STEL: Secure TELnet

David Vincenzetti
Stefano Taino
Fabio Bolognesi

{vince || taino || bolo}@dsi.unimi.it

CERT-IT
Computer Emergency Response Team ITaly
Department of Computer Science
University of Milan
ITALY

June 1995 †

1. Abstract

Eavesdropping is becoming rampant on the Internet. We, as CERT-IT, have recorded a great number of sniffing attacks in the Italian community. In fact, sniffing is the most popular hacker's attack technique all over the Internet. This paper presents a secure telnet implementation which has been designed by the Italian CERT, to make eavesdropping ineffective to remote terminal sessions. It is not to be considered as a definitive solution but rather as a "bandaid" solution, to deal with one of the most serious security threats of the moment.

2. Introduction

STEL stands for Secure TELnet. We started developing STEL, at the University of Milan, when we realized that eavesdropping was a very serious problem and we did not like the freeware solutions that were available at that time. It was about three years ago. Still, as far as we know, there are no really satisfying packages able to solve the line active and passive tapping problem, and to be simple enough to use and to maintain by the average system administrator. In fact, in our honest opinion, we believe that the actual security packages available on the Internet suffer from at least one of the following drawbacks:

- Not secure enough

  The SRA package [6] is an example. It is based on Sun's Secure RPC, and makes use of the Diffie Hellman key exchange algorithm to negotiate session keys. However, the modulus used with DH is too small (it is only 192 bits [4]) and the session key is only used to encrypt user's login and password. The remainder of the session is left unencrypted.

† Published in the proceedings of the Fifth USENIX Unix Security Symposium, Salt Lake City, Utah

- Too large and complicated to install

  I.e., Kerberos. Several version of kerberized telnet exist. But Kerberos is large, and you have to set up a whole Kerberos environment in order to make use of kerberized telnet. The point is that Kerberos's paradigm is very different from STEL's. STEL, in fact, is intended as a plug and play solution. You just need `stel` and `steld` in order to perform point to point, authenticated encryption.

- Too complex to use and maintain

  STEL's configuration system is very scalable. STEL can perform extensive checks and support different authentication methods. But such checks are not mandatory, and the basic installation is accomplished by just compiling the executables and running `steld`.

- Unable to cope with a distributed environment

  STEL is not a distributed system. But is supports distributed S/Key management, so S/Key keys can be centralized in a single point and network accessed in a secure fashion.

- Subject to export regulations, so not suitable for users outside the USA

  STEL has been developed in Italy. Italy is not subject to crypto regulations and export restrictions. STEL uses encryption algorithms and functions that have been developed or "reinvented" outside the United States. It uses, in fact, a libdes package developed in Australia, an implementation of the Swiss IDEA cipher, an implementation of the Diffie Hellman key exchange system which was developed in Italy and a collection of some other minor crypto routines, developed in Italy by the authors.

3. Description

STEL is a simple package, and it is intended to act as a "secure surrogate replacement" for telnetd, rlogind, rcmd or rshd. However, STEL is not compatible with standard telnet system.

STEL is able to provide secure communications between the client and the server, and supports different authentication methods, in order to be as simple, flexible and convenient as possibile.

STEL has several advantages in respect to other systems:

- It is very simple to install, use and maintain. There are very few configuration files and two executables only
- All data transmissions, any kind, are sent encrypted with a "random" session key
- There are three encryption algorithms available: DES, Triple DES and IDEA
- The method uses the Diffie Hellman algorithm to exchange session keys, and the DH modulus is reasonably sized [4] [6]
- Since Diffie Hellman is vulnerable to "Man In The Middle" attacks, the system has been strenghtened with the Interlock protocol, in order to make such an attack infeasible [1]
- The method is able to provide mutual authentication

- The method is reasonably fast (it adds about a couple of seconds to a connection on a Sparcstation II class machine)
- Upon establishing a secure channel, a variety of authentication methods are available: standard Unix passwords, S/Key and SecurID
- Even standard Unix passwords, being the communication channel encrypted, provide a reasonable level of security in respect to telnet and rlogin
- An optional S/Key daemon is included in the package. `skeyd` makes it possible to centralize S/Key passwords in order to administer passwords in a single point
- It is possible to control logins by checking IP address, authentication type, terminal name, user name, as described in [2]
- Optional automatic killing of IP-OPTIONS
- A built-in S/Key calculator is included in the escape (^]) menu.
- As a feature, S/Key can make use of a keypad file to make dictionary attacks infeasible [3]
- Sources are freely available

4. Practical examples

STEL is client / server based. The client is named `stel`, and it is intended to be directly run by users; the server is named `steld` and can be run as a standalone daemon by the superuser or be invoked by inetd.

The purpose of STEL is to provide the user a remote terminal session, very similar to a telnet or rlogin remote terminal session. The difference is, of course, that all the traffic between client and server is encrypted and that the resulting authentication is much stronger in respect to telnet or rlogin.

First, let us have a look at the usage:

```
$ stel
STEL: Secure TELnet, BETA -- bugs to stel-authors@idea.sec.dsi.unimi.it
@(#) $ stel.c,v 1.61 1995/05/01 18:28:42 vince Exp $
Usage:  stel <hostname> [-l logname] [-p portnum] [-r3imentvD] [commands...]
        hostname:      the system you want to connect to
        -l logname:    the username on the remote system
        -p portnumber: set port number (default port is 10005)
        -r:            enter a random string to enhance randomness
        -3:            use triple DES encryption (default is single DES)
        -i:            use IDEA encryption (default is DES)
        -m:            use 1024 bits modulus (default is 512 bits)
        -e:            disable escape features (^] is enabled by default)
        -n:            do not use data encryption at all
        -t:            do not use pseudo terminals
        -v:            be verbose
        -D:            be extra verbose
```

```
$
$
```

In the following example the user connects with STEL to idea.sec.dsi.unimi.it as root. The authentication is performed using the SecurID system, being root a registered SecurID user on idea.

```
$ stel idea.sec.dsi.unimi.it -l root
Connected to idea.sec.dsi.unimi.it.
This session is using DES encryption for all data transmissions.
Escape character is '^]'.
SECURID authentication required
Enter PASSCODE for root:
Passcode accepted.
------------------
 Welcome to idea!
------------------
idea# hostname
idea
idea# date
Fri Feb 10 17:33:27 MET 1995
idea# exit
Connection with idea.sec.dsi.unimi.it closed.
$
$
```

So, the user has STEL connected to idea.sec.dsi.unimi.it and he/she has authenticated his/herself using SecurID. All data transmission are DES (optionally 3DES or IDEA) encrypted with a session key that is generated with the DH algorithm.

When the user is authenticated, he/she is provided with a very comprehensive set of environment variables that are inherited from the client, and all the terminal settings are inherited from the client as well. For instance, since the DISPLAY, LINES and COLUMNS environment variables are inherited in the remote session, it is possible to remotely execute commands like:

```
$ stel bar xterm -fn 10x20 -bg black -fg white

$ stel foo -l root vi /etc/rc

$ stel somesite -l joe "ps aux > /tmp/xx; vi /tmp/xx"
```

In the following example the user "verbosely" connects to ghost as vince and specifies `/bin/csh` as the command to be executed. The authentication is performed using S/Key.

```
$
$ stel ghost -l vince -v /bin/csh
Connected to ghost on port 10005.
Diffie-Hellman modulo is 512 bits, secret exponent is 510 bits
Exchanging keys with DH scheme (can be a lengthy process)...
Shared encryption key:  CAA90477A9CEA71D
This session is using DES encryption for all data transmissions.
Escape character is '^]'.
S/KEY authentication required
[s/key 93 cr520201]
```

```
Response:  DRUB BARB NOD RET COCO OUTS
SunOS Release 4.1.3_U1 (GENERIC) #1:  Wed Oct 13 17:48:35 PDT 1993
1 @ghost~> hostname
ghost
2 @ghost~> exit
Connection with ghost closed.
$
$
$
```

In the last example, the user STEL connects to idea again, and makes use of some features of the escape menu. In particular, he/she takes advantage of the built-in S/Key calculator to locally (and thus safely) generate the required S/Key response.

```
$ stel idea
Connected to idea.
This session is using DES encryption for all data transmissions.
Escape character is '^]'.
S/KEY authentication required
[s/key 92 cr520201]
Response:
stel> ?
Commands may be abbreviated.  Commands are:

close           close current connection
skey            generate skey response
status          display operating parameters
escape          set escape character
!               shell escape
z               suspend telnet
?               print help information
stel> stat
Connected to idea.
Connection time:  Fri Feb 10 19:07:42 1995
Elapsed time:  0 hours, 0 minutes, 47 seconds
User keystrokes:  1
Session output is 62 bytes
Escape character is '^]'
DES data encryption is on
stel> skey
Reminder - Do not use this while logged in via telnet or dial-in.
Enter seed:  cr520201
enter sequence number:  92
enter secret password:
use mjr DES mode [n] ?  y
CORD AD FLY FEAR NAY ARAB
stel>
CORD AD FLY FEAR NAY ARAB
------------------
 Welcome to idea!
------------------
idea% date
Fri Feb 10 19:11:16 MET 1995
idea% hostname
```

```
idea
idea%
idea% exit
Connection with idea closed.
$
$
```

5. The protocol

STEL is *not* compatible with the standard telnet system; in fact, it uses its own protocol. Formal details about STEL's protocol will be given in a forthcoming Internet Draft. Informally, STEL's insights can be summarized in a number of steps:

5.1. Key exchange

STEL uses the Diffie Hellman exponential based method to determine a common DES (or 3DES, IDEA) key. This completely eliminates the need of keyservers to store and manage user keys, and greatly simplifies the overall system design.

Initially, client and server, whom we call X and Y being the system symmetric, share a large prime number, P, and a generator, that is 3. Then both parties choose a secret value, S_X is chosen by X and S_Y is chosen by Y, making the choice at random among the set of values in modulo P arithmetic. The length of the modulo can be 512 or 1024 bits. Moduli whose length is 192 bits [4], or, in general, smaller than 512 bits, are not secure.

In the next stage, X and Y form the exponentials 3^{S_X} and 3^{S_Y} respectively, and they exchange the exponentials. A malicious hacker could intercept 3^{S_X} and 3^{S_Y} by reading the network but, due the difficulty of computing logarithms in finite fields [1] [4], he/she can not calculate the S_X, S_Y values.

In the final stage, X and Y compute a further exponential. In the case of X the received value 3^{S_Y} is raised to the power S_X. In the case of Y, the received value 3^{S_X} is raised to the power S_Y. As a consequence, both partecipants now share the same secret (that is, ${3^{S_Y}}^{S_X} = {3^{S_X}}^{S_Y} = 3^{S_X S_Y}$), thus a session key is generated by digesting the shared value through MD5.

5.2. Mutual authentication

The basic DH method is very clever, but it provides no authentication between the two parties. A malicious hacker Z using an active line tap could intercept and change all messages, impersonating X to Y and Y to X. X and Y would not realize that; for example, all messages sent by X to Y would be received by Z, the Man In The Middle, decrypted by Z using X's session key, encrypted again using Y's session key and sent to Y. X and Y would share different session keys, yet not realizing that because they have exchanged no information before key exchange.
This is why we added a further step to the protocol in order to make this attack ineffective.

If the user wishing to login on the remote system owns a file named `.stelsecret` in his/her remote home directory then the information contained in the file is exploited to perform mutual authentication between the parties. This is the Interlock Protocol, an idea by Shamir and Rivest described in [1].

Let DH_X be equal to X's session key, as generated by the Diffie Hellman key exchange system. Let DH_Y be equal to Y's session key. Under normal conditions (that is, there is no Monkey In The Middle), DH_X should be equal to DH_Y.

Let ST be the secret shared by the parties by means of the contents of the `~/.stelsecret` file on the remote system. This is a "a priori" secret: it should have been previously transmitted via a trusted path, i.e., by physically accessing the remote system, logging on at the console and directly editing `~/.stelsecret`.

X and Y now construct authenticators A_X and A_Y respectively. Let A_X be equal to $E_{ST}(DH_X)$ that is, X's session key DES encrypted using ST as encryption key. Similarly, let A_Y be equal to $E^2{}_{ST}(DH_Y)$ that is, Y's session key DES encrypted twice using ST as encryption key.

In case Z is actively tapping the communication line, the session keys used by the parties are different ($DH_X \neq DH_Y$). What is more, Z does not know ST. The goal of this authentication step is, in fact, to verify that the session keys are the same at both sides.

A_X and A_Y are divided in two halves. Let A_X be equal to A_{X1} followed by A_{X2} and A_Y be equal to A_{Y1} followed by A_{Y2}.

The exchange is by alternating messages: X sends its first half A_{X1}, Y replies with A_{Y1}, then X sends A_{X2} and Y sends A_{Y2} It is not possible for Z to translate A_X1, knowing only half of the cipher block, yet Y will not reply until he/she receives something, so Z is forced to concoct a value A_X in order to receive A_Y. Y's reply cannot be traslated by Z and passed on at the correct time to X, since Z only receives one half of A_X at a time. After the exchange is complete X and Y can decrypt the authenticators A_X and A_Y respectively and find out if they are really sharing the same session key. If not, they are being impersonated by Z.

It should be noticed that A_X is single encrypted while A_Y is *double* encrypted. This is done on purpose, to prevent a malicious hacker from tricking one side into encrypting other's side exchange.

What is more, since X and Y share a common secret (that is, ST) they can calculate all the halves in advance and thus validate each single half as soon as it as arrives. In case that Y, for instance, receives an incorrect A_{X1} value from X, a random A_{Y1} half is generated and sent to X; as a consequence, The Man In The Middle can not launch a chosen plaintext attack. This method is supposed to foil the attacks exposed in [5].

5.3. Encryption

All data transmitted from the client to the server and vice versa is encrypted using the specified encryption algorithm. The default algorithm is single DES, since it is faster, but triple DES and IDEA are available as well.

DES is used in CBC mode when transmitting environment variables or exchanging "large numbers" in the DH scheme. CFB mode is used when making terminal I/O between client and server. A "random" Initialization Variable is used, and the source of randomness is, optionally, combined with a garbage random string which the user is required to type in.

5.4. Environment settings

A pseudo terminal is usually allocated by the server and attached to the remote process; it is possible, however, to specify that no pseudo terminal should be used in the remote session, a la rshd.

Terminal settings are transmitted encrypted from client to server, so it is not usually needed to "stty" parameters at all.

Also, the following environment variables are sent to the server, in order to make the session as friendly as possibile.

- TERM
- DISPLAY
- LINES
- COLUMNS
- WINDOWID

Prior to executing the user's shell or, eventually, the commands specified by the remote user, the server sets the following environment variables:

- LOGNAME
- USER
- USERNAME
- SHELL
- MAIL

5.5. User authentication

It has been said already that there are three authentication methods available. These methods can be listed according to their security, in decreasing order:

- 1. SecurID
- 2. S/Key
- 3. Unix passwords

The first method is very strong, since SecurID cards are hardware one time password generators. SecurID cards basically contains a DES chip and a clock. The clock is synchronized with the computer's clock and all the user should do to authenticate him/herself is to read the password on the LCD card's display and type it in at the "Enter PASSCODE" prompt. The card, of course, cannot be copied, analyzed or tampered with. This is probably the strongest point in favour of SecurID cards and similar authentication devices such as those marketed by Enigma Logic or Digital Pathways. It must be said that, despite STEL considers SecurID the preferred authentication method, it is understood that SecurID cards are not largely widespread in the Internet community. So using SecurID is not mandatory, and SecurID's support code can be compiled out from STEL completely.

S/Key is a very popular authentication system. It is able to generate one time passwords based upon a seed. More details in [3]. S/Key is vulnerable to dictionary attacks, so STEL uses a modification of the S/Key system as proposed by M. J. Ranum [3]. S/Key system is cheap and convenient, yet it is not as secure as SecurID since one time passwords, when printed on paper, can be stolen and xerox copied.

An optional S/Key daemon has been introduced, to administer all S/Key passwords on a single host. We consider the ability to centralize S/Key passwords as a major feature in our S/Key system. `skeyd` severely simplifies S/Key passwords management, since passwords are stored in a single file and thus the passwords synchronization problem is solved. Data transmission between S/Key clients and the S/Key server is DES-CBC encrypted. The encryption key is stored in /etc/skeydconf, for all the clients and the server (this is an approach common to other security systems, i.e., Kerberos's kdb5_stash).

Unix passwords are insecure, and are used by STEL as the last resort authentication method. Conventional passwords can be stolen by reading the network and are vulnerable to dictionary, cracking and replay attacks. However, STEL never sends Unix passwords unencrypted, so sniffing the network is useless even if Unix password authentication is used.

The three methods are checked in order. If the user is SecurID registered, then SecurID authentication is required. Else, if the user is S/Key registered he/she is prompted with an S/Key challenge. If the user is not S/Key registered, Unix passwords are used but, before that, username is checked against a set of rules as defined in /etc/skey.access. It is possible, in fact, to permit / deny Unix passwords using a wide range of criteria, as described in [2]. Finally, when the user is authenticated, his/her username is checked against /etc/login.access to control login using similar criteria.

STEL verbosely reports errors and login failures via syslog.

6. Availability

STEL has already been ported to HPUX, SunOS, IRIX, Solaris and Linux. At present, a selected team of beta testers is working on STEL; by proceeding this way, we expect to make STEL more secure, reliable and bug free. When the beta testing process is over, STEL will be made available as:

ftp://ftp.dsi.unimi.it/pub/security/cert-it/stel.tar.gz

7. Future directions

Our development efforts will be focused on:

- Having a more comprehensive set of crypto algorithms and authentication methods
- Making STEL optionally work at the session layer [7]

8. Acknowledgements

The authors thank Giordano Pezzoli and Paul Leyland for very helpful comment on STEL's overall design and the security of some of STEL's crypto code. Many thanks to Wietse Venema, Marcus J. Ranum and _H* (The Hobbit) for granting us permission to use bytes of their excellent C code inside the STEL package.

9. References

[1] "Security for Computer Networks", D.W. Davies and L. Price

[2] The logdaemon package, by Wietse Venema <wietse@wzv.win.tue.nl>, available as ftp://ftp.win.tue.nl/security/logdaemon-4.6.tar.gz

[3] A modification of the S/Key client program by Marcus J. Ranum <mjr@tis. com> to make dictionary attacks infeasible. Available as ftp://ftp.tis.com/ pub/firewalls/toolkit/patches/ skey.-tar.Z

[4] "Computation of Discrete Logarithms in Prime Fields", Brian A. La Macchia and Adrew M. Odlyzko, available as ftp://ftp.dsi.unimi.it/pub/security/ crypt/docs/field.ps

[5] "An Attack on the Interlock Protocol When Used for Authentication", Steven Bellovin, Michael Merrit

[6] "Secure RPC authentication (SRA) for TELNET and FTP", D. Safford, D. Hess, D. Schales

[7] "Session-Layer Encryption", M.Blaze, S.Bellovin, to appear in the proceedings of the Fifth USENIX Unix Security Symposium