Dispatch
Internet-Draft
Intended status: Standards Track
Expires: July 13, 2014

O. Johansson
Edvina AB

January 9, 2014

TLS sessions in SIP using DNS-based Authentication of Named Entities (DANE) TLSA records
draft-johansson-dispatch-dane-sip-01

Abstract

Use of TLS in the SIP protocol is defined in multiple documents, starting with RFC 3261. The actual verification that happens when setting up a SIP TLS connection to a SIP server based on a SIP URI is described in detail in RFC 5922 - SIP Domain Certificates.

In this document, an alternative method is defined, using DNS-Based Authentication of Named Entities (DANE). By looking up TLSA DNS records and using DNSsec protection of the required queries, including lookups for NAPTR and SRV records, a SIP Client can verify the identity of the TLS SIP server in a different way, matching on the SRV host name in the X.509 PKIX certificate instead of the SIP domain. This provides more scalability in hosting solutions and make it easier to use standard CA certificates (if needed at all).

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

RFC 3261 [RFC3261] defines how to use TLS in the SIP protocol, but doesn't describe the actual verification between a SIP request and a TLS server certificate in detail. RFC 5922 [RFC5922] updates RFC 3261 with a definition of how a SIP client matches a PKIX X.509 [RFC5280] certificate provided by a TLS-enabled SIP server with the domain of a SIP request that caused the connection to be set up. Verification is done using the domain part of the SIP URI and the X.509 SubjectAltName extension of type dNSName or uniformResourceIdentifier. This is called "domain verification" as opposed to "host verification" in RFC 5922.

Including all domains hosted by a server in a server's certificate doesn't provide for a scalable and easy-managed solution. Every time

a service adds a domain, a new certificate will need to be provided, unless TLS Server Name Identification (SNI) is used, where each domain can have it’s own certificate. Having one certificate per domain and subdomain adds to the administration of a service. In addition, no known commercial CA offers certificate services with SIP URI’s in the certificates.

Using DNSsec and DNS-based Authentication of Named Entities (DANE)[RFC6698] the chain from a SIP uri to a TLS certificate changes, as outlined in this document. With DNSsec, the DNS lookups are authenticated and can be verified and trusted. [I-D.ietf-dane-srv] describes a DANE-based chain of trust, matching the SRV host name with the contents of the certificate.

This document describes how a SIP implementation can use DANE to set up a secure connection to a SIP server with TLS support. In addition, we describe how a server can provide support for RFC 5922-style clients with the same certificate, if needed.

This document adds an alternative to RFC5922 so that SIP implementations supporting DANE can validate a SIP domain identity using secure DNS queries and the identity of the SIP host by verifying the certificate using the SRV host name found in a SubjectAltName extension of type DNSName in the certificate. The domain verification will now happen based on DNSsec and the TLS verification will be based on host names (host verification in RFC 5922).

In order to learn about DANE and the different ways a TLSA record can be constructed, readers of this document needs to also read RFC 6698 [RFC6698].

2. Terminology and Conventions Used in This Document

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].

RFC 3261 [RFC3261] defines additional terms used in this document that are specific to the SIP domain such as "proxy"; "registrar"; "redirect server"; "user agent server" or "UAS"; "user agent client" or "UAC"; "back-to-back user agent" or "B2BUA"; "dialog"; "transaction"; "server transaction".

This document uses the term "SIP Server" that is defined to include the following SIP entities: user agent server (UAS), registrar, redirect server, a SIP proxy in the role of user agent server, and a B2BUA in the role of a user agent server.

This document uses the term "SIP client" that is defined to include the following SIP entities: user agent client(UAC), a SIP proxy in the role of user agent client, and a B2BUA in the role of a user agent client.

3. Using DNS in the SIP protocol

RFC 3263[RFC3263] describes how a SIP implementation use DNS to find the next hop server. The first step is to look up a DNS NAPTR record for the domain part of the URI. NAPTR records are used by the target domain to indicate reachability using different transports. NAPTR may be used to indicate a preference for TLS/TCP connections.

The result of the NAPTR lookup is a DNS name used to query for DNS SRV records. The list of DNS SRV records indicate host names that are queried for to find A or AAAA records with IP addresses.

SIP SRV records for TLS/TCP are using the prefix _sips._tcp, as in the DNS name _sips._tcp.example.com.

A SIP implementation with no support for NAPTR may, based on configuration or URI scheme, choose to set up a TLS session to the target domain.

> In rare cases, no SRV lookup is done. This means that the implementation lacks capability to do load balancing and failover based on the information in the DNS. These type of clients are not considered in this document.

4. Why DNSsec is important for SIP

DNS relies on DNS lookups not only to find the next hop server, but also for server administrators to provide failover and to load balance clients. The result of querying for one domain may need to SRV records or host names in another domain. Without DNSsec, an attacker can forge DNS replies and issue bogus DNS records, directing traffic to a bad server. This applies to calls as well as instant messaging, chat and presense.

5. Secure delegation is required for DANE to apply

It is important for implementors to understand the concept of "secure" DNSsec validation according to RFC 4033[RFC4033]. For this specification to take effect, all DNS RRsets in the chain from SIP URI to IP address and TLSA record must be secure. (This corresponds to the A.D. bit being set in the responses).

If any RRset is not secure, this specification doesn’t apply and the implementation should fall back to RFC 5922[RFC5922] behaviour. If any of the responses are "bogus" according to DNSsec validation, the client MUST abort the connection.

6. TLSA record name

For the SIP protocol DANE usage, TLSA records are to be found in accordance with [I-D.ietf-dane-srv]. If the domain example.com’s TLS SRV records points to sip01.example.com port 5042 then the corresponding TLSA record will be found using the name _5042._tcp.sip01.example.com.

7. Procedures for DANE-capable SIP implementations

DANE capable SIP implementations follow the procedures above to find a SRV host name and look for a TLSA record. If no TLSA record is found, the client should fall back to RFC 5922 behaviour.

If a TLSA record is found, the client should never fall back to RFC 5922 behaviour. If TLSA-based validation fails, the client MUST abort the connection attempt.

8. X.509 certicate validation

When using DANE-based validation the client validates the SRV hostname with the certificate using RFC 5922 rules. A DANE-capable SIP implementation looks for the SRV hostname in the list of SubjAltName DNSName extension fields. Only if there are no SubjAltName extension fields may the client look in the CN of the X.509 certificate (according to RFC 5922).

If the SRV host name is not found in the certificate, DANE validation fails and the client MUST abort the connection.

Using the SRV host name for validation of a SIP domain identity is an update to RFC 5922

9. Backward Compatibility with RFC 5922

RFC 5922[RFC5922] implementations with no DANE support will be able to connect with the matching described in that document. SIP Servers can use certificates that are compatible with both this specification and RFC5922.

[I-D.ietf-dane-srv] requires use of the TLS Server Name Indication (SNI) extension [RFC6066]. This is not a requirement in this

document, since SIP certificates can support both RFC 5922 style validation and DANE-based validation with the same certificate.

10. Examples on certificate content

This section gives examples on certificate content and how the match a given URI. The X.509 PKIX Subject field CN value is abbreviated as "CN", the SubjectAltName extension DNSName and uniformResourceIdentifier are abbreviated as "SAN-DNS" and "SAN-URI". The certificates are tested with three different clients. A DANE-aware client, a RFC 5922 client with no DANE support and a client that matches the SIP domain with the Common Name in the Subject of the certificate. The last example is not really covered by any SIP-related RFC and should be avoided.

10.1. Example 1: johansson.example.com

- Domain: johansson.example.com
- DNS SRV host for TLS: siphosting.example.net

Certificate content:

- CN: siphosting.example.net
- SAN-URI: -
- SAN-DNS: -
- Matching for DANE-aware SIP clients: Yes
- Matching for only RFC 5922 SIP clients: No
- Matching on CNAME only: No

10.2. Example 2: lundholm.example.com

- Domain: lundholm.example.com
- DNS SRV host for TLS: sipcrew.example.net

Certificate content:

- CN: randomname.example.net
- SAN-URI: sip:lundholm.example.com
- SAN-DNS: lundholm.example.com

- o Matching for DANE-aware SIP clients: Yes

- o Matching for only RFC 5922 SIP clients: Yes

Note: More examples is coming here.

11. Security Considerations

This document use already published solutions for providing credentials for setting up a secure connection to a SIP server. By depending on secure lookups of DNS NAPTR and SRV records as well as using TLSA records to verify a SIP servers TLS certificate it describes a secure method for making sure that a SIP request for a domain is sent to an authoritative server.

In addition to this document, many security considerations are covered in ID.ietf-dane-srv.

12. IANA Considerations

This document does not require actions by IANA.

13. Acknowledgements

The author wishes to acknowledge Jakob Schlyter for inspiration and .SE for promoting DNSsec and DANE. Victor Dubovn

14. References

14.1. Normative References

[I-D.ietf-dane-srv]
Finch, T., Miller, M., and P. Saint-Andre, "Using DNS-Based Authentication of Named Entities (DANE) TLSA records with SRV and MX records.", draft-ietf-dane-srv-03 (work in progress), December 2013.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3263] Rosenberg, J. and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", RFC 3263, June 2002.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[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, May 2008.

[RFC5922] Gurbani, V., Lawrence, S., and A. Jeffrey, "Domain Certificates in the Session Initiation Protocol (SIP)", RFC 5922, June 2010.

[RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

[RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, August 2012.

14.2. Informative References

[I-D.ogud-dane-vocabulary]
Gudmundsson, O., "Harmonizing how applications specify DANE-like usage", draft-ogud-dane-vocabulary-01 (work in progress), October 2013.

[RFC5589] Sparks, R., Johnston, A., and D. Petrie, "Session Initiation Protocol (SIP) Call Control - Transfer", BCP 149, RFC 5589, June 2009.

Appendix A. Appendix A. Implementation notes

Developers of SIP implementations are strongly encouraged to implement RFC 5922 and this document for secure verification of a SIP domain with a TLS server. This document also encourages implementation of TLS SNI both in client and server implementations. In order to get support of this function, update to new versions of the TLS libraries and make sure that the implementation supports new versions of TLS - TLS 1.1 [RFC4346] and TLS 1.2 [RFC5246].

Implementatinos that do support TLS are encouraged to always start with attempting TLS, even if the URI is a SIP: uri. If there are NAPTR records for the domain and the domain indicates support of TLS, use it. If there are no NAPTR records, start SRV lookup with the _sips._tcp prefix. This way, the SIP network will gradually shift to always using secure and authenticated TLS sessions.

Author's Address

Olle E. Johansson
Edvina AB
Runbovaegen 10
Sollentuna SE-192 48
SE

Email: oej@edvina.net

Network Working Group
Internet-Draft
Updates: 3515 (if approved)
Intended status: Standards Track
Expires: December 18, 2014

R. Sparks
Oracle
A. Roach
Mozilla

June 16, 2014

Clarifications for the use of REFER with RFC6665
draft-sparks-sipcore-refer-clarifications-00

Abstract

An accepted SIP REFER method creates an implicit subscription using the SIP-Specific Event Notification Framework. That framework was revised by RFC6665. This document highlights the implications of the requirement changes in RFC6665, and updates the definition of the REFER method, RFC3515, to clarify and disambiguate the impact of those changes.

Status of This Memo

Copyright Notice

Table of Contents

1. Conventions and Definitions

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].

2. Introduction

An accepted SIP REFER method creates an implicit subscription using the SIP-Specific Event Notification Framework. That framework was revised by [RFC6665]. This document highlights the implications of the requirement changes in RFC6665, and updates [RFC3515] to clarify and disambiguate the impact of those changes.

3. Use of GRUU is mandatory

Section 4.5.1 of [RFC6665] makes GRUU [RFC5627] mandatory to implement and use as the local target in the subscription created by the REFER request.

A user agent constructing a REFER request MUST populate its Contact header field with a GRUU.

As RFC6665 details, this is necessary to ensure that NOTIFY requests sent in the implicitly created subscription arrive at this user agent without creating a second usage inside an existing dialog. Using the "norefersub" option tag [RFC4488] does not change this requirement, even if used in a "Require" header field. Even if the recipient supports the "norefersub" mechanism, and accepts the request with the option tag in the "Require" header field, it is allowed to return a "Refer-Sub" header field with a value of "true" in the response, and create an implicit subscription.

4. Dialog reuse is prohibited

As a direct consequence of requiring the use of GRUU, and the requirements in section 4.5.2 of RFC6665, sending a REFER within any existing dialog is prohibited.

A user agent constructing a REFER request MUST built it as an out-of-dialog message as defined in [RFC3261]. Thus, the REFER request will have no tag parameter in its To: header field.

A user agent wishing to identify an existing dialog (such as for call transfer as defined in [RFC5589] MUST use the "Target-Dialog" extension defined in [RFC4538] to do so.

5. Security Considerations

This document introduces no new security considerations directly. The updated considerations in [RFC6665] apply to the implicit subscription created by an accepted REFER request.

6. IANA Considerations

This document has no actions for IANA.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3515] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[RFC4538] Rosenberg, J., "Request Authorization through Dialog Identification in the Session Initiation Protocol (SIP)", RFC 4538, June 2006.

[RFC5627] Rosenberg, J., "Obtaining and Using Globally Routable User Agent URIs (GRUUs) in the Session Initiation Protocol (SIP)", RFC 5627, October 2009.

[RFC6665] Roach, A., "SIP-Specific Event Notification", RFC 6665, July 2012.

7.2. Informative References

[RFC4488] Levin, O., "Suppression of Session Initiation Protocol (SIP) REFER Method Implicit Subscription", RFC 4488, May 2006.

[RFC5589] Sparks, R., Johnston, A., and D. Petrie, "Session Initiation Protocol (SIP) Call Control - Transfer", BCP 149, RFC 5589, June 2009.

Authors' Addresses

Robert Sparks
Oracle
7460 Warren Parkway
Suite 300
Frisco, Texas 75034
US

Email: RjS@nostrum.com

Adam Roach
Mozilla
Dallas, TX
US

Phone: +1 650 903 0800 x863
Email: adam@nostrum.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 19, 2014

R. Sparks
Oracle

June 17, 2014

Explicit Subscriptions for the REFER Method
draft-sparks-sipcore-refer-explicit-subscription-00

Abstract

The SIP REFER request, as defined by RFC3515, triggers an implicit SIP-Specific Event Notification framework subscription. Conflating the start of the subscription with handling the REFER request makes negotiating SUBSCRIBE extensions impossible, and complicates avoiding SIP dialog sharing. This document defines an extension to REFER to remove the implicit subscription and replace it with an explicit one.

Status of This Memo

Copyright Notice

Table of Contents

1. Conventions and Definitions

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].

2. Status

This version of the document is a strawman proposal of mechanism, and seeds for discussion of that proposal. The material in this version of the document is not yet appropriate for early implementation.

3. Introduction

REFER as defined by [RFC3515] triggers an implicit SIP-Specific Event Framework subscription. Sending a REFER within a dialog established by an INVITE results in dialog reuse and the associated problems described in [RFC5057]. The SIP-Specific Event Notification framework definition [RFC6665] disallows such dialog reuse. Call transfer, as defined in [RFC5589], thus requires sending a REFER request on a new dialog, associating it with an existing dialog using the ’Target-Dialog’ mechanism defined in [RFC4538].

Because there is no explicit SUBSCRIBE request, the tools for negotiating subscription details are unavailable for REFER

subscriptions. This includes negotiating subscription duration and providing information through Event header field parameters. The use of the SIP 'Supported' and 'Require' extension mechanisms [RFC3261] is complicated by the implicit subscription. Avoiding potential confusion around whether the extension applies to handling the REFER request itself, or to the messages in the subscription created by the REFER, or both, requires careful specification in each extension. Many existing extensions do not provide this clarity.

This document proposes a strawman mechanism to remove the implicit subscription and replace it with an explicit one. The benefits of doing so include:

- o Allowing REFER to be used within INVITE-created dialogs without creating dialog reuse.

- o Allowing standard subscription parameter negotiation.

- o Allowing standard negotiation of SIP extensions.

4. Strawman Mechanism

In this version of the draft, the mechanism is described at a high level. Future versions (if the group decides to explore this idea further) will contain a formal specification.

Define an option tag ('explicitsub') for use in the Require: header field of REFER requests.

Define a header field ('Refer-Events-At') to be included in 200 class REFER responses, containing a URL with GRUU-properties for which to subscribe to event 'refer'

Modify the REFER specifiation to allow accepting a new SUBSCRIBE for event 'refer'

The authorization policy for accepting a REFER subscription is based on possession of the URL for the resource. (Any adversary capable of obtaining the URL is in a position to see any NOTIFY contents anyhow). The notifier may accept more than one subscription for the resource as long as the resource exists. The mechanics called out in RFC3515 for indicating that the resource no longer exists still apply.

5. Discussion Points

5.1. Backward compatibility

A 3515 Notifier will reject an initial REFER because of the Require: option, allowing the updated Subscriber to learn that it needs to retry the REFER allowing for an implicit subsccription.

An updated Notifier receiving a REFER without the option tag can create an implicit subscription per 3515.

5.2. Should this be a different method?

Instead of using Require:, this could be something like REFERBIS.

The strawman proposes no, on the weak grounds that new option tags are easier to deploy than new methods.

5.3. Should this use a different event package?

The strawman proposes no. Neither the payload of NOTIFY messages, nor the meaning of the state being subscribed to changes.

5.4. Could this deprecate RFC4488?

The extension could define that a server not wishing to provide subscriptions return a URL from the .invalid family in the ’Refer-Events-At’ header field (alternatively, no Refer-Events-At header field at all). That, along with the use of Require (and not Supported) from the subscriber simplifies the negotiation flows over those in [RFC4488] when the client does not know ahead of time if the server supports the extension. A client not interested in subscriptions can simply not subscribe.

5.5. Should this tighten down what can appear in a Refer-To header field?

The strawman proposal is no. As with 3515, a REFER recipient that doesn’t know what to do with a non-SIP URL (for example) can decline the REFER.

6. Security Considerations

If the group decides to pursue this idea, this section will need to add detail to the authorization policy mentioned above, and consider the ramifications of being asked to subscribe to a URL this way (which will be very similar to the security considerations that apply to the URI in a Refer-To header field in the first place).

7. IANA Considerations

This document has no actions for IANA.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3515] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[RFC4488] Levin, O., "Suppression of Session Initiation Protocol (SIP) REFER Method Implicit Subscription", RFC 4488, May 2006.

[RFC4538] Rosenberg, J., "Request Authorization through Dialog Identification in the Session Initiation Protocol (SIP)", RFC 4538, June 2006.

[RFC5589] Sparks, R., Johnston, A., and D. Petrie, "Session Initiation Protocol (SIP) Call Control - Transfer", BCP 149, RFC 5589, June 2009.

[RFC6665] Roach, A., "SIP-Specific Event Notification", RFC 6665, July 2012.

8.2. Informative References

[RFC5057] Sparks, R., "Multiple Dialog Usages in the Session Initiation Protocol", RFC 5057, November 2007.

Author’s Address

Robert Sparks
Oracle
7460 Warren Parkway
Suite 300
Frisco, Texas 75034
US

Email: RjS@nostrum.com

SIPCORE Working Group
INTERNET-DRAFT
Intended Status: Standards Track
Expires: July 29, 2014

R. Shekh-Yusef
Avaya

January 25, 2014

The Session Initiation Protocol (SIP) Digest Authentication Scheme
draft-yusef-sipcore-digest-scheme-04

Abstract

This document updates the Digest Access Authentication scheme used by the Session Initiation Protocol (SIP) to add support for SHA2 digest algorithms to replace the MD5 algorithm.

Status of this Memo

Copyright and License Notice

Table of Contents

1 Introduction

The SIP protocol [RFC3261] uses the same mechanism used by the HTTP protocol for authenticating users, which is a simple challenge-response authentication mechanism that allows a server to challenge a client request and allows a client to provide authentication information in response to that challenge.

The SIP protocol uses the Digest Authentication scheme that is used with the HTTP authentication mechanism, which by default uses MD5 as the default algorithm.

The HTTP Digest Access Authentication [HTTP-DIGEST] document defines the challenge-response authentication mechanism and the Digest Authentication scheme, and defines few algorithms that could be used with the Digest Authentication scheme, and establishes a registry for these algorithms to allow for additional algorithms to be added in the future.

In 2008 the US-CERT issued a note that MD5 "should be considered cryptographically broken and unsuitable for further use" [CERT-VU].

This document updates the Digest Access Authentication scheme used by SIP to add support for SHA2 digest algorithms to replace the MD5 algorithm.

1.1 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].

2 The SIP Digest Authentication Scheme

This section describes the modifications to the operation of the Digest mechanism as specified in RFC3261 in order to support the SHA-256 and SHA-512/256 algorithms as described in [HTTP-DIGEST], and also to require support for the "qop" option."

2.1 Hash Algorithms

The Digest scheme has an ’algorithm’ parameter that specifies the algorithm to be used to compute the digest of the response. The IANA registry named "HTTP Digest Hash Algorithms" specifies the algorithms that correspond to ’algorithm’ values, and specifies a priority for each algorithm.

RFC3261 specifies only one algorithm, MD5, which is used by default. This document adds two new algorithms, to align with the [HTTP-DIGEST], that SHOULD be used instead of MD5: SHA-256 & SHA-512/256.

The priority of the algorithm defines its usage preference. UAs SHOULD prefer algorithms with higher priorities.

Note that [HTTP-DIGEST] defines a -sess variant for each algorithm; the -sess variants are not used with SIP.

2.2 Representation of Digest Values

The size of the digest depends on the algorithm used. The bits in the digest are converted from the most significant to the least significant bit, four bits at a time to the ASCII representation as follows. Each four bits is represented by its familiar hexadecimal notation from the characters 0123456789abcdef, that is binary 0000 is represented by the character ’0’, 0001 by ’1’ and so on up to the representation of 1111 as ’f’. If the MD5 algorithm is used to calculate the digest, then the digest will be represented as 32 hexadecimal characters, SHA-256 and SHA-512/256 by 64 hexadecimal characters.

2.3 The Authenticate Response Header

When a UAS receives a request from a UAC, and an acceptable Authorization header is not sent, the UAS can challenge the originator to provide credentials by rejecting the request with a 401/407 status code with the WWW-Authenticate/Proxy-Authenticate header field. The UAS MAY include multiple WWW-Authenticate/Proxy-

Authenticate headers to allow the UAS to utilize the best available algorithm supported by the client.

If the UAS challenges with multiple WWW-Authenticate/Proxy-Authenticate headers with the same realm, then each one of these headers MUST use a different digest algorithm. The UAS MUST add these headers to the response in the order that it would prefer to see them used, starting with the most preferred algorithm at the top, followed by the less preferred algorithms.

2.4 The Authorization Request Header

When the UAC receives a response with multiple headers with the same realm it SHOULD use the topmost header that it supports, unless a local policy dictates otherwise. The client should ignore any challenge it does not understand.

When the UAC receives a response with multiple headers with different realms it SHOULD provide all credentials that it possesses that match any one of the challenges.

If the UAC cannot respond to any of the challenges in the response, then it should abandon attempts to send the request; e.g., if the UAC does not have credentials for any of the realms.

2.5 Forking

RFC3261, section 22.3, discusses the operation of the proxy-to-user authentication, which describes the operation of the proxy when it forks a request. This section introduces some clarification to that operation.

If a request is forked, various proxy servers and/or UAs may wish to challenge the UAC. In this case, the forking proxy server is responsible for aggregating these challenges into a single response. Each WWW-Authenticate and Proxy-Authenticate value received in responses to the forked request MUST be placed into the single response that is sent by the forking proxy to the UA.

When the forking proxy places multiple WWW-Authenticate and Proxy-Authenticate header fields from one received response into the single response it MUST maintain the order of these header fields. The ordering of the header field values from the various proxies is not significant.

2.6 HTTP Modifications

This section describes the modifications and clarifications required to apply the HTTP Digest authentication scheme to SIP. The SIP scheme usage is similar to that for HTTP.

SIP clients and servers MUST NOT accept or request Basic authentication.

The rules for Digest authentication follow those defined in HTTP, with "HTTP/1.1" replaced by "SIP/2.0" in addition to the following differences:

1. The URI included in the challenge has the following BNF:

   URI = SIP-URI / SIPS-URI

2. The BNF for digest-uri-value is:

   digest-uri-value = Request-URI ; as defined in Section 25

3. The example procedure for choosing a nonce based on Etag does not work for SIP.

4. The text in [HTTP-DIGEST] regarding cache operation does not apply to SIP.

5. [HTTP-DIGEST] requires that a server check that the URI in the request line and the URI included in the Authorization header field point to the same resource. In a SIP context, these two URIs may refer to different users, due to forwarding at some proxy. Therefore, in SIP, a server MAY check that the Request-URI in the Authorization header field value corresponds to a user for whom the server is willing to accept forwarded or direct requests, but it is not necessarily a failure if the two fields are not equivalent.

6. As a clarification to the calculation of the A2 value for message integrity assurance in the Digest authentication scheme, implementers should assume, when the entity-body is empty (that is, when SIP messages have no body) that the hash of the entity-body resolves to the hash of an empty

string:

H(entity-body) = chosen-algorithm("")

For example, when the chosen algorithm is SHA-256, then:

H(entity-body) = SHA-256("") =
"e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855"

7. Servers MUST be able to properly handle "qop" parameter received in an authorization header field, and clients MUST be able to properly handle "qop" parameter received in WWW-Authenticate and Proxy-Authenticate header fields.

   Servers MUST always send a "qop" parameter in WWW-Authenticate and Proxy-Authenticate header field values, and clients MUST send the "qop" parameter in any resulting authorization header field.

The usage of the Authentication-Info header field continue to be allowed, since it provides integrity checks over the bodies and provides mutual authentication.

[OPEN ISSUE]

This section does NOT maintain backward compatibility with RFC 2069. Are there SIP servers and clients out there that support only RFC 2069 that would break because of this?

3 Augmented BNF for the SIP Protocol

This document updates the Augmented BNF for the SIP Protocol as follows.

It extends the request-digest as follows to allow for different digest sizes:

```
request-digest    =  LDQUOT *LHEX RDQUOT
```

The number of hex digits must be specified by the specification of the algorithm used.

It extends the algorithm parameter as follows to allow for SHA2 algorithms to be used:

```
algorithm =  "algorithm" EQUAL (
             <digest-alg-name-from-IANA-registry>
             / token )
```

4 Security Considerations

<Security considerations text>

5 IANA Considerations

The [HTTP-DIGEST] defines an IANA registry named "HTTP Digest Hash Algorithms" to simplify the introduction of new algorithms in the future. This document will use the algorithms defined in that registry.

6 Acknowledgments

<Acknowledgments text>

7 References

7.1 Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[HTTP-DIGEST] Shekh-Yusef, R., Ahrens, D., and Bremer, S., "HTTP Digest Access Authentication", Work in Progress, January 2014.

https://datatracker.ietf.org/doc/draft-ietf-httpauth-digest/

7.2 Informative References

Authors' Addresses

Rifaat Shekh-Yusef
Avaya
250 Sydney Street
Belleville, Ontario
Canada

Phone: +1-613-967-5267
Email: rifaat.ietf@gmail.com

SIPCORE Working Group
INTERNET-DRAFT
Intended Status: Standards Track
Expires: November 27, 2014

R. Shekh-Yusef, Ed.
Avaya
V. Pascual
Quobis

May 26, 2014

The Session Initiation Protocol (SIP) OAuth
draft-yusef-sipcore-sip-oauth-00

Abstract

The Session Initiation Protocol (SIP) uses a challenge-response framework to authenticate the user, but it does not have an authorization framework to control the user’s access to various services in the system.

This document defines an authorization framework for SIP that is based on the OAuth 2.0 framework.

Status of this Memo

Copyright and License Notice

Table of Contents

1 Introduction

The SIP protocol [RFC3261] uses the framework used by the HTTP protocol for authenticating users, which is a simple challenge-response authentication mechanism that allows a server to challenge a client request and allows a client to provide authentication information in response to that challenge.

The SIP protocol does not have an authorization framework to allow the system to control access to various services provided by the system.

OAuth 2.0 [RFC6749] defines a token based authorization framework to allow clients to access resources on behalf of their user. It also defines four types of authorization grants, which the client uses to request the access token.

This document defines a new authorization mechanism for SIP that is based on the OAuth 2.0 protocol. The new mechanism allows the proxy to avoid challenging every request from the client. The use of tokens is a Single Sing-On enabler, which allows for the definition of fine grained scopes that could be used by proxies and application servers to authorize clients to perform certain actions of behalf of the user but not others.

1.1 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].

2 OAuth 2.0 Roles

2.1 Resource Owner

An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.

2.2 Resource Server

The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

The Proxy will play this role with SIP.

2.3 Client

An application making protected resource requests on behalf of the resource owner and with its authorization. The term "client" does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices).

2.4 Authorization Server

The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

The Registrar will play this role with SIP when first-party authentication is used.

[[OPEN ISSUE]]

The rest of the document assumes that the Authorization Server plays two roles: the Authorization Endpoint role and the Token Endpoint role.

Do we need to consider separating them?

3 Authorization Code Grant type

3.1 Operations Overview

The following figure provides a high level view of flow of messages for the Authorization Code Grant type:

```
User                            Proxy                   Authorization
Agent                                                          Server
-----------------------------------------------------------------------
  |                               |                                |
  | REGISTER username@domain.com  |                                |
  |------------------------------>|                                |
  |                           302 |                                |
  |<------------------------------|                                |
  |                               |                                |
  | GET /authorize?response_type=code&...                          |
  |---------------------------------------------------------------->|
  |                               |                            401 |
  |<----------------------------------------------------------------|
  |                               |                                |
  |                               |                                |
o master-key = HMAC-SHA256(HA1, realm + nonce)                     |
  |                               |                                |
  | GET /authorize?response_type=code&... with credentials         |
  |---------------------------------------------------------------->|
  |                               |                                |
  |                               |                                |
  |                      o master-key=HMAC-SHA256(HA1, realm + nonce)
  |                               |                                |
  |                               |                     302 [code] |
  |<----------------------------------------------------------------|
  |                               |                                |
  |                               |                                |
  | REGISTER username@domain.com & code                            |
  |------------------------------>|                                |
  |                               | POST /token [code]             |
  |                               |------------------------------->|
  |                               |    200 OK [ token,             |
  |                               |             refresh token,     |
  |                               |             master-key]        |
  |                               |<-------------------------------|
  |                       200 OK  |                                |
  |<------------------------------|                                |
  |                               |                                |
  |                               |                                |
```

Subsequent Requests

```
  |                               |                               |
o pop = HMAC-SHA256(master-key, some-request-headers)             |
  |                               |                               |
  | INVITE pop                    |                               |
  |------------------------------>|                               |
  |                               |                               |
  |                               |                               |
  |               o The proxy verifies the pop.                   |
  |                               |                               |
  |                  180 Ringing  |                               |
  |<------------------------------|                               |
  |                               |                               |
```

Token Refresh

```
  |                               |                               |
  |                               | POST /token                   |
  |                               |   [ grant_type=refresh_token& |
  |                               |     refresh_token=<ref_token> |
  |                               |------------------------------>|
  |                               |      200 OK [ token,          |
  |                               |               refresh_token ] |
  |                               |<------------------------------|
  |                               |                               |
```

During registration the UA initially sends a REGISTER request without providing any credentials.

The registrar then redirect the UA by responding with 302 that includes the address of the Authorization Server.

The UA will then contact the Authorization Server without providing any credentials in the first request. The Authorization Server challenges the request using the Digest scheme, and the client retries the request and provide the user’s credentials.

The Authorization Server verifies the request from the client; if the verification is successful, the Authorization Server responds with 302 to redirect the UA back to the registrar and include a code in the body of the 302.

The UA then retries the request and include the code in the body of the request. The registrar then contacts the Authorization Server and exchanges the code for a token.

3.2 Registration

The UA initiates the process by sending a REGISTER request to the proxy. The proxy will redirect the UA to the Authorization Server by responding with 302 that include the address of the Authorization Server in the form of an HTTP URI.

The UA will then follow the authorization steps defined in section 3.3. At the end of the authorization process the UA will have a code that it will use to complete the registration process.

The UA will send a new REGISTER request and include the code in the body of the request with the following parameters:

grant_type (REQUIRED)

Value MUST be set to "authorization_code".

code (REQUIRED)

The authorization code received from the authorization server.

The proxy will then use the code to get a token from the Authorization Server as defined in section 3.4. If the proxy is able to obtain the token, the proxy will respond with 200 OK to the UA to complete the registration process.

3.3 Authorization

The UA constructs the initial request without providing any user credentials, but with the following URI parameters in the query component:

response_type (REQUIRED)

Value MUST be set to "code".

scope (OPTIONAL)

The scope of the access request as described by Section x.x.

state (RECOMMENDED)

The value of this parameter is a nonce created by the client to prevent replay attack. The nonce is a uniquely generated value for each request.

The Authorization Server challenges the request by responding with 401 with Digest scheme.

The UA will generate a master-key that is based on an HMAC-Hash algorithm, e.g. HMAC-SHA256, that takes an input the user’s HA1 and the concatenation of realm and nonce received in the challenge from the server.

The UA will then send a new authorization request, but this time include the credentials requested by the server. The UA will use the same parameters values used in the initial authorization request with the exception of the state parameter which will get a new nonce value.

When the server receives the request with the credentials, the server will verify the digest provided by the UA; if that is successful, the server will respond with 302 and include a code in the body of the response with the following parameters:

grant_type (REQUIRED)

Value MUST be set to "authorization_code".

code (REQUIRED)

The authorization code received from the authorization server.

The server then generates a master-key that is based on an HMAC-Hash algorithm, e.g. HMAC-SHA256, that takes an input the user’s HA1, and the concatenation of realm and nonce sent in the challenge to the client.

3.4 Acquiring Access Token

The proxy receives the REGISTER request that includes a body with a code, as specified in section 3.3. The proxy will then contact

the Authorization Server to exchange the code with a token.

The proxy sends a POST request to the Authorization Server and include the following parameters in the body:

grant_type (REQUIRED)

Value MUST be set to "authorization_code".

code (REQUIRED)

The authorization code received from the authorization server.

If the request is valid and authorized, the authorization server responds with a 200 OK to complete the registration process, with toke, token refresh, and the master-key in the body.

[[OPEN ISSUE]]

Should the proxy forward the tokens to the UA and expect the UA agent to provide the token with subsequent requests and take care of refreshing the token?

3.5 Token Refresh

The proxy makes a refresh request to the token by sending a refresh POST request that includes a body with the grant_type and the refresh_token.

For example:

grant_type=refresh_token&refresh_token=<some-token>

3.6 Authenticated Requests

When the UA wants to send any request to the proxy, it MUST include the Authorization header and use the MAC scheme to carry the proof-of-possession of the master-key.

The pop is calculated using the master-key as follows:

```
pop = HMAC-SHA256(master-key, some-request-headers)
```

The following is an example of an Authorization header with Bearer scheme:

```
Authorization: Bearer pop=<some-proof>
```

See rfc4474, section 9, for the SIP headers to hash to create the value for the proof.

[[OPEN ISSUE]]

The Bearer scheme is used to deliver tokens without providing any proof of possession. We probably need to use different scheme later on.

4 Resource Owner Password Credentials Grant type

4.1 Operations Overview

The following figure provides a high level view of flow of messages for the Resource Owner Password Credentials Grant type:

```
     UA                                                            Proxy
   ---------------------------------------------------------------------
     |                                                                 |
     | REGISTER username@domain.com                                    |
     |-----------------------------------------------------------------|
     |                                                                 |
     |                                 401 WWW-Authenticate: Digest    |
     |-----------------------------------------------------------------|
     |                                                                 |
     |                                                                 |
   o master-key = HMAC-SHA256(HA1, realm + nonce)                      |
     |                                                                 |
     | REGISTER username@domain.com with Authorization                 |
     |-----------------------------------------------------------------|
     |                                                                 |
     |                                                                 |
     |                      o master-key=HMAC-SHA256(HA1, realm + nonce)
     |                                                                 |
     |                                 200 OK [token, expires, ...]    |
     |-----------------------------------------------------------------|
     |                                                                 |
     |                                                                 |
   o pop = HMAC-SHA256(master-key, token + some-request-headers)       |
     |                                                                 |
     | INVITE token, pop                                               |
     |-----------------------------------------------------------------|
     |                                                                 |
     |                                   o The server verifies the pop.
     |                                                                 |
     |                                                   180 Ringing   |
     |-----------------------------------------------------------------|
     |                                                                 |
```

During registration the UA initially sends a REGISTER request without providing any credentials.

The registrar then challenges the UA by responding with 401 that includes the Digest scheme in the www-authenticate header.

The UA will generate a master-key that is based on an HMAC-Hash algorithm, e.g. HMAC-SHA256, that takes an input the user’s HA1 and the concatenation of realm and nonce received in the challenge from the server. The UA will continue to use the existing operation of handling the Digest challenge and then sends a new REGISTER request with the credentials to the server.

When the server receives the request with the credentials, the server will verify the digest provided by the UA; if that is successful, the server will accept the registration and include the details of the token in the response.

The server then generates a master-key that is based on an HMAC-Hash algorithm, e.g. HMAC-SHA256, that takes an input the user’s HA1, and the concatenation of realm and nonce sent in the challenge to the client.

At the end of the above process the UA would have registered with the proxy and both the UA and the registrar would have created the same master-key without sending the master key on the wire.

Later when the UA wants to send a request to the proxy it MUST always include the token and SHOULD use the master-key to hash the concatenation of the token and the following headers from the SIP request: See rfc4474, section 9.

The resulted hash will be included in the request as a proof-of-possession of the master-key.

4.2 Registration and Acquiring Tokens

The UA MUST request the access token during the registration process with the proxy, by including a body with the grant_type as "password". Initially, the UA sends a REGISTER request without providing any credentials.

The registrar MUST then challenge the UA by responding with 401 with the Digest scheme in the WWW-Authenticate header.

When the UA gets challenged by the proxy to provide its credentials, the UA MUST include its credentials in the new REGISTER request in the authorization header as it is done with the existing mechanism, and MUST include a body with the grant_type as "password".

In addition, the UA MUST generate a master-key as follows:

```
master-key = HMAC-SHA256(HA1, realm + nonce)
```

o HA1 - this is the user’s H(A1) as defined in [HTTP-DIGEST].

o realm - this is the realm that is returned by the server in the response to the initial request from the UA.

o nonce - this is the nonce that is returned by the server in the response to the initial request from the UA.

When the server receives the request with the credentials, the server will verify the digest provided by the UA; if that is successful, the server will accept the registration and include the details of the token in the response.

The server then generates a master-key following the same procedure followed by the client.

As a result of this procedure both the UA and the server would have created the same master-key without sending the master key on the wire.

4.3 Discarding Credentials

After successfully receiving the access and refresh tokens from the registrar, the UA SHOULD discard the user credentials.

4.4 Token Refresh

The UA makes a refresh request to the token by sending a refresh REGISTER request that includes the authorization header and a body with the grant_type, the refresh_token, and the proof-of-possession of the master-key.

For example:

grant_type=refresh_token&refresh_token=<some-token>&pop=<some-proof>

4.5 Authenticated Requests

When the UA wants to send any request to the proxy, it MUST include the Authorization header and use the Bearer scheme to carry the access token, and the proof-of-possession of the master-key. For example:

Authorization: Bearer token=<some-token>, pop=<some-proof>

See rfc4474, section 9, for the SIP headers to hash to create the value for the proof.

[[OPEN ISSUE]]

The Bearer scheme is used to deliver tokens without providing any proof of possession. We probably need to use different scheme later on.

4.6 Examples

```
REGISTER sip:registrar.biloxi.com SIP/2.0
Via: SIP/2.0/TCP bobspc.biloxi.com:5060;branch=z9hG4bKnashds7
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Bob <sip:bob@biloxi.com>;tag=456248
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER
Contact: <sip:bob@192.0.2.4>
Expires: 7200
Content-Length: 19

grant_type=password&pop=<some-proof>
```

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP bobspc.biloxi.com:5060;branch=z9hG4bKnashds7
     ;received=192.0.2.4
To: Bob <sip:bob@biloxi.com>;tag=2493k59kd
From: Bob <sip:bob@biloxi.com>;tag=456248
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER
Contact: <sip:bob@192.0.2.4>
Expires: 7200
Content-Length: 0

{
   "access_token":"2YotnFZFEjr1zCsicMWpAA",
   "token_type":"example",
   "expires_in":3600,
   "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
   "example_parameter":"example_value"
}
```

5 Client Credentials Grant

The following flow assumes that the UA is able to get a token using some out-of-band mechanism, and the UA wants to use the token to register, subscribe, and get service.

The flow uses a combination of the following from RFC6749:

- o Client Credentials Grant defined in section 4.4
- o Extensions Grants defined in section 4.5.

```
User                            Proxy                   Authorization
Agent                                                          Server
-----------------------------------------------------------------------
  |                               |                               |
  | REGISTER username@domain.com, token                           |
  |------------------------------>|                               |
  |                               |                               |
  |                               | POST /authorize               |
  |                               |     [ grant_type = <some-urn] |
  |                               |       token=<some-token> ]    |
  |                               |------------------------------>|
  |                               |                               |
  |                               |        200 OK                 |
  |                               |          [validity, services] |
  |                               |------------------------------>|
  |                               |                               |
  |                       200 OK  |                               |
  |<------------------------------|                               |
  |                               |                               |
  |                               |                               |
```

5.1 Registration

The UA is in possession of a token that was obtained through some out-of-band mechanism.

The UA sends a REGISTER request and include the token in the Authorization header using the Bearer scheme as defined in RFC6750.

If the proxy is able to verify the token, the proxy accepts the registration request and responds with 200 OK.

5.2 Authorization

When the proxy receives the REGISTER request with the token, the proxy will try to first validate the token before responding to the UA request.

The proxy sends a POST request and include the following parameters in the body of the request:

grant_type (REQUIRED)

Some well defined URN.

username (REQUIRED)

The resource owner username.

access_token (REQUIRED)

The token received from the UA.

scope (OPTIONAL)

The scope of the token.

If the authorization server is able to validate and authorize the request, it will respond with 200 OK with a body that contains the following parameters:

access_token, token_type, expires, refresh_token, scope

6 Security Considerations

<Security considerations text>

7 IANA Considerations

8 Acknowledgments

<Acknowledgments text>

9 References

9.1 Normative References

[KEYWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, May 2010.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

[HTTP-DIGEST] Shekh-Yusef, R., Ahrens, D., and Bremer, S., "HTTP Digest Access Authentication", Work in Progress, January 2014.

https://datatracker.ietf.org/doc/draft-ietf-httpauth-digest/

9.2 Informative References

Authors' Addresses

Rifaat Shekh-Yusef (Editor)
Avaya
250 Sydney Street
Belleville, Ontario
Canada

Phone: +1-613-967-5267
Email: rifaat.ietf@gmail.com

Victor Pascual
Quobis
Spain

Email: victor.pascual@quobis.com