Network Working Group
Request for Comments: 2951
Category: Informational

R. Housley
T. Horting
P. Yee
SPYRUS

September 2000

TELNET Authentication Using KEA and SKIPJACK

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This document defines a method to authenticate TELNET using the Key Exchange Algorithm (KEA), and encryption of the TELNET stream using SKIPJACK. Two encryption modes are specified; one provides data integrity and the other does not. The method relies on the TELNET Authentication Option.

1. Command Names and Codes

AUTHENTICATION 37

Authentication Commands:

IS	0
SEND	1
REPLY	2
NAME	3

Authentication Types:

KEA_SJ	12
KEA_SJ_INTEG	13

Modifiers:

AUTH_WHO_MASK	1
AUTH_CLIENT_TO_SERVER	0
AUTH_SERVER_TO CLIENT	1

```
   AUTH_HOW_MASK             2
   AUTH_HOW_ONE_WAY          0
   AUTH_HOW_MUTUAL           2

   ENCRYPT_MASK             20
   ENCRYPT_OFF               0
   ENCRYPT_USING_TELOPT      4
   ENCRYPT_AFTER_EXCHANGE   16
   ENCRYPT_RESERVED         20

   INI_CRED_FWD_MASK         8
   INI_CRED_FWD_OFF          0
   INI_CRED_FWD_ON           8
```

Sub-option Commands:

```
   KEA_CERTA_RA              1
   KEA_CERTB_RB_IVB_NONCEB   2
   KEA_IVA_RESPONSEB_NONCEA  3
   KEA_RESPONSEA             4
```

2. TELNET Security Extensions

TELNET, as a protocol, has no concept of security. Without negotiated options, it merely passes characters back and forth between the NVTs represented by the two TELNET processes. In its most common usage as a protocol for remote terminal access (TCP port 23), TELNET normally connects to a server that requires user-level authentication through a user name and password in the clear. The server does not authenticate itself to the user.

The TELNET Authentication Option provides for:

* User authentication -- replacing or augmenting the normal host password mechanism;
* Server authentication -- normally done in conjunction with user authentication;
* Session parameter negotiation -- in particular, encryption key and attributes;
* Session protection -- primarily encryption of the data and embedded command stream, but the encryption algorithm may also provide data integrity.

In order to support these security services, the two TELNET entities must first negotiate their willingness to support the TELNET Authentication Option. Upon agreeing to support this option, the parties are then able to perform sub-option negotiations to determine

the authentication protocol to be used, and possibly the remote user name to be used for authorization checking. Encryption is negotiated along with the type of the authentication.

Authentication and parameter negotiation occur within an unbounded series of exchanges. The server proposes a preference-ordered list of authentication types (mechanisms) that it supports. In addition to listing the mechanisms it supports, the server qualifies each mechanism with a modifier that specifies whether encryption of data is desired. The client selects one mechanism from the list and responds to the server indicating its choice and the first set of authentication data needed for the selected authentication type. The client may ignore a request to encrypt data and so indicate, but the server may also terminate the connection if the client refuses encryption. The server and the client then proceed through whatever number of iterations is required to arrive at the requested authentication.

Encryption is started immediately after the Authentication Option is completed.

3. Use of Key Exchange Algorithm (KEA)

This paper specifies the method in which KEA is used to achieve TELNET Authentication. KEA (in conjunction with SKIPJACK) [4] provides authentication and confidentiality. Integrity may also be provided.

TELNET entities may use KEA to provide mutual authentication and support for the setup of data encryption keys. A simple token format and set of exchanges delivers these services.

NonceA and NonceB used in this exchange are 64-bit bit strings. The client generates NonceA, and the server generates NonceB. The nonce value is selected randomly. The nonce is sent in a big endian form. The encryption of the nonce will be done with the same mechanism that the session will use, detailed in the next section.

Ra and Rb used in this exchange are 1024 bit strings and are defined by the KEA Algorithm [4].

The IVa and IVb are 24 byte Initialization Vectors. They are composed of "THIS IS NOT LEAF" followed by 8 random bytes.

CertA is the client's certificate. CertB is the server's certificate. Both certificates are X.509 certificates [6] that contain KEA public keys [7]. The client must validate the server's certificate before using the KEA public key it contains. Likewise, the server must validate the client's certificate before using the KEA public key it contains.

On completing these exchanges, the parties have a common SKIPJACK key. Mutual authentication is provided by verification of the certificates used to establish the SKIPJACK encryption key and successful use of the derived SKIPJACK session key. To protect against active attacks, encryption will take place after successful authentication. There will be no way to turn off encryption and safely turn it back on; repeating the entire authentication is the only safe way to restart it. If the user does not want to use encryption, he may disable encryption after the session is established.

3.1. SKIPJACK Modes

There are two distinct modes for encrypting TELNET streams; one provides integrity and the other does not. Because TELNET is normally operated in a character-by-character mode, the SKIPJACK with stream integrity mechanism requires the transmission of 4 bytes for every TELNET data byte. However, a simplified mode SKIPJACK without integrity mechanism will only require the transmission of one byte for every TELNET data byte.

The cryptographic mode for SKIPJACK with stream integrity is Cipher Feedback on 32 bits of data (CFB-32) and the mode of SKIPJACK is Cipher Feedback on 8 bits of data (CFB-8).

3.1.1. SKIPJACK without stream integrity

The first and least complicated mode uses SKIPJACK CFB-8. This mode provides no stream integrity.

For SKIPJACK without stream integrity, the two-octet authentication type pair is KEA_SJ AUTH_CLIENT_TO_SERVER | AUTH_HOW_MUTUAL | ENCRYPT_AFTER_EXCHANGE | INI_CRED_FWD_OFF. This indicates that the SKIPJACK without integrity mechanism will be used for mutual authentication and TELNET stream encryption. Figure 1 illustrates the authentication mechanism of KEA followed by SKIPJACK without stream integrity.

```
-------------------------------------------------------------------
 Client (Party A)                    Server (Party B)

                                     <-- IAC DO AUTHENTICATION

 IAC WILL AUTHENTICATION         -->

                                     <-- IAC SB AUTHENTICATION SEND
                                         <list of authentication options>
                                         IAC SE

 IAC SB AUTHENTICATION
 NAME <user name>                -->

 IAC SB AUTHENTICATION IS
 KEA_SJ
 AUTH_CLIENT_TO_SERVER |
     AUTH_HOW_MUTUAL |
     ENCRYPT_AFTER_EXCHANGE |
     INI_CRED_FWD_OFF
 KEA_CERTA_RA
 CertA||Ra IAC SE                -->

                                     <-- IAC SB AUTHENTICATION REPLY
                                         KEA_SJ
                                         AUTH_CLIENT_TO_SERVER |
                                             AUTH_HOW_MUTUAL |
                                             ENCRYPT_AFTER_EXCHANGE |
                                             INI_CRED_FWD_OFF
                                         IVA_RESPONSEB_NONCEA
                                         KEA_CERTB_RB_IVB_NONCEB
                                         CertB||Rb||IVb||
                                             Encrypt( NonceB )
                                         IAC SE

 IAC SB AUTHENTICATION IS
 KEA_SJ
 AUTH_CLIENT_TO_SERVER |
     AUTH_HOW_MUTUAL |
     ENCRYPT_AFTER_EXCHANGE |
     INI_CRED_FWD_OFF
 KEA_IVA_RESPONSEB_NONCEA
 IVa||Encrypt( (NonceB XOR 0x0C12)||NonceA )
 IAC SE                          -->
```

```
 Client (Party A)                    Server (Party B)

 <client begins encryption>
                                     <-- IAC SB AUTHENTICATION REPLY
                                         KEA_SJ
                                         AUTH_CLIENT_TO_SERVER |
                                             AUTH_HOW_MUTUAL |
                                             ENCRYPT_AFTER_EXCHANGE |
                                             INI_CRED_FWD_OFF
                                         KEA_RESPONSEA
                                         Encrypt( NonceA XOR 0x0C12 )
                                         IAC SE

                                         <server begins encryption>
```

Figure 1.

3.1.2. SKIPJACK with stream integrity

SKIPJACK with stream integrity is more complicated. It uses the SHA-1 [3] one-way hash function to provide integrity of the encryption stream as follows:

```
Set H0 to be the SHA-1 hash of a zero-length string.
Cn is the nth character in the TELNET stream.
Hn = SHA-1( Hn-1||Cn ), where Hn is the hash value
     associated with the nth character in the stream.
ICVn is set to the three most significant bytes of Hn.
Transmit Encrypt( Cn||ICVn ).
```

The ciphertext that is transmitted is the SKIPJACK CFB-32 encryption of (Cn||ICVn). The receiving end of the TELNET link reverses the process, first decrypting the ciphertext, separating Cn and ICVn, recalculating Hn, recalculating ICVn, and then comparing the received ICVn with the recalculated ICVn. Integrity is indicated if the comparison succeeds, and Cn can then be processed normally as part of the TELNET stream. Failure of the comparison indicates some loss of integrity, whether due to active manipulation or loss of cryptographic synchronization. In either case, the only recourse is to drop the TELNET connection and start over.

For SKIPJACK with stream integrity, the two-octet authentication type pair is KEA_SJ_INTEG AUTH_CLIENT_TO_SERVER | AUTH_HOW_MUTUAL | ENCRYPT_AFTER_EXCHANGE | INI_CRED_FWD_OFF. This indicates that the KEA SKIPJACK with integrity mechanism will be used for mutual authentication and TELNET stream encryption. Figure 2 illustrates the authentication mechanism of KEA SKIPJACK with stream integrity.

```
-------------------------------------------------------------------
 Client (Party A)                    Server (Party B)

                                     <-- IAC DO AUTHENTICATION

 IAC WILL AUTHENTICATION         -->

                                     <-- IAC SB AUTHENTICATION SEND
                                         <list of authentication options>
                                         IAC SE

 IAC SB AUTHENTICATION
 NAME <user name>                -->

 IAC SB AUTHENTICATION IS
 KEA_SJ_INTEG
 AUTH_CLIENT_TO_SERVER |
     AUTH_HOW_MUTUAL |
     ENCRYPT_AFTER_EXCHANGE |
     INI_CRED_FWD_OFF
 KEA_CERTA_RA
 CertA||Ra IAC SE                -->

                                     <-- IAC SB AUTHENTICATION REPLY
                                         KEA_SJ_INTEG
                                         AUTH_CLIENT_TO_SERVER |
                                             AUTH_HOW_MUTUAL |
                                             ENCRYPT_AFTER_EXCHANGE |
                                             INI_CRED_FWD_OFF
                                         IVA_RESPONSEB_NONCEA
                                         KEA_CERTB_RB_IVB_NONCEB
                                         CertB||Rb||IVb||
                                             Encrypt( NonceB )
                                         IAC SE

 IAC SB AUTHENTICATION IS
 KEA_SJ_INTEG
 AUTH_CLIENT_TO_SERVER |
     AUTH_HOW_MUTUAL |
     ENCRYPT_AFTER_EXCHANGE |
     INI_CRED_FWD_OFF
 KEA_IVA_RESPONSEB_NONCEA
 IVa||Encrypt( (NonceB XOR 0x0D12)||NonceA )
 IAC SE                          -->
```

```
 Client (Party A)                    Server (Party B)

 <client begins encryption>
                                     <-- IAC SB AUTHENTICATION REPLY
                                         KEA_SJ_INTEG
                                         AUTH_CLIENT_TO_SERVER |
                                             AUTH_HOW_MUTUAL |
                                             ENCRYPT_AFTER_EXCHANGE |
                                             INI_CRED_FWD_OFF
                                         KEA_RESPONSEA
                                         Encrypt( NonceA XOR 0x0D12 )
                                         IAC SE

                                         <server begins encryption>
```

Figure 2

4.0. Security Considerations

This entire memo is about security mechanisms. For KEA to provide the authentication discussed, the implementation must protect the private key from disclosure. Likewise, the SKIPJACK keys must be protected from disclosure.

Implementations must randomly generate KEA private keys, initialization vectors (IVs), and nonces. The use of inadequate pseudo-random number generators (PRNGs) to generate cryptographic keys can result in little or no security. An attacker may find it much easier to reproduce the PRNG environment that produced the keys, searching the resulting small set of possibilities, rather than brute force searching the whole key space. The generation of quality random numbers is difficult. RFC 1750 [8] offers important guidance in this area, and Appendix 3 of FIPS Pub 186 [9] provides one quality PRNG technique.

By linking the enabling of encryption as a side effect of successful authentication, protection is provided against an active attacker. If encryption were enabled as a separate negotiation, it would provide a window of vulnerability from when the authentication completes, up to and including the negotiation to turn on encryption. The only safe way to restart encryption, if it is turned off, is to repeat the entire authentication process.

5. IANA Considerations

The authentication types KEA_SJ and KEA_SJ_INTEG and their associated suboption values are registered with IANA. Any suboption values used to extend the protocol as described in this document must be registered with IANA before use. IANA is instructed not to issue new suboption values without submission of documentation of their use.

6.0. Acknowledgements

We would like to thank William Nace for support during implementation of this specification.

7.0. References

[1] Postel, J. and J. Reynolds, "TELNET Protocol Specification", ASTD 8, RFC 854, May 1983.

[2] Ts'o, T. and J. Altman, "Telnet Authentication Option", RFC 2941, September 2000.

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

[4] "SKIPJACK and KEA Algorithm Specification", Version 2.0, May 29, 1998. Available from http://csrc.nist.gov/encryption/skipjack-kea.htm

[5] Postel, J. and J. Reynolds, "TELNET Option Specifications", STD 8, RFC 855, May 1983.

[6] Housley, R., Ford, W., Polk, W. and D. Solo, "Internet X.509 Public Key Infrastructure: X.509 Certificate and CRL Profile", RFC 2459, January 1999.

[7] Housley, R. and W. Polk, "Internet X.509 Public Key Infrastructure - Representation of Key Exchange Algorithm (KEA) Keys in Internet X.509 Public Key Infrastructure Certificates", RFC 2528, March 1999.

[8] Eastlake, D., Crocker, S. and J. Schiller, "Randomness Recommendations for Security", RFC 1750, December 1994.

[9) National Institute of Standards and Technology. FIPS Pub 186: Digital Signature Standard. 19 May 1994.

8.0. Authors' Addresses

Russell Housley
SPYRUS
381 Elden Street, Suite 1120
Herndon, VA 20170
USA

EMail: housley@spyrus.com

Todd Horting
SPYRUS
381 Elden Street, Suite 1120
Herndon, VA 20170
USA

EMail: thorting@spyrus.com

Peter Yee
SPYRUS
5303 Betsy Ross Drive
Santa Clara, CA 95054
USA

EMail: yee@spyrus.com

9. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.