DebuggingSSLProblems

Debugging SSL Problems

Here are some tips on what to do if the SSL connection to your server just isn't working as it should.

This article reflects the limited knowledge of it's author(s). If you discover anything incorrect when reading this article, you are asked to please either correct the text, or to leave a note in the text stating the problem.

Helpful documents

The SSL topic is a non trivial one. Do try to read and understand the documentation available:

- Apache SSL/TLS encryption documentation
- Openssl documentation
- SSL alert messages

Make sure you are following the howtos very closely and you do understand what they are doing. Even small SSL missconfigurations can prevent completely your server from communicating with clients.

Understanding modssl's components

Modssl does not implement the SSL protocol. It uses the openssl library to do the SSL negotiation, handshaking and encoding into the SSL protocol.

That has the implication that if you need to debug what's happening during a connection you'll need to read openssl's documentation.

However the configuration of the handshake phase, that is:

- which certificates you want to be used by your server
- which certificates are to be sent to the client
- which certificates the client should send back to the server (in the case of client authentication)

are done with modssl means, in the apache configuration.

That is as far as the author of this article understands, modssl proper is only there to properly parametrize the openssl library, as required by a web server.

Understanding SSL communications setup

SSL/TLS Strong Encryption: An Introduction provides some intermediate level information on how SSL communication works, particularly the paragraph Secure Sockets Layer (SSL).

When an SSL communication is being set up, all the phases up to the final data transfer, that is the handshaking and certificate exchanges are done unencrypted. That means they can be examined and thus debugged from the outside of the two communication parties.

Debugging tools

Since, as noted in the last paragraph the setup of the SSL connection is not encrypted, we can sniff the traffic. That can be done with:

- Wireshark or
- Microsoft Network Monitor(runs on Windows only)

which both include SSL protocol dissectors, and thus are able to decode and display SSL handshakes in a human understandable format.

If you have to transfer traffic seen on a server to your own machine for local analysis, then you can use tcpdump.

If you need to analyse traffic that is happening during the data transfer phase, then you'll need:

- sslsniff or
- ssldump

Both are able to decode traffic when given the appropriate certificate keys.

Certificates can be analyzed with the

- openssl command line tool

Enable SSL logging

The first step when debugging SSL problems is to setup proper logging:

```
<IfModule mod_ssl.c>
  ErrorLog /var/log/apache2/ssl_engine.log
  LogLevel debug
</IfModule>
```

See also:

- http://httpd.apache.org/docs/trunk/ssl/ssl_howto.html#logging
- http://httpd.apache.org/docs/trunk/mod/core.html#loglevel

Unfortunately the "info" LogLevel is not enough and "debug" is overkill. modssl by Ralf S. Engelschall on which Apache's modssl is based had a "trace" Level, which is still present in Apache's modssl source code. But it is not known how that "trace" log level can be activated from the configuration file.

Making sure that the browser trusts the certificate

Internet Explorer (under Internet Options->Content->Certificates) and Firefox both offer an interface for certificate management. It appears that Firefox will trust a certificate that the user installs even if it can't follow and verify the certificate chain. In contrast Internet Explorer will **not** trust a certificate where it can't verify the certificate.

Also Internet Explorer has a very comprehensive and well structured certificate management interface, that is helpful for seeing certificate paths and certificate properties.

Unfortunately IE is not helpful at all in its failure mode. When something's wrong, it will not finalize the setup of the SSL connection and not display any useful error. FF instead will at least display a semi useful error. Additionaly, since FF is using the openssl library as its SSL engine, Firefox' error messages correspong to openssl's alert messages.

Manually verifying certificates

You can use the openssl command line tool to do all sorts of certificate manipulation and analysis tasks:

- Verify that a private key matches a certificate (originally from http://kb.wisc.edu/middleware/page.php?id=4064)

```
$ (openssl x509 -noout -modulus \
          -in /etc/apache2/ssl.crt/www.mysite.org.crt | openssl md5 ;\
   openssl rsa  -noout -modulus
          -in /etc/apache2/ssl.key/www.mysite.org.key | openssl md5) \
  | uniq
```

- display the RSA private key:

```
$ openssl rsa -in /etc/apache2/ssl.key/www.mysite.org.key -noout -text
```

- display a X509 SSL certificate:

```
$ openssl x509 -in /etc/apache2/ssl.crt/www.mysite.org.crt -noout -text
```

- verify a certificate:

```
$ openssl verify -CAfile ca.crt www.mysite.org.crt
```

Finding out what caused a handshake to fail

If client and server fail to setup an SSL communication channel between them, you'll see something like the following in apache's ssl log (see the paragraph on "SSL Error Loggin" on how to set it up):

```
[Thu Oct 06 16:39:06 2011] [debug] ssl_engine_kernel.c(1791): OpenSSL: Exit: error in SSLv3 read client
certificate B
[Thu Oct 06 16:39:06 2011] [error] Re-negotiation handshake failed: Not accepted by client!?
```

The log entry is only half useful, since first it doesn't say what exactly the reason for the client not accepting the certificate was, and second in this specific case it's missleading, because in fact it was the server that told the client that id wouldn't accept the certificate that the client was presenting to it.

A more specific reason for the communications breakdown can be found in the SSL protocol trace (see the "Debugging tools" section on how to do a trace).

This document explains how to dissect the handshake and how to find the relevant message containing the specific error code. Note that one doesn't need the Microsoft Network Monitor to do the message dissecting: Wireshark works equally well.

The important thing to take away from the the document is that SSL contains an alert protocol, that can be seen and found in the transmitted TCP packets of an SSL communication, that contains an error code specifying the reason why a communication failed to be set up.

`http://tpo.sourcepole.ch/opaque/wireshark-https.png`|Wireshark screenshot

As you can see in the screenshot, the two bytes inside the "Alert Message" contain the error code "2f". That error code can be looked up in the respective rf
. In this case it's the code 47 (0x2f), which means "illegal_parameter" - there was some property of the certificate that the server 💡 didn't like and refuses to accept. In our case the server was expecting a different issuer CN.