2014

Hands-On Labs+

HeartBleed Bug: Risk Assessment and Mitigation

Table of Contents

Introduction

By now, you have heard about the HeartBleed Bug that has been identified and publicized. There has been a flurry of activity as distributions, vendors, consultants, websites and corporations all rush to address, fix or mitigate both the technical and political fallout of this security risk.

These types of situations happen more often than you might think, although they rarely get this type of public attention (which is both a good and bad thing). Since this is a very visible scenario that you will run into again and again throughout any career in Linux, we are going to take a look at how to deal with it.

Goals

This Hands-On Lab will demonstrate how to do a risk assessment of your system for the HeartBleed Bug. We will set up a common server scenario and walk through determining where the risks lie, how to address or mitigate them and finally how to protect our systems from being compromised.

Once you complete this lab, you will have a good understanding how to make corrections to your security configuration when attack vectors like the HeartBleed Bug come up.

Packages, Resources and Prerequisites

- Build-essential
- GCC
- Automake and make
- CheckInstall
- OpenSSL package and source
- Apache 2.2
- Fakeroot
- Dpkg-dev

The resources you will be accessing during the course of this lab are:

- Ubuntu 13.10 Server: Apache Web and OpenSSL Server
    - You will access this server in order to explore the risks of the HeartBleed Bug and how to mitigate/correct them

Prerequisites to this lab:

- A LinuxAcademy.com Lab+ Subscription

- Internet Access and SSH Client
  - You will need to connect to the public IP of the server in order to install and configure the packages listed above
    - SSH client can be Windows (i.e. Putty) or from another Linux system shell
  - For testing, use any browser from a system with internet access
- Login Information (Provided When The Server Starts Up)

Document Conventions

Just a couple of housekeeping items to go over so you can get the most out of this Hands On Lab without worrying about how to interpret the contents of the document.

When we are demonstrating a command that you are going to type in your shell while connected to a server, you will see a box with a dark blue background and some text, like this:

```
linuxacademy@ip-10-0-0-0:~$ sudo apt-get install package
[sudo] password for linuxacademy: <PASSWORD PROVIDED>
```

That breaks down as follows:

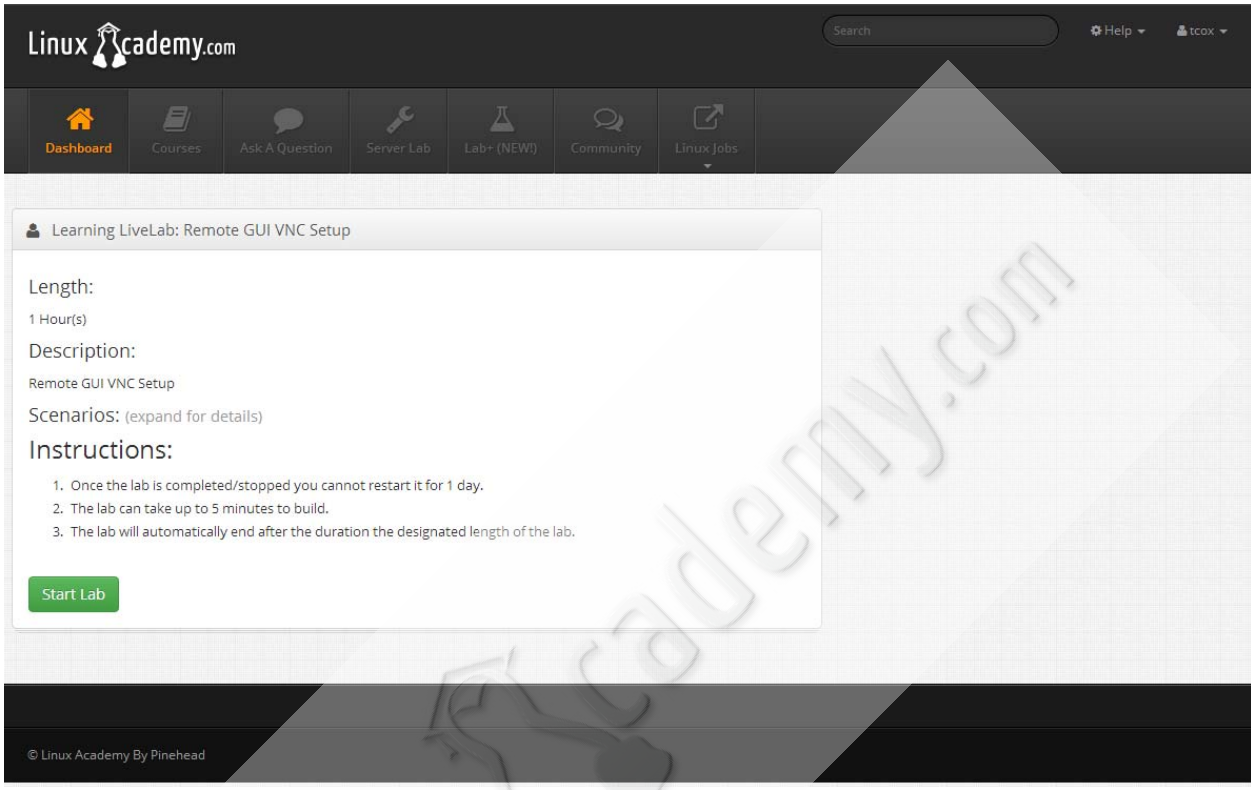
- The white text that looks something like "linuxacademy@ip-10-0-0-0:~$: "should be interpreted as the console prompt your cursor will be sitting at when you are logged into the server. You are not typing that into the console, it is just there. Note that the portion that says "ip-10-0-0-0" will be different for you based on the IP address of your system.
- The bold yellow text in any command example is the actual command you will type in your shell.
- Any lines subsequent to the bold yellow command that you type in is to be considered as the response you can expect from your command entry.

If you do not see the command prompt as the first line in the example, then all the white text is an example of a text, script or configuration file and is intended to be typed in its entirety (in the event you are instructed to create a previously non-existent file) or to compare against the contents of the file on your system.

One final convention, if you see a "~" at the end of a line in a command or text example, that will indicate that the line overflowed the end of line. Meaning you should just keep typing without hitting enter to start a new line until the natural end of the command.

General Process

When you are ready to begin the Hands-On Lab, log into your Linux Academy Lab+ subscription and navigate to the "Live Labs" section on the Dashboard. Once you choose the "HeartBleed Bug Scenario" Lab from the list, you will see a screen like the one below:



Learning LiveLab: Remote GUI VNC Setup

Length:
1 Hour(s)

Description:
Remote GUI VNC Setup

Scenarios: (expand for details)

Instructions:

1. Once the lab is completed/stopped you cannot restart it for 1 day.
2. The lab can take up to 5 minutes to build.
3. The lab will automatically end after the duration the designated length of the lab.

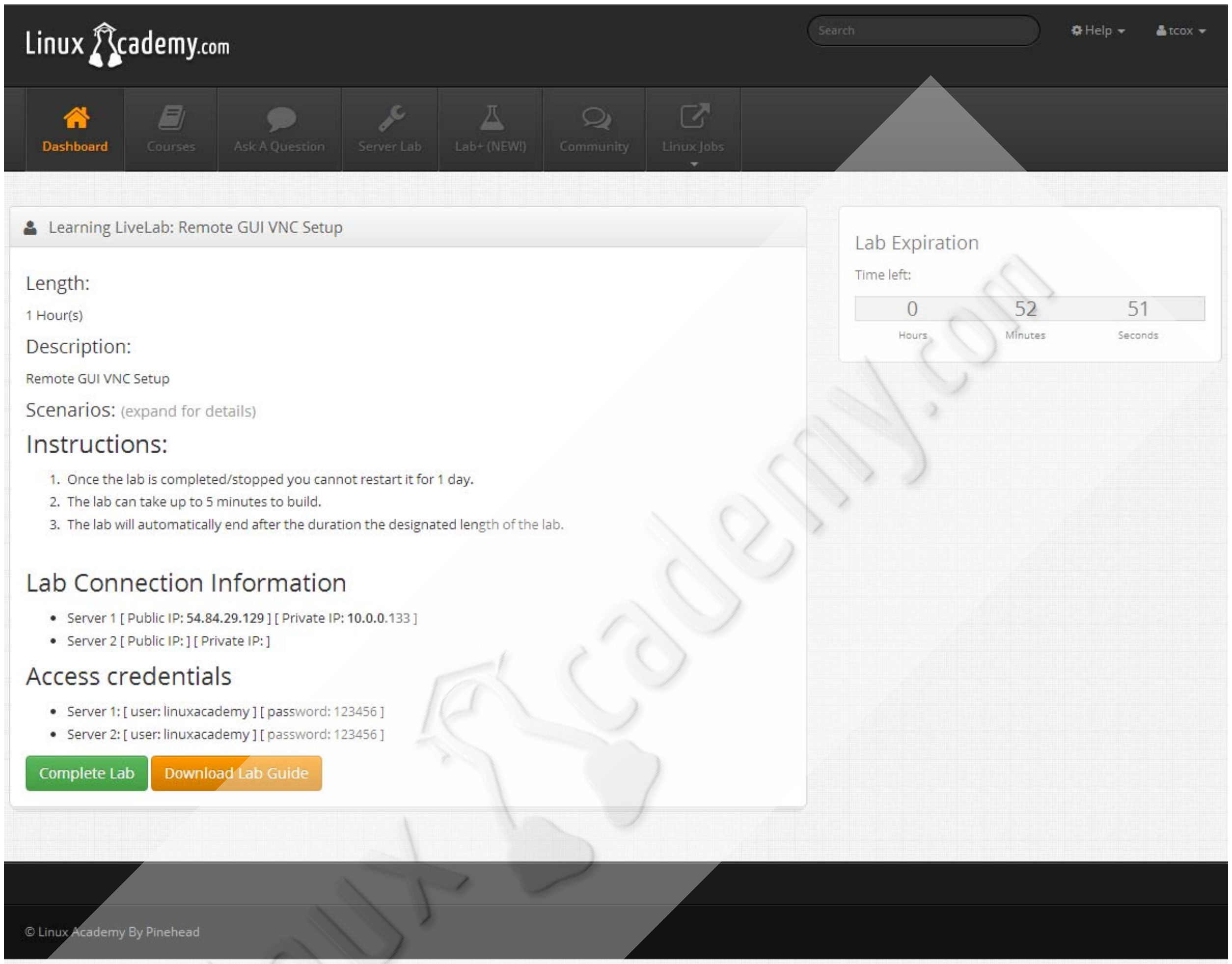
© Linux Academy By Pinehead

[Linux Academy Live Labs+ - Start Screen]

A few things to note before you start this process:

- When you launch the lab from this screen, it may take up to FIVE MINUTES for your servers to be deployed and be available for your use.
- Do not leave your desktop and come back, once the servers are launched, you will only have THREE HOURS to complete this lab from start to finish. After that point, the servers time out and are deleted permanently. Any and all work that you have done will then be lost.
- You can only use this lab ONCE PER DAY. If you try to use it more than that after completing the lab or the servers timing out, the screen will tell you when it will be available to you again.
- Other than those descriptions, you may retry any of the Labs+ labs as many times as you wish as long as you are a subscriber.

Once you have clicked on the 'Start Lab' button that you see above, a process will launch on our servers that will deploy the two servers we will use in our lab for testing. After a few minutes of processing (and you will see a status message that says "Creating Lab… Please Wait"), you should see a screen that looks like this one:



Learning LiveLab: Remote GUI VNC Setup

Length:
1 Hour(s)

Description:
Remote GUI VNC Setup

Scenarios: (expand for details)

Instructions:

1. Once the lab is completed/stopped you cannot restart it for 1 day.
2. The lab can take up to 5 minutes to build.
3. The lab will automatically end after the duration the designated length of the lab.

Lab Connection Information

- Server 1 [Public IP: 54.84.29.129] [Private IP: 10.0.0.133]
- Server 2 [Public IP:] [Private IP:]

Access credentials

- Server 1: [user: linuxacademy] [password: 123456]
- Server 2: [user: linuxacademy] [password: 123456]

Lab Expiration
Time left:

Hours	Minutes	Seconds
0	52	51

[Linux Academy Live Labs+ - Start Screen]

You will see all the information you need to access your servers from another system. Specifically, you need:

- The server public IP address
- Access credentials

One thing to note is that, in addition to the IP that you see above, the server may have another IP assigned to it in the 10.0.0.x subnet. This is a private IP address and will not route outside of your private server pool. Your server will have a static private address. We will be using the external IP address to connect over SSH as well as when configuring new OpenSSL package stack.

Linux Academy Lab Server – Scenario Setup

The best way to get a handle on something as high profile as the HeartBleed Bug is to truly get your hands dirty. We are going to set up a common (if not the most common) scenario where you have to assess and then mitigate or correct this vulnerability.

The situation is this – you are running an application that utilizes SSL certificates. Recent news of a huge security vulnerability called the "HeartBeat Bug" has made its way to your management team and further investigation indicates that the following web server and its certificates have been compromised by taking advantage of a bug in the OpenSSL implementation in use. The patch is not available in your Linux Distribution repository yet, so you need to assess the situation and mitigate the vulnerability by compiling your software from source and regenerating all the SSL keys that your application uses:

- Ubuntu 13.10 Server
- Apache 2.2 Web Server
- OpenSSL Server (vulnerable version)
- Serves SSL Content
- Uses Generated Certificates
    - These certificates have to be considered compromised if the server is determined to be affected by the bug
- Your Goals
    - Determine the status of your server
    - Obtain the appropriate source for patched OpenSSL Server
    - Remove the compromised package(s)
    - Compile and install the patched version
    - Regenerate, install and verify the new certificates

Risk Assessment

Fortunately, assessing the risk to your system is fairly easy. All current Linux distributions running OpenSSL Version 1.0 to 1.0.1f (including 1.0.2-beta) are affected. After performing an update to your distribution (in our case 'sudo apt-get upgrade'), you can check your version of the OpenSSL Server like so:

```
sudo openssl version
OpenSSL 1.0.1e 11 Feb 2013
```

In this case, we can see that we have a vulnerable version of the OpenSSL server and its associated libraries and dependencies. This means that if you have generated SSL keys and then installed SSL certificates (from anyone, the issuer does not matter), you have to assume that those decryption keys have been compromised and the certificates are no longer safe to use.

As of April 10, 2014, Ubuntu (nor Debian) have issued a fully patched version of the OpenSSL server. They have recompiled their package to exclude the bug and released that to their repositories, however, we are going to assume until the fully patched versions are available that we need to compile and install this mitigation ourselves.

Obtaining the Source Package

Since we have a vulnerable version of the package even after doing an update from our standard Ubuntu repositories, we need to obtain the source of the latest available package and compile the OpenSSL server ourselves. That process is simple enough and you probably know it well:

```
sudo apt-get source openssl
```

This will pull down the source package of the latest OpenSSL Server available in your configured repositories. Again, we have confirmed that we have a vulnerable version of the package, even if Ubuntu has recompiled and release a copy without the vulnerability.

In addition to pulling down the source, it will create a directory and expand the source in it in order to prepare for the compilation that will come after. Be sure you are in your home directory when you pull down the source so you can work locally.

We have our source package downloaded and set up for compile. Let's pull all the packages needed to compile everything in it. Please note that you will likely not need all these packages, they are listed here as a comprehensive list in order to be certain your compile succeeds. Your system will tell you what you have installed (or you can query aptitude for any of them prior to executing the command):

```
sudo apt-get install gcc automake make checkinstall fakeroot ~
dpkg-dev
```

This may take a few minutes to download and install everything but you will need all of these tools to be sure you get a successful compile of our package source.

Stop Apache and Remove the OpenSSL Package

We have everything we need to mitigate our reliance on the vulnerable OpenSSL package. Since out site is using SSL to server connections, we need to stop Apache so we can remove the module. Let's open a command prompt and execute the following:

```
sudo service apache2 stop
* Stopping web server apache2
```

At this point, you can remove your SSL package if you want, be sure to back up the certificates you have defined in your '/etc/ssl' directory (or whichever directory you installed them in during generation). You do not have to perform an uninstall, in our case, we will be compiling and then installing the OpenSSL package source right on top of our existing copy. If you choose to uninstall, execute:

```
sudo apt-get remove openssl
```

Compile and Install the Patched Version

Now we need to compile our package and remove the vulnerability. In our case, the OpenSSL Server version we have is not officially "patched", we are "mitigating" the risk by removing the function that has been compromised. The compromise has to do with how the server and client exchange a "heartbeat request" message that consists of a payload with a 16-bit integer length. Unfortunately, the bug is as a result of a lack of bounds checking of the message received from the client (in other words, instead of making sure the length of the reply is 16-bit, it will accept both the expected payload and then anything else following it). We need to remove that function by recompiling. Our first step is to get to the right place:

```
cd openssl-1.0.1e
```

Our source tree starts in this directory. Please note that the directory will be dependent on where exactly you installed the source package on your system. In this case, we installed in it our home directory and we executed the above command from there.

We have to configure the source tree for our system and pass in some parameters so our "makefile" will contain the path to install to as well as the "mitigation" for our vulnerability (note that we are mitigating our risk by creating a package that does not have the bug, we are not fixing the bug by correcting the errant code, we are excluding the function that can be compromised altogether).

At the command prompt, let's generate our makefile for use during compilation:

```
sudo ./config –prefix=/usr –openssldir=/usr/lib/ssl ~
-DOPENSSL_NO_HEARTBEATS
```

By default, the OpenSSL package is installed in /usr/local/ssl when compiled from source, a historical leftover default if not changed. In our case, for Ubuntu, we are placing it in the same location the package would be installed when downloaded and installed from the repositories. This also makes sure we overwrite the libraries and configurations with the freshly compiled copied. This command will end with something like the following on screen:

```
ssl.h => ../include/openssl/ssl.h
ssl2.h => ../include/openssl/ssl2.h
```

```
ssl3.h => ../include/openssl/ssl3.h
ssl23.h => ../include/openssl/ssl23.h
tls1.h => ../include/openssl/tls1.h
dtls1.h => ../include/openssl/dtls1.h
kssl.h => ../include/openssl/kssl.h
srtp.h => ../include/openssl/srtp.h
ssltest.c => ../test/ssltest.c
make[1]: Leaving directory `/home/user/openssl-1.0.1e/ssl'
making links in engines...
make[1]: Entering directory `/home/user/openssl-1.0.1e/engines'
making links in engines/ccgost...
```

As long as you do not see any errors here. We can move on to the compile. We have to complete a couple of compile steps and we are ready to install. Execute the following:

```
sudo make
. . .
make[2]: Entering directory `/home/user/openssl-1.0.1e'
Doing certs/demo
pca-cert.pem => e83ef475.0
pca-cert.pem => 8caad35e.0
dsa-pca.pem => de4fa23b.0
dsa-pca.pem => 24867d38.0
ca-cert.pem => 3f77a2b5.0
ca-cert.pem => 1f6c59cd.0
dsa-ca.pem => cbdbd8bc.0
dsa-ca.pem => 73912336.0
make[2]: Leaving directory `/home/user/openssl-1.0.1e'
make[1]: Leaving directory `/home/user/openssl-1.0.1e/apps'
making all in tools...
make[1]: Entering directory `/home/user/openssl-1.0.1e/tools'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/home/user/openssl-1.0.1e/tools'
```

And then the following (with output):

```
sudo make test
. . .
ALL TESTS SUCCESSFUL.
make[1]: Leaving directory `/home/user/openssl-1.0.1e/test'
OPENSSL_CONF=apps/openssl.cnf util/opensslwrap.sh version -a
OpenSSL 1.0.1e 11 Feb 2013
built on: Tue Apr 15 21:10:39 CDT 2014
platform: linux-x86_64
options:  bn(64,64) rc4(16x,int) des(idx,cisc,16,int) idea(int)
blowfish(idx)
compiler: gcc -DOPENSSL_THREADS -D_REENTRANT -DDSO_DLFCN -
DHAVE_DLFCN_H -DOPENSSL_NO_HEARTBEATS -Wa,--noexecstack -m64 -
DL_ENDIAN -DTERMIO -O3 -Wall -DOPENSSL_IA32_SSE2 -
DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -
DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -DMD5_ASM -DAES_ASM -DVPAES_ASM
-DBSAES_ASM -DWHIRLPOOL_ASM -DGHASH_ASM
```

```
OPENSSLDIR: "/usr/lib/ssl"
```

At this point, we have complete both our compile and the test of all the libraries that are available to link to with OpenSSL. All we have left is to install the package and then we can reissue our certificates and complete our risk assessment and mitigation

A simple command to install:

```
sudo make install
```

This will take the results of all our efforts, particularly the newly protected OpenSSL binaries and libraries and place them in our /usr/lib and /usr/lib/ssl directories as we indicated.

Let's move on to reissuing either our CSR keys (used to obtain an SSL certificate from a Certificate Authority) or reissuing our own certificates (in the case of a Self Signed SSL Certificate, which is what we will be using for the purposes of our lab).

Create a New Self-Signed Certificated

First, let's create a directory for our local SSL keys and certificate files.

```
sudo mkdir /etc/apache2/ssl
```

After we have a directory for our certificates, let's go ahead and create what we need:

```
sudo openssl req –x509 –nodes –days 365 –newkey rsa:2048 ~
-keyout /etc/apache2/ssl/apache.key –out ~
/etc/apache2/ssl/apache.crt
```

After you execute this command, you will be asked to enter a bunch of information that will be incorporated into your key and your certificate. Since this is a self-signed certificate that will show a browser warning in any event (since you are not considered a trusted certificate issuing authority), what you enter is largely unimportant. If we were generating just the key file to send to a valid issuer, it would. You will see something like the following:

```
Generating a 2048 bit RSA private key
......................+++
.........................................................................
..............................................................+++
writing new private key to 'apache.key'
-----
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) [AU]:US
State or Province Name (full name) [Some-State]:WY
Locality Name (eg, city) []:Lourdes
Organization Name (eg, company) [Internet Widgits Pty Ltd]:My Name
Company
Organizational Unit Name (eg, section) []:Information Technology
Common Name (e.g. server FQDN or YOUR name) []:ip-10-0-0-
100.linuxacademy.com
Email Address []:admin@linuxacademy.com
```

At this point, our certificate key and the certificate have been written to the previously created '/etc/apache2/ssl' directory.

Enable the Apache SSL Module

Like the vhost configuration, in Debian based distributions (of which Ubuntu is one), we have to enable modules as they are needed. We can either create the appropriate soft link manually or use the Debian utility to do so. Run the following command to enable the SSL module:

```
cd /etc/apache2
sudo a2enmod ssl
Considering dependency setenvif for ssl:
Module setenvif already enabled
Considering dependency mime for ssl:
Module mime already enabled
Considering dependency socache_shmcb for ssl:
Enabling module socache_shmcb.
Enabling module ssl
```

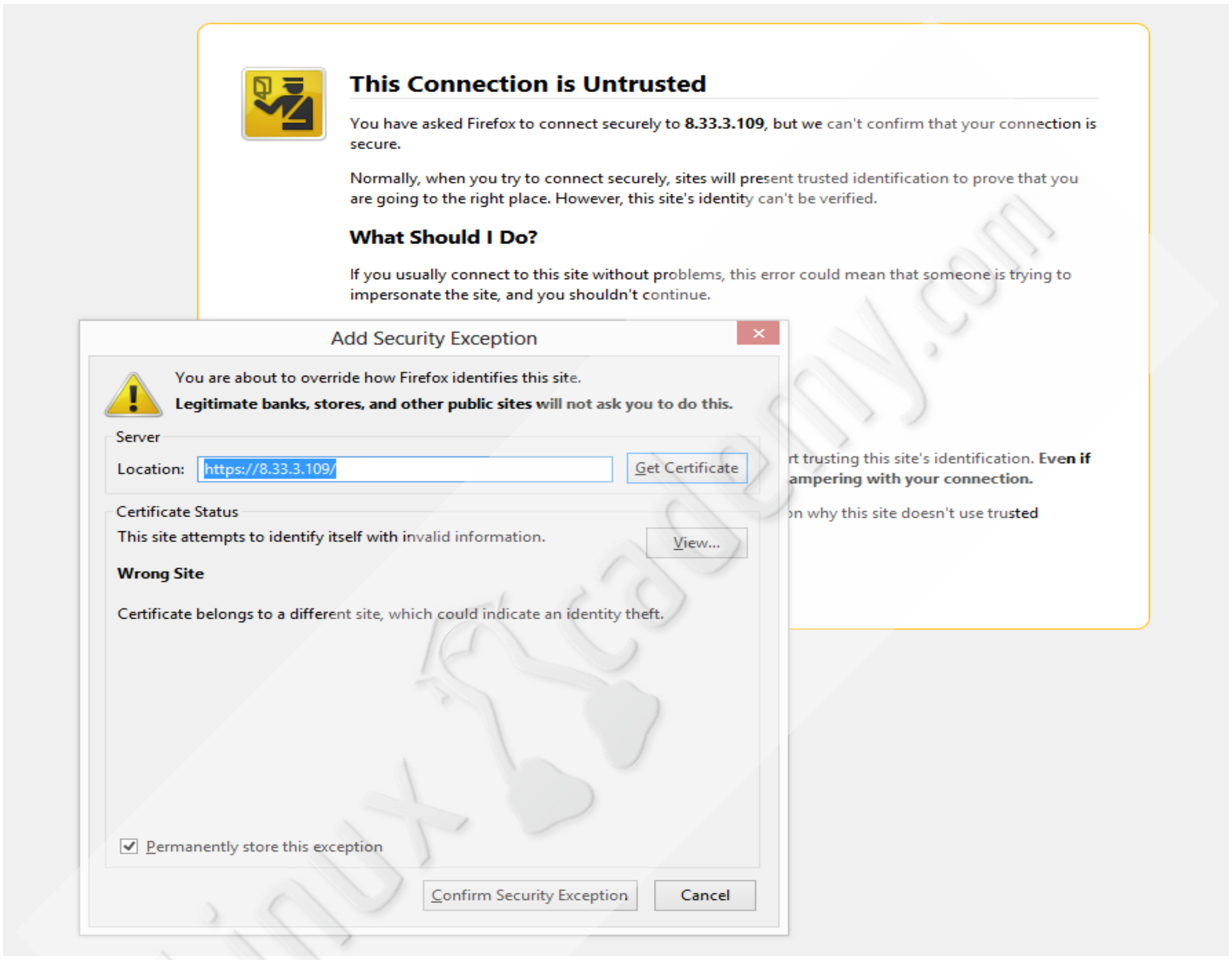
Keep in mind that some of the output may differ slightly for your installation depending on whether you are using the Linux Academy Servers or your own distribution. As long as you don't receive an error message that the SSL Module doesn't exist, you are ready to enable our certificates from within the system vhost files for our web server.

Test the Connection

Finally, all we need to do is open a browser and navigate to the external IP address of the site and confirm it comes up.

NOTE: This is a self-signed certificate and, although the traffic between your client browser and the Apache server is encrypted via SSL, your browser will warn you that the connection is potentially insecure. That is because it was not issued or verified by a trusted certificate authority (like Verisign or Entrust). We are issuing a self-signed certificate as a matter of demonstration and because it is generated at no cost. A certificate authority can issue verified certificates, but they come at a cost from $49 to more than $3500 depending on the encryption type and verification level (more expensive for securing credit card ecommerce transactions).

Your browser warning will look something like this:



This Connection is Untrusted

You have asked Firefox to connect securely to **8.33.3.109**, but we can't confirm that your connection is secure.

Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

What Should I Do?

If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

rt trusting this site's identification. **Even if ampering with your connection.**

on why this site doesn't use trusted

Add Security Exception

You are about to override how Firefox identifies this site.
Legitimate banks, stores, and other public sites will not ask you to do this.

Server

Location: https://8.33.3.109/

Certificate Status

This site attempts to identify itself with invalid information.

Wrong Site

Certificate belongs to a different site, which could indicate an identity theft.

☑ Permanently store this exception

However, you will then know that your system is indeed answering on port 443 externally, using your certificate. You can view the certificate details to confirm. You have now set up SSL and tested your configuration.

You can then visit the following location to confirm that your new site is not vulnerable to the HeartBleed Bug (https://filippo.io/Heartbleed/) – congratulations!

Appendix A: /etc/apache2/sites-available/default-ssl.conf Example

```
<IfModule mod_ssl.c>
<VirtualHost _default_:443>
      ServerAdmin webmaster@localhost

      DocumentRoot /var/www
      <Directory />
            Options FollowSymLinks
            AllowOverride None
      </Directory>
      <Directory /var/www/>
            Options Indexes FollowSymLinks MultiViews
            AllowOverride None
            Order allow,deny
            allow from all
      </Directory>

      ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
      <Directory "/usr/lib/cgi-bin">
            AllowOverride None
            Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
            Order allow,deny
            Allow from all
      </Directory>

      ErrorLog ${APACHE_LOG_DIR}/error.log

      LogLevel warn

      CustomLog ${APACHE_LOG_DIR}/ssl_access.log combined

      SSLEngine on

      SSLCertificateFile    /etc/ssl/certs/ssl-cert-snakeoil.pem
      SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key

      #   Server Certificate Chain:
      #   Point SSLCertificateChainFile at a file containing the
      #   concatenation of PEM encoded CA certificates which form the
      #   certificate chain for the server certificate. Alternatively
      #   the referenced file can be the same as SSLCertificateFile
      #   when the CA certificates are directly appended to the server
      #   certificate for convinience.
      #SSLCertificateChainFile /etc/apache2/ssl.crt/server-ca.crt

      #   Certificate Authority (CA):
      #   Set the CA certificate verification path where to find CA
      #   certificates for client authentication or alternatively one
      #   huge file containing all of them (file must be PEM encoded)
      #   Note: Inside SSLCACertificatePath you need hash symlinks
      #         to point to the certificate files. Use the provided
      #         Makefile to update the hash symlinks after changes.
      #SSLCACertificatePath /etc/ssl/certs/
      #SSLCACertificateFile /etc/apache2/ssl.crt/ca-bundle.crt

      #   Certificate Revocation Lists (CRL):
      #   Set the CA revocation path where to find CA CRLs for client
      #   authentication or alternatively one huge file containing all
      #   of them (file must be PEM encoded)
      #   Note: Inside SSLCARevocationPath you need hash symlinks
      #         to point to the certificate files. Use the provided
      #         Makefile to update the hash symlinks after changes.
      #SSLCARevocationPath /etc/apache2/ssl.crl/
      #SSLCARevocationFile /etc/apache2/ssl.crl/ca-bundle.crl

      #   Client Authentication (Type):
      #   Client certificate verification type and depth.  Types are
      #   none, optional, require and optional_no_ca.  Depth is a
      #   number which specifies how deeply to verify the certificate
      #   issuer chain before deciding the certificate is not valid.
      #SSLVerifyClient require
      #SSLVerifyDepth  10

      #   Access Control:
      #   With SSLRequire you can do per-directory access control based
      #   on arbitrary complex boolean expressions containing server
      #   variable checks and other lookup directives.  The syntax is a
      #   mixture between C and Perl.  See the mod_ssl documentation
      #   for more details.
      #<Location />
      #SSLRequire (    %{SSL_CIPHER} !~ m/^(EXP|NULL)/ \
      #            and %{SSL_CLIENT_S_DN_O} eq "Snake Oil, Ltd." \
      #            and %{SSL_CLIENT_S_DN_OU} in {"Staff", "CA", "Dev"} \
      #            and %{TIME_WDAY} >= 1 and %{TIME_WDAY} <= 5 \
      #            and %{TIME_HOUR} >= 8 and %{TIME_HOUR} <= 20       ) \
      #           or %{REMOTE_ADDR} =~ m/^192\.76\.162\.[0-9]+$/
      #</Location>

      #   SSL Engine Options:
      #   Set various options for the SSL engine.
      #   o FakeBasicAuth:
      #     Translate the client X.509 into a Basic Authorisation.  This means
that
      #      the standard Auth/DBMAuth methods can be used for access control.
The
```

```
      #       user name is the `one line' version of the client's X.509
certificate.
      #     Note that no password is obtained from the user. Every entry in the
user
      #     file needs this password: `xxj31ZMTZzkVA'.
      #   o ExportCertData:
      #     This exports two additional environment variables: SSL_CLIENT_CERT
and
      #     SSL_SERVER_CERT. These contain the PEM-encoded certificates of the
      #     server (always existing) and the client (only existing when client
      #     authentication is used). This can be used to import the certificates
      #     into CGI scripts.
      #   o StdEnvVars:
      #       This exports the standard SSL/TLS related `SSL_*' environment
variables.
      #    Per default this exportation is switched off for performance reasons,
      #     because the extraction step is an expensive operation and is usually
      #     useless for serving static content. So one usually enables the
      #     exportation for CGI and SSI requests only.
      #   o StrictRequire:
      #      This denies access when "SSLRequireSSL" or "SSLRequire" applied
even
      #      under a "Satisfy any" situation, i.e. when it applies access is
denied
      #     and no other module can change it.
      #   o OptRenegotiate:
      #     This enables optimized SSL connection renegotiation handling when
SSL
      #     directives are used in per-directory context.
      #SSLOptions +FakeBasicAuth +ExportCertData +StrictRequire
      <FilesMatch "\.(cgi|shtml|phtml|php)$">
            SSLOptions +StdEnvVars
      </FilesMatch>
      <Directory /usr/lib/cgi-bin>
            SSLOptions +StdEnvVars

       </Directory>

     </VirtualHost>

</IfModule>
```