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. It relies on the Telnet Authentication Option [1].

1 Introduction

The Telnet protocol provides no protocol security. Telnet servers may require users to login. This is typically a host level login consisting of a user name and a password, transmitted in the clear.

The mechanism specified in this document relies on the Telnet
Authentication Option [1].

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 and Data Encryption Options. Upon agreeing to support these options, the parties are then able to perform subnegotiations 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) which it supports. In addition to listing the mechanisms it supports, the server qualifies each mechanism with a modifier that specifies whether the authentication is to be one-way or mutual, and in which direction the authentication is to be performed, and if 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.
If encryption is requested, it is started immediately after the Authentication options are completed. Afterwards either party may use the Data Encryption Options to turn off encryption, but once this has been disabled, there is no ability to re-enable encryption without repeating the complete authentication phase.

3 Use of Key Exchange Algorithm (KEA)

This paper specifies the method in which KEA may be used to achieve certain security services when used in conjunction with the Telnet Authentication. KEA (in conjunction with SKIPJACK) is provides authentication, integrity and confidentiality. Figure 1 illustrates the authentication mechanism.

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.

The nonce used in this exchanged is a 64 bit unsigned integer. The server generates one, and the client generates another. 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.

In figure 1 the two-octet authentication type pair is denoted by ‘XXXX’. This will be filled in with either hexadecimal ‘0C06’ for KEA SKIPJACK without integrity mechanism or ‘0D06’ for KEA SKIPJACK with integrity mechanism.
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 from an active attacker, 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 will have to logoff and logon with the desired security mechanism.
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 KEA SKIPJACK with stream integrity mechanism requires the transmission of 4 bytes for every telnet data byte. However, a more simplified mode KEA SKIPJACK without integrity mechanism will only require the transmission of one byte for every telnet data byte.

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

3.1.1 SKIPJACK without stream integrity

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

For SKIPJACK without stream integrity, the two-octet authentication type pair (XXXX in figure 1) is "KEA_SJ CLIENT_TO_SERVER MUTUAL ENCRYPT_ON INI_CRED_FWD_OFF". This is encoded as two-octets: ‘0C06’ in hexadecimal. This indicates that the KEA SKIPJACK without integrity mechanism will be used for mutual authentication and telnet stream encryption.

3.1.2 SKIPJACK with stream integrity

SKIPJACK with stream integrity is more complicated. It uses a hash function to bind integrity into the encryption stream as follows:

Set H0 to be the SHA-1 [3] hash of a zero length string.
Cn is the nth character in the 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 (XXXX in figure 1) is "KEA_SJ_INTEG_CLIENT_TO_SERVER MUTUAL ENCRYPT_ON INI_CRED_FWD_OFF". This is encoded as two-octets: ‘0D06’ in hexadecimal. This indicates that the KEA SKIPJACK with integrity mechanism will be used for mutual authentication and telnet stream encryption.

3.1.3 Telnet SYNCH Handling

Telnet supports a "Synch" mechanism to solve the problem of Telnet control commands from being blocked by network flow control. Basically, the sender of the Synch is telling the recipient to discard any incoming data up to a Telnet DATA MARK (DM), searching the data stream for TELNET IP, AO, or AYT commands. The Synch signal is sent via a TCP Urgent notification, and therefore is likely to arrive out of sequence from other data that may have already been transmitted but not received.

The implication for a secured Telnet implementation is that for many encryption algorithms, maintenance of cryptographic synchronization requires special care. The TCP Urgent notification will consist of a single-use IV generated specifically for the Synch signal, and the required DATA MARK terminating octet, encrypted under this IV. The recipient of the TCP Urgent notification first saves its current decryption data, and then uses the IV to decrypt the DATA MARK in the notification. The recipient then restores its decryption state, and continues decrypting the data stream, now looking for certain Telnet commands, up to the next DATA MARK. The appendix (see Figure 5) makes this clear by showing the specific CI library calls that the sender and recipient invoke in order to perform Synch handling.

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 form disclosure.

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.
5.0 Acknowledgements

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

6.0 References


7.0 Author’s Address

Russell Housley
SPYRUS
PO Box 1198
Herndon, VA 20172
USA
Phone: +1 703 435-7344
Email: housley@spyrus.com

Todd Horting
SPYRUS
PO Box 1198
Herndon, VA 20172
USA
Phone: +1 703 435-4711
Email: thorthing@spyrus.com

Peter Yee
SPYRUS
2460 N. First Street
Suite 100
San Jose, CA 95131-1023
USA
Phone: +1 408 432-8180
Email: yee@spyrus.com