Secure Communication: Is it possible with SSL and or SSH?

Edvard Wikström
edvma905@student.liu.se

Supervisor,
Tina Lindkvist, tina@isy.liu.se

TDDC03 Information Security Project
Linköping University
May 8, 2004

Abstract

The purpose of this project is to gather information about SSL and SSH as tools for security communication and present them in relation to their features, usability and vulnerabilities. Short instructions of installation and configuration will be presented for both technologies, this will only provide the necessary information to get a quick start. The focus is directed towards vulnerabilities and attacks against these utilities. SSL and SSH are both among the most used methods for transporting information securely through networks. The conclusion is that they offer good security in open networks if proper configuration and updates are applied

1. Introduction

Secure communication is essential for exchange of sensitive information. For a long time weak protocols like Telnet, FTP, the old UNIX utilities, rlogin, rsh and rcp have been used but they do not provide a secure way for the interchange of data. The growth and broad expansion of the World Wide Web has changed its intended use from the beginning. More and more businesses have established online-shops, banks allow their customers to connect to their website and make transactions instantaneously. This new e-commerce era is changing the way we purchase products and services. In this document I will present and analyze two methods for the secure interchange of information in open networks. The objective with this document is to answer three questions about these tools:

- How easy are they to use?
- How secure are they?
- Do you need to be an expert to use them?

2. Formatting your paper

This document will analyze the SSH and SSL protocols for secure communication. To achieve this, white papers and information from reliable sources related to this domain have been studied. Descriptions of how the protocols work and their vulnerabilities are presented in sufficiently detail to understand their weaknesses.

3. SSL

3.2 Background

The SSL protocol is intended to provide a practical, application-layer, widely connection oriented mechanism for Internet client/server communication security developed by Netscape. To use the SSL protocol, the host computer, the client, must be equipped with an SSL Certificate. A digital certificate is an electronic message that verifies that a user sending a message is who he or she claims to be, and to provide a way to the receiver the means to encode a reply. A digital certificate contains the public key of the certificate holder and a variety of other identification information. Digital certificates are issued by a Certificate Authority, such as VeriSign [1], which are trusted third-party sources that have been authorized by banks, governments and other institutions to guarantee the identity of the holder of the certificate.

SSL works by using a public key to encrypt data that is transferred over the SSL connection. The biggest browsers like Internet Explorer, Netscape, Mozilla and Opera supports it. Currently two versions of SSL are available, 2.0 and 3.0. Many websites use the protocol to obtain confidential information, such as credit card numbers. By convention, URL: s

(Uniform Resource Locator) that requires an SSL connection starts with *https://* instead of the normal *http://*. Today, SSL has become the de facto standard for cryptographic protection of Web http traffic. Improvements and development to the protocol is made by the IETF TLS (Transport Layer Security) workgroup. Its is defined by RFC 2246.

3.2 Features and functionality

The SSL protocol runs above TCP/IP and below high-level application protocols such as HTTP or IMAP. It uses TCP/IP on behalf of the higher-level protocols, and allows an SSL-enabled server to authenticate itself to an SSL enabled client, at the same time the client authenticates itself to the server and allows both machines to establish an encrypted connection. SSL client authentication allows a server to confirm the user's identity using the same techniques as those used for the server.

The fundamental steps for the SSL protocol are the following:

- SSL server authentication – allows a user to confirm a server's identity. The SSL enabled client software can use standard techniques of public key cryptography to check that a server's certificate and public ID are valid and have been issued by a certificate authority (CA) which is listed in the client's list of trusted CA: s. This confirmation is important if the user sends the credit card number to the server. It also allows the client and server to select the cryptographic algorithms or ciphers supported by both parts.

- SSL client authentication allows a server to confirm the user's identity. Using the same techniques as those used for server authentication, SSL enabled server software can check that a client's certificate and public ID are valid and have been issued by a certificate authority (CA), listed in the server's list of trusted CA: s. This confirmation is important if the server sends confidential financial information to a customer.

- An encrypted SSL connection requires all information sent between a client and a server to be encrypted by the sending software and decrypted by the receiving software, thus providing a high degree of confidentiality. Confidentiality is important for both parties to any private transaction. In addition, all data sent over an encrypted SSL connection is protected with a mechanism for detecting tampering, whether data has been altered in transit.

The picture below shows where the SSL layer is placed in relation to the other layers in the protocol stack.

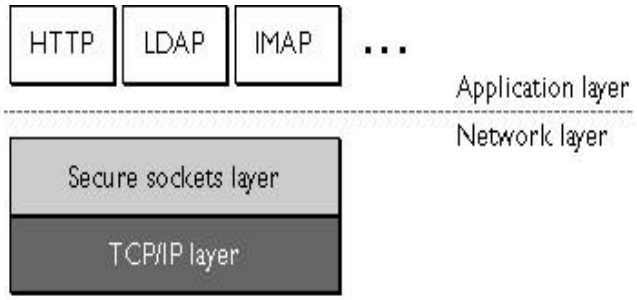

Figure 1. The SSL layer

There are a great number of SSL implementations; here is a list of the most common companies and organizations providing SSL implementations:

- Cisco
- Sun
- IBM
- SSL.com
- OpenSSL – open source.

OpenSSL is an open source variant widely used in both the open source community and in the commercial world due to its attractive license.

The SSL protocol consists of two sub-protocols, the SSL record protocol and the SSL handshake protocol. The record part defines the format used to transmit data, the handshake part involves using the SSL record protocol to exchange a series of messages between an SSL enabled server and an SSL enabled client at the establishment of a connection. The exchange of messages is designed to facilitate the following actions:

More specifically, the record layer provides confidentiality, authenticity and replay protection over a connection-oriented reliable transport protocol such as TCP. The handshake layer which takes care of the key-exchange which initializes and synchronized cryptographic state at the two endpoints of communication.

Both SSL 2.0 and 3.0 protocols support overlapping sets of ciphers suites. These can be enabled or disabled by an administrator. When a client and server exchange information during the SSL handshake, they identify the strongest enabled cipher suit they have in common for the SSL session.

These ciphers below are used to authenticate the client and server, transmit certificates and establish session keys.

- DES - Data Encryption Standard, an encryption algorithm used by the U.S Government.
- DSA - Digital Signature Algorithm, part of the digital authentication standard used by the U.S Government.
- KEA - Key Exchange Algorithm, used for key exchange by the U. S Government.
- MD5. Message - Digest algorithm developed by Rivest.
- RC2 and RC4 - Rivest encryption ciphers developed for RSA Data Security
- RSA - A public key-algorithm for both encryption and authentication. Developed by Rivest, Shamir and Adleman
- RSA key exchange - A key-exchange algorithm for SSL based on RSA.
- SHA-1- Secure Hash Algorithm, a hash function used by the U.S Government.
- SKIPJACK- A classified symmetric-key algorithm used by the U.S Government.
- Triple-DES – DES applied three times.

3.3 Installation and Configuration

All major browsers and e-mail clients have SSL enabled by default. No installation or configuration is necessary to get started. Of course, a preferred algorithm or cipher to use for communication is possible to configure in some applications.

3.4 Usability

The nice thing about SSL is that its use is almost transparent to the user. Having a SSL capable browser or e-mail client it's all that is needed, which today applications supports by default. When connecting for the first time to a server requiring SSL, it will first ask you to download a certificate so you can verify your identity from then on. This is normally done by entering some credentials validating the user. If this verification step is approved the server sends to the client a certificate. The user simply needs to check that the certificate comes from the claimed source by inspecting the information presented. If the source is trusted, accepting the certificate is all that is required. Further visits to the server will not ask for the certificate again, the browser will automatically take care of it. The user needs to be aware of the padlock icon in the bottom corner of the browser which indicates that a secure connection with SSL is taking place.

Recently, the simplicity of SSL has caused a vast amount of new VPN (Virtual Private Network) products to use SSL as the communication protocol instead of the IPSec (IP Security) protocol. With IPSec special software is normally needed at the client and server side, careful and detailed configuration makes it an inappropriate solution compared to SSL based VPN: s.

3.5 Vulnerabilities

SSL is not invulnerable. It has shown a vast number of flaws during the years. I will only name the most serious ones implicating different implementations of the SSL protocol. A paper from the security expert Bruce Schneier and David Wagner [2] gives an analysis of the cryptographic strength of the SSL 3.0 protocol. A number of flaws in the protocol are presented, but his conclusion is that it is a good improvement compared to the SSL 2.0 version. As we will see, other vulnerabilities were found and exploited. I will first present the different possible attack to be made on both versions.

1. Version rollback attack:

The SSL 2.0 protocol contains more flaws and vulnerabilities than the 3.0 version. The 3.0 implementations are likely flexible enough to accept SSL 2.0 connections, at least in the short term. This threatens the potential for version rollback, where an opponent modifies a SSL 3.0 client hello messages to

look like SSL 2.0 message and proceeds to exploit any of the numerous vulnerabilities that exist in SSL 2.0. Another related problem is the possibility to resume a session initiated with the SSL 3.0 protocol with a SSL 2.0 client, this could have subtle and obscure implications.

2. The ciphersuite rollback attack:

An active attacker can force both endpoints of communication to use a weaker form of encryption than they otherwise would choose. This is done by editing the cleartext list of ciphersuit preferences in the hello messages by the attacker. This kind of attack was rather simple to execute. Fortunately this flaw was fixed with SSL 3.0.

3. Key-exchange algorithm rollback attack:

A server can send short-lived public key parameters, signed under its long term certified signing key, in the server key-exchange messages. Several key-exchange algorithms are supported, like ephemeral RSA and Diffie-Hellman public keys. The problem relies on the signature used on the short-lived parameters, it does not protect the field which specifies which type of key-exchange algorithm is in use. Not only should it sign the public parameters but also the relevant data to interpret those parameters. This can lead to secrets exchanged by the server and client to be compromised, which can spoof the rest of the key exchange, including forging finished messages to both endpoints. Therefore an attacker could decrypt all the sensitive application data on the SSL connection.

4. Remote Timing Attack:

Timing attacks enable an attacker to extract secrets maintained in a security system by observing the time it takes the system to respond to various queries [7]. This attack exposes the secret keys used for RSA decryption. To accomplish this, the use of statistical techniques and careful measuring of the amount of time required to complete an RSA decryption operation on known cyphertext can reveal one of the factors (q) of the key. With the public key and the factor (q), the attacker can compute the private key. This kind of attack was proved to be successful with the OpenSSL library [3].

Here follows the vulnerabilities that have been actually found on the different implementations.

A very serious vulnerability in the OpenSSL version 2 caused by a client-exploitable remote buffer overflow in the handshake process, was the reason behind the Slapper worm that spread in the Linux community in September of 2002. This implementation of SSL was used in the popular Apache Web server. More than 6700 severs were affected by creating a peer-to-peer network among the infected computers resulting in a DDoS (Distributed Denial of Service) attack. This caused many corporate Internet connections to be shut down and unavailable.

In March 2004, the development group behind OpenSSL found two serious flaws. Both fall into the DoS category. In the first one a remote attacker could exploit a Null-pointer assignment to perform a SSL handshake which would crash the server [8]. The second one is related to the handshaking code when using Kerberos cipher suites. A remote attacker could perform a carefully crafted SSL/TLS handshake against a server using Kerberos causing it to crash.

In April 2004, working exploits were found for the Windows SSL/TLS implementation in the ISS (Internet Information Server) software that resulted in DoS (Denial of Service) attacks [4]. The exploit, known as the SSL Bomb, would send malformed SSL packets which could force Windows 2000 and XP machines to stop accepting SSL connections and a Windows 2003 machine to reboot. This exploit could also be used to compromise the servers by executing remote code.

3.6 Conclusions

Despite the amount of vulnerabilities, SSL is today the preferred solution to protect data when clients connect to Web-servers on the Internet. There is currently no other alternative that offers the simplicity and security of SSL. Its ease of use makes it a favorable solution. Due to its broad use, improvement and enhancements are continuously being made. SSL is getting more attention in VPN solutions, proving the utility and maturity of the protocol in secure communications.

One problem with the certificate which SSL uses to identify the parties is that certificates can be forged and misused [5]. A malicious sever could impersonate

the identity of a well known server and send a forged certificate to the client. Warning windows supposedly to inform the user about suspicious behavior can be spoofed by the malicious server [6]. It can also be difficult for normal and inexperienced user how to verify the authentication of a server for the first time.

The proper use of SSL demands that the user reacts correctly to all possible security notifications, which could be misunderstood or hard to understand. The user must be aware that the server site has a certificate in the first place. Bruce Schneier and Carl Ellison have described some problems and flaws related to certificates [5].Also, just having the little padlock in the corner which indicates a secure connection is easy to overlook. The user must take care of always react suspiciously if the padlock is missing. Unless users are made sufficiently aware of the importance to look of security measures on a site, the presence of security measures may not be optimally effective.

4. SSH

4.1 Background

SSH stands for Secure Shell, and is both a program and an internet protocol used to log into another computer over a network, to execute commands in a remote machine and to move files from one machine to another via an encrypted link. It is widely used by network administrators to control the Web and other kinds of servers remotely. SSH is actually a suite of three utilities, slogin, ssh, and scp that are secure versions of the earlier UNIX utilities, rlogin, rsh, and rcp. The big difference being that SSH provides strong authentication and secure communications over unsecured channels.

The information through SSH is encrypted and secure in several ways. Both ends of the client/server connection are authenticated using a digital certificate, and passwords are protected by being encrypted.

The old UNIX utilities, rlogin, rsh and rcp have been heavily used for a long time. These are the traditional BSD (Berkeley Software Distribution) programs. They have been the most used programs for accessing and moving files between machines in the UNIX environment. A big flaw regarding these utilities is their vulnerability to different kinds of attacks. Somebody who has root access to machines on the network, or physical access to the wire, can gain unauthorized access to systems in a variety of ways. It is also possible for such a person to log all the traffic to and from your system, including passwords. SSH never sends password in the clear.

Because of their weak communication security, SSH was created. Tatu Ylönen from Finland created the fist version in 1995 and later formed the SSH Communications Security company to exploit this innovation. This company subsequently relicensed SSH to F-secure. The SSH protocol is now under standardization by the IETF secsh [9] working group.

There are several implementations of the SSH protocol available for most modern platforms, including Microsoft Windows and Mac OS. There are commercial versions, freeware versions and open source versions. Several commercial variants exists, among the most popular seems to be:

SSH clients:

- SSH Secure Shell client, SSH Communications Security [10]
- F-secure SSH, F-Secure [11]
- SecureCRT and SecureFX, Van Dyke Software [12]
- RemotelyAnywhere, RemotelyAnywhere [13]
- TinyTERM, Century Software [14]
- PowerTermPro, Ericom Software [15]

SSH servers:

- SSH Secure Server, SSH Communication Security [10]
- F-secure SSH Server, F-Secure [11]
- VShell Server, Van Dyke Software [12]
- Pragma SecureShell Server, Pragma Systems [16]
- WinSSHD, Bitvise [17]
- RemotelyAnywhere Server Edition, RemotelyAnywhere [13]

All these versions offer different features and solutions, but they try to implement the core of the SSH protocol. Although some of them do not fully comply with the standard. The software from SSH Communication Security and F-secure are the only ones certified in the list above.

There are a number of open source variants to choose from. These offer the same core services like

the commercial variants, although they are not as rich in features. PuTTY is the only one in this list that has a version for Microsoft Windows.

SSH clients:

- OpenSSH client [18]
- lsh client, The GNU project [19]
- ossh client [20]
- MacSSH [22]
- Putty [22]
- sshtools [23}

SSH servers:

- OpenSSH daemon
- lsh daemon, The GNU project
- ossh daemon
- Dropbear SSH server [24]

OpenSSH is an open source implementation of SSH. OpenSSH derives from the original, free implementation of SSH. This implementation is the one most used in the UNIX and Linux environments.

Something to have in mind is that there exits legal issues involved with cryptos, restricted by cryptography laws. It is illegal in countries where encryption is outlawed. Fortunately Sweden is not affected. Crypto Law Survey [25] it's a survey of cryptography laws in many countries but I can't vouch for its correctness.

4.2 Features and functionality

There are two versions of SSH available: SSH1 and SSH2. SSH1 is the first and original version.SSH2 is the new protocol version, it is rewritten with improved cryptography and is designed for more general purpose VPN: s (Virtual Private Network). The differences between the SSH1 and SSH2 versions are that they are two entirely different protocols. SSH1 and SSH2 encrypt at different parts of the packets, and SSH1 uses server and host keys to authenticate systems where SSH2 only uses host keys. SSH2 is a complete rewrite of the protocol and does not use the same network implementation that SSH1 does. SSH2 was designed with more security in mind. In addition, it offers more improvements in performance and portability. Because of the different protocol implementation, they are not compatible. The SSH1 protocol is not being developed anymore, SSH2 is the new standard. Although, the first version is brought up and explained because of its broad use even today. The recommendation is to use the second version for enhanced security.

The SSH2 version includes:

1. sftp (secure ftp) and SSH2 tunneled ftp.
2. Uses separate configuration files for the SSH1 and SSH2 protocols.
3. Compatible with SSH1, when SSH1 has been installed prior to SSH2.
4. Supports DSA and Diffie-Hellman key exchange.

Secure Shell uses the following ciphers for encryption:

Cipher	SSH1	SSH2
DES	Yes	No
3DES	Yes	Yes
IDEA	Yes	No
Blowfish	Yes	Yes
Twofish	No	Yes
Arcfour	No	Yes
Cast128-cbc	No	Yes

There are different modes of authentication available:

1. Password (the /etc/passwd or /etc/shadow in UNIX). This authenticates the user by comparing the credentials submitted with the password file to find a match.

2. User public key (RSA or DSA, depending on the release). A mathematical key that can be made publicly available and which is used to verify signatures created with its corresponding private key. Depending on the algorithm, public keys are also used to encrypt messages or files which can then be decrypted with the corresponding private key

3. Kerberos (for SSH1). An authentication system developed at the Massachusetts Institute of Technology (MIT). Kerberos is designed to enable two parties to exchange private information across an otherwise open network. It works by assigning a unique key, called a ticket, to each user that logs on to the network.

The ticket is then embedded in messages to identify the sender of the message.

4. Hostbased (.rhosts or /etc/hots.equiv in SSH1 or public key in SSH2). This method only allows the host, computers, that are listed in this these files to be accepted.

The main use of SSH is to protect information that travels inside a network or through different networks that could be exposed to the entire world, like the Internet. SSH gives protection against this known attacks or weaknesses:

- IP spoofing, where a remote host sends out packets which pretend to come from another trusted host. SSH protects against a spoofer on the local network, who can pretend to be the local router to the outside.
- IP source routing, where a host can pretend that an IP packet comes from another trusted host.
- DNS spoofing, where an attacker forges name server records.
- Interception of clear text passwords and other data sent by intermediate hosts.
- Manipulation of data by people in control of intermediate hosts
- Attacks based on listening to X authentication data and spoofed connection to the X11 server.

Among the features provided by the popular OpenSSH implementation is the ability to forward X11 connections and arbitrary TCP ports over the secure channel. Making a secure access to applications on a remote X11 server, or other applicatins in general that can be accessed through the TCP protocol.

4.3 Installation and Configuration

For the Windows environment the recommended application is PuTTY, it is free and has a small footprint.

The installation is very straight forward, get the program from this location [22] and store it on the desired location on your computer. No installation or special configuration is necessary, just run the application when needed. It is very user friendly. See next section for explanation and picture of the program.

Just to mention something about the server application. If you are installing the server – daemon – check to make sure that the remote clients are connecting to you with the right version of SSH. An SS1 daemon will only work with SSH1 clients, while an SSH2 daemon with built SSH1 capability, will support both versions. For more information about this se the SSH-FAQ [26].

OpenSSH is the preferred SSH implementation in the UNIX and Linux environment. To install it, grab the sources from http://www.openssh.org. Then, it's only a matter of entering these commands, the last as user root:

```
./configure
make
make install
```

These are all necessary steps to install it on the computer, after this, the client is available for usage as explained above. For more information, see the *man* page.

4.4 Usability

All of these implementations are command line based. That is they are invoked and used in a shell (terminal). The CLI (Command Line Interface) is the normal way to access and use programs in the UNIX environment. They give a lot of flexibility to the power user but can be cumbersome to master.

There is a GUI (Graphical User Interface) front end to use on top of this applications called SecPanel [27] that gives a nicer look and a less intimidated feeling. This package is easy to install and configure but will not be brought up in this document. PuTTY is the only exception of all SSH implementations mentioned that is a GUI application. The downside is that it that the GUI version is only available on Windows.

In a Windows environment, the PuTTY appication is very simple and easy to use through its GUI. Start the application enter the server you want to access and choose port number 22, the SSH port, for the connection and click the open button. Is that simple to establish a connection. See figure below for a illustration. After this a new terminal window will appear which will ask for your account and password. If validated, your are granted access. If this is the first time you connect to the specified server, the applicatin

will not recognize it and ask for your approval. You will continue by simply answering yes or no, depending of your trust to the server.

As you can see in figure 2, there are a great number of options available, as you get more acquainted with the application the new options will be easily mastered. The application offer possibility to remember settings for a specific server, appearance, connection settings and preferred algorithm for the session.

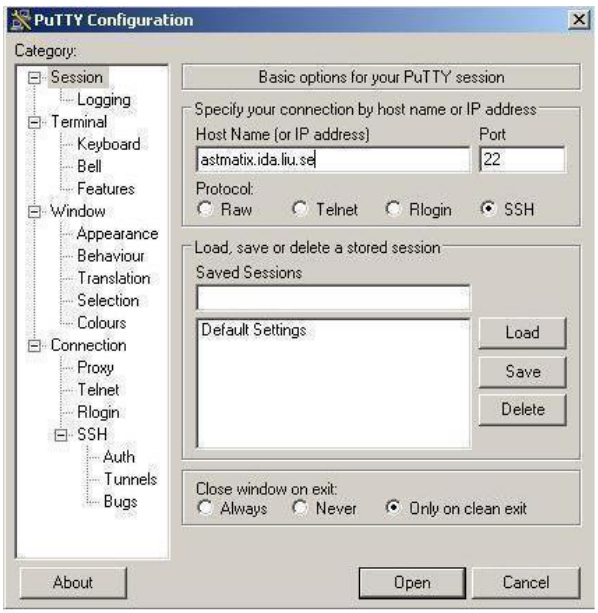

Figure 2. PuTTY application connecting to IDA

As mentioned earlier, in the UNIX and Linux world, OpenSSH is the preferred implementation of the SSH protocol. It is a command line application. To connect to a remote host the normal procedure is as follows:

$ ssh username@remotehost

The $ sign represent the terminal prompt, *ssh* invokes the program, *username* is the name of the account on the remote host, and *remotehost* the host you want to connect to. This is simple to learn and remember, though there are security aspects related to its use that can complicate things. The first time you connect to a new host you need to verify its authenticity by checking the key fingerprint it presents. This verification is obviously difficult to establish for a novice or normal user. The configuration is done by changing parameters in a text configuration file using a normal editor. This approach is known to be obscure and error prone for new users. The proper configuration can be a daunting task. For more detailed descriptions of its usage and possible configuration, use the command *man ssh* to get further information.

4.5 Vulnerabilities

1. SSH1 insertion attack:

SSH1 uses a 32-bit cyclic redundancy check (CRC-32) algorithm to verify that a packet contains only legitimate data. If certain cipher modes are used, a remote attacker could create an ssh packet that could decrypt to arbitrary plaintext. Another weakness in the CRC-32 algorithm could allow the attacker to forge a valid checksum so that the packet will seem to be legitimate. By inserting such packets into an existing session, the attacker could execute arbitrary commands on the system.

2. Man in the middle attack:

This attack affects the SSH1 protocol. In an SSH session, the server hands the client a public key, then proceeds to prove that it has access to the corresponding private key. For this transaction to be valid, the client must independently verify that this particular public key identifies the host the client originally intended to contact. The problem is that client may be configured to automatically accept and record host public keys on first contact. The user should verify somehow that the key actually belongs to the host it claims to be. The problem behind this lies in the host-key verification step. It is not a required step in the protocol. By spoofing DNS replies from a client and redirect the SSH connection to an intermediate system, it is possible to intercept all traffic.

3. Connection redirection attack:

When making a connection to localhost, SSH disables host key checking to provide compatibility with NFS filesystems. If the machine uses a poisoned DNS server to resolve localhost, it is possible to redirect the SSH session to a different host. Users are normally asked to confirm the acceptance of a host key the first time it is presented. If the user accepts it, he or she is asserting that the key represents the host they intended to connect to. But, if an attacker exploits this

vulnerability, the user will not be asked for this confirmation because host key checking has been disabled. Not even the most alert user will be able to detect the redirection.

4. Session key recovery:

The SSH1 protocol contains a flaw in the key exchange protocol. By sniffing a connection between a client and the server, it is possible to detect when the connection starts and get the packet containing the encrypted session key. Then, by working in parallel, saving all successive packets exchanged between server and client, and at the same time attempt a session key decryption. Once the session key is decrypted, the saved encrypted packets sent between this client and the server can be decrypted in a straight-forward manner.

5. ssh-agent vulnerability:

This vulnerability enables users to use RSA credentials belonging to other users who use the ssh-agent program [28]. The cause of this is because the ssh-agent manages the RSA keys for the ssh program, and is primarily used to help users avoid typing their pass phrase every time they wish to use ssh, slogin or scp. This is accomplished by the fact that when the ssh client connects to a server using the AF_UNIX socket, it is running as super-user, also known as root, and performs insufficient permission checking. This makes it possible for users to trick their clients into using credentials belonging to other users.

Any user who utilize RSA authentication and use the ssh-agent program can have their credentials improperly used by a malicious user, who then can access services or programs on a host machine.

4.6 Conclusions

SSH2 is the new standard established by the IETF secsh workgroup. Though many people still use the SSH1 version, there are arguments against running it:

- It is subject to man-in-the-middle attacks.
- There are structural weaknesses in SSH1 which leaves it open to additional attacks.
- It has less supported platforms.
- It supports .rhosts. authentication, which it's against the draft for SSH2

At the same time if offers some advantages that the second version does not meet:

- It has more diverse authentication support like, AFS and Kerberos.
- It is supported by a great number of platforms, this is changing has SSH2 is catching up.
- Less license issues than SSH2, this is a major reason why it is still popular.

SSH is a good replacement for the old utilities used to access remote host and move files between them. The SSH2 version removes some flaws from the first versions and adds some enhancements to make it even more secure. It's definitely a good improvement compared to other alternatives. By using SSH the secure communication is possible, but configuration and the maintenance of patches is necessary to stay up to date with vulnerabilities. Though the use of SSH in the command interface can seem daunting for the first timer, it is simple to use. A little introduction and guidance to a complete novice should be enough to get the user started with the basic usage. Although, the proper configuration of the client and the server can be a difficult task, depending of the functionality needed. The PuTTY client for the Windows environment is a good example of a great and easy to use SSH client, unfortunately, it is currently not available for the UNIX and Linux community.

5. Summary and final words

Both methods presented in this document give good security for the interchange of information in open networks. SSH is convenient for login and executing commands to remote machines. It offers a great number of features that facilitates operability between hosts. SSL is preferred choice for accessing sensitive information in Web and mail servers. The popularity and simplicity of the protocol is making its way into VPN products which will make it a preferred alternative in the future for secure communication.

Analysis and improvements are continuously being made to both protocols assuring their maturity. It is imperative in this kind of applications to keep up with the vulnerabilities and security advisories presented for each protocol. The proper configuration and

maintenance of updates and patches are necessary to guard against possible exploits and misuses of the software. Secure communication is definitely possible with these technologies.

The integration of SSL into mainstream browsers and email clients are giving us the possibility of secure communication transparent to their usage. Make sure you are using the 3.0 version of the SSL protocol. Although, the user needs some knowledge about the identification of the certificate at the first stage of communication, to avoid possible misuse.

SSH is definitely a good replacement for the older tools that are still available. The recommended version to use is SSH2. It provides a great set of features that improves its functionality and use. SSH might not be the easiest thing around, but a basic introduction should get the user acquainted with it, especially with the warning messages that could mean strange behavior.

References

[1] VeriSign, http://www.versign.com, accessed April 2004

[2] Analysis of the SSL 3.0 protocol, http://www.schneier.com/paper-ssl-revised.pdf, accessed April 2004

[3] OpenSLL, http://www.openssl.org, accessed April 2004

[4] SSL Bomb, http://news.netcraft.com/archives/2004/04/19/exploit_targets_windows_ssl_vulnerability.html, accessed April 2004

[5] Ten Risks of PKI, http://www.counterpane.com/pki-risks-ft.txt, accessed April 2004

[6] Web Spoofing Revisited: SSL and beyond, http://www.cs.dartmouth.edu/~pkilab/papers/tr417.pdf, accessed April 2004

[7] Timing attack against RSA private keys, http://lists.netsys.com/pipermail/full-disclosure/2003-March/004142.html, accessed April 2004.

[8] Apache-SSL Client Certificate Forging Vulnerability, http://www.securityfocus.com/bid/9590, accessed April 2004

[9] ITEF SECSH, http://www.ietf.org/html.charters/secsh-charter.html, accessed April 2004

[10] SSH Communication Security, http://www.ssh.com, accessed April 2004

[11] F-Secure, http://www.f-secure.com/, accessed April 2004

[12] Van Dyke Software, http://www.vandyke.com, accessed April 2004

[13] RemotelyAnywhere, http://www.remotelyanywhere.com, accessed April 2004

[14] TinyTERM, Century Software, http://te.censoft.com/products/tinyterm.php?src=, accessed April 2004

[15] PowerTermPro, Ericom Software, http://www.ericom.com/default.asp, accessed April 2004

[16] Pragma SecureShell Server, Pragma Systems, http://www.pragmasys.com/SecureShell, accessed April 2004

[17] Bitvise, http://www.bitvise.com/winsshd.html, accessed April 2004

[18] OpenSSH, http://www.oopenssh.org, accessed April 2004

[19] GNU, http://www.gnu.org, accessed April 2004

[20] ossh, ftp://ftp.pdc.kth.se/pub/krypto/ossh/, accessed April 2004

[21] MacSSH, http://pro.wanadoo.fr/chombier/, accessed April 2004

[22] PuTTY, http://www.chiark.greenend.org.uk/~sgtatham/putty/, accessed April 2004

[23] sshtools, http://www.sshtools.com, accessed April 2004

[24] Dropbear SSH Server, http://freshmeat.net/dropbear/, accessed April 2004

[25] Crypto Law Survey, http://rechten.uvt.nl/koops/cryptolaw/, accessed April 2004

[26] SSH-FAQ, http://www.dreamwvr.com/ssh2-faq/ssh-faq-1.html, accessed April 2004

[27] SecPanel, http://www.pingx.net/secpanel, accessed April 2004

[28] SSH Agent Vulnerability, http://www-arc.com/sara/cve/SSH_vulnerabilities.html, accessed April 2004