An NCC Group Publication

“SS-Hell: the Devil is in the details”

Or

“How organisations can properly configure SSL services to ensure the integrity and confidentiality of data in transit”

Prepared by:

Will Alexander

Jerome Smith

Contents

Document History

Issue No.	**Issue Date**	**Change Description**
1.0	06-11-2014	Initial release
1.1	27-11-2014	Additional advice added

1 Introduction

Penetration test reports commonly contain mention of vulnerabilities in SSL/TLS (hereafter referred to as just SSL). In many cases, this is due to system administrators not understanding the details of these services' configuration, and assuming that simply using SSL provides security. The issues identified during penetration tests are usually low in severity, as an average attacker would find them difficult to exploit, but they should be taken seriously to ensure that data in transit is properly secured.

It is worth noting that as well as providing confidentiality and integrity, SSL (when implemented well) also provides organisations with a means to demonstrate to users that they can be trusted to handle their information. This should not be overlooked as organisations increasingly use security to differentiate themselves from their competition.

In this whitepaper we discuss how organisations can avoid SSL issues commonly found during penetration tests, ensure that data in transit is properly secured and ultimately instil in users a sense of confidence that their information is adequately protected.

2 Protocols

SSL and TLS are cryptographic protocols designed to provide communication security on the Internet. Without these protocols, e-commerce and other activities on the Internet requiring confidentiality and integrity would not be possible. There are a number of different versions of SSL and TLS, with newer versions fixing security vulnerabilities in those that came before. It is worth noting the technicality that in 1999 SSL was superseded by TLS, which is an active protocol under constant review; in contrast, SSL will never be updated.

- **SSL version 2 must not be used** by SSL services because it suffers from several serious cryptographic flaws and, as such, has been deprecated for a number of years.

  SSL version 2 is known to suffer from a number of problems[1] including:

  - No protection from man-in-the-middle attacks during the handshake.
  - Weak message authentication relying on the MD5 hash function.
  - The same cryptographic keys used for message authentication and encryption.
  - Vulnerability to truncation attacks by forged TCP FIN packets.
- **SSL version 3 must not be used** by SSL services. Although SSLv3 has many significant security advantages over SSLv2, such as using key-based message authentication that also protects the handshake, TLS nonetheless uses more robust algorithms. This gap between SSLv3 and TLS was highlighted by an attack nicknamed "POODLE", which was publicly disclosed in October 2014 and is covered separately in the "Vulnerabilities" section. It is also noteworthy that SSL version 3 is not FIPS 140-2 compliant[2].
- **TLS 1.0 should not be used** by SSL services. TLS 1.0 was defined as an upgrade to SSL 3.0, and can be considered to be SSL version 3.1. That being said, the RFC[3] states that "*the differences between this protocol and SSL 3.0 are not dramatic, but they are significant enough to preclude interoperability between TLS 1.0 and SSL 3.0*". For this reason TLS 1.0 includes a mechanism that allows a TLS implementation to downgrade the connection to SSL version 3, and as a result security is weakened.

[1] https://tools.ietf.org/html/rfc6176
[2] http://csrc.nist.gov/publications/nistpubs/800-135-rev1/sp800-135-rev1.pdf
[3] https://www.ietf.org/rfc/rfc2246.txt

TLS 1.0 is also vulnerable to the BEAST attack[4], which is covered separately in the "Vulnerabilities" section. Although browser vendors produced workarounds, later versions of TLS were not inherently vulnerable to this cipher block chaining attack since the implicit initialisation vector (IV) used in earlier versions was replaced with an explicit IV[5].

Therefore, **TLS versions 1.1 and 1.2 should be used** where possible. There are currently no known serious security issues affecting these versions but only v1.2 supports some of the more modern cipher suites that add improvements such as the use of SHA-256 for message integrity. As older browsers may not fully support these protocols, they should therefore be *preferred* to allow modern browsers to use them, with TLSv1 cipher suites acting as a fallback. The only significant reason to continue to support SSLv3 is to cover Internet Explorer version 6, which does not support TLSv1 by default – and the POODLE attack has put that reasoning under close scrutiny. Thankfully, this browser is no longer widely used[6].

3 Cipher Suites

Cipher suites specify various algorithms and properties by which data being sent is authenticated and encrypted. It is vitally important that only strong cipher suites are used, to provide resilience against attack. There are a number of cipher suites that are deemed insecure, and their use is discouraged:

- **Null ciphers must never be used** by SSL services. They offer no encryption whatsoever. They are disabled by default in most SSL implementations, and many clients are unlikely to support them.
- **Cipher suites offering no authentication must never be used** by SSL services. These ciphers offer no authentication, instead using anonymous Diffie-Hellman key exchanges, and as such are vulnerable to man-in-the-middle attacks. Keys exchanged between client and server are not authenticated, and can be altered by an attacker able to intercept and tamper with traffic between the two parties.
- **EXPORT ciphers must not be used** by SSL services as they are, by design, weak. The key lengths used by these ciphers (usually 40 or 56 bits) are such that they can be cracked within a reasonable amount of time using sophisticated, but commercially available, hardware. These ciphers were originally intended to comply with US export laws, but nowadays these restrictions do not apply.
- Similarly, **DES ciphers must not be used** by SSL services since they are based on the deprecated Data Encryption Standard (DES) and use keys with effective lengths of only 56 bits. (In reality, the key length used by DES is 64 bits, but only 56 are used for encryption.)
- **Triple DES ciphers should not be used** as, while they nominally use a 168-bit key, they have been shown to provide at best an effective key strength of 112 bits[7], which is less than the recommended 128 bits. Cipher suites that use 3DES are identifiable by the "3DES" or "CBC3" label. (Triple DES is also relatively slow, so there may be performance benefits from dropping support for it.)
- **RC4 ciphers should not be used** by SSL services since they are based on a deprecated, and theoretically weak, algorithm. Attacks against communications encrypted using these ciphers are currently infeasible, but (as cryptographers love to quote) "*attacks always get*

[4] https://bug665814.bugzilla.mozilla.org/attachment.cgi?id=540839
[5] http://tools.ietf.org/html/rfc4346#appendix-F.3
[6] IE6 disappeared from the top ten W3Counter list (http://www.w3counter.com/globalstats.php) in January 2012; more information can be found at https://www.modern.ie/en-us/ie6countdown)
[7] http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf

better; they never get worse"[8]. These attacks are thus only going to become more practical in the future, and some browser vendors, including Microsoft[9], have begun to take steps to discourage the use of RC4.

Forward Secrecy is provided by a number of cipher suites and their use is encouraged. They can be identified by the keyword "ephemeral", which in the cipher suite nomenclature is seen as an additional "E" to the abbreviation of the key exchange mechanism. For example, DH**E** in contrast with DH (Diffie-Hellman) or ECDH**E** in contrast with ECDH (Elliptic Curve Diffie-Hellman). These cipher suites **should be preferred wherever possible** because they:

- Generate random public keys per session for the purposes of key agreement; and
- Do not use any sort of deterministic algorithm when doing so.

Therefore, even if an attacker is able to obtain the private key, they would be unable to decrypt messages from earlier sessions. It should be noted that cipher suites that afford perfect secrecy add a computational overhead to the set-up of the connection, but those that employ elliptic curve cryptography are currently the most efficient[10].

Servers offering SSL services are able to support a number of cipher suites to maintain compatibility with different clients. They can be arranged in order of preference to encourage the use of more robust cipher suites by clients that support them. Mozilla[11] offers a resource on cipher suites, which includes information on ordering.

4 Certificates

SSL certificates for web servers provide a mechanism to bind together a domain name, a public key and an organisation that exists in the physical world. The certificates are cryptographically signed by trusted third parties, known as Certificate Authorities (CAs), which verify the organisation owns the associated domain name. Client software explicitly trusts the CAs and, therefore, implicitly trusts the identity of any server using a certificate signed by a trusted CA[12]. Of course, whether or not the CA's verification checks are robust or the authenticated endpoint is trustworthy in other respects is another matter altogether. It's important to note that some certificates are just "Domain Validated" (DV), where the CA only verifies that the owner controls an email address within the domain specified in the certificate. Issuing DV certificates does not entail any checks on the organisation, in contrast to "Organisation Validated" (OV) certificates. While often cheaper, it is recommended that **organisations should not use DV certificates** in production environments.

4.1 Self-Signed or Untrusted Certificates

Not all certificates are required to be signed by a trusted CA. Certificates can be signed by any third party, or even self-signed by the server hosting the SSL service. However, these certificates would not be implicitly trusted by client software because they have not been signed by a recognised authority, and typically a warning message would be displayed warning the user that the server's identity cannot be verified.

Many users are unlikely to be able to distinguish between warnings about the genuine but cryptographically untrusted certificate and those triggered by spoofed certificates, thus making them

[8] http://tools.ietf.org/html/rfc4270#section-6
[9] http://technet.microsoft.com/en-gb/security/advisory/2868725
[10] http://vincent.bernat.im/en/blog/2011-ssl-perfect-forward-secrecy.html#some-benchmarks
[11] https://wiki.mozilla.org/Security/Server_Side_TLS
[12] Usually the certificate used to sign a website's SSL certificate is an *intermediate* CA, which is only trusted because it has been signed by a *root* CA. Only root CAs are explicitly trusted by browsers, while any certificate in a chain of trust that terminates at a root CA is implicitly trusted.

susceptible to man-in-the-middle attacks. Therefore, **self-signed certificates, or certificates signed by an untrusted third party, must not be used** on SSL services accessible from the Internet.

Ideally, all SSL certificates should be signed by a trusted CA, but doing so for all SSL services hosted internally could be expensive. A risk assessment should be performed to determine whether the cost of having genuine certificates for every internal SSL service is out-weighed by the potential impact of a man-in-the-middle attack.

Alternatively, browsers can be instructed to explicitly trust self-signed or untrusted certificates if they are known to be genuine, to avoid warnings being generated.

4.2 Mismatched Hostnames

It is important to mention that the underlying encryption used to protect communications is unaffected by the use of self-signed or untrusted certificates. The issue lies in the fact that the user is unable to verify the identity of the server hosting the SSL service. In other words, the data sent to the server will be encrypted but the end-point has not been authenticated. Similarly, when the Common Name (CN) field of the certificate does not match the hostname on which the SSL service is running, clients will generate a warning because they are unable to verify the server's identity. Again, this does not affect the encryption, but does undermine the security of the service, since the identity of the server cannot be verified. The likelihood of man-in-the-middle attacks succeeding increases if users become accustomed to clicking through browser warnings. Therefore, **the CN field of the certificate must match the fully-qualified domain name of the server** running the SSL service.

If a server has multiple hostnames, then there are two solutions:

- Multiple certificates can be used. For example, consider a web application accessible at https://foo.nccgroup.com and https://bar.nccgroup.com. One certificate should be acquired for foo.nccgroup.com, and a second certificate for bar.nccgroup.com. In both cases, the hostname would be present in the Subject's Common Name (CN). This relies on the browser supporting the Server Name Indication (SNI) extension, which indicates the hostname the client is requesting during the TLS handshake. Without this, the server would not know which certificate to send to the client.
- A single certificate can carry Subject Alternative Names (SANs), which provide a specific list of hostnames for which the certificate is valid. In the previous example, the certificate could list the Subject's CN as foo.nccgroup.com, with bar.nccgroup.com as a Subject Alternative Name (in fact foo.nccgroup.com should also be listed in the Subject Alternative Name list to conform to RFC 6125). These certificates are sometimes referred to as "multiple domain certificates". This method also allows one certificate to be installed on different servers.
  A common pitfall of certificates is that they are invalid if the hostname is missing the “www” prefix, which users often leave out to save typing. This can be avoided if the shortened hostname is added as a SAN.

4.3 Wildcard Certificates

An alternative to SANs is the use of a wildcard certificate. A wildcard certificate can be used for an SSL service running on any host matching the Common Name (CN) field. For example, if the CN field is *.nccgroup.com, then the certificate could be used on foo.nccgroup.com or bar.nccgroup.com. The use of SSL wildcard certificates minimises certificate management costs, but their use is not advised. Care should be taken if using wildcard certificates to reduce operational costs.

If a server using a wildcard certificate is compromised, the certificate and its corresponding private key could be stolen and used to target any of the domains covered by the wildcard. For example, an attacker could use a man-in-the-middle attack to proxy the TLS traffic, compromising the confidentiality and integrity of the data, or use a DNS attack to direct victims to spoofed sites that would look legitimate to users. Wildcard certificates can be thought to violate the principle of least privilege, since they require users to implicitly trust all hosts in the domain.

Finally, note that **wildcard certificates** cannot achieve Extended Validation (EV) status[13], and **should not be used**.

4.4 Extended Validation Certificates

Extended Validation (EV) certificates do not offer any additional technical security for SSL services, but do increase user confidence in the service. This is because organisations are required to undergo more rigorous checks to ensure that the certificate requester is the verified legal entity for the domain name. Support for EV certificates varies between browsers, but typically a portion of the address bar will be highlighted green to indicate to users that the site has an EV certificate.

EV certificates, like those used by https://www.nccgroup.com, **should be used** to increase user confidence when connecting to a secure site.

4.5 Certificate Validity Period

Expired certificates (that is, certificates used after the timestamp in the “Not Valid After” field) will also cause clients to generate warnings, making it difficult for many users to distinguish between a warning about the genuinely expired certificate and one triggered by spoofed certificates. Once again, this undermines the trust placed in the secured communications between client and server.

Many high profile organisations have been caught out by letting certificates expire. In May 2014 a certificate used for one of Apple’s software update servers expired[14], producing a warning for anyone trying to connect to it. Although it is clearly in the issuing CA’s commercial interest to remind you to renew a certificate, it is recommended that organisations try to be proactive by setting reminders to renew certificates in time.

Similarly, certificates used before the timestamp in the “Not Valid Before” field will cause clients to generate warnings. Therefore, **the current date must fall between the timestamps in the “Not Valid Before” and “Not Valid After” fields of the certificate**.

4.6 Revoked Certificates

Sometimes it might be necessary to revoke a certificate before its expiry date. For example, a large number of certificates were revoked after it was revealed that exploitation of the Heartbleed vulnerability could lead to the recovery of a certificate’s private key[15].

When connecting to a TLS service, clients should verify the revocation status as part of the overall validity check for the certificate, using either a Certificate Revocation List (CRL) or an Online Certificate Status Protocol (OCSP) record. CRLs, however, have several problems: they have grown huge, are updated only periodically, and can take a significant amount of time to download.

OCSP is much more lightweight, as only one record is retrieved at a time, but the request must be made to a third party OCSP responder, which adds latency. Under some circumstances the request for the OCSP record might fail completely and clients will generally “fail open” in this event, assuming the certificate to be valid. A man-in-the-middle attacker in possession of a compromised certificate

---

[13] https://cabforum.org/wp-content/uploads/EV-SSL-Certificate-Guidelines-Version-1.5.1.pdf (section 9.2.3)
[14] http://www.macrumors.com/2014/05/25/apple-software-update-invalid/
[15] http://blog.cloudflare.com/searching-for-the-prime-suspect-how-heartbleed-leaked-private-keys/

can take advantage of this fact by interfering with the OCSP request or response in some way, thus rendering the check useless. (It is primarily for this reason that Chrome decided to use a proprietary way of managing revocation driven by its so-called "CRLset"[16].)

The next advance was to allow the server to send its cached OCSP record during the TLS handshake, therefore bypassing the OCSP responder. This mechanism saves a roundtrip between the client and the OCSP responder, and is called OCSP Stapling.

The server will send a cached OCSP response only if the client requests it. Most servers will cache OCSP responses for up to 48 hours. At regular intervals, the server will connect to the OCSP responder of the CA to retrieve a fresh OCSP record. The location of the OCSP responder is taken from the Authority Information Access field of the signed certificate.

There is still the scenario in which an attacker could steal a valid certificate, create a duplicate web site using the stolen certificate, and disable OCSP stapling. If the client is set to fail silently, then it will assume that everything is fine and proceed with the connection to the malicious web site.

There are currently two proposals to solve this problem:

- Add a "must staple" assertion to the site's security certificate[17]
- Create a new HTTP response header[18]

OCSP stapling should be used to prevent clients from accepting revoked certificates, with the anticipation that proposals such as those above will be implemented in the future to strengthen the effectiveness of the OCSP mechanism.

4.7 Certificates Signed Using Weak Algorithms or Containing a Short Key

Certificates signed using, or containing, keys with lengths less than 2048 bits are considered insecure. Industry standards set by the Certification Authority/Browser (CA/B) Forum state that certificates issued after 01/01/2014 must use keys at least 2048 bits in length[19]. Furthermore, some browsers[20] will issue warnings when certificates containing keys less than 2048 bits in length are encountered.

Such certificates are considered insecure because it might be possible to create spoofed certificates by brute-forcing the signing key. To brute-force the signing key it would be necessary to factor very large numbers that form the RSA modulus. Currently, numbers as large as 768 bits have been factored[21]. Factoring techniques are likely to improve over the coming years so it is prudent to future-proof current certificates by using 2048-bit keys.

Similarly, certificates signed using weak algorithms such as MD2, MD4, or MD5 are considered insecure. These algorithms are prone to collision attacks which could be exploited by a determined attacker to generate another certificate with the same digital signature, thus allowing the attacker to masquerade as the legitimate SSL service. SHA-1 is also showing its age. Some browser vendors, such as Microsoft[22] and Google[23], have announced plans to "*sunset*" the algorithm by the end of 2015 or sooner. In order to avoid these warnings, **certificates must not be signed using weak algorithms or contain short keys**.

[16] https://www.imperialviolet.org/2014/04/19/revchecking.html
[17] http://tools.ietf.org/html/draft-hallambaker-muststaple-00
[18] http://www.ietf.org/mail-archive/web/tls/current/msg10351.html
[19] https://www.cabforum.org/Baseline_Requirements_V1.pdf (Appendix A)
[20] https://cabforum.org/pipermail/public/2013-September/002233.html
[21] http://www.emc.com/emc-plus/rsa-labs/historical/the-rsa-challenge-numbers.htm
[22] https://technet.microsoft.com/en-us/library/security/2880823.aspx
[23] http://googleonlinesecurity.blogspot.co.uk/2014/09/gradually-sunsetting-sha-1.html

5 Vulnerabilities

A number of vulnerabilities have been identified in the SSL and TLS protocols and their various implementations. The following is a list of the most serious vulnerabilities disclosed in recent years, but is by no means an exhaustive list. **The latest stable and supported versions of SSL implementations must be used**, wherever possible.

5.1 CVE-2009-3555 (SSL Session Renegotiation Vulnerability)

SSL renegotiation allows clients and browsers to renegotiate how data is secured in transit by triggering a new SSL handshake. It was initially designed as a mechanism to increase the security of an ongoing SSL communication, by triggering the renewal of the cryptographic keys, and could be performed by both client and server. This renewal isn't needed with modern cipher suites but renegotiation can be used by a server to request a client certificate (in order to perform client authentication) when the client tries to access specific, protected resources on the server. Therefore, there is no longer a need for *clients* to renegotiate a connection. Servers that allow client-initiated renegotiations are vulnerable to denial of service attacks since the amount of computational resources required to renegotiate a connection is much greater for the server than it is for the client. Therefore, **SSL services must not support client-initiated renegotiation**.

Furthermore, in 2009 a protocol flaw in the renegotiation mechanism was found[24]. This flaw allowed a man-in-the-middle to inject plaintext at the beginning of a SSL communication. For example, the attacker could inject an HTTP request into a connection a victim was making in such a way that it could be authenticated by the victim's pre-existing cookies. As well as not supporting client-initiated renegotiations, SSL services should be updated to include a fix that makes session renegotiation secure (as defined by RFC 5746[25]).

5.2 CVE-2011-3389 (BEAST)

The BEAST (**B**rowser **E**xploit **A**gainst **S**SL/**T**LS) exploit is a client-based attack that exploits a vulnerability in the use of Initialisation Vectors with block ciphers (such as AES) in Cipher Block Chaining (CBC) mode. The attack would allow a man-in-the-middle attacker to recover plaintext values by encrypting the same message multiple times.

Most modern browsers are now immune to the BEAST attack, but the server workaround is to either:

- Use ciphers based on RC4, which has its own problems; or
- Support only versions of TLS greater than 1.0.

Wherever possible, the latter is recommended since the theoretical issues with RC4 are likely to be exploitable in the future.

5.3 CVE-2012-4929 (CRIME)

CRIME (**C**ompression **R**atio **I**nfo-leak **M**ade **E**asy) is an exploit that reveals web cookies transmitted using the HTTPS or SPDY protocols with data compression to perform session hijacking. The attack works by observing changes in the size of requests, which contain the cookie and variable content controlled by the attacker. When the size of the compressed content is reduced, it can be inferred that some part of the injected content is likely to match another part of the request, including the cookie.

[24] https://community.qualys.com/blogs/securitylabs/2009/11/05/ssl-and-tls-authentication-gap-vulnerability-discovered
[25] http://tools.ietf.org/html/rfc5746

The CRIME exploit attacks browsers, and the latest releases are not vulnerable. However, it is prudent that servers mitigate the threat by disabling HTTPS compression. Regarding SPDY, the situation is a little more complicated. Firefox tackled the problem by disabling header compression[26], which will protect cookies. Chrome also disabled header compression, but then quickly adopted another trick to regain some of the lost efficiency[27]. It was less clear what the position of other browsers was (probably because they were not supporting SPDY at the time that the CRIME attack was disclosed). Until the release of version 4 of SPDY, which is said to tackle CRIME but is still in development in tandem with HTTP/2, disabling support for SPDY would be the most prudent approach.

5.4 CVE-2014-0160 (Heartbleed)

Heartbleed is a serious vulnerability in versions of OpenSSL between 1.0.1 and 1.0.1f inclusive (also 1.0.2-beta1). Exploiting this vulnerability allows an attacker to read the contents of the host's memory, which could compromise usernames and passwords, session cookies, and even the private key corresponding to a certificate.

5.5 CVE-2014-0224 (ChangeCipherSpec)

This vulnerability, when exploited, allows an attacker acting as a man-in-the-middle to force the use of weak keys, which ultimately allows them to decrypt traffic between the client and server.

This vulnerability is only exploitable if both the server *and* client are using vulnerable versions of OpenSSL. That is, this vulnerability can only be exploited if:

- The server is using a version of OpenSSL less than 1.0.1h; and
- The client is also using a vulnerable version of OpenSSL, namely:
  - less than 1.0.1h;
  - less than 1.0.0m; and
  - less than 0.98za.

Note that many browser vendors do not use OpenSSL for their TLS implementations. The exception to this is Android, and thus mobile users would be the most likely user-base to be affected by an attack.

5.6 CVE-2014-3566 (POODLE)

POODLE (**P**adding **O**racle **O**n **D**owngraded **L**egacy **E**ncryption) is related to the BEAST attack. It targets block ciphers in CBC mode within a SSLv3 connection. Like BEAST, there are a number of caveats that the attacker must fulfil such as being a man-in-the-middle and forcing the victim's browser to issue multiple requests to a website of interest. In most cases, SSLv3 will not be selected by the client and server but it has long been known that a man-in-the-middle can interfere with the TLS handshake of a victim's browser to try to force it to use SSLv3 as a fallback. Users in hostile environments such as open Wi-Fi networks will be more at risk.

Disabling support for SSLv3 is the best solution, as it also affords protection from future vulnerabilities. Old browsers that support SSLv3 as their highest available protocol will no longer be able to connect, however. As previously stated, the most notable example of this is Internet Explorer 6 in its default state. If support for SSLv3 is required, it is recommended to apply the "TLS_FALLBACK_SCSV" mechanism to ensure that clients are not subject to a forced downgrade

[26] https://bugzilla.mozilla.org/show_bug.cgi?id=779413
[27] https://www.imperialviolet.org/2012/09/21/crime.html

attack. This works by allowing the browser to signal that it is intentionally attempting to connect with a lower protocol version than it supports, ensuring that SSLv3 is only used when necessary[28]. Support for this is currently limited but is expected to increase in the light of the POODLE attack. OpenSSL added an implementation of TLS_FALLBACK_SCSV in version 1.0.1j.

6 HTTP Considerations

The majority of SSL services exposed will be used to encrypt web traffic between the browser and web server via HTTPS. There are a number of additional factors to consider when using SSL with HTTPS.

6.1 Reversion to HTTP Post-Authentication

Some applications, upon successful authentication over an encrypted HTTPS channel, will revert to using HTTP under the assumption that the username and password are all that needs to be secured from prying eyes. This is a flawed assumption, as any post-authentication requests and responses, including session tokens, are transmitted in clear text. This could allow an attacker in a position to sniff network traffic to eavesdrop on the session, steal the session cookie, and hijack the user's active session.

Facebook and Twitter previously contained this flaw until it was infamously exploited by the Firesheep Firefox extension to hijack users' sessions[29].

All post-authentication traffic, as well as the login forms themselves, must be transmitted over HTTPS, and any attempt to downgrade a request to HTTP should cause the application to redirect the browser to HTTPS. There is an argument that any application requiring security should *always* use HTTPS throughout because any content delivered using HTTP is susceptible to man-in-the-middle attacks, which could be used to tamper with HTTPS links. This is discussed below.

6.2 Mixed Content

Mixed content issues arise when web applications deliver page content using plaintext HTTP as well as HTTPS. An attacker would struggle to compromise the content delivered using HTTPS, but tampering with the plaintext content delivered using HTTP can lead to a range of problems from theft of data in the page acquired over HTTPS to phishing attacks. Exactly what can be achieved depends on the context of the insecure content.

One important aspect of this context is the nature of the content being served over HTTP. Clearly, tampering with images, videos and sound ("passive" content) delivered unencrypted will yield little for an attacker. However, by tampering with scripts or HTML ("active" content) a man-in-the-middle attacker may be able to capture sensitive information or hijack sessions – even if subsequent content is delivered using HTTPS.

Mixed content produces browser warnings that could reduce user confidence in the website, and some browsers[30] are now blocking cases of mixed *active* content, which could lead to functional side-effects.

All "active" content must be delivered using HTTPS. "Passive" content can be delivered using HTTP, but ideally all content should be delivered using HTTPS.

[28] A detailed treatment can be found at http://www.exploresecurity.com/poodle-and-the-tls_fallback_scsv-remedy/
[29] http://codebutler.github.io/firesheep/
[30] https://blog.mozilla.org/tanvi/2013/04/10/mixed-content-blocking-enabled-in-firefox-23/

6.3 HTTP Strict Transport Security Headers

The HTTP Strict Transport Security header – literally “Strict-Transport-Security” (STS) – is used to instruct the browser to access a site only over a secure connection. It also specifies a period of time during which this instruction is valid.

The header primarily mitigates SSL-stripping man-in-the-middle attacks, made famous by Moxie Marlinspike in his 2009 BlackHat talk "New Tricks for Defeating SSL in Practice"[31]. This attack transparently converts links to secure HTTPS resources into insecure HTTP equivalents. This can only be achieved if the page with the HTTPS links is itself delivered over HTTP. The user can see that the main page is not encrypted, but has no simple way to determine if and when the site will switch to a secure connection. The impact is that sensitive data (such as credentials or session cookies) are leaked over HTTP, which can be sniffed and reused by the attacker. The STS header prevents this attack by instructing the browser to only access the site using a secure connection. Of course, this mitigation requires that the user has visited the site over SSL before and has received a STS directive (unless it features in Chrome’s pre-load list[32]).

Most modern browsers support the Strict-Transport-Security header. Notably absent is Internet Explorer, but support for this header is expected in an upcoming version[33].

The HTTP Strict-Transport-Security header should be used, with appropriate directives. A recommended Strict-Transport-Security HTTP header might look like:

```
Strict-Transport-Security: max-age= 31536000; includeSubDomains
```

This example ensures that all connections for the next 31536000 seconds (365 days) must be made using SSL. The *includeSubDomains* directive indicates that all subdomains of the site meet the requirements of the header. It’s also important to note that the header must not be set within a HTML meta tag as compliant browsers will ignore this (as well as when the header is set within an insecure HTTP response).

6.4 Secure Cookies

By specifying that a cookie is "Secure", the server can guarantee that conforming browsers will not send the cookie over unsecured HTTP connections, even if the cookie domain and scope matches that of the URL being requested. The absence of this flag means that affected cookies are at risk of being transmitted over unencrypted HTTP connections. Note that setting the Secure flag does not protect the cookie in all scenarios – for example, without the HttpOnly flag the cookie would be at risk from cross-site scripting attacks.

All sensitive cookies issued over HTTPS must make use of the Secure flag if they are to be kept confidential from network eavesdroppers. A correct Set-Cookie HTTP header might look like:

```
Set-Cookie: JSESSIONID=1A530637289A03B07199A44E8D531427; path=/; HttpOnly; Secure
```

[31] https://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf
[32] https://hstspreload.appspot.com/
[33] http://status.modern.ie/httpstricttransportsecurityhsts

6.5 Cacheable HTTPS Content

Some content delivered using HTTPS is likely to be considered sensitive, and as a result it is important that browsers are instructed about what should and should not be cached. Unless otherwise directed, browsers may store a local cached copy of received content, often with the aim of improving website responsiveness for the end user when the same content is subsequently requested. However, if sensitive information in application responses is stored in the local cache, then it could be retrieved by other users, malware, or attackers who have access to the same computer at a later date.

Cache control directives must be used to instruct browsers (and any intermediary HTTPS proxies) not to store local copies of any sensitive information. Ideally, the following HTTP headers should be included in all responses containing sensitive content, which can often be achieved by configuring the web server to prevent caching for relevant paths within the web root:

```
Cache-control: no-store, no-cache

Pragma: no-cache
```

Alternatively, most web development platforms allow control over the web server's caching directives from within individual scripts. These directives could be included in the HTML code:

```
<meta http-equiv='pragma' content='no-cache'>

<meta http-equiv='cache-control' content='no-store'>
```

Using HTTP headers is preferable, however, as meta directives may not be compatible across all browsers and platforms, and would not be applicable to a non-HTML response, such as a binary document format.

7 Conclusion

Configuring SSL services is not as straightforward as some might think, and there is much to consider when doing so correctly. In short, though, SSL services must (wherever possible and appropriate):

- Prefer TLS 1.1 and 1.2 cipher suites;
- Disable support for SSLv3 (along with SSLv2);
- Not use ciphers that are known to contain cryptographic weaknesses;
- Use ciphers that provide forward secrecy;
- Not use cipher suites that employ symmetric key lengths less than 128 bits;
- Use a valid OV certificate that:
  - Contains a key 2048 bits or more in length;
  - Is valid for a specific whitelist of hosts, and
  - Is signed by a trusted CA with a key 2048 bits or more in length using a secure hashing algorithm;
- Not support TLS compression;
- Not allow client-initiated renegotiations; and
- Be fully patched against all known security vulnerabilities.

Furthermore, if the SSL service is HTTPS, then the underlying application must:

- Ensure that all sensitive information is delivered securely;

- Ensure that all components of sensitive pages are delivered securely;
- Ensure that no sensitive information is cached by the browser; and
- Ensure that all cookies are marked with the *Secure* flag.