Global Information Assurance Certification Paper

Name: Wesley Wong
Version: 2.0

Stunnel: SSLing Internet Services Easily

Purpose

This paper provides a method to securely use existing clear-text protocols under SSL without any need to modify the existing software or source code.

Introduction

The Secure Sockets Layer (SSL) is a protocol designed for authenticating and encrypting communications on the web designed by Netscape Communications in 1994. SSL is being used in many of the popular protocols such as IMAP, POP, LDAP and SMTP. SSL is currently in version 3 and the Internet Engineering Task Force (IETF) created a standard based on SSL version 3 called Transport Layer Security (TLS) protocol published as RFC 2246[8].

Stunnel is a program that functions as a SSL client and/or a server to allow encrypted communications over a TCP-based clear-text protocol using the Secure Sockets Layer (SSL) protocol without making any modifications to the original program. Stunnel can run in daemon mode or as an inetd service. Stunnel is licensed under the GNU General Public License, which makes the source code freely available for everyone to inspect and modify. Stunnel is currently available for both Unix and Windows platforms but this paper will be mainly focusing on the Unix platform. The OpenSSL (SSLeay is acceptable) cryptographic library will be needed, as Stunnel does not contain any cryptographic algorithms.

There are reasons for using Stunnel to add security on a protocol instead of adding SSL capability into a program. One of the major reasons for using Stunnel is that source code may not be available for the program especially if it is a commercial application and the vendor decides not to add SSL to the program or the vendor goes out of business. In extreme cases, the complete source code to the program may not be available. Another reason is that it takes time to add SSL capability to in-house programs. Stunnel could be useful as a "stopgap" measure while SSL is being implemented and tested in the program.

How it works

Stunnel can function on both the server and client sides of a SSL connection. On the server side, Stunnel functions by listening on a secured port, encrypting/decrypting data received/sent and relaying the data from/to the port of the existing clear-text service on the same server or a remote server. On the client side, a port listening on the loopback interface of the local system would send/receive data on the clear-text protocol and tunnel out on an ephemeral port to connect to the secured port of the remote system.

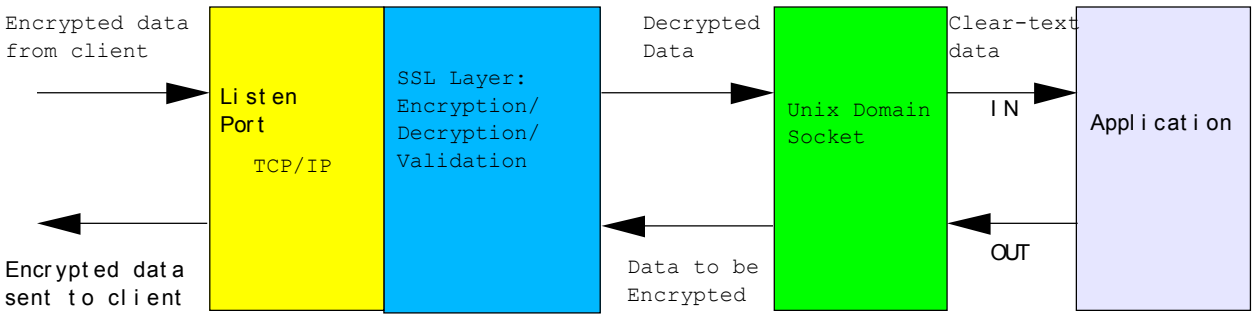

Figure 1

Figure 1 shows how Stunnel works on the server side. On the left side of the diagram, we start with the client sending encrypted data from its ephemeral port to the host's port on the TCP/IP network stack monitored by Stunnel. When Stunnel receives the data, it will decrypt the data based on the negotiated encryption algorithm and the shared secret key. Once the data is decrypted, the clear-text is passed onto a port on the system's Unix domain Socket and from there, the data is passed onto the real application for it to process. The reverse process has the application sending encrypted data back to the client, and works in a similar way to when data is received.

On the server-side of Stunnel, it is best to have the existing clear-text protocol running on a loopback network interface (also known as localhost) instead of the regular IP network interface. There are significant performance improvements running the network service over a Unix Domain Socket. Unlike the traditional IP network interface, the Unix Domain Socket does not have to deal with error-correcting protocols. Performance can actually double when using a Unix Domain Socket versus IP depending how well the implementation of Unix Domain Socket was done [5].

It is important to note that Stunnel will not work with any UDP (User Datagram Protocol)-based protocol such as named DNS, syslog and media streaming protocols. SSL was primarily designed to work with the TCP protocol. In addition for Stunnel to function, the TCP-based protocol must also meet three other requirements. First, it must not use multiple connections like the ftp protocol (control and data ports). Second, it "doesn't depend on Out Of Band (OOB) data". And finally, the "remote site can't use an application-specific protocol, like ssltelnet, where SSL is a negotiated option"[1].

Building and Configuring Stunnel

We will use the RedHat Linux 6.2 distribution for our build and examples but these instructions should work with most Unix platforms. Before the Stunnel binary can be built, the OpenSSL toolkit must be built as Stunnel gets its encryption capability from the toolkit. The latest version of the OpenSSL toolkit source code can be download from the OpenSSL website (http://www.openssl.org). Before building the toolkit, there are currently two encryption algorithms (RC5 and IDEA) patented in the toolkit. In countries with software patents, these two encryption algorithms can be left out of the toolkit. The RSA encryption algorithm patent expired in September of 2000 so there are no problems in using the algorithm.

Here are the steps to building OpenSSL:

1.) Unzip and untar the OpenSSL binary: `gunzip -dc openssl-0.9.6b.tar.gz | tar -xf -`
2.) Change into `openssl-0.9.6b` directory and run configuration program with options to exclude rc5 and idea algorithms: `./config no-rc5 no-idea`
3.) Run `make` to build the toolkit.
4.) Run `make test` to make sure the build passes it tests.
5.) Run `make install` to install the toolkit under the `/usr/local/ssl` directory.

After OpenSSL is built and installed, we can proceed with the installation of the Stunnel program. Source code for the latest version of Stunnel can downloaded from the Stunnel website (http://www.stunnel.org).

One optional package that can be built into Stunnel is TCP-Wrappers to restrict access of SSL services. TCP-Wrappers is included with most Linux distributions and can be easily built for most Unix platforms. The source code can be downloaded from the following site: ftp://ftp.porcupine.org/pub/security/tcp_wrappers_7.6.tar.gz (for the IPv4 version) or ftp://ftp.porcupine.org/pub/security/tcp_wrappers_7.6-ipv6.1.tar.gz (for the IPv6 version). When installing TCP-Wrappers, the libwrap.a library should be installed in the /usr/local/lib or /usr/lib directory so that it can be statically linked into the stunnel binary.

In this build, Stunnel will use TCP-Wrappers and a random file (/dev/urandom on Linux) to help generate better keys. Here are the steps to building Stunnel:

1.) Unzip and untar the Stunnel distribution: `gunzip -dc stunnel-3.21b.tar.gz |tar -xf -`
2.) Change into the stunnel-3.21b directory and run the configuration program:
```
./configure --with-random-file=/dev/urandom --with-tcp-wrappers
```
3.) Run make to build. After the make is finished, it will generate a private key and a self-signed certificate into the stunnel.pem. Recommended for testing use only.
```
Generating a 1024 bit RSA private key
..........++++++
.....................++++++
writing new private key to 'stunnel.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [PL]:US
State or Province Name (full name) [Some-State]:California
Locality Name (eg, city) []:Los Angeles
Organization Name (eg, company) [Stunnel Developers Ltd]:The Dot Com Corporation
Organizational Unit Name (eg, section) []:IT Support
Common Name (FQDN of your server) [localhost]:my.mysecuredot.com
/usr/local/ssl/bin/openssl x509 -subject -dates -fingerprint -noout \
        -in stunnel.pem
subject= /C=US/ST=California/L=Los Angeles/O=The Dot Com Corporation/OU=IT
Support/CN=my.mysecuredot.com
notBefore=Oct  9 07:43:26 2001 GMT
notAfter=Oct  9 07:43:26 2002 GMT
MD5 Fingerprint=EC:97:B2:E2:45:D6:F5:63:A5:C0:B9:86:12:D1:A5:4A
```

4.) Run `make install` to install the stunnel program.
5.) Copy the stunnel.pem to the /usr/local/ssl/certs directory.

For production use, it is recommended that a Certificate Authority (such as Verisign) sign the SSL certificate so that users can be assured of a server's authenticity when connecting to it.

The procedure of generating a SSL certificate using the OpenSSL toolkit is as follows:

1.) Generate a 1024 RSA private key and store in www.mysecuredot.com.key file:

```
<openssl-path>/bin/openssl genrsa –rand file1:file2:file3 –out www.mysecuredot.com.key
1024
```

file1, file2, file3 and file4 are large files used as random seeds. This file should be protected to be readable only by root as the key is not password protected. The key should also be backed up to floppy or tape and stored in a safe location.

2.) Generate a Certificate Signing Request from the private key:

```
<openssl-path>/bin/openssl req –new –key www.mysecuredot.com.key -out
www.mysecuredot.com.csr
```

The program will ask for X.509 attributes of the certificate similar to the generation of the self-sign certificate above. The information in the www.mysecuredot.com.csr file should look like the following:

```
-----BEGIN CERTIFICATE REQUEST-----
[certificate data here…]
-----END CERTIFICATE REQUEST-----
```

This information is needed for submission to the Certificate Authority for signing.

3.) After the Certificate Authority replies with a final certificate, we should save it to a file called www.mysecuredot.com.crt and concatenate the private key file and the final certificate file together to generate a new file for Stunnel. Copy the file to the /usr/local/ssl/cert directory.

```
cat www.mysecuredot.com.key www.mysecuredot.com.crt > www.mysecuredot.com.pem
chown root:sys www.mysecuredot.com.pem
chmod 400 www.mysecuredot.com.pem
cp www.mysecuredot.com.pem /usr/local/ssl/certs
```

After the certificate is completed, we can proceed with SSLing our existing applications.

Example 1: Securing an existing IMAP server

In the first example, we will secure an existing IMAP server when SSL capability cannot be easily added (e.g. access to the source code of the server is not available). The existing IMAP server runs on port 143 using inetd to spawn new connections to the IMAP service. In order to run a SSL IMAP service, the SSL IMAP service must run on a separate designated port. Fortunately, there is a port designated for SSL IMAP service (port 993) and many others defined for other TCP-based services. Many of these mappings are defined in the /etc/services file on any Unix server. If the particular port is not available, the information can obtained from the Internet Assigned Number Authority (IANA) website at http://www.iana.org/assignments/port-numbers.

If the /etc/services file doesn't contain a line that begins with "imaps", then the following line should be appended to the file:

```
imaps              993/tcp    #imap4 protocol over TLS/SSL
```

Next we will define the following line in /etc/hosts.allow file:

```
imaps: ALL
```

This is to permit imaps sessions from all client connections since we have configured Stunnel with TCP-Wrappers capability.

Once that is done, we can startup the SSL service for imaps. The following command line will startup the imaps service (must execute as "root"):

```
/usr/local/sbin/stunnel  -d imaps -N imaps -p /usr/local/ssl/certs/www.mysecuredot.com.pem -r
localhost:143
```

We now have a SSL IMAP service, running in daemon, available for use and we can test the connection with an IMAP client that supports SSL. Currently, Netscape Mail, Mulberry, Microsoft Outlook and Eudora have built-in support for SSL IMAP. If connections are tested with a self-signed certificate, some clients (Netscape Mail in particular) will give some warnings about authenticity of the SSL certificate. As long as the certificate is accepted, the connection will continue.

In addition, we can force users to connect to the IMAP server through SSL only and disable the access to the IMAP server via clear-text. One method is to move the IMAP server to a port not used by other protocols and forward the Stunnel connection to the newly assigned port. Another method is use TCP-Wrappers and limit all outside connections except for the loopback interface and possibly some internal systems. This assumes that the IMAP server is spawned from inetd or supports TCP-Wrappers internally.

Example 2: Securing a database connection

In this example, we will secure a database connection between the database server and a client. The database server used will be a version of MySQL 3.x running on the Linux and the client system will be a Windows 2000 system. Current versions of MySQL do not possess any SSL capabilities so Stunnel must be used on the server and clients. Precompiled versions of Stunnel and OpenSSL library for the Win32 platform are available from the Stunnel website. The dll libraries for OpenSSL should be placed in the same directory as the Stunnel binary.

In our existing MySQL setup, we have the MySQL database server listening on port 3306 for requests. Stunnel will listen on port 7100 for SSL requests to the MySQL database.

```
/usr/local/sbin/stunnel -d 7100 -N mysql -p /usr/local/ssl/certs/www.mysecuredot.com.pem
localhost:3306
```

Remember to add a "mysql" to the /etc/hosts.allow file to allow or restrict requests to the

MySQL server.

Next, we need to have Stunnel running on the Windows 2000 system to redirect clear-text MySQL connections as SSL connections. We assume that the Stunnel binary is located in the C:\Stunnel directory and the dll files are also located in the same directory. The following command will start up Stunnel as a client:

```
/usr/local/sbin/stunnel -c -d 7100 -r db.mysecuredot.com:7100
```

We now have the Windows 2000 system forwarding requests from port 7100 of its loopback interface to port 7100 of the database server at db.mysecuredot.com. Any database programs running on the Windows 2000 system should be able to connect to the MySQL server securely using port 7100 on the loopback interface. The method above is useful for transferring sensitive data safely from remote locations. Also, it prevents database login and passwords from being sniffed since they are clear-text by default.

Important Notes

There are a few important issues to consider when implementing Stunnel services on your system. Stunnel may possibly require more system resources than running a native SSL program. When Stunnel receives a connection for a service and forwards the request, it would require at least two processes to handle the request because most Unix daemons (sendmail, imap) and all inetd-based services spawn new processes for every new connection. The Stunnel process counts as the second process. Tuning the kernel of the operating system is one way to deal with the problem of handling large process tables. In addition, it may be necessary to add more RAM and virtual memory to handle the additional processes.

In addition, Stunnel performance is improved by running it in daemon mode instead of inetd. Running Stunnel in daemon mode provides the following benefits: no SSL initialization necessary with every connection, and SSL session caching[1].

Another issue to consider before implementing Stunnel is the capacity of the system. If the number of SSL connections per unit time is large, this may possibly bog down the system, as SSL requires extra CPU computing power for executing the encrypting and decrypting functions. One possibility is to use a SSL accelerator board or appliance, which offloads the cryptographic processing away from the main processor. The OpenSSL toolkit contains the capability to link with the accelerator so there is no need to modify the Stunnel program.

A very important issue is the security of the private key, as Stunnel does not have the capability to handle encrypted private keys. It is important to secure the system that is holding the private key and limit the number of people who can access the system.

Conclusion

Stunnel is a program that adds SSL capability to existing Internet services without having to modify existing source code and spending a lot of time testing. It helps remote users connect securely and easily to their home networks.

Bibliography:

[1]Hatch, Brian. "Stunnel.org Frequently Asked Questions", October 2001. URL: http://www.stunnel.org/faq/ (October 10, 2001)

[2]Netscape Communications. "Introduction to SSL", October 1998. URL: http://developer.netscape.com/docs/manuals/security/sslin/index.htm (October 10, 2001)

[3]Viega, John. "How to encrypt arbitrary TCP connections inside SSL", May 2001. URL: http://www-106.ibm.com/developerworks/security/library/s-stun.html (October 12, 2001)

[4]Cecchini, Roberto. "Secure IMAP (and POP3)", December 18, 2000. URL: http://security.fi.infn.it/tools/stunnel/index-en.html (October 10, 2001)

[5]Stevens, W. Richard. "TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP, and the Unix Domain Protocols", 1996, pp. 222-224

[6]Verisign, Inc. "Generating a Key Pair and CSR for an Apache Server with modssl", 2000. URL: http://www.verisign.com/support/tlc/csr/modssl/v00.html (November 1, 2001)

[7]Engelschall, Ralf S. "OpenSSL Frequently Asked Questions", October 2001. URL: http://www.openssl.org/support/faq.html (November 1, 2001)

[8]Dierks, T. and Allen, C. "The TLS Protocol Version 1.0", January 1999. URL: ftp://ftp.isi.edu/in-notes/rfc2246.txt (November 10, 2001)

Useful Links:

MySQL Database: http://www.mysql.org/
OpenSSL toolkit: http://www.openssl.org/
TCP-Wrappers: ftp://ftp.porcupine.org/pub/security/tcp_wrappers_7.6-ipv6.1.tar.gz
Internet Assigned Number Authority (IANA): http://www.iana.org/
Internet Engineering Task Force: http://www.ietf.org/