Network Working Group
Internet-Draft
Expires: December 20, 2010

A. Langley
Google Inc

June 18, 2010

Transport Layer Security (TLS) Snap Start
draft-agl-tls-snapstart-00

Abstract

This document describes a Transport Layer Security (TLS) extension for eliminating the latency of handshakes when the client has prior knowledge about the server. Unlike resumption, this prior knowledge is not secret and may be obtained from third parties and stored on disk for significant periods of time.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on December 20, 2010.

Copyright Notice

Table of Contents

1. Introduction

Snap Start aims to remove the latency overhead of TLS handshakes in the case that the application protocol involves the client speaking first. Currently, TLS handshaking imposes additional latency and is costly for time-sensitive applications.

In order to achieve this, the initial flow from the client must contain application data and, therefore, everything needed for the server to complete a handshake and process it. Starting from this premise we can derive the essential features of Snap Start.

2. Design

At first we are only considering the case of a full handshake. Assume, for the sake of argument, a client that is capable of predicting the contents of a server's first handshake flow (i.e. the "ServerHello" message through to the "ServerHelloDone"). Such a client could send "ClientKeyExchange", "ChangeCipherSpec" and "Finished" messages immediately following its "ClientHello". It would then be able to transmit application data records and we have successfully eliminated the TLS handshake latency.

However, several elements of the server's first handshake flow are unpredictable. Fundamentally, any ephemeral Diffie-Hellman based cipher suite is incompatible with Snap Start so the following assumes that the server is using non-ephemeral key agreement.

The chosen cipher suite, compression method, supported extensions, ordering of those extensions, certificate etc are all somewhat unpredictable for a given server but are highly correlated across time. Given a previous handshake with the same server, assuming that it will make the same choices when presented with a similar "ClientHello" is sufficiently accurate to expect a very high rate of prediction of these elements of the handshake.

The server's chosen session id is unpredictable. However, this can be eliminated by using Session Tickets [RFC5077]. When Session Tickets are in use the "ServerHello" doesn't include a session id and the "NewSessionTicket" message itself is not part of the first flow.

Lastly, the "server_random" is unpredictable. The "server_random" exists to provide uniqueness and freshness. When the server picks a random value it can be assured that no previous TLS connection has ever used the same value. Therefore the connection cannot be a replay of the client's traffic. Additionally, the server knows when its unpredictable random value was created, relative to its local clock, and therefore knows that handshake hasn't been delayed for an arbitrary amount of time.

In order for the "server_random" to be predictable by the client, it will have to be chosen by the client and suggested to the server. In order for the server to be assured of uniqueness, it will have to remember every "server_random" value that has been used so that it may reject duplicates. (Several methods of limiting the amount of state required for this are introduced below.)

Even without Snap Start, an attacker can delay an application data record in an established connection. However, both parties to the connection are likely to timeout after some period of inactivity,

bounding the amount of possible delay introduced. With Snap Start, since we are assuming that the client's initial flow includes a full handshake and application data, an attacker could arbitrarily delay the flow and have the server process the application data at a time of their choosing. As the server lacks an nonce it has no way of detecting this.

Without a similar mechanism to bound the delay of a Snap Start handshake, an attacker could perform a pseudo-replay attack: a handshake is delayed until the client retries, but the first handshake can still be used to deliver the same application data a second time. Because of this (and for other reasons other reasons given below) we require some degree of clock synchronisation between the client and server with respect the the timestamp in the "ClientHello". The level of synchronisation required is left to the application layer to determine although it's recommended that the permitted clock skew be shorter than the application's retry timeout.

3. Details

The essential features of Snap Start are now established: the client predicts the server's handshake flow using a client suggested "server_random", SessionTickets and knowledge from a previous connection to the same server.

We now note that this functions for both full and abbreviated handshakes. The abbreviated handshake retains its lower computational requirements but both now complete in the same number of round trips.

In order to limit the amount of state required for the server to reject repeated "server_random"s, we allow the server to bound this state temporally and spatially. In the temporal dimension, we define

that the "gmt_unix_time" of the server random is taken from "gmt_unix_time" of the client's random value. The server may use this timestamp to reject all suggested random values outside some window around the current time.

Spatially, we define that the subsequent eight bytes of the server's random value are the server's 'orbit' value. This value must be discovered by the client from a previous (non Snap Start) handshake. In the event that several, geographically separated servers share the same certificates, they may use different orbit values. This allows one to reject "server_random" values for the other without any communication between them. (The term 'orbit' was chosen only to be short and otherwise reasonably meaningless in this context.)

4. Requirements Notation

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

5. Snap Start Extension

A new extension type ("snap_start(TBD)") is defined and MAY be included by the client in its "ClientHello" message. If, and only if, the server sees this extension in the "ClientHello", it MAY choose to include the extension in its "ServerHello".

```
enum {
  snap_start(TBD), (65535)
} ExtensionType;
```

The "extension_data" field of a "snap_start" extension in a "ClientHello" MAY be empty.

If the server chooses to echo a "snap_start" extension then it is indicating that it MAY support Snap Start on future connections. The contents of the "extension_data" in this case MUST be:

```
struct {
  opaque orbit[8];
  CipherSuite snap_start_cipher_suite;
} ServerSnapStart;
```

"orbit" is the server's current orbit value and "snap_start_cipher_suite" contains the CipherSuite value that the client should assume that the server will use in a Snap Start handshake.

If the client wishes to attempt a Snap Start connection then it includes a non-empty "snap_start" extension in its "ClientHello". If the extension is not empty, then its contents MUST be:

```
struct {
  opaque orbit[8];
  opaque random_bytes[20];
  opaque predicted_server_handshake[8];
  // TLSCiphertext structures follow
} ClientSnapStart;
```

Following this, without a length prefix, the client may include one or more "TLSCiphertext" structures to be processed by the server in the case that the Snap Start is successful. These records are as described in RFC 5246 [RFC5246] section 6.2

The "orbit" MUST contain an orbit value obtained from a previous connection to the same server. "random_bytes" MUST contain 20 random bytes from a cryptographic random source equal in strength to the one used for the "client_random".

"predicted_server_handshake" MUST contain an FNV1a64 [fnv1a] hash of the server's predicted response flow. This hash is taken over the bytes of the "Handshake" structures, as defined in RFC 5246 section 7.4. If the client is attempting to resume a connection then this is calculated over the server's "ServerHello" message. Otherwise, this is calculated over all handshake messages from the "ServerHello" to the "ServerHelloDone" (inclusive). The client's prediction MUST assume that the server chooses its server random as detailed below. The client's prediction MUST also assume that the server includes a Snap Start extension with the same "ServerSnapStart" contents as previously observed.

If the client's prediction is correct then the server MAY perform a Snap Start handshake. If the server wishes to perform a Snap Start handshake then it MUST form its "random" value from the "gmt_unix_time" of the client's "random", followed by the server's orbit value, followed by the contents of "random_bytes" from the client's Snap Start extension.

The server MUST NOT transmit any predicted handshake messages and MUST start processing records from the client's Snap Start extension. For the purposes of the Finished calculation, the client's "ClientHello" is hashed as if the "snap_start" extension were not included (the "length" field of the "Handshake" structure from RFC 5246 section 7.4, and the prefixed length of the "extensions" member of the "ClientHello" are updated accordingly). When processing records from the extension, they are hashed as usual. Once the set of records embedded in the "ClientHello" has been exhausted, the server resumes reading records from the network. Records must not be partially contained within the "ClientHello" and partially read from the network.

(Since the "ClientHello" is likely to have a "Finished" message embedded within it, it cannot be hashed into the Finished calculation as normal as its contents would then depend in its own hash. One could consider hashing with the embedded hash zeroed out, however a "Finished" message is encrypted under the negotiated cipher suite and a CBC-based ciphersuite would spread the effects of the embedded hash beyond its apparent boundaries. Likewise, the "Finished" message is compressed using the negotiated compression algorithm thus the length of the record, and thus the extension, depend on the contents of the hash. Given these limitations, removing the extension altogether is the simplest way to break the cycle.)

If the server chooses not to perform a Snap Start handshake (for any reason, including that the application rejected the suggested random

value or that the client mispredicted the handshake) then the handshake MUST continue as normal. The server MUST NOT change the

way that it hashes the "ClientHello". The server MAY echo a "snap_start" extension.

6. Active attack considerations

Given that this draft makes changes to the Finished hash, it's required to show that no active attack can cause a handshake to complete which differs from that which would have occurred otherwise.

Since the "snap_start" extension is not included in the Finished hash of a Snap Start handshake, we have to consider the results of an attacker manipulating its contents.

Firstly, since the embedded records are hashed as usual, the same security properties hold: any manipulation will be detected by the Finished hash, or by a MAC verification failure.

An attacker could manipulate the orbit. This would typically cause the server to reject the suggested random value and a normal handshake would detect the manipulation. In the case that the server still proceeds with a Snap Start handshake, the orbit is copied into the "server_random" in the "ServerHello", which is then hashed into the Finished calculation (although not transmitted).

An attacker could manipulate the suggested server random. The same argument as for the orbit holds here.

An attacker could manipulate the hash of the predicted messages. Assuming that the client correctly predicted the hash, the manipulation would cause the server to not perform a Snap Start handshake and the manipulation would be detected. Assuming that the client mispredicted, and that the manipulation results in a different, but also incorrect, value then the same argument applies. Assuming that the client mispredicted and the manipulation corrects the hash, the server could perform a Snap Start handshake, but the

differing contents of the predicted handshake messages will be hashed into the Finished calculation and the manipulation will be detected.

7. Requirements on the application

Applications are required to ensure that no suggested random value is accepted twice within the scope of any given certificate. In general, validation of the suggested random value is outside the scope a TLS implementation (although it may handle simple cases and provide utility code for others). Applications may use the orbit value and client timestamp to aid them in this. Applications may always safely reject a suggested random value. Applications SHOULD limit the allowed difference between the timestamp in the suggested random value and the current time in order to prevent arbitrary delays, as detailed in the design section of this document.

For a single server deployment, the server may generate a new, random orbit value each time that it starts and, thereafter, maintain an in-memory data structure. Each random value seen should be "struck off" by recording it in this data structure (the "strike register"). The strike register's size can be bound by fixed limits and by rejecting all random values where the timestamp is outside a certain window around the current time.

For a multi-server deployment in a single location, the servers should share an orbit value and a strike register. The strike register is likely to be a held in a single location which the TLS servers access over an internal network.

For a multi-cluster deployment, where the clusters are geographically separated, each cluster should have its own orbit value and shared strike register. The effectiveness of Snap Start in this setup is limited by the probability of a given client repeatedly being served by the same cluster. With pure round robin scheduling, a Snap Start handshake is unlikely to be successful.

8. Interactions with the Session Tickets extension

A successful Snap Start abbreviated handshake can occur without the use of session tickets. A successful Snap Start full handshake, without session tickets, can only occur if the server doesn't generate a random session id. A server MAY choose not to generate a session id if the client presents a Snap Start extension but not a session tickets extension. However, all TLS implementations of Snap Start SHOULD implement session tickets. TLS clients which send a Snap Start extension SHOULD also send a Session Tickets extension.

9. Interactions with OCSP stapling

Clients attempting a Snap Start handshake MUST trust the server's cached certificate. This includes validating revocation information (via OCSP [RFC2560], CRLs [RFC5280] etc) as the local policy dictates.

TLS clients which send a Snap Start extension SHOULD NOT send a "status_request" extension as defined in RFC 4366 [RFC4366] section 3.6. A client may be able to predict the contents of a "CertificateStatus" message but, if it can predict it, then it

doesn't need it and, if it needs fresh OCSP information, then it shouldn't have attempted a Snap Start handshake using a certificate that it cannot validate.

This does preclude the case where the client has cached a valid OCSP response that is still timely, but the server has a response valid further into the future. We can only suggest that opportunistic OCSP stapling additionally be included in application level protocols for this situation.

10. Interactions with client certificates

A Snap Start handshake can include client-side authentication. In this case the client must predict that the server will send a

"CertificateRequest" message, calculate its "predicted_server_handshake" accordingly and embed "Certificate" and "CertificateVerify" messages in the Snap Start extension.

The "handshake_messages" over which the "CertificateVerify" is calculated MUST omit the Snap Start extension as detailed for the Finished calculation, above.

11. Examples

Firstly, a client contacting a previously unknown server for the first time may include an empty Snap Start extension in its ClientHello. The server, if so capable, could reply with:

```
01 02 03 04 05 06 07 08         (Orbit value)
00 2f                           (Cipher suite)
```

A future connection may now attempt a Snap Start by including a Snap Start extension in the ClientHello with the following contents:

```
01 02 03 04 05 06 07 08         (Orbit value)
88 c0 1e 1c a9 4e ...           (Random bytes)
aa bb cc dd ee ff 00 11         (predicted server handshake)
16 03 03 00 84 10 00 00 80 ...  (ClientKeyExchange)
14 03 03 00 01 01               (ChangeCipherSpec)
16 03 03 00 20 ...              (Finished)
17 03 03 00 50 ...              (Application data)
```

If the Snap Start is successful, then the message flow looks like this:

```
ClientHello     -------->
                                ChangeCipherSpec
                                        Finished
                <--------       Application data
```

If the client mispredicts the server's handshake, however, then the flow is unaltered from Figure 1 in RFC 5246 section 7.3.

12. Prior work

The idea of cache-side caching of long lived server parameters has been discussed in Client Side Caching for TLS [fasttrack] and specified in draft-ietf-tls-cached-info [cached-info]. Client Side Caching for TLS [fasttrack] also considered including an opportunistic "ClientKeyExchange" message in the client's initial flow.

13. IANA Considerations

This document requires IANA to update its registry of TLS extensions to assign an entry, referred herein as "snap_start".

14. Acknowledgements

This document benefited specifically from discussions with Wan-Teh Chang, Bodo Moeller and Nagendra Modadugu.

15. Normative References

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

[RFC2560] Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 2560, June 1999.

[RFC4366] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, April 2006.

[RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, January 2008.

[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.

[fnv1a] Noll, L., "FNV hash".

[cached-info] Santesson, S., "Transport Layer Security (TLS) Cached Information Extension", Internet Draft (work in progress), April 2010.

[fasttrack] Shacham, H., Boneh, D., and E. Rescorla, "Client Side Caching for TLS", Nov 2004.

Appendix A. Changes

To be removed by RFC Editor before publication

Author's Address

Adam Langley
Google Inc

Email: agl@google.com