SSL - Configurations

Allows insecure protocol SSLv2:

This protocol is vulnerable to the DROWN attack. SSLv2 has been deprecated and has many known security vulnerabilities.
Update your company server configurations to disable SSLv2.

Allows insecure protocol SSLv3:

This protocol is vulnerable to the POODLE attack.
Update your company servers' software to disable SSLv3.

Diffie-Hellman prime is less than 2048 bits

Primes shorter than 2048 bits are estimated to be breakable by adversaries with nation-state-level resources.
Use OpenSSL or a TLS implementation of your choice to generate a new Diffie-Hellman group and key pair for your server greater than or equal to 2048 bits. Read more at weakdh.org

Short Diffie-Hellman prime is very commonly used:

A common Diffie-Hellman prime indicates poor server-side TLS configuration; servers with common primes are more susceptible to compromise, as demonstrated by the Logjam attack.
Use OpenSSL or a TLS implementation of your choice to generate a new Diffie-Hellman group for your server. Read more at weakdh.org, create one at 2048 bits or greater.

Allows insecure cipher: Export Ciphers

This server accepts the RSA_EXPORT cipher suite, making it susceptible to the FREAK attack.
Update your company web server software to disable export cipher suites.

SSL – Certificates

Expired certificate:

While the traffic to the host is still encrypted, it may be vulnerable to new attacks.
Obtain and install an updated certificate as soon as possible, obtain a certificate from an industry certificate authority.

Insecure signature algorithm SHA1:

SHA-1 is vulnerable to partial-message collision attacks; Internet Explorer, Chrome, and Firefox will not accept SHA-1-signed certificates starting in 2017.
TLS certificate uses a signing algorithm which is no longer supported by the security industry. Renew TLS certificate with certificate vendor and specify a stronger signature algorithm, such as SHA-256.

Self-signed certificate:

This certificate was signed by the server that is hosting the domain rather than a trusted certificate authority. In these cases, the Subject and Issuer fields will be the same. Self-signed certificates can be more easily compromised and are flagged by most browsers. For more information about the dangers of self-signed certificates, see Thawte's "The Hidden Costs of Self-Signed SSL Certificates."
Use a TLS certificate provided by an industry certificate authority provider. A list of common providers can be found here (https://en.wikipedia.org/wiki/Certificate_authority#Providers).

Missing intermediate certificates or root anchor:

The certificate chain received from the server is incomplete and not signed by a known trust anchor.
Servers may be missing necessary intermediate certificates initially provided by TLS vendor (they may not have been installed), which certificate vendor can send to you again.

RSA public key is less than 2048 bits:

RSA keys short than 2048 bits may be insecure. According to NIST (PDF), keys between 1024 and 2048 bits became acceptable for legacy use only after 2010.
TLS certificate will need to be re-issued or regenerated by certificate provider with an RSA public key strength greater than 2048 bits.

Insecure signature algorithm MD5:

MD5 is not collision-resistant and its support has been discontinued.
TLS certificate uses a signing algorithm which is no longer supported by the security industry. Renew TLS certificate with certificate vendor and specify a stronger signature algorithm, such as SHA-256.

Application Security – HTTP Response Headers

View HTTP Request and Response Header:

http://web-sniffer.net/

1. Cache-Control

A "safe" Cache-Control header would be:

Cache-Control: private, no-cache, no-store, max-age=0

http://stackoverflow.com/questions/49547/making-sure-a-web-page-is-not-cached-across-all-browsers

How to set it:

Using PHP:

```
header("Cache-Control: no-cache, no-store, must-revalidate"); // HTTP 1.1.
header("Pragma: no-cache"); // HTTP 1.0.
header("Expires: 0"); // Proxies.
```

Using ASP.NET-MVC

```
Response.Cache.SetCacheability(HttpCacheability.NoCache);  // HTTP 1.1.
Response.Cache.AppendCacheExtension("no-store, must-revalidate");
Response.AppendHeader("Pragma", "no-cache"); // HTTP 1.0.
Response.AppendHeader("Expires", "0"); // Proxies.
```

Using Apache `.htaccess` file:

```
<IfModule mod_headers.c>
    Header set Cache-Control "no-cache, no-store, must-revalidate"
    Header set Pragma "no-cache"
    Header set Expires 0
</IfModule>
```

Using HTML4:

```
<meta http-equiv="Cache-Control" content="no-cache, no-store, must-revalidate" />
<meta http-equiv="Pragma" content="no-cache" />
<meta http-equiv="Expires" content="0" />
```

2. Content-Security-Policy

Details and Examples about Content-Security-Policy:

https://content-security-policy.com/

It is known that having both Content-Security-Policy and X-Content-Security-Policy or X-Webkit-CSP causes unexpected behaviours on certain versions of browsers. Please avoid using deprecated X-* headers.

Content-Security-Policy Examples:

Allow everything but only from the same origin

```
default-src 'self';
```

Only Allow Scripts from the same origin

```
script-src 'self';
```

Allow Google Analytics, Google AJAX CDN and Same Origin

```
script-src 'self' www.google-analytics.com ajax.googleapis.com;
```

Starter Policy

This policy allows images, scripts, AJAX, and CSS from the same origin, and does not allow any other resources to load (eg object, frame, media, etc). It is a good starting point for many sites.

```
default-src 'none'; script-src 'self'; connect-src 'self'; img-src 'self'; style-src 'self';
```

3. Strict-Transport-Security

Background and Examples about Strict-Transport-Security:

https://https.cio.gov/hsts/

HTTP Strict Transport Security (HSTS) is a standard to protect visitors by ensuring that their browsers *always* connect to a website over HTTPS. HSTS exists to remove the need for the common, insecure practice of redirecting users from `http://` to `https://` URLs.

In its simplest form, the policy tells a browser to enable HSTS for that exact domain or subdomain, and to remember it for a given number of seconds:

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

In its **strongest and recommended form**, the HSTS policy **includes all subdomains**, and indicates a willingness to be "preloaded" into browsers:

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

When using this form, bear in mind:

The policy should be deployed at `https://domain.gov`, *not* `https://www.domain.gov`.

4. X-Content-Type-Options

See also about X-Content-Type-Options:

https://kb.sucuri.net/warnings/hardening/headers-x-content-type

It is recommended to add the following header:

```
X-Content-Type-Options: nosniff
```

Enabling in Apache:

In the `.htaccess` file add the following line:

```
<IfModule mod_headers.c>
  Header set X-Content-Type-Options nosniff
</IfModule>
```

Patching Cadence

The SSL protocol 3.0, as used in OpenSSL through 1.0.1i and other products, makes it easier for man-in-the-middle attackers to obtain cleartext data via a padding-oracle attack, aka the "POODLE"

The SSLv2 protocol, as used in OpenSSL before 1.0.1s and 1.0.2 before 1.0.2g and other products, makes it easier for remote attackers to decrypt TLS ciphertext data by leveraging a Bleichenbacher RSA padding oracle, aka a "DROWN" attack.

Servers which have the DHE_EXPORT cipher enabled for TLS-dependent services are at risk for the Logjam attack. This flaw allows attackers to eavesdrop on and possibly tamper with encrypted connections.

TODO:

- Ensure that all of your SSL / TLS libraries on the affected machines are up to date.
- Disable SSLv3 support on those servers, as described here: https://poodle.io/servers.html

- Ensure that your private keys are not used in SSLv2 connections on the affected server(s).
- Instructions for a variety of systems and software are available here: https://drownattack.com/#check

- Ensure that TLS is properly configured on the affected server(s), as outlined here: https://weakdh.org/sysadmin.html