http://www.coincoin.fr.eu.org/?Finding-out-what-TLS-SSL

Finding out what TLS/SSL cryptography people actually get with your servers

- 6- Webographie -

Date de mise en ligne : lundi 18 février 2013

Finding out what TLS/SSL cryptography people actually get with your servers

One of my hobbies is slowly improving the SSL (okay, TLS) security settings on our various TLS-enabled servers, in pursuit of both better practical security with real clients and things like forward secrecy. In an ideal world things would come preconfigured with the best setups possible, but that doesn't always happen in the real world (note that those settings are from 2010, which means that they are now obsolete). Part of doing a good job of this is testing things to make sure that the server settings actually do what I want them to do, especially with real clients. There are so many SSL/TLS bits and it is easy to miss something or set configurations that look good but which will have no meaningful effect in the real world when you interact with real (and imperfect) clients. Sadly, this is more difficult than you would like. Much more difficult.

There are several things going on to get in the way. The first question is what cipher suites your server actually supports and what your specifications have turned into in practice. If you're using OpenSSL (as most people are) the way to find this out is with '`openssl ciphers -v <ciphers spec>`'. As a bonus this will print out detailed information about each of the cipher suites, showing you the key exchange, certificate authentication, stream encryption, stream MAC, and which SSL/TLS standard things came from. This listing comes out in the server's preference order, most preferred first.

(If you want to see how your web server's TSL settings stack up in general, I quite like Qaulys's SSL Server Test. This doesn't help for internal web servers or for things like IMAP servers.)

There's an immediate gotcha for the unwary : the server preference often doesn't matter. In the TLS handshake the client gives the server a list of what cipher suites it supports, in its preference order, and **servers usually defer to the client's preferences**. You want to turn this off, forcing use of the server's preference order instead of the client's.

The next question is what cipher suites any particular client supports and what cipher suite was actually picked for a conversation (which is the real test). In an ideal world clients would tell you at least the latter (either natively or with an extension). In the real world, not so much ; many clients give you little or no information and you need go around behind their back to get it.

(For example Firefox won't tell you what key exchange was used for a

HTTPS connection, only what stream cipher is in effect for it (I'm pleased to see that modern versions of Chrome will tell you both), and I can't see how to get Thunderbird to tell me anything about a TLS protected IMAP connection.)

Fortunately the initial TLS handshake is mostly unencrypted, which means that we can snoop in on the conversation and see the client's list of supported cipher suites as well as the server's actual choice of cipher suite. If it works for you, the most convenient tool for doing this is ssldump. If it doesn't the best general tool for this is probably Wireshark, which has full TLS/SSL protocol decoding. In theory you can use `tshark` to dump this in ASCII from the command line ; in practice I haven't been able to get this working as nicely as the GUI.

(Since I already had to reconstruct this stuff from my cryptic notes once, it's clearly high time I wrote it down in a somewhat more comprehensible form.)

Sidebar : ssldump and Wireshark

I once had `ssldump` working, but these days all it does is dump the initial ClientHello message and then report '`ERROR: Length mismatch`'. This is actually good enough if all you care about is knowing what cipher suites the client supports (and what its preference order is).

There is probably some clever way to use `tshark` options to display just the TLS parts of the packets, but at the moment the best I can do is :

```
tshark -i INTERFACE -R ssl.handshake -V -p -n "host IP and port WHATEVER"
```

This unfortunately dumps a verbose decode of everything in the TLS handshake packets, from the raw frame on up.

If you want just the ClientHello and the ServerHello, you can use '

```
-R "ssl.handshake.type == 1 or ssl.handshake.type == 2"
```

'.

Cet article est repris du site http://utcc.utoronto.ca/~cks/space/...