Detection and Mitigation of HTTPS Man-in-the-Middles and Impersonators

John Mattsson, Mats Näslund, Ericsson Research

Abstract. Trust in TLS and the Web's PKI is fundamental for the further development of the Internet economy, including e-commerce, Internet banking, and e-governance. Unfortunately, there are several ways to perform impersonation and Man-in-the-Middle attacks on HTTPS. There are some ongoing activities to detect and mitigate such attacks, but this is not enough. In light of the mass-surveillance revelations, W3C and the Internet community should take firm action to detect and mitigate all types of Man-in-the-Middle and impersonation attacks on HTTPS, this includes specifying different APIs for channel binding.

1. Introduction

Web application and browser security relies heavily on TLS and the Web's Public-Key Infrastructure (PKI), both for communication security and origin server authentication. Trust in the TLS protocol and the Web's PKI is fundamental for the further development of the Internet economy, including e-commerce, Internet banking, and e-governance.

The standard way to provide security for a HTTPS connection is to set up a TLS connection to the origin server, and to authenticate the server, but not the client, with a certificate. The server certificate is validated if it's signed by a Certification Authority (CA) trusted by the client. In practice the client trusts a lot of CAs, in most cases all CAs belonging to a certificate chain where the root certificate is distributed with the operating system and not blocked by the browser. If client authentication/authorization is done, this is usually done at higher protocol layers, completely decoupled from the TLS connection, and without any mutual authentication. This means that:

- The server can only verify that's it's talking to someone with access to the password/security token, but not if that someone is also terminating the TLS connection or if there is a Man-in-the-Middle.
- The client does not get any two-factor server authentication. If the TLS connection is terminated by an Impersonator, the additional authentication does not help to reveal that.

While TLS should in theory protect against Man-in-the-Middles (MITM) and Impersonators, there are several possible attacks were an attacker is falsely authenticated as the legitimate server, most of which have been seen in reality:

- Stealing the server's private key and using that with the server's valid certificate.
- Stealing a CA's private key and using that to issue a new valid server certificate.
- Getting a CA in the Web's PKI to issue a new valid server certificate. This could be achieved by fooling the CA, hacking the CA, or cooperating with the CA.
- Installing a new trusted root certificate in the client, and then creating a new server certificate without help of any CA in the Web's PKI. New trusted root certificates can be installed by owning or controlling the client, hacking the client, or fooling the user.
- Using a self-signed certificate and hoping the user ignores the security warning.
- Taking advantage of some implementation weakness in the client software, like the Apple "goto fail" bug.

There are some ongoing activities [4] [5] [6] on using DNS and HTTP headers to detect and mitigate such attacks, but none of these provide real-time information to the server. They

also require either DNSSEC or that the user has recently visited the web site and that no MITM attacker was present during the first visit. The way HTTPS is used; the client may insert a MITM without the server knowing or being able to reject. Even if the client has accepted the presence of an MITM, the server may not want to disclose sensitive information (e.g. financial, health care) or accept requests (e.g. money transfers) in such cases.

A more detailed overview of HTTPS MITMs and Impersonators, as well as mechanisms for detection and mitigation can be found in Appendixes A and B.

2. Way Forward

From the pervasive monitoring debate we know that there are many organizations actively performing large scale pervasive monitoring, including HTTPS MITMs. IETF has stated that pervasive monitoring is an attack on the Internet that should be mitigated [10].

The current Web Cryptography API specification [17] states as a use case:

> "*Further, the authentication data could be further enhanced by binding the authentication to the TLS session that the client is authenticating over, by deriving a key based on properties of the underlying transport.*"

Unfortunately this has not yet made it into the current specification. In light of the mass-surveillance revelations and the ongoing pervasive monitoring debate, these thoughts should not only be brought up again, they should be expanded and prioritized. Trust in TLS and the PKI are fundamental parts of the Web. W3C and the Internet community should take firm action to enable detection and mitigation of all types of MITM and impersonation attacks on HTTPS, this includes specifying different APIs for channel binding.

Today, servers are in most cases not able to detect the presence of HTTPS MITMs. This is not acceptable, even if the client has accepted the presence of an MITM, the server may not want to disclose sensitive information (e.g. financial, health care) or accept requests (e.g. money transfers) in such cases. Even HTTPS Man-in-the-Middles used for legitimate use cases should be detectable and visible for both the client and the server.

- W3C should together with the rest of the community strive to increase and normalize the use of DNS, HTTP, and third-party detection and mitigation mechanisms such as [3] [4] [5] [6] [18] as well as the use of DNSSEC. If possible, they should be built into existing server management and configuration tools so that the usage is automatic. Another important activity is to create best current practices, recommendations, and guidelines.

- W3C and the Internet community should research and standardize developer friendly mechanisms, interfaces, and APIs for secure channel binding in web applications with different usability and security requirements. This includes web applications performing the authentication processing in JavaScript, software and hardware plugins, as well as separate hardware tokens.

References

[1] Dierks, Rescorla, IETF RFC 5246, "The Transport Layer Security (TLS) Protocol Version 1.2", 2008
http://tools.ietf.org/html/rfc5246

[2] Sheffer et al., "Recommendations for Secure Use of TLS and DTLS", (IETF work in progress)
http://tools.ietf.org/html/draft-ietf-uta-tls-bcp

[3] Hallam-Baker, Stadling, IETF RFC 6844, "DNS Certification Authority Authorization (CAA) Resource Record", 2013
http://tools.ietf.org/html/rfc6844

[4] Hoffman, Schlyter, IETF RFC 6698, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", 2012
http://tools.ietf.org/html/rfc6698

[5] Hodges et al., IETF, RFC 6797, "HTTP Strict Transport Security (HSTS)", 2012
http://tools.ietf.org/html/rfc6797

[6] Evans et al., "Public Key Pinning Extension for HTTP" (IETF work in progress)
http://tools.ietf.org/html/draft-ietf-websec-key-pinning

[7] Bhargavan et al., "Strengthening Master Secrets! (to get good TLS Channel Identifiers)", 2014
http://tools.ietf.org/agenda/89/slides/slides-89-tls-3.pdf

[8] 3GPP TS 33.220 "Generic Authentication Architecture (GAA); Generic Bootstrapping Architecture (GBA)"
http://www.3gpp.org/DynaReport/33220.htm

[9] 3GPP TR 33.823 "Security for usage of General Bootstrapping Architecture (GBA) with a User Equipment (UE) browser"
http://www.3gpp.org/DynaReport/33823.htm

[10] Farrell, Tschofenig, IETF, RFC 7258, "Pervasive Monitoring Is an Attack", 2014
http://tools.ietf.org/html/rfc6797

[11] Gondrom, "Securing the SSL/TLS channel against man-in-the-middle attacks", 2012
https://www.owasp.org/images/4/4b/OWASP_defending-MITMA_APAC2012.pdf

[12] STREWS, "Web-platform security guide: Security assessment of the Web ecosystem", 2013
http://www.strews.eu/images/STREWS-D1.1-final.pdf

[13] Karapanos, Capkun, "On the Effective Prevention of TLS Man-In-The-Middle Attacks in Web Applications", 2014
https://eprint.iacr.org/2014/150.pdf

[14] Oppliger et al., "A Proof of Concept Implementation of SSL/TLS Session-Aware User Authentication (TLS-SA)", 2007
http://www.inf.ethz.ch/personal/basin/pubs/kivs06.pdf

[15] Holz et al. "X.509 Forensics: Detecting and Localising the SSL/TLS Men-in-the-Middle)", 2012
http://www.net.in.tum.de/fileadmin/bibtex/publications/papers/holz_x509forensics_esorics2012.pdf

[16] FIDO Alliance, "Universal 2nd Factor (U2F)"
https://fidoalliance.org/specifications/download

[17] W3C, "Web Cryptography API"
http://www.w3.org/TR/WebCryptoAPI/

[18] Laurie et al., IETF RFC 6962, "Certificate Transparency", 2014
http://tools.ietf.org/html/rfc6962

A. HTTPS Man-in-the-Middles and Impersonators

Web application and browser security relies heavily on TLS and the Web's Public-Key Infrastructure (PKI), both for communication security and origin server authentication. Trust in the TLS protocol and the Web's PKI is fundamental for the further development of the Internet economy, including e-commerce, Internet banking, and e-governance. While the TLS protocol comes in many versions, with various options, extensions, and ciphersuites, the protocol weaknesses have largely been fixed, and the latest version [1] with the current best practices [2] can for the moment be considered secure (unless there are implementation weaknesses).

The standard way to provide security for a HTTPS connection is to set up a TLS connection to the origin server, and to authenticate the server, but not the client, with a certificate. The server certificate is validated if it's signed by a Certification Authority (CA) trusted by the client. In practice the client trusts a lot of CAs, in most cases all CAs belonging to a certificate chain where the root certificate is distributed with the operating system and not blocked by the browser.

If client authentication/authorization is done, this is usually done at higher protocol layers, completely decoupled from the TLS connection, and without any mutual authentication. This means that:

- The server can only verify that's it's talking to someone with access to the password/security token, but not if that someone is also terminating the TLS connection or if there is a Man-in-the-Middle.
- The client does not get any two-factor server authentication. If the TLS connection is terminated by an Impersonator, the additional authentication does not help to reveal that.

We differentiate between a HTTPS Man-in-the-Middle (MITM) and HTTPS Impersonator. While all HTTPS Man-in-the-Middles are Impersonators, all Impersonators are not Man-in-the-Middles. We use the term Pure Impersonator to describe an Impersonator that is not a MITM. The possible detection and mitigation techniques for a MITM are a superset of those for a Pure Impersonator. While TLS should in theory protect against Man-in-the-Middles and Impersonators, there are several possible attacks were an attacker is falsely authenticated as the legitimate server, most of which have been seen in reality:

- Stealing the server's private key and using that with the server's valid certificate.
- Stealing a CA's private key and using that to issue a new valid server certificate.
- Getting a CA in the Web's PKI to issue a new valid server certificate. This could be achieved by fooling the CA, hacking the CA, or cooperating with the CA.
- Installing a new trusted root certificate in the client, and then creating a new server certificate without help of any CA in the Web's PKI. New trusted root certificates can be installed by owning or controlling the client, hacking the client, or fooling the user.
- Using a self-signed certificate and hoping the user ignores the security warning.
- Taking advantage of some implementation weakness in the client software, like the Apple "goto fail" bug.

There are other attacks on HTTPS were the attacker is not authenticated as the server, such as downgrade, renegotiation, and stripping attacks. See [11], [12], and [13] for more details.

Figure 1: HTTPS Man-in-the-Middle

Figure 1 shows a HTTPS MITM. If the client has not requested or accepted the man-in-the-middle, this is clearly an attack on the client as the MITM impersonates the server. And if

the client is authenticated, or the MITM modifies the HTTP messages, it is clearly an attack on the server. However, there are legitimate use cases for HTTPS MITMs. If the MITM is requested by the client, the client is not authenticated to the server, and the MITM does not modify the HTTP messages (except blocking malicious or sensitive data), it cannot really be seen as an attack on either part.

As the server is involved in the communication, the MITM may be detected and mitigated with or without the help of a third party.

Figure 2: HTTPS Pure Impersonator

Figure 2 shows a HTTPS Pure Impersonator. This is always an attack on the server and the client. As the server is not involved in the communication, real-time detection and mitigation can only be done with the help of a third party.

B. Mechanisms for Detection and Mitigation

As discussed above, an attack on one of the parties may or may not be an attack on the other party. They way HTTPS is used; the client may insert a MITM without the server knowing or being able to reject. Even if the client has accepted the presence of an MITM, the server may not want to disclose sensitive information (e.g. financial, health care) or accept requests (e.g. money transfers) in such cases. After detecting a MITM, the server may or may not choose to terminate the connection. Other actions could be to limit the information sent to the client or display the existence of an MITM to the user.

Detection and mitigation using prior knowledge

If the client has information (in addition to a set of trusted root certificates) prior to connecting to the server, the client may be able to detect MITM, impersonation, downgrade or stripping attacks. Several approaches to accomplish this exist:

- DNS Certification Authority Authorization (DNS record type CAA) [3] mitigates attacks on CAs by restricting which CAs that can issue certificates for a host.
- DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol (DNS record type TLSA) [4] mitigates attacks on servers by sending the server certificate to clients in the DNS response.
- HTTP Strict Transport Security (HSTS) [5] mitigates attacks on servers by enabling web sites to declare themselves accessible only via HTTPS.
- Public Key Pinning Extension for HTTP [6] mitigates attacks on servers by enabling web sites to declare fingerprints of allowed server and CA certificates.

The problems with the above mechanisms is that the DNS-based mechanisms requires deployment and use of DNSSEC, while the HTTP-based mechanism requires that the user recently visited the web site, and that no MITM attacker was present during the first visit.

Instead of relying on DNS, out-of-band signaling with another third party over a secure channel might be used. See [12], [15], and [18] for more details and examples. The third party could be the browser vendor, the operating system vendor, or a trusted third party chosen by the client.

While hard-coding certificate information for all servers on the web is not possible, hard-coding certificate information for a single third-party is plausible and something e.g. Google Chrome already does for Google services.

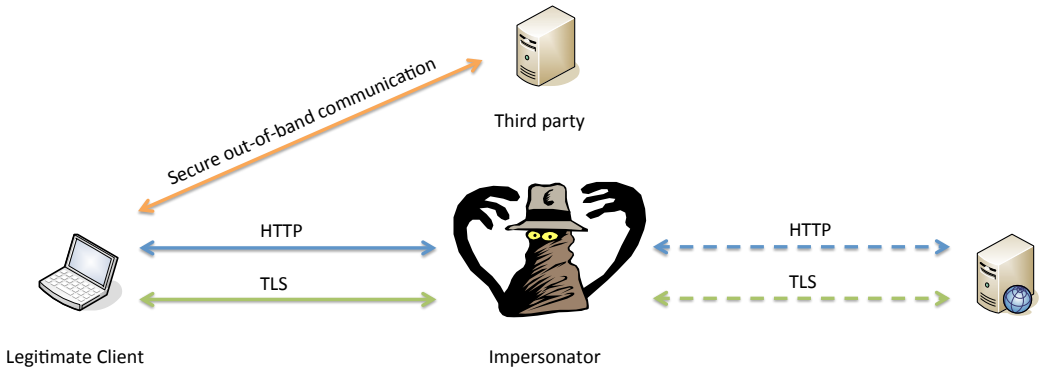

Figure 3: Prior knowledge using a third party

The third party communicates with the client over a secure channel, forwarding information about acceptable server and CA certificates (either the whole certificates or fingerprints thereof), or just information that the server supports HTTPS, i.e. the type of information discussed above for usage with DNS and HTTP. The third party approach has the benefits of working even if DNSSEC is not deployed and even if the client has not recently visited the web site.

Detection and mitigation using in-band authentication channel binding

In traditional channel binding, the cryptographic keys used to protect the connection are derived (partially) in dependence of some secret information related to the user authentication. This enables the parties to determine that the TLS end-points are the same as the end-points in the authentication protocol. But this form of channel binding is not possible to combine with HTML form-based authentication, and is problematic to use in web applications that require per-transaction authentication.

Another alternative is to extract some unique channel identifier from the TLS connection and use that identifier as input when calculating the authentication response. The channel identifier must be constructed so that it's computationally infeasible for a MITM to make the same channel identifier appear in the two TLS connections. An overview and discussion of TLS channel identifiers can be found in [7].

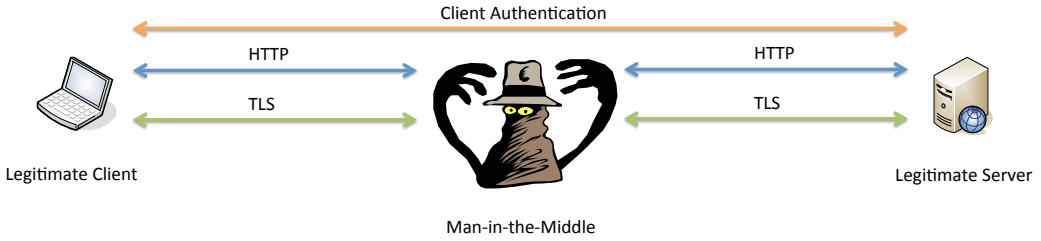

Figure 4: In-band detection using client authentication

Examples of extracting a unique channel identifier from TLS for the purpose of MITM detection can be found in [8], [9], and [16]. The simple way to extract and use such a value for a web application would be:

1. A JavaScript API for fetching the unique identifier from the TLS connection to use in calculation of the authentication response sent to the web application.

But, as browsers rely on so-called host-based security, a MITM has total control of the JavaScript runtime environment. A solution relying on information from, or giving secret information to, the JavaScript runtime environment can therefore only protect against passive attackers (passive on the layers above TLS). As a MITM also have complete information of the two TLS connections, the only way to protect against active MITMs is to make sure that the MITM do not get access to any secret authentication information and that the used channel identifier cannot be modified by the MITM. Some ways of accomplishing this are:

2. The channel identifier is displayed in the browser chrome and used by the user to calculate the authentication response (with the help of an hardware token). The identifier could e.g. be displayed next to the padlock.

Figure 5: Channel identifier displayed in browser chrome

3. The channel identifier is transferred by the browser to the module performing the authentication processing in hardware or software (e.g. one of the national eID systems).
4. Using the channel identifier as input, the browser calculates the authentication response after requesting input from the user, e.g. as part of an updated HTTP digest mechanism, or as part of the TLS handshake.

These four approaches fit web applications with different security requirements. Alternative 1 only protects against passive attackers, 2 and 3 depend on external software or hardware modules, and 4 relies on using HTTP digest or TLS handshake for client authentication. See [13] and [14] for more details on some channel binding approaches.