Stream: Internet Engineering Task Force (IETF)
RFC: 9149
Category: Standards Track
Published: April 2022
ISSN: 2070-1721
Authors: T. Pauly, *Apple Inc.*; D. Schinazi, *Google LLC*; C.A. Wood, *Cloudflare*

RFC 9149
TLS Ticket Requests

Abstract

TLS session tickets enable stateless connection resumption for clients without server-side, per-client state. Servers vend an arbitrary number of session tickets to clients, at their discretion, upon connection establishment. Clients store and use tickets when resuming future connections. This document describes a mechanism by which clients can specify the desired number of tickets needed for future connections. This extension aims to provide a means for servers to determine the number of tickets to generate in order to reduce ticket waste while simultaneously priming clients for future connection attempts.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9149.

Copyright Notice

Table of Contents

1. Introduction
   1.1. Requirements Language
2. Use Cases
3. Ticket Requests
4. IANA Considerations
5. Performance Considerations
6. Security Considerations
7. References
   7.1. Normative References
   7.2. Informative References

Acknowledgements

Authors' Addresses

1. Introduction

As described in [RFC8446], TLS servers vend clients an arbitrary number of session tickets at their own discretion in NewSessionTicket messages. There are at least three limitations with this design.

First, servers vend some (often hard-coded) number of tickets per connection. Some server implementations return a different default number of tickets for session resumption than for the initial connection that created the session. No static choice, whether fixed or dependent upon resumption, is ideal for all situations.

Second, clients do not have a way of expressing their desired number of tickets, which can impact future connection establishment. For example, clients can open parallel TLS connections to the same server for HTTP, or they can race TLS connections across different network interfaces. The latter is especially useful in transport systems that implement Happy Eyeballs [RFC8305]. Since

clients control connection concurrency and resumption, a standard mechanism for requesting more than one ticket is desirable for avoiding ticket reuse. See Appendix C.4 of [RFC8446] for discussion of ticket reuse risks.

Third, all tickets in the client's possession ultimately derive from some initial connection. Especially when the client was initially authenticated with a client certificate, that session may need to be refreshed from time to time. Consequently, a server may periodically force a new connection even when the client presents a valid ticket. When that happens, it is possible that any other tickets derived from the same original session are equally invalid. A client avoids a full handshake on subsequent connections if it replaces all stored tickets with new ones obtained from the just-performed full handshake. The number of tickets the server should vend for a new connection may therefore need to be larger than the number for routine resumption.

This document specifies a new TLS extension, "ticket_request", that clients can use to express their desired number of session tickets. Servers can use this extension as a hint for the number of NewSessionTicket messages to vend. This extension is only applicable to TLS 1.3 [RFC8446], DTLS 1.3 [RFC9147], and future versions of (D)TLS.

1.1. Requirements Language

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Use Cases

The ability to request one or more tickets is useful for a variety of purposes:

Parallel HTTP connections: To improve performance, a client may open parallel connections. To avoid ticket reuse, the client may use distinct tickets on each connection. Clients must therefore bound the number of parallel connections they initiate by the number of tickets in their possession or risk ticket reuse.

Connection racing: Happy Eyeballs V2 [RFC8305] describes techniques for performing connection racing. The Transport Services Implementation document [TAPS] also describes how connections can race across interfaces and address families. In such cases, clients may use more than one ticket while racing connection attempts in order to establish one successful connection. Having multiple tickets equips clients with enough tickets to initiate connection racing while avoiding ticket reuse and ensuring that their cache of tickets does not empty during such races. Moreover, as some servers may implement single-use tickets, distinct tickets prevent premature ticket invalidation by racing.

Less ticket waste: Currently, TLS servers use application-specific, and often implementation-specific, logic to determine how many tickets to issue. By moving the burden of ticket count to clients, servers do not generate wasteful tickets. As an example, clients might only request one ticket during resumption. Moreover, as ticket generation might involve expensive computation, e.g., public key cryptographic operations, avoiding waste is desirable.

Decline resumption: Clients can indicate they do not intend to resume a connection by sending a ticket request with count of zero.

3. Ticket Requests

As discussed in Section 1, clients may want different numbers of tickets for new or resumed connections. Clients may indicate to servers their desired number of tickets to receive on a single connection, in the case of a new or resumed connection, via the following "ticket_request" extension:

```
enum {
    ticket_request(58), (65535)
} ExtensionType;
```

Clients **MAY** send this extension in ClientHello. It contains the following structure:

```
struct {
    uint8 new_session_count;
    uint8 resumption_count;
} ClientTicketRequest;
```

new_session_count: The number of tickets desired by the client if the server chooses to negotiate a new connection.

resumption_count: The number of tickets desired by the client if the server is willing to resume using a ticket presented in this ClientHello.

A client starting a new connection **SHOULD** set new_session_count to the desired number of session tickets and resumption_count to 0. Once a client's ticket cache is primed, a resumption_count of 1 is a good choice that allows the server to replace each ticket with a new ticket without over-provisioning the client with excess tickets. However, clients that race multiple connections and place a separate ticket in each will ultimately end up with just the tickets from a single resumed session. In that case, clients can send a resumption_count equal to the number of connections they are attempting in parallel. (Clients that send a resumption_count less than the number of parallel connection attempts might end up with zero tickets.)

When a client presenting a previously obtained ticket finds that the server nevertheless negotiates a new connection, the client **SHOULD** assume that any other tickets associated with the same session as the presented ticket are also no longer valid for resumption. This includes tickets

obtained during the initial (new) connection and all tickets subsequently obtained as part of subsequent resumptions. Requesting more than one ticket when servers complete a new connection helps keep the session cache primed.

Servers **SHOULD NOT** send more tickets than requested for the connection type selected by the server (new or resumed connection). Moreover, servers **SHOULD** place a limit on the number of tickets they are willing to send, whether for new or resumed connections, to save resources. Therefore, the number of NewSessionTicket messages sent will typically be the minimum of the server's self-imposed limit and the number requested. Servers **MAY** send additional tickets, typically using the same limit, if the tickets that are originally sent are somehow invalidated.

A server that supports and uses a client "ticket_request" extension **MUST** also send the "ticket_request" extension in the EncryptedExtensions message. It contains the following structure:

```
struct {
    uint8 expected_count;
} ServerTicketRequestHint;
```

expected_count: The number of tickets the server expects to send in this connection.

Servers **MUST NOT** send the "ticket_request" extension in any handshake message, including ServerHello or HelloRetryRequest messages. A client **MUST** abort the connection with an "illegal_parameter" alert if the "ticket_request" extension is present in any server handshake message.

If a client receives a HelloRetryRequest, the presence (or absence) of the "ticket_request" extension **MUST** be maintained in the second ClientHello message. Moreover, if this extension is present, a client **MUST NOT** change the value of ClientTicketRequest in the second ClientHello message.

4. IANA Considerations

IANA has added the following entry to the "TLS ExtensionType Values" registry [RFC8446] [RFC8447]:

Value	Extension Name	TLS 1.3	DTLS-Only	Recommended
58	ticket_request	CH, EE	N	Y

Table 1: Addition to TLS ExtensionType Values Registry

5. Performance Considerations

Servers can send tickets in NewSessionTicket messages any time after the server Finished message (see Section 4.6.1 of [RFC8446]). A server that chooses to send a large number of tickets to a client can potentially harm application performance if the tickets are sent before application data. For example, if the transport connection has a constrained congestion window, ticket messages could delay sending application data. To avoid this, servers should prioritize sending application data over tickets when possible.

6. Security Considerations

Ticket reuse is a security and privacy concern. Moreover, clients must take care when pooling tickets as a means of avoiding or amortizing handshake costs. If servers do not rotate session ticket encryption keys frequently, clients may be encouraged to obtain and use tickets beyond common lifetime windows of, e.g., 24 hours. Despite ticket lifetime hints provided by servers, clients **SHOULD** dispose of cached tickets after some reasonable amount of time that mimics the session ticket encryption key rotation period. Specifically, as specified in Section 4.6.1 of [RFC8446], clients **MUST NOT** cache tickets for longer than 7 days.

In some cases, a server may send NewSessionTicket messages immediately upon sending the server Finished message rather than waiting for the client Finished message. If the server has not verified the client's ownership of its IP address, e.g., with the TLS cookie extension (see Section 4.2.2 of [RFC8446]), an attacker may take advantage of this behavior to create an amplification attack proportional to the count value toward a target by performing a (DTLS) key exchange over UDP with spoofed packets. Servers **SHOULD** limit the number of NewSessionTicket messages they send until they have verified the client's ownership of its IP address.

Servers that do not enforce a limit on the number of NewSessionTicket messages sent in response to a "ticket_request" extension could leave themselves open to DoS attacks, especially if ticket creation is expensive.

7. References

7.1. Normative References

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8447] Salowey, J. and S. Turner, "IANA Registry Updates for TLS and DTLS", RFC 8447, DOI 10.17487/RFC8447, August 2018, <https://www.rfc-editor.org/info/rfc8447>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/info/rfc9147>.

7.2. Informative References

[RFC8305] Schinazi, D. and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency", RFC 8305, DOI 10.17487/RFC8305, December 2017, <https://www.rfc-editor.org/info/rfc8305>.

[TAPS] Brunstrom, A., Ed., Pauly, T., Ed., Enghardt, T., Tiesel, P., and M. Welzl, "Implementing Interfaces to Transport Services", Work in Progress, Internet-Draft, draft-ietf-taps-impl-12, 7 March 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-taps-impl-12>.

Acknowledgements

The authors would like to thank David Benjamin, Eric Rescorla, Nick Sullivan, Martin Thomson, Hubert Kario, and other members of the TLS Working Group for discussions on earlier draft versions of this document. Viktor Dukhovni contributed text allowing clients to send multiple counts in a ticket request.

Authors' Addresses

Tommy Pauly
Apple Inc.
One Apple Park Way
Cupertino, CA 95014
United States of America
Email: tpauly@apple.com

David Schinazi
Google LLC
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States of America
Email: dschinazi.ietf@gmail.com

Christopher A. Wood
Cloudflare
101 Townsend St
San Francisco, CA 94107
United States of America
Email: caw@heapingbits.net