TLS Best Practices

Testing TLS Configuration

TLS provides confidentiality, integrity, and authenticity if used correctly. When we conduct penetration tests on a web server, it is important to assess the TLS configuration. If TLS is misconfigured, there is not only a risk to the server but also to all clients establishing TLS sessions with that server. Therefore, a web server should always be configured according to the latest TLS best practices to provide the maximum amount of security for all clients.

Key Management Best Practices

Before jumping into some TLS best practices, let's first discuss some general key management best practices that should be followed whenever cryptographic algorithms are used. We will only discuss this briefly here, for more details have a look at the NIST best practices.

The first thing to keep in mind is that each key should only be used for a single use case. This can either be encryption, signing, authentication, or something else entirely. However, using a single key for multiple purposes is a bad idea. This limits the impact of a potential key compromise, as an attacker is limited to the use case the key is dedicated for.

Additionally, it is important to define `cryptoperiods` after which keys are expired and no longer used. This limits the amount of exposure of a single key and ensures that there is a limited timeframe for computationally intensive attacks such as cryptoanalysis or brute-force attacks. After a key is compromised, it should immediately be treated as deprecated and replaced.

There are of course a lot more things to keep in mind when it comes to using cryptography correctly. However, these are among the most important ones. Here is a short list of additional things to keep in mind:

- Ensure the existence of full documentation of the key management processes
- Ensure the key generation process generates strong keys
- Ensure keys are not stored unencrypted
- Ensure that expired, weakened, or compromised keys are replaced and not used anymore
- Ensure that cryptographic keys are stored in a different location than the data it is used on
- Ensure that no hardcoded cryptographic keys are used
- Ensure that no custom cryptographic algorithms are used but only state-of-the-art and known good algorithms that are considered secure

- Ensure that encryption at rest and encryption in transit is used whenever possible

TLS Versions

Generally, only TLS 1.2 and TLS 1.3 should be offered. TLS 1.0 and 1.1 are considered deprecated, though it might be necessary to support them for legacy reasons. In any way, SSL 2.0 and SSL 3.0 are completely broken and should not be offered under any circumstances.

We can configure the supported TLS versions in Apache in the `ssl.conf` file:

```
SSLProtocol +TLSv1.2 +TLSv1.3
```

The same configuration in Nginx's config file looks like this:

```
ssl_protocols TLSv1.2 TLSv1.3;
```

Cipher Suites

After the TLS version, the cipher suite is the most important configuration for the session as it determines all the cryptographic algorithms used. Therefore, servers ideally should only offer the most secure cipher suites. However, this is infeasible in most scenarios since not all clients support strong cipher suites, so weaker cipher suites have to be supported to allow legacy clients to use the service. Otherwise, these clients would be locked out from using the service as they are unable to establish a TLS connection with the server.

However, some rules of thumb should be followed when it comes to cipher suites:

- do not offer any `NULL` cipher suites that do not offer encryption
- do not offer any `EXPORT` cipher suites that only offer weak encryption
- preferably use cipher suites that offer PFS. These are all TLS 1.3 cipher suites and the `ECDHE` and `DHE` cipher suites in TLS 1.2
- preferably use cipher suites in `GCM` mode over cipher suites in `CBC` mode

We can limit the offered cipher suites in Apache to cipher suites with at least 128-bit key length in the `ssl.conf` file:

```
SSLCipherSuite HIGH
```

Alternatively, we can explicitly specify a list of cipher suites in preferred order:

```
SSLCipherSuite ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-
SHA384:ECDHE-ECDSA-CHACHA20-POLY1305
```

The same configuration in Nginx's config file looks like this:

```
ssl_ciphers ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-
ECDSA-CHACHA20-POLY1305;
```

Best Practices & Tools

To assess the TLS configuration of a webserver and determine whether the server is vulnerable to any common TLS vulnerabilities, the tool testssl.sh can be used. It can be downloaded from GitHub:

mayala@htb[/htb] `$ git clone --depth 1 https://github.com/drwetter/testssl.sh.git $ cd testssl.sh/ $ bash testssl.sh <SNIP>`

We can run all default tests against a server by just specifying the URL:

mayala@htb[/htb] `$ bash testssl.sh https://hackthebox.com <SNIP> Testing protocols via sockets except NPN+ALPN SSLv2 not offered (OK) SSLv3 not offered (OK) TLS 1 offered (deprecated) TLS 1.1 offered (deprecated) TLS 1.2 offered (OK) TLS 1.3 offered (OK): final NPN/SPDY h2, http/1.1 (advertised) ALPN/HTTP2 h2, http/1.1 (offered) Testing cipher categories NULL ciphers (no encryption) not offered (OK) Anonymous NULL Ciphers (no authentication) not offered (OK) Export ciphers (w/o ADH+NULL) not offered (OK) LOW: 64 Bit + DES, RC[2,4], MD5 (w/o export) not offered (OK) Triple DES Ciphers / IDEA offered Obsoleted CBC ciphers (AES, ARIA etc.) offered Strong encryption (AEAD ciphers) with no FS offered (OK) Forward Secrecy strong encryption (AEAD ciphers) offered (OK) <SNIP> Server Certificate #1 Signature Algorithm SHA256 with RSA Server key size RSA 2048 bits (exponent is 65537) Server key usage Digital Signature, Key Encipherment Server extended key usage TLS Web Server Authentication, TLS Web Client Authentication Serial 041068326595F10235DB8C1A08635245212F (OK: length 18) Fingerprints SHA1 0405C0130A0579A80F0FAB7E23443A4A0B16129C SHA256 BCE478826AAC55381D60B520FEB1A20B086554B1A50D88CB8DD9F47030737ABE Common Name (CN) *.enterprise.hackthebox.com (request w/o SNI didn't succeed) subjectAltName (SAN) *.enterprise.hackthebox.com hackthebox.com Trust (hostname) Ok via SAN (SNI mandatory) Chain of trust Ok EV cert (experimental) no Certificate Validity (UTC) 86 >= 30 days (2022-12-26 14:27 --> 2023-03-26 14:27) <SNIP> Testing vulnerabilities Heartbleed (CVE-2014-0160) not vulnerable (OK), no heartbeat`

```
extension CCS (CVE-2014-0224) not vulnerable (OK) Ticketbleed (CVE-2016-9244), experiment. not vulnerable (OK), no session tickets ROBOT not vulnerable (OK) Secure Renegotiation (RFC 5746) OpenSSL handshake didn't succeed Secure Client-Initiated Renegotiation not vulnerable (OK) CRIME, TLS (CVE-2012-4929) not vulnerable (OK) BREACH (CVE-2013-3587) no gzip/deflate/compress/br HTTP compression (OK) - only supplied "/" tested POODLE, SSL (CVE-2014-3566) not vulnerable (OK), no SSLv3 support <SNIP> Rating (experimental) Rating specs (not complete) SSL Labs's 'SSL Server Rating Guide' (version 2009q from 2020-01-30) Specification documentation https://github.com/ssllabs/research/wiki/SSL-Server-Rating-Guide Protocol Support (weighted) 95 (28) Key Exchange (weighted) 90 (27) Cipher Strength (weighted) 90 (36) Final Score 91 Overall Grade B Grade cap reasons Grade capped to B. TLS 1.1 offered Grade capped to B. TLS 1.0 offered Grade capped to A. HSTS is not offered
```

From the output, we can see that the tool automatically tests the entire TLS configuration including cipher suites, offered TLS versions, the certificate, and the existence of common vulnerabilities. Finally, the tool gives a grade and reasons for the grading. We can see that the HackTheBox web server still offers TLS 1.0 and TLS 1.1 and does not have the HSTS header configured to prevent SSL Stripping attacks. In a penetration test, we could add these findings to our report as low-risk findings. A lower grade might result in a higher severity rating for misconfigured TLS during a real-life engagement.