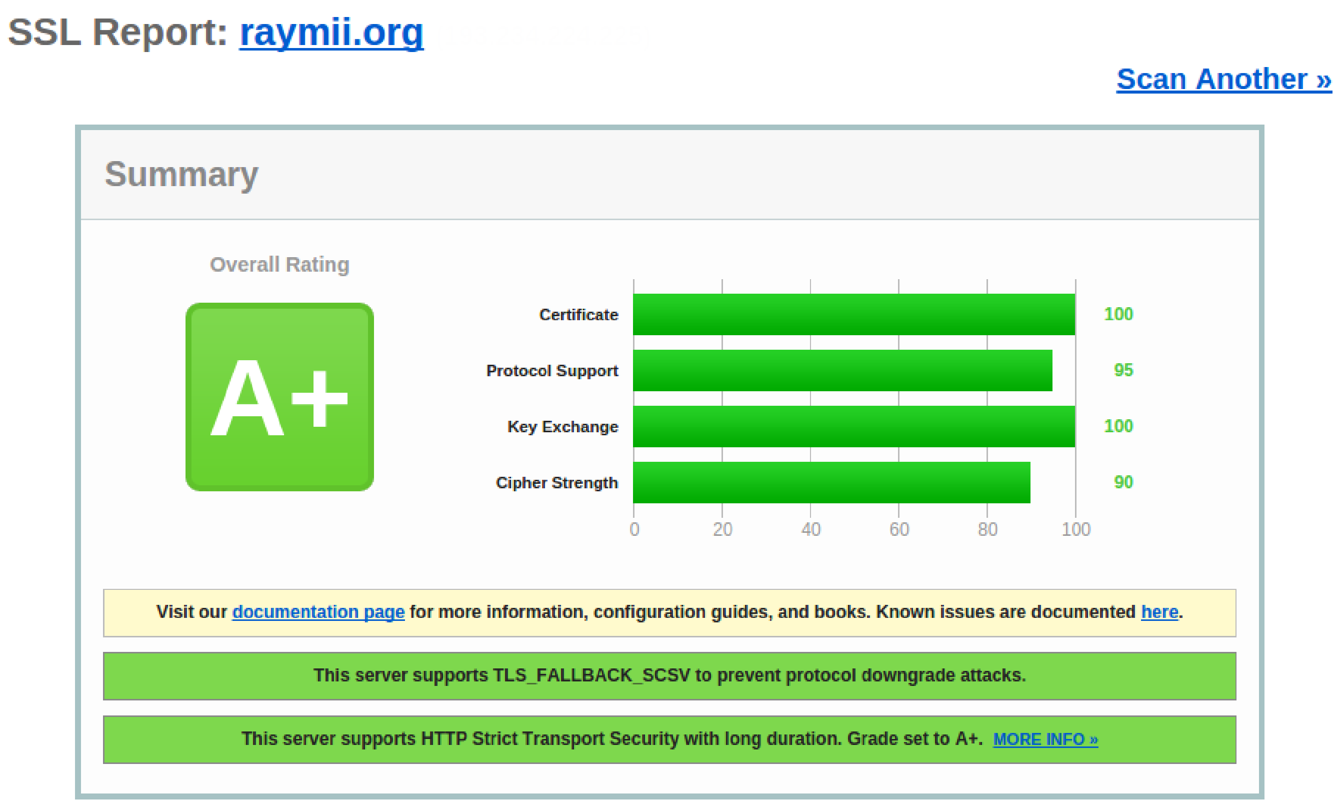
Strong SSL Security on Apache2

14-06-2015 | Remy van Elst

Table of Contents



SSL Report: raymii.org

Scan Another »

Summary

Visit our documentation page for more information, configuration guides, and books. Known issues are documented here.

This server supports TLS_FALLBACK_SCSV to prevent protocol downgrade attacks.

This server supports HTTP Strict Transport Security with long duration. Grade set to A+. MORE INFO »

(https://www.ssllabs.com/ssltest/analyze.html?d=raymii.org)

This tutorial shows you how to set up strong SSL security on the Apache2 webserver. We do this by updating OpenSSL to the latest version to mitigate attacks like Heartbleed, disabling SSL Compression and EXPORT ciphers to mitigate attacks like FREAK, CRIME and LogJAM, disabling SSLv3 and below because of vulnerabilities in the protocol and we will set up a strong ciphersuite that enables Forward Secrecy when possible. We also enable HSTS and HPKP. This way we have a strong and future proof ssl configuration and we get an A+ on the Qually Labs SSL Test.

I've created a website with Copy-pastable strong cipherssuites for NGINX, Apache, Lighttpd and other software: https://cipherli.st(https://cipherli.st/). Handy if you don't want to read this entire tutorial. This tutorial and https://cipherli.st(https://cipherli.st/) are updated continuously as new vulnerabilities are discovered.

I've written an Open Source SSL server test(https://raymii.org/s/software/OpenSSL_Decoder.html). You can use it to test your configuration, as an addition to the other SSL tests our there. It is open source so you can host it yourself internally to test local resources. It is fast, shows you all the information so you can make your own informed decision (no ratings), and the results are saved so you can compare different settings. You can test your site via https://ssldecoder.org(https://ssldecoder.org).

I've also written a handy tool which notifies you when your certificates are about to expire. It is open source so you can host it yourself internally and there is a hosted version available at https://certificatemonitor.org(https://certificatemonitor.org).

If you like this article, consider sponsoring me by trying out a Digital Ocean VPS. With this link you'll get a $5 VPS for 2 months free (as in, you get $10 credit). (referral link)(https://www.digitalocean.com/?refcode=7435ae6b8212)

This tutorial works with the strict requirements of the SSL Labs test(http://blog.ivanristic.com/2014/01/ssl-labs-stricter-security-requirements-for-2014.html)

This tutorial is also available for NGINX(https://raymii.org/s/tutorials/Strong_SSL_Security_On_nginx.html)
This tutorial is also available for Lighttpd(https://raymii.org/s/tutorials/Pass_the_SSL_Labs_Test_on_Lighttpd_%28Mitigate_the_CRIME_and_BEAST_attack_-_Disable_SSLv2_-_Enable_PFS%29.html)
This tutorial is also available for FreeBSD, NetBSD and OpenBSD over at the BSD Now podcast(http://www.bsdnow.tv/episodes/2014_08_20-engineering_nginx)

You can find more info on the topics by following the links below:

- BEAST Attack(https://en.wikipedia.org/wiki/Transport_Layer_Security#BEAST_attack)
- CRIME Attack(https://en.wikipedia.org/wiki/CRIME_%28security_exploit%29)
- Heartbleed(http://heartbleed.com/)
- FREAK Attack(http://blog.cryptographyengineering.com/2015/03/attack-of-week-freak-or-factoring-nsa.html)

- Perfect Forward Secrecy(https://en.wikipedia.org/wiki/Perfect_forward_secrecy)
- Dealing with RC4 and BEAST(https://en.wikipedia.org/wiki/Transport_Layer_Security#Dealing_with_RC4_and_BEAST)

Make sure you back up the files before editing them!

The BEAST attack and RC4

In short, by tampering with an encryption algorithm's CBC - cipher block chaining - mode's, portions of the encrypted traffic can be secretly decrypted. More info on the above link.

Recent browser versions have enabled client side mitigation for the beast attack. The recommendation was to disable all TLS 1.0 ciphers and only offer RC4. However, [RC4 has a growing list of attacks against it],(http://www.isg.rhul.ac.uk/tls/) many of which have crossed the line from theoretical to practical. Moreover, there is reason to believe that the NSA has broken RC4, their so-called "big breakthrough."

Disabling RC4 has several ramifications. One, users with shitty browsers such as Internet Explorer on Windows XP will use 3DES in lieu. Triple-DES is more secure than RC4, but it is significantly more expensive. Your server will pay the cost for these users. Two, RC4 mitigates BEAST. Thus, disabling RC4 makes TLS 1.0 users susceptible to that attack, by moving them to AES-CBC (the usual server-side BEAST "fix" is to prioritize RC4 above all else). I am confident that the flaws in RC4 significantly outweigh the risks from BEAST. Indeed, with client-side mitigation (which Chrome and Firefox both provide), BEAST is a nonissue. But the risk from RC4 only grows: More cryptanalysis will surface over time.

Factoring RSA-EXPORT Keys (FREAK)

FREAK is a man-in-the-middle (MITM) vulnerability discovered by a group of cryptographers at INRIA, Microsoft Research and IMDEA(https://www.smacktls.com/). FREAK stands for "Factoring RSA-EXPORT Keys."

The vulnerability dates back to the 1990s, when the US government banned selling crypto software overseas, unless it used export cipher suites which involved encryption keys no longer than 512-bits.

It turns out that some modern TLS clients - including Apple's SecureTransport and OpenSSL - have a bug in them. This bug causes them to accept RSA export-grade keys even when the client didn't ask for export-grade RSA. The impact of this bug can be quite nasty: it admits a 'man in the middle' attack whereby an active attacker can force down the quality of a connection, provided that the client is vulnerable and the server supports export RSA.

There are two parts of the attack as the server must also accept "export grade RSA."

The MITM attack works as follows:

- In the client's Hello message, it asks for a standard 'RSA' ciphersuite.
- The MITM attacker changes this message to ask for 'export RSA'.
- The server responds with a 512-bit export RSA key, signed with its long-term key.
- The client accepts this weak key due to the OpenSSL/SecureTransport bug.
- The attacker factors the RSA modulus to recover the corresponding RSA decryption key.
- When the client encrypts the 'pre-master secret' to the server, the attacker can now decrypt it to recover the TLS 'master secret'.
- From here on out, the attacker sees plaintext and can inject anything it wants.

The ciphersuite offered here on this page does not enable EXPORT grade ciphers. Make sure your OpenSSL is updated to the latest available version and urge your clients to also use upgraded software.

Logjam (DH EXPORT)

Researchers(https://weakdh.org/) from several universities and institutions conducted a study that found an issue in the TLS protocol. In a report the researchers report two attack methods.

Diffie-Hellman key exchange allows that depend on TLS to agree on a shared key and negotiate a secure session over a plain text connection.

With the first attack, a man-in-the-middle can downgrade a vulnerable TLS connection to 512-bit export-grade cryptography which would allow the attacker to read and change the data. The second threat is that many servers and use the same prime numbers for Diffie-Hellman key exchange instead of generating their own unique DH parameters.

The team estimates that an academic team can break 768-bit primes and that a nation-state could break a 1024-bit prime. By breaking one 1024-bit prime, one could eavesdrop on 18 percent of the top one million HTTPS domains. Breaking a second prime would open up 66 percent of VPNs and 26 percent of SSH servers.

Later on in this guide we generate our own unique DH parameters and we use a ciphersuite that does not enable EXPORT grade ciphers. Make sure your OpenSSL is updated to the latest available version and urge your clients to also use upgraded software. Updated browsers refuse DH parameters lower than 768/1024 bit as a fix to this.

Cloudflare has a detailed guide(https://blog.cloudflare.com/logjam-the-latest-tls-vulnerability-explained/) on logjam.

Heartbleed

Heartbleed is a security bug disclosed in April 2014 in the OpenSSL cryptography library, which is a widely used implementation of the Transport Layer Security (TLS) protocol. Heartbleed may be exploited regardless of whether the party using a vulnerable OpenSSL instance for TLS is a server or a client. It results from improper input validation (due to a missing bounds check) in the implementation of the DTLS heartbeat extension (RFC6520), thus the bug's name derives from "heartbeat". The vulnerability is classified as a buffer over-read, a situation where more data can be read than should be allowed.

What versions of the OpenSSL are affected by Heartbleed?

Status of different versions:

- OpenSSL 1.0.1 through 1.0.1f (inclusive) are vulnerable
- OpenSSL 1.0.1g is NOT vulnerable
- OpenSSL 1.0.0 branch is NOT vulnerable
- OpenSSL 0.9.8 branch is NOT vulnerable

The bug was introduced to OpenSSL in December 2011 and has been out in the wild since OpenSSL release 1.0.1 on 14th of March 2012. OpenSSL 1.0.1g released on 7th of April 2014 fixes the bug.

By updating OpenSSL you are not vulnerable to this bug.

SSL Compression (CRIME attack)

The CRIME attack uses SSL Compression to do its magic, so we need to disable that. On Apache 2.2.24+ we can add the following line to the SSL config file we also edited above:

```
SSLCompression off
```

If you are using al earlier version of Apache and your distro has not backported this option then you need to recompile OpenSSL without ZLIB support. This will disable the use of OpenSSL using the DEFLATE compression method. If you do this then you can still use regular HTML DEFLATE compression.

SSLv2 and SSLv3

SSL v2 is insecure, so we need to disable it. We also disable SSLv3, as TLS 1.0 suffers a downgrade attack, allowing an attacker to force a connection to use SSLv3 and therefore disable forward secrecy.

SSLv3 allows exploiting of the POODLE(https://raymii.org/s/articles/Check_servers_for_the_Poodle_bug.html) bug. This is one more major reason to disable this!

Again edit the config file:

```
SSLProtocol All -SSLv2 -SSLv3
```

All is a shortcut for `+SSLv2 +SSLv3 +TLSv1` or - when using OpenSSL 1.0.1 and later - `+SSLv2 +SSLv3 +TLSv1 +TLSv1.1 +TLSv1.2`, respectively. The above line enables everything except SSLv2 and SSLv3. More info on the apache website(http://httpd.apache.org/docs/2.2/mod/mod_ssl.html#sslprotocol)

Poodle and TLS-FALLBACK-SCSV

SSLv3 allows exploiting of the POODLE(https://raymii.org/s/articles/Check_servers_for_the_Poodle_bug.html) bug. This is one more major reason to disable this.

Google have proposed an extension to SSL/TLS named TLS*FALLBACK*SCSV(https://tools.ietf.org/html/draft-ietf-tls-downgrade-scsv-00) that seeks to prevent forced SSL downgrades. This is automatically enabled if you upgrade OpenSSL to the following versions:

- OpenSSL 1.0.1 has TLS*FALLBACK*SCSV in 1.0.1j and higher.
- OpenSSL 1.0.0 has TLS*FALLBACK*SCSV in 1.0.0o and higher.
- OpenSSL 0.9.8 has TLS*FALLBACK*SCSV in 0.9.8zc and higher.

The Cipher Suite

(Perfect) Forward Secrecy ensures the integrity of a session key in the event that a long-term key is compromised. PFS accomplishes this by enforcing the derivation of a new key for each and every session.

This means that when the private key gets compromised it cannot be used to decrypt recorded SSL traffic.

The cipher suites that provide Perfect Forward Secrecy are those that use an ephemeral form of the Diffie-Hellman key exchange. Their disadvantage is their overhead, which can be improved by using the elliptic curve variants.

The following two ciphersuites are recommended by me, and the latter by the Mozilla Foundation(https://wiki.mozilla.org/Security/Server_Side_TLS).

The recommended cipher suite:

```
SSLCipherSuite EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH
```

The recommended cipher suite for backwards compatibility (IE6/WinXP):

```
SSLCipherSuite EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:ECDHE-RSA-AES128-SHA:DHE-RSA-AES128-GCM-SHA256:AES256+EDH:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES256-GCM-SHA384:AES128-GCM-SHA256:AES256-SHA256:AES128-SHA256:AES256-SHA:AES128-SHA:DES-CBC3-SHA:HIGH:!aNULL:!eNULL:!EXPORT:!DES:!MD5:!PSK:!RC4
```

If your version of OpenSSL is old, unavailable ciphers will be discarded automatically. Always use the full ciphersuite above and let OpenSSL pick the ones it supports.

The ordering of a ciphersuite is very important because it decides which algorithms are going to be selected in priority. The recommendation above prioritizes algorithms that provide perfect forward secrecy.

Older versions of OpenSSL may not return the full list of algorithms. AES-GCM and some ECDHE are fairly recent, and not present on most versions of OpenSSL shipped with Ubuntu or RHEL.

Prioritization logic

- ECDHE+AESGCM ciphers are selected first. These are TLS 1.2 ciphers. No known attack currently target these ciphers.
- PFS ciphersuites are preferred, with ECDHE first, then DHE.
- AES 128 is preferred to AES 256. There has been discussions(http://www.mail-archive.com/dev-tech-crypto@lists.mozilla.org/msg11247.html) on whether AES256 extra security was worth the cost, and the result is far from obvious. At the moment, AES128 is preferred, because it provides good security, is really fast, and seems to be more resistant to timing attacks.
- In the backward compatible ciphersuite, AES is preferred to 3DES. BEAST attacks on AES are mitigated in TLS 1.1 and above, and difficult to achieve in TLS 1.0. In the non-backward compatible ciphersuite, 3DES is not present.
- RC4 is removed entirely. 3DES is used for backward compatibility. See discussion in #RC4_weaknesses(https://wiki.mozilla.org/Security/Server_Side_TLS#RC4_weaknesses)

Mandatory discards

- aNULL contains non-authenticated Diffie-Hellman key exchanges, that are subject to Man-In-The-Middle (MITM) attacks
- eNULL contains null-encryption ciphers (cleartext)
- EXPORT are legacy weak ciphers that were marked as exportable by US law
- RC4 contains ciphers that use the deprecated ARCFOUR algorithm
- DES contains ciphers that use the deprecated Data Encryption Standard
- SSLv2 contains all ciphers that were defined in the old version of the SSL standard, now deprecated
- MD5 contains all the ciphers that use the deprecated message digest 5 as the hashing algorithm

With Apache 2.2.x you have only DHE suites to work with, but they are not enough. Internet Explorer (in all versions) does not support the required DHE suites to achieve Forward Secrecy. (Unless youre using DSA keys, but no one does; that's a long story.) Apache does not support configurable DH parameters in any version, but there are patches you could use if you can install from source.

Even if openssl can provide ECDHE the apache 2.2 in debian stable does not support this mechanism. You need apache 2.4 to fully support forward secrecy.

A workaround could be the usage of nginx as a reverse proxy because it fully supports ECDHE.

Make sure you also add this line:

SSLHonorCipherOrder on

When choosing a cipher during an SSLv3 or TLSv1 handshake, normally the client's preference is used. If this directive is enabled, the server's preference will be used instead.

Forward Secrecy & Diffie Hellman Ephemeral Parameters

The concept of forward secrecy is simple: client and server negotiate a key that never hits the wire, and is destroyed at the end of the session. The RSA private from the server is used to sign a Diffie-Hellman key exchange between the client and the server. The pre-master key obtained from the Diffie-Hellman handshake is then used for encryption. Since the pre-master key is specific to a connection between a client and a server, and used only for a limited amount of time, it is called Ephemeral.

With Forward Secrecy, if an attacker gets a hold of the server's private key, it will not be able to decrypt past communications. The private key is only used to sign the DH handshake, which does not reveal the pre-master key. Diffie-Hellman ensures that the pre-master keys never leave the client and the server, and cannot be intercepted by a MITM.

Apache prior to version 2.4.7 and all versions of Nginx as of 1.4.4 rely on OpenSSL for input parameters to Diffie-Hellman (DH). Unfortunately, this means that Ephemeral Diffie-Hellman (DHE) will use OpenSSL's defaults, which include a 1024-bit key for the key-exchange. Since we're using a 2048-bit certificate, DHE clients will use a weaker key-exchange than non-ephemeral DH clients.

For Apache, there is no fix except to upgrade to 2.4.7 or later. With that version, Apache automatically selects a stronger key.

If you have Apache 2.4.8 or later and OpenSSL 1.0.2 or later, you can generate and specify your DH params file:

```
#Generate the parameters
cd /etc/ssl/certs
openssl dhparam -out dhparam.pem 4096

# Add the following to your Apache config.
SSLOpenSSLConfCmd DHParameters "/etc/ssl/certs/dhparam.pem"
```

If you are using Apache with LibreSSL, or Apache 2.4.7 and OpenSSL 0.9.8a or later, you can append the DHparams you generated earlier to the end of your certificate file. The documentation for that is here(https://httpd.apache.org/docs/2.4/mod/mod_ssl.html#sslcertificatefile) and below:

Custom DH parameters and an EC curve name for ephemeral keys, can also be added to end of the first file configured using `SSLCertificateFile`. This is supported in version 2.4.7 or later. Such parameters can be generated using the commands `openssl dhparam` and `openssl ecparam`. The parameters can be added as-is to the end of the first certificate file. Only the first file can be used for custom parameters, as they are applied independently of the authentication algorithm type.

Around May, Debian backported ECDH ciphers to work with apache 2.2, and it's possible to get PFS: http://metadata.ftp-master.debian.org/changelogs//main/a/apache2/apache2*2.2.22-13+deb7u3*changelog(http://metadata.ftp-master.debian.org/changelogs//main/a/apache2/apache2_2.2.22-13+deb7u3_changelog)

```
> apache2 (2.2.22-13+deb7u2) wheezy; urgency=medium

  * Backport support for SSL ECC keys and ECDH ciphers.
```

HTTP Strict Transport Security

When possible, you should enable HTTP Strict Transport Security (HSTS)(https://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security), which instructs browsers to communicate with your site only over HTTPS.

View my article on HTST to see how to configure it. (https://raymii.org/s/tutorials/HTTP_Strict_Transport_Security_for_Apache_NGINX_and_Lighttpd.html)

HTTP Public Key Pinning Extension

You should also enable the HTTP Public Key Pinning Extension(https://wiki.mozilla.org/SecurityEngineering/Public_Key_Pinning).

Public Key Pinning means that a certificate chain must include a whitelisted public key. It ensures only whitelisted Certificate Authorities (CA) can sign certificates for `*.example.com`, and not any CA in your browser store.

I've written an article about it that has background theory and configuration examples for Apache, Lighttpd and NGINX: https://raymii.org/s/articles/HTTP*Public*Key*Pinning*Extension_HPKP.html(https://raymii.org/s/articles/HTTP_Public_Key_Pinning_Extension_HPKP.html)

OCSP Stapling

When connecting to a server, clients should verify the validity of the server certificate using either a Certificate Revocation List (CRL), or an Online Certificate Status Protocol (OCSP) record. The problem with CRL is that the lists have grown huge and takes forever to download.

OCSP is much more lightweight, as only one record is retrieved at a time. But the side effect is that OCSP requests must be made to a 3rd party OCSP responder when connecting to a server, which adds latency and potential failures. In fact, the OCSP responders operated by CAs are often so unreliable that browser will fail silently if no response is received in a timely manner. This reduces security, by allowing an attacker to DoS an OCSP responder to disable the validation.

The solution is 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, by announcing support for the status_request TLS extension in its CLIENT HELLO.

Most servers will cache OCSP response 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.

View my tutorial on enabling OCSP stapling on Apache(https://raymii.org/s/tutorials/OCSP_Stapling_on_Apache2.html)

Conclusion

If you have applied the above config lines you need to restart apache:

```
# Check the config first:
apache2ctl -t
# Then restart:
/etc/init.d/apache2 restart

# If you are on RHEL/CentOS:
apachectl -t
/etc/init.d/httpd restart
```

Now use the SSL Labs test(https://www.ssllabs.com/ssltest/) to see if you get a nice A+. And, of course, have a safe, strong and future proof SSL configuration!

Tags: apache, (../tags/apache.html)pfs, (../tags/pfs.html)security, (../tags/security.html)ssl, (../tags/ssl.html)ssl-labs, (../tags/ssl-labs.html)tls, (../tags/tls.html)
