Australian Government
Australian Signals Directorate

ASD AUSTRALIAN SIGNALS DIRECTORATE
ACSC Australian Cyber Security Centre

Implementing certificates, TLS, HTTPS and opportunistic TLS

First published: October 2019
Last updated: October 2021

Introduction

Transport Layer Security (TLS) is a widely used encryption protocol which enables parties to communicate securely over the internet. Through the use of certificates and Public Key Infrastructure (PKI), parties can identify each other through a trusted intermediary and establish encrypted tunnels for the secure transfer of information.

Using the TLS configuration guidelines outlined in this publication will help strengthen encryption and authentication for web and email communications.

Encrypting web traffic

TLS and Hypertext Transfer Protocol Secure (HTTPS) are protocols that provide encryption and authentication to reassure people (herein referred to as users) that they are connecting to websites they intend to, and that their interactions are not able to be viewed or modified. These protocols are underpinned by cryptographic documents, known as certificates, which can authenticate the identity of websites.

All public-facing websites and Hypertext Transfer Protocol (HTTP)-enabled application programming interfaces (APIs) should use HTTPS to protect the confidentiality and integrity of website communications.

Encrypting email traffic

Opportunistic TLS can be used with the Simple Mail Transfer Protocol (SMTP) to protect the confidentiality and integrity of email. Using TLS and certificates, mail servers are able to authenticate one another and established encrypted communications before transferring email.

All mail servers should offer and use TLS to protect the confidentiality and integrity of email messages whenever possible.

Public Key Infrastructure

PKI arrangements are a key component of the assurance process that enables websites and mail servers to be authenticated. This is achieved by trusted Certificate Authorities (CAs) that vouch for server identities and attest the public key contained in a certificate belongs to the entity noted on the certificate.

How certificates, TLS, HTTPS and opportunistic TLS work

Certificates

Many internet protocols use X.509 certificates to allow web browsers and systems to authenticate websites and servers using public-key cryptography. For example, when a user attempts to access a HTTPS-enabled website, the web server sends the user's web browser its public key contained in a certificate, and demonstrates it has the corresponding private key. The web browser then checks that the certificate was issued by a trusted CA, is valid and was issued to the domain of the website the user is accessing.

To receive a certificate, a server owner will need to apply to a CA and demonstrate that they have control over the domain for which they are requesting a certificate.

Certificate Authorities in web browsers

Common web browsers (e.g. Google Chrome, Firefox, Microsoft Edge and Safari) maintain their own list of trusted CAs. If a user visits a website which offers a certificate signed by a trusted CA, the web browser will accept the certificate without displaying a trust error. However, if a user visits a website which offers a certificate which is not signed by a trusted CA, the web browser will display an error message and refuse to make an encrypted connection until the user acknowledges the risk.

In practice, common web browsers' trusted CAs are closely aligned. Further, when one web browser developer chooses to remove a trusted CA they will often coordinate with other web browser developers, or other web browser developers will shortly follow suit.

If the CA which has signed a website's certificate is removed from a web browsers' trusted CA list, the website's users will begin to receive security error messages when attempting to connect to the website. This may cause some disruption and embarrassment to the organisation that owns the website until such time that the certificate is replaced.

When choosing a CA to issue a certificate for a website or HTTP-enabled API, organisations should consider the reputation and history of the CA, and other organisations that support or utilise the CA. Finally, the price of certificates is not an indicator of security, quality or longevity.

Certificate Authorities in mail servers

Mail servers support TLS via a mechanism known as opportunistic TLS. Under opportunistic TLS, any encryption is considered better than no encryption. Consequently, mail servers have historically accepted certificates which were expired, not signed by a trusted CA or didn't match the name of the receiving mail server. However, new standards and improved server hygiene are beginning to challenge these practices.

Mail servers can use certificates to authenticate one another over the internet as part of establishing TLS. In particular, sending mail servers can use the server certificate offered by the receiving mail server to identify that the mail server is the intended destination of an email message.

As verification of certificates historically was not critical to email flow, mail servers may not have kept their internal list of trusted CAs up to date and aligned with a lists of well-known and trusted public CAs. Hence, mail server operators may find they need to implement processes to ensure the list of acceptable CAs on their mail servers are maintained.

In the absence of their own specific policy, mail server operators can choose one of the well-known web browsers and follow the web browser's list of approved CAs as trusted root CAs.

Types of certificates

CAs typically offer three products: Domain Validation (DV) certificates, Organisation Validation (OV) certificates and Extended Validation (EV) certificates:

- DV certificates can be issued to anyone who demonstrates control over a domain.
- OV certificates include more details about the organisation to whom they are issued.
- EV certificates are considered to be of higher identity assurance than OV and DV certificates as they require an organisation to provide extensive documentation to prove identity and ownership of a domain. Procuring EV certificates takes greater time and resources compared to DV certificates. The issuance of these certificates cannot be automated.

DV certificates provide sufficient confidentiality and authentication to suit most websites and can be obtained for free, or at a price. Additional security or assurances are not achieved by paying for a DV certificate.

OV certificates are priced above DV certificates and allow for more organisational data to be added to the certificate. Users won't know the difference between a DV and an OV certificate without closely examining certificate details.

EV certificates are the most expensive and require additional identity checks before one can be issued. Common web browsers such as Google Chrome, Firefox and Safari used to provide users with additional visual cues when an EV certificate was being used, but this no longer occurs.

The three certificates types are differentiated by the level of assurance that a domain is owned by an organisation, not by the level of encryption offered. A more expensive certificate does not provide a greater level of confidentiality.

Encryption algorithms

Certificate encryption and hashing algorithms should be selected from the *Information security manual* (ISM). These can be specified with a CA when requesting a certificate. To change the algorithms being used a new certificate will need to be issued.

Transport Layer Security

TLS is a cryptographic protocol that allows for end-to-end encrypted communications over a network. It is used in a variety of applications and builds on the deprecated Secure Socket Layer (SSL) protocol developed by Netscape in 1994.

Versions of TLS earlier than TLS 1.3 may be susceptible to cryptographic compromise. If organisations need to use older versions of TLS, care should be taken to select cipher suites and configure protocol features in accordance with this guide and the ISM. SSL should not be used.

Cipher suites

A key element of understanding how TLS works is understanding what a cipher suite is. A cipher suite is a set of algorithms that help secure a TLS connection. Cipher suites generally include three different algorithms:

- a key establishment algorithm for securely establishing a symmetric key between two devices

- a bulk encryption algorithm (which uses the symmetric key) for encrypting traffic that is sent over the connection
- a Message Authentication Code algorithm for providing assurance that traffic is not modified in transit.

There are many different cipher suites. Selecting the right cipher suite is important as weak cipher suites increase the risk to users' confidentiality.

TLS 1.3 removed vulnerable cipher suites found in TLS 1.2, while introducing stronger cipher suites. Advice on acceptable cipher suites is outlined in Annex A while advice on potentially acceptable cipher suites is outlined in Annex B.

TLS handshake process

The following is a simplified explanation of the TLS handshake process:

- the client and server agree on the cryptographic protocol (e.g. TLS 1.3) and cipher suite
- the client authenticates the server:
  - the server offers its certificate and proves that it holds the private key by signing a message which the client can verify using the public key contained in the certificate
  - the client verifies the identity of the server by checking the name of the server matches the name on the certificate
  - the client verifies that the certificate is valid by being in date, not revoked and issued by a CA which the client trusts
- the server and client exchange the symmetric key.

More information on the TLS handshake process is available from Cloudflare.

Perfect Forward Secrecy

Perfect Forward Secrecy (PFS) is a feature of certain cipher suites that reduces the risk posed by compromised session keys. With PFS, individual sessions have unique session keys. As a result, a compromised session key cannot be used to decrypt other sessions. This means that attacks that rely on long-term storage of encrypted data become infeasible. Organisations should only use cipher suites that support PFS.

Secure renegotiation and client-initiated renegotiation

TLS 1.3 does not use renegotiation, however, if using TLS 1.2 or earlier, renegotiation may be required under certain circumstances. For example, when a session has expired but parties wish to send more data, a peer wants to change cipher suites or there is a need for the parties to perform authentication. Unfortunately renegotiation is susceptible to person-in-the-middle attacks. For TLS 1.2 or earlier, secure renegotiation should be enabled to reduce susceptibility to person-in-the-middle attacks.

Client-initiated renegotiation, secure or otherwise, imposes a performance impact on web servers. A malicious client can send many renegotiation requests to consume server resources causing a denial of service. For TLS 1.2 or earlier, client-initiated renegotiation should be disabled to prevent denial-of-service attacks.

TLS compression

TLS compression was used to decrease the bandwidth of TLS communications. However, TLS compression has been found to inadvertently leak information, as illustrated by the CRIME exploit. TLS compression should be disabled.

Review TLS settings regularly

Best practice TLS settings and cipher suites change as new standards are released, computing power increases and vulnerabilities are discovered. Organisations should review their TLS and cipher suite configurations annually, or whenever major vulnerabilities are publically disclosed, to ensure that protection remains effective and in line with their customers' expectations.

Hypertext Transfer Protocol Secure

HTTPS allows for secure communication across an untrusted network, reducing the ability of malicious actors to monitor a user's activity. Importantly, lack of HTTPS can erode trust in a website as common web browsers will alert users when visiting such websites, and again when they try to enter any information on such websites. It should also be noted that some web browsers will block content that is delivered over HTTP when connecting to a website over HTTPS.

All website content should be delivered using HTTPS, and any attempted access to resources using HTTP should be automatically redirected to the same resource over HTTPS. In addition, applications which make use of HTTP-enabled APIs should also use TLS to protect the confidentiality and integrity of information in transit.

There are many myths suggesting reasons for not using HTTPS. These are debunked at *Does My Site Need HTTPS*.

HTTP Strict Transport Security

HTTP Strict Transport Security (HSTS) is a web security policy mechanism that helps protect users. It achieves this by allowing web servers to tell web browsers that they should only interact with a web server over HTTPS. As such, web browsers will dynamically adjust any HTTP requests to HTTPS requests. HSTS also helps to protect against eavesdropping, person-in-the-middle attacks and active network attacks. Organisations should use HSTS to protect users' confidentiality.

Opportunistic TLS and email

Opportunistic TLS allows mail servers to use encryption to protect email in transit when the sending and receiving mail servers both support TLS.

Before transferring data the sending mail server (if it is TLS capable) will ask the receiving mail server if it supports TLS. If the receiving mail server indicates that it does then the two mail servers will begin trying to negotiate a TLS protocol version and acceptable set of cipher suites. However, if the receiving mail server indicates it does not support TLS then the sending mail server will proceed to transmit the email in plain text.

The necessity to be able to support plain text transfers to mail servers that do not support TLS makes opportunistic TLS highly susceptible to downgrade (aka TLS strip) attacks by a person-in-the-middle.

Mail Transfer Agent Strict Transport Security and SMTP TLS Reporting

Similar to HSTS, Mail Transfer Agent Strict Transport Security (MTA-STS) provides a mechanism for a domain owner to indicate that their mail servers support TLS encryption and that email should not be sent to those mail servers unless satisfactory encryption can be established.

An MTA-STS aware sending mail server which tries to send email to a domain that has published an MTA-STS policy in enforce mode will only send the email if:

- it can negotiate at least a TLS 1.2 connection with the recipient mail server
- the recipient mail server presents a certificate to authenticate itself that:
    - is signed by a CA trusted by the sending mail server
    - aligns with the recipient mail server's name as published in the domain's MX record and MTA-STS policy
    - is otherwise valid (e.g. not expired or revoked).

MTA-STS can prevent:

- downgrades in order to perform plain text attacks
- downgrades in order to perform lower protocol version attacks
- fake, untrusted, self-signed or otherwise invalid certificate attacks (where the person-in-the-middle terminates the TLS session on their system relying on the weak certificate validation in opportunistic TLS).

SMTP TLS Reporting (TLS-RPT) is a related standard to MTA-STS. TLS-RPT provides a mechanism for a domain owner to publish a location where other mail server operators can submit reports about their success or failure trying to initiate encrypted sessions when sending email to the specified domain.

MTA-STS is defined in Internet Engineering Task Force (IETF) RFC 8461 and is supported by TLS-RPT as defined in IETF RFC 8460.

MTA-STS policies are made available by publishing two pieces of information:

- a TXT record in DNS that indicates that the domain has an MTA-STS policy and the current version number (*_mta-sts.<example domain>*)
- a MTA-STS policy (a plain text file) at the standard-required website location for the domain (*https://mta-sts.<example domain>/.well-known/mta-sts.txt*).

An example of an MTA-STS TXT record is *v=STSv1; id=2020060141732* where:

- *v* is the version
- *id* is the identification of the record that MTA-STS aware mail servers track to determine if they need to request the policy file, which is usually only checked when the identification number in DNS is changed.

An example of a MTS-STS policy is:

version: STSv1

mode: testing

*mx: *.example1.org.au*

max_age: 600

where:

- *version* is the MTA-STS version (currently only version 1 is supported)
- *mode* is the current approach to requiring TLS for email connections, testing should be used initially and then increased in strictness to enforce as you become more confident
- *mx* is the mail server to which this policy applies
- *max_age* is the minimum time in seconds that this policy should be cached for.

TLS-RPT policies are implemented by:

- deciding whether to receive reports by email or HTTPS and creating a location to receive those reports (such as an email address)
- publishing a TXT record in DNS for your domain (*_smtp._tls.<your domain>*) that indicates the reporting point for TLS reports.

While TLS-RPT is not strictly required for MTA-STS, it should be enabled as part of your MTA-STS implementation to detect any issues.

An example of a TLS-RPT policy is *v=TLSRPTv1;rua=mailto:tlsresports@example1.org.au* where:

- *v* is the version
- *rua* is the reporting address for TLS-RPT, specifying that reports should be sent by email.

Risks from MTA-STS

Implementing MTA-STS in enforce mode is an indication from a domain owner that confidentiality and integrity of email flows are more important than availability. Consequently, the domain owner accepts that email flows may be interrupted in preference to email potentially being intercepted, disclosed or modified by third parties.

Organisations looking to implement MTA-STS should:

- ensure adequate monitoring of email flows and that TLS-RPT is in place
- accept that, even with good monitoring, MTA-STS may occasionally cause disruption to email flows.

It is worth noting that once properly bedded in and configured, disruptions to email flows are likely to be either the result of misconfiguration or malicious activity.

How to implement certificates, TLS and HTTPS

Which cipher suites and encryption methods to support

Picking strong cipher suites is important for users' confidentiality. For the certificate signature algorithm, organisations are encouraged to use Elliptic Curve Digital Signature Algorithm (ECDSA) (256-bit or larger) or Rivest-Shamir-Adleman (RSA) (2048-bit or larger). If using elliptic curve cryptography, a curve from Federal Information Processing Standard 186-4 should be used.

Organisations should not use any cipher suite that uses the following algorithms as they have cryptographic weaknesses:

- Rivest Cipher 2
- Rivest Cipher 4
- Message-Digest 5
- Data Encryption Standard
- EXPORT
- NULL
- Secure Hash Algorithm 1
- Anonymous Diffie-Hellman
- Anonymous Elliptic Curve Diffie-Hellman.

Advice on acceptable cipher suites is outlined in Annex A while advice on potentially acceptable cipher suites is outlined in Annex B.

Pick and install a certificate

As mentioned previously, the three certificate types provide the same level of security differing only in the level of assurance that a domain is owned by an organisation. Unless there is a business requirement for a higher assurance certificate, a free DV certificate should be used. In addition, using a CA that provides support for the Automated Certificate Management Environment (ACME) protocol will enable support for HTTPS in a low maintenance manner.

Let's Encrypt is an example of a free DV certificate provider that supports ACME. A case study on setting up Let's Encrypt with automatic renewal of certificates is provided at the end of this publication. For more information on Let's Encrypt see their *Getting Started* guide.

Automate renewal of certificate

Organisations should endeavour to automate certificate renewal. This can be done using a CA that supports the ACME protocol. Automation of certificate renewal minimises the risk that a website become insecure, or even inaccessible, if its certificate isn't renewed in time.

Certbot is a common ACME client that assists in obtaining and installing Let's Encrypt certificates. It is simple, has comprehensive documentation and works on many platforms. Certbot can setup HTTP redirects, HSTS and load all resources through HTTPS. Alternatively, a list of alternate ACME clients is available on the Let's Encrypt website.

Manual setup

If choosing to install a certificate manually, a Certificate Signing Request (CSR) will need to be created, the corresponding certificate will need to be ordered and then the certificate will need to be installed.

There are several resources available to assist with this process such as wikiHow's text tutorial and GlobalSign's video tutorials:

- How to Create a CSR using MMC
- How to Create a CSR in IIS 10
- How to Create a Java Key Store and Generate a CSR
- How to Create a CSR in Apache OpenSSL
- How to Install an SSL/TLS Certificate on Microsoft IIS
- How to install SSL/TLS Certificate on an Apache Tomcat Server
- How to Install an SSL/TLS Certificate on an NGINX server.

The SSL Store also has advice on creating a CSR and installing a TLS certificate.

Once manual setup has been completed, a TLS tester such as SSL Labs can check the implementation.

Redirect HTTP requests

To ensure that users trying to access a website over HTTP are redirected to HTTPS, the web server's or reverse proxy's configuration should be edited. While syntax will differ between web server software, an example for Apache HTTP Server is shown below that causes an HTTP 301 redirect whenever the HTTP version of the website is requested.

- *RewriteEngine On*
- *RewriteCond %{HTTPS} off*
- *RewriteRule ^(.*)$ https://%{HTTP_HOST}%{REQUEST_URI} [L,R=301]*

Note, rewrite rules can be complex and it is important to test configurations before and after deployment.

Alternatively, redirection can be achieved in IIS by using the Uniform Resource Locator (URL) Rewrite Module while Let's Encrypt and Certbot can setup redirects automatically when a certificate is obtained.

Regardless of how HTTP to HTTPS redirects are implemented, websites should redirect to the HTTPS version of the website before redirecting elsewhere. For example, if redirecting requests for *website.com.au* to *www.website.com.au*, make sure to first redirect to the HTTPS version of *website.com.au* before redirecting to *www.website.com.au*. This is required for HSTS to operate correctly. Once the redirection configuration has been tested and confirmed, it is best to use a 301 (permanent) redirect as this typically improves search engine rankings.

Check for mixed content

While users may be able to access a website over HTTPS, parts of the website may still be retrieved using HTTP, such as images or embedded content. Most web browsers will block this mixed content. To check if a website has mixed content issues, either use Firefox's Web Console to view individual pages or use free online services such as *Missing Padlock – SSL Checker*.

Improving search engine optimisation

Moving a website to HTTPS will result in search engines treating the website as new, which could place it lower in search engine rankings. However, moving to HTTPS can also improve search engine rankings as many search engines place websites loaded over HTTPS higher in their search results. To minimise any negative impact on websites' search

engine optimisations, new HTTPS websites can be added and verified, for example, in the Google Search Console. This will re-crawl the website and submit a new XML sitemap for the HTTPS version.

HSTS header

Even when HTTPS configurations and HTTP redirects are setup correctly, a HSTS header should still be used. This will ensure that for a specified period of time users will only be able to connect to a website using HTTPS. This is recommended to prevent malicious actors from intercepting users' initial HTTP requests. If using Let's Encrypt and Certbot, sending HSTS headers can be setup automatically when obtaining a certificate.

Further information on HSTS implementation is available from GlobalSign.

Sign up for the HSTS Preload List

Signing up for the HSTS Preload List ensures that a user's web browser will prevent HTTP requests being made to specified websites. This is different from a HSTS header as it occurs before a user has visited a website for the first time, provided they are using a web browser which supports the pre-load list. Thus, users are protected from attacks where malicious actors prevent the HSTS header from being sent, such as in person-in-the-middle attacks.

The advice at hstspreload.org should be followed to register websites for the HSTS Preload List.

Drop support for TLS 1.0 and TLS 1.1 immediately

All versions of TLS earlier than TLS 1.2, including SSL, are considered unsafe due to either having a flaw in the protocol itself or using vulnerable cipher suites that could leak confidential information. As such, support for versions of TLS earlier than TLS 1.2 will be dropped by Google, Mozilla, Microsoft and Apple starting in 2020. Importantly, if ongoing support for TLS 1.0 or TLS 1.1 is required for any reason, organisations are strongly encouraged to contact the Australian Signals Directorate to discuss possible risk mitigation strategies.

Be aware of TLS 1.2 limitations

TLS 1.2 or higher is required for HTTP Version 2 (HTTP/2), which provides significant performance improvements over HTTP and should be supported in order to provide a better user experience. However, due to deficiencies in TLS 1.2 and some of the included cipher suites, use of HTTP/2 with TLS 1.2 is limited. For example, HTTPS may fail if non-secure renegotiation and compression has not been disabled for TLS 1.2 in web server settings. Further advice on limitations associated with the use of TLS 1.2 are available from the HTTP/2 standard.

Transitioning from TLS 1.2 to TLS 1.3

As noted earlier, TLS 1.3 provides many security and performance benefits over TLS 1.2 and earlier versions. A simple way to disable older versions of TLS is to update web server configurations (e.g. an example configuration for Apache HTTP Server would be to include *SSLProtocol TLSv1.3* in the configuration file). In some cases, web server software will need to be updated to the latest version to support TLS 1.3. In addition, if using a Content Distribution Network (CDN), enabling TLS 1.3 for external clients may only require ticking a box in configuration settings. The encryption configuration between any origin servers and the CDN may also require adjustment.

It is important to note though, as TLS is not backwards compatible, dropping support for TLS 1.2 will result in legacy web browsers being unable to access websites using TLS 1.3. In such cases it is recommended that users be encouraged to update their web browser to the latest version to receive support for TLS 1.3 and to protect their confidentiality. However, if transitioning from TLS 1.2 to TLS 1.3 is not possible in the short term, risks to users can be reduced by:

- removing support for compression

- disabling client-initiated renegotiation
- disabling renegotiation if the version of TLS does not support secure renegotiation
- only enabling secure ciphers.

Advice on acceptable cipher suites is outlined in Annex A while advice on potentially acceptable cipher suites is outlined in Annex B.

Mozilla's wiki provides advice on TLS configuration, including example configurations for security, compatibility and legacy systems. It also specifies what web browsers a particular configuration will be compatible with.

Additional resources

Additional resources on the implementation of TLS are available from the following sources:

- SSL Labs have advice on *SSL and TLS Deployment Best Practices*
- the Internet Engineering Task Force have published *Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*
- Google's Lighthouse tool can assist with locating HTTP-only content and also provides general tips on website security
- Mozilla has a TLS configuration generator for different web server software
- Mozilla Observatory provides a HTTP and TLS score card for websites.

Case study using Let's Encrypt and Certbot

This case study shows how to enable HTTPS using Let's Encrypt and Certbot on Ubuntu 16.04 with Apache HTTP Server. It uses a simplified deployment scenario where the web server will perform its own TLS termination and generate its own certificate renewal requests. Depending on the security context, organisations may make use of proxies and other security infrastructure as part of their solution.

Firstly, add the required software repositories and install Certbot by running the following commands:

- *sudo apt update*
- *sudo apt install software-properties-common*
- *sudo add-apt-repository universe*
- *sudo add-apt-repository ppa:certbot/certbot*
- *sudo apt update*
- *sudo apt install certbot python-certbot-apache*.

Once everything is installed, obtain a certificate by running the following command, *sudo certbot --apache --rsa-key-size 2048 --redirect –hsts* where:

- *--rsa-key-size 2048* sets the bit length of the RSA key to 2048

- *--redirect* automatically makes the website redirect to the HTTPS version
- *--hsts* makes sure the HSTS header is automatically sent.

There is also a *--uir* flag that will attempt to change every HTTP resource on the website to HTTPS. This is potentially dangerous as some resources might not have a HTTPS address, but if all resources on the website can be loaded securely this is a quick way to automate the transition of resources from HTTP to HTTPS.

After running the above certificate generation command, Certbot will ask for an email address for renewal and security notices. Type in the preferred email address and press Enter. Next, it will ask for agreement to their Terms of Service. After reviewing, type 'A' and press Enter. Then, it will offer to send email from the Electronic Frontier Foundation. Press 'Y' or 'N' and press Enter. Finally, Certbot will ask for the website's domain. Type it in and press Enter.

As certificates only last 90 days, certificates will need to be renewed often. Luckily, the Certbot packages come with a Cron Job that will renew certificates automatically before they expire. To test that it is all working correctly, run the following command, *sudo certbot renew --dry-run*. Note, the extra flags used when creating the certificate don't have to be specified as 'renew' will use the settings of the last successfully created/renewed certificate.

How to encrypt email with opportunistic TLS

Implementing opportunistic TLS to protect email flows shares many similarities with implementing HTTPS to protect web traffic. Readers of this section should familiarise themselves with the section on implementing HTTPS as many concepts are shared, and not repeated here to avoid repetition.

Protocol and cipher suite selection

Opportunistic TLS for email presents a slightly different threat scenario to HTTPS thereby requiring a different approach to protocol and cipher selection. Key differences are:

- a downgrade attack against opportunistic TLS results in plain text communication as opposed to a weaker version of TLS
- mail server TLS support is not as well homogenised or regularly updated as TLS support in web browser and web server software.

By choosing to only support modern versions of protocols and cipher suites you may prevent older mail servers from using any encryption at all. Consequently, you may wish to accept the risk of continuing to support older, less secure protocol version and ciphers on the basis that some encryption is better than none. However, care should be taken in configuring these older versions of TLS to minimise risk. Further, you should consider using MTA-STS to reduce the risk of downgrade attacks.

Mail servers should:

- support the latest version of TLS (currently 1.3), TLS 1.2 (for compatibility with MTA-STS) and optionally TLS versions as old as TLS 1.0
- not support any version of SSL
- not support TLS compression
- not support TLS renegotiation, or only support secure server-initiated renegotiation

- avoid ciphers noted as weak in the 'which cipher suites and encryption methods to support' section above.

Certificates for mail servers

Mail servers should have valid server certificates issued by well-known CAs, such as Let's Encrypt, so that other mail servers can positively authenticate them over the internet. This becomes critical if implementing an MTA-STS policy.

The advice on obtaining and keeping certificates up to date in the HTTPS section above is also applicable.

Mail Transfer Agent Strict Transport Security

Google, who are one of the key sponsors of MTA-STS and TLS-RPT, have provided a useful guide to implementing MTA-STS. This guide can be read in conjunction with the RFCs. In addition, there are tools that can help verify the correctness of your MTA-STS policy.

The following sections provide further guidance on implementing MTA-STS and TLS-RPT as an email sender (making your mail servers aware of MTA-STS and TLS-RPT policies published by others) and email recipient (publishing MTA-STS and TLS-RPT policies).

Making your mail servers MTA-STS aware

To make others' MTA-STS policies effective, and prevent downgrade attacks on email sent by your organisation, your mail servers need to look for, interpret and apply MTA-STS policies published by other domain owners.

Your ability to support the application of MTA-STS policy to outbound email will be largely contingent on your mail server offering an appropriate feature or policy agent. However, once this is available, enabling it will be a relatively low risk change.

The following checklist will assist in making your mail servers MTA-STS aware:

- Identify a policy agent or configuration option for your edge mail server (the mail servers that relay email, using MX records, to other domains over the internet) which can interrogate and apply MTA-STS and TLS-RPT policies.
- Review how you currently maintain trusted CAs on the mail server where you are implementing the MTA-STS policy agent, and ensure that there is a process (ideally automated) for keeping this list aligned with one of the major web browser lists of accepted CAs.
- Ensure your mail server can support TLS 1.2.
- Implement the policy agent.
- Monitor the sending of email by your mail server and ensure it continues to send successfully to MTA-STS enabled domains.

Note, MTA-STS requires mail servers to support TLS 1.2. Mail servers should also be configured with TLS 1.3 where possible. This allows negotiation of a more secure cryptographic protocol if both servers are able.

Publishing an MTA-STS policy for your domain

Publishing an MTA-STS policy for your domain is a change you can immediately implement, provided your inbound mail servers (the ones identified by your domain's MX records) support at least TLS 1.2.

A suggested implementation plan for publishing an MTA-STS policy for your domain is:

- Review your domain's mail servers' (the servers identified in your domain's MX record) cryptographic configuration and certificate. Ensure that your email severs:
  - support TLS 1.2, and ideally TLS 1.3, noting the advice earlier in this publication that you may want to risk manage using TLS versions as low as 1.0
  - have disabled weak ciphers, and all forms of SSL
  - are configured with a certificate which correctly authenticates the server, and is issued by a well-known and trusted CA in all major web browsers' trusted CA list; also consider automating the renewal of certificates as after implementing MTA-STS a certificate expiry will interrupt email flow and potentially be time consuming to identify
  - use online TLS checking tools to confirm the mail server is operating as intended.
- Create a reporting point for TLS-RPT (either email or HTTPS).
- Publish a TLS-RPT policy.
- Monitor reports received via your reporting point and use them to confirm that other mail servers are able to successfully negotiate TLS, at a sufficient level, when communicating with your mail servers. If reports identify any issues try to correct them before moving on.
- Prepare and publish a test MTA-STS policy:
  - create a new website with a new certificate issued by a well-known and trusted CA (*mta-sts.<yourdomain>*)
  - create an MTA-STS policy for testing
  - publish your test MTA-STS policy at the standard-required location on your new website (*https://mta-sts.<yourdomain>/.well-known/mta-sts.txt*)
  - use tools such as a web browser to confirm that the MTA-STS policy is correctly served when going to the standard-required location.
- Publish your MTA-STS TXT record in DNS:
  - create the relevant DNS entry
  - use a short timeout (TTL) for the DNS record so you can reverse the change if you encounter any errors
  - verify it is configured correctly.
- Continue to monitor TLS-RPT for any issues. Once satisfied that everything is behaving as expected, move to testing an enforcing policy.
- Prepare and publish a new enforcing policy:
  - create a new policy file, using enforcing mode and a new policy ID, using a low lifetime for the policy so that changes can be reversed if necessary
  - publish the new policy file to the standard-required location and adjusting the MTA-STS TXT record in DNS to the new ID reference number.

- Once published:
  - confirm correct configuration using tools
  - verify email continues to flow by monitoring mail server logs
  - continue to monitor TLS-RPT to identify if any mail servers on the internet report issues.
- Once you are confident your enforcing policy is working as expected:
  - increase the lifetime timeout on your policy, note that every time you adjust this policy you should also change the policy ID in the MTA-STS TXT record in DNS so that sending mail servers are aware of the policy change
  - increase the cache time for the MTA-STS TXT record in DNS, noting increasing your policy lifetime and the DNS cache time reduces the opportunity for malicious actors to try and bypass your policy by preventing others from being able to retrieve your policy.

Implementing MTA-STS with a service provider

If you use service providers, implementing MTA-STS may require a different approach. Unfortunately, it is not possible to provide an exhaustive list of all possible arrangements, but two common scenarios are where a service provider is used to scan inbound email before it is delivered to your own mail server, or host your entire email system (such as Microsoft 365 or Google Gmail). In these cases, you, as the domain owner, will be responsible for determining and publishing MTA-STS and TLS-RPT policies, but your service provider will be responsible for configuring adequate encryption and valid certificates on the mail servers they operate.

While most service providers will likely operate their mail servers at the necessary standard, you should still confirm they meet and will maintain the necessary standards before attempting to implement MTA-STS. In most cases, implementing TLS_RPT will be a safe option irrespective of your service provider's position. This will allow you to review what encryption other mail servers on the internet achieve when connecting to your service provider's systems.

The implementation of an MTA-STS policy should largely follow the plan laid out in the 'publishing an MTA-STS policy for your domain' section above, although it will be your service provider's mail servers being specified.

Edge cases

MTA-STS includes no inheritance mechanism for managing subdomains. If MTA-STS protection is required for subdomains, they will need their own DNS TXT record (e.g. *_mta-sts.subdomain.example1.org.au*) and standard-required policy location. However, in general, subdomains will only require protection if they receive email (i.e. they have an MX record published).

Example implementation of MTA-STS

Maria is the systems administrator for Example1 (*example1.org.au*) and wishes to implement MTA-STS to protect the confidentiality and integrity of email. Inbound email to Example1 is handled by two mail servers, *mail1.example1.org.au* and *mail2.example1.org.au*.

example1.org.au.	*86400*	*MX*	*10 mail1.example1.org.au.*
	86400	*MX*	*10 mail2.example1.org.au.*

Maria begins by reviewing the TLS configurations on *mail1.example1.org.au* and *mail2.example1.org.au*. Maria confirms each server supports the necessary versions of TLS, disables SSL and weak ciphers, and ensures that each mail server has a certificate issued by a well-known and trusted CA. Because Maria doesn't want email flows interrupted by certificates expiring, she also takes the time to automate certificate renewal.

When Maria is finished she uses an online tool to verify the offered cryptography meets the intended configuration and makes a diary note to check in 80 days (as the certificates expire in 90 days) that the mail server certificates have been renewed correctly.

After reviewing IETF RFC 8460, Maria decides to use email to receive reports and creates the *tlsreports@example1.org.au* email address for this purpose.

After confirming this email address can receive email, Maria moves to the next step and publishes the TLS-RTP policy. Maria is aware that publishing the TLS-RTP address is relatively safe and sets a moderately long timeout of one day (86,400 seconds):

_smtp._tls.example1.org.au. 86400 TXT v=TLSRPTv1;rua=mailto:tlsresports@example1.org.au

Over the next week Maria reviews the reports received until she is confident things are operating as they should.

Maria then reviews IETF RFC 8641, to re-familiarise herself with its requirements, and also reviews other online material.

Maria begins to devise an MTA-STS policy based on her understanding of RFC 8641 and examples she has seen elsewhere. Maria decides to set the policy cache lifetime to 600 seconds (ten minutes) while she is testing. Maria also decides to specify the MX servers using wildcard MX syntax, allowing her to add or change MX servers later provided she maintains the same naming standard. The resultant policy is thus:

version: STSv1

mode: testing

*mx: *.example1.org.au*

max_age: 600

Having drafted a MTA-STS policy, Maria moves to the next step of creating the necessary website to host the policy file. Maria notes the website only needs to host a static file and considers hosting the content on a cloud service provider's bucket service combined with a content delivery network to provide TLS, but in the end chooses to create a new website on an existing webserver. As such, Maria:

- creates a new website on the server for *mta-sts.example1.org.au*
- creates a subdirectory *.well-known* and puts the policy file from above in there as *mta-sts.txt*
- configures TLS on the website using a wildcard certificate (**.example1.org.au*) that she already has on the web server for other websites
- creates a DNS A record pointing *mta-sts.example1.org.au* at the web server's IP address
- uses a web browser on an external connection to ensure that *https://mta-sts.example1.org.au/.well-known/mta-sts.txt* serves the MTA-STS policy file she created earlier.

Having completed all prerequisites, Maria is now ready to make the MTA-STS policy live by publishing a DNS record to indicate the policy is available. Maria wants to be able to quickly change the policy in case there are problems so she choose a low DNS cache time (aka TTL) of 300 seconds (5 minutes). From the RFC she knows the value of the ID number in the TXT record is not important, except that changing it indicates a new policy is available. Maria notes from the examples she has seen that using a timestamp in YYYYMMDDHHMM format (e.g. 202006071753 for 5:53pm on 07/06/2020) appears to be a common practice and decides to adopt it.

Maria sets the following DNS record:

_mta-sts.example1.org.au. 300 TXT v=STSv1; id=202006071753

Maria uses an online tool to confirm that the configuration appears correct.

Maria doesn't anticipate any email flow issues as the policy is in testing mode, but nevertheless closely monitors email flows immediately afterwards, and also monitors web server logs to determine the MTA-STS policy is being requested.

For the next week Maria keeps a close eye on mail flows and also reviews any SMTP TLS reports received by the address she created earlier. After a week there have been no problems and Maria decides to implement an enforcing policy. As such, Maria opens the policy file on the *mta-sts.example1.org.au* website and changes the mode to enforce, but leaves the *max_age* the same so she can reverse the change quickly if needs be. The file now reads:

version: STSv1

mode: enforce

*mx: *.example1.org.au*

max_age: 600

Having adjusted the policy, Maria notifies the change to other mail servers on the internet by changing the ID value in the *_mta-sts.example1.org.au* TXT record, but leaves the DNS cache time at a relatively low 300 seconds so she can undo any changes if there are issues.

_mta-sts.example1.org.au. 300 TXT v=STSv1; id=2020060141732

Maria uses an online tool to confirm that the configuration appears correct.

Maria knows this change may impact email flow, so in addition to closely monitoring email flows immediately afterwards, she also uses an online account of an MTA-STS aware webmail service to send a test email to her email address at *example1.org.au* and confirms receipt.

With the enforcing policy in place, Maria keeps a close eye on email flows over the next month. Satisfied that everything is working as it should, Maria makes two final changes:

- she sets the *max_age* in the *mta-sts.txt* policy file to 86,400 (one day)
- she adjusts the ID number in the *_mta-sts.example1.org.au* to the current time in YYYYMMDDHHMM format to notify other mail servers on the internet of the change.

Finally, Maria makes a note in her diary to re-evaluate the *max_age* setting in six months' time, and lift it to a week, a month or even more depending on whether any issues with email flow are encountered.

Further information

The *Information security manual* is a cybersecurity framework that organisations can apply to protect their systems and data from cyberthreats. The advice in the *Strategies to mitigate cybersecurity incidents*, along with its Essential Eight, complements this framework.

Contact details

If you have any questions regarding this guidance you can write to us or call us on 1300 CYBER1 (1300 292 371).

Annex A: Acceptable cipher suites

This annex contains a list of acceptable cipher suites. Cipher suites are listed in their order of preference.

All cipher suites below are listed in their Internet Assigned Numbers Authority names. Different web server software may use different syntax. Websites are available to assist in translating cipher suite names.

Cipher suites for TLS 1.3

The following cipher suites are recommended for use with TLS 1.3:

- TLS_AES_256_GCM_SHA384
- TLS_AES_128_GCM_SHA256
- TLS_AES_128_CCM_SHA256
- TLS_AES_128_CCM_8_SHA256.

Annex B: Potentially acceptable cipher suites

This annex contains a list of potentially acceptable cipher suites. Cipher suites are listed in their order of preference.

All cipher suites below are listed in their Internet Assigned Numbers Authority names. Different web server software may use different syntax. Websites are available to assist in translating cipher suite names.

Cipher suites for TLS 1.3

Organisations may make a risk-based decision on the use of TLS_CHACHA20_POLY1305_SHA256 as it may provide better performance on platforms which do not have Advanced Encryption Standard (AES) hardware support, such as ARM-based mobile phones, tablets and low-end laptops. However, while there are no publically disclosed weaknesses with this cipher suite, it has not been subjected to the same standard of review and analysis as cipher suites that comply with the ISM.

Cipher suites for TLS 1.2

Organisations continuing to use TLS 1.2 will need to make a risk-based decision on its use as it no longer complies with the ISM. In doing so, the following cipher suites are recommended for use with TLS 1.2:

- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_DHE_DSS_WITH_AES_256_GCM_SHA384
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_DHE_DSS_WITH_AES_128_GCM_SHA256
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_CCM
- TLS_DHE_RSA_WITH_AES_256_CCM
- TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8
- TLS_DHE_RSA_WITH_AES_256_CCM_8
- TLS_ECDHE_ECDSA_WITH_AES_128_CCM
- TLS_DHE_RSA_WITH_AES_128_CCM
- TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8
- TLS_DHE_RSA_WITH_AES_128_CCM_8.

Organisations may make a risk-based decision on the use of the following cipher suites as they may provide better performance on platforms which do not have AES hardware support, such as ARM-based mobile phones, tablets and low-end laptops. However, while there are no publically disclosed weaknesses with these cipher suites, they have not been subjected to the same standard of review and analysis as cipher suites that comply with the ISM:

- TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_CAMELLIA_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384
- TLS_DHE_DSS_WITH_CAMELLIA_256_GCM_SHA384
- TLS_DHE_DSS_WITH_ARIA_256_GCM_SHA384
- TLS_DHE_RSA_WITH_CAMELLIA_256_GCM_SHA384
- TLS_DHE_RSA_WITH_ARIA_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256
- TLS_DHE_DSS_WITH_CAMELLIA_128_GCM_SHA256
- TLS_DHE_DSS_WITH_ARIA_128_GCM_SHA256
- TLS_DHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
- TLS_DHE_RSA_WITH_ARIA_128_GCM_SHA256.

Disclaimer

The material in this guide is of a general nature and should not be regarded as legal advice or relied on for assistance in any particular circumstance or emergency situation. In any important matter, you should seek appropriate independent professional advice in relation to your own circumstances.

The Commonwealth accepts no responsibility or liability for any damage, loss or expense incurred as a result of the reliance on information contained in this guide.

For more information, or to report a cybersecurity incident, contact us:

cyber.gov.au | 1300 CYBER1 (1300 292 371)

Australian Government

Australian Signals Directorate