Hands-on Exercise Part

Due Saturday, May 28th, 2022 @ 11:59pm

(**18%** of the total course grade)

Secure Communication with Stunnel[1]

1. Objective

One of the major challenges facing any network applications is that of securing data in transit, where encryption is the most promising solutions to address this issue. However, it is possible that we have an old, legacy application that has no support for encrypted communication. Also, we might not have the time and/or money to upgrade that old system to the latest version which does support encrypted communication, or simply can't afford the downtime required for such an upgrade. Furthermore, even there exists such support for encrypted communication to the application, but it is still possible that their implementation does not meet the minimum requirements of the running environment.

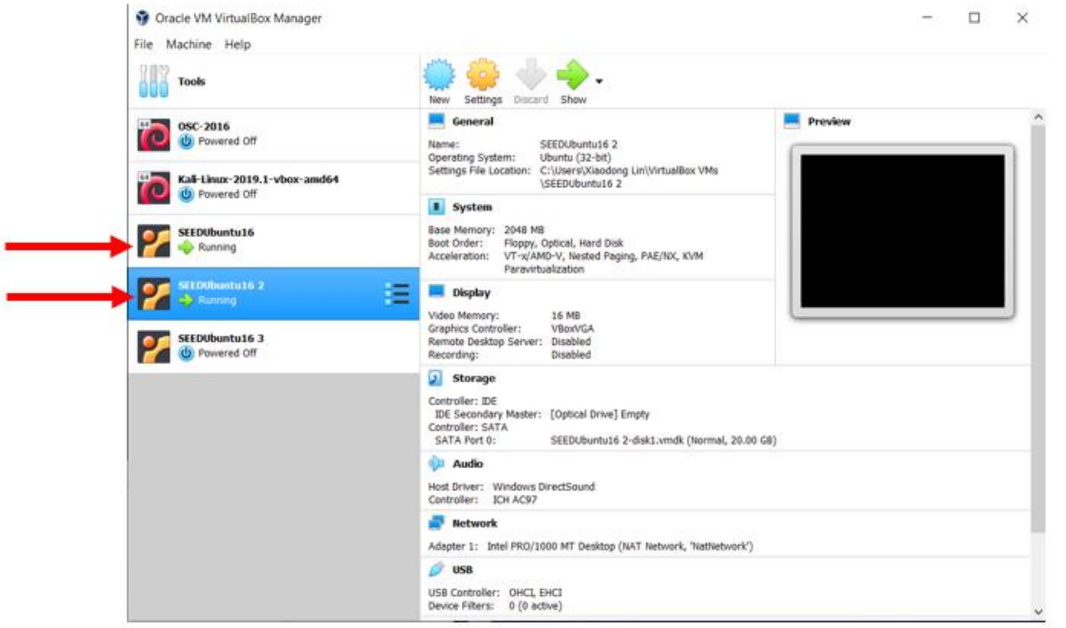
In this hands-on exercise, we will learn how to use open-source tools, Stunnel [1] and OpenSSL[6], to secure the communication over a public network. The purpose of this experiment is to better understand the concept of secure communication by transmitting an encrypted message over a public network without the message being able to be read by anyone sniffing the network.

This hands-on exercise will be graded. It has to be completed INDIVIUALLY, but you may want to discuss the lab exercises with your fellow students. After you have finished the lab exercises, please submit your answers (or lab report) to your TA before the due date. You must demonstrate your lab in your report through a step-by-step MOP (Method of procedure). Note that **for your reference, a sample report is provided (courtesy of Eddy Fung, a former student in my MITS 6100G Attack and Defence class). The sample report cannot be reused or redistributed for other purposes.**

2. Environment Setup

If you haven't set up your virtual Cyber Security Lab environment using VirtualBox on your laptop, please install and setup VirtualBox and Ubuntu 16.04 virtual machines by following instructions provided in **Lab 1**.

Please note, for this Hands-on Exercise, you will need to use two Ubuntu 16.04 VMs to experience in securing communication between two computers.



3. Figure 1. Required Lab Environment Settings

1) Install the **Stunnel** -- Universal SSL Wrapper onto two Ubuntu 16.04 virtual machines. To install **Stunnel** on Ubuntu, type

sudo apt-get install stunnel

4. Stunnel

Stunnel is an open source program that allows you to encrypt arbitrary TCP connections inside SSL (Secure Sockets Layer) available on both Unix and Windows. Stunnel can allow you to secure non-SSL aware daemons and protocols (like POP, IMAP, LDAP, etc) by having Stunnel provide the encryption, requiring no changes to the daemon's code [1].

As shown in the Figure 2, we run Netcat to listen on port 8181 on an Ubuntu 16.04 virtual machine, also known as **Host B** here. If we directly establish a TCP connection on **Host B** from another Ubuntu 12.04 virtual machine by using Netcat, also known as **Host A**, communication between two hosts are not protected and anyone may intercept network traffic and find out what have been exchanged between two hosts. However, if we first set up a secure channel between two hosts, using Stunnel[1] and then use Netcat to connect to the client side's Stunnel proxy on the **Host A** machine, the network traffic will be automatically forwarded to the listening TCP

server on **Host B** through the established secure Stunnel channel. As a result, the network traffic is protected by Stunnel.

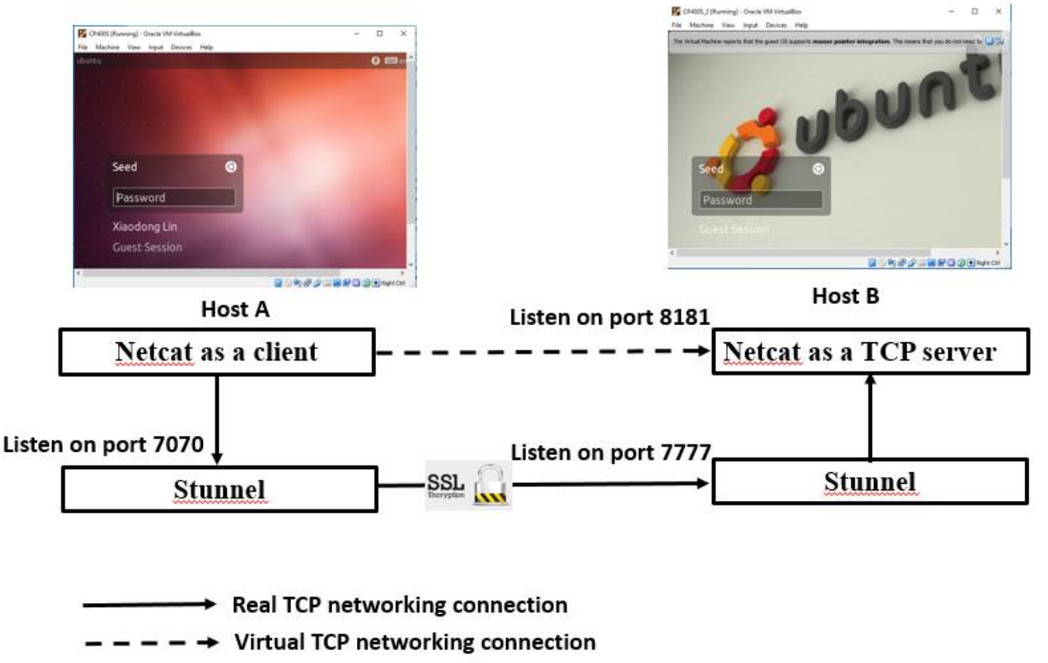


Figure 2: STunnel TCP Server & Client

In this lab, you are required to establish a secure channel by using Stunnel, which are shown in the above Figure 2. In the end, you can use Netcat to listen on a TCP port 8181 on one Ubuntu 16.04 virtual machine (referred to as **Host B**). Then, you can use Netcat to connect to the Listening TCP port 8181 from another Ubuntu 16.04 virtual machine (referred to as **Host A**), but over a secure SSL/TLS channel through Stunnel, as shown in Figure 2.

5. Secure Sockets Layer Protocol (SSL) [4,5]

Secure Sockets Layer (SSL) protocol is designed to provide secure communication. It performs server authentication, and optionally client authentication. With SSL, private information is protected through encryption, and a user is ensured through server authentication that he/she is communicating with the desired web site and not with some bogus web site. In addition, SSL provides data integrity, i.e., protection against any attempt to modify the data transferred during a communication session.

For convenience, we consider the case where an SSL session is set up to request a web page (or https). To better understand how SSL works, we show in Figure 3 the events and activities as seen by the web server. The SSL session starts when a web client (browser) connects to a secure web server, which is indicated by the reception of a TCP connection request on port 443 - the default port number assigned to https. The web server then proceeds with setting up the TCP connection, and this activity ends when the TCP connection is established. The web server then waits for a "client hello" message. When such a message is received, the server parses the message, and prepares a "server hello" message for transmission back to the client. Also, the server sends its public certificate to the client, the client validates it against CA certificates. Note that a public key Certificate (SSL Certificate) is required to be installed onto a web server to initiate a secure session with the web client (the browser). In other words, when you use Stunnel to secure the communication over a public network, you need to include a public key Certificate while configuring Stunnel on your server side.

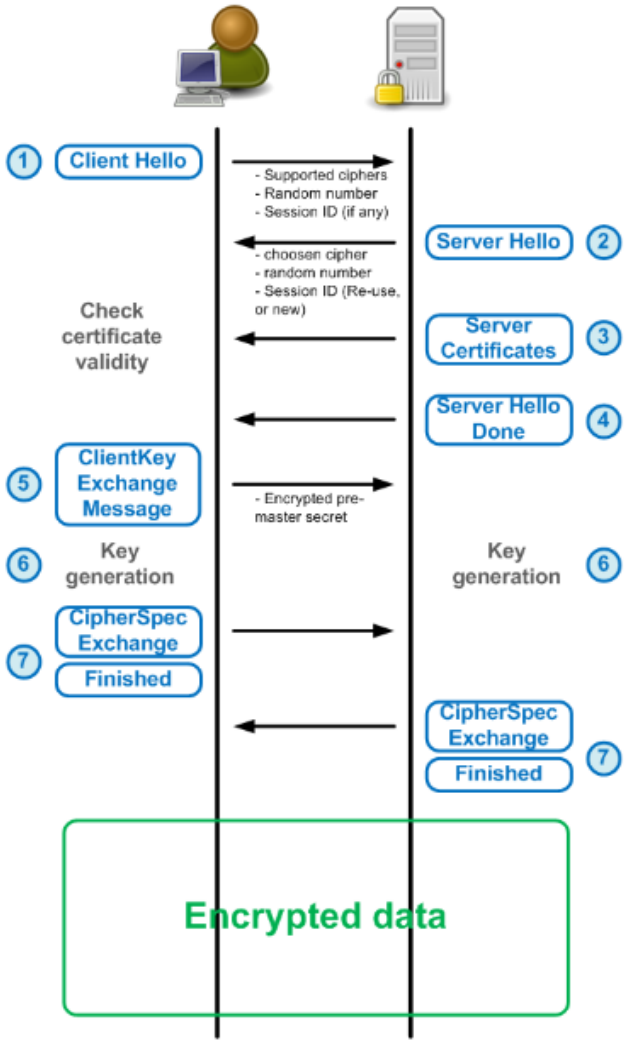


Figure 3. SSL session [5]

After the "server hello" message has been transmitted, the web server waits for a "client master key" message from the client. Upon receiving this message, the web server decrypts the client master key, and prepares a "server verify" message and transmits it to the client. At end of transmission, the server waits for a "client finish" message. When this message is received, the server prepares a "server finish" message for transmission to the client. The SSL handshake ends when this message is transmitted. Afterwards, the web server next waits for an http request from the client. Upon receiving this request, the web server prepares an http response. All data exchanges between the web server and the client are now encrypted. When this response is transmitted, the SSL session ends.

6. Public key encryption with OpenSSL

As mentioned in Section 4, when you use Stunnel to secure the communication over a public network, you need to include a public key Certificate while configuring Stunnel on your server side. It is because Stunnel uses SSL. This time, we need to create SSL Certificates using OpenSSL, an open-source implementation of the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. The openssl program can perform a wide variety of cryptographic operations. It is already installed on your Ubuntu VMs. Extensive documentation is available through the man pages as well as online resources, and through the program itself.

Next, you will need to create a self-signed SSL Certificate using OpenSSL. This self-signed SSL Certificate will be used by Stunnel on your server. The following commands create a 2048-bit private key and a self-signed certificate from scratch in the /home/seed/ folder:

```
cd /home/seed

openssl req -newkey rsa:2048 -nodes -keyout stunnel.pem -x509 -days 365 -out stunnel.pem
```

Answer the certificate signing request (CSR) information prompt to complete the process.

Basically, the command above is for creating a private key, creating a certificate using that key (a self-signed certificate), and then put into one file named "stunnel.pem" to use with Stunnel

Note that the order of contents of the .pem file is important. It should contain the unencrypted private key first, then a signed certificate. Your .pem file should look like the following

```
-----BEGIN RSA PRIVATE KEY-----
[encoded key]
-----END RSA PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
[encoded certificate]
-----END CERTIFICATE-----
```

Please note that in cryptography and computer security, a self-signed certificate is a security certificate that is not signed by a certificate authority (CA). Instead, it is signed by the owner itself, but these certificates are easy to make and do not cost money. However, they do not provide all of the security properties that certificates signed by a CA aim to provide [7, 8].

7. Configure Stunnel on the server side (**Host B**)

First, we create a "stunnel.conf" file in the "/etc/stunnel" directory:

sudo vi /etc/stunnel/stunnel.conf

An overall "stunnel.conf" file for Stunnel on the server side looks like the following:

```
#Stunnel server configuration file

# Provide the full path to your certificate-key pair file
key=/home/seed/stunnel.pem
cert=/home/seed/stunnel.pem

# up this number to 7 to get full log details
debug=7

# instruct Stunnel to run in the foreground
foreground = yes

client = no

[stunnel]
accept = 8888
connect = 3128
```

Note that the "client = no" part isn't necessary, Stunnel by default is set to server mode. **In order to successfully complete the lab exercise, you must tell Stunnel to listen on which TCP port (e.g., "accept = 8888") and which TCP port Stunnel has to forward accepted connections to (e.g., "connect = 3128"). Note that the above port numbers are only examples, and you must figure out what TCP ports are from the Figure 2.**

Enabling Stunnel by modifying the */etc/default/stunnel4* file

Note that SSL tunnels are disabled by default.

To enable Stunnel, modify the */etc/default/stunnel4* file

- Navigate to /etc/default
- Open stunnel4 using emacs
  sudo vi stunnel4
- Change the following line in the */etc/default/stunnel4* file

```
# Change to one to enable stunnel automatic startup
ENABLED=0
```

to

# Change to one to enable stunnel automatic startup
ENABLED=1

Otherwise, you may get this error when you try to start Stunnel

Starting Stunnel: Afterwards, you can start Stunnel for configuration to take effect, open a terminal and using the following command:

To start stunnel, run the following command as root:
To become root user type:
su -
When promoted provide the password “seedubuntu”

stunnel /etc/stunnel/stunnel.conf

By default, stunnel uses /var/log/secure to log its output.

To terminate stunnel, kill the process by running the following command as root:
kill `cat /var/run/stunnel4.pid`

If you edit the configuration file while stunnel is running, terminate stunnel and start it again for your changes to take effect

8. Configure Stunnel on the client side (**Host A**)

Similarly, we create a "stunnel.conf" file in the “/etc/stunnel” directory:

sudo vi /etc/stunnel/stunnel.conf

An overall “stunnel.conf” file for Stunnel on the client side looks like the following:

#Stunnel client configuration file

# up this number to 7 to get full log details
debug=7

# instruct Stunnel to run in the foreground
foreground = yes

# enable client mode
client = yes

[cis4510]
accept =8080
connect=[Server’s IP Address]:8888

In order to successfully complete the lab exercise, you must tell Stunnel to listen on which TCP port (e.g., "accept = 8080") and which TCP port on which server Stunnel has to forward accepted connections to (e.g., "connect = [Server's IP Address]:8888"). Note that the above port numbers and IP address are only examples, and you must figure out what TCP ports as well as the IP address of your server are from the Figure 2.

Enabling Stunnel by modifying the */etc/default/stunnel4* file

Note that SSL tunnels are disabled by default.

To enable Stunnel, modify the */etc/default/stunnel4* file

- Navigate to /etc/default
- Open stunnel4 using emacs
  sudo vi stunnel4
- Change the following line in the */etc/default/stunnel4* file

# Change to one to enable stunnel automatic startup
ENABLED=0

to

# Change to one to enable stunnel automatic startup
ENABLED=1

Otherwise, you may get this error when you try to start Stunnel

Starting Stunnel: Afterwards, you can start Stunnel for configuration to take effect, using the following command:

To start stunnel, run the following command as root:
To become root user type:
su -
When promoted provide the password "seedubuntu"

stunnel /etc/stunnel/stunnel.conf

By default, stunnel uses /var/log/secure to log its output.

To terminate stunnel, kill the process by running the following command as root:
kill `cat /var/run/stunnel4.pid`

If you edit the configuration file while stunnel is running, terminate stunnel and start it again for your changes to take effect

9. Final Lab Report

After you have done with your lab experiments, you need to submit a report to the TA by describing the experiment procedures and results with all support materials including important screenshots and configuration files. What is a good lab report? A good report is that if we use your report, it should be easy for us to set up and conduct the experiment and obtain the same results that you did. In other words, your experiment should be repeatable to us.

Reference:

[1] Stunnel -- Universal SSL Wrapper. http://www.stunnel.org/
[2] Secure Communication with Stunnel. http://linuxgazette.net/107/odonovan.html
[3] Virtual Appliance Marketplace. http://www.vmware.com/appliances/
[4] X. Lin, J. W. Wong, W. Kou, “Performance Analysis of Secure Web Server Based on SSL”, In Proc. Third International Workshop on Information Security (ISW 2000), Wollongong, NSW, Australia, December 20-21, 2000
[5] Baptiste Assmann. Benchmarking SSL Performance.
https://www.haproxy.com/blog/benchmarking_ssl_performance/
[6] OpenSSL. https://www.openssl.org/
[7] The Dangers of Self-Signed SSL Certificates
https://www.globalsign.com/en/ssl-information-center/dangers-self-signed-certificates
[8] Creating a Self-Signed SSL Certificate
https://devcenter.heroku.com/articles/ssl-certificate-self