Paul Ford-Hutchinson
IBM UK Ltd
Tim Hudson
CryptSoft Pty Ltd
Eric Murray
Independent Security Consultant
24th March, 1997

<draft-murray-auth-ftp-ssl-01.txt>

INTERNET-DRAFT

This document expires on 24th September, 1997

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 the IETF TLS working group and the extensions to
   the ftp protocol defined by the IETF CAT working group. 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; describes some of the implications of those
   policies and discusses some expected behaviours of implementations to
   allow interoperation.

   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 them. This allows clients
      and servers to dynamically, during a session, decide on the level
      of security required for a particular data transfer,

      - 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 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 and
   relies on the TLS protocol and the CAT ftp security extensions.

[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

draft-ietf-tls-protocol-01.txt [TLS-DESC]

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

draft-ietf-cat-ftpsec-09.txt [CAT-FTPSEC]

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 [CAT-FTPSEC] 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 way in 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 [CAT-FTPSEC] 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.

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. 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.)

  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 [FTP-SOCKS]) using the PASV command to reverse the
  deriection of the data connection.

  - The client requests that the server open a port, the server

obtains a port and returns it 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] will always be the TLS client as defined in [TLS-DESC].

In scenario iii) there will be a problem in that neither server A nor
server B will be the TLS client given the fact that an FTP server
must act as a TLS server for Firewall Friendly FTP [FTP-SOCKS]. This
is explicitly excluded in the security extensions document [CAT-
FTPSEC] and this document.

6. 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 at the
application level. 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.
This is "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 C 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 compatability.

To allow for streamlining 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 allowing for strict compliance
with [CAT-FTPSEC] 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 protcol over TLS; or explicitly by use of
the PBSZ/PROT sequence) the data connection state may be modified by
the client issueing 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 the IANA Considerations section.

7. 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 [CAT-FTPSEC] 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, the security of subsequent data
   connections should be at that level until another PROT command is
   issued, 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 [CAT-FTPSEC], there is no facility for securing
   the Data connection with an insecure Control connection.

   The command defined in [CAT-FTPSEC] 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 [CAT-FTPSEC], 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.

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 [CAT-FTPSEC], 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 [CAT-FTPSEC], 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 <Mechanism>" command.
   The initial state of the data connection will be "Clear"
   (this is the correct behaviour as indicated by [CAT-
   FTPSEC].)

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

8. 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.

8.1. The server's view of the control connection

   A server will have a policy statement somewhere which may:

      - Deny any command before TLS is negotiated (this might cause
      problems if a SITE or somesuch 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 succesfully, but the server
decides that the session parameters are not acceptable (e.g.
Distinguished Name in 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.

8.2. 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.
The reply to indicate that the PROT setting is incorrect is "521

data connection cannot be opened with this PROT setting"

            STOR
            RETR

          NLST
          LIST
          STOU
          APPE

     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 do a full negotiation to a client which it has already
     authenticated.

     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.

   8.3. 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 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 client should issue an unprotected
QUIT command to end the session.

The TLS negotiation completed succesfully, 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.

8.4. 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.

[9](). Who negotiates what, where and how

   9.1. Do we protect at all ?

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

   9.2. What level of protection do we use ?

      Decided entirely by the TLS CipherSuite negotiation.

   9.3. Do we protect data connections in general ?

      Client issues PROT command, server accepts or rejects.

   9.4. 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 this time.

   9.5. 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.

[10](). Timing Diagrams

   10.1. Establishing a protected session.

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

                                                    socket()
                                                    bind()
  socket()
  connect()  ----------------------------------------> accept()
  AUTH TLS   --------------------------------------->
             <--------------------------------------- 234
  SSLneg()   <---------------------------------------> SSLneg()
  PBSZ 0     --------------------------------------->
             <--------------------------------------- 200
  PROT P     --------------------------------------->
             <--------------------------------------- 200
  USER fred  --------------------------------------->
             <--------------------------------------- 331
  PASS pass  --------------------------------------->
             <--------------------------------------- 230
```
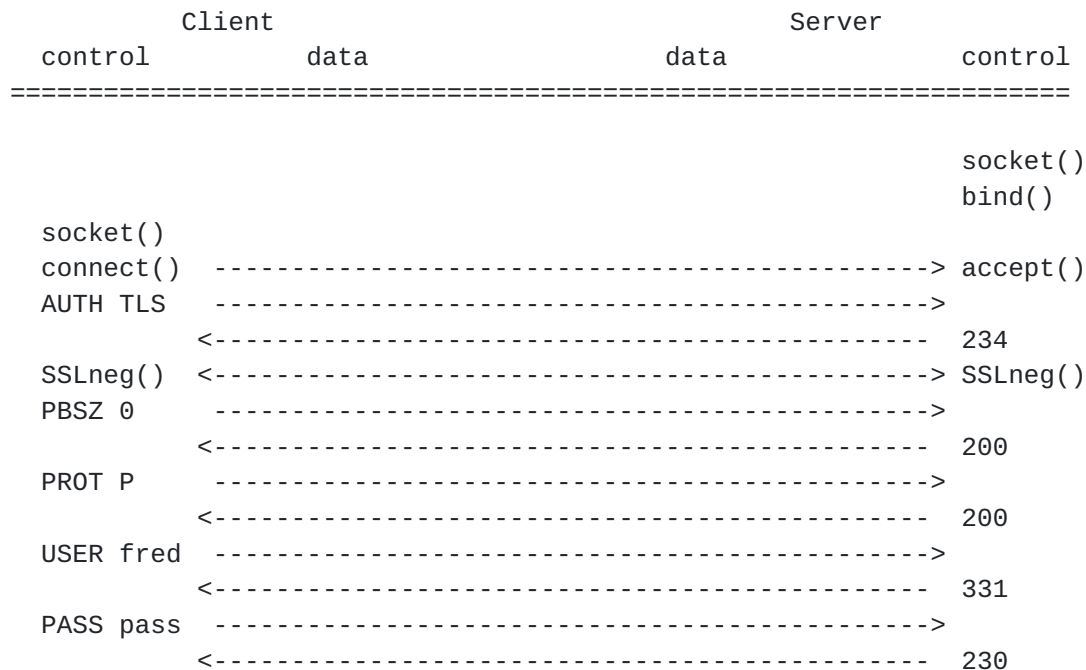
NOTE: the order of the PBSZ/PROT pair and the USER/PASS pair (with
respect to each other) is not important (i.e. the USER/PASS can happen
prior to the PBSZ/PROT - or indeed the server can refuse to allow a

PBSZ/PROT pair until the USER/PASS pair has happened).

10.2. 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
```

   10.3. A firewall friendly data transfer without protection.
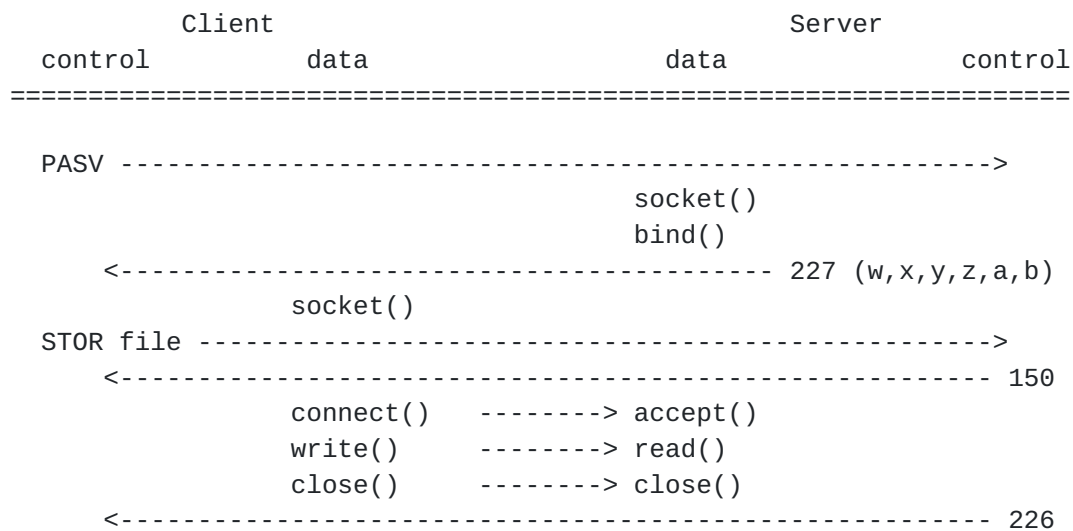
```
          Client                              Server
  control          data               data              control
================================================================

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

   10.4. 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()
                 SSLneg()   <--------> SSLneg()
                 SSLwrite() --------> SSLread()
                 close()    --------> close()
      <----------------------------------------------------- 226
```

   10.5. 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 -------------------------------------------------->
     <-------------------------------------------------- 150
              connect()   --------> accept()
              SSLneg()    <-------> SSLneg()
              SSLwrite()  --------> SSLread()
              close()     --------> close()
     <-------------------------------------------------- 226
```

11. Security Considerations

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

12. 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"
   (all uppercase only) 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".

13. Network Management

   NONE

14. Internationalization

   NONE

15. Scalability & Limits

   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.

[16]. 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).

17. Acknowledgements

o Netscape Communications Corp 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 Martin Carpenter for having the time and patience to criticise and
correct.

18. References

[FTP-SOCKS] Bellovin, S., "Firewall-Friendly FTP"

RFC 1579, February 1994.

[TLS-DESC] A description of the TLS protocol.
   TLS (Transport Layer Security) is the IETF version of and
   enhancement to the Netscape SSL protocol. TLS is backwards
   compatible with SSL and discussions in this document are relevant
   to all versions of TLS and SSL. The current version of TLS is
   described in

   Freier, A., "The TLS Protocol Version 1.0"
      draft-ietf-tls-protocol-01.txt.

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

[SRA-FTP] "SRA - Secure RPC Authentication for TELNET and FTP Version
1.1"

```
     file://ftp.funet.fi/security/login/telnet/doc/sra/sra.README
```

   [CAT-FTPSEC] Horowitz, M., "FTP Security Extensions"
      draft-ietf-cat-ftpsec-09.txt

[19]. Author's Contact Addresses

Paul Ford-Hutchinson   Tim Hudson   Eric Murray
  OSU 1   CryptSoft Pty Ltd   LNE Consulting
  IBM UK Ltd   PO Box 6324
  PO Box 31   Fairfield 4103
  Birmingham Road   Queensland
  Warwick   Australia
  England
  CV34 5JL

Tel - +44 (0)1926 464836   +61 7 32781581
Fax - +44 (0)1926 496482
email - pfh@uk.ibm.com   tjh@cryptsoft.com   ericm@lne.com
  pvfh@compuserve.com

Apendices

A. **Summary of [TLS-DESC]**

   The TLS protocol is developed by the IETF TLS working group. It is
   based on the SSL protocol proposed by Netscape. 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
   sessions 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 [CAT-FTPSEC]**

Extension 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 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, also it 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 should be replied
   to with a 536 reply.

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.

This document expires on 24th September, 1997