Paul Ford-Hutchinson
IBM UK Ltd

<draft-murray-auth-ftp-ssl-14.txt>

INTERNET-DRAFT (draft)

16th June, 2004

This document expires on 16th December, 2004

Securing FTP with TLS

Status of this Memo

Index

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 which 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] and HTTP in [RFC-2817].

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 confidentiality, 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 TLS identities to authenticate client users and not just client hosts.

- Formalised public key management. By use of well established client identity mechanisms (supported by TLS) during the authentication phase, certificate management may 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.

- 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. This document will only refer to the TLS protocol, however, it is understood that the Client and Server MAY actually be using SSL if they are so configured.

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

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY" and "OPTIONAL" that appear in this document are to be interpreted as described in [RFC-2119].

This document describes how three other documents should be 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, confidentiality 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 confidentiality 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 ; 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 implement TLS as a security mechanism to secure FTP clients and/or servers.

Systems administrators and architects should be fully aware of the security implications discussed in [RFC-2228] which need to be considered when choosing an implementation of this protocol and configuring it to provide their required security.

4. Overview

A full description of the FTP security protocol enhancements is contained in [RFC-2228]. This document describes how the AUTH, PROT, PBSZ and CCC commands, defined therein, should be implemented with the TLS protocol.

In summary; an FTP session is established on the normal control port. A client requests TLS with the AUTH command and then decides if it wishes to secure the data connections by use of the PBSZ:PROT commands. Should a client wish to make the control connection revert back into plaintext (once the authentication phase is completed, for example) then the CCC command can be used.

Implementation of this protocol extension does not ensure that each and every session and data transfer is secure, it merely provides the tools to allow a client and/or server to negotiate an acceptable or required level of security for that given session or data transfer. However, it is possible to have a server implementation that is capable of refusing to operate in an insecure fashion.

5. Session negotiation on the control port

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.

5.1 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 {TLS-PARM} ('TLS').

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.

5.2 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.

The server response to an 'AUTH TLS' command which it will honour, is '234'.

Note. The '334' response as defined in [RFC-2228] implies that an ADAT exchange will follow. This document does not use the ADAT command and so the '334' reply is incorrect.

Note. The FTP protocol insists that a USER command be used to identify the entity attempting to use the ftp server. Although the TLS negotiation may be providing authentication information the USER command must still be issued by the client. However, it will be a server implementation issue to decide which credentials to accept and what consistency checks to make between any client cert used and the parameter on the USER command.

6. Clearing the control port

There are circumstances where it may be desirable to protect the control connection only during part of the session and then revert back to a plaintext connection. (This is often due to the limitations of boundary devices such as NAT and firewalls which expect to be able to examine the content of the control connection in order to modify their behaviour.)

Typically the AUTH, USER, PASS, PBSZ and PROT commands would be protected within the TLS protocol and then the CCC command would be issued to return to a plaintext socket state. This does have important Security Issues (which are discussed in the Security Considerations section) but this document does describe how the command should be used, should the client and server, having considered the issues, still wish to use it.

When a server receives the CCC command, it should behave as follows:

If the server does not accept CCC commands (or does not understand them) then a 500 reply should be sent.

Otherwise, if the control connection is not protected with TLS, then a 533 reply should be sent.

Otherwise, if the server does not wish to allow the control connection to be cleared at this time, then a 534 reply should be sent.

Otherwise, the server is accepting the CCC command and should do the following:

- o send a 200 reply
- o Shutdown the TLS session on the socket and leave it open
- o Continue the control connection in plaintext, expecting the next command from the client to be in plaintext.
- o Not accept any more PBSZ or PROT commands. All subsequent data transfers must be protected with the current PROT settings.

7. 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 'TLS-C' synonym separately.

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

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

8. 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 MUST be the TLS client and the FTP server MUST 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.

9. 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 TLS will be the character string identified in {TLS-PARM}. This will allow the client and server to negotiate 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 MUST be used.

Note: 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 15, "IANA Considerations".

10. Data Connection Security

The Data Connection security level is determined by the PROT command

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 returning the '200' reply, the security of subsequent data connections MUST be at that level until another PROT command is issued and accepted; the session ends; a REIN command is issued; 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. Specifically, the PROT command MUST be preceded by a PBSZ command and a PBSZ command MUST be preceded by a successful security data exchange (the TLS negotiation in this case)

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 (requested by 'PROT C')

2)Private (requested by 'PROT P')

With 'Clear' protection level, the data connection is made without TLS at all. Thus the connection is unauthenticated and has no confidentiality 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 on the data connection, 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

The initial state of the data connection MUST be 'Clear' (this is the behaviour as indicated by [RFC-2228].)

11. A Discussion of Negotiation Behaviour

As [RFC-2228] allows security qualities to be negotiated, enabled and disabled dynamically, this can make implementations seem quite complex. However, in any given instance the behaviour should be quite straightforward. Either the server will be enforcing the policy of the server host or it will be providing security capabilities requested by the client. Either the client will be conforming to the server's policy or will be endeavouring to provide the capabilities which the user desires.

11.1. 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 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)

It is possible that 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 MAY either continue to refuse to service commands or issue a protected '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 which are against policy.

11.2. 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, defined in [RFC-959] 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
```

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).

Section 15.1.1 discusses the issue of cross-checking a certificate used to authenticate the data connection with the one used to authenticate the control connection. This is an important security step.
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 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.

11.3. 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 SHOULD 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 SHOULD 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.

11.4. 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.

12. Who negotiates what, where and how

12.1. Do we protect at all ?

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

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

Decided entirely by the TLS CipherSuite negotiation.

12.3. Do we protect data connections in general ?

Client issues PROT command, server accepts or rejects.

12.4. 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.

12.5. 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 which are to be/have been negotiated and make decisions accordingly.

13. Timing Diagrams

These timing diagrams aim to help explain exactly how the TLS handshake and session protection fits into the existing logic of the FTP protocol. Of course, the FTP protocol itself is not well described with respect to timing of commands and responses in [RFC-959], so this is partly based on empirical observation of existing widespread client and server implementations.

13.1. Establishing a protected session

```
           Client                                 Server
  control          data                  data              control
====================================================================

                                                           socket()
                                                           bind()
  socket()
  connect()  ----------------------------------------------> accept()
            <----------------------------------------------  220
  AUTH TLS   ---------------------------------------------->
            <----------------------------------------------  234
  TLSneg()  <----------------------------------------------> TLSneg()
  PBSZ 0     ---------------------------------------------->
            <----------------------------------------------  200
  PROT P     ---------------------------------------------->
            <----------------------------------------------  200
  USER fred  ---------------------------------------------->
            <----------------------------------------------  331
  PASS pass  ---------------------------------------------->
            <----------------------------------------------  230
```

Note 1: 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).

Note 2: the PASS command might not be required at all (if the USER parameter and any client identity presented provide sufficient authentication). The server would indicate this by issuing a '232' reply to the USER command instead of the '331' which requests a PASS from the client. (see below)

Note 3: the AUTH command might not be the first command after the receipt of the 220 welcome message.

   13.2. Establishing a protected session without a password request
   (the TLS authentication is sufficient)

```
           Client                                      Server
  control          data                    data              control
====================================================================

                                                             socket()
                                                             bind()
  socket()
  connect()  ----------------------------------------------> accept()
            <----------------------------------------------  220
  AUTH TLS   ---------------------------------------------->
            <----------------------------------------------  234
  TLSneg()  <----------------------------------------------> TLSneg()
  PBSZ 0     ---------------------------------------------->
            <----------------------------------------------  200
  PROT P     ---------------------------------------------->
            <----------------------------------------------  200
  USER fred  ---------------------------------------------->
            <----------------------------------------------  232
```

13.3. Establishing a protected session and then clearing with the CCC command

```
           Client                                 Server
  control          data                    data              control
====================================================================

                                                             socket()
                                                             bind()
  socket()
  connect()  ----------------------------------------------> accept()
            <----------------------------------------------  220
  AUTH TLS   ---------------------------------------------->
            <----------------------------------------------  234
  TLSneg()  <----------------------------------------------> TLSneg()
  PBSZ 0     ---------------------------------------------->
            <----------------------------------------------  200
  PROT P     ---------------------------------------------->
            <----------------------------------------------  200
  USER fred  ---------------------------------------------->
            <----------------------------------------------  232
  CCC        ---------------------------------------------->
            <----------------------------------------------  200
  TLSshutdown()  <-------------------------------------> TLSshutdown()
```

- rest of control session continues in plaintext with protected data transfers (due to PROT P)

Note: this has serious security issues (see Security Considerations section) but may be useful in a firewall/NAT scenario.

```
   13.4. A standard data transfer without protection.

           Client                                 Server
  control          data                   data              control
====================================================================

                   socket()
                   bind()
  PORT w,x,y,z,a,b ----------------------------------------->
      <----------------------------------------------------- 200
  STOR file ------------------------------------------------>
                                          socket()
                                          bind()
      <----------------------------------------------------- 150
                   accept() <-----------  connect()
                   write()   -----------> read()
                   close()   -----------> close()
      <----------------------------------------------------- 226
```

13.5. A firewall-friendly data transfer without protection

```
           Client                                 Server
  control          data                   data              control
====================================================================

  PASV -------------------------------------------------------->
                                          socket()
                                          bind()
      <------------------------------------------ 227 (w,x,y,z,a,b)
                   socket()
  STOR file --------------------------------------------------->
                   connect()  ----------> accept()
      <-------------------------------------------------------- 150
                   write()    ----------> read()
                   close()    ----------> close()
      <-------------------------------------------------------- 226
```

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.

13.6. A standard data transfer with protection

```
           Client                                 Server
  control          data                  data              control
====================================================================

                   socket()
                   bind()
  PORT w,x,y,z,a,b ----------------------------------------->
      <----------------------------------------------------- 200
  STOR file ------------------------------------------------>
                                         socket()
                                         bind()
      <----------------------------------------------------- 150
                   accept()  <----------  connect()
                   TLSneg()  <----------> TLSneg()
                   TLSwrite() ----------> TLSread()
                   TLSshutdown() -------> TLSshutdown()
                   close()    ----------> close()
      <----------------------------------------------------- 226
```

13.7. A firewall-friendly data transfer with protection

```
           Client                                 Server
  control          data                   data              control
====================================================================

  PASV -------------------------------------------------------->
                                          socket()
                                          bind()
      <------------------------------------------- 227 (w,x,y,z,a,b)
                   socket()
  STOR file --------------------------------------------------->
                   connect()  ----------> accept()
      <-------------------------------------------------------- 150
                   TLSneg()   <---------> TLSneg()
                   TLSwrite()  ---------> TLSread()
                   TLSshutdown() -------> TLSshutdown()
                   close()     ---------> close()
      <-------------------------------------------------------- 226
```

14. Discussion of the REIN command

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

```
   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.
```

When this command is processed by the server, the TLS session(s) MUST be cleared and the control and data connections revert to unprotected, clear communications. It MAY be acceptable to use cached TLS sessions for subsequent connections, however a server MUST NOT mandate this.

15. Discussion of the STAT and ABOR commands

The ABOR and STAT commands and the use of TCP Urgent Pointers

[RFC-959] describes the use of Telnet commands (IP and DM) and the TCP Urgent pointer to indicate the transmission of commands on the control channel during the execution of a data transfer. FTP uses the Telnet Interrupt Process and Data Mark commands in conjunction with Urgent data to preface two commands: ABOR (Abort Transfer) and STAT (Status request).

The Urgent Pointer was used because in a Unix implementation the receipt of a TCP packet marked as Urgent would result in the execution of the SIGURG interrupt handler. This reliance on interrupt handlers was necessary on systems which did not implement select() or did not support multiple threads. TLS does not support the notion of Urgent data.

When TLS is implemented as a security method in FTP the server SHOULD NOT rely on the use of SIGURG to process input on the control channel during data transfers. The client MUST send all data including Telnet commands across the TLS session. The TLS session will be corrupted if any data is sent on a socket while TLS is active.

16. Security Considerations

This document discusses how TLS may be used in conjunction with [RFC-2228] to provide mechanisms for securing FTP sessions. Discussions about security rationale and security properties are contained within the [RFC-2228] document and are not repeated here.

16.1. Verification of Authentication tokens

16.1.1. 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. (see [RFC-2818])

Similarly, it is recommended that the certificate used for server authentication of Data connections is the same certificate as that used for the corresponding Control connection.

16.1.2. 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.

16.2. Addressing FTP Security Considerations [RFC-2577]

16.2.1. 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...)

16.2.2. Restricting Access

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

16.2.3. 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.

16.2.4. 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)

16.2.5. Protecting Usernames

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

16.2.6. 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.

16.2.7. Software-Base Security Problems

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

16.3. Issues with the CCC command

Using the CCC command can create security issues. For a full description, see the "CLEAR COMMAND CHANNEL (CCC)" section of

[RFC-2228]. Clients should not assume that a server will allow the CCC command to be processed.

17. IANA Considerations

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

18. Other Parameters

{TLS-PARM} - The parameter for the AUTH command to indicate that TLS is required. To request the TLS protocol in accordance with this document, the client MUST use 'TLS'

To maintain backward compatibility with older versions of this document, the server SHOULD accept 'TLS-C' as a synonym for 'TLS'

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

19. Network Management

NONE

20. Internationalization

NONE

21. 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.

22. 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 confidentiality).

23. 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.

o Jeff Altman for the ABOR and STAT discussion.

o The various people who have help author this document throughout its protracted draft stages, namely Martin Carpenter, Eric Murray, Tim Hudson and Volker Wiegand.

24. References

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

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

[RFC-2119] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels"
RFC 2119, March 1997.

[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.

[RFC-2817] R. Khare, S. Lawrence, "Upgrading to TLS Within HTTP/1.1"
RFC 2817, May 2000.

[RFC-2818] E. Rescorla, "HTTP Over TLS"
RFC 2818, May 2000.

25. Author's Contact Address

The FTP-TLS draft information site is at http://www.ford-hutchinson.com/~fh-1-pfh/ftps-ext.html

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

```
        Paul Ford-Hutchinson
           IBM UK Ltd
           PO Box 31
           Birmingham Road
           Warwick
           United Kingdom
  tel -   +44 1926 462005
email - paulfordh@uk.ibm.com
```

26.1. Additional Authors' Contact Addresses

```
        Tim Hudson                   Volker Wiegand
           RSA Data Security            SuSE Linux
             Australia Pty Ltd
  tel -   +61 7 3227 4444
email - tjh@rsasecurity.com.au    wiegand@suse.de

        Martin Carpenter             Eric Murray
           Verisign Ltd                 Wave Systems Inc.
email -  mcarpenter@verisign.com    ericm@lne.com
```

This document expires on 16th December, 2004