INTERNET-DRAFT
Intended Status: Standard
Expires: January 2019

T. Herbert
Quantonium

July 24, 2018

Firewall and Service Tickets
draft-herbert-fast-02

Abstract

This document describes the Firewalls and Service Tickets protocol. A ticket is data that accompanies a packet and indicates a granted right to traverse a network or a request for network service to be applied. Applications request tickets from a local agent in the network and attach issued tickets to packets. Firewall tickets are issued to grant packets the right to traverse a network; service tickets indicate the desired service to be applied to a packets. A single ticket may provide both firewall and service ticket information. Tickets are sent in IPv6 Hop-by-Hop options.

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

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1 Introduction

Firewall and Service Tickets (FAST) is a technique to allow an application to signal to the network requests for admission and services for packets. A ticket is data that is attached to a packet by the source node, and it is then inspected and validated by certain intermediate nodes in a network. Tickets express a grant or right for packets to traverse a network or have services applied to them.

An application requests tickets for admission or services from a ticket agent in their local network. The agent issues tickets to the application which in turn attaches these to its packets. In the forwarding path, intermediate network nodes interpret tickets and apply requested services on packets.

Tickets are validated for authenticity by the network and contain an expiration time so that they cannot be easily forged. Tickets do not have a global interpretation, they can only be interpreted within the network that issues them. In order to apply services to inbound packets for a communication, remote peers reflect received tickets in packets they send without interpreting them. Tickets are stateless within the network, however they can be used to attain per flow semantics. Firewall and service tickets are are non-transferable and revocable.

Tickets are coded in IPv6 Hop-by-Hop options.

2 Motivation

This section presents the motivation for Firewall and Service Tickets.

2.1 Current mechanisms

Current solutions for controlling admission to the network and requesting services are mostly ad hoc and architecturally limiting.

2.1.1 Stateful firewalls and proxies

Stateful firewalls and proxies are the predominantly deployed techniques to control access to a network on a per flow basis. While they provide some benefits of security, they break the end-to-end model and have otherwise restricted the Internet in several ways:

- They require parsing over transport layer headers in the fast path of forwarding.

- They are limited to work only with a handful of protocols and

protocol features thereby ossifying protocols.

- They break the ability to use mulithoming and multipath. All packets for a flow must traverse a specific network device in both directions of a communication.

- They can break end to end security. NAT for instance breaks the TCP authentication option.

- They have created single points of failure and become network bottlenecks.

2.1.2 QoS signaling

In the current Internet, there is little coordination between hosts and the network to provide services based on characteristics of the application. Differentiated services provides an IP layer means to classify and manage traffic, however it is lacking in richness of expression and lacks a ubiquitous interface that allows applications to request service with any granularity. Without additional state, there is no means for the network infrastructure to validate that a third party application has requested QoS that adheres to network policies.

2.1.3 Deep packet inspection

Some network devices perform Deep Packet Inspection (DPI) into the application data to determine whether to admit packets or what services to apply. For instance, HTTP is commonly parsed to determine URL, content type, and other application level information the network is interested in. DPI can only be effective with the application layer protocols that a device is programmed to parse. More importantly, application level DPI is being effectively obsoleted in the network due the pervasive use of TLS. TLS interception and SSL inspection, whereby an intermediate node implements a proxy that decrypts a TLS session and re-encrypts, is considered a security vulnerability [TLSCERT].

2.2 Proposals for applications to signal the network

This section surveys some proposals to address the need for applications to signal the network.

2.2.1 SPUD/PLUS

SPUD (Session Protocol Underneath Datagrams) [SPUD] and its successor PLUS (Path Layer UDP Substrate) [PLUS] proposed a UDP based protocol to allow applications to signal a rich set of characteristics and

service requirements to the network.

SPUD had a number of drawbacks:

- SPUD is based on a specific protocol used over UDP. This requires applications to change to use a new protocol. In particular SPUD is incompatible with TCP which is the predominant transport protocol on the Internet.

- SPUD requires that intermediate nodes parse and process UDP payloads. Since UDP port numbers do not have global meaning [RFC7605] there is the possibility of misinterpretation and of silent data corruption if intermediate nodes modify UDP payloads. SPUD attempts to mitigate this issue with the use of magic numbers, however that can only ever be probabilistically correct.

- SPUD included stateful flow tracking in the network. This problematic because:

  - Not all communications have well defined connection semantics. For instance, a unidirectional data stream has no connection semantics at all.

  - Stateful network devices breaks multi-homing and multi-path; they assume that all packets of a flow in both directions are seen by the node doing tracking flow state. Stateful firewalls, for instance, require all packets for a flow to always go through the same device in both directions. This disallows flexibility and optimized traffic flow that a multi-homed network affords.

  - Maintaining per flow state in the network is an obvious scaling problem.

  - Keepalives to maintain a network state in a device, such as those sent to prevent a NAT state from being evicted, carry no useful information to the end user and in large numbers can become a source of congestion.

- The meta data information in SPUD would have global definition. This problematic because:

  - Application specific information could be leaked to unknown and untrusted parties.

  - Establishing a specification on what data should be conveyed in SPUD will be difficult. Different service providers may want different pieces of information, applications may also

have different ideas about what information is safe to make visible.

2.2.2 Path aware networking

Path aware networking (PAN) [PAN] is an IRTF effort to allow applications to select paths through the Internet for their traffic. The idea is that in choosing different paths an application can select from the different characteristics that are associated with different paths. Path aware networking requires a means to express the various paths and their associated characteristics to applications. In the data path, a method to express desired path for a packet is needed-- this presumably could be by a mechanism such as segment routing.

PAN and FAST have similar characteristics, particularly with respect to the need for applications and networks to communicate about network path characteristics. However, where PAN presumably endeavors to allow path selection by an application, FAST allows applications to select their desired path characteristics and it is up to the network to select the actual path. This distinction is important to maximize flexibility, especially in situations where providing any detailed path information to untrusted end device is a security risk (which would be the typical case in a provider network or open Internet).

2.3 Emerging use cases

In a typical client/server model of serving content, end host clients communicate with servers on the Internet. Clients are typically user devices that are connected to the Internet through a provider network. In the case of mobile devices, such as smart phones, the devices are connected to the Internet through a mobile provider network. Content providers (web servers and content caches) tend to be more directly connected to the Internet, the largest of which can connect at exchange points.

Provider networks can be architected to provide different services and levels of services to their users based on characteristics of applications. For example, a mobile carrier network can provide different latency and throughput guarantees for different types of content. A network may offer different services for optimizing video: streaming an HD movie might need high throughput but not particularly low latency; a live video chat might have lower throughput demands but have stringent low latency requirements.

The emerging 3GPP standard for 5G defines a set of mechanisms to provide a rich array of services for users. These mechanisms employ

Network Function Virtulization (NFV), Service Function Chaining (SFC), and network slices that divide physical network resources into different virtualized slices to provide different services. To make use of these mechanisms, the applications running in UEs (User Equipment) will need to indicate desired services of the RAN (Radio Access Network). For instance, a video chat application may request bounded latency that is implemented by the network as a network slice; so packets sent by the application should be mapped to that network slice.

Note that an application service applies to both packets sent by an end node and those sent from a peer towards the end node. For the latter case, the network needs to be able to map packets sent from hosts on the Internet to the services requested by the receiving application.

3 Architecture

The figure below illustrates an example network path between two hosts on the Internet. Each host connects to the Internet via a provider network, and provider networks are connected in the Internet by transit networks.

```
                                _____
               __________      (     )      __________
+--------+    (          )    (       )    (          )    +--------+
| User 1 +---( Provider A )--( Transit )--( Provider B )---+ User 2 |
+--------+    (__________)    (       )    (__________)    +--------+
                               (_____)
```

Figure 1

Within each provider network, services may be provided on behalf of the users of the network. In the figure above, Provider 1 may provide services and service agreements for users in its network including User 1; and likewise Provider B can provide services to users in its network including User 2. Transit networks service all users and don't typically provide user specific services or service differentiation.

Services provided by different provider networks may be very different and dependent on the implementation of the network as well as the policies of the provider.

Based on this model, services and service differentiation can be considered local to each network provider. FAST is a mechanism whereby each user and application can request from its local provider the services to be applied to its traffic. A request for service is made to a FAST "ticket agent". The contents of the request describe

the services that application desires. The ticket agent responds with a "ticket" that the application sets in its packets. When a packet is sent by the application with a ticket attached, the ticket is interpreted in the provider network to allow the packet to traverse the network and to map the packet to the appropriate services. The ticket is only relevant to the provider network that issued the ticket, to the application itself and nodes outside of the provider network the ticket is an uninterpretable opaque object.

To facilitate network traversal and service mapping in the reverse direction for a flow, that is packets sent from a peer host, peer hosts reflect tickets without modification or interpretation. This is done by saving the ticket received in packets of a flow and attaching that as a reflected ticket to packets being sent on the flow.

The use of tickets may be bilateral for a flow so that each peer requests service from its local network. Therefore packets may contain two types of tickets: one that is set by the sending host to signal its local provider network, and the other is the reflected ticket that is a signal to the provider network of the peer endpoint.

Tickets are scoped values, they only have meaning in the network in which they were issued. The format, meaning, and interpretation of tickets is network specific. By mutual agreement, two networks may share the policy and interpretations of tickets. For instance, there could be an agreement between two provider networks to interpret each others tickets or to use a common format.

3.1 Example packet flow

Referencing the diagram in figure 1, consider that User 1 is establishing a video chat with User 2 and wishes to have low latency service for video applied by its local network (Provider 1). The flow of events may be:

1. User 1 makes a ticket request to a ticket agent of Provider A that describes the video application and may include detailed characteristics such as resolution, frame rate, latency, etc.

2. The ticket agent issues a ticket to User 1 that indicates that packets of the flow have a right to traverse the network and the services to be applied to the packets of the flow.

3. The video chat application sends packets with the ticket attached for the video chat.

4. The first hop node in Provider A's network interprets the ticket in packets and applies the appropriate services (e.g.

sets diffserv, forwards on a network slice, encapsulates in MPLS, etc.).

5. Packets traverse Provider A's network with the appropriate services being applied.

6. Packets traverse transit networks and Provider B network, the attached tickets are ignored.

7. Packets are received at User 2. Attached tickets are saved in the context of the flow for the video chat.

8. User 2's video chat application sends packets to User 1. The last ticket previously received from User 1 is now reflected in these packets.

9. Packets traverse Provider B network and transit networks, the reflected ticket is ignored.

10. An ingress node in Provider A's network interprets the reflected ticket and applies appropriate services to the packets for traversing the local network.

11. Packets are forwarded within Provider's A network with the appropriate services applied. No other intermediate nodes should need to interpret the tickets.

12. Packets are received at the host for User 1. The reflected ticket is validated by comparing the received ticket with that being sent for the flow. It the ticket is determined valid then the packet is accepted.

3.2 Requirements

The requirements for Firewall and Service Tickets are:

- Tickets MUST be stateless within the network. In particular intermediate nodes MUST NOT be required to create and maintain state for transport layer connections.

- Tickets MUST work in a multi-homed and multi-path environments.

- Outside of the network that issued a ticket, tickets MUST be opaque and obfuscated so that no application specific information is derivable.

- Tickets MUST work with any transport protocol as well as in the presence of any IP protocol feature (e.g. other extension

headers are present).

- o Tickets SHOULD minimize the changes to an application. Their use should be an "add-on" to the existing communications of an application.

- o Tickets MUST deter spoofing and other misuse that might result in illegitimate use of network services or denial of service attack.

- o Tickets MUST be contained in the IP layer protocol. In particular, tickets MUST NOT require parsing transport layer headers.

- o Tickets MUST allow services to be applied in the return path of a communication. In a client/server application it is often the packets in the reverse path that require the most service (for instance if a video is being streamed to a client).

- o A fallback MUST be present to handle the case that extension headers are dropped within the network or a peer node does not reflect tickets. A fallback allows functional communications but provides it in a degrade mode of service.

4 Packet format

A ticket is sent in a Hop-by-Hop option.

4.1 Option format

The format of an Hop-by-Hop option containing a ticket is:

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Option Type  |  Opt Data Len | Type  |                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                       +
|                                                               |
~                            Ticket                             ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fields:

- o Option type: Type of Hop-by-Hop option. This document proposes two possible values for ticket: an unmodifiable and a modifiable variant.

- Opt Data Len: Length of the option data field. This is two plus the length of the ticket field

- Type: Indicates the type and reflection property of the ticket. Possible values are:

  - 0x0: Ticket from origin, don't reflect at receiver

  - 0x1: Ticket from origin, reflect at receiver

  - 0x2: Reflected ticket

  - 0x3-0xf: Reserved

4.2 Option types

The are two option numbers requested for the ticket option: 0x0F and 0x2F. The latter allows modification. This would be used in situations where the network nodes needed to modify the option with new information.

4.3 Ticket format

A ticket encodes service parameters that describe the desired services as well as additional fields that would be used to provide privacy and integrity.

The format of a ticket is defined by the network in which the ticket is issued. A ticket should be obfuscated or encrypted for privacy so that only the local network can interpret it. It should be uniterpretable to any nodes outside the network and to the application or host that is granted a ticket. It should be resistant to spoofing so that an attacker cannot illegitimately get service by applying a ticket seen on other flows.

It is recommended that tickets are encrypted and each ticket has an expiration time. For instance, a ticket may be created by encrypting the ticket data with an expiration time and using the source address, destination address, and a shared key as the key for encryption.

For example, a ticket with an expiration time may have the format:

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Expiration time                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                    Service parameters                         ~
```

```
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Where the expiration time is in a format understood by the local network nodes which maintain synchronized time. The Service parameters are relevant to network nodes and describe the services to be applied. The service parameters could simply be a set of flags for services, an index to a service profile known by the network nodes, or possibly have more elaborate structure that could indicate numerical values for characteristics that have a range. The service parameters could also include a type field to allow a network to define different representations of service parameters.

A simple ticket containing a service protocol index might be:

```
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                       Expiration time                         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Type  |            Service Profile Index                      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Where Type indicates the type of ticket and in this case indicates it is a service profile index. Service Profile Index could be an index into a table that describes the services to be applied.

5 Operation

5.1 Ticket types and ordering

There are three types of tickets that may be contained in a packet:

- origin tickets that are not reflected

- origin tickets to be reflected

- reflected tickets

Origin tickets are those set by an application that was issued a ticket and have an additional property indicating whether they are to be reflected by a peer host. Reflected tickets are those that have been received and reflected by a peer host.

A sender SHOULD set at most one of each type of ticket in a packet. If different types of ticket are sent within a single packet they SHOULD have the following ordering:

1. origin tickets that are not reflected

2. origin tickets to be reflected

3. reflected tickets

If a packet contains more than one ticket option with the same ticket type, only the first option appearing in the list for the ticket type is processed. Additional options for the same ticket type are parsed but not processed.

5.2 Origin application processing

An origin application requests tickets, sets them in packets, and validates reflected tickets.

5.2.1 Ticket requests

An application that wishes to use network services first requests tickets from a ticket agent. The application request could be in the form of an XML structure with a canonical elements (the definition is outside the scope of this document). The application makes a request to the ticket agent for the local network. This could be done via a web service using REST APIs. Internally in the host, the ticket agent might be accessed through a library that interfaces to a ticket daemon that in turn arbitrates requests between the applications and a ticket agent in the network.

An issued ticket is opaque to the application and the application should not attempt to interpret it or take any other action other than attaching the ticket to its packets.

A ticket agent MAY provide both a origin ticket not to be reflected and one that is to be reflected. The intent is that different tickets can be used between the outbound and inbound paths for the flow. In the case that two tickets are provided, the origin ticket not to be reflected MUST appear first in the options list.

5.2.2 Ticket identification

Tickets are valid for a specific IP source and destination address for which they were issued. Transport layer ports and other transport layer information are not included ticket identification, however an application can request tickets and validate reflected tickets on a per flow basis. Issued tickets are stored in the flow context and the saved information is used to validate reflected tickets.

5.2.3 Ticket use

When the ticket agent issues an returns a ticket, the application

sets the ticket as a Hop-by-Hop option. This is typically done by setting a socket option on a socket (in the case of TCP) or by indicating the option in the ancillary data when sending on a unconnected socket (in the case of UDP). The application SHOULD continue to use the same ticket for the flow until it is updated with a new ticket.

The ticket agent SHOULD return an expiration time with the ticket. An application can use the ticket until the expiration time, at which point it can request a new ticket to continue communications. In order to make the ticket transition process seamless an application MAY request a new ticket before the old one expires.

A host SHOULD set in a packet at most one origin ticket not to be reflected and one origin ticket to be reflected.

5.2.4 Ticket scope

Tickets are valid for a specific IP source and destination address for which they were issued. Transport layer ports and other transport layer information are not included ticket identification, however an application can request tickets and validate reflected tickets on a per flow basis. Issued tickets are stored in the flow context and the saved information is used to validate reflected tickets.

A network MAY delegate creation of tickets to hosts in a limited fashion. This would entail the network ticket agent issuing a master ticket to a host ticket agent which in turn can use the master ticket to create a limited number of tickets. The details of ticket agent delegation are outside the scope of this document.

5.3 Origin network processing

When a packet with a ticket enters a network, a network node can determine if the ticket originated in its network and must be processed. This is done by considering the origin of the ticket and the source or destination IP address. For an origin ticket (i.e. ticket is not reflected), the source address is considered. If the source address is local to the network then the ticket can be interpreted. For a reflected ticket, the destination address is considered. If the destination address is local to the network then the ticket can be interpreted.

If a ticket origin is determined to be the local network then the ticket is processed. The ticket is decrypted if necessary and the expiration time is checked. If the ticket is verified to be authentic and valid then the packet is mapped to be processed by the requested services. For instance, in a 5G network the packet may be forwarded

on a network slice for the characteristics the application has requested (real-time video for instance).

If an origin ticket cannot be verified, for instance the ticket cannot be authenticated, then the ticket SHOULD be ignored and the packet processed as though no ticket were present.

Note that there are logically only two ingress points into the network at which a provider needs to process tickets: when a local user sends a packet into the provider network with an origin ticket, and when a packet from an external network enters the provider's network with a reflected ticket. Any ticket should be processed at most once within a network. Once a ticket is processed and mapped to the network's service mechanisms it should not need further examination.

If there is more than one origin ticket present, then the first one encountered is processed and any additional origin tickets SHOULD be ignored by a network node. Note that this will be the case if a ticket agent issued both a origin ticket not to be reflected and one to be reflected; the ticket not to be reflected should appear first in the packet and thus would be the one processed by the node.

If there is more than one reflected ticket present, then the first one encountered is processed and any additional reflected tickets SHOULD be ignored.

5.4 Peer host processing

When a host receives a packet with a ticket whose type is "from origin and needs to be reflected", it SHOULD save the ticket in its flow context and reflect it on subsequent packets. When the application reflects the option, it copies the whole option and only modifies the type to indicate a "reflected ticket". The application SHOULD continue to reflect the ticket until a different one is received from the origin or a packet without a service ticket option is received on the flow. Note that the latest ticket that is received is the one to be reflected, if packets have been received out of order for a flow it is possible that the reflected ticket is from an earlier packet in a flow.

If there is more than one origin ticket to be reflected present, then the first one encountered is processed and any additional origin tickets to be reflected SHOULD be ignored.

A peer host MUST ignore received origin tickets that are not to be reflected.

5.5 Processing reflected tickets

5.5.1 Network processing

When a packet with a reflected ticket enters the origin network of the ticket, the ticket MUST be processed. The ticket is validated. Validation entails decoding or decrypting the ticket and checking the expiration time. If the ticket is valid and has not expired time then the packet is verified for forwarding.

A network MAY accept expired reflected tickets for some configurable period after the expiration time. Rate limiting SHOULD be applied to packets with expired reflected tickets. Accepting expired tickets is useful in the case that a connection goes idle and after sometime the remote peer starts to send. The ticket it reflects may be expired and presumably the receiving host will quickly respond with a new ticket.

5.5.2 Host processing

Upon receiving a packet with a reflected ticket an end host MUST validate the ticket before accepting the packet. This verification is done by comparing the received ticket to that which is set to be sent on the corresponding flow. If the tickets do not match then the packet is dropped and the event SHOULD be logged.

A host SHOULD retain and validate expired tickets that are reflected to allow a peer time to receive and reflect an updated ticket.

5.6 Handling dropped extension headers

The downside of using IPv6 extension headers on the Internet is that they are currently not completely reliable. Some intermediate nodes will drop extension headers with rates described in [RFC7872].

5.6.1 Mitigation for dropped extension headers

There are some mitigating factors for this problem:

- A provider network that implements tickets would need to ensure that extension headers are at least usable within their network.

- Transit networks are less likely to arbitrarily drop packets with extension headers.

- Many content providers, especially the larger ones, may be directly connected to the Internet. For example, front end web servers may be co-located as exchange points.

- o The requirement that nodes must process Hop-by-hop options has been relaxed in [RFC8200]. It is permissible for intermediate nodes to ignore them.

- o Increased deployment of IPv6 and viable use cases of extension headers, such as described here, may motivate vendors to fix issues with extension headers.

5.6.2 Fallback for dropped extension headers

Since the possibility that extension headers are dropped cannot be completely eliminated, a fallback is included for use with tickets.

When an application connects to a new destination for which it has no history about the viability of extension headers, it can perform a type of Happy Eyeballs probing. The concept is for a host to send a number of packets with and without tickets. The application can observe whether packets with tickets are being dropped or not being reflected.

There are a few possible outcomes of this process:

- o A packet with a ticket is dropped and an ICMP for extension headers [ICMPEH] processing limits is received. This is a strong signal that extension headers are not viable to the destination and should not be used for the flow.

- o A packet with a ticket is dropped and no ICMP error is received. This is a signal that extension headers may not be usable. If such drops are observed for all or a significant fraction of packets and there are no drops for packets that were sent without tickets, then extension headers should be considered not viable for the flow.

- o Packets with tickets are not being dropped, however tickets are not being reflected. This is a signal that the peer application does not support reflection. Tickets may be sent, however they are only useful in the outbound path.

- o Packets with tickets are not being dropped and tickets are properly being reflected. Tickets are useful in both directions.

If extension headers are found to not be viable or tickets are not being properly reflected, a possible fallback is to not use tickets. In this case, communications might remain functional, however they would be operate in a degraded mode of service. The network may fallback to creating per flow state in the network; the ticket that an application sent with packets during probing could be used to

instantiate the service characteristics maintained in a flow state.

6 Implementation considerations

6.1 Origin applications

Existing client applications can be modified to request tickets and set them in packets. The kernel may need some small changes or configuration to enable an application to specify the option for its packets.

The interface to the ticket agent would likely be via a library API.

Using the BSD sockets interface, for a connected socket (TCP, SCTP, or connected UDP socket) a Hop-by-Hop option can be set on the socket via the setsockopt system call. For an unconnected socket (UDP) the ticket option can be set as ancillary data in the sendmsg system call.

Happy Eyeballs for extension headers, described in section 5.6.2, could be implemented in the kernel for a connection oriented transport protocol such a TCP. For connectionless protocols, probing could be handled by an application library.

6.2 Ticket reflection

To perform ticket reflection, servers must be updated. In the case of a connected socket (TCP, SCTP, or a connected UDP socket) this can be done as relatively minor change to the kernel networking stack which would be transparent to applications. For unconnected UDP, an application could receive the ticket as part of the ancillary data in recvmsg system call, and then send the reflected ticket in a reply using ancillary data in sendmsg.

7 Security Considerations

There are two main security considerations:

- o Leakage of content specific information to untrusted third parties must be avoided.

- o Tickets cannot be forged or illegitimately used.

Tickets may be visible to the Internet including untrusted and unknown networks in the path of sent packets. Therefore, tickets should be encrypted or obfuscated by the origin network.

Tickets need to have an expiration time, must be resistant to

forgery, and must be nontransferable. A ticket should be valid for the specific source and destination addresses that it was issued for. Tickets are revocable by implemented a black-list contained revoked tickets.

8 IANA Considerations

IANA is requested to assigned the following Hop-By-Hop options:

Hex Value	Binary value	Description	Reference
	act chg rest		
0x0F	00 0 01111	Firewall and Service Ticket	This document
0x2F	00 1 01111	Modifiable Firewall and Service Ticket	This document

IANA is requested to set up a registry for the Ticket types. These types are 4 bit values. New values for control types 0x3-0xf are assigned via Standards Action [RFC5226].

Ticket type	Description	Reference
0x0	Ticket from origin and don't reflect	This document
0x1	Ticket from origin and reflect	This document
0x2	Reflected ticket	This document

9 References

9.1 Normative References

[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>.

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

9.2 Informative References

[RFC7605] Touch, J., "Recommendations on Using Assigned Transport Port Numbers", BCP 165, RFC 7605, DOI 10.17487/RFC7605, August 2015, <http://www.rfc-editor.org/info/rfc7605>.

[RFC7872] Got, 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>.

[TLSCERT] United States Computer Emergency Readiness Team (US-CERT), "Alert (TA17-075A), HTTPS Interception Weakens TLS Security, March 2017

[SPUD] Hildebrand, J. and Trammell, B., "Substrate Protocol for User Datagrams (SPUD) Prototype", draft-hildebrand-spud-prototype-03, March 2015

[PLUS] Trammell, B. and Kuehlewind, M., "Path Layer UDP Substrate Specification", draft-trammell-plus-spec-01, March 2017

[PAN] Trammell, B., "Open Questions in Path Aware Networking", draft-trammell-panrg-questions-02, December 2017

[ICMPEH] Herbert, T., "ICMPv6 errors for discarding packets due to processing limits", draft-herbert-6man-icmp-limits-00.txt

Author's Address

Tom Herbert
Quantonium
Santa Clara, CA
USA

Email: tom@quantonium.net