# Apache 2 with SSL / TLS: Step-by-step instructions (Part III)

In this section, we will try to access the website URL again. If the above section has been successfully completed, each time it is requested, we can view and select the established certificate from the list.

**Review part I**
**Review section II**
**Review the beginning of Part III**

**Use Client certificate**

In this section, we will try to access the website URL again. If the above has been successfully completed, each time it is requested, we can view and select the established certificate from the list, as shown in Figure 6:

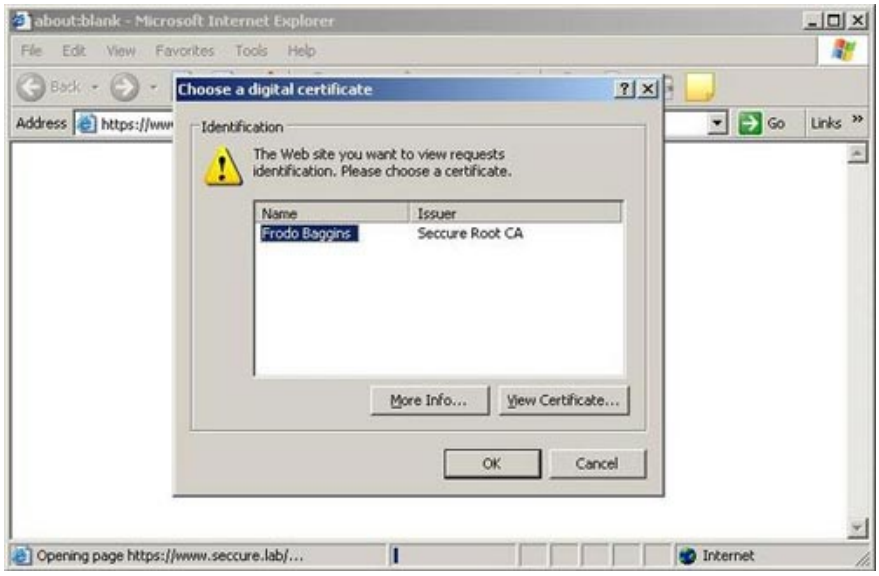


Figure 6. Select the client certificate when the background is done. Select the client certificate when the background is done.

After selecting the certificate, the user must enter the required password to decrypt the corresponding private key, as shown in Figure 7:


Figure 7. Enter the passphrase for the certificate. Enter the passphrase for the certificate .

Now, we can access the website safely, as illustrated in Figure 8:


Figure 8. Safe access, using accepted certificate. Secure access, using an approved certificate.

**Custom access control**

With additional server-side directories, we can control which parts of the website the individual or group of users are approved or denied. For example, when an organization must trade securely with many different companies, we can limit website access to just one specific company (O-secure) by adding these httpd.conf. instructions:

SSLRequire% {SSL_CLIENT_S_DN_O} eq "Seccure"

Another example illustrates how to allow access to only one office ( *OU = 'Secure Labs'* ) within a company ( *O = 'Seccure'* ).

SSLRequire% {SSL_CLIENT_S_DN_O} eq "Seccure" and
% {SSL_CLIENT_S_DN_OU} eq "Seccure Labs"

Or just provide access to some departments ( *OU = 'Seccure Labs or OU =' development '* ) in the same company ( *O =' Seccure '* ).

SSLRequire% {SSL_CLIENT_S_DN_O} eq "Seccure" and
% {SSL_CLIENT_S_DN_OU} in {"Seccure Labs", "Development"}

Finally, it is even possible to grant access to a specific user ( *CN = 'Frodo Baggins'* ) from a specific company ( *O = 'Seccure'* ):

SSLRequire% {SSL_CLIENT_S_DN_O} eq "Seccure" and
% {SSL_CLIENT_S_DN_CN} in {"Frodo Baggins"}

Note that the above environment variables can be provided for CGI scripts (including PHP and other languages) by adding the "+ StdEnvVars" parameter in the SSLOptions instruction. This component allows us to use DN names inside web applications (such as PHP and some other languages) to provide more detailed authentication or access controls.

SSLOptions + StdEnvVars

**Revoke a client certificate**

If a certificate becomes harmful or lost, what will you do? In this case, we need to revoke the certificate as stated in the previous sections. Next we have to copy a CRL file into the Apache directory as follows:

install -m 644 -o root -g sys crl.pem /usr/local/apache2/conf/ssl.crl/

We also need to make sure that " *SSLCARevocationFile* " in httpd.conf points to the file above:

SSLCARevocationFile /usr/local/apache2/conf/ssl.crl/crl.pem

Then we need to restart Apache to get the effect of the change. The revoked certificate will not allow access to the website, as shown in Figure 9:


Figure 9. Access attempt with revoked certificate. Try to access with revoked certificate.

**Chrooting server**

In order to improve your web server's security and make Apache have fewer vulnerabilities to be exploited, you should run Apache in a chrooted environment. Chrooting web server isolates the process of creating a new root directory so that they cannot see the rest of the server files.

The Apache chooting process is described in " *Securing Apache 2.0: Step-by-Step"* . So you should follow the steps mentioned above. Along with adding support for SSL / TLS, we need to install some additional libraries and create several subdirectories. In the case of FreeBSD 5.1, the list of libraries and directories is as follows:

cp /usr/lib/libssl.so.3 / chroot / httpd / usr / lib /
cp /usr/lib/libcrypto.so.3 / chroot / httpd / usr / lib /
cp -R /usr/local/apache2/conf/ssl.key / chroot / httpd / usr / local / apache2 / conf /
cp -R /usr/local/apache2/conf/ssl.crt / chroot / httpd / usr / local / apache2 / conf /
cp -R /usr/local/apache2/conf/ssl.crl / chroot / httpd / usr / local / apache2 / conf /

We also need to add random drives as follows. The example below is taken from FreeBSD 5.1:

ls -al / dev / * random
crw-rw-rw- 1 root wheel 2, 3 Jan 4 12:10 / dev / random
lrwxr-xr-x 1 root wheel 7 Jan 4 12:10 / dev / urandom -> random
cd / chroot / httpd / dev
mknod ./random c 2 3
ln -s ./random ./urandom
chown root: sys ./random
chmod 666./random

In the case of other operating systems, the reader can create the list of necessary files by using commands such as truss, strace, ktrace, etc., as described in detail in the section: 'Chrooting the server' in the article Security focus: ' *Secirity Apache: Step-by-step* '

Each time these steps are completed, you can run Apache in the enviroment environment as follows:

chroot / chroot / httpd / usr / local / apache2 / bin / httpd

**SSL / TLS attacks are known**

Although the SSL / TLS protocol theoretically has a high level of security, it may in fact be attacked by some of the following methods. There are many ways to attack, in which there are two most interesting ways:

***Man in the middle (MITM)***

In this type of attack, vandals prevent traffic from exchanging between the client and server, such as DNS spoofing, such as changing the direction of ARP, then acting as a client for the server and vice versa. During this attack the user's web browser did not connect directly to the destination server. But instead of destroying the host, it plays the web browser and acts basically like a proxy.

There are two pieces of information for administrators who want to protect the system against attacks: The good news is that the web browser warns users when the web server's identity cannot be verified, and may indicate a man-in-the-middle attack. -in-middle by displaying a warning message box. The bad news is, in fact, users often ignore messages. So if the user's web browser accepts connections to SSL websites that the identity cannot be verified, we can only trust the user's consciousness, and believe that they will press 'proceed' if Warning appears.

***Brute-force attacks on session keys:***

This type of attack is used when an attacker knows or captures part of the text sent in an SSL / TLS session, such as 'GET / HTTP / 1.0'. And an attacker can eavesdrop on the conversation in this session (like using tcpdump, Ethereal or other functions). Then he decodes the captured code using the possible keys, trying to find its variants in SSL / TLS encrypted data. If the key is found, the message may be used to decode the entire text in that session.

The good news is that the maximum number of keys to be tested is 2 ^ 128 when 128-bit symmetric encryption is used. Today, we believe it is enough to protect sessions several times a year. However, since CPUs are increasing in size, we cannot predict whether 12-bit synchronous keys can be considered safe. Specifically, hackers can access a large number of supercomputers.

In case of export class encoder (40 bits, and some 56 bits extensions) such as brute force attack can succeed in a certain amount of time, sometimes even a few days, sub belongs to the number of CPU. If you can use a strong encoder, you should use it explicitly.

In addition to the above two types, there are other quite dangerous types such as algorithms rollback attacks, timing attacks, traffic analysis, Bleinchenbacher's attacks . It's interesting to understand them. You can find more information in the documents of Bruce Schneier and David Wagner as well as many other materials on the internet.

**Typical SSL / TLS execution errors**

***The web browser does not recognize certificates signed by a CA***

The biggest error while executing SSL / TLS is the unbelievable web browser certificate of the web server signed by a CA. In other words, the CA certificate is not installed in the web browser. This makes Man in the middle attacks very easy. Because users have no way to verify the identity of the web server.

To avoid this problem, it is important to make sure the signed certificate (usually a trusted CA) is installed in the

web browser. If a local CA certificate is used, we must make sure that all clients that want to access it have installed that local CA certificate. The same principle applies to self-signed certificates.

***Certificate expired***

The web browser certificate should be refreshed before the previous certificate expires. Otherwise, it will have the same result because the web clients cannot authenticate the web server and once again cause SSL / TLS connections to be compromised by the Man-In-The-Middle attack. In this case, the user may be used to seeing a warning message saying that the certificate has expired without noticing that it is a fake certificate.

***Vulnerable versions of OpenSSL and Apache***

It is very important that you always run the latest versions of OpenSSL and Apache. Programmers who write larger code without errors are impossible. So we must always use the latest versions (because it is usually less error-prone than previous versions).

***SSL v2.0 acceptance, asynchronous verification (aNULL), and synchronous encryption (NULL) at Web server***

As we discussed, using an encoder that supports asynchronous format or does not require encryption should stop.

On the other hand, the danger is that the client may be fooled by the parameter settling abruptly below the security level of the connection. Therefore, we should not use SSL v2.0 protocol but should replace it with TSL v1.0 or SSL v3.0.

***Use weak encryption method***

Early days of SSL could only use 40-bit keys with symmetric encryption due to limited US government. Unfortunately, encrypted data this way can now be decoded in a short time. Therefore, a 40-bit or 56-bit key should not be used. Most modern web browsers support 128-bit synchronous encryption. And now, it is the shortest key you can use with SSL / TLS

***Using NIDS (Network Intruder Detection System)***

In addition to the NIDS that can decrypt SSL (like via the web browser private key), it is not easy to detect attacks on web applications. To detect the final result, either we use HIDS, or place the NIDS in the SSL / TLS traffic segment sent into the text, such as between SSL Reverse Proxy and the web server part). Otherwise, we cannot detect any attacks, except for denial-of-service attacks.

***Allow access not only through SSL but also unencrypted protocols (like HTTP)***

Setting up SSL / TLS and opening the self-opening port 443 / TCP for the web server does not mean anything if the user is still able to access via the unencrypted HTTP protocol, port 80 / TCP. Therefore, a protocol must be double-checked to protect content that is not accessible via HTTP or other protocols (such as FPT, Samba, NFS, .).

***Client mechanism is easily compromised***

When we focus on safety in web browsers, we can easily forget the security of client computers. If they are

damaged, the security of SSL / TLS is also destroyed. In this case the attacker can do so with workstations, such as replacing web browser certificates, installing keyloggers, changing the direction of the host to direct the web request, web server falsification, or even steal client certificates if they are marked as exportable
Therefore, if we structure the client under Administrative domain, we need to take care of it carefully. Use personal firewalls, anti-virus, anti-spyware and turn on automatic updates. Most importantly, the web browser must always be up to date.

*In short, you should do the following:*
- check the manufacturer's certificate revocation
- check the server certificate revocation
- SSLv2.0 should not be used but only TLSv1.0 or SSL v3.0
- shows warnings about inappropriate web server certificates
- use the same Session IDs / cookies for both SSL and HTTP

***Use the same session IDs / cookies for SSL and HTTP***

There may be an Apache configuration that accepts both HTTP and HTTPS with the same service request. For example, website information is accessed via HTTP, and part of the transaction is accessible via HTTPS. In this case we have to be very careful with web applications, do not use the same session IDs / cookies for both protocols. Otherwise, the vandals can sniff out HTTP transactions between the web server and the victim. And also try to use IDs to access the verified parts of the web application over SSL.

**Conclude**

This article concludes a series of three articles about Apache 2.0 configuration supporting SSL / TLS. It shows how to install SSL / TLS protocol to get the highest level of security and optimal performance. It also teaches how to create and revoke certificates, and how to practice them.