Internet Engineering Task Force
INTERNET-DRAFT draft-ietf-tn3270e-telnet-tls-02.txt
expires January 2000

Michael Boe
Cisco
Systems

July, 1999

TLS-based Telnet Security

Abstract

Telnet service has long been a standard Internet protocol. However, a standard way of ensuring privacy and integrity of Telnet sessions has been lacking. This document proposes a standard method for Telnet servers and clients to use the Transport Layer Security (TLS) protocol. It describes how two Telnet participants can decide whether or not to attempt TLS negotiation, and how the two participants should process authentication credentials exchanged as a part of TLS startup.

Changes since -01 Draft

- o Drop possibility of a continuing a Telnet session if the TLS negotiation fails.

- Assert that server sending DO STARTTLS must be willing to negotiate a TLS session

- Change SHOULD to MUST with respect to a server requesting a client certificate.

- Add paragraph on commonName to section on check of X.509 certificate.

- Sharpen language concerning notification of possible server-certificate mismatch.

- drop place-holder section on Kerberos 5 security; replace with section on non-PKI-based authentication (after TLS negotiation).

- Prohibit fallback to SSL 2.0.

- Give more details about how authentication-after-TLS-negotiation can be achieved.

- Simplify post-TLS Telnet negotiation state-assumptions by resetting them to initial-state.

Changes since -00 Draft

- Add local authentication/authorization operational model.

- Change behavior of Telnet machine to reset at start of TLS negotiation.

- Insert narrative questioning the utility of allowing continuation of Telnet session after TLS has ended.

- change examples to reflect the above changes.

- Fix several typos.

Contents

1   Introduction

We are interested in addressing the interaction between the Telnet client and server that will support this secure requirement with the knowledge that this is only a portion of the total end-user to application path. Specifically, it is often true that the Telnet server does not reside on the target machine (it does not have access to a list of identities which are allowed to access to that mainframe), and it is often true (e.g. 3270 access) that the TN server can not even identify that portion of the emulation stream which contains user identification/password information. Additionally, it may be the case that the Telnet client is not co-resident with the end user and that it also may be unable to identify that portion of the data stream that deals with user identity. We make the assumption here that there is a trust relationship and appropriate protection to support that relationship between the TN Server and the ultimate application engine such that data on this path is protected and that the application will authenticate the end user via the emulation stream as well as use this to control access to information. We further make the assumption that the path between the end user and the client is protected.

To hold up the Telnet part of the overall secure path between the user and the mainframe, the Telnet data stream must appear unintelligible to a third party. This is done by creating a shared secret between the client and server. This shared secret is used to encrypt the flow of data and (just as important) require the client to verify that it is talking to correct server (the one that the mainframe trusts rather than an unintended man-in-the-middle) with the knowledge that the emulation stream itself will be used by the mainframe to verify the identity of the end-user. Rather than create a specialized new protocol which accomplishes these goals we instead have chosen to use existing protocols with certain constraints.

One such existing protocol is the TLS protocol (formerly known as SSL). And the Telnet [TELNET] application protocol can certainly benefit from the use of TLS. Other security mechanisms have been used with Telnet (e.g. KERBEROS [RFC1416]), but TLS offers a broad range of security levels that allow sites to proceed at an "evolutionary" pace in deploying authentication, authorization and confidentiality policies, databases and distribution methods.

TLS is used to provide the following:

- creation and refresh of a shared secret;

- negotiation and execution of data encryption and optional

compressesion;

- o primary negotiation of authentication; and, if chosen

- o execution of public-key or symmetric-key based authentication.

Since both encryption and authentication is always needed for a secure channel that meets the requirements of these emulation types, we can use the anonymous authentication negotiation option of TLS as an indication that the client wants to negotiate some non-TLS-based authentication exchange. Note that as per usual TLS rules, the client must always authenticate the server's credentials.

TLS at most offers only authentication of the peers conducting the TLS dialog. In particular, it does not offer the possibility of the client providing separate credentials for authorization than were presented for authentication. It is expect that other RFCs will be produced describing how other authentication mechanisms can be used in conjunction with TLS.

Traditional Telnet servers have operated without such early presentation of authorization credentials for many reasons (most of which are historical). However, recent developments in Telnet server technology make it advantageous for the Telnet server to know the authorized capabilities of the remote client before choosing a communications link (be it `pty' or SNA LU) and link-characteristics to the host system (be that "upstream" link local or remote to the server). Thus, we expect to see the use of client authorization to become an important element of the telnet evolution. Such authorization methods may require certificates presented by the client via TLS, or by the use of SASL or RFC1416, or some other as yet unstandardized method.

This document defines extensions to Telnet which allow TLS to be activated early in the lifetime of the Telnet connection. It defines a set of advisory security-policy response codes for use when negotiating TLS over Telnet.

Conventions Used in this Document
The key words "REQUIRED", "MUST", "MUST NOT", "SHOULD", "SHOULD NOT" and "MAY" in this document are to be interpreted as described in [KEYWORDS].

Formal syntax is defined using ABNF [ANBF].

In examples, "C:" and "S:" indicate lines sent by the the client and server, respectively.

2 Telnet STARTTLS Option and Subnegotiation

The STARTTLS option is an asymmetric option, with only the server side being able to send IAC DO STARTTLS. The client may initiate by sending IAC WILL STARTTLS. There are some more rules due to the need to clear the link of data (or to synchronize the link). This synchronization takes the form of a three-way handshake:

1. As per normal Telnet option processing rules, the client MUST respond to the server's IAC DO STARTTLS with either IAC WONT STARTTLS or IAC WILL STARTTLS (if it hasn't already done so). An affirmative response MUST be followed eventually by IAC SB STARTTLS FOLLOWS IAC SE. If the client sends the affirmative response, it must then not initiate any further Telnet options or subnegotiations except for the STARTTLS subnegotiation until after the TLS negotiation has completed.

2. The server SHOULD NOT send any more Telnet data or commands after sending IAC DO STARTTLS except in response to client Telnet options received until after it receives either a negative response from the client (IAC WONT STARTTLS) or the affirmative (both IAC WILL STARTTLS and followed eventually by IAC SB STARTTLS FOLLOWS IAC SE). If the client's STARTTLS option response is negative, the server is free to again send Telnet data or commands. If the client's response is affirmative, then the server MUST send only IAC SB STARTTLS FOLLOWS IAC SE. If the server sends IAC SB STARTTLS FOLLOWS IAC SE, then the server MUST also arrange at this time, for the normal Telnet byte-stuff/destuff processing to be turned off for the duration of the TLS negotiation.

3. The client, upon receipt of the server's IAC SB STARTTLS FOLLOWS IAC SE, MUST also arrange for normal Telnet byte-stuff/destuff processing to be turned off for the duration of the TLS negotiation. It MUST then enter into TLS negotiation by sending the TLS ClientHello message.

Here's a breakdown of the Telnet command phrases defined in this document:

S: IAC DO STARTTLS-- Sent only by server. Indicates that server strongly desires that the client enter into TLS negotiations.

C: IAC WILL STARTTLS-- Sent only by client. Indicates that client strongly desires that the server enter into TLS negotiations.

S: IAC DONT STARTTLS-- Sent only by the server. Indicates that the server is not willing to enter into TLS negotations.

C: IAC WONT STARTTLS--   Sent only by the client. Indicates that the client is not willing to enter into TLS negotiations.

C: IAC SB STARTTLS FOLLOWS IAC SE--   When sent by the client, this indicates that the client is preparing for TLS negotiations, and that the next thing sent by the client will be the TLS ClientHello. Also indicates that the client has reset its Telnet state.

S: IAC SB STARTTLS FOLLOWS IAC SE--   When sent by the server, this indicates that the server has prepared to receive the TLS ClientHello from the client. Also indicates that the server has reset its Telnet state.

2.1 Abnormal Negotiation Failure

The behavior regarding TLS negotiation failure is covered in [TLS], and does not indicate that the TCP connection be broken; the semantics are that TLS is finished and all state variables cleaned up. The TCP connection may be retained.

However, it's not clear that either side can detect when the last of the TLS data has arrived. So if TLS negotiation fails, the TCP connection SHOULD be reset and the client MAY reconnect. To avoid infinite loops of TLS negotiation failures, the client MUST remember to not negotiate TLS upon reconnecting after a TLS negotiation failure.

2.2 Assigned Values for STARTTLS Negotiation

The assigned value of the Telnet STARTTLS option octet is 46. The assigned value of the FOLLOWS subnegotiation octet is 1. In ABNF, this is:

```
    STARTTLS = %d46
    FOLLOWS  = %d1
```

3 Telnet Authentication and Authorization

Telnet servers and clients can be implemented in a variety of ways that impact how clients and servers authenticate and authorize each other.

However, most (if not all) the implementations can be abstracted via the following four communicating processes:

SES Server End System. This is an application or machine to which client desires a connection. Though not illustrated here, a single Telnet connection between client and server could have multiple SES terminations.

Server The Telnet server.

Client The Telnet client, which may or may not be co-located with the CES. The Telnet client in fact be a gateway or proxy for downstream clients; it's immaterial.

CES Client End System. The system communicating with the Telnet Client. There may be more than one actual CES communicating to a single Telnet Client instance; this is also immaterial to how Client and Server can sucessfully exchange authentication and authorization details. However, see Section 5.4 for a discussion on trust implications.

What is of interest here is how the Client and Server can exchange authentication and authorization details such that these components can direct Telnet session traffic to authorized End Systems in a reliable, trustworthy fashion.

What is beyond the scope of this specification are several related topics, including:

- How the Server and SES are connected, and how they exchange data or information regarding authorization or authentication (if any).

- How the Client and CES are connected, and how they exchange data or information regarding authorization or authentication (if any).

4   TLS, Authentication and Authorization

System-to-system communications using the Telnet protocol have traditionally used no authentication techniques at the Telnet level. More recent techniques have used Telnet to transport authentication exchanges (e.g.[TLSKERB]). In none of these systems, however, is a remote system allowed to assume more than one identity once the Telnet preamble negotiation is over and the remote is connected to the application-endpoint. The reason for this is that the local party must in some way inform the end-system of the remote party's identity (and perhaps authorization). This process must take place before the remote party starts communicating with the end-system. At that point it's too late to change what access a client may have to an server end-system: that end-system has been selected, resources have been allocated and capability restrictions set.

[Author's note: I dislike providing state models of how local systems should behave in relation to a wire protocol; it seems to me that such models often do more harm than good when describing a wire protocol. The model often misrepresents reality, and/or makes unnecessarily limiting assumptions about implementation behavior. However, the alternative methods of explanation seemed even less useful in this case....]

This process of authentication, authorization and resource allocation can be modeled (one hopes!) by the following simple set of states and transitions:

`unauthenticated'   The local party has not received any credentials offered by the remote. A new Telnet connection starts in this state.

The `authenticating' state will be entered from this state if the local party initiates the authentication process of the peer. The Telnet STARTTLS negotiation is considered an initiation of the authentication process.

The `authorizing' state will be entered from this state either if the local party decides to begin authorization and resource allocation procedures unilaterally...or if the local party has received data from the remote party destined for local end-system.

`authenticating'   The local party has received at least some of the credentials needed to authenticate its peer, but has not finished the process.

The `authenticated' state will be entered from this state if the local party is satisfied with the credentials proferred by the client.

The `unauthenticated' state will be entered from this state if the local party cannot verify the credentials proffered by the client or if the client has not proffered any credentials. Alternately, the local party may terminate the Telnet connection instead of returning it to the `unauthenticated' state.

`authenticated'   The local party has authenticated its peer, but has not yet authorized the client to connect to any end-system resources.

The `authenticating' state will be entered from this state if the local party decides that further authentication of the client is warranted.

The `authorizing' state will be entered from this state if the local party either initiates authorization dialog with the client (or engages in some process to authorize and allocate resources on behalf of the client), or has received data from the remote party destined for a local end-system.

`authorizing'   The local party is in the process of authorizing its peer to use end-system resources, or may be in the process of allocating or reserving those resources.

The `transfer-ready' state will be entered when the local party is ready to allow data to be passed between the local end-system and remote peer.

The `authenticated' state will be entered if the local party determines that the current authorization does not allow any access to a local end-system. If the remote peer is not currently authenticated, then the `unauthenticated' state will be entered instead.

`transfer-ready'   The party may pass data between the local end-system to its peer.

The `authorizing' state will be entered if the local party (perhaps due to a request by the remote peer) deallocates the communications resources to the local-end system. Alternately, the local party may enter the `authenticated' or the `unauthenticated' state.

In addition to the "orderly" state transitions noted above, some extraordinary transitions may also occur:

1. The absence of a guarantee on the integrity of the data stream between the two Telnet parties also removes the guarantee that the remote peer is who the authentication credentials say the peer is. Thus, upon being notified that the Telnet session is no longer using an integrity layer, the local party must at least deallocate all resources associated with a Telnet connection which would not have been allocable had the remote party never authenticated itself.

   In practice, this deallocation-of-resources restriction is hard to interpret consistently by both Telnet endpoints. Therefore, both parties MUST return to the initial Telnet state after negotiation of TLS. That is, it is as if the Telnet session had just started.

This means that the states may transition from whatever the current state is to `unauthenticated'. Alternately, the local party may break the Telnet connection instead.

2. If the local party is notified at any point during the Telnet connection that the remote party's authorizations have been reduced or revoked, then the local party must treat the remote party as being unauthenticated. The local party must deallocate all resources associated with a Telnet connection which would not have been allocable had the remote party never authenticated itself.

   This too may mean that the states may transition from whatever the current state is to `unauthenticated'. Alternately, the local party may break the Telnet connection instead.

The above model explains how each party should handle the authentication and authorization information exchanged during the lifetime of a Telnet connection. It is deliberately fuzzy as to what constitutes internal processes (such as "authorizing") and what is meant by "resources" or "end-system" (such as whether an end-system is strictly a single entity and communications path to the local party, or multiples of each, etc).

Here's a state transition diagram, as per [RFC2360]:

	0	1	2	3	4
Events	unauth	auth'ing	auth'ed	authorizing	trans-ready
auth-needed	sap/1	sap/1	sap/1	sap/1	der,sap/1
auth-accept	-	ain/2	-	-	-
auth-bad	-	0	wa/0	wa,der/0	der,sap/1
authz-start	szp/3	-	szp/3	-	-
data-rcvd	szp/3	qd/1	szp/3	qd/3	pd/4
authz-ok	-	-	-	4	-
authz-bad	-	-	-	der/2	wa,der,szp/3

Action	Description
sap	start authentication process
der	deallocate end-system resources
ain	authorize if needed
szp	start authorization process
qd	queue incoming data
pd	process data

wa	wipe authorization info

Event	Description
auth-needed	authentication deemed needed by local party
auth-accept	remote party's authentication creds accepted
auth-bad	remote party's authentication creds rejected or expired
authz-start	local or remote party starts authorization proceedings
data-rcvd	data destined for end-system received from remote party
authz-ok	authorization and resource allocation succeeded
authz-bad	authorization or resource allocation failed or expired

4.1 Authentication of the remote party

If TLS has been successfully negotiated, the client will have the server's certificate. This indicates that the server's identity can be verified. Client implementations MUST always verify the server identity as part of TLS processing and fail the connection if such verification fails. See Section 5.1 for a general discussion of the server authentication apropos Telnet clients.

The server, however, may not have the client's identity (verified or not). This is because the client need not provide a certificate during the TLS exchange. Or it may be server site policy not to use the identity so provided. In any case, the server may not have enough confidence in the client to move the connection to the authenticated state.

If further client, server or client-server authentication is going to occur after TLS has been negotiated, it MUST occur before any non-authentication-related Telnet interactions take place on the link after TLS starts. When the first non-authentication-related Telnet interaction is received by either participant, then the receiving participant MAY drop the connection due to dissatisfaction with the level of authentication.

If the server wishes to request a client certificate after TLS is initially started (presumably with no client certificate requested), it may do so. However, the server MUST make such a request immediately after the initial TLS handshake is complete.

No TLS negotiation outcome, however trustworthy, will by itself provide the server with the authorization identity if that is different from the authentication identity of the client. See [SASL] for why this might be

a desirable function to have with proxy clients, etc. How such authorization is done is outside the scope of this document.

The following subsections detail how the client can provide the server with authentication and authorization credentials separate to the TLS mechanism.

4.2 PKI-based Authentication and certificate extensions

When PKI authentication is used no special X.509 certificate extensions will be require but a client or server may choose to support an extension if found. For example, they may use a contained certificate revocation URL extension if provided. The intent is to allow sharing of certificates with other services on the same host, for example, a Telnet server might use the same certificate to identify itself that a co-located WEB server uses. The method of sharing certificates is outside the scope of this document.

An X.509 certificate received by a client may include the `subjectAltName' extension. If this extension is present and contains a `dNSName' object, then the client MUST check the hostname used to connect to the host against the dNSName found in the certificate. The method used is outlined in [TLSHTTP].

If the subjectAltName extension is not present, then the client may use the commonName field as long as the contents of the field are interpreted as fully qualified domain names.

4.3 Pre-TLS Authentication

It could be that the server had moved the Telnet connection to the authenticated state sometime previous to the negotiation of TLS. If this is the case, then the server MUST NOT use the credentials proffered by the client during the TLS negotiations for authorization of that client. The server should, of course, verify the client's TLS-proffered credentials.

5   Security

Security is discussed throughout this document. Most of this document concerns itself with wire protocols and security frameworks. But in this section, client and server implementation security issues are in focus.

5.1   Authentication of the Server by the Client

How the client can verify the server's proferred identity varies according to the key-exchange algorithm used in the selected TLS cipher-suite. However, it's important for the client to follow good security practice in verifying the proffered identity of the server.

5.1.1   PKI-based certificate processing

When PKI authentication is used no special X.509 certificate extensions will be required but a client or server may choose to support an extension if found. For example, they may use a contained certificate revocation URL extension if provided. The intent is to allow sharing of certificates with other services on the same host, for example, a Telnet server might use the same certificate to identify itself that a co-located WEB server uses. The method of sharing certificates is not a topic of this document.

The verification of the server's certificate by the client MUST include, but isn't limited to, the verification of the signature certificate chain to the point where the a signature in that chain uses a known good signing certificate in the clients local key chain. The verification SHOULD then continue with a check to see if the fully qualified host name which the client connected to appears anywhere in the server's certificate subject (DN). If no match is found then either:

- o the end user MUST see a display of the server's certificate and be asked if he/she is willing to proceed with the session; or,

- o the end user MUST NOT see a display of server's certificate, but the certificate details are logged on whatever media is used to log other session details. This option may be preferred to the first option in environments where the end-user cannot be expected to make an informed decision about whether a mismatch is harmful. The connection MUST be closed automatically by the client UNLESS the client has been configured to explicitly allow all mismatches.

- o the connection MUST be closed on the user's behalf, and an error describing the mismatch logged to stable storage.

If the client side of the service is not interactive with a human end-user, the Telnet connection SHOULD be dropped if this host check fails.

5.2 Non-PKI-based Authentication of client or server

Authentication of the client (or further mutual authentication between client and server) can be accomplished via non-PKI means, too. Implementors should be careful to remember that authentication must occur after successful TLS session negotiation. Further, the timing of the authentication MUST be as per Section 4.

Practically speaking, this means that non-TLS authentication should immediately follow the TLS negotiation. See 7.4 for an example of how this can work.

5.3 Display of security levels

The Telnet client and server MAY, during the TLS protocol negotiation phase, choose to use a weak cipher suite due to policy, law or even convenience. It is, however, important that the choice of weak cipher suite be noted as being commonly known to be vulnerable to attack. In particular, both server and client software should note the choice of weak cipher-suites in the following ways:

- o If the Telnet endpoint is communicating with a human end-user, the user-interface SHOULD display the choice of weak cipher-suite and the fact that such a cipher-suite may compromise security.

- o The Telnet endpoints SHOULD log the exact choice of cipher-suite as part of whatever logging/accounting mechanism normally used.

5.4 Trust Relationships and Implications

Authentication and authorization of the remote Telnet party is useful, but can present dangers to the authorizing party even if the connection between the client and server is protected by TLS using strong encryption and mutual authentication. This is because there are some trust-relationships assumed by one or both parties:

- o Each side assumes that the authentication and authentication details proferred by the remote party stay constant until explicitly changed (or until the TLS session is ended).

- o More stringently, each side trusts the other to send a timely notification if authentication or authorization details of the other party's end system(s) have changed.

Either of these trust relationships may be false if an intruder has breached communications between a client or server and its respective end system. And either may be false if a component is badly implemented or configured. Implementers should take care in program construction to avoid invalidating these trust relationships, and should document to configuring-users the proper ways to configure the software to avoid invalidation of these relationships.

6 TLS Variants and Options

TLS has different versions and different cipher suites that can be supported by client or server implementations. The following subsections detail what TLS extensions and options are mandatory. The subsections also address how TLS variations can be accommodated.

6.1 Support of previous versions of TLS

TLS has its roots in SSL 2.0 and SSL 3.0. Server and client implementations may wish to support for SSL 3.0 as a fallback in case TLS **3.1 or higher is not supported. This is permissible; however, client** implementations which negotiate SSL3.0 MUST still follow the rules in Section 5.3 concerning disclosure to the end-user of transport-level security characteristics.

Negotiating the use of SSL 3.0 is done as part of the TLS negotiation; it is detailed in [TLS]. Negotiating SSL 2.0 is MUST NOT be attempted.

6.2 Using Kerberos V5 with TLS

If the client and server are both amenable to using Kerberos V5, then using non-PKI authentication techniques within the confines of TLS may be acceptable (see [TLSKERB]). Note that clients and servers are under no obligation to support anything but the cipher-suite(s) mandated in [TLS]. However, if implementations do implement the KRB5 authentication as a part of TLS ciphersuite, then these implementations SHOULD support at least the TLS_KRB5_WITH_3DES_EDE_CBC_SHA ciphersuite.

7 Protocol Examples

The following sections provide skeletal examples of how Telnet clients and servers can negotiate TLS.

7.1 Successful TLS negotiation

The following protocol exchange is the typical sequence that starts TLS:

```
// typical successful opening exchange
  S: IAC DO STARTTLS
  C: IAC WILL STARTTLS IAC SB STARTTLS FOLLOWS IAC SE
  S: IAC SB STARTTLS FOLLOWS IAC SE
// server now readies input stream for non-Telnet, TLS-level negotiation
  C: [starts TLS-level negotiations with a ClientHello]
  [TLS transport-level negotiation ensues]
  [TLS transport-level negotiation completes with a Finished exchanged]
// either side now able to send further Telnet data or commands
```

7.2 Successful TLS negotiation, variation

The following protocol exchange is the typical sequence that starts TLS, but with the twist that the (TN3270E) server is willing but not aggressive about doing TLS; the client strongly desires doing TLS.

```
// typical successful opening exchange
  S: IAC DO TN3270E
  C: IAC WILL STARTTLS IAC
  S: IAC DO STARTTLS
  C: IAC WILL STARTTLS IAC SB STARTTLS FOLLOWS IAC SE
  S: IAC SB STARTTLS FOLLOWS IAC SE
// server now readies input stream for non-Telnet, TLS-level negotiation
  C: [starts TLS-level negotiations with a ClientHello]
  [TLS transport-level negotiation ensues]
  [TLS transport-level negotiation completes with a Finished
                 exchanged]
// note that server retries negotiation of TN3270E after TLS
// is done.
  S: IAC DO TN3270E
  C: IAC WILL TN3270E
// TN3270E dialog continues....
```

7.3 Unsuccessful TLS negotiation

This example assumes that the server does not wish to allow the Telnet session to proceed without TLS security; however, the client's version of TLS does not interoperate with the server's.

```
//typical unsuccessful opening exchange
  S: IAC DO STARTTLS
  C: IAC WILL STARTTLS IAC SB STARTTLS FOLLOWS IAC SE
  S: IAC SB STARTTLS FOLLOWS IAC SE
// server now readies input stream for non-Telnet, TLS-level negotiation
  C: [starts TLS-level negotiations with a ClientHello]
  [TLS transport-level negotiation ensues]
  [TLS transport-level negotiation fails with server sending
               ErrorAlert message]
  C: IAC DO TERMINAL-TYPE
  S: IAC WONT TERMINAL-TYPE
  S: [TCP level disconnect]
//  server (or both) initiate TCP session disconnection
```

This example assumes that the server wants to do TLS, but is willing to allow the session to proceed without TLS security; however, the client's version of TLS does not interoperate with the server's.

```
//typical unsuccessful opening exchange
  S: IAC DO STARTTLS
  C: IAC WILL STARTTLS IAC SB STARTTLS FOLLOWS IAC SE
  S: IAC SB STARTTLS FOLLOWS IAC SE
// server now readies input stream for non-Telnet, TLS-level negotiation
  C: [starts TLS-level negotiations with a ClientHello]
  [TLS transport-level negotiation ensues]
  [TLS transport-level negotiation fails with server sending
               ErrorAlert message]
  C: IAC DO TERMINAL-TYPE
  S: IAC WILL TERMINAL-TYPE
// regular Telnet dialog ensues
```

7.4   Authentication via Kerberos 4 after TLS negotiation

Here's an implementation example of using Kerberos 4 to authenticate the client after encrypting the session with TLS. Note the following details:

- The client strictly enforces a security policy of proposing Telnet AUTH first, but accepting TLS. This has the effect of producing a rather verbose pre-TLS negotiation sequence; however, the end-result is correct. A more efficient pre-TLS sequence can be obtained by changing the client security policy to be the same as the server's for this connection (and implementing policy-aware negotiation code in the Telnet part of the client).

  A similar efficient result can be obtained even in the absence of a clear client security policy if the client has cached server security preferences from a previous Telnet session to the same server.

- The server strictly enforces a security policy of proposing TLS first, but falling back to Telnet AUTH.

```
  C: IAC WILL AUTHENTICATION
  C: IAC WILL NAWS
  C: IAC WILL TERMINAL-TYPE
  C: IAC WILL NEW-ENVIRONMENT
  S: IAC DO STARTTLS
  C: IAC WILL STARTTLS
  C: IAC SB STARTTLS FOLLOWS IAC SE
  S: IAC DO AUTHENTICATION
  S: IAC DO NAWS
  S: IAC WILL SUPPRESS-GO-AHEAD
  S: IAC DO SUPPRESS-GO-AHEAD
  S: IAC WILL ECHO
  S: IAC DO TERMINAL-TYPE
  S: IAC DO NEW-ENVIRONMENT
  S: IAC SB AUTHENTICATION SEND
    KERBEROS_V4 CLIENT_TO_SERVER|MUTUAL|ENCRYPT_REQ
    KERBEROS_V4 CLIENT_TO_SERVER|MUTUAL
    KERBEROS_V4 CLIENT_TO_SERVER|ONE_WAY
    SSL CLIENT_TO_SERVER|ONE_WAY   IAC SE
  S: IAC SB TERMINAL-TYPE SEND  IAC SE
  S: IAC SB NEW-ENVIRONMENT SEND  IAC SE
  S: IAC SB STARTTLS FOLLOWS  IAC SE
  [TLS - handshake starting]
  [TLS - OK]
  C: IAC WILL AUTHENTICATION
  C: IAC WILL NAWS
  C: IAC WILL TERMINAL-TYPE
  C: IAC WILL NEW-ENVIRONMENT
  <wait for outstanding negotiations>
  S: IAC DO AUTHENTICATION
  S: IAC DO NAWS
  S: IAC WILL SUPPRESS-GO-AHEAD
  C: IAC DO SUPPRESS-GO-AHEAD
  S: IAC DO SUPPRESS-GO-AHEAD
  C: IAC WILL SUPPRESS-GO-AHEAD
  S: IAC WILL ECHO
  C: IAC DO ECHO
  S: IAC DO TERMINAL-TYPE
  S: IAC DO NEW-ENVIRONMENT
  S: IAC SB AUTHENTICATION SEND
    KERBEROS_V4 CLIENT_TO_SERVER|MUTUAL
    KERBEROS_V4 CLIENT_TO_SERVER|ONE_WAY   IAC SE
  C: IAC SB AUTHENTICATION NAME jaltman IAC SE
  C: IAC SB AUTHENTICATION IS
    KERBEROS_V4 CLIENT_TO_SERVER|MUTUAL AUTH
    04 07 0B "CC.COLUMBIA.EDU" 00 "8(" 0D 9E 9F AB A0 "L" 15 8F A6
    ED "x" 19 F8 0C "wa" CA "z`" 1A E2 B8 "Y" B0 8E "KkK" C6 AA "<" FF
    FF 98 89 "|" 90 AC DF 13 "2" FC 8E 97 F7 BD AE "e" 07 82 "n" 19 "v"
    7F 10 C1 12 B0 C6 "|" FA BB "s1Y" FF FF 10 B5 14 B3 "(" BC 86 "`"
    D2 "z" AB "Qp" C4 "7" AB "]8" 8A 83 B7 "j" E6 "IK" DE "|YIVN"
    IAC SE
  S: IAC SB TERMINAL-TYPE SEND  IAC SE
  S: IAC SB NEW-ENVIRONMENT SEND  IAC SE
  S: IAC SB AUTHENTICATION REPLY
   KERBEROS_V4 CLIENT_TO_SERVER|MUTUAL ACCEPT  IAC SE
  C: IAC SB AUTHENTICATION IS
   KERBEROS_V4 CLIENT|MUTUAL CHALLENGE "[" BE B7 96 "j" 92 09 "~" IAC SE
  S: IAC SB AUTHENTICATION REPLY
   KERBEROS_V4 CLIENT_TO_SERVER|MUTUAL RESPONSE "df" B0 D6 "vR_/"  IAC SE
<no outstanding negotiations>
  C: IAC SB TERMINAL-TYPE IS VT320 IAC SE
  C: IAC SB NEW-ENVIRONMENT IS USERAjaltmanSYSTEMTYPEAWIN32 IAC SE
  C: IAC SB NAWS 162 49 IAC SE
```

Here are several things to note about the above example:

- After TLS is successfully negotiated, all non-TLS Telnet settings are forgotten and must be renegotiated.

- After TLS is successfully negotiated, the server offers all authentication types that are appropriate for a session using TLS. Note that the server, post TLS-negotiation, isn't offering Telnet ENCRYPT or AUTH SSL, since (a) it's useless and perhaps dangerous to encrypt twice, and (b) TLS and/or SSL can be applied only once to a Telnet session.

References

[ANBF] D. Crocker, Ed., P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC2235, November 1997.

[KEYWORDS] Bradner, S. "Key words for use in RFCs to Indicate Requirement Levels", RFC2119, March 1997.

[RFC927] Brian A. Anderson. "TACACS User Identification Telnet Option", RFC927, December 1984

[RFC1416] D. Borman, Editor. "Telnet Authentication Option", RFC1416, February 1993.

[SASL] Myers, J. "Simple Authentication and Security Layer (SASL)", RFC2222, October 1997.

[RFC2360] G. Scott, Editor. "Guide for Internet Standard Writers", RFC2360, June 1998.

[TELNET] J. Postel, J. Reynolds. "Telnet Protocol Specifications", RFC854, May 1983.

[TLS]   Tim Dierks. "The TLS Protocol", Internet Draft, November 1997.

[TLSKERB]   Ari Medvinsky, Matthew Hur. "Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)", Internet Draft, July 1997.

[TLSHTTP]   E. Rescorla. "HTTP Over TLS", Internet Draft <draft-ietf-tls-https-01.txt>, March 1998.

Author's Address

Michael Boe
Cisco Systems Inc.
170 West Tasman Drive
San Jose, CA 95134

Email: Michael Boe <mboe@cisco.com>

Full Copyright Statement

Acknowledgement

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