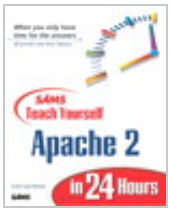
Setting Up a Secure Apache 2 Server

By Daniel Lopez

Nov 29, 2002

This chapter is from the book



Sams Teach Yourself Apache 2 in 24 Hours

Learn More Buy

SSL Configuration

The previous sections introduced the (not-so-basic) concepts behind SSL and you have learned how to generate keys and certificates. Now, finally, you can configure Apache to support SSL. `mod_ssl` must either be compiled statically or, if you have compiled as a loadable module, the appropriate `LoadModule` directive must be present in the file.

If you compiled Apache yourself, a new Apache configuration file, named `ssl.conf`, should be present in the `conf/` directory. That file contains a sample Apache SSL configuration and is referenced from the main `httpd.conf` file via an `Include` directive.

If you want to start your configuration from scratch, you can add the following configuration snippet to your Apache configuration file:

```
Listen 80
Listen 443
```

Related Resources

Store | Articles | Blogs

Hadoop Fundamentals LiveLessons (Video Training), 2nd Edition
By Doug Eadline
Downloadable Video
$239.99

Apache Hadoop YARN LiveLessons (Video Training)
By Arun C. Murthy, Vinod C. Vavilapalli, Doug Eadline
Downloadable Video
$239.99

Apache Hadoop YARN: Moving beyond MapReduce and Batch Processing with Apache Hadoop 2
By Arun C. Murthy, Vinod Kumar Vavilapalli, Doug Eadline, Joseph Niemiec, Jeff Markham
Book $31.99

See All Related Store Items

```
<VirtualHost _default_:443>
ServerName http://www.example.com
SSLEngine on
SSLCertificateFile \
/usr/local/ssl/install/openssl/certs/http://www.example
SSLCertificateKeyFile \
/usr/loca/ssl/install/openssl/certs/http://www.example.
</VirtualHost>
```

With the previous configuration, you set up a new virtual host that will listen to port 443 (the default port for HTTPS) and you enable SSL on that virtual host with the SSLEngine directive.

You need to indicate where to find the server's certificate and the file containing the associated key. You do so by using `SSLCertificateFile` and `SSLCertificateKeyfile` directives.

Starting the Server

Now you can stop the server if it is running, and start it again. If your key is protected by a pass phrase, you will be prompted for it. After this, Apache will start and you should be able to connect securely to it via the `https://http://www.example.com/` URL.

If you compiled and installed Apache yourself, in many of the vendor configuration files, you can see that the SSL directives are surrounded by an `<IfDefine SSL>` block. That allows for conditional starting of the server in SSL mode. If you start the `httpd` server binary directly, you can pass it the `-DSSL` flag at startup. You can also use the `apachectl` script by issuing the `apachectl startssl` command. Finally, if you always want to start Apache with SSL support, you can just remove the `<ifDefine>` section and start Apache in the usual way.

If you are unable to successfully start your server, check the Apache error log for clues about what might have gone wrong. For example, if you cannot bind to the port, make sure that another Apache is not running already. You must have administrator privileges to bind to port 443; otherwise, you can change the port to 8443 and access the URL via `https://http://www.example.com:8443`.

Configuration Directives

`mod_ssl` provides comprehensive technical reference documentation. This information will not be reproduced here; rather, I will explain what is possible and which configuration directives you need to use. You can then refer to the online SSL documentation bundled with Apache for the specific syntax or options.

Algorithms

You can control which ciphers and protocols are used via the

`SSLCipherSuite` and `SSLProtocol` commands. For example, you can configure the server to use only strong encryption with the following configuration:

```
SSLProtocol all
SSLCipherSuite HIGH:MEDIUM
```

See the Apache documentation for a detailed description of all available ciphers and protocols.

Client Certificates

Similarly to how clients can verify the identity of servers using server certificates, servers can verify the identity of clients by requiring a client certificate and making sure that it is valid.

`SSLCACertificateFile` and `SSLCACertificatePath` are two Apache directives used to specify trusted Certificate Authorities. Only clients presenting certificates signed by these CAs will be allowed access to the server.

The `SSLCACertificateFile` directive takes a file containing a list of CAs as an argument. Alternatively, you could use the `SSLCACertificatePath` directive to specify a directory containing trusted CA files. Those files must have a specific format, described in the documentation. `SSLVerifyClient` enables or disables client certificate verification. `SSLVerifyDepth` controls the number of delegation levels allowed for a client certificate. The `SSLCARevocationFile` and `SSLCARevocationPath` directives enable you to specify certificate revocation lists to invalidate certificates.

Performance

SSL is a protocol that requires intensive calculations. `mod_ssl` and OpenSSL allow several ways to speed up the protocol by caching some of the information about the connection. You can cache certain settings using the `SSLSessionCache` and `SSLSessionCacheTimeout` directives. There is also built-in support for specialized cryptographic hardware that will perform the CPU-intensive computations and offload the main processor. The `SSLMutex` directive enables you to control the internal locking mechanism of the SSL engine. The `SSLRandomSeed` directive enables you to specify the mechanism to seed the random-number generator required for certain operations. The settings of both directives can have an impact on performance.

Logging

`mod_ssl` hooks into Apache's logging system and provides support for logging any SSL-related aspect of the request, ranging from the protocol

used to the information contained in specific elements of a client certificate. This information can also be passed to CGI scripts via environment variables by using the `StdEnvVars` argument to the `Options` directive. You can get a listing of the available SSL variables at http://httpd.apache.org/docs-2.0/ssl/ssl_compat.html.

The `SSLOptions` Directive

Many of these options can be applied in a per-directory or per-location basis. The SSL parameters might be renegotiated for those URLs. This can be controlled via the `SSLOptions` directive.

The `SSLPassPhraseDialog` directive can be used to avoid having to enter a pass phrase at startup by designating an external program that will be invoked to provide it.

Access Control

The `SSLRequireSSL` directive enables you to force clients to access the server using SSL. The `SSLRequire` directive enables you to specify a set of rules that have to be met before the client is allowed access. `SSLRequire` syntax can be very complex, but itallows an incredible amount of flexibility. Listing 17.1 shows a sample configuration from the `mod_ssl` documentation that restricts access based on the client certificate and the network the request came from. Access will be granted if one of the following is met:

- The SSL connection does not use an export (weak) cipher or a NULL cipher, the certificate has been issued by a particular CA and for a particular group, and the access takes place during workdays (Monday to Friday) and working hours (8:00 a.m. to 8:00 p.m.).
- The client comes from an internal, trusted network.

You can check the documentation for `SSLRequire` for a complete syntax reference.

Listing 17.1 `SSLRequire` Example

```
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]+$/
```

Reverse Proxy with SSL

Although at the time this book was written the SSL reverse proxy

functionality was not included in `mod_ssl` for Apache 2.0, it is likely to be included in the future. That functionality enables you to encrypt the reverse proxy connection to backend servers and to perform client and server certificate authentication on that connection. The related directives are `SSLProxyMachineCertificatePath`, `SSLProxyMachineCertificateFile`, `SSLProxyVerify`, `SSLProxyVerifyDepth`, `SSLProxyCACertificatePath`, `SSLProxyEngine`, and `SSLProxyCACertificateFile`. Their syntax is similar to their regular counterparts. You can find more information about the Apache reverse proxy in Hour 15.

Discussions

Comments for this thread are now closed.

Comments Community Login

Recommend 1 Sort by Oldest

Abid • 3 years ago

Simply put, and this makes it the best article I have ever read!

Cheers,
Abid

• Share ›