SSL/TLS Deployment Best Practices

Ivan Ristić

version 1.4 (8 December 2014)

Abstract

SSL/TLS is a deceptively simple technology. It is easy to deploy, and it just works… except when it does not. The main problem is that encryption is not often easy to deploy *correctly*. To ensure that TLS provides the necessary security, system administrators and developers must put extra effort into properly configuring their servers and developing their applications.

In 2009, we began our work on SSL Labs because we wanted to understand how TLS was used and to remedy the lack of easy-to-use TLS tools and documentation. We have achieved some of our goals through our global surveys of TLS usage, as well as the online assessment tool, but the lack of documentation is still evident. This document is a step toward addressing that problem.

Our aim here is to provide clear and concise instructions to help overworked administrators and programmers spend the minimum time possible to deploy a secure site or web application. In pursuit of clarity, we sacrifice completeness, foregoing certain advanced topics. The focus is on advice that is practical and easy to follow. For those who want more information, Section 6 gives useful pointers.

QUALYS® SSL LABS

www.ssllabs.com

1. Private Key and Certificate

The quality of the protection provided by TLS depends entirely on the private key, which lays down the foundation for the security, and the certificate, which communicates the identity of the server to its visitors.

1.1. Use 2048-bit Private Keys

Use 2048-bit RSA or 256-bit ECDSA private keys for all your servers. Keys of this strength are secure and should stay secure for a considerable amount of time. If you have 1024-bit RSA keys in production, replace them with stronger keys as soon as possible. If you believe that you need more than 2048 bits of security, consider using ECDSA keys, which have better performance characteristics. The drawback is that there is a small number of clients that don't support ECDSA and that you might need to deploy RSA and ECDSA keys in parallel to maintain good interoperability.

1.2. Protect Private Keys

Treat your private keys as an important asset, restricting access to the smallest possible group of employees while still keeping the arrangements practical. Recommended policies include the following:

- Generate private keys and *Certificate Signing Requests* (CSRs) on a trusted computer. Some CAs offer to generate keys and CSRs for you, but that's inappropriate.
- Password-protect keys to prevent compromise when they are stored in backup systems. Private key passwords don't help much in production because a knowledgeable attacker can always retrieve the keys from process memory. There are hardware devices that can protect private keys even in the case of server compromise, but they are expensive and thus justifiable only by organizations with strict security requirements.
- After compromise, revoke old certificates and generate new keys.
- Renew certificates every year, always with new private keys.

1.3. Ensure Sufficient Hostname Coverage

Ensure that your certificates cover all the names you wish to use with a site. For example, your main name is *www.example.com*, but you may also have *www.example.net* configured. Your goal is to avoid invalid certificate warnings, which will confuse your users and weaken their trust.

Even when there is only one name configured on your servers, remember that you cannot control how your users arrive at the site or how others link to it. In most cases, you should ensure that the certificate works with and without the *www* prefix (e.g., for both *example.com* and *www.example.com*). The rule of thumb is this: a secure web server should have a certificate that is valid for every DNS name configured to point to it.

Wildcard certificates have their uses, but should be avoided if using them means exposing the underlying keys to a larger group of people, and especially if crossing organizational boundaries. In other words, the fewer people who have access to the private keys, the better. Further, be aware that certificate sharing creates a bond that can be abused to transfer vulnerabilities from one web site to all other sites that use the same certificate.

1.4. Obtain Certificates from a Reliable CA

Select a *Certification Authority* (CA) that is reliable and serious about its certificate business and about security. Consider the following criteria when selecting your CA:

Security posture
: All CAs undergo regular audits (otherwise they wouldn't be able to operate as CAs), but some are more serious about security than others. Figuring out which ones are better in this respect is not easy, but one option is to examine their security history, and, more important, how they reacted to compromises and if they learned from their mistakes.

Substantial market share
: A CA that meets this criterion will not likely have all its certificates easily recalled, which was the case with some smaller ones in the past.

Business focus
: CAs whose activities constitute a substantial part of their business have everything to lose if something goes terribly wrong, and they probably won't neglect their certificate division by chasing potentially more lucrative opportunities elsewhere.

Services offered
: At minimum, your selected CA should provide support for both *Certificate Revocation List* (CRL) and *Online Certificate Status Protocol* (OCSP) revocation and provide an OCSP service with good performance. They should offer both domain-validated and *Extended Validation* (EV) certificates, ideally with your choice of public key algorithm. (Most web sites use RSA today, but ECDSA may become important in the future because of its performance advantages.)

Certificate management options
: If you need a large number of certificates and operate in a complex environment, choose a CA that will give you good tools to manage them.

Support

Choose a CA that will give you good support if and when you need it.

1.5. Use Strong Certificate Signature Algorithms

Certificate signature security depends on the strength of the signing private key and the strength of the used hashing function. Today, most certificates use the SHA1 hashing function, which is considered weak and borderline insecure. The industry is currently moving away from SHA1, which a process that must be completed by the end of 2016. After that SHA1 certificates won't be accepted any more.[1]

However, because Google Chrome warns about SHA1 certificates that expire even before the ultimate deadline,[2] you should immediately replace all your SHA1 certificates if they expire after 2015. Alternatively, you could move straight away to certificates that rely on the SHA2 algorithm family. But, before you do that, check that enough of your user base supports SHA2. Some older clients, for example IE6 running on Windows XP Service Pack 2 (still heavily used in some countries and organizations) don't.

2. Configuration

With correct TLS server configuration, you ensure that your credentials are properly presented to the site's visitors, that only secure cryptographic primitives are used, and that all known weaknesses are mitigated.

2.1. Deploy with Valid Certificate Chains

In most deployments, the server certificate alone is insufficient; two or more certificates are needed to establish a complete chain of trust. A common problem is configuring the server certificate correctly but forgetting to include other required certificates. Further, although these other certificates are typically valid for longer periods of time, they too expire, and when they do, they invalidate the entire chain. Your CA should be able to provide you with all the additional certificates required.

An invalid certificate chain renders the actual server certificate invalid and results in browser warnings. In practice, this problem is sometimes difficult to diagnose because some browsers can deal with these problems and reconstruct a complete correct chain, and some can't.

[1] SHA1 Deprecation Policy (Windows PKI blog, 12 November 2013)
[2] Gradually Sunsetting SHA-1 (The Chromium Blog, 5 September 2014)

2.2. Use Secure Protocols

There are five protocols in the SSL/TLS family: SSL v2, SSL v3, TLS v1.0, TLS v1.1, and TLS v1.2. Of these:

- SSL v2 is insecure and must not be used.
- SSL v3 is insecure when used with HTTP and weak when used with other protocols. It's also obsolete, which is why it shouldn't be used.
- TLS v1.0 is largely still secure; we do not know of major security flaws when they are used for protocols other than HTTP. When used with HTTP, it can *almost* be made secure with careful configuration.
- TLS v1.1 and v1.2 are without known security issues.

TLS v1.2 should be your main protocol. This version is superior because it offers important features that are unavailable in earlier protocol versions. If your server platform (or any intermediary device) does not support TLS v1.2, make plans to upgrade at an accelerated pace. If your service providers do not support TLS v1.2, require that they upgrade.

In order to support older clients, you need to continue to support TLS v1.0 and TLS v1.1 for the time being. With some workarounds (explained in subsequent sections), these protocols can still be considered secure enough for most web sites.

2.3. Use Secure Cipher Suites

To communicate securely, you must first ascertain that you are communicating directly with the desired party (and not through someone else who will eavesdrop), as well as exchanging data securely. In SSL and TLS, cipher suites are used to define how secure communication takes place. They are composed from varying building blocks with the idea of achieving security through diversity. If one of the building blocks is found to be weak or insecure, you should be able to switch to another.

Your goal should be thus to use only suites that provide authentication and encryption of 128 bits or stronger. Everything else must be avoided:

- Anonymous Diffie-Hellman (ADH) suites do not provide authentication.
- NULL cipher suites provide no encryption.
- Export key exchange suites use authentication that can easily be broken.
- Suites with weak ciphers (typically of 40 and 56 bits) use encryption that can easily be broken.

- RC4 is weaker than previously thought.[3] You should remove support for this cipher as soon as possible, but after checking for potential negative interoperability impact.
- 3DES provides about 112 bits of security. This is below the recommended minimum of 128 bits, but it's still strong enough. A bigger practical problem is that 3DES is much slower than the alternatives. Thus, we don't recommend it for performance reasons, but it can be kept at the end of the cipher list for interoperability with very old clients.

2.4. Control Cipher Suite Selection

In SSL v3 and later protocol versions, clients submit a list of cipher suites that they support, and servers choose one suite from the list to negotiate a secure communication channel. Not all servers do this well, however—some will select the first supported suite from the list. Having servers select the right cipher suite is critical for security (more about that in Section 2.7).

2.5. Support Forward Secrecy

Forward Secrecy[4] is a protocol feature that enables secure conversations that are not dependent on the server's private key. With cipher suites that do not support Forward Secrecy, someone who can recover a server's private key can decrypt all earlier encrypted conversations if they have them recorded. You need to support and prefer ECDHE suites in order to enable Forward Secrecy with modern web browsers. To support a wider range of clients, you should also use DHE suites as fallback after ECDHE.[5]

2.6. Disable Client-Initiated Renegotiation

In SSL/TLS, renegotiation allows parties to stop exchanging data in order to renegotiate how the communication is secured. There are some cases in which renegotiation needs to be initiated by the server, but there is no known need for clients to do so. Further, client-initiated renegotiation may make your servers easier to attack using *Denial of Service* (DoS) attacks.[6]

[3] On the Security of RC4 in TLS and WPA (Kenny Paterson et al.; 13 March 2013)
[4] Deploying Forward Secrecy (Qualys Security Labs; 25 June 2013)
[5] Increasing DHE strength on Apache 2.4.x (Ivan Ristić's blog; 15 August 2013)
[6] TLS Renegotiation and Denial of Service Attacks (Qualys Security Labs Blog, October 2011)

2.7. Mitigate Known Problems

Nothing is perfectly secure, and at any given time there may be issues with the security stack. It is good practice to keep an eye on what happens in the security world and to adapt to situations as necessary. At the very least, you should apply vendor patches as soon as they become available.

The following issues require your attention:

Disable insecure renegotiation

In 2009, the renegotiation feature was found to be insecure and the protocols needed to be updated.[7] Most vendors have issued patches by now or, at the very least, provided workarounds for the problem. Insecure renegotiation is dangerous because it is easy to exploit and has effects similar to *Cross-Site Request Forgery* (CSRF) and, in some cases, *Cross-Site Scripting* (XSS).

Disable TLS compression

In 2012, the CRIME attack[8] showed how information leakage introduced by TLS compression can be used by attackers to uncover parts of sensitive data (e.g., session cookies). Very few clients supported TLS compression then (and even fewer support it now), which means that it is unlikely that you will experience any performance issues by disabling TLS compression on your servers. Attacks against TLS compression are of low risk.

Mitigate information leakage stemming from HTTP compression

Two variations of the CRIME attack were disclosed in 2013. Rather than focus on TLS compression (which is what CRIME did), TIME and BREACH attacks focus on secrets in HTTP response bodies compressed using HTTP compression. Given that HTTP compression is very important to a great many companies, these problems are more difficult to address. Mitigation might require changes to application code.[9]

TIME and BREACH attacks require significant resources to carry out. But, if someone is motivated enough to use them, the impact is equivalent to CSRF.

Disable RC4

The RC4 cipher is insecure and should be disabled.[10] At the moment, the best attacks we know require millions of requests, a lot of bandwidth and time. Thus, the risk is still relatively low, but it's possible that the attacks will improve in the future. Before removing RC4, check if your existing users will be impacted; in other words, check if you have clients that support only RC4.

[7] SSL and TLS Authentication Gap Vulnerability Discovered (Qualys Security Labs Blog; November 2009)
[8] CRIME: Information Leakage Attack against SSL/TLS (Qualys Security Labs Blog; September 2012)
[9] Defending against the BREACH Attack (Qualys Security Labs; 7 August 2013)
[10] Internet-Draft: Prohibiting RC4 Cipher Suites (A. Popov, 1 October 2014)

Be aware of the BEAST attack

The 2011 BEAST attack[11] targets a 2004 vulnerability in TLS 1.0 and earlier protocol versions, previously thought to be impractical to exploit. The impact of a successful BEAST attack is similar to that of session hijacking. For a period of time, server-side mitigation of the BEAST attack was considered appropriate, even though the weakness is on the client side. Unfortunately, to mitigate server-side requires RC4, which can no longer be recommended. Because of that, and because the BEAST attack is by now largely mitigated client-side, we no longer recommend server-side mitigation.[12] In some situations, when there is a great number of old clients vulnerable to the BEAST attack, it might be more secure to use RC4 with TLS 1.0 and earlier protocol versions. A decision to do this should be made carefully and only after fully understanding the environment and its threat model.

Disable SSL v3

SSL v3 is vulnerable against the POODLE attack, which was disclosed in October 2014.[13] This attack is easy to carry out against HTTP clients, which can be tricked to execute JavaScript malware. They can also usually be tricked into downgrading from a better protocol (e.g., TLS 1.2) down to the vulnerable SSL v3. The best way to mitigate POODLE is to disable SSL v3, which most sites can do safely.

3. Performance

Security is our main focus in this guide, but we must also pay attention to performance; a secure service that does not satisfy performance criteria will no doubt be dropped. However, because TLS configuration does not usually have a significant overall performance impact, we are limiting the discussion in this section to the common configuration problems that result in serious performance degradation.

3.1. Do Not Use Too Much Security

The cryptographic handshake, which is used to establish secure connections, is an operation whose cost is highly influenced by private key size. Using a key that is too short is insecure, but using a key that is too long will result in "too much" security and slow operation. For most web sites, using RSA keys stronger than 2048 bits and ECDSA keys stronger than 256 bits is a waste of CPU power and might impair user experience. Similarly, there is little benefit to increasing the strength of the ephemeral key exchange beyond 2048 bits for DHE and 256 bits for ECDHE.

[11] Mitigating the BEAST attack on TLS (Qualys Security Labs Blog; October 2011)
[12] Is BEAST Still a Threat? (Qualys Security Labs; 10 September 2013)
[13] This POODLE bites: exploiting the SSL 3.0 fallback (Google Online Security Blog, 14 October 2014)

3.2. Ensure That Session Resumption Works Correctly

Session resumption is a performance-optimization technique that makes it possible to save the results of costly cryptographic operations and to reuse them for a period of time. A disabled or nonfunctional session resumption mechanism may introduce a significant performance penalty.

3.3. Use Persistent Connections (HTTP)

These days, most of the overhead of TLS comes not from the CPU-hungry cryptographic operations but from network latency. An TLS handshake is performed after the TCP handshake completes; it requires a further exchange of packets. To minimize the cost of latency, you enable HTTP persistence (keep-alives), allowing your users to submit many HTTP requests over a single TCP connection.

3.4. Enable Caching of Public Resources (HTTP)

When communicating over TLS, browsers assume that all traffic is sensitive. They will typically use the memory to cache certain resources, but once you close the browser, all the content may be lost. To get a performance boost and enable long-term caching of some resources, mark public resources (e.g., images) as public by attaching the `Cache-Control: public` response header to them.

3.5. Use OCSP Stapling

OCSP Stapling is a modification of the OCSP protocol that allows revocation information to be delivered as part of the TLS handshake, directly from the server to the browser. As a result, the browser does not need to contact OCSP servers for out-of-band validation and the connection time is significantly reduced.

4. Application Design (HTTP)

The HTTP protocol and the surrounding platform for web application delivery continued to evolve rapidly after SSL was born. As a result of that evolution, the platform now contains features that can be used to defeat encryption. In this section, we list those features, as well as ways to use them securely.

4.1. Encrypt 100% of Your Web Site

The fact that encryption is optional is probably one of the biggest security problems today. We see the following problems:

- No TLS on sites that need it
- Sites that have TLS but that do not enforce it
- Sites that mix TLS and non-TLS content, sometimes even within the same page
- Sites with programming errors that subvert TLS

Although many of these problems can be mitigated if you know exactly what you're doing, the only way to reliably protect web site communication is to enforce encryption throughout—without exception.

4.2. Avoid Mixed Content

Mixed-content pages are those that are transmitted over TLS but include resources (e.g., JavaScript files, images, CSS files) that are not transmitted over TLS. Such pages are not secure. An active man-in-the-middle (MITM) attacker can piggyback on a single unprotected JavaScript resource, for example, and hijack the entire user session. Even if you follow the advice from the previous section and encrypt your entire web site, you might still end up retrieving some resources unencrypted from third-party web sites.

4.3. Understand and Acknowledge Third-Party Trust

Web sites often use third-party services activated via JavaScript code downloaded from another server. A good example of such a service is Google Analytics, which is used on large parts of the Web. Such inclusion of third-party code creates an implicit trust connection that effectively gives the other party full control over your web site. The third party may not be malicious, but large providers of such services are increasingly seen as targets. The reasoning is simple: if a large provider is compromised, the attacker is automatically given access to all the sites that depend on the service.

If you follow the advice from Section 4.2, at least your third-party links will be encrypted and thus safe from MITM attacks. However, you should go a step further than that: learn what services your sites use, and either remove them, replace them with safer alternatives, or accept the risk of their continued use.

4.4. Secure Cookies

To be properly secure, a web site requires TLS, but also that all its cookies are marked as secure. Failure to secure the cookies makes it possible for an active MITM attacker to tease some information out through clever tricks, even on web sites that are 100% encrypted.

4.5. Deploy HTTP Strict Transport Security

HTTP Strict Transport Security (HSTS) is a safety net for TLS: it was designed to ensure that security remains intact even in the case of configuration problems and implementation errors. To activate HSTS protection, you set a single response header in your web sites. After that, browsers that support HSTS (at this time, Chrome, Firefox, Safari and Opera; IE soon) will enforce it.

The goal of HSTS is simple: after activation, it does not allow any insecure communication with the web site that uses it. It achieves this goal by automatically converting all plaintext links to secure ones. As a bonus, it also disables click-through certificate warnings. (Certificate warnings are an indicator of an active MITM attack. Studies have shown that most users click through these warnings, so it is in your best interest to never allow them.)

Adding support for HSTS is the single most important improvement you can make for the TLS security of your web sites. New sites should always be designed with HSTS in mind and the old sites converted to support it wherever possible.

4.6. Disable Caching of Sensitive Content

The goal of this recommendation is to ensure that sensitive content is communicated to only the intended parties and that it is treated as sensitive. Although proxies do not see encrypted traffic and cannot share content among users, the use of cloud-based application delivery platforms is increasing, which is why you need to be very careful when specifying what is public and what is not.

4.7. Ensure That There are No Other Vulnerabilities

This item is a reminder that TLS does not equal security. TLS is designed to address only one aspect of security – confidentiality and integrity of the communication between you and your users—but there are many other threats that you need to deal with. In most cases, that means ensuring that your web site does not have other weaknesses.

5. Validation

With many configuration parameters available for tweaking, it is difficult to know in advance what impact certain changes will have. Further, changes are sometimes made accidentally; software upgrades can introduce changes silently. For that reason, we advise that you use a comprehensive SSL/TLS assessment tool initially to verify your configuration to ensure that you start out secure, and then periodically to ensure that you stay secure. For public web sites, we recommend our free online assessment tool on

the SSL Labs web site. The *Handshake Simulation* feature, in particular, is very useful, because it shows exactly what security parameters would be used by a variety of commonly used TLS clients.

6. Advanced Topics

The following advanced topics are outside the scope of our guide. They require a deeper understanding of SSL/TLS and *Public Key Infrastructure* (PKI), and they are still being debated by experts.

Extended Validation certificates

EV certificates are high-assurance certificates issued only after thorough offline checks.[14] Their purpose is to provide a strong connection between an organization and its online identity. EV certificates are more difficult to forge, provide slightly better security, and are better treated when browsers present them to end users.

Public Key Pinning

Public Key Pinning is designed to give web site operators the means to restrict which CAs can issue certificates for their web sites. This feature has been deployed by Google for some time now (hardcoded into their browser, Chrome) and has proven to be very useful in preventing attacks and making the public aware of them. In 2014, Firefox also added support for hardcoded pinning. A standard called *Public Key Pinning Extension for HTTP* has been in development for a long time, but will be published soon. We expect that it will be supported by at least some major browsers in the near future.

ECDSA private keys

Today, most web sites rely on RSA private keys. This algorithm is thus the key to the security of the Web, which is why attacks against it continue to improve. Whereas before most sites used to use 1024-bit RSA keys, virtually everyone moved to 2048 bits. There are some concerns, however, that further RSA key length increases might lead to performance issues. Elliptic Curve cryptography uses different math and provides strong security assurances at smaller key lengths. RSA keys can be replaced with ECDSA. They are currently supported by only a small number of CAs, but we expect that most will offer them in the future. When migrating to ECDSA, one concern is that not all clients support this algorithm. If you're considering ECDSA, check whether the move will impact the ability of your users to connect to your servers. Some platforms support dual-key deployments, enabling you to use RSA and ECDSA keys in parallel, satisfying all clients.

[14] About EV SSL Certificates (CA/B Forum web site)

Changes

The first release of this guide was on 24 February 2012. This section tracks the document changes over time, starting with version 1.3.

Version 1.3 (17 September 2013)

The following changes were made in this version:

- Recommend replacing 1024-bit certificates straight away.
- Recommend against supporting SSL v3.
- Remove the recommendation to use RC4 to mitigate the BEAST attack server-side.
- Recommend that RC4 is disabled.
- Recommend that 3DES is disabled in the near future.
- Warn about the CRIME attack variations (TIME and BREACH).
- Recommend supporting Forward Secrecy.
- Add discussion of ECDSA certificates.

Version 1.4 (8 December 2014)

The following changes were made in this version:

- Discuss SHA1 deprecation and recommend migrating to the SHA2 family.
- Recommend that SSL v3 is disabled and mention the POODLE attack.
- Expand Section 3.1 to cover the strength of the DHE and ECDHE key exchanges.
- Recommend OCSP Stapling as a performance-improvement measure, promoting it to Section 3.5.

Acknowledgments

Special thanks to Marsh Ray, Nasko Oskov, Adrian F. Dimcev, and Ryan Hurst for their valuable feedback and help in crafting the initial version of this document. Also thanks to many others who generously share their knowledge of security and cryptography with the world. The guidelines presented here draw on the work of the entire security community.

About SSL Labs

SSL Labs is Qualys's research effort to understand SSL/TLS and PKI as well as to provide tools and documentation to assist with assessment and configuration. Since 2009, when SSL Labs was launched, hundreds of thousands of assessments have been performed using the free online assessment tool. Other projects run by SSL Labs include periodic Internet-wide surveys of TLS configuration and SSL Pulse, a monthly scan of about 150,000 most popular TLS-enabled web sites in the world.

About Qualys

Qualys, Inc. (NASDAQ: QLYS), is a pioneer and leading provider of cloud security and compliance solutions with over 6,700 customers in more than 100 countries, including a majority of each of the Forbes Global 100 and Fortune 100. The QualysGuard Cloud Platform and integrated suite of solutions help organizations simplify security operations and lower the cost of compliance by delivering critical security intelligence on demand and automating the full spectrum of auditing, compliance, and protection for IT systems and web applications. Founded in 1999, Qualys has established strategic partnerships with leading managed service providers and consulting organizations, including BT, Dell SecureWorks, Fujitsu, IBM, NTT, Symantec, Verizon, and Wipro. The company is also a founding member of the Council on CyberSecurity and the Cloud Security Alliance (CSA).