Network Working Group
Internet-Draft
Intended status: Experimental
Expires: September 10, 2015

C. Huitema
Microsoft

March 9, 2015

# TLS over HTTP
## draft-huitema-tls-tlsoverhttp-00.txt

Abstract

We observe that attacks against HTTPS are getting more and more popular. The attacks typically exploit weaknesses in PKI certificate verification software. These weaknesses allow a third party to insert itself as a Man-In-The-Middle in a TLS connection, accessing the content of messages that were previously encrypted and in some case changing these messages.

TLS over HTTP allows clients and servers to carry a TLS conversation on top of HTTP, and thus bypass the man-in-the-middle attackers. Different deployment models are possible, e.g., HTTP over TLS over HTTP, application-layer-protocol over TLS over HTTP, or HTTP over TLS over HTTP over TLS.

The proposed solution allows for reuse of the existing TLS implementation, thus minimizing the development costs and risks. It includes an optional obfuscation layer, to maximize the likelihood of working unnoticed by firewalls and other MITM boxes.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 10, 2015.

Table of Contents



## 1. Introduction

Following the Snowden revelation, there has been an increased push to deploy encryption over the Internet. The IAB reinforced this trend by publishing the IAB Statement on Internet Confidentiality [IABConfidentiality]. Of course, every action brings a reaction, and the push for encryption is countered by the deployment of encryption-breaking middleboxes. The middle boxes typically use weaknesses in the Certificate Verification system of popular TLS implementations to insert themselves as "Man in the middle" (MITM). In another class of attacks, the middleboxes will simply block encrypted communication, only allowing clear-text HTTP.

The MITM attacks against TLS can be detected in various ways. For example, the TLS client might have obtained in advanced a good copy of the certificate used by the TLS server. The client can then compare the value used in the TLS exchange to the value in the good

copy. If they differ, the MITM attack is detected. However, in the current state of the art, the only choice of the client is either to abort the connection, or to tolerate the encryption compromise by the middlebox.

Both the MITM attack and the clear-text only attacks can be mitigated by running TLS on top of HTTP (HTLS).

## 2. TLS over HTTP architecture

The basic TLS over HTTP architecture has four layers, including a framing layer which is conceptually placed between TLS and HTTP.

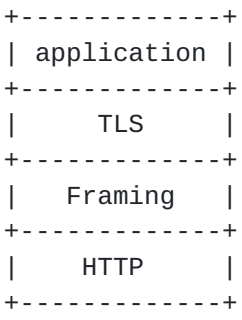


Figure 1

The framing layer is responsible for encapsulating the TLS frames into the HTTP protocol.

Figure 1 describes the simplest stack, in which we assume that HTTP is run "end to end." There are many possible variants, including in particular the HTTPS variant, in which HTTP is run over TLS. In that case, we will effectively see two instances of TLS running on top of each other.

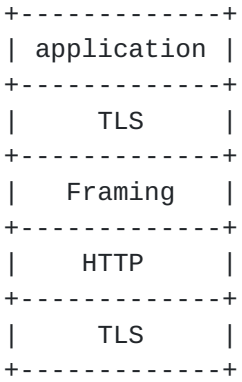


Figure 2

Having two instances of TLS obviously causes some amount of overhead. It is however the price to pay in order to actually traverse series of middleboxes while establishing end-to-end encryption.

This document defines framing of HTLS over web sockets [RFC6455]. There may be a need in the future to provide more options. For example, if middleboxes started to deploy inspection software to detect and prevent usage of HTLS over web sockets, there will be a need to define some form of obfuscation, rendering the content as indistinguishable as possible from regular HTTP content. The new semantics introduced in HTTP/2.0 [I-D.ietf-httpbis-http2] will probably allow for efficient implementation of such encapsulations.

## 3. Framing with web sockets

The WebSocket protocol [RFC6455] enables applications to transmit arbitrary binary messages over a channel that is initiated as HTTP or HTTPS. The WebSocket protocol is compatible with HTTP/1.1 [RFC2616] and with HTTP over TLS [RFC2818].

When using the web socket framing, the client will establish a web socket using the URI reserved by the server for HTLS over web sockets. Once the web socket has reached the OPEN state, both client and server will be able to send and receive data frames, as defined in section 6 of [RFC6455]. Each TLS frame will be sent as a separate web socket data frame.

The client will start the TLS handshake as defined for the client-chosen version of TLS in [RFC2246], [RFC5246], or [I-D.ietf-tls-tls13]. The client and server will then execute the TLS protocol, and carry the application protocol inside TLS data frames.

We should note that, while the framing layer is conceptually layered above HTTP, the WebSocket protocol really replaces HTTP after the initial negotiations. The architecture diagram with WebSocket framing really looks like:

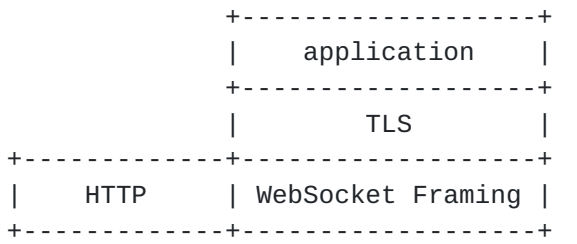


Figure 3

## 4. HTLS parameters

An application that starts an HTLS connection will need to specify a the HTTP or HTTPS URI used to establish the underlying HTTP connection and the type of framing required for the encapsulation. We assume that these parameters will be passed as an end to end exchange between server and clients, prior to the establishment of the HTLS connection.

## 5. MITM bypass scenario

As stated in Section 1, one of the main motivation for HTLS is to offer a better choice to the user when a secure connection is compromised by an MITM attack.

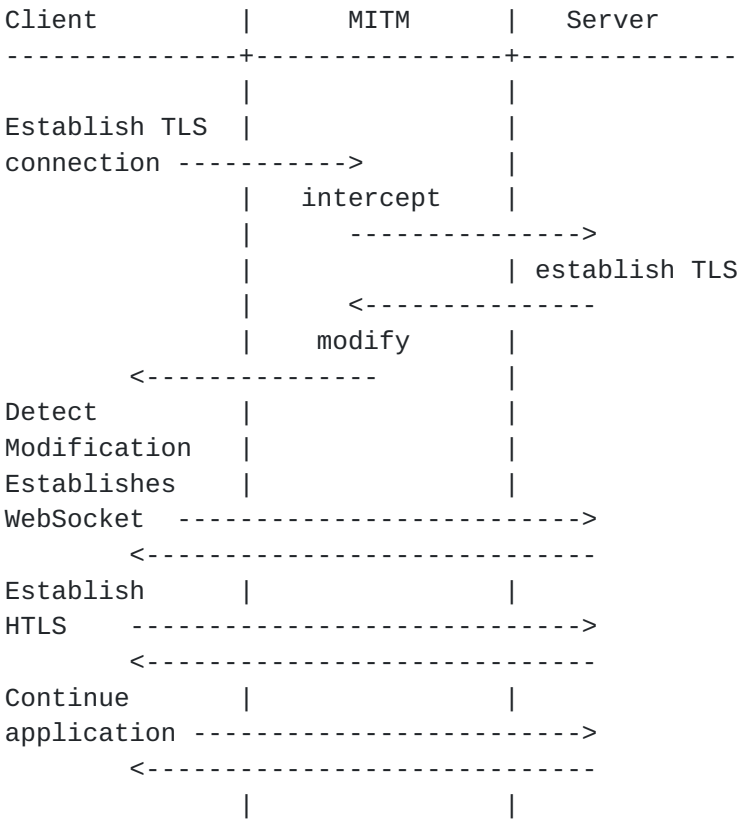


Figure 4

The exchanges depicted in Figure 4 are easy to understand, but a few of the actions require further explanation:

o How does the client detect the modification?

- How does the client know that the server can use HTLS, and what parameters should it use?

- What about performance degradation?

There is lots of work going on in the IETF and in the industry to detect attacks against TLS. The most popular attack involves the forging of certificates by the MITM, as done for example by the "Superfish" software [Superfish]. The MITM will generate a fake certificate for the target, have it signed by a certificate authority under its control, and somehow convince the security software on the client side to trust that certificate authority. The client can detect the attack if it knows the "right" certificate for the server and observes a mismatch. The client could also apply some logic to find suspicious characteristics in the certificate provided by the MITM, and detect the fakery.

Once the client decides that the certificate is probably fake, it has to establish a WebSocket session to the server, and for that it needs parameters. In fact, since the client already knows the name of the server and has already decided to use the WebSocket protocol over TLS, it needs just one parameter, the URI of the HTLS web socket service on the server. The simplest solution is to use a constant value, "HTLS." Successful establishment of a web socket through that URI indicates that the server supports HTLS.

Clearly, using HTLS degrades performance. There are a number of additional handshakes before any application data can be exchanged, some additional overhead due to encapsulation errors, and a computing charge for running the encryption twice. Running HTLS is thus a tradeoff. In the absence of HTLS, the client's only choices are to drop the connection, or accept working in an insecure manner. HTLS adds a third choice, a way to bypass the MITM attack at the cost of reduced performance. There will be cases where secure and slow is better than either no connection at all or an insecure connection.

## 6. Winning the cat and mouse game

Deceiving the MITM attacker is of course a game of "cat and mouse." The mice (the victims), in that case, are taking evasive action to escape the cat (the MITM attacker). HTLS is one of these evasive actions. It will work as long as the attackers are not aware of it, but we have to assume that smarter mice beget smart cats. That is the reason for having an optional "framing" layer as part of the architecture.

The first instantiation of the framing uses web sockets and a standard URI at the server, but it is reasonably easy for an attacker

to detect that. The answer is of course to include some level of obfuscation in the framing layer, so that HTLS traffic is difficult to distinguish from other web traffic. The simplest obfuscation is to keep the web socket encapsulation, but make the URI parameter hard to discover by third parties. Instead of using the constant string "HTLS," servers can come up with their own creative names. If the server and the client cooperate, they can vary these names over time according to some pre-established schedule. At that point, it becomes hard for the MITM to distinguish HTLS from other types of web sockets. The MITM attacker now is left with two options: block all web socket traffic, or allow HTLS to pass unmolested.

If it turns out that MITM attackers are willing to ban all web socket traffic, we will have to study a different form of framing. This should probably be based on HTTP/2.0 support for multiple streams and bidirectional transmissions. We could imagine sending each TLS frame as a separate page, with the set of pages multiplexed over the HTTP/2.0 connection. The frame will be encoded as page content, which could be binary, but could also be some sophisticated text or image representation designed to elude MITM censorship. This is of course for further study.

In the end game, the MITM attacker should be left with the choice of allowing HTLS or blocking all web traffic. Some attackers may indeed choose to block everything, but most will probably just give up.

## 7. Security Considerations

The draft addresses a major security issue, the interception of secure connections by middleboxes that break the end to end nature of encryption.

## 8. IANA Considerations

This draft does not require any IANA action.

## 9. Acknowledgments

Thanks to Dave Thaler for suggesting the use of Web Sockets for encapsulating TLS in HTTPS.

## 10. References

### 10.1. Normative References

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.

## 10.2. Informative References

[I-D.ietf-httpbis-http2]
Belshe, M., Peon, R., and M. Thomson, "Hypertext Transfer Protocol version 2", draft-ietf-httpbis-http2-17 (work in progress), February 2015.

[I-D.ietf-tls-tls13]
Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-04 (work in progress), January 2015.

[IABConfidentiality]
"IAB Statement on Internet Confidentiality", November 2014, <http://www.iab.org/2014/11/14/iab-statement-on-internet-confidentiality/>.

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

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

[Superfish]
Osborne, C., "Lenovo's Superfish spectacle: 'Catastrophic' security failures discovered", February 2015, <http://www.zdnet.com/article/lenovos-superfish-its-worse-than-we-thought/>.

## Author's Address

Christian Huitema
Microsoft
Redmond, WA 98052
U.S.A.

Email: huitema@microsoft.com