CAT Working Group
<draft-ietf-cat-ftpdsaauth-00.txt>
Updates: RFC 959
Internet-Draft Expire in six months
July 1997

Russell Housley (SPYRUS)
William A. Nace (NSA)
Peter Yee (SPYRUS)

FTP Authentication Using DSA

Status of this Memo

Abstract

This document defines a method to secure file transfers using the FTP specification RFC 959, "FILE TRANSFER PROTOCOL (FTP)" (October 1985) and the work in progress document "FTP Security Extensions" <Draft-ietf-cat-ftpsec-09.txt>[1]. This method will use the extensions proposed in the "FTP Security Extensions" Draft document along with a public/private digital signature.

1 Introduction

The File Transfer Protocol (FTP) provides no protocol security except for a user authentication password which is transmitted in the clear. In addition, the protocol does not protect the file transfer session beyond the original authentication phase.

The Internet Engineering Task Force (IETF) Common Authentication Technology (CAT) Working Group has specified security extensions to FTP. These extensions allow the protocol to use more flexible security schemes, and in particular allows for various levels of protection for the FTP command and data connections. This document describes a profile for the FTP Security Extensions by which these mechanisms may be provisioned using the DSA[2] and SHA-1[3] algorithms. The FTP Security Extensions do not attempt to provide for security when FTP is used in proxy mode. The profile proposed in this document does not remove this limitation.

2 FTP Security Extensions

The IETF CAT Working Group has produced an Internet Draft that seeks to improve the security of FTP. This Internet Draft is likely to become a standards track RFC in 1997. It provides:

* user authentication -- augmenting the normal password mechanism;

* server authentication -- normally done in conjunction with user authentication;

* session parameter negotiation -- in particular, encryption keys and attributes;

* command connection protection -- integrity, confidentiality, or both;

* data transfer protection -- same as for command connection protection.

In order to support the above security services, the two FTP entities negotiate a mechanism. This process is open-ended and completes when both entities agree on an acceptable mechanism or when the initiating party (always the client) is unable to suggest an agreeable mechanism. Once the entities agree upon a mechanism, they may commence authentication and/or parameter negotiation.

Authentication and parameter negotiation occur within an unbounded series of exchanges. At the completion of the exchanges, the entities will either be authenticated (unilateral or mutually), and may be ready to protect FTP commands and data.

Following the exchanges, the entities negotiate the size of the buffers they will use in protecting the commands and data that follow. This process is accomplished in two steps: the client offers a suggested buffer size and the server may either refuse it, counter

it, or accept it.

At this point, the entities may issue protected commands within the bounds of the parameters negotiated through the security exchanges. Protected commands are issued by applying the protection services required to the normal commands and Base64 encoding the results. The encoded results are sent as the data field within a MIC (integrity). Base64 is an encoding for mapping binary data onto a textual character set that is able to pass through 7-bit systems without loss. The server sends back responses in new result codes which allow the identical protections and Base64 encoding to be applied to the results. Protection of the data transfers can be specified via the PROT command which supports the same protections as those afforded the other FTP commands. PROT commands may be sent on a transfer-by-transfer basis, however, the session parameters may not be changed within a session.

3 Use of Digital Signature Algorithm (DSA)

This paper a profile in which DSA may be used to achieve certain security services when used in conjunction with the FTP Security Extensions framework. As stated above, the reader should be familiar with the extensions in order to understand the protocol steps that follow. In the context of the FTP Security Extensions, we use DSA with SHA-1 for authentication and integrity.

3.1 DSA Profile

FTP entities may use DSA to give either unilateral or mutual authentication as well as to provide integrity services for commands and data transfers. This specification follows the tokens and exchanges defined in FIPS PUB 196[4], including Appendix A on ASN.1 encoding of messages and tokens.

3.1.1 Unilateral Authentication with DSA

A client may unilaterally authenticate its identity to a server. What follows are the protocol steps necessary to perform DSA authentication as specified in FIPS PUB 196 under the FTP Security Extensions framework. Where failure modes are encountered, the return codes follow those specified in the Extensions. They are not enumerated here as they are invariant among mechanisms. FIPS PUB 196 employs a set of exchanges which are transferred to provide authentication. Each exchange employs various fields and tokens, some of which are optional. In addition, each token has several subfields that are optional. A conformant subset of the fields and subfields for use with FTP have been selected. Therefore, the exchanges below do not show the FIPS PUB 196 notation indicating optional fields,

while only the mandatory subfields are allowed. The tokens are ASN.1 encoded per Appendix A of FIPS PUB 196, and each token is named to indicate the direction in which it flows (i.e., TokenBA flows from Party B to Party A). In Figure 1, the client binds the last transmission (token identifier, certificate, and token) together as an ASN.1 sequence.

The exchanges detailed below presume a knowledge of FIPS PUB 196 and the FTP Security Extensions. The client is Party A, while the server is Party B. The notation for concatenation is " || ". The pseudo-function Sequence is used to indicate that its parameters are to be joined as an ASN.1 SEQUENCE. Verification of signed data, and in particular certification path verification is implicitly assumed, but is not shown.

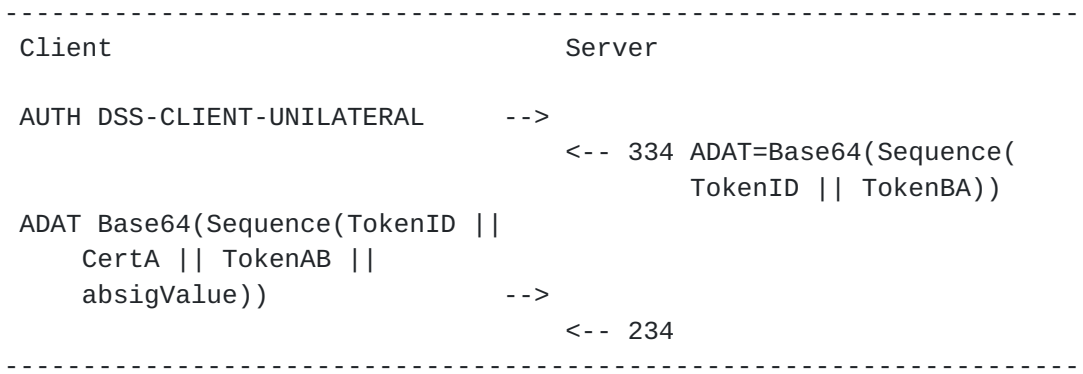

Figure 1

With this example, the client is now authenticated to the server. Additional functionality available to client and server includes the use of MIC protected commands and the ability to send signed data. The Sign function used in the figures below appends a nonce to the end of the parameter, and then computes a DSA with SHA-1 signature over the parameter followed by a nonce. The 40 octet signature value, as described in FIPS PUB 186, is composed of two 20 octet values, r followed by s. The nonce is comprised of a two octet command counter and the Ra value from TokenAB. Since both the client and server have the Ra value from TokenAB and both can calculate the command counter, the nonce is not transmitted. The command counter is initialized to the least significant two octets of the Ra value from TokenAB, and it is incremented each time the client issues a command. The Sign function output is composed of the 40 octet signature value followed by the parameter. An example follows:

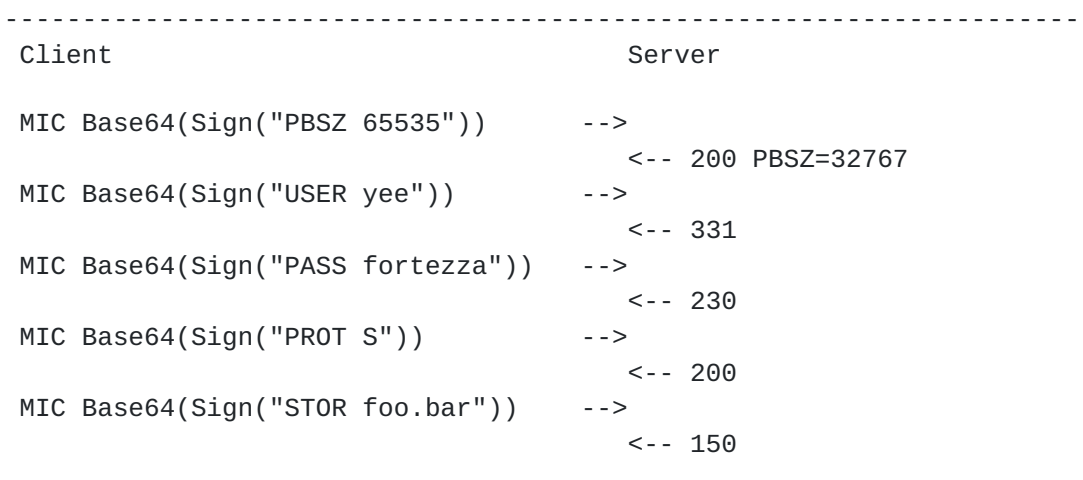

Figure 2

Data now flows from the client to the server as specified in the Extensions. Specifically, the file is broken up into blocks of data of less than the negotiated protection buffer size (32768 bytes in the example exchanges). Each protection buffer contains a safe token followed by file data. A common safe token structure is used for unilateral and mutual authentication. The safe token structure is described in section 3.1.3.

3.1.2 Mutual Authentication with DSA

The PDU flow for mutual authentication is slightly more complex, as shown:

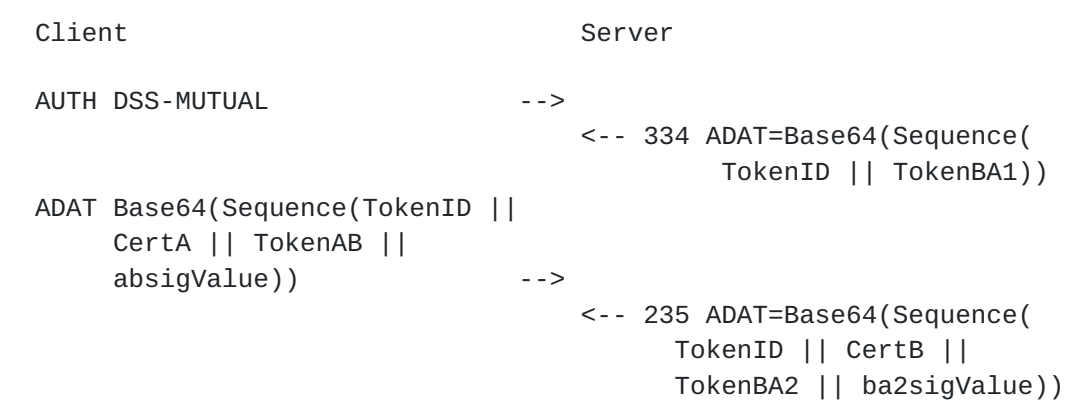

Figure 3

Data retrieval and other FTP operations can now be performed with signature protection. As before, the FTP entities negotiate a protection buffer size. Likewise, a two octet command counter combined with the Ra value from TokenAB is used as a nonce in the Sign

function.

```
------------------------------------------------------------------------
  Client                                Server

  MIC Base64(Sign("PBSZ 65535"))        -->
                                           <-- 631 Base64(Sign
                                                 ("200 PBSZ=32767"))
  MIC Base64(Sign("USER yee"))          -->
                                           <-- 631 Base64(Sign("331"))
  MIC Base64(Sign("PASS fortezza"))     -->
                                           <-- 631 Base64(Sign("230"))
  MIC Base64(Sign("PROT S"))            -->
                                           <-- 631 Base64(Sign("200"))
  MIC Base64(Sign("RETR foo.bar"))      -->
------------------------------------------------------------------------
```

Figure 4

Data now flows from the client to the server as well as the server to the client as specified in the Extensions. Specifically, the file is broken up into blocks of data of less than the negotiated protection buffer size. Each protection buffer contains a safe token followed by file data. A common safe token structure is used for unilateral and mutual authentication. The safe token structure is described in section 3.1.3.

3.1.3 File Protection with DSA

The next figure shows the structure of the safe token and file data.

```
------------------------------------------------------------------------
                      Signature         40 octets
                      Buffer Size       4 octets
                      Nonce             8 octets
                      File Count        2 octets
                      Buffer Count      4 octets
                      File Data Buffer  Buffer Size
------------------------------------------------------------------------
```

Figure 5

The safe token is comprised of the signature value, buffer size, nonce, file count, and buffer count. The signature covers the buffer size, nonce, file count, buffer count, and the file data buffer. Buffer size is the number of octets contained in file data buffer. This buffer size must be less than the negotiated PBSZ value, and the maximum buffer size is PBSZ minus 58. If the buffer size is not equal to PBSZ minus 58, then this buffer is the final buffer of the file. If the file ends on a full buffer boundary, a buffer with the

buffer size set to zero will be sent. The nonce is the least significant 64 bits of the Rb field of TokenBA1. File count is a sequence counter of protected files transferred, starting at zero. Buffer count is the number of protected buffers sent for this file, starting at zero.

4.0 Security Considerations

This entire memo is about security mechanisms. For DSA to provide the authentication discussed, the implementation must protect the private key from disclosure.

5.0 Acknowledgements

I would like to thank Todd Horting for insights gained during implementation of this specification.

6.0 References

[1] - M. Horowitz and S. J. Lunt. FTP Security Extensions. Internet-Draft <draft-ietf-cat-ftpsec-09.txt>, November, 1996.

[2] - Digital Signature Standard (DSS). FIPS Pub 186. May 19, 1994.

[3] - Secure Hash Standard. FIPS Pub 180-1. April 17, 1995.

[4] - Standard for Entity Authentication Using Public Key Cryptography. FIPS Pub 196. February 18, 1997.

7.0 Author's Address

Russell Housley
SPYRUS
PO Box 1198
Herndon, VA 20172
USA

Phone: +1 703 435-7344
Email: housley@spyrus.com

DIRNSA
Attn: X22 (W. Nace)
9800 Savage Road
Fort Meade, MD 20755-6000
USA

Phone: +1 410 859-4464
Email: WANace@missi.ncsc.mil

Peter Yee
SPYRUS
2460 N. First Street
Suite 100
San Jose, CA 95131-1023
USA

Phone: +1 408 432-8180
Email: yee@spyrus.com