FTPS

This article is about FTP over SSL. For FTP Software, the defunct network software company, see FTP Software.

FTPS (also known as **FTP-ES**, **FTP-SSL** and *FTP Secure*) is an extension to the commonly used File Transfer Protocol (FTP) that adds support for the Transport Layer Security (TLS) and the Secure Sockets Layer (SSL) cryptographic protocols.

FTPS should not be confused with the SSH File Transfer Protocol (SFTP), an incompatible secure file transfer subsystem for the Secure Shell (SSH) protocol. It is also different from FTP over SSH, the practice of tunneling FTP through an SSH connection.

1 Background

The File Transfer Protocol was drafted in 1971 for use with the scientific and research network, ARPANET.[1] Access to the ARPANET during this time was limited to a small number of military sites and universities and a narrow community of users who could operate without data security and privacy requirements within the protocol.

As the ARPANET gave way to the NSFnet and then the Internet, a broader population potentially had access to the data as it traversed increasingly longer paths from client to server. The opportunity for unauthorized third parties to eavesdrop on data transmissions increased proportionally.

In 1994, the internet browser company Netscape developed and released the application layer wrapper, Secure Sockets Layer.[2] This protocol enabled applications to communicate across a network in a private and secure fashion, discouraging eavesdropping, tampering, and message forgery. While it could add security to any protocol that uses reliable connections, such as TCP, it was most commonly used by Netscape with HTTP to form HTTPS.

The SSL protocol was eventually applied to FTP, with a draft Request for Comments (RFC) published in late 1996.[3] An official IANA port was registered shortly thereafter. However, the RFC was not finalized until 2005.[4]

2 Methods of invoking security

Two separate methods were developed to invoke client security for use with FTP clients: *Explicit* or *Implicit*. The explicit method is a legacy compatible implementation where FTPS-aware clients can invoke security with an FTPS-aware server without breaking overall FTP functionality with non-FTPS-aware clients. The implicit method requires that all clients of the FTPS server be aware that SSL is to be used on the session, and thus is incompatible with non-FTPS-aware clients.

2.1 Explicit

In explicit mode (also known as FTPES), an FTPS client must "explicitly request" security from an FTPS server and then step up to a mutually agreed encryption method. If a client does not request security, the FTPS server can either allow the client to continue in unsecure mode or refuse/limit the connection.

The mechanism for negotiating authentication and security with FTP was added under RFC 2228, which included the new FTP command AUTH. While this RFC does not explicitly define any required security mechanisms, e.g. SSL or TLS, it does require the FTPS client to challenge the FTPS server with a mutually known mechanism. If the FTPS client challenges the FTPS server with an unknown security mechanism, the FTPS server will respond to the AUTH command with error code *504 (not supported)*. Clients may determine which mechanisms are supported by querying the FTPS server with the FEAT command, although servers are not necessarily required to be honest in disclosing what levels of security they support. Common methods of invoking FTPS security included AUTH TLS and AUTH SSL.

In the later RFC 4217, FTPS compliance required that clients always negotiate using the AUTH TLS method. The RFC also recommended FTPS servers to accept the draft mechanism AUTH TLS-C.

2.2 Implicit

Negotiation is not allowed with implicit FTPS configurations. A client is immediately expected to challenge the FTPS server with a TLS/SSL *ClientHello* message. If such a message is not received by the FTPS server, the server should drop the connection.

In order to maintain compatibility with existing non-TLS/SSL-aware FTP clients, implicit FTPS was expected to listen on the IANA Well Known Port 990/TCP for the FTPS control channel, and to 989/TCP for the FTPS data channel. This allowed administrators to retain legacy-compatible services on the original 21/TCP FTP control channel.

Note that implicit negotiation was not defined in RFC 4217. As such, it is considered an earlier, deprecated method of negotiating TLS/SSL for FTP.

3 Transport Layer Security (TLS)/Secure Socket Layer (SSL)

Main article: Transport Layer Security

3.1 General support

FTPS includes full support for the TLS and SSL cryptographic protocols, including the use of server-side public key authentication certificates and client-side authorization certificates. It also supports compatible ciphers, including AES, RC4, RC2, Triple DES, and DES. It further supports hash functions SHA, MD5, MD4, and MD2.

3.2 Scope of use

In implicit mode, the entire FTPS session is encrypted. Explicit mode differs in that the client has full control over what areas of the connection are to be encrypted. Enabling and disabling of encryption for the FTPS control channel and FTPS data channel can occur at any time. The only restriction comes from the FTPS server, which has the ability to deny commands based on server encryption policy.

3.2.1 Secure command channel

The secure command channel mode can be entered through the issue of either the AUTH TLS or AUTH SSL commands. After such time, all command control between the FTPS client and server are assumed to be encrypted. It is generally advised to enter such a state prior to user authentication and authorization in order to avoid the eavesdropping of user name and password data by third parties.

3.2.2 Secure data channel

The secure data channel can be entered through the issue of the PROT command. It is *not* enabled by default when the AUTH TLS command is issued. After such time, all data channel communication between the FTPS client and server is assumed to be encrypted.

The FTPS client may exit the secure data channel mode at any time by issuing a CDC (clear data channel) command.

3.2.3 Reasons to disable encryption

It may not be advantageous to use data channel encryption when performing transfers under the following scenarios:

- Files being transferred are of a non-sensitive nature, making encryption unnecessary,
- Files being transferred are already encrypted at the file level or are passing over an encrypted VPN, making encryption redundant,
- Available TLS or SSL encryption modes do not meet desired level of encryption. This is common with older FTPS clients or servers that may have been limited to 40-bit SSL due to previous United States high-encryption export laws.

It may not be advantageous to use control channel encryption under the following scenarios:

- Use of FTPS when the client and/or server resides behind a network firewall or network address translation (NAT) device. (See Firewall Incompatibilities below.)
- Repeated use of AUTH and CCC/CDC commands by anonymous FTP clients within the same session. Such behavior can be utilized as a resource-based denial of service attack as the TLS/SSL session must be regenerated each time, utilizing server processor time.

3.2.4 SSL certificates

Much like HTTPS, FTPS servers must provide a public key certificate. These certificates can be requested and created using tools such as OpenSSL.

When these certificates are signed by a trusted certificate authority, this provides assurance that the client is connected to the requested server, avoiding a man-in-the-middle attack. If the certificate is not signed by a trusted CA (a self-signed certificate), the FTPS client may generate a warning stating that the certificate is not valid. The client can choose to accept the certificate or reject the connection.

This is in contrast to the SSH File Transfer Protocol (SFTP), which does not present signed certificates, but instead relies on Out-of-band authentication of public keys.

4 Firewall incompatibilities

Because FTP utilizes a dynamic secondary port (for data channels), many firewalls were designed to snoop FTP protocol control messages in order to determine which secondary data connections they need to allow. However, if the FTP control connection is encrypted using TLS/SSL, the firewall cannot determine the TCP port number of a data connection negotiated between the client and FTP server. Therefore, in many firewalled networks, an FTPS deployment will fail when an unencrypted FTP deployment will work. This problem can be solved with the use of a limited range of ports for data and configuring the firewall to open these ports.

5 See also

- List of file transfer protocols
- Comparison of FTP client software
- List of FTP server software
- Secure copy (SCP), a protocol for securely transferring files using the Secure Shell (SSH) protocol.
- FTP over SSH
- SSH File Transfer Protocol (SFTP)
- FISH
- SSH
- List of TCP and UDP port numbers

6 Notes

[1] RFC-265: File Transfer Protocol (FTP)

[2] The SSL Protocol, Feb. 9th, 1995

[3] RFC draft, Secure FTP Over SSL, revision 1996-11-26

[4] RFC-4217: Securing FTP with TLS

7 External links

- Overview of FTPS, and lists of clients, servers, and proxies supporting FTPS
- Curl-loader - an open-source FTPS loading/testing tool

8 Text and image sources, contributors, and licenses

8.1 Text

- **FTPS** *Source:* http://en.wikipedia.org/wiki/FTPS?oldid=627120628 *Contributors:* Zundark, Ebeisher, Ruakh, Robert Brockway, Rich Farmbrough, Sietse Snel, Cwolfsheep, Minghong, Wrs1864, Lightdarkness, Jopxton, Karnesky, Oliphaunt, Dovid, Jemiller226, Isaac Rabinovitch, Penguin, Typhoonhurricane, Martin Hinks, Bovineone, Brandon, Ospalh, JECompton, ClaesWallin, FF2010, SmackBot, Hux, AutumnSnow, Eskimbot, Thumperward, MalafayaBot, Josteinn, Metalim, Frap, Dinjiin, OranL, Paulfh, Amalas, MeekMark, Chrisahn, Thijs!bot, Epbr123, Druiloor, Isilanes, JAnDbot, Netzen, Bongwarrior, NegativeIQ, CommonsDelinker, Ronchristie, AlnoktaBOT, Mezzaluna, Wenli, Prolixium, Adaviel, VVVBot, Vulcan's Forge, BSoD, ClueBot, Anon lynx, Vivio Testarossa, Ykhwong, Feinoha, Addbot, Pietrow, Ettrig, JaBbA64, Tancee, Krinkle, Westquote, Ver-bot, MegaSloth, Lopifalko, EmausBot, Mjdtjm, ZéroBot, ClueBot NG, Encycloshave, Chmarkine, ArmbrustBot and Anonymous: 52

8.2 Images

- **File:Crypto_key.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/6/65/Crypto_key.svg *License:* CC-BY-SA-3.0 *Contributors:* Own work based on image:Key-crypto-sideways.png by MisterMatt originally from English Wikipedia *Original artist:* MesserWoland

8.3 Content license

- Creative Commons Attribution-Share Alike 3.0