Transport Layer Protection Cheat Sheet

Introduction

This cheat sheet provides guidance on how to implement transport layer protection for an application using Transport Layer Security (TLS). When correctly implemented, TLS can provides a number of security benefits:

- Confidentiality - protection against an attacker from reading the contents of traffic.
- Integrity - protection against an attacker modifying traffic.
- Replay prevention - protection against an attacker replaying requests against the server.
- Authentication - allowing the client to verify that they are connected to the real server (note that the identity of the *client* is not verified unless client certificates are used).

TLS is used by many other protocols to provide encryption and integrity, and can be used in a number of different ways. This cheatsheet is primarily focused on how to use TLS to protect clients connecting to a web application over HTTPS; although much of the guidance is also applicable to other uses of TLS.

SSL vs TLS

Secure Socket Layer (SSL) was the original protocol that was used to provide encryption for HTTP traffic, in the form of HTTPS. There were two publicly released versions of SSL - versions 2 and 3. Both of these have serious cryptographic weaknesses and should no longer be used.

For various reasons the next version of the protocol (effectively SSL 3.1) was named Transport Layer Security (TLS) version 1.0. Subsequently TLS versions 1.1, 1.2 and 1.3 have been released.

The terms "SSL", "SSL/TLS" and "TLS" are frequently used interchangeably, and in many cases "SSL" is used when referring to the more modern TLS protocol. This cheatsheet will use the term "TLS" except where referring to the legacy protocols.

Server Configuration

Only Support Strong Protocols

The SSL protocols have a large number of weaknesses, and should not be used in any circumstances. General purpose web applications should default to TLS 1.3 (support TLS 1.2 if necessary) with all other protocols disabled. Where it is known that a web server must support legacy clients with unsupported an insecure browsers (such as Internet Explorer 10), it may be necessary to enable TLS 1.0 to provide support.

Where legacy protocols are required, the "TLS_FALLBACK_SCSV" extension should be enabled in order to prevent downgrade attacks against clients.

Note that PCI DSS forbids the use of legacy protocols such as TLS 1.0.

Only Support Strong Ciphers

There are a large number of different ciphers (or cipher suites) that are supported by TLS, that provide varying levels of security. Where possible, only GCM ciphers should be enabled. However, if it is necessary to support legacy clients, then other ciphers may be required.

At a minimum, the following types of ciphers should always be disabled:

- Null ciphers
- Anonymous ciphers
- EXPORT ciphers

See the TLS Cipher String Cheat Sheet for full details on securely configuring ciphers.

Use Strong Diffie-Hellman Parameters

Where ciphers that use the ephemeral Diffie-Hellman key exchange are in use (signified by the "DHE" or "EDH" strings in the cipher name) sufficiently secure Diffie-Hellman parameters (at least 2048 bits) should be used

The following command can be used to generate 2048 bit parameters:

```
openssl dhparam 2048 -out dhparam2048.pem
```

The Weak DH website provides guidance on how various web servers can be configured to use these generated parameters.

Disable Compression

TLS compression should be disabled in order to protect against a vulnerability (nicknamed CRIME) which could potentially allow sensitive information such as session cookies to be recovered by an attacker.

Patch Cryptographic Libraries

As well as the vulnerabilities in the SSL and TLS protocols, there have also been a large number of historic vulnerability in SSL and TLS libraries, with Heartbleed being the most well known. As such, it is important to ensure that these libraries are kept up to date with the latest security patches.

Test the Server Configuration

Once the server has been hardened, the configuration should be tested. The OWASP Testing Guide chapter on SSL/TLS Testing contains further information on testing.

There are a number of online tools that can be used to quickly validate the configuration of a server, including:

- SSL Labs Server Test
- CryptCheck
- CypherCraft
- Hardenize
- ImmuniWeb
- Observatory by Mozilla
- Scanigma
- OWASP PurpleTeam `cloud`

Additionally, there are a number of offline tools that can be used:

- O-Saft - OWASP SSL advanced forensic tool
- CipherScan
- CryptoLyzer
- SSLScan - Fast SSL Scanner
- SSLyze
- testssl.sh - Testing any TLS/SSL encryption
- tls-scan

- OWASP PurpleTeam `local`

Certificates

Use Strong Keys and Protect Them

The private key used to generate the cipher key must be sufficiently strong for the anticipated lifetime of the private key and corresponding certificate. The current best practice is to select a key size of at least 2048 bits. Additional information on key lifetimes and comparable key strengths can be found here and in NIST SP 800-57.

The private key should also be protected from unauthorized access using filesystem permissions and other technical and administrative controls.

Use Strong Cryptographic Hashing Algorithms

Certificates should use SHA-256 for the hashing algorithm, rather than the older MD5 and SHA-1 algorithms. These have a number of cryptographic weaknesses, and are not trusted by modern browsers.

Use Correct Domain Names

The domain name (or subject) of the certificate must match the fully qualified name of the server that presents the certificate. Historically this was stored in the `commonName` (CN) attribute of the certificate. However, modern versions of Chrome ignore the CN attribute, and require that the FQDN is in the `subjectAlternativeName` (SAN) attribute. For compatibility reasons, certificates should have the primary FQDN in the CN, and the full list of FQDNs in the SAN.

Additionally, when creating the certificate, the following should be taken into account:

- Consider whether the "www" subdomain should also be included.
- Do not include non-qualified hostnames.
- Do not include IP addresses.
- Do not include internal domain names on externally facing certificates.
    - If a server is accessible using both internal and external FQDNs, configure it with multiple certificates.

Carefully Consider the use of Wildcard Certificates

Wildcard certificates can be convenient, however they violate the principal of least privilege, as a single certificate is valid for all subdomains of a domain (such as *.example.org). Where multiple systems are sharing a wildcard certificate, the likelihood that the private key for the certificate is compromised increases, as the key may be present on multiple systems. Additionally, the value of this key is significantly increased, making it a more attractive target for attackers.

The issues around the use of wildcard certificates are complicated, and there are various other discussions of them online.

When risk assessing the use of wildcard certificates, the following areas should be considered:

- Only use wildcard certificates where there is a genuine need, rather than for convenience.
  - Consider the use of the ACME to allow systems to automatically request and update their own certificates instead.
- Never use a wildcard certificates for systems at different trust levels.
  - Two VPN gateways could use a shared wildcard certificate.
  - Multiple instances of a web application could share a certificate.
  - A VPN gateway and a public webserver **should not** share a wildcard certificate.
  - A public webserver and an internal server **should not** share a wildcard certificate.
- Consider the use of a reverse proxy server which performs TLS termination, so that the wildcard private key is only present on one system.
- A list of all systems sharing a certificate should be maintained to allow them all to be updated if the certificate expires or is compromised.
- Limit the scope of a wildcard certificate by issuing it for a subdomain (such as `*.foo.example.org`), or a for a separate domain.

Use an Appropriate Certification Authority for the Application's User Base

In order to be trusted by users, certificates must be signed by a trusted certificate authority (CA). For Internet facing applications, this should be one of the CAs which are well-known and automatically trusted by operating systems and browsers.

The LetsEncrypt CA provides free domain validated SSL certificates, which are trusted by all major browsers. As such, consider whether there are any benefits to purchasing a certificate from a CA.

For internal applications, an internal CA can be used. This means that the FQDN of the certificate will not be exposed (either to an external CA, or publicly in certificate transparency lists).

However, the certificate will only be trusted by users who have imported and trusted the internal CA certificate that was used to sign them.

Use CAA Records to Restrict Which CAs can Issue Certificates

Certification Authority Authorization (CAA) DNS records can be used to define which CAs are permitted to issue certificates for a domain. The records contains a list of CAs, and any CA who is not included in that list should refuse to issue a certificate for the domain. This can help to prevent an attacker from obtaining unauthorized certificates for a domain through a less-reputable CA. Where it is applied to all subdomains, it can also be useful from an administrative perspective by limiting which CAs administrators or developers are able to use, and by preventing them from obtaining unauthorized wildcard certificates.

Always Provide All Needed Certificates

In order to validate the authenticity of a certificate, the user's browser must examine the certificate that was used to sign it and compare it to the list of CAs trusted by their system. In many cases the certificate is not directly signed by a root CA, but is instead signed by an intermediate CA, which is in turn signed by the root CA.

If the user does not know or trust this intermediate CA then the certificate validation will fail, even if the user trusts the ultimate root CA, as they cannot establish a chain of trust between the certificate and the root. In order to avoid this, any intermediate certificates should be provided alongside the main certificate.

Consider the use of Extended Validation Certificates

Extended validation (EV) certificates claim to provide a higher level of verification of the entity, as they perform checks that the requestor is a legitimate legal entity, rather than just verifying the ownership of the domain name like normal (or "Domain Validated") certificates. This can effectively be viewed as the difference between "This site is really run by Example Company Inc." vs "This domain is really example.org".

Historically these displayed differently in the browser, often showing the company name or a green icon or background in the address bar. However, as of 2019 both Chrome and Firefox have announced that they will be removing these indicators, as they do not believe that EV certificates provide any additional protection.

There is no security downside to the use of EV certificates. However, as they are significantly more expensive than domain validated certificates, an assessment should be made to determine

whether they provide any additional value

Application

Use TLS For All Pages

TLS should be used for all pages, not just those that are considered sensitive such as the login page. If there are any pages that do not enforce the use of TLS, these could give an attacker an opportunity to sniff sensitive information such as session tokens, or to inject malicious JavaScript into the responses to carry out other attacks against the user.

For public facing applications, it may be appropriate to have the web server listening for unencrypted HTTP connections on port 80, and then immediately redirecting them with a permanent redirect (HTTP 301) in order to provide a better experience to users who manually type in the domain name. This should then be supported with the HTTP Strict Transport Security (HSTS) header to prevent them accessing the site over HTTP in the future.

Do Not Mix TLS and Non-TLS Content

A page that is available over TLS should not include any resources (such as JavaScript or CSS) files which are loaded over unencrypted HTTP. These unencrypted resources could allow an attacker to sniff session cookies or inject malicious code into the page. Modern browsers will also block attempts to load active content over unencrypted HTTP into secure pages.

Use the "Secure" Cookie Flag

All cookies should be marked with the "Secure" attribute, which instructs the browser to only send them over encrypted HTTPS connections, in order to prevent them from being sniffed from an unencrypted HTTP connection. This is important even if the website does not listen on HTTP (port 80), as an attacker performing an active man in the middle attack could present a spoofed webserver on port 80 to the user in order to steal their cookie.

Prevent Caching of Sensitive Data

Although TLS provides protection of data while it is in transit, it does not provide any protection for data once it has reached the requesting system. As such, this information may be stored in the cache of the user's browser, or by any intercepting proxies which are configured to perform TLS decryption.

Where sensitive data is returned in responses, HTTP headers should be used to instruct the browser and any proxy servers not to cache the information, in order to prevent it being stored or returned to other users. This can be achieved by setting the following HTTP headers in the response:

```
Cache-Control: no-cache, no-store, must-revalidate
Pragma: no-cache
Expires: 0
```

Use HTTP Strict Transport Security

HTTP Strict Transport Security (HSTS) instructs the user's browser to always request the site over HTTPS, and also prevents the user from bypassing certificate warnings. See the HTTP Strict Transport Security cheatsheet for further information on implementing HSTS.

Consider the use of Client-Side Certificates

In a typical configuration, TLS is used with a certificate on the server so that the client is able to verify the identity of the server, and to provide an encrypted connection between them. However, there are two main weaknesses with this approach:

- The server does not have any mechanism to verify the identity of the client
- The connection can be intercepted by an attacker who is able to obtain a valid certificate for the domain.
    - This is most commonly used by businesses to carry out inspection of TLS traffic by installing a trusted CA certificate on there client systems.

Client certificates address both of these issues by requiring that the client proves their identity to the server with their own certificate. This not only provides strong authentication of the identity of the client, but also prevents an intermediate party from performing TLS decryption, even if they have trusted CA certificate on the client system.

Client certificates are rarely used on public systems due to a number of issues:

- Issuing and managing client certificates introduces significant administrative overheads.
- Non-technical users may struggle to install client certificates.
- TLS decryption used by many organisations will cause client certificate authentication to fail.

However, they should be considered for high-value applications or APIs, especially where there are a small number of technically sophisticated users, or where all users are part of the same organisation.

Consider Using Public Key Pinning

Public key pinning can be used to provides assurance that the server's certificate is not only valid and trusted, but also that it matches the certificate expected for the server. This provides protection against an attacker who is able to obtain a valid certificate, either by exploiting a weakness in the validation process, compromising a trusted certificate authority, or having administrative access to the client.

Public key pinning was added to browsers in the HTTP Public Key Pinning (HPKP) standard. However, due to a number of issues, it has subsequently been deprecated and is no longer recommended or supported by modern browsers.

However, public key pinning can still provide security benefits for mobile applications, thick clients and server-to-server communication. This is discussed in further detail in the Pinning Cheat Sheet.

Related Articles

- OWASP - TLS Cipher String Cheat Sheet
- OWASP - Testing for SSL-TLS, and OWASP Guide to Cryptography
- OWASP - Application Security Verification Standard (ASVS) - Communication Security Verification Requirements (V9)
- Mozilla - Mozilla Recommended Configurations
- NIST - SP 800-52 Rev. 1 Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations
- NIST - NIST SP 800-57 Recommendation for Key Management, Revision 3, Public DRAFT
- NIST - SP 800-95 Guide to Secure Web Services
- IETF - RFC 5280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
- IETF - RFC 2246 The Transport Layer Security (TLS) Protocol Version 1.0 (JAN 1999)
- IETF - RFC 4346 The Transport Layer Security (TLS) Protocol Version 1.1 (APR 2006)
- IETF - RFC 5246 The Transport Layer Security (TLS) Protocol Version 1.2 (AUG 2008)

- Bettercrypto - Applied Crypto Hardening: HOWTO for secure crypto settings of the most common services)