a state exhaustion attack.

The Security Consideration sections of [RFC6146] and [RFC6147] list the comprehensive issues. A specific issue with the use of NAT64 is that it will interfere with most IPsec deployments unless UDP encapsulation is used. DNS64 has an incidence on DNSSEC see section 3.1 of [RFC7050].

2.7.3.3. DS-Lite

Dual-Stack Lite (DS-Lite) [RFC6333] is a transition technique that enables a service provider to share IPv4 addresses among customers by combining two well-known technologies: IP in IP (IPv4-in-IPv6) and Network Address and Port Translation (NAPT).

Security considerations with respect to DS-Lite mainly revolve around logging data, preventing DoS attacks from rogue devices (as the AFTR function is stateful) and restricting service offered by the AFTR only to registered customers.

Section 11 of [RFC6333] describes important security issues associated with this technology.

2.8. General Device Hardening

There are many environments which rely too much on the network infrastructure to disallow malicious traffic to get access to critical hosts. In new IPv6 deployments it has been common to see IPv6 traffic enabled but none of the typical access control mechanisms enabled for IPv6 device access. With the possibility of network device configuration mistakes and the growth of IPv6 in the overall Internet it is important to ensure that all individual devices are hardened agains miscreant behavior.

The following guidelines should be used to ensure appropriate hardening of the host, be it an individual computer or router, firewall, load-balancer,server, etc device.

- Restrict access to the device to authorized individuals
- Monitor and audit access to the device
- Turn off any unused services on the end node
- Understand which IPv6 addresses are being used to source traffic and change defaults if necessary
- Use cryptographically protected protocols for device management if possible (SCP, SNMPv3, SSH, TLS, etc)
- Use host firewall capabilities to control traffic that gets processed by upper layer protocols
- Use virus scanners to detect malicious programs

3. Enterprises Specific Security Considerations

Enterprises generally have robust network security policies in place to protect existing IPv4 networks. These policies have been distilled from years of experiential knowledge of securing IPv4 networks. At the very least, it is recommended that enterprise networks have parity between their security policies for both protocol versions.

Security considerations in the enterprise can be broadly categorized into two sections - External and Internal.

3.1. External Security Considerations:

The external aspect deals with providing security at the edge or perimeter of the enterprise network where it meets the service providers network. This is commonly achieved by enforcing a security policy either by implementing dedicated firewalls with stateful packet inspection or a router with ACLs. A common default IPv4 policy on firewalls that could easily be ported to IPv6 is to allow all traffic outbound while only allowing specific traffic, such as established sessions, inbound (see also [RFC6092]). Here are a few more things that could enhance the default policy:

- Filter internal-use IPv6 addresses at the perimeter

- o Discard packets from and to bogon and reserved space, see also [CYMRU]

- o Accept certain ICMPv6 messages to allow proper operation of ND and PMTUD, see also [RFC4890]

- o Filter specific extension headers by accepting only the required ones (white list approach) such as ESP, AH (not forgetting the required transport layers: ICMP, TCP, UDP, ...) , where possible at the edge and possibly inside the perimeter; see also [I-D.gont-opsec-ipv6-eh-filtering]

- o Filter packets having an illegal IPv6 headers chain at the perimeter (and possible inside as well), see Section 2.2

- o Filter unneeded services at the perimeter

- o Implement anti-spoofing

- o Implement appropriate rate-limiters and control-plane policers

3.2. Internal Security Considerations:

The internal aspect deals with providing security inside the perimeter of the network, including the end host. The most significant concerns here are related to Neighbor Discovery. At the network level, it is recommended that all security considerations discussed in Section 2.3 be reviewed carefully and the recommendations be considered in-depth as well.

As mentioned in Section 2.6.2, care must be taken when running automated IPv6-in-IP4 tunnels.

Hosts need to be hardened directly through security policy to protect against security threats. The host firewall default capabilities have to be clearly understood, especially 3rd party ones which can have different settings for IPv4 or IPv6 default permit/deny behavior. In some cases, 3rd party firewalls have no IPv6 support whereas the native firewall installed by default has it. General device hardening guidelines are provided in Section 2.8

It should also be noted that many hosts still use IPv4 for transport for things like RADIUS, TACACS+, SYSLOG, etc. This will require some extra level of due diligence on the part of the operator.

4. Service Providers Security Considerations

4.1. BGP

The threats and mitigation techniques are identical between IPv4 and IPv6. Broadly speaking they are:

- Authenticating the TCP session;
- TTL security (which becomes hop-limit security in IPv6);
- Prefix Filtering.

These are explained in more detail in section Section 2.5.

4.1.1. Remote Triggered Black Hole Filtering

RTBH [RFC5635] works identically in IPv4 and IPv6. IANA has allocated 100::/64 as discard prefix RFC6666 [RFC6666].

4.2. Transition Mechanism

SP will typically use transition mechanisms such as 6rd, 6PE, MAP, DS-Lite which have been analyzed in the transition Section 2.7.2 section.

4.3. Lawful Intercept

The Lawful Intercept requirements are similar for IPv6 and IPv4 architectures and will be subject to the laws enforced in varying geographic regions. The local issues with each jurisdiction can make this challenging and both corporate legal and privacy personnel should be involved in discussions pertaining to what information gets logged and what the logging retention policies will be.

The target of interception will usually be a residential subscriber (e.g. his/her PPP session or physical line or CPE MAC address). With the absence of NAT on the CPE, IPv6 has the provision to allow for intercepting the traffic from a single host (a /128 target) rather than the whole set of hosts of a subscriber (which could be a /48, a /60 or /64).

In contrast, in mobile environments, since the 3GPP specifications allocate a /64 per device, it may be sufficient to intercept traffic from the /64 rather than specific /128’s (since each time the device powers up it gets a new IID).

A sample architecture which was written for informational purposes is found in [RFC3924].

5. Residential Users Security Considerations

The IETF Homenet working group is working on how IPv6 residential network should be done; this obviously includes operational security considerations; but, this is still work in progress.

Residential users have usually less experience and knowledge about security or networking. As most of the recent hosts, smartphones, tablets have all IPv6 enabled by default, IPv6 security is important for those users. Even with an IPv4-only ISP, those users can get IPv6 Internet access with the help of Teredo tunnels. Several peer-to-peer programs (notably Bittorrent) support IPv6 and those programs can initiate a Teredo tunnel through the IPv4 residential gateway, with the consequence of making the internal host reachable from any IPv6 host on the Internet. It is therefore recommended that all host security products (personal firewall, ...) are configured with a dual-stack security policy.

If the Residential Gateway has IPv6 connectivity, [RFC7084] (which obsoletes [RFC6204]) defines the requirements of an IPv6 CPE and does not take position on the debate of default IPv6 security policy as defined in [RFC6092]:

- outbound only: allowing all internally initiated connections and block all externally initiated ones, which is a common default security policy enforced by IPv4 Residential Gateway doing NAT-PT but it also breaks the end-to-end reachability promise of IPv6. [RFC6092] lists several recommendations to design such a CPE;

- open/transparent: allowing all internally and externally initiated connections, therefore restoring the end-to-end nature of the Internet for the IPv6 traffic but having a different security policy for IPv6 than for IPv4.

[RFC6092] REC-49 states that a choice must be given to the user to select one of those two policies.

There is also an alternate solution which has been deployed notably by Swisscom: open to all outbound and inbound connections at the exception of an handful of TCP and UDP ports known as vulnerable.

6. Further Reading

There are several documents that describe in more details the security of an IPv6 network; these documents are not written by the IETF but are listed here for your convenience:

1. Guidelines for the Secure Deployment of IPv6 [NIST]

2. North American IPv6 Task Force Technology Report - IPv6 Security Technology Paper [NAv6TF_Security]

3. IPv6 Security [IPv6_Security_Book]

7. Acknowledgements

The authors would like to thank the following people for their useful comments: Mikael Abrahamsson, Fred Baker, Brian Carpenter, Tim Chown, Markus deBruen, Tobias Fiebig, Fernando Gont, Jeffry Handal, Panos Kampanakis, Erik Kline, Jouni Korhonen, Mark Lentczner, Bob Sleigh,Tarko Tikan, Ole Troan, Bernie Volz (by alphabetical order).

8. IANA Considerations

This memo includes no request to IANA.

9. Security Considerations

This memo attempts to give an overview of security considerations of operating an IPv6 network both in an IPv6-only network and in utilizing the most widely deployed IPv4/IPv6 coexistence strategies.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <https://www.rfc-editor.org/info/rfc2460>.

[RFC6104] Chown, T. and S. Venaas, "Rogue IPv6 Router Advertisement Problem Statement", RFC 6104, DOI 10.17487/RFC6104, February 2011, <https://www.rfc-editor.org/info/rfc6104>.

[RFC6105] Levy-Abegnoli, E., Van de Velde, G., Popoviciu, C., and J. Mohacsi, "IPv6 Router Advertisement Guard", RFC 6105, DOI 10.17487/RFC6105, February 2011, <https://www.rfc-editor.org/info/rfc6105>.

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

10.2. Informative References

[CYMRU] "Packet Filter and Route Filter Recommendation for IPv6 at xSP routers", <http://www.team-cymru.org/ReadingRoom/Templates/IPv6Routers/xsp-recommendations.html>.

[europol-cgn] "ARE YOU SHARING THE SAME IP ADDRESS AS A CRIMINAL? LAW ENFORCEMENT CALL FOR THE END OF CARRIER GRADE NAT (CGN) TO INCREASE ACCOUNTABILITY ONLINE", October 2017, <https://www.europol.europa.eu/newsroom/news/are-you-sharing-same-ip-address-criminal-law-enforcement-call-for-end-of-carrier-grade-nat-cgn-to-increase-accountability-online>.

[I-D.chakrabarti-nordmark-6man-efficient-nd] Chakrabarti, S., Nordmark, E., Thubert, P., and M. Wasserman, "IPv6 Neighbor Discovery Optimizations for Wired and Wireless Networks", draft-chakrabarti-nordmark-6man-efficient-nd-07 (work in progress), February 2015.

[I-D.gont-opsec-ipv6-eh-filtering] Gont, F., Will, W., and R. Bonica, "Recommendations on Filtering of IPv6 Packets Containing IPv6 Extension Headers", draft-gont-opsec-ipv6-eh-filtering-02 (work in progress), August 2014.

[I-D.ietf-dhc-sedhcpv6] Li, L., Jiang, S., Cui, Y., Jinmei, T., Lemon, T., and D. Zhang, "Secure DHCPv6", draft-ietf-dhc-sedhcpv6-21 (work in progress), February 2017.

[I-D.ietf-opsec-ipv6-eh-filtering] Gont, F., LIU, W., and R. Bonica, "Recommendations on the Filtering of IPv6 Packets Containing IPv6 Extension Headers", draft-ietf-opsec-ipv6-eh-filtering-04 (work in progress), October 2017.

[I-D.kampanakis-6man-ipv6-eh-parsing]
Kampanakis, P., "Implementation Guidelines for parsing IPv6 Extension Headers", draft-kampanakis-6man-ipv6-eh-parsing-01 (work in progress), August 2014.

[I-D.thubert-savi-ra-throttler]
Thubert, P., "Throttling RAs on constrained interfaces", draft-thubert-savi-ra-throttler-01 (work in progress), June 2012.

[IEEE-802.1X]
IEEE, "IEEE Standard for Local and metropolitan area networks - Port-Based Network Access Control", IEEE Std 802.1X-2010, February 2010.

[IPv6_Security_Book]
Hogg and Vyncke, "IPv6 Security", ISBN 1-58705-594-5, Publisher CiscoPress, December 2008.

[NAv6TF_Security]
Kaeo, Green, Bound, and Pouffary, "North American IPv6 Task Force Technology Report - IPv6 Security Technology Paper", 2006, <http://www.ipv6forum.com/dl/white/NAv6TF_Security_Report.pdf>.

[NIST] Frankel, Graveman, Pearce, and Rooks, "Guidelines for the Secure Deployment of IPv6", 2010, <http://csrc.nist.gov/publications/nistpubs/800-119/sp800-119.pdf>.

[RFC0826] Plummer, D., "Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware", STD 37, RFC 826, DOI 10.17487/RFC0826, November 1982, <https://www.rfc-editor.org/info/rfc826>.

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, DOI 10.17487/RFC2131, March 1997, <https://www.rfc-editor.org/info/rfc2131>.

[RFC2529] Carpenter, B. and C. Jung, "Transmission of IPv6 over IPv4 Domains without Explicit Tunnels", RFC 2529, DOI 10.17487/RFC2529, March 1999, <https://www.rfc-editor.org/info/rfc2529>.

[RFC2740] Coltun, R., Ferguson, D., and J. Moy, "OSPF for IPv6", RFC 2740, DOI 10.17487/RFC2740, December 1999, <https://www.rfc-editor.org/info/rfc2740>.

[RFC2784] Farinacci, D., Li, T., Hanks, S., Meyer, D., and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 2784, DOI 10.17487/RFC2784, March 2000, <https://www.rfc-editor.org/info/rfc2784>.

[RFC2827] Ferguson, P. and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", BCP 38, RFC 2827, DOI 10.17487/RFC2827, May 2000, <https://www.rfc-editor.org/info/rfc2827>.

[RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, DOI 10.17487/RFC2866, June 2000, <https://www.rfc-editor.org/info/rfc2866>.

[RFC2993] Hain, T., "Architectural Implications of NAT", RFC 2993, DOI 10.17487/RFC2993, November 2000, <https://www.rfc-editor.org/info/rfc2993>.

[RFC3056] Carpenter, B. and K. Moore, "Connection of IPv6 Domains via IPv4 Clouds", RFC 3056, DOI 10.17487/RFC3056, February 2001, <https://www.rfc-editor.org/info/rfc3056>.

[RFC3068] Huitema, C., "An Anycast Prefix for 6to4 Relay Routers", RFC 3068, DOI 10.17487/RFC3068, June 2001, <https://www.rfc-editor.org/info/rfc3068>.

[RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, DOI 10.17487/RFC3315, July 2003, <https://www.rfc-editor.org/info/rfc3315>.

[RFC3627] Savola, P., "Use of /127 Prefix Length Between Routers Considered Harmful", RFC 3627, DOI 10.17487/RFC3627, September 2003, <https://www.rfc-editor.org/info/rfc3627>.

[RFC3756] Nikander, P., Ed., Kempf, J., and E. Nordmark, "IPv6 Neighbor Discovery (ND) Trust Models and Threats", RFC 3756, DOI 10.17487/RFC3756, May 2004, <https://www.rfc-editor.org/info/rfc3756>.

[RFC3924] Baker, F., Foster, B., and C. Sharp, "Cisco Architecture for Lawful Intercept in IP Networks", RFC 3924, DOI 10.17487/RFC3924, October 2004, <https://www.rfc-editor.org/info/rfc3924>.

[RFC3964] Savola, P. and C. Patel, "Security Considerations for 6to4", RFC 3964, DOI 10.17487/RFC3964, December 2004, <https://www.rfc-editor.org/info/rfc3964>.

[RFC3971] Arkko, J., Ed., Kempf, J., Zill, B., and P. Nikander, "SEcure Neighbor Discovery (SEND)", RFC 3971, DOI 10.17487/RFC3971, March 2005, <https://www.rfc-editor.org/info/rfc3971>.

[RFC3972] Aura, T., "Cryptographically Generated Addresses (CGA)", RFC 3972, DOI 10.17487/RFC3972, March 2005, <https://www.rfc-editor.org/info/rfc3972>.

[RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, DOI 10.17487/RFC4193, October 2005, <https://www.rfc-editor.org/info/rfc4193>.

[RFC4293] Routhier, S., Ed., "Management Information Base for the Internet Protocol (IP)", RFC 4293, DOI 10.17487/RFC4293, April 2006, <https://www.rfc-editor.org/info/rfc4293>.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <https://www.rfc-editor.org/info/rfc4301>.

[RFC4302] Kent, S., "IP Authentication Header", RFC 4302, DOI 10.17487/RFC4302, December 2005, <https://www.rfc-editor.org/info/rfc4302>.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <https://www.rfc-editor.org/info/rfc4303>.

[RFC4364] Rosen, E. and Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4364, DOI 10.17487/RFC4364, February 2006, <https://www.rfc-editor.org/info/rfc4364>.

[RFC4380] Huitema, C., "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)", RFC 4380, DOI 10.17487/RFC4380, February 2006, <https://www.rfc-editor.org/info/rfc4380>.

[RFC4381] Behringer, M., "Analysis of the Security of BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4381, DOI 10.17487/RFC4381, February 2006, <https://www.rfc-editor.org/info/rfc4381>.

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <https://www.rfc-editor.org/info/rfc4443>.

[RFC4552] Gupta, M. and N. Melam, "Authentication/Confidentiality for OSPFv3", RFC 4552, DOI 10.17487/RFC4552, June 2006, <https://www.rfc-editor.org/info/rfc4552>.

[RFC4649] Volz, B., "Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Relay Agent Remote-ID Option", RFC 4649, DOI 10.17487/RFC4649, August 2006, <https://www.rfc-editor.org/info/rfc4649>.

[RFC4659] De Clercq, J., Ooms, D., Carugi, M., and F. Le Faucheur, "BGP-MPLS IP Virtual Private Network (VPN) Extension for IPv6 VPN", RFC 4659, DOI 10.17487/RFC4659, September 2006, <https://www.rfc-editor.org/info/rfc4659>.

[RFC4798] De Clercq, J., Ooms, D., Prevost, S., and F. Le Faucheur, "Connecting IPv6 Islands over IPv4 MPLS Using IPv6 Provider Edge Routers (6PE)", RFC 4798, DOI 10.17487/RFC4798, February 2007, <https://www.rfc-editor.org/info/rfc4798>.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, DOI 10.17487/RFC4861, September 2007, <https://www.rfc-editor.org/info/rfc4861>.

[RFC4864] Van de Velde, G., Hain, T., Droms, R., Carpenter, B., and E. Klein, "Local Network Protection for IPv6", RFC 4864, DOI 10.17487/RFC4864, May 2007, <https://www.rfc-editor.org/info/rfc4864>.

[RFC4890] Davies, E. and J. Mohacsi, "Recommendations for Filtering ICMPv6 Messages in Firewalls", RFC 4890, DOI 10.17487/RFC4890, May 2007, <https://www.rfc-editor.org/info/rfc4890>.

[RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, DOI 10.17487/RFC4941, September 2007, <https://www.rfc-editor.org/info/rfc4941>.

[RFC4942] Davies, E., Krishnan, S., and P. Savola, "IPv6 Transition/ Co-existence Security Considerations", RFC 4942, DOI 10.17487/RFC4942, September 2007, <https://www.rfc-editor.org/info/rfc4942>.

[RFC5157] Chown, T., "IPv6 Implications for Network Scanning", RFC 5157, DOI 10.17487/RFC5157, March 2008, <https://www.rfc-editor.org/info/rfc5157>.

[RFC5214] Templin, F., Gleeson, T., and D. Thaler, "Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)", RFC 5214, DOI 10.17487/RFC5214, March 2008, <https://www.rfc-editor.org/info/rfc5214>.

[RFC5340] Coltun, R., Ferguson, D., Moy, J., and A. Lindem, "OSPF for IPv6", RFC 5340, DOI 10.17487/RFC5340, July 2008, <https://www.rfc-editor.org/info/rfc5340>.

[RFC5635] Kumari, W. and D. McPherson, "Remote Triggered Black Hole Filtering with Unicast Reverse Path Forwarding (uRPF)", RFC 5635, DOI 10.17487/RFC5635, August 2009, <https://www.rfc-editor.org/info/rfc5635>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <https://www.rfc-editor.org/info/rfc5952>.

[RFC5969] Townsley, W. and O. Troan, "IPv6 Rapid Deployment on IPv4 Infrastructures (6rd) -- Protocol Specification", RFC 5969, DOI 10.17487/RFC5969, August 2010, <https://www.rfc-editor.org/info/rfc5969>.

[RFC6092] Woodyatt, J., Ed., "Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for Providing Residential IPv6 Internet Service", RFC 6092, DOI 10.17487/RFC6092, January 2011, <https://www.rfc-editor.org/info/rfc6092>.

[RFC6144] Baker, F., Li, X., Bao, C., and K. Yin, "Framework for IPv4/IPv6 Translation", RFC 6144, DOI 10.17487/RFC6144, April 2011, <https://www.rfc-editor.org/info/rfc6144>.

[RFC6145] Li, X., Bao, C., and F. Baker, "IP/ICMP Translation Algorithm", RFC 6145, DOI 10.17487/RFC6145, April 2011, <https://www.rfc-editor.org/info/rfc6145>.

[RFC6146] Bagnulo, M., Matthews, P., and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", RFC 6146, DOI 10.17487/RFC6146, April 2011, <https://www.rfc-editor.org/info/rfc6146>.

[RFC6147] Bagnulo, M., Sullivan, A., Matthews, P., and I. van Beijnum, "DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers", RFC 6147, DOI 10.17487/RFC6147, April 2011, <https://www.rfc-editor.org/info/rfc6147>.

[RFC6164] Kohno, M., Nitzan, B., Bush, R., Matsuzaki, Y., Colitti, L., and T. Narten, "Using 127-Bit IPv6 Prefixes on Inter-Router Links", RFC 6164, DOI 10.17487/RFC6164, April 2011, <https://www.rfc-editor.org/info/rfc6164>.

[RFC6169] Krishnan, S., Thaler, D., and J. Hoagland, "Security Concerns with IP Tunneling", RFC 6169, DOI 10.17487/RFC6169, April 2011, <https://www.rfc-editor.org/info/rfc6169>.

[RFC6192] Dugal, D., Pignataro, C., and R. Dunn, "Protecting the Router Control Plane", RFC 6192, DOI 10.17487/RFC6192, March 2011, <https://www.rfc-editor.org/info/rfc6192>.

[RFC6204] Singh, H., Beebee, W., Donley, C., Stark, B., and O. Troan, Ed., "Basic Requirements for IPv6 Customer Edge Routers", RFC 6204, DOI 10.17487/RFC6204, April 2011, <https://www.rfc-editor.org/info/rfc6204>.

[RFC6221] Miles, D., Ed., Ooghe, S., Dec, W., Krishnan, S., and A. Kavanagh, "Lightweight DHCPv6 Relay Agent", RFC 6221, DOI 10.17487/RFC6221, May 2011, <https://www.rfc-editor.org/info/rfc6221>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6264] Jiang, S., Guo, D., and B. Carpenter, "An Incremental Carrier-Grade NAT (CGN) for IPv6 Transition", RFC 6264, DOI 10.17487/RFC6264, June 2011, <https://www.rfc-editor.org/info/rfc6264>.

[RFC6269] Ford, M., Ed., Boucadair, M., Durand, A., Levis, P., and P. Roberts, "Issues with IP Address Sharing", RFC 6269, DOI 10.17487/RFC6269, June 2011, <https://www.rfc-editor.org/info/rfc6269>.

[RFC6296] Wasserman, M. and F. Baker, "IPv6-to-IPv6 Network Prefix Translation", RFC 6296, DOI 10.17487/RFC6296, June 2011, <https://www.rfc-editor.org/info/rfc6296>.

[RFC6302] Durand, A., Gashinsky, I., Lee, D., and S. Sheppard, "Logging Recommendations for Internet-Facing Servers", BCP 162, RFC 6302, DOI 10.17487/RFC6302, June 2011, <https://www.rfc-editor.org/info/rfc6302>.

[RFC6324] Nakibly, G. and F. Templin, "Routing Loop Attack Using IPv6 Automatic Tunnels: Problem Statement and Proposed Mitigations", RFC 6324, DOI 10.17487/RFC6324, August 2011, <https://www.rfc-editor.org/info/rfc6324>.

[RFC6333] Durand, A., Droms, R., Woodyatt, J., and Y. Lee, "Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion", RFC 6333, DOI 10.17487/RFC6333, August 2011, <https://www.rfc-editor.org/info/rfc6333>.

[RFC6343] Carpenter, B., "Advisory Guidelines for 6to4 Deployment", RFC 6343, DOI 10.17487/RFC6343, August 2011, <https://www.rfc-editor.org/info/rfc6343>.

[RFC6434] Jankiewicz, E., Loughney, J., and T. Narten, "IPv6 Node Requirements", RFC 6434, DOI 10.17487/RFC6434, December 2011, <https://www.rfc-editor.org/info/rfc6434>.

[RFC6459] Korhonen, J., Ed., Soininen, J., Patil, B., Savolainen, T., Bajko, G., and K. Iisakkila, "IPv6 in 3rd Generation Partnership Project (3GPP) Evolved Packet System (EPS)", RFC 6459, DOI 10.17487/RFC6459, January 2012, <https://www.rfc-editor.org/info/rfc6459>.

[RFC6506] Bhatia, M., Manral, V., and A. Lindem, "Supporting Authentication Trailer for OSPFv3", RFC 6506, DOI 10.17487/RFC6506, February 2012, <https://www.rfc-editor.org/info/rfc6506>.

[RFC6547] George, W., "RFC 3627 to Historic Status", RFC 6547, DOI 10.17487/RFC6547, February 2012, <https://www.rfc-editor.org/info/rfc6547>.

[RFC6564] Krishnan, S., Woodyatt, J., Kline, E., Hoagland, J., and M. Bhatia, "A Uniform Format for IPv6 Extension Headers", RFC 6564, DOI 10.17487/RFC6564, April 2012, <https://www.rfc-editor.org/info/rfc6564>.

[RFC6583] Gashinsky, I., Jaeggli, J., and W. Kumari, "Operational Neighbor Discovery Problems", RFC 6583, DOI 10.17487/RFC6583, March 2012, <https://www.rfc-editor.org/info/rfc6583>.

[RFC6598] Weil, J., Kuarsingh, V., Donley, C., Liljenstolpe, C., and M. Azinger, "IANA-Reserved IPv4 Prefix for Shared Address Space", BCP 153, RFC 6598, DOI 10.17487/RFC6598, April 2012, <https://www.rfc-editor.org/info/rfc6598>.

[RFC6620] Nordmark, E., Bagnulo, M., and E. Levy-Abegnoli, "FCFS SAVI: First-Come, First-Served Source Address Validation Improvement for Locally Assigned IPv6 Addresses", RFC 6620, DOI 10.17487/RFC6620, May 2012, <https://www.rfc-editor.org/info/rfc6620>.

[RFC6666] Hilliard, N. and D. Freedman, "A Discard Prefix for IPv6", RFC 6666, DOI 10.17487/RFC6666, August 2012, <https://www.rfc-editor.org/info/rfc6666>.

[RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <https://www.rfc-editor.org/info/rfc6762>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/info/rfc6763>.

[RFC6810] Bush, R. and R. Austein, "The Resource Public Key Infrastructure (RPKI) to Router Protocol", RFC 6810, DOI 10.17487/RFC6810, January 2013, <https://www.rfc-editor.org/info/rfc6810>.

[RFC6939] Halwasia, G., Bhandari, S., and W. Dec, "Client Link-Layer Address Option in DHCPv6", RFC 6939, DOI 10.17487/RFC6939, May 2013, <https://www.rfc-editor.org/info/rfc6939>.

[RFC6964] Templin, F., "Operational Guidance for IPv6 Deployment in IPv4 Sites Using the Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)", RFC 6964, DOI 10.17487/RFC6964, May 2013, <https://www.rfc-editor.org/info/rfc6964>.

[RFC6980] Gont, F., "Security Implications of IPv6 Fragmentation with IPv6 Neighbor Discovery", RFC 6980, DOI 10.17487/RFC6980, August 2013, <https://www.rfc-editor.org/info/rfc6980>.

[RFC7011] Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, DOI 10.17487/RFC7011, September 2013, <https://www.rfc-editor.org/info/rfc7011>.

[RFC7012] Claise, B., Ed. and B. Trammell, Ed., "Information Model for IP Flow Information Export (IPFIX)", RFC 7012, DOI 10.17487/RFC7012, September 2013, <https://www.rfc-editor.org/info/rfc7012>.

[RFC7039] Wu, J., Bi, J., Bagnulo, M., Baker, F., and C. Vogt, Ed., "Source Address Validation Improvement (SAVI) Framework", RFC 7039, DOI 10.17487/RFC7039, October 2013, <https://www.rfc-editor.org/info/rfc7039>.

[RFC7045] Carpenter, B. and S. Jiang, "Transmission and Processing of IPv6 Extension Headers", RFC 7045, DOI 10.17487/RFC7045, December 2013, <https://www.rfc-editor.org/info/rfc7045>.

[RFC7050] Savolainen, T., Korhonen, J., and D. Wing, "Discovery of the IPv6 Prefix Used for IPv6 Address Synthesis", RFC 7050, DOI 10.17487/RFC7050, November 2013, <https://www.rfc-editor.org/info/rfc7050>.

[RFC7084] Singh, H., Beebee, W., Donley, C., and B. Stark, "Basic Requirements for IPv6 Customer Edge Routers", RFC 7084, DOI 10.17487/RFC7084, November 2013, <https://www.rfc-editor.org/info/rfc7084>.

[RFC7112] Gont, F., Manral, V., and R. Bonica, "Implications of Oversized IPv6 Header Chains", RFC 7112, DOI 10.17487/RFC7112, January 2014, <https://www.rfc-editor.org/info/rfc7112>.

[RFC7113] Gont, F., "Implementation Advice for IPv6 Router Advertisement Guard (RA-Guard)", RFC 7113, DOI 10.17487/RFC7113, February 2014, <https://www.rfc-editor.org/info/rfc7113>.

[RFC7166] Bhatia, M., Manral, V., and A. Lindem, "Supporting Authentication Trailer for OSPFv3", RFC 7166, DOI 10.17487/RFC7166, March 2014, <https://www.rfc-editor.org/info/rfc7166>.

[RFC7217] Gont, F., "A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC)", RFC 7217, DOI 10.17487/RFC7217, April 2014, <https://www.rfc-editor.org/info/rfc7217>.

[RFC7381] Chittimaneni, K., Chown, T., Howard, L., Kuarsingh, V., Pouffary, Y., and E. Vyncke, "Enterprise IPv6 Deployment Guidelines", RFC 7381, DOI 10.17487/RFC7381, October 2014, <https://www.rfc-editor.org/info/rfc7381>.

[RFC7404] Behringer, M. and E. Vyncke, "Using Only Link-Local Addressing inside an IPv6 Network", RFC 7404, DOI 10.17487/RFC7404, November 2014, <https://www.rfc-editor.org/info/rfc7404>.

[RFC7422] Donley, C., Grundemann, C., Sarawat, V., Sundaresan, K., and O. Vautrin, "Deterministic Address Mapping to Reduce Logging in Carrier-Grade NAT Deployments", RFC 7422, DOI 10.17487/RFC7422, December 2014, <https://www.rfc-editor.org/info/rfc7422>.

[RFC7454] Durand, J., Pepelnjak, I., and G. Doering, "BGP Operations and Security", BCP 194, RFC 7454, DOI 10.17487/RFC7454, February 2015, <https://www.rfc-editor.org/info/rfc7454>.

[RFC7513] Bi, J., Wu, J., Yao, G., and F. Baker, "Source Address Validation Improvement (SAVI) Solution for DHCP", RFC 7513, DOI 10.17487/RFC7513, May 2015, <https://www.rfc-editor.org/info/rfc7513>.

[RFC7526] Troan, O. and B. Carpenter, Ed., "Deprecating the Anycast Prefix for 6to4 Relay Routers", BCP 196, RFC 7526, DOI 10.17487/RFC7526, May 2015, <https://www.rfc-editor.org/info/rfc7526>.

[RFC7597] Troan, O., Ed., Dec, W., Li, X., Bao, C., Matsushima, S., Murakami, T., and T. Taylor, Ed., "Mapping of Address and Port with Encapsulation (MAP-E)", RFC 7597, DOI 10.17487/RFC7597, July 2015, <https://www.rfc-editor.org/info/rfc7597>.

[RFC7599] Li, X., Bao, C., Dec, W., Ed., Troan, O., Matsushima, S., and T. Murakami, "Mapping of Address and Port using Translation (MAP-T)", RFC 7599, DOI 10.17487/RFC7599, July 2015, <https://www.rfc-editor.org/info/rfc7599>.

[RFC7610] Gont, F., Liu, W., and G. Van de Velde, "DHCPv6-Shield: Protecting against Rogue DHCPv6 Servers", BCP 199, RFC 7610, DOI 10.17487/RFC7610, August 2015, <https://www.rfc-editor.org/info/rfc7610>.

[RFC7707] Gont, F. and T. Chown, "Network Reconnaissance in IPv6 Networks", RFC 7707, DOI 10.17487/RFC7707, March 2016, <https://www.rfc-editor.org/info/rfc7707>.

[RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", RFC 7721, DOI 10.17487/RFC7721, March 2016, <https://www.rfc-editor.org/info/rfc7721>.

[RFC7872] Gont, F., Linkova, J., Chown, T., and W. Liu, "Observations on the Dropping of Packets with IPv6 Extension Headers in the Real World", RFC 7872, DOI 10.17487/RFC7872, June 2016, <https://www.rfc-editor.org/info/rfc7872>.

[RFC8064] Gont, F., Cooper, A., Thaler, D., and W. Liu, "Recommendation on Stable IPv6 Interface Identifiers", RFC 8064, DOI 10.17487/RFC8064, February 2017, <https://www.rfc-editor.org/info/rfc8064>.

[RFC8190] Bonica, R., Cotton, M., Haberman, B., and L. Vegoda, "Updates to the Special-Purpose IP Address Registries", BCP 153, RFC 8190, DOI 10.17487/RFC8190, June 2017, <https://www.rfc-editor.org/info/rfc8190>.

[RFC8273] Brzozowski, J. and G. Van de Velde, "Unique IPv6 Prefix per Host", RFC 8273, DOI 10.17487/RFC8273, December 2017, <https://www.rfc-editor.org/info/rfc8273>.

[SCANNING] "Mapping the Great Void - Smarter scanning for IPv6", <http://www.caida.org/workshops/isma/1202/slides/aims1202_rbarnes.pdf>.

Authors' Addresses

Eric Vyncke (editor)
Cisco
De Kleetlaan 6a
Diegem 1831
Belgium

Phone: +32 2 778 4677
Email: evyncke@cisco.com

Kiran K. Chittimaneni
Dropbox Inc.
185 Berry Street, Suite 400
San Francisco, CA 94107
USA

Email: kk@dropbox.com

Merike Kaeo
Double Shot Security
3518 Fremont Ave N 363
Seattle 98103
USA

Phone: +12066696394
Email: merike@doubleshotsecurity.com

Enno Rey
ERNW
Carl-Bosch-Str. 4
Heidelberg, Baden-Wuertemberg 69115
Germany

Phone: +49 6221 480390
Email: erey@ernw.de