Network Working Group
Request for Comments: 2228
Updates: 959
Category: Standards Track

M. Horowitz
Cygnus Solutions
S. Lunt
Bellcore

October 1997

FTP Security Extensions

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This document defines extensions to the FTP specification STD 9, RFC 959, "FILE TRANSFER PROTOCOL (FTP)" (October 1985). These extensions provide strong authentication, integrity, and confidentiality on both the control and data channels with the introduction of new optional commands, replies, and file transfer encodings.

The following new optional commands are introduced in this specification:

AUTH (Authentication/Security Mechanism),
ADAT (Authentication/Security Data),
PROT (Data Channel Protection Level),
PBSZ (Protection Buffer Size),
CCC (Clear Command Channel),
MIC (Integrity Protected Command),
CONF (Confidentiality Protected Command), and
ENC (Privacy Protected Command).

A new class of reply types (6yz) is also introduced for protected replies.

None of the above commands are required to be implemented, but interdependencies exist. These dependencies are documented with the commands.

Note that this specification is compatible with STD 9, RFC 959.

1. Introduction

The File Transfer Protocol (FTP) currently defined in STD 9, RFC 959 and in place on the Internet uses usernames and passwords passed in cleartext to authenticate clients to servers (via the USER and PASS commands). Except for services such as "anonymous" FTP archives, this represents a security risk whereby passwords can be stolen through monitoring of local and wide-area networks. This either aids potential attackers through password exposure and/or limits accessibility of files by FTP servers who cannot or will not accept the inherent security risks.

Aside from the problem of authenticating users in a secure manner, there is also the problem of authenticating servers, protecting sensitive data and/or verifying its integrity. An attacker may be able to access valuable or sensitive data merely by monitoring a network, or through active means may be able to delete or modify the data being transferred so as to corrupt its integrity. An active attacker may also initiate spurious file transfers to and from a site of the attacker’s choice, and may invoke other commands on the server. FTP does not currently have any provision for the encryption or verification of the authenticity of commands, replies, or transferred data. Note that these security services have value even to anonymous file access.

Current practice for sending files securely is generally either:

1. via FTP of files pre-encrypted under keys which are manually distributed,

2. via electronic mail containing an encoding of a file encrypted under keys which are manually distributed,

3. via a PEM message, or

4. via the rcp command enhanced to use Kerberos.

None of these means could be considered even a de facto standard, and none are truly interactive. A need exists to securely transfer files using FTP in a secure manner which is supported within the FTP protocol in a consistent manner and which takes advantage of existing security infrastructure and technology. Extensions are necessary to the FTP specification if these security services are to be introduced into the protocol in an interoperable way.

Although the FTP control connection follows the Telnet protocol, and Telnet has defined an authentication and encryption option [TELNET-SEC], [RFC-1123] explicitly forbids the use of Telnet option negotiation over the control connection (other than Synch and IP).

Also, the Telnet authentication and encryption option does not provide for integrity protection only (without confidentiality), and does not address the protection of the data channel.

2. FTP Security Overview

At the highest level, the FTP security extensions seek to provide an abstract mechanism for authenticating and/or authorizing connections, and integrity and/or confidentiality protecting commands, replies, and data transfers.

In the context of FTP security, authentication is the establishment of a client’s identity and/or a server’s identity in a secure way, usually using cryptographic techniques. The basic FTP protocol does not have a concept of authentication.

Authorization is the process of validating a user for login. The basic authorization process involves the USER, PASS, and ACCT commands. With the FTP security extensions, authentication established using a security mechanism may also be used to make the authorization decision.

Without the security extensions, authentication of the client, as this term is usually understood, never happens. FTP authorization is accomplished with a password, passed on the network in the clear as the argument to the PASS command. The possessor of this password is assumed to be authorized to transfer files as the user named in the USER command, but the identity of the client is never securely established.

An FTP security interaction begins with a client telling the server what security mechanism it wants to use with the AUTH command. The server will either accept this mechanism, reject this mechanism, or, in the case of a server which does not implement the security extensions, reject the command completely. The client may try multiple security mechanisms until it requests one which the server accepts. This allows a rudimentary form of negotiation to take place. (If more complex negotiation is desired, this may be implemented as a security mechanism.) The server’s reply will indicate if the client must respond with additional data for the security mechanism to interpret. If none is needed, this will usually mean that the mechanism is one where the password (specified by the PASS command) is to be interpreted differently, such as with a token or one-time password system.

If the server requires additional security information, then the client and server will enter into a security data exchange. The client will send an ADAT command containing the first block of security data. The server's reply will indicate if the data exchange is complete, if there was an error, or if more data is needed. The server's reply can optionally contain security data for the client to interpret. If more data is needed, the client will send another ADAT command containing the next block of data, and await the server's reply. This exchange can continue as many times as necessary. Once this exchange completes, the client and server have established a security association. This security association may include authentication (client, server, or mutual) and keying information for integrity and/or confidentiality, depending on the mechanism in use.

The term "security data" here is carefully chosen. The purpose of the security data exchange is to establish a security association, which might not actually include any authentication at all, between the client and the server as described above. For instance, a Diffie-Hellman exchange establishes a secret key, but no authentication takes place. If an FTP server has an RSA key pair but the client does not, then the client can authenticate the server, but the server cannot authenticate the client.

Once a security association is established, authentication which is a part of this association may be used instead of or in addition to the standard username/password exchange for authorizing a user to connect to the server. A username specified by the USER command is always required to specify the identity to be used on the server.

In order to prevent an attacker from inserting or deleting commands on the control stream, if the security association supports integrity, then the server and client must use integrity protection on the control stream, unless it first transmits a CCC command to turn off this requirement. Integrity protection is performed with the MIC and ENC commands, and the 63z reply codes. The CCC command and its reply must be transmitted with integrity protection. Commands and replies may be transmitted without integrity (that is, in the clear or with confidentiality only) only if no security association is established, the negotiated security association does not support integrity, or the CCC command has succeeded.

Once the client and server have negotiated with the PBSZ command an acceptable buffer size for encapsulating protected data over the data channel, the security mechanism may also be used to protect data channel transfers.

Policy is not specified by this document. In particular, client and server implementations may choose to implement restrictions on what operations can be performed depending on the security association which exists. For example, a server may require that a client authorize via a security mechanism rather than using a password, require that the client provide a one-time password from a token, require at least integrity protection on the command channel, or require that certain files only be transmitted encrypted. An anonymous ftp client might refuse to do file transfers without integrity protection in order to insure the validity of files downloaded.

No particular set of functionality is required, except as dependencies described in the next section. This means that none of authentication, integrity, or confidentiality are required of an implementation, although a mechanism which does none of these is not of much use. For example, it is acceptable for a mechanism to implement only integrity protection, one-way authentication and/or encryption, encryption without any authentication or integrity protection, or any other subset of functionality if policy or technical considerations make this desirable. Of course, one peer might require as a matter of policy stronger protection than the other is able to provide, preventing perfect interoperability.

3. New FTP Commands

The following commands are optional, but dependent on each other. They are extensions to the FTP Access Control Commands.

The reply codes documented here are generally described as recommended, rather than required. The intent is that reply codes describing the full range of success and failure modes exist, but that servers be allowed to limit information presented to the client. For example, a server might implement a particular security mechanism, but have a policy restriction against using it. The server should respond with a 534 reply code in this case, but may respond with a 504 reply code if it does not wish to divulge that the disallowed mechanism is supported. If the server does choose to use a different reply code than the recommended one, it should try to use a reply code which only differs in the last digit. In all cases, the server must use a reply code which is documented as returnable from the command received, and this reply code must begin with the same digit as the recommended reply code for the situation.

AUTHENTICATION/SECURITY MECHANISM (AUTH)

The argument field is a Telnet string identifying a supported mechanism. This string is case-insensitive. Values must be registered with the IANA, except that values beginning with "X-" are reserved for local use.

If the server does not recognize the AUTH command, it must respond with reply code 500. This is intended to encompass the large deployed base of non-security-aware ftp servers, which will respond with reply code 500 to any unrecognized command. If the server does recognize the AUTH command but does not implement the security extensions, it should respond with reply code 502.

If the server does not understand the named security mechanism, it should respond with reply code 504.

If the server is not willing to accept the named security mechanism, it should respond with reply code 534.

If the server is not able to accept the named security mechanism, such as if a required resource is unavailable, it should respond with reply code 431.

If the server is willing to accept the named security mechanism, but requires security data, it must respond with reply code 334.

If the server is willing to accept the named security mechanism, and does not require any security data, it must respond with reply code 234.

If the server is responding with a 334 reply code, it may include security data as described in the next section.

Some servers will allow the AUTH command to be reissued in order to establish new authentication. The AUTH command, if accepted, removes any state associated with prior FTP Security commands. The server must also require that the user reauthorize (that is, reissue some or all of the USER, PASS, and ACCT commands) in this case (see section 4 for an explanation of "authorize" in this context).

AUTHENTICATION/SECURITY DATA (ADAT)

The argument field is a Telnet string representing base 64 encoded security data (see Section 9, "Base 64 Encoding"). If a reply code indicating success is returned, the server may also use a string of the form "ADAT=base64data" as the text part of the reply if it wishes to convey security data back to the client.

The data in both cases is specific to the security mechanism specified by the previous AUTH command. The ADAT command, and the associated replies, allow the client and server to conduct an arbitrary security protocol. The security data exchange must include enough information for both peers to be aware of which optional features are available. For example, if the client does not support data encryption, the server must be made aware of this, so it will know not to send encrypted command channel replies. It is strongly recommended that the security mechanism provide sequencing on the command channel, to insure that commands are not deleted, reordered, or replayed.

The ADAT command must be preceded by a successful AUTH command, and cannot be issued once a security data exchange completes (successfully or unsuccessfully), unless it is preceded by an AUTH command to reset the security state.

If the server has not yet received an AUTH command, or if a prior security data exchange completed, but the security state has not been reset with an AUTH command, it should respond with reply code 503.

If the server cannot base 64 decode the argument, it should respond with reply code 501.

If the server rejects the security data (if a checksum fails, for instance), it should respond with reply code 535.

If the server accepts the security data, and requires additional data, it should respond with reply code 335.

If the server accepts the security data, but does not require any additional data (i.e., the security data exchange has completed successfully), it must respond with reply code 235.

If the server is responding with a 235 or 335 reply code, then it may include security data in the text part of the reply as specified above.

If the ADAT command returns an error, the security data exchange will fail, and the client must reset its internal security state. If the client becomes unsynchronized with the server (for example, the server sends a 234 reply code to an AUTH command, but the client has more data to transmit), then the client must reset the server’s security state.

PROTECTION BUFFER SIZE (PBSZ)

The argument is a decimal integer representing the maximum size, in bytes, of the encoded data blocks to be sent or received during file transfer. This number shall be no greater than can be represented in a 32-bit unsigned integer.

This command allows the FTP client and server to negotiate a maximum protected buffer size for the connection. There is no default size; the client must issue a PBSZ command before it can issue the first PROT command.

The PBSZ command must be preceded by a successful security data exchange.

If the server cannot parse the argument, or if it will not fit in 32 bits, it should respond with a 501 reply code.

If the server has not completed a security data exchange with the client, it should respond with a 503 reply code.

Otherwise, the server must reply with a 200 reply code. If the size provided by the client is too large for the server, it must use a string of the form "PBSZ=number" in the text part of the reply to indicate a smaller buffer size. The client and the server must use the smaller of the two buffer sizes if both buffer sizes are specified.

DATA CHANNEL PROTECTION LEVEL (PROT)

The argument is a single Telnet character code specifying the data channel protection level.

This command indicates to the server what type of data channel protection the client and server will be using. The following codes are assigned:

C - Clear
S - Safe
E - Confidential
P - Private

The default protection level if no other level is specified is Clear. The Clear protection level indicates that the data channel will carry the raw data of the file transfer, with no security applied. The Safe protection level indicates that the data will be integrity protected. The Confidential protection level indicates that the data will be confidentiality protected. The Private protection level indicates that the data will be integrity and confidentiality protected.

It is reasonable for a security mechanism not to provide all data channel protection levels. It is also reasonable for a mechanism to provide more protection at a level than is required (for instance, a mechanism might provide Confidential protection, but include integrity-protection in that encoding, due to API or other considerations).

The PROT command must be preceded by a successful protection buffer size negotiation.

If the server does not understand the specified protection level, it should respond with reply code 504.

If the current security mechanism does not support the specified protection level, the server should respond with reply code 536.

If the server has not completed a protection buffer size negotiation with the client, it should respond with a 503 reply code.

The PROT command will be rejected and the server should reply 503 if no previous PBSZ command was issued.

If the server is not willing to accept the specified protection level, it should respond with reply code 534.

If the server is not able to accept the specified protection level, such as if a required resource is unavailable, it should respond with reply code 431.

Otherwise, the server must reply with a 200 reply code to indicate that the specified protection level is accepted.

CLEAR COMMAND CHANNEL (CCC)

This command does not take an argument.

It is desirable in some environments to use a security mechanism to authenticate and/or authorize the client and server, but not to perform any integrity checking on the subsequent commands. This might be used in an environment where IP security is in place, insuring that the hosts are authenticated and that TCP streams cannot be tampered, but where user authentication is desired.

If unprotected commands are allowed on any connection, then an attacker could insert a command on the control stream, and the server would have no way to know that it was invalid. In order to prevent such attacks, once a security data exchange completes successfully, if the security mechanism supports integrity, then integrity (via the MIC or ENC command, and 631 or 632 reply) must be used, until the CCC command is issued to enable non-integrity protected control channel messages. The CCC command itself must be integrity protected.

Once the CCC command completes successfully, if a command is not protected, then the reply to that command must also not be protected. This is to support interoperability with clients which do not support protection once the CCC command has been issued.

This command must be preceded by a successful security data exchange.

If the command is not integrity-protected, the server must respond with a 533 reply code.

If the server is not willing to turn off the integrity requirement, it should respond with a 534 reply code.

Otherwise, the server must reply with a 200 reply code to indicate that unprotected commands and replies may now be used on the command channel.

INTEGRITY PROTECTED COMMAND (MIC) and
CONFIDENTIALITY PROTECTED COMMAND (CONF) and
PRIVACY PROTECTED COMMAND (ENC)

The argument field of MIC is a Telnet string consisting of a base 64 encoded "safe" message produced by a security mechanism specific message integrity procedure. The argument field of CONF is a Telnet string consisting of a base 64 encoded "confidential" message produced by a security mechanism specific confidentiality procedure. The argument field of ENC is a Telnet string consisting of a base 64 encoded "private" message produced by a security mechanism specific message integrity and confidentiality procedure.

The server will decode and/or verify the encoded message.

This command must be preceded by a successful security data exchange.

A server may require that the first command after a successful security data exchange be CCC, and not implement the protection commands at all. In this case, the server should respond with a 502 reply code.

If the server cannot base 64 decode the argument, it should respond with a 501 reply code.

If the server has not completed a security data exchange with the client, it should respond with a 503 reply code.

If the server has completed a security data exchange with the client using a mechanism which supports integrity, and requires a CCC command due to policy or implementation limitations, it should respond with a 503 reply code.

If the server rejects the command because it is not supported by the current security mechanism, the server should respond with reply code 537.

If the server rejects the command (if a checksum fails, for instance), it should respond with reply code 535.

If the server is not willing to accept the command (if privacy is required by policy, for instance, or if a CONF command is received before a CCC command), it should respond with reply code 533.

Otherwise, the command will be interpreted as an FTP command. An end-of-line code need not be included, but if one is included, it must be a Telnet end-of-line code, not a local end-of-line code.

The server may require that, under some or all circumstances, all commands be protected. In this case, it should make a 533 reply to commands other than MIC, CONF, and ENC.

4. Login Authorization

The security data exchange may, among other things, establish the identity of the client in a secure way to the server. This identity may be used as one input to the login authorization process.

In response to the FTP login commands (AUTH, PASS, ACCT), the server may choose to change the sequence of commands and replies specified by RFC 959 as follows. There are also some new replies available.

If the server is willing to allow the user named by the USER command to log in based on the identity established by the security data exchange, it should respond with reply code 232.

If the security mechanism requires a challenge/response password, it should respond to the USER command with reply code 336. The text part of the reply should contain the challenge. The client must display the challenge to the user before prompting for the password in this case. This is particularly relevant to more sophisticated clients or graphical user interfaces which provide dialog boxes or other modal input. These clients should be careful not to prompt for the password before the username has been sent to the server, in case the user needs the challenge in the 336 reply to construct a valid password.

5. New FTP Replies

The new reply codes are divided into two classes. The first class is new replies made necessary by the new FTP Security commands. The second class is a new reply type to indicate protected replies.

5.1. New individual reply codes

```
232 User logged in, authorized by security data exchange.
234 Security data exchange complete.
235 [ADAT=base64data]
      ; This reply indicates that the security data exchange
      ; completed successfully.  The square brackets are not
      ; to be included in the reply, but indicate that
      ; security data in the reply is optional.

334 [ADAT=base64data]
      ; This reply indicates that the requested security mechanism
      ; is ok, and includes security data to be used by the client
      ; to construct the next command.  The square brackets are not
      ; to be included in the reply, but indicate that
      ; security data in the reply is optional.
335 [ADAT=base64data]
      ; This reply indicates that the security data is
      ; acceptable, and more is required to complete the
      ; security data exchange.  The square brackets
      ; are not to be included in the reply, but indicate
      ; that security data in the reply is optional.
```

```
   336 Username okay, need password.  Challenge is "...."
         ; The exact representation of the challenge should be chosen
         ; by the mechanism to be sensible to the human user of the
         ; system.

   431 Need some unavailable resource to process security.

   533 Command protection level denied for policy reasons.
   534 Request denied for policy reasons.
   535 Failed security check (hash, sequence, etc).
   536 Requested PROT level not supported by mechanism.
   537 Command protection level not supported by security mechanism.
```

5.2. Protected replies.

One new reply type is introduced:

6yz Protected reply

There are three reply codes of this type. The first, reply code 631 indicates an integrity protected reply. The second, reply code 632, indicates a confidentiality and integrity protected reply. the third, reply code 633, indicates a confidentiality protected reply.

The text part of a 631 reply is a Telnet string consisting of a base 64 encoded "safe" message produced by a security mechanism specific message integrity procedure. The text part of a 632 reply is a Telnet string consisting of a base 64 encoded "private" message produced by a security mechanism specific message confidentiality and integrity procedure. The text part of a 633 reply is a Telnet string consisting of a base 64 encoded "confidential" message produced by a security mechanism specific message confidentiality procedure.

The client will decode and verify the encoded reply. How failures decoding or verifying replies are handled is implementation-specific. An end-of-line code need not be included, but if one is included, it must be a Telnet end-of-line code, not a local end-of-line code.

A protected reply may only be sent if a security data exchange has succeeded.

The 63z reply may be a multiline reply. In this case, the plaintext reply must be broken up into a number of fragments. Each fragment must be protected, then base 64 encoded in order into a separate line of the multiline reply. There need not be any correspondence between the line breaks in the plaintext reply and the encoded reply. Telnet end-of-line codes must appear in the plaintext of the encoded reply, except for the final end-of-line code, which is optional.

The multiline reply must be formatted more strictly than the continuation specification in RFC 959. In particular, each line before the last must be formed by the reply code, followed immediately by a hyphen, followed by a base 64 encoded fragment of the reply.

For example, if the plaintext reply is

```
123-First line
Second line
  234 A line beginning with numbers
123 The last line
```

then the resulting protected reply could be any of the following (the first example has a line break only to fit within the margins):

```
631 base64(protect("123-First line\r\nSecond line\r\n  234 A line
631-base64(protect("123-First line\r\n"))
631-base64(protect("Second line\r\n"))
631-base64(protect("  234 A line beginning with numbers\r\n"))
631 base64(protect("123 The last line"))

631-base64(protect("123-First line\r\nSecond line\r\n  234 A line b"))
631 base64(protect("eginning with numbers\r\n123 The last line\r\n"))
```

6. Data Channel Encapsulation

When data transfers are protected between the client and server (in either direction), certain transformations and encapsulations must be performed so that the recipient can properly decode the transmitted file.

The sender must apply all protection services after transformations associated with the representation type, file structure, and transfer mode have been performed. The data sent over the data channel is, for the purposes of protection, to be treated as a byte stream.

When performing a data transfer in an authenticated manner, the authentication checks are performed on individual blocks of the file, rather than on the file as a whole. Consequently, it is possible for insertion attacks to insert blocks into the data stream (i.e., replays) that authenticate correctly, but result in a corrupted file being undetected by the receiver. To guard against such attacks, the specific security mechanism employed should include mechanisms to protect against such attacks. Many GSS-API mechanisms usable with the specification in Appendix I, and the Kerberos mechanism in Appendix II do so.

The sender must take the input byte stream, and break it up into blocks such that each block, when encoded using a security mechanism specific procedure, will be no larger than the buffer size negotiated by the client with the PBSZ command. Each block must be encoded, then transmitted with the length of the encoded block prepended as a four byte unsigned integer, most significant byte first.

When the end of the file is reached, the sender must encode a block of zero bytes, and send this final block to the recipient before closing the data connection.

The recipient will read the four byte length, read a block of data that many bytes long, then decode and verify this block with a security mechanism specific procedure. This must be repeated until a block encoding a buffer of zero bytes is received. This indicates the end of the encoded byte stream.

Any transformations associated with the representation type, file structure, and transfer mode are to be performed by the recipient on the byte stream resulting from the above process.

When using block transfer mode, the sender’s (cleartext) buffer size is independent of the block size.

The server will reply 534 to a STOR, STOU, RETR, LIST, NLST, or APPE command if the current protection level is not at the level dictated by the server’s security requirements for the particular file transfer.

If any data protection services fail at any time during data transfer at the server end (including an attempt to send a buffer size greater than the negotiated maximum), the server will send a 535 reply to the data transfer command (either STOR, STOU, RETR, LIST, NLST, or APPE).

7. Potential policy considerations

While there are no restrictions on client and server policy, there are a few recommendations which an implementation should implement.

- Once a security data exchange takes place, a server should require all commands be protected (with integrity and/or confidentiality), and it should protect all replies. Replies should use the same level of protection as the command which produced them. This includes replies which indicate failure of the MIC, CONF, and ENC commands. In particular, it is not meaningful to require that AUTH and ADAT be protected; it is meaningful and useful to require that PROT and PBSZ be protected. In particular, the use of CCC is not recommended, but is defined in the interest of interoperability between implementations which might desire such functionality.

- A client should encrypt the PASS command whenever possible. It is reasonable for the server to refuse to accept a non-encrypted PASS command if the server knows encryption is available.

- Although no security commands are required to be implemented, it is recommended that an implementation provide all commands which can be implemented, given the mechanisms supported and the policy considerations of the site (export controls, for instance).

8. Declarative specifications

These sections are modelled after sections 5.3 and 5.4 of RFC 959, which describe the same information, except for the standard FTP commands and replies.

8.1. FTP Security commands and arguments

```
AUTH <SP> <mechanism-name> <CRLF>
ADAT <SP> <base64data> <CRLF>
PROT <SP> <prot-code> <CRLF>
PBSZ <SP> <decimal-integer> <CRLF>
MIC <SP> <base64data> <CRLF>
CONF <SP> <base64data> <CRLF>
ENC <SP> <base64data> <CRLF>

<mechanism-name> ::= <string>
<base64data> ::= <string>
        ; must be formatted as described in section 9
<prot-code> ::= C | S | E | P
<decimal-integer> ::= any decimal integer from 1 to (2^32)-1
```

8.2. Command-Reply sequences

```
Security Association Setup
   AUTH
      234
      334
      502, 504, 534, 431
      500, 501, 421
   ADAT
      235
      335
      503, 501, 535
      500, 501, 421
Data protection negotiation commands
   PBSZ
      200
      503
      500, 501, 421, 530
   PROT
      200
      504, 536, 503, 534, 431
      500, 501, 421, 530
Command channel protection commands
   MIC
      535, 533
      500, 501, 421
   CONF
      535, 533
      500, 501, 421
   ENC
      535, 533
      500, 501, 421
Security-Enhanced login commands (only new replies listed)
   USER
      232
      336
Data channel commands (only new replies listed)
   STOR
      534, 535
   STOU
      534, 535
   RETR
      534, 535
   LIST
      534, 535
   NLST
      534, 535
   APPE
      534, 535
```

In addition to these reply codes, any security command can return 500, 501, 502, 533, or 421. Any ftp command can return a reply code encapsulated in a 631, 632, or 633 reply once a security data exchange has completed successfully.

9. State Diagrams

This section includes a state diagram which demonstrates the flow of authentication and authorization in a security enhanced FTP implementation. The rectangular blocks show states where the client must issue a command, and the diamond blocks show states where the server must issue a response.

```
          ,------------------,  USER
       __\| Unauthenticated  |_________\
      |  /| (new connection) |         /|
      |   `------------------'          |
      |            |                    |
      |            | AUTH               |
      |            V                    |
      |           / \                   |
      | 4yz,5yz  /   \   234            |
      |<--------<     >------------->.  |
      |          \   /               |  |
      |           \_/                |  |
      |            |                 |  |
      |            | 334             |  |
      |            V                 |  |
      |  ,--------------------,      |  |
      |  | Need Security Data |<--.  |  |
      |  `--------------------'   |  |  |
      |            |              |  |  |
      |            | ADAT         |  |  |
      |            V              |  |  |
      |           / \             |  |  |
      | 4yz,5yz  /   \   335      |  |  |
      `<--------<     >-----------'  |  |
                 \   /               |  |
                  \_/                |  |
                   |                 |  |
                   | 235             |  |
                   V                 |  |
           ,---------------.         |  |
      ,--->| Authenticated |<--------'  |  After the client and server
      |    `---------------'            |  have completed authenti-
      |            |                    |  cation, command must be
      |            | USER               |  integrity-protected if
      |            |                    |  integrity is available.  The
      |            |<-------------------'  CCC command may be issued to
      |            V                       relax this restriction.
      |           / \
      | 4yz,5yz  /   \   2yz
      |<--------<     >------------->.
      |          \   /               |
      |           \_/                |
      |            |                 |
      |            | 3yz             |
      |            V                 |
      |    ,---------------.         |
      |    | Need Password |         |
      |    `---------------'         |
      |            |                 |
      |            | PASS            |
      |            V                 |
      |           / \                |
      | 4yz,5yz  /   \   2yz         |
      |<--------<     >------------->|
      |          \   /               |
      |           \_/                |
      |            |                 |
      |            | 3yz             |
      |            V                 |
      |    ,--------------.          |
      |    | Need Account |          |
      |    `--------------'          |
      |            |                 |
      |            | ACCT            |
      |            V                 |
      |           / \                |
      | 4yz,5yz  /   \   2yz         |
      `<--------<     >------------->|
                 \   /               |
                  \_/                |
                   |                 |
                   | 3yz             |
                   V                 |
             ,-------------.         |
             | Authorized  |/________|
             | (Logged in) |\
             `-------------'
```

10. Base 64 Encoding

Base 64 encoding is the same as the Printable Encoding described in Section 4.3.2.4 of [RFC-1421], except that line breaks must not be included. This encoding is defined as follows.

Proceeding from left to right, the bit string resulting from the mechanism specific protection routine is encoded into characters which are universally representable at all sites, though not necessarily with the same bit patterns (e.g., although the character "E" is represented in an ASCII-based system as hexadecimal 45 and as hexadecimal C5 in an EBCDIC-based system, the local significance of the two representations is equivalent).

A 64-character subset of International Alphabet IA5 is used, enabling 6 bits to be represented per printable character. (The proposed subset of characters is represented identically in IA5 and ASCII.) The character "=" signifies a special processing function used for padding within the printable encoding procedure.

The encoding process represents 24-bit groups of input bits as output strings of 4 encoded characters. Proceeding from left to right across a 24-bit input group output from the security mechanism specific message protection procedure, each 6-bit group is used as an index into an array of 64 printable characters, namely "[A-Z][a-z][0-9]+/". The character referenced by the index is placed in the output string. These characters are selected so as to be universally representable, and the set excludes characters with particular significance to Telnet (e.g., "<CR>", "<LF>", IAC).

Special processing is performed if fewer than 24 bits are available in an input group at the end of a message. A full encoding quantum is always completed at the end of a message. When fewer than 24 input bits are available in an input group, zero bits are added (on the right) to form an integral number of 6-bit groups. Output character positions which are not required to represent actual input data are set to the character "=". Since all canonically encoded output is an integral number of octets, only the following cases can arise: (1) the final quantum of encoding input is an integral multiple of 24 bits; here, the final unit of encoded output will be an integral multiple of 4 characters with no "=" padding, (2) the final quantum of encoding input is exactly 8 bits; here, the final unit of encoded output will be two characters followed by two "=" padding characters, or (3) the final quantum of encoding input is exactly 16 bits; here, the final unit of encoded output will be three characters followed by one "=" padding character.

Implementors must keep in mind that the base 64 encodings in ADAT, MIC, CONF, and ENC commands, and in 63z replies may be arbitrarily long. Thus, the entire line must be read before it can be processed. Several successive reads on the control channel may be necessary. It is not appropriate to for a server to reject a command containing a base 64 encoding simply because it is too long (assuming that the decoding is otherwise well formed in the context in which it was sent).

Case must not be ignored when reading commands and replies containing base 64 encodings.

11. Security Considerations

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

Third party file transfers cannot be secured using these extensions, since a security context cannot be established between two servers using these facilities (no control connection exists between servers over which to pass ADAT tokens). Further work in this area is deferred.

12. Acknowledgements

I would like to thank the members of the CAT WG, as well as all participants in discussions on the "cat-ietf@mit.edu" mailing list, for their contributions to this document. I would especially like to thank Sam Sjogren, John Linn, Ted Ts’o, Jordan Brown, Michael Kogut, Derrick Brashear, John Gardiner Myers, Denis Pinkas, and Karri Balk for their contributions to this work. Of course, without Steve Lunt, the author of the first six revisions of this document, it would not exist at all.

13. References

[TELNET-SEC] Borman, D., "Telnet Authentication and Encryption Option", Work in Progress.

[RFC-1123] Braden, R., "Requirements for Internet Hosts -- Application and Support", STD 3, RFC 1123, October 1989.

[RFC-1421] Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures", RFC 1421, February 1993.

14. Author's Address

Marc Horowitz
Cygnus Solutions
955 Massachusetts Avenue
Cambridge, MA 02139

Phone: +1 617 354 7688
EMail: marc@cygnus.com

Appendix I: Specification under the GSSAPI

In order to maximise the utility of new security mechanisms, it is desirable that new mechanisms be implemented as GSSAPI mechanisms rather than as FTP security mechanisms. This will enable existing ftp implementations to support the new mechanisms more easily, since little or no code will need to be changed. In addition, the mechanism will be usable by other protocols, such as IMAP, which are built on top of the GSSAPI, with no additional specification or implementation work needed by the mechanism designers.

The security mechanism name (for the AUTH command) associated with all mechanisms employing the GSSAPI is GSSAPI. If the server supports a security mechanism employing the GSSAPI, it must respond with a 334 reply code indicating that an ADAT command is expected next.

The client must begin the authentication exchange by calling GSS_Init_Sec_Context, passing in 0 for input_context_handle (initially), and a targ_name equal to output_name from GSS_Import_Name called with input_name_type of Host-Based Service and input_name_string of "ftp@hostname" where "hostname" is the fully qualified host name of the server with all letters in lower case. (Failing this, the client may try again using input_name_string of "host@hostname".) The output_token must then be base 64 encoded and sent to the server as the argument to an ADAT command. If GSS_Init_Sec_Context returns GSS_S_CONTINUE_NEEDED, then the client must expect a token to be returned in the reply to the ADAT command. This token must subsequently be passed to another call to GSS_Init_Sec_Context. In this case, if GSS_Init_Sec_Context returns no output_token, then the reply code from the server for the previous ADAT command must have been 235. If GSS_Init_Sec_Context returns GSS_S_COMPLETE, then no further tokens are expected from the server, and the client must consider the server authenticated.

The server must base 64 decode the argument to the ADAT command and pass the resultant token to GSS_Accept_Sec_Context as input_token, setting acceptor_cred_handle to NULL (for "use default credentials"), and 0 for input_context_handle (initially). If an output_token is returned, it must be base 64 encoded and returned to the client by including "ADAT=base64string" in the text of the reply. If GSS_Accept_Sec_Context returns GSS_S_COMPLETE, the reply code must be 235, and the server must consider the client authenticated. If GSS_Accept_Sec_Context returns GSS_S_CONTINUE_NEEDED, the reply code must be 335. Otherwise, the reply code should be 535, and the text of the reply should contain a descriptive error message.

The chan_bindings input to GSS_Init_Sec_Context and GSS_Accept_Sec_Context should use the client internet address and server internet address as the initiator and acceptor addresses, respectively. The address type for both should be GSS_C_AF_INET. No application data should be specified.

Since GSSAPI supports anonymous peers to security contexts, it is possible that the client's authentication of the server does not actually establish an identity.

The procedure associated with MIC commands, 631 replies, and Safe file transfers is:

GSS_Wrap for the sender, with conf_flag == FALSE

GSS_Unwrap for the receiver

The procedure associated with ENC commands, 632 replies, and Private file transfers is:

GSS_Wrap for the sender, with conf_flag == TRUE
GSS_Unwrap for the receiver

CONF commands and 633 replies are not supported.

Both the client and server should inspect the value of conf_avail to determine whether the peer supports confidentiality services.

When the security state is reset (when AUTH is received a second time, or when REIN is received), this should be done by calling the GSS_Delete_sec_context function.

Appendix II: Specification under Kerberos version 4

The security mechanism name (for the AUTH command) associated with Kerberos Version 4 is KERBEROS_V4. If the server supports KERBEROS_V4, it must respond with a 334 reply code indicating that an ADAT command is expected next.

The client must retrieve a ticket for the Kerberos principal "ftp.hostname@realm" by calling krb_mk_req(3) with a principal name of "ftp", an instance equal to the first part of the canonical host name of the server with all letters in lower case (as returned by krb_get_phost(3)), the server's realm name (as returned by krb_realmofhost(3)), and an arbitrary checksum. The ticket must then be base 64 encoded and sent as the argument to an ADAT command.

If the "ftp" principal name is not a registered principal in the Kerberos database, then the client may fall back on the "rcmd" principal name (same instance and realm). However, servers must accept only one or the other of these principal names, and must not be willing to accept either. Generally, if the server has a key for the "ftp" principal in its srvtab, then that principal only must be used, otherwise the "rcmd" principal only must be used.

The server must base 64 decode the argument to the ADAT command and pass the result to krb_rd_req(3). The server must add one to the checksum from the authenticator, convert the result to network byte order (most significant byte first), and sign it using krb_mk_safe(3), and base 64 encode the result. Upon success, the server must reply to the client with a 235 code and include "ADAT=base64string" in the text of the reply. Upon failure, the server should reply 535.

Upon receipt of the 235 reply from the server, the client must parse the text of the reply for the base 64 encoded data, decode it, convert it from network byte order, and pass the result to krb_rd_safe(3). The client must consider the server authenticated if the resultant checksum is equal to one plus the value previously sent.

The procedure associated with MIC commands, 631 replies, and Safe file transfers is:

```
krb_mk_safe(3) for the sender
krb_rd_safe(3) for the receiver
```

The procedure associated with ENC commands, 632 replies, and Private file transfers is:

```
krb_mk_priv(3) for the sender
krb_rd_priv(3) for the receiver
```

CONF commands and 633 replies are not supported.

Note that this specification for KERBEROS_V4 contains no provision for negotiating alternate means for integrity and confidentiality routines. Note also that the ADAT exchange does not convey whether the peer supports confidentiality services.

In order to stay within the allowed PBSZ, implementors must take note that a cleartext buffer will grow by 31 bytes when processed by krb_mk_safe(3) and will grow by 26 bytes when processed by krb_mk_priv(3).

Full Copyright Statement