Paul Ford-Hutchinson
IBM UK Ltd
Martin Carpenter
Verisign Ltd
Tim Hudson
RSA Australia Ltd
Eric Murray
Wave Systems Inc
Volker Wiegand
SuSE Linux

<draft-murray-auth-ftp-ssl-06.txt>

INTERNET-DRAFT (draft)

18th September, 2000

This document expires on 17th March, 2001

Securing FTP with TLS

Status of this Memo

Index

1. ..... Abstract
2. ..... Introduction
3. ..... Audience
4. ..... Session negotiation on the control port
5. ..... Response to FEAT command
6. ..... Data Connection Behaviour
7. ..... Mechanisms for the AUTH Command
8. ..... SASL Considerations
9. ..... Data Connection Security
10. ..... A discussion of negotiation behaviour
11. ..... Who negotiates what, where and how
12. ..... Timing Diagrams
13. ..... Implications of [FTP-EXT]
14. ..... Discussion of the 'REIN' command
15. ..... Security Considerations
16. ..... IANA Considerations
17. ..... Network Management
18. ..... Internationalization
19. ..... Scalability & Limits
20. ..... Applicability
21. ..... Acknowledgements
22. ..... References
23. ..... Authors' Contact Addresses

Appendices

A. ..... Summary of [RFC-2246]
B. ..... Summary of [RFC-2228]

1. Abstract

This document describes a mechanism that can be used by FTP clients and servers to implement security and authentication using the TLS protocol defined by [RFC-2246] and the extensions to the FTP protocol defined by [RFC-2228]. It describes the subset of the extensions that are required and the parameters to be used; discusses some of the policy issues that clients and servers will need to take; considers some of the implications of those policies and discusses some expected behaviours of implementations to allow interoperation. This document is intended to provide TLS support for FTP in a similar way to that provided for SMTP in [RFC-2487].

TLS is not the only mechanism for securing file transfer, however it does offer some of the following positive attributes:-

- Flexible security levels. TLS can support privacy, integrity, authentication or some combination of all of these. This allows clients and servers to dynamically, during a session, decide on the level of security required for a particular data transfer,

- It is possible to use X.509 certificates to authenticate client users and not just client hosts.

- Formalised public key management. By use of X.509 public certificates during the authentication phase, certificate management can be built into a central function. Whilst this may not be desirable for all uses of secured file transfer, it offers advantages in certain structured environments such as access to corporate data sources.

- Co-existence and interoperation with authentication mechanisms that are already in place for the HTTPS protocol. This allows web browsers to incorporate secure file transfer using the same infrastructure that has been set up to allow secure web browsing.

The TLS protocol is a development of the Netscape Communication Corporation's SSL protocol and this document can be used to allow the FTP protocol to be used with either SSL or TLS. The actual protocol used will be decided by the negotiation of the protected session by the TLS/SSL layer.

Note that this specification is in accordance with the FTP RFC [RFC-959] and relies on the TLS protocol [RFC-2246] and the FTP security extensions [RFC-2228].

2. Introduction

This document is an attempt to describe how three other documents should combined to provide a useful, interoperable, secure file transfer protocol. Those documents are:-

RFC 959 [RFC-959]

The description of the Internet File Transfer Protocol

RFC 2246 [RFC-2246]

The description of the Transport Layer Security protocol (developed from the Netscape Secure Sockets Layer (SSL) protocol version 3.0).

RFC 2228 [RFC-2228]

Extensions to the FTP protocol to allow negotiation of security mechanisms to allow authentication, privacy and message integrity.

The File Transfer Protocol (FTP) currently defined in [RFC-959] and in place on the Internet is an excellent mechanism for exchanging files. The security extensions to FTP in [RFC-2228] offer a comprehensive set of commands and responses that can be used to add authentication, integrity and privacy to the FTP protocol. The TLS protocol is a popular (due to its wholesale adoption in the HTTP environment) mechanism for generally securing a socket connection. There are many ways in which these three protocols can be combined which would ensure that interoperation is impossible. This document describes one method by which FTP can operate securely in such a way as to provide both flexibility and interoperation. This necessitates a brief description of the actual negotiation mechanism (if used); a much more detailed description of the policies and practices that would be required and a discussion of the expected behaviours of clients and servers to allow either party to impose their security requirements on the FTP session.

3. Audience

This document is aimed at developers who wish to use TLS as a security mechanism to secure FTP clients and/or servers.

4. Session negotiation on the control port

4.1 Negotiated Session Security

In this scenario, the server listens on the normal FTP control port {FTP-PORT} and the session initiation is not secured at all. Once the client wishes to secure the session, the AUTH command is sent and the server may then allow TLS negotiation to take place.

4.1.1 Client wants a secured session

If a client wishes to attempt to secure a session then it should, in accordance with [RFC-2228] send the AUTH command with the parameter requesting TLS or SSL {TLS-PARM}.

The client then needs to behave according to its policies depending on the response received from the server and also the result of the TLS negotiation. i.e. A client which receives an 'AUTH' rejection may choose to continue with the session unprotected if it so desires.

4.1.2 Server wants a secured session

The FTP protocol does not allow a server to directly dictate client behaviour, however the same effect can be achieved by refusing to accept certain FTP commands until the session is secured to an acceptable level to the server.

4.2 Implicit Session Security

In this scenario, the server listens on a distinct port {FTP-TLSPORT} to the normal unsecured FTP server. Upon connection, the client is expected to start the TLS negotiation. If the negotiation fails or succeeds at an unacceptable level of security then it will be a client and/or server policy decision to disconnect the session.

5. Response to the FEAT command

The FEAT command (introduced in [RFC-2389]) allows servers with additional features to advertise these to a client by responding to the FEAT command. If a server supports the 'FEAT' command then it MUST advertise supported 'AUTH', 'PBSZ' and 'PROT' commands in the reply as described in section 3.2 of [RFC-2389]. Additionally, the 'AUTH' command should have a reply that identifies 'TLS' as one of the possible parameters to 'AUTH'. It is not necessary to identify the 'SSL', 'TLS-P' or 'TLS-C' parameters separately.

Example reply (in same style is [RFC-2389])

```
C> FEAT
S> 211-Extensions supported
S>  AUTH TLS
S>  PBSZ
S>  PROT
S> 211 END
```

6. Data Connection Behaviour

The Data Connection in the FTP model can be used in one of three ways. (Note: these descriptions are not necessarily placed in exact chronological order, but do describe the steps required. - See diagrams later for clarification)

i) Classic FTP client/server data exchange

- The client obtains a port, sends the port number to the server, the server connects to the client. The client issues a send or receive request to the server on the control connection and the data transfer commences on the data connection.

ii) Firewall-Friendly client/server data exchange (as discussed in [RFC-1579]) using the PASV command to reverse the direction of the data connection.

- The client requests that the server open a port, the server obtains a port and returns the address and port number to the client. The client connects to the server on this port. The client issues a send or receive request on the control connection and the data transfer commences on the data connection.

iii) Client initiated server/server data exchange (proxy or PASV connections)

- The client requests that server A opens a port, server A obtains a port and returns it to the client. The client sends this port number to server B. Server B connects to server A. The client sends a send or receive request to server A and the complement to server B and the data transfer commences. In this model server A is the proxy or PASV host and is a client for the Data Connection to server B.

For i) and ii) the FTP client will be the TLS client and the FTP server will be the TLS server.

That is to say, it does not matter which side initiates the

connection with a connect() call or which side reacts to the connection via the accept() call, the FTP client as defined in [RFC-959] is always the TLS client as defined in [RFC-2246].

In scenario iii) there is a problem in that neither server A nor server B is the TLS client given the fact that an FTP server must act as a TLS server for Firewall-Friendly FTP [RFC-1579]. Thus this is explicitly excluded in the security extensions document [RFC-2228], and in this document.

7. Mechanisms for the AUTH Command

The AUTH command takes a single parameter to define the security mechanism to be negotiated. As the SSL/TLS protocols self-negotiate their levels there is no need to distinguish SSL vs TLS in the application layer. The proposed mechanism name for negotiating SSL/TLS will be the character string 'TLS'. This will allow the client and server to negotiate SSL or TLS on the control connection without altering the protection of the data channel. To protect the data channel as well, the PBSZ, PROT command sequence should be used. We call this "Explicit Data Channel Protection".

However, there are clients and servers that exist today which use the string 'SSL' to indicate that negotiation should take place on the control connection and that the data connection should be implicitly protected (i.e. the PBSZ 0, PROT P command sequence is not required but the client and server will protect the data channel as if it had). This is "Implicit Data Channel Protection" and is included primarily for backward compatibility.

To allow for streamlining of the negotiation, whilst allowing the 'SSL' string to sink peacefully into disuse, the strings 'TLS-P' and 'TLS-C' will also be defined. 'TLS-C' will be a synonym for 'TLS' and 'TLS-P' a synonym for 'SSL'. Thus we allow for strict compliance with [RFC-2228] by use of 'TLS' or 'TLS-C' and a quicker (2 less commands) and perhaps more sensible option 'TLS-P' which will implicitly secure the data connection at the same time as securing the control connection.

Note: Regardless of the manner in which the data connection is secured (either implicitly by use of 'TLS-P', 'SSL' or connection to a well-known port for FTP protocol over TLS, or explicitly by use of the PBSZ/PROT sequence) the data connection state may be modified by the client issuing the PROT command with the new desired level of data channel protection and the server replying in the affirmative. This data channel protection negotiation can happen at any point in

the session (even straight after a PORT or PASV command) and as often as is required.

See also Section 16, "IANA Considerations".

8. SASL Considerations

SASL is the Simple Authentication Security Layer. Currently, its definition can be found in the internet draft [RFC-2222]. This document attempts to define the means by which a connection-based protocol may identify and authenticate a client user to a server, with additional optional negotiation of protection for the remainder of that session.

Unfortunately, the SASL paradigm does not fit in neatly with the FTP-TLS protocol, mainly due to the fact that FTP uses two (independent) connections, and under FTP-TLS these may be at different (and possibly renegotiable) protection levels. Consequently, it is envisaged that SASL will sit underneath TLS on the control connection, and TLS (on both, either or neither connection) will be used for privacy and integrity (with optional authentication from TLS on either connection).

9. Data Connection Security

The Data Connection security level is determined by two factors.

1) The mechanism used to negotiate security on the control connection will dictate the default (i.e. un-negotiated) security level of the data port.

2) The PROT command, as specified in [RFC-2228] allows client/server negotiation of the security level of the data connection. Once a PROT command has been issued by the client and accepted by the server by returning the '200' reply, the security of subsequent data connections should be at that level until another PROT command is issued and accepted; the session ends; or the security of the session (via an AUTH command) is re-negotiated).

Data Connection Security Negotiation (the PROT command)

Note: In line with [RFC-2228], there is no facility for securing the Data connection with an insecure Control connection.

The command defined in [RFC-2228] to negotiate data connection

security is the PROT command. As defined there are four values that the PROT command parameter can take.

'C' - Clear - neither Integrity nor Privacy

'S' - Safe - Integrity without Privacy

'E' - Confidential - Privacy without Integrity

'P' - Private - Integrity and Privacy

As TLS negotiation encompasses (and exceeds) the Safe/Confidential/Private distinction, only Private (use TLS) and Clear (don't use TLS) are used.

For TLS, the data connection can have one of two security levels.

1) Clear

2)Private

With 'Clear' protection level, the data connection is made without TLS at all. Thus the connection is unauthenticated and has no privacy or integrity. This might be the desired behaviour for servers sending file lists, pre-encrypted data or non-sensitive data (e.g. for anonymous FTP servers).

If the data connection security level is 'Private' then a TLS negotiation must take place, to the satisfaction of the Client and Server prior to any data being transmitted over the connection. The TLS layers of the Client and Server will be responsible for negotiating the exact TLS Cipher Suites that will be used (and thus the eventual security of the connection).

In addition, the PBSZ (protection buffer size) command, as detailed in [RFC-2228], is compulsory prior to any PROT command. This document also defines a data channel encapsulation mechanism for protected data buffers. For FTP-TLS, which appears to the FTP application as a streaming protection mechanism, this is not required. Thus the PBSZ command must still be issued, but must have a parameter of '0' to indicate that no buffering is taking place and the data connection should not be encapsulated. Note that PBSZ 0 is not in the grammar of [RFC-2228], section 8.1, where it is stated:

PBSZ <sp> <decimal-integer> <CRLF> <decimal-integer> ::= any decimal integer from 1 to (2^32)-1

However it should be noted that using a value of '0' to mean a

streaming protocol is a reasonable use of '0' for that parameter and is not ambiguous.

Initial Data Connection Security

For backward compatibility and ease of implementation the following rules govern the initial expected protection setting of the data connection.

Connections accepted on the 'secure FTP' port (see {FTP-TLSPORT}).
The initial state of the data connection will be 'Private' (Although this does not follow [RFC-2228], this is how such clients tend to work today).

Connections accepted on the normal FTP port {FTP-PORT} with TLS/SSL negotiated via an 'AUTH SSL' command.
The initial state of the data connection will be 'Private' (Although this does not follow [RFC-2228], this is how such clients tend to work today).

Connections accepted on the normal FTP port {FTP-PORT} with TLS/SSL negotiated via an 'AUTH TLS' command.
The initial state of the data connection will be 'Clear' (this is the correct behaviour as indicated by [RFC-2228].)

Note: Connections made on other ports may be still behave in one of these ways, but that will be a local configuration issue.

10. A Discussion of Negotiation Behaviour

All these discussions assume that the negotiation has taken place by issuing the AUTH command with a mechanism that does not implicitly protect the data channel. Using a mechanism which does implicitly secure the data channel or connecting to a port which is implicitly protected will have similar issues.

10.1. The server's view of the control connection

A server may have a policy statement somewhere that might:

- Deny any command before TLS is negotiated (this might cause problems if a SITE or some such command is required prior to login)
- Deny certain commands before TLS is negotiated (such as USER, PASS or ACCT)
- Deny insecure USER commands for certain users (e.g. not

ftp/anonymous)
- Deny secure USER commands for certain users (e.g. ftp/anonymous)
- Define the level(s) of TLS/SSL to be allowed
- Define the CipherSuites allowed to be used (perhaps on a per host/domain/... basis)
- Allow TLS authentication as a substitute for local authentication.
- Define data connection policies (see next section)

Note: The TLS negotiation may not be completed satisfactorily for the server, in which case it can be one of these states.

The TLS negotiation failed completely

In this case, the control connection should still be up in unprotected mode and the server should issue an unprotected '421' reply to end the session.

The TLS negotiation completed successfully, but the server decides that the session parameters are not acceptable (e.g. Distinguished Name in the client certificate is not permitted to use the server)

In this case, the control connection should still be up in a protected state, so the server can either continue to refuse to service commands or issue a '421' reply and close the connection.

The TLS negotiation failed during the TLS handshake

In this case, the control connection is in an unknown state and the server should simply drop the control connection.

Server code will be responsible for implementing the required policies and ensuring that the client is prevented from circumventing the chosen security by refusing to service those commands that are against policy.

10.2. The server's view of the data connection

The server can take one of four basic views of the data connection

1 - Don't allow encryption at all (in which case the PROT command should not allow any value other than 'C' - if it is allowed at all)
2 - Allow the client to choose protection or not
3 - Insist on data protection (in which case the PROT command

must be issued prior to the first attempted data transfer)
4 - Decide on one of the above three for each and every data connection

The server should only check the status of the data protection level (for options 3 and 4 above) on the actual command that will initiate the data transfer (and not on the PORT or PASV). The following commands cause data connections to be opened and thus may be rejected (before any 1xx) message due to an incorrect PROT setting.

STOR
RETR
NLST
LIST
STOU
APPE
MLST (if [FTP-EXT] is implemented)
MLSD (if [FTP-EXT] is implemented)

The reply to indicate that the PROT setting is incorrect is
'521 data connection cannot be opened with this PROT setting'
If the protection level indicates that TLS is required, then it should be negotiated once the data connection is made. Thus, the '150' reply only states that the command can be used given the current PROT level. Should the server not like the TLS negotiation then it will close the data port immediately and follow the '150' command with a '522' reply indicating that the TLS negotiation failed or was unacceptable. (Note: this means that the application can pass a standard list of CipherSuites to the TLS layer for negotiation and review the one negotiated for applicability in each instance).

It is quite reasonable for the server to insist that the data connection uses a TLS cached session. This might be a cache of a previous data connection or of the control connection. If this is the reason for the the refusal to allow the data transfer then the '522' reply should indicate this.
Note: this has an important impact on client design, but allows servers to minimise the cycles used during TLS negotiation by refusing to perform a full negotiation with a previously authenticated client.

It should be noted that the TLS authentication of the server will be authentication of the server host itself and not a user on the server host.

10.3. The client's view of the control connection

In most cases it is likely that the client will be using TLS because the server would refuse to interact insecurely. To allow for this, clients must be able to be flexible enough to manage the securing of a session at the appropriate time and still allow the user/server policies to dictate exactly when in the session the security is negotiated.

In the case where it is the client that is insisting on the securing of the session, it will need to ensure that the negotiations are all completed satisfactorily and will need to be able to inform the user sensibly should the server not support, or be prepared to use, the required security levels.

Clients must be coded in such a manner as to allow the timing of the AUTH, PBSZ and PROT commands to be flexible and dictated by the server. It is quite reasonable for a server to refuse certain commands prior to these commands, similarly it is quite possible that a SITE or quoted command might be needed by a server prior to the AUTH. A client must allow a user to override the timing of these commands to suit a specific server.
For example, a client should not insist on sending the AUTH as the first command in a session, nor should it insist on issuing a PBSZ, PROT pair directly after the AUTH. This may well be the default behaviour, but must be overridable by a user.

Note: The TLS negotiation may not be completed satisfactorily for the client, in which case it will be in one of these states:

The TLS negotiation failed completely

In this case, the control connection should still be up in unprotected mode and the client should issue an unprotected QUIT command to end the session.

The TLS negotiation completed successfully, but the client decides that the session parameters are not acceptable (e.g. Distinguished Name in certificate is not the actual server expected)

In this case, the control connection should still be up in a protected state, so the client should issue a protected QUIT command to end the session.

The TLS negotiation failed during the TLS handshake

In this case, the control connection is in an unknown state

and the client should simply drop the control connection.

10.4. The client's view of the data connection

Client security policies

Clients do not typically have 'policies' as such, instead they rely on the user defining their actions and, to a certain extent, are reactive to the server policy. Thus a client will need to have commands that will allow the user to switch the protection level of the data connection dynamically, however, there may be a general 'policy' that attempts all LIST and NLST commands on a Clear connection first (and automatically switches to Private if it fails). In this case there would need to be a user command available to ensure that a given data transfer was not attempted on an insecure data connection.

Clients also need to understand that the level of the PROT setting is only checked for a particular data transfer after that transfer has been requested. Thus a refusal by the server to accept a particular data transfer should not be read by the client as a refusal to accept that data protection level in toto, as not only may other data transfers be acceptable at that protection level, but it is entirely possible that the same transfer may be accepted at the same protection level at a later point in the session.

It should be noted that the TLS authentication of the client should be authentication of a user on the client host and not the client host itself.

11. Who negotiates what, where and how

11.1. Do we protect at all ?

Client issues AUTH <Mechanism>, server accepts or rejects.
If server needs AUTH, then it refuses to accept certain commands until it gets a successfully protected session.

11.2. What level of protection do we use on the Control connection ?

Decided entirely by the TLS CipherSuite negotiation.

11.3. Do we protect data connections in general ?

Client issues PROT command, server accepts or rejects.

11.4. Is protection required for a particular data transfer ?

A client would already have issued a PROT command if it required the connection to be protected.
If a server needs to have the connection protected then it will reply to the STOR/RETR/NLST/... command with a '522' indicating that the current state of the data connection protection level is not sufficient for that data transfer at that time.

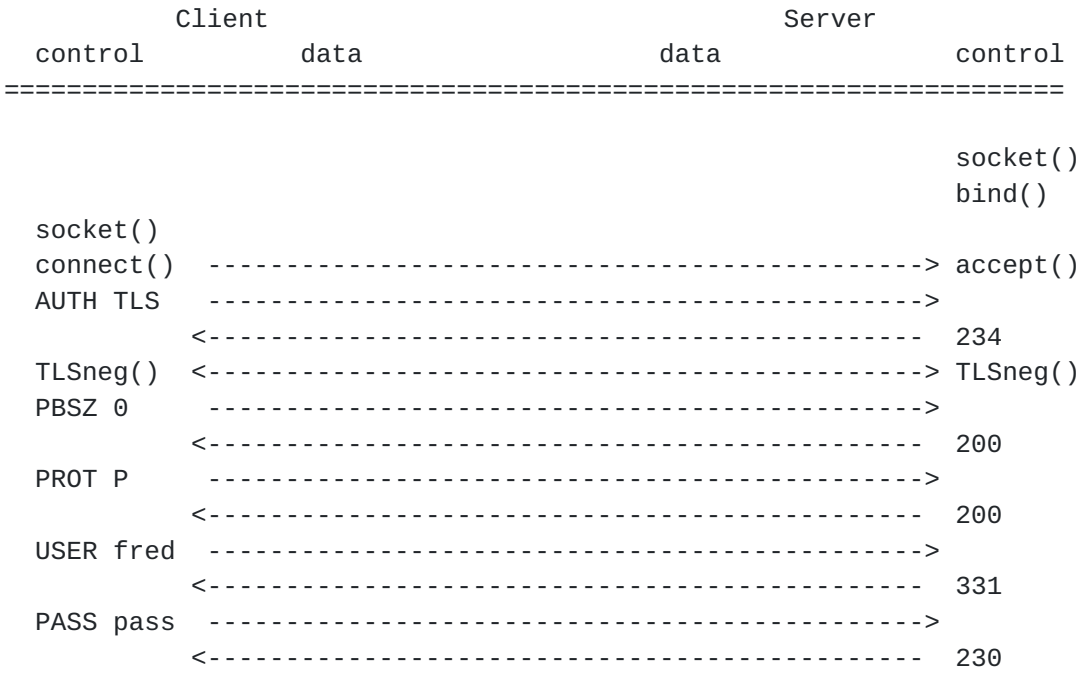
11.5. What level of protection is required for a particular data transfer ?

Decided entirely by the TLS CipherSuite negotiation.

Thus it can be seen that, for flexibility, it is desirable for the FTP application to be able to interact with the TLS layer upon which it sits to define and discover the exact TLS CipherSuites that are to be/have been negotiated and make decisions accordingly. However it should be entirely possible, using the mechanisms described in this document, to have a TLS client or server sitting on top of a generic 'TLS socket layer'. In this case, interoperability for a client with a smart TLS-aware server may not be possible due to server policies.

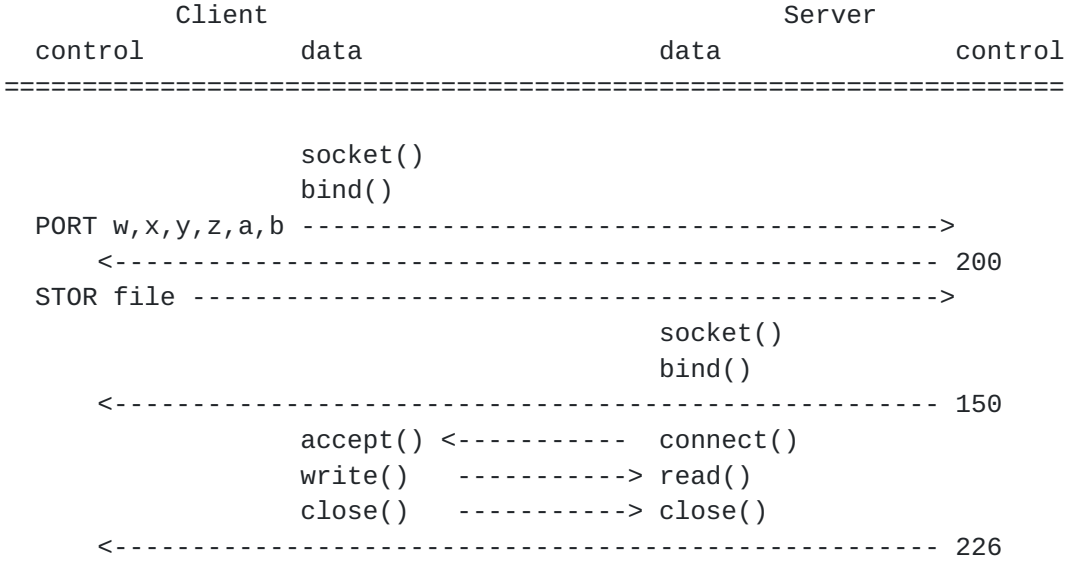
12. Timing Diagrams

12.1. Establishing a protected session

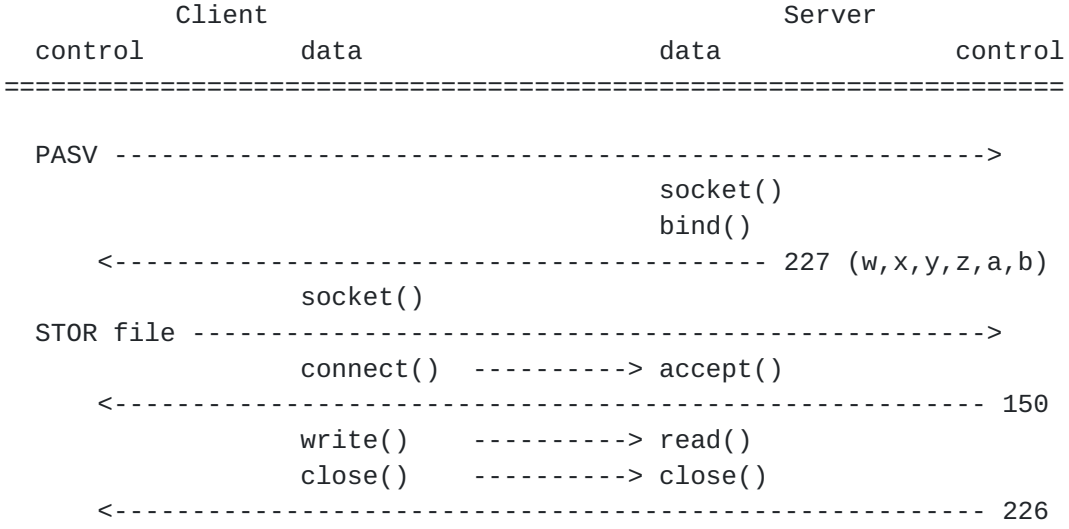


Note: the order of the PBSZ/PROT pair and the USER/PASS pair (with respect to each other) is not important (i.e. the USER/PASS can happen prior to the PBSZ/PROT - or indeed the server can refuse to allow a PBSZ/PROT pair until the USER/PASS pair has happened).

12.2. A standard data transfer without protection.

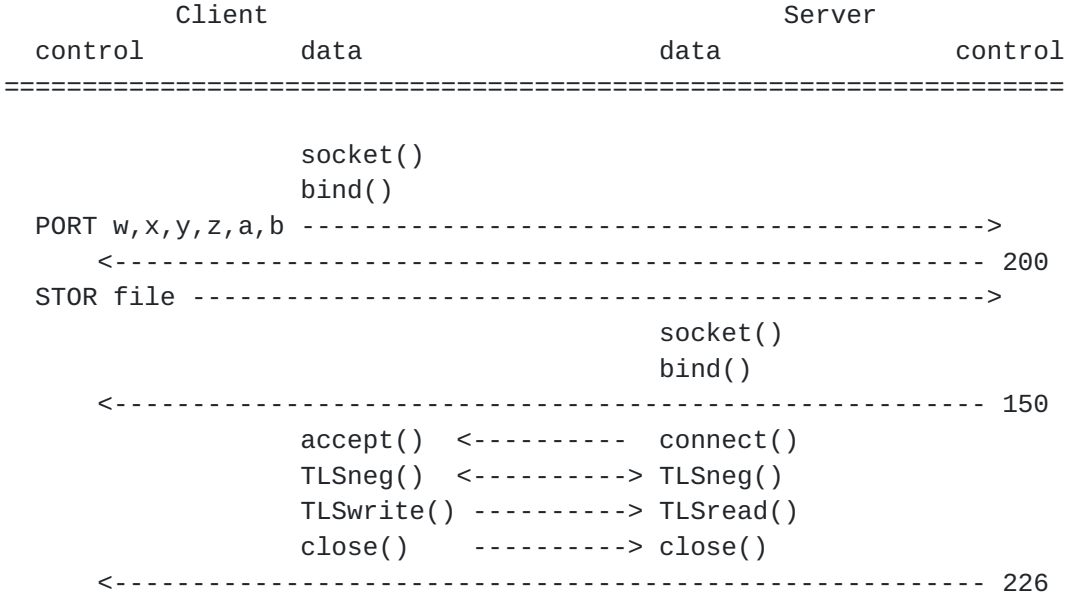


12.3. A firewall-friendly data transfer without protection

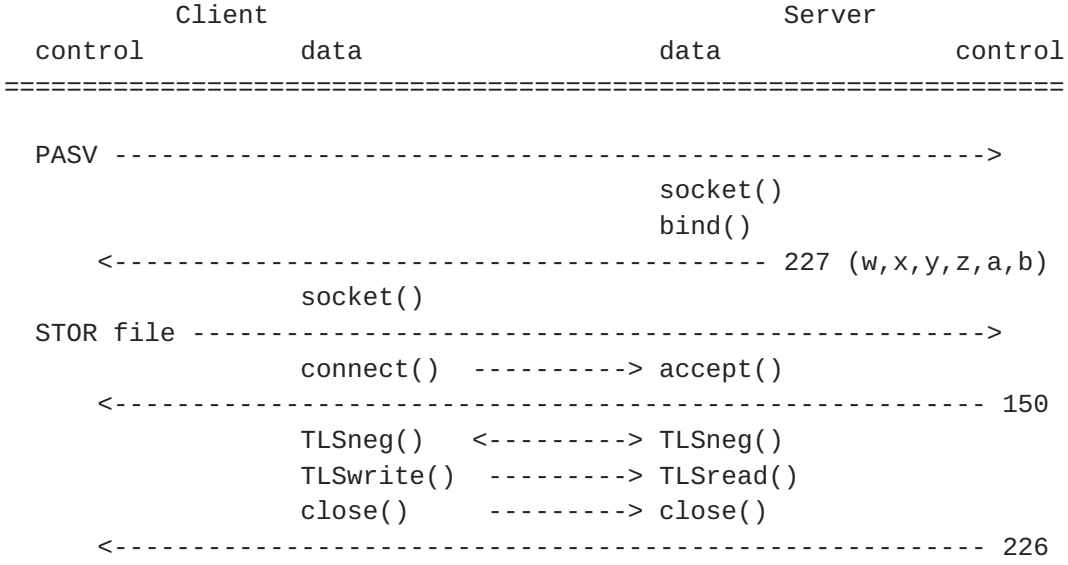


Note: Implementors should be aware that then connect()/accept() function is performed prior to the receipt of the reply from the STOR command. This contrasts with situation when (non-firewall-friendly) PORT is used prior to the STOR, and the accept()/connect() is performed after the reply from the aforementioned STOR has been dealt with.

12.4. A standard data transfer with protection



12.5. A firewall-friendly data transfer with protection



13. Implications of [FTP-EXT]

13.1. MLST and MLSD

MLST and MLSD are directory listing commands and should be treated in the same manner as NLST and LIST for the purposes of this document.

14. Discussion of the 'REIN' command

The 'REIN' command, defined in [RFC-959], allows the user to reset the state of the FTP session.

REINITIALIZE (REIN)

This command terminates a USER, flushing all I/O and account information, except to allow any transfer in progress to be completed. All parameters are reset to the default settings and the control connection is left open. This is identical to the state in which a user finds himself immediately after the control connection is opened. A USER command may be expected to follow.

The defined behaviour for TLS protected FTP sessons will depend on the manner of session initialisation.

If the session has been explicity protected (see section 4.1) then the TLS session(s) will be cleared and the control and data connections revert to unprotected, clear communications. It will be acceptable to use cached TLS sessions for subsequent connections, however a server should not mandate this.

If the session is implicitly protected (see section 4.2) then the control connection will continue to be protected using the exisiting negotiated TLS session and the data connection will revert to being implicitly protected, irrespective of any 'PROT' commands preceding the 'REIN'.

15. Security Considerations

This entire document deals with security considerations related to the File Transfer Protocol.

15.1. Verification of Authentication tokens

15.1.1. Server Certificates

Although it is entirely an implementation decision, it is recommended that certificates used for server authentication of the TLS session contain the server identification information in a similar manner to those used for http servers. (i.e. SubjectCommonName or SubjectAltName of type dNSName).

Note that, if there is any future extensions to the FTP protocol to allow multi-homed servers, then the interaction of such a mechanism, the REIN commands and the certificate presented by the server in the TLS handshake will need to be considered carefully.

15.1.2. Client Certificates

- Deciding which client certificates to allow and defining which fields define what authentication information is entirely a server implementation issue.

- It is also server implementation issue to decide if the authentication token presented for the data connection must match the one used for the corresponding control connection.

15.2. Addressing FTP Security Considerations [RFC-2577]

15.2.1. Bounce Attack

A bounce attack should be harder in a secured FTP environment because:

- The FTP server that is being used to initiate a false connection will always be a 'server' in the TLS context. Therefore, only services that act as 'clients' in the TLS context could be vulnerable. This would be a counter-intuitive way to implement TLS on a service.

- The FTP server would detect that the authentication credentials for the data connection are not the same as those for the control connection, thus the server policies could be set to drop the data connection.

- Genuine users are less likely to initiate such attacks when the authentication is strong and malicious users are less likely to gain access to the FTP server if the authentication is not easily subverted (password guessing, network tracing, etc...)

15.2.2. Restricting Access

This document presents a strong mechanism for solving the issue raised in this section.

15.2.3. Protecting Passwords

The twin solutions of strong authentication and data confidentiality ensure that this is not an issue when TLS is used to protect the control session.

15.2.4. Privacy

The TLS protocol ensures data confidentiality by encryption. Privacy (e.g. access to download logs, user profile information, etc...) is outside the scope of this document (and [RFC-2577] presumably)

15.2.5. Protecting Usernames

This is not an issue when TLS is used as the primary authentication mechanism.

15.2.6. Port Stealing

This proposal will do little for the Denial of Service element of this section, however, strong authentication on the data connection will prevent unauthorised connections retrieving or submitting files.

15.2.7. Software-Base Security Problems

Nothing in this proposal will affect the discussion in this section.

16. IANA Considerations

{FTP-PORT} - The port assigned to the FTP control connection is 21.

{FTP-TLSPORT} - A port to be assigned by the IANA for native TLS FTP connections on the control socket. This has been provisionally reserved as port 990.

{TLS-PARM} - A parameter for the AUTH command to indicate that TLS is required. It is recommended that 'TLS', 'TLS-C', 'SSL' and 'TLS-P' are acceptable, and mean the following :-

'TLS' or 'TLS-C' - the TLS protocol or the SSL protocol will be negotiated on the control connection. The default protection setting for the Data connection is 'Clear'.

'SSL' or 'TLS-P' - the TLS protocol or the SSL protocol will be negotiated on the control connection. The default protection setting for the Data connection is 'Private'. This is primarily for backward compatibility.

Note - [RFC-2228] states that these parameters are case-insensitive.

17. Network Management

NONE

18. Internationalization

NONE

19. Scalability & Limits

There are no issues other than those concerned with the ability of the server to refuse to have a complete TLS negotiation for each and every data connection, which will allow servers to retain throughput whilst using cycles only when necessary.

20. Applicability

This mechanism is generally applicable as a mechanism for securing the FTP protocol. It is unlikely that anonymous FTP clients or servers will require such security (although some might like the authentication features without the privacy).

21. Acknowledgements

o Netscape Communications Corporation for the original SSL protocol.

o Eric Young for the SSLeay libraries.

o University of California, Berkley for the original implementations of FTP and ftpd on which the initial implementation of these extensions were layered.

o IETF CAT working group.

o IETF TLS working group.

o IETF FTPEXT working group.

22. References

[RFC-959] J. Postel, "File Transfer Protocol"
RFC 959, October 1985.

[RFC-1579] Bellovin, S., "Firewall-Friendly FTP"
RFC 1579, February 1994.

[RFC-2222] J. Myers, "Simple Authentication and Security Layer"
RFC 2222, October 1997.

[RFC-2228] M. Horowitz, S. Lunt, "FTP Security Extensions"
RFC 2228, October 1997.

[RFC-2246] T. Dierks, C. Allen, "The TLS Protocol Version 1.0"
RFC 2246, January 1999.

[RFC-2389] P Hethmon, R.Elz, "Feature Negotiation Mechanism for the File Transfer Protocol"
RFC 2389, August 1998.

[RFC-2487] P Hoffman, "SMTP Service Extension for Secure SMTP over TLS"
RFC 2487, January 1999.

[RFC-2577] M Allman, S Ostermann "FTP Security Considerations"
RFC 2577, May 1999.

[FTP-EXT] R Elz, P Hethmon "Extensions to FTP"
draft-ietf-ftpext-mlst-07.txt, June 1999.

[SRA-FTP] "SRA - Secure RPC Authentication for TELNET and FTP Version 1.1"
file://ftp.funet.fi/security/login/telnet/doc/sra/sra.README

23. Authors' Contact Addresses

Please send comments to Paul Ford-Hutchinson at the address below

Tim Hudson
RSA Data Security
Australia Pty Ltd

tel - +61 7 3227 4444
fax - +61 7 3227 4400
email - tjh@rsasecurity.com.au

Paul Ford-Hutchinson
IBM UK Ltd
PO Box 31
Birmingham Road
Warwick
United Kingdom

+44 1926 462005
+44 1926 496482
paulfordh@uk.ibm.com

Martin Carpenter
Verisign Ltd
email - mcarpenter@verisign.com

Eric Murray
Wave Systems Inc.
ericm@lne.com

Volker Wiegand
SuSE Linux
email - wiegand@suse.de

Appendices

A. Summary of [RFC-2246]

The TLS protocol was developed by the IETF TLS working group. It is based on the SSL protocol proposed by Netscape Communications Corporation. The structure of the start of a TLS session allows negotiation of the level of the protocol to be used - in this way, a client or server can simultaneously support TLS and SSL and negotiate the most appropriate for the connection.

The TLS protocol defines three security mechanisms that may be used (almost) independently. They are Authentication, Integrity and Privacy. It is possible to have an Authenticated session with no Privacy and with or without Integrity (useful for anonymous FTP sites, or sites with pre-encrypted data). For example, sessions with Authentication, Privacy and Integrity would be useful for control connections over an insecure network and data connections transferring confidential material.

The TLS protocol allows unauthenticated sessions; server authentication or client and server authentication. There is no mechanism for authenticating a client without first authenticating the server.

The basic mechanism of the TLS protocol is that (for an Authenticated, Private session) asymmetric encryption is used to authenticate clients and servers and exchange a session key for symmetric encryption which is to be used for the rest of the session.

The structure of the TLS session initialisation is that the client initiates the session with a 'ClientHello' message. The server will respond with a 'ServerHello' and the session negotiation will continue.

The TLS protocol allows session caching which is achieved by the client requesting that the server re-use a session context (Cipher Suite and symmetric key) in the ClientHello message. There is no reason why a second connection could not request a 'cached' session with the same context as an existing session.

B. Summary of [RFC-2228]

Extensions to FTP

The FTP Security Extensions document has 8 new commands to enhance the FTP protocol to allow negotiation of security and exchange of security data. Three of these commands (the AUTH, PBSZ and PROT commands) are used by this document to allow an FTP client to negotiate TLS with the server. The other commands are not required.

i) AUTH

This command is a request by the client to use an authentication and/or security mechanism.

The client will issue an 'AUTH <Mechanism>' command which will be a request to the server to secure the control connection using the TLS (or SSL) protocol. It also governs the initial protection setting of the data channel (which may be changed by a subsequent PROT command).

ii) ADAT

This command is used to transmit security data required by the security mechanism agreed in a preceding AUTH command. This document does not use the ADAT command.

iii) PROT

This command is used by the client to instruct the type of security that is required on the Data connection.

The 'PROT C' command will mean that TLS should not be used to secure the data connection; 'PROT P' means that TLS should be used. 'PROT E' and 'PROT S' are not defined and generate a '536' reply from the server.

iv) PBSZ

This command is used to negotiate the size of the buffer to be used during secured data transfer.

The PBSZ command must be issued prior to the PROT command. The PBSZ command cannot be sent on an insecure control connection. For FTP and TLS the only valid value for the parameter is '0', all other values should receive a '200' reply with the text 'PBSZ=0'

included.

v) CCC

This command is used to specify that the control channel no longer requires protection.
This document does not use the CCC command.

vi) MIC

This command is used to send a normal FTP command with integrity protection.
This document does not use the MIC command.

vii) CONF

This command is used to send a normal FTP command with confidentiality protection (encrypted).
This document does not use the CONF command.

viii) ENC

This command is used to send a normal FTP command with confidentiality and integrity protection.
This document does not use the ENC command.

Full Copyright Statement

This document expires on 17th March, 2001