2014

Hands On Labs+

Apache and Self Signed SSL Certificates

Table of Contents

Introduction

Security is a commonly overlooked topic of conversation when discussing Linux servers. Although it is not uncommon to talk about hardening the server itself, client access to the server can sometimes be an afterthought. Securing your Apache web server's content can be accomplished simply by using an encrypted web session. This is accomplished by installed an SSL Certificate in Apache.

We will talk about the security around generating SSL keys and then how to install a Self Signed Certificate can be a means to that end (and keep in mind, a certificate from a Third Party issuer is accomplished the same way).

Goals

This Hands On Lab will show you how to generate a valid Self Signed Certificate and Key for Apache that can be installed on the local server.

Although we will be using Ubuntu 14.04 LTS during the course of this document, the process is exactly the same for all Linux distributions with the exception of the location of the VHOST files on the web server. In Red Hat (RPM distributions), by default, all vhost entries are in the httpd.conf file itself and not in the 'sites-available and sites-enabled' directories as indicated here.

Packages, Resources and Prerequisites

- Apache 2.2+ Web Server
- OpenSSL Server
- Text or GUI Based Web Browser

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

- Ubuntu 14.04 LTS Server: Test Client
    - You will use this to install Apache and the certificate on and then connect for testing

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 configure Apache and generate the certificate
        - 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:

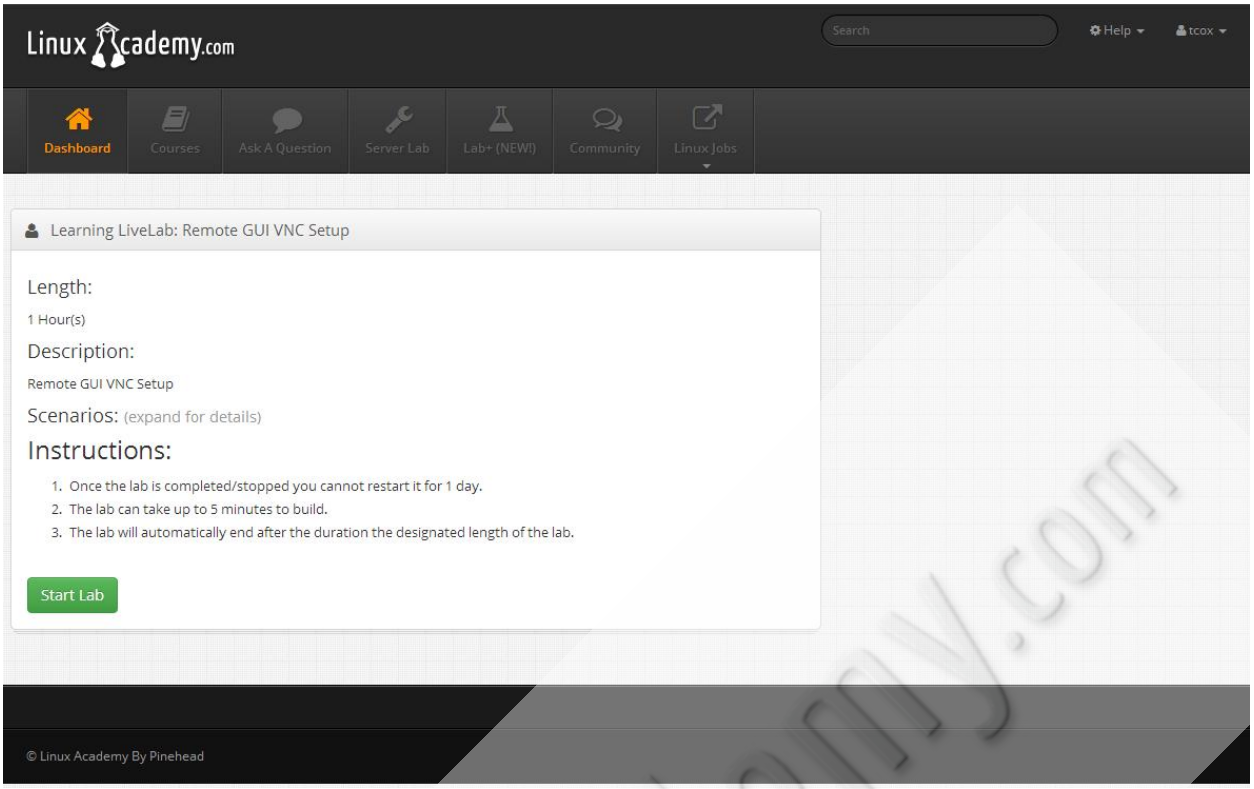
- The white text that looks something like "linuxacademy@ip-10-0-0-0:~$: "should be interpreted as the console prompt your cursor will be sitting at when you are logged into the server. You are not typing that into the console, it is just there. Note that the portion that says "ip-10-0-0-0" will be different 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 "Secure VNC" Lab from the list, you will see the screen below:

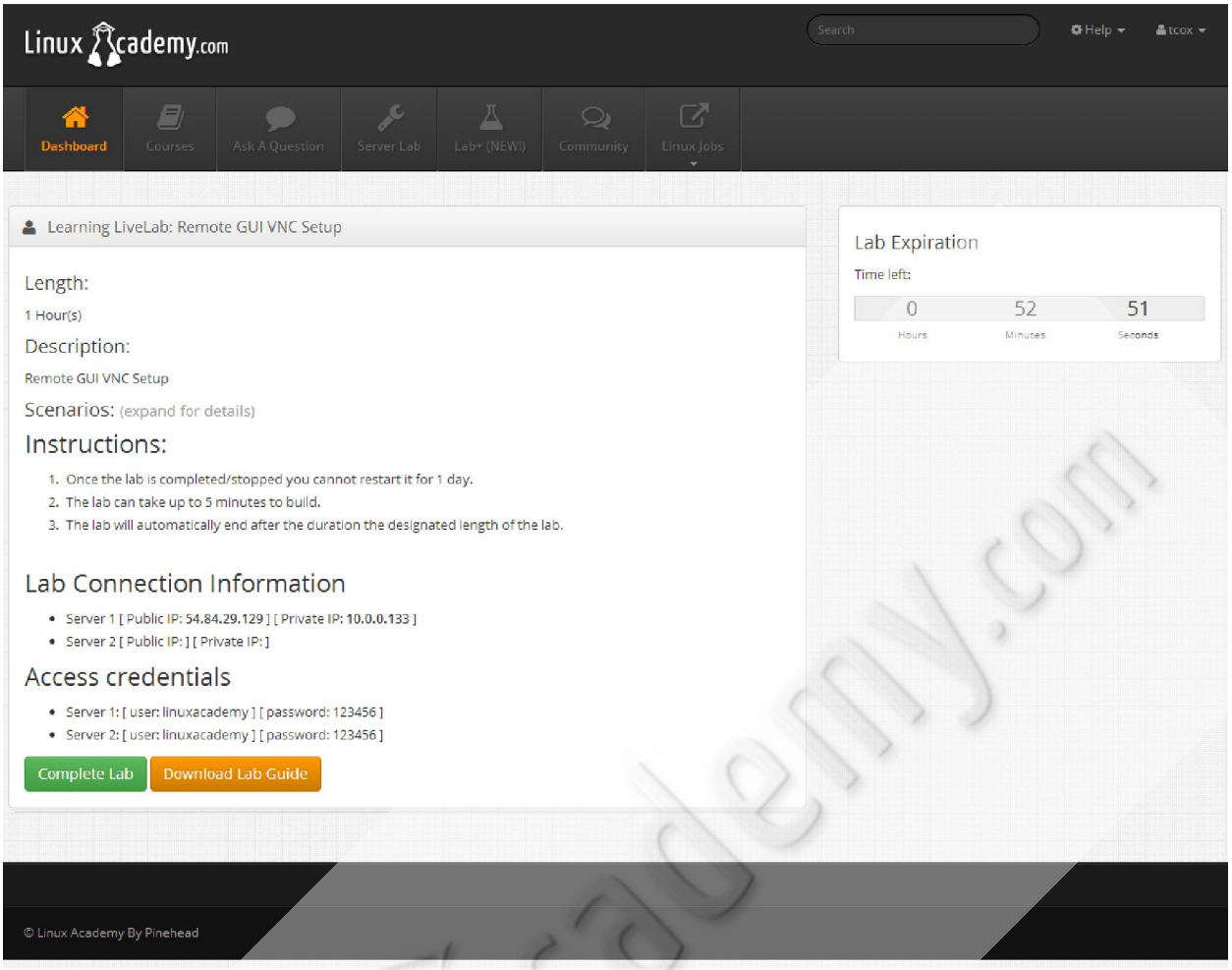


[Linux Academy Live Labs+ - Start Screen]

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 TWO 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:



[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 addresse
- Access credentials

One thing to note is that, in addition to the IP that you see above, the server will 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 of 10.0.0.100. We will be using the external IP address to connect over SSH as well as when configuring VNC.

Linux Academy Lab Server – Server Installation and Configuration

Connect to your Linux Academy Lab server over SSH using any Windows or Linux SSH client or shell. We will generate all of our configuration and certificates from the command line while connected to the Ubuntu 14.04 LTS Server in this lab.

Apache Installation

Once we are logged into our Linux Academy Lab Server, we need to install a couple of things in order to generate, install and test our secure server. Open a command prompt and type in the following:

```
sudo apt-get install apache2 apache2-doc apache2-utils openssl
```

This will install all the necessary packages and services to run Apache as well as the plugins, modules and libraries necessary to support SSL traffic. You can now test whether your system is running Apache by navigating to the external IP address of your server in a browser. You should get the default Apache "It Works" web page. If you do, we are ready to move on to the certificate installation.

Create a Self-Signed Certificated

Now that we have Apache installed and running, along with the SSL modules and support libraries we need, let's generate the keys and certificate files needed to secure our website traffic. 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.

Edit the Default SSL VHOST File

Our Apache server is running, we enabled the SSL Apache module and installed our certificate and key. Now, we just need to enable everything within the vhost file for our web server. Using your favorite command line editor, edit your '/etc/apaches/sites-available/default-ssl.conf' file to look contain the following:

```
<VirtualHost 54.85.117.33:443>
   ServerAdmin webmaster@localhost
   ServerName ip-10-0.0.100.linuxacademy.com:443

   SSLEngine on
   SSLCertificateFile /etc/apache2/ssl/apache.crt
   SSLCertificateKeyFile /etc/apache2/ssl/apache.key

   ...   ← Leave the rest of the file alone, defaults are fine

</VirtualHost>
```

Now we just have to activate the vhost file for our new SSL configured site. We can do this the same way we activated the SSL module:

```
sudo a2ensite default-ssl.conf
Enabling site default-ssl.
To activate the new configuration, you need to run:
service apache2 reload
```

Finally, we have to restart Apache so that the new configuration file is read in by Apache and it knows to answer for that IP address over HTTPS (SSL). We do that as directed by:

```
sudo service apache2 restart
```

A quick test can be performed locally to be sure we are now listening on port 443 (SSL) by doing the following:

```
telnet localhost 443
Trying ::1...
Connected to localhost.
Escape character is '^]'.
```

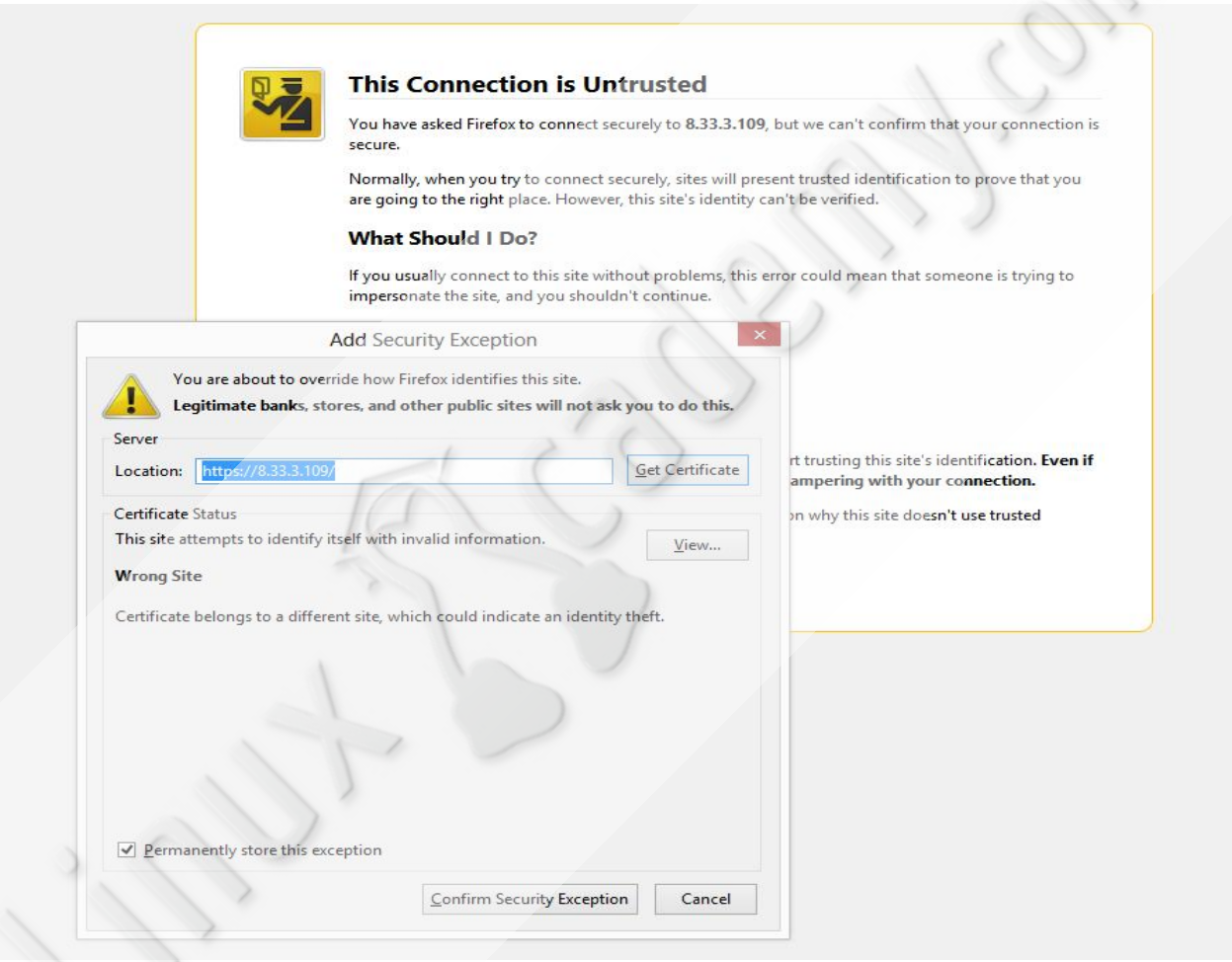
We have successfully enabled HTTPS traffic requests on our system and Apache is now listening on port 443 to incoming requests.

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:



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!

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>
```