sslcaudit 1.0rc2
User Guide

May 7th 2012

GREMWELL BVBA
Sint-Katherinastraat 24, 1742 Ternat
Tel: +32 2 215 53 58 Fax: +32 2 791 98 38

Document Properties

Subject	Sslcaudit 1.0rc2 User Guide
Author	Alexandre Bezroutchko E-mail: abb@gremwell.com Web site: http://www.gremwell.com/
Document history	1/05/2012 - Initial version, for sslcaudit 1.0rc1 7/05/2012 - Updated for sslcaudit 1.0rc2

Table of Contents

1 INTRODUCTION

The goal of sslcaudit project is to develop a utility to automate testing SSL/TLS clients for resistance against MITM attacks. It is useful for testing thick clients, mobile applications, appliances, pretty much anything communicating over SSL/TLS over TCP.

This document is based on sslcaudit version 1.0rc2. It contains some background information, explanations of how the tool works, and several examples. An impatient reader can jump directly to 4 HOW TO USE THE TOOL.

2 BACKGROUND INFORMATION

SSL/TLS suite of protocols are widely used to protect confidentiality and integrity of communications over untrusted networks. For protection to be effective, client and server both have to be implemented correctly. Security properties and common implementation flaws in servers are well understood and documented [WIKI-TLS, SCANIT-SSL, OWASP-TLS]. There is the OWASP Testing Guide [OWASP-TLS], a rating guide [SSL-RATING], and tools to automate the tests, such as sslaudit [SSLAUDIT].

When it comes to client security, things are less advanced. Till recently sslsniff [SSLSNIFF] attacking tool was probably the most interesting effort in this direction. A recent Blackhat presentation [BH-SSL-TTRUST] focuses on security issues introduced by SSL-aware proxies and describes common implementation flaws in SSL clients. The authors of that presentation have published an on-line testing service [SSLTEST] suitable for testing web browsers.

3 WHAT WE TEST FOR

The goal of sslcaudit project is to develop a utility to automate testing SSL/TLS clients for resistance against MITM attacks, focusing on flaws exploitable in practice. On high level sslcaudit tests:

- what server certificates the client trusts enough to establish SSL/TLS connection,
- what flavors of SSL protocol the client supports (coming in sslcaudit v1.1).

In general, a correctly implemented SSL/TLS client exhibits the following testable behavior. Related to the server certificate validation:

C1	Rejects self-signed certificates, certificates not signed by a trusted CA	In practice a failure to implement C1, C2, or C3 is the most dangerous and allows for a straightforward MITM attack.
C2	Validates basic constraints of intermediate CAs	
C3	Only accepts server certificate with CN matching the intended destination	
C4	Does not accept expired certificates	To abuse an expired certificate an attacker being able

		to obtain a legitimate, but expired or revoked certificate for the server or an intermediate/root CA trusted by the client. Under normal circumstance this is not possible for MITM attacker.
C5	Does not accept revoked certificates	Majority of SSL/TLS clients do not support CRL/OSCP support by design.
C6	Do not be fooled by NUL-character in CN.	To exploit C5 a valid certificate with NUL-byte in CN is needed. According to author's knowledge, nowadays public CAs do not issue such certificates.

Testable behavior related to SSL/TLS protocol support:

P1	Do not support SSLv2 (version/cipher downgrade)	The failure to implement P1 leads to theoretical possibility of cipher downgrade attacks. To author's knowledge practical exploitation is very tricky, there is no free or commercial tool for it.
P2	Do not support SSL and TLS 1.0 (CBC attack)	An attack leading to cookie theft in web browsers was demonstrated [BS-BEAST]. It has a prerequisite of an attacker being able to inject a malicious JavaScript code into victim's browser. According to author's knowledge the attack is generally not applicable to applications using SSL/TLS to machine-to-machine communication.
P3	Do not support weak key exchange protocols, low key lengths, low ciphers strengths	If strong ciphers are supported by the peers, the presence of weak ones is only exploitable via cipher downgrade attack.

Testing for C1, C2, C3 (chain of trust, CN mismatch) are already implemented in sslcaudit v1.0. Protocol-level tests will come in v1.1. C4, C5, C6 appear to have lower practical interest and might be implemented in later versions.

More specifically, sslcaudit uses the following algorighm to generate test server certificates.

If user has supplied a certificate via –user-cert/--user-key options,

- sslcaudit tries to use the user-supplied certificate as is,

Next, sslcaudit generates certificate requests with the following properties:

- default hardcoded CN (www.example.com), unless disabled by --no-default-cn
- user-specified CN, if supplied via --user-cn
- matching attributes of a certificate fetched from user-specified SSL/TLS server, if set by --server HOST:PORT option

Each certificate request gets signed in the following ways:

- self-signed, unless --no-self-signed is specified
- signed by the user-supplied certificate, to disable use --no-user-cert-signed
- signed by the user-supplied CA (--user-ca-cert / --user-ca-key)

- signed by the user-supplied CA with an intermediate CA
  - without basicConstraints
  - with basicConstraints CA:FALSE
  - with basicConstraints CA:TRUE

This way, sslcaudit will treat clients with up to 19 specially crafted certificates.

4 WHAT WE DO NOT TEST FOR

- Testing for protocol version and cipher support will come in v1.1. The functionality will be similar to sslaudit [SSLAUDIT], but backwards.
- "SSL 3.0/TLS 1.0 renegotiation attack" [TLS-RENEG]
- [OSCP-ATTACK], reportedly implemented by ssnsliff. Will be implemented in the future versions.

5 HOW TO USE THE TOOL

5.1 Installation and dependencies

There is no procedure for installation yet. Just grab the code:

- Download ZIP archive at https://github.com/grwl/sslcaudit/zipball/release_1_0_rc1
- Or clone leading edge master GIT repository: git clone git://github.com/grwl/sslcaudit.git
- Find sslcaudit in the top level directory and run it with -h option.

NB: To terminate an instance of sslcaudit running on the console, press Ctrl-\.

Sslcaudit uses M2Crypto Python library. If you dependencies problem, you might see following:

```
$ ./sslcaudit
Traceback (most recent call last):
…
ImportError: No module named M2Crypto
```

On Debian-based systems M2Crypto library can be installed with the following command:

```
$ sudo apt-get install python-m2crypto
```

5.2 Network and client setup

To use sslcaudit, a penetration tester has to convince the client under test to establish a series of connections to the listener of sslcaudit. Relevant TCP connections are supposed to be redirected to the local listener created by sslcaudit. This can be done in number of ways, for example by changing hosts file on the client under test or using Marvin [MARVIN]. The matter of connection redirection is outside of the scope of this document.

Sslcaudit plays a role of a rogue SSL/TLS server, presenting the client with various certificates and logging the outcome of the tests.

For best test coverage sslcaudit should be provided with additional information:

1. If possible, a user-controlled CA should be added to the list of CAs trusted by the client under test. Certificate and a key of that CA should be passed to sslcaudit. This will allow for generation of the widest range of certificates and perform all relevant tests and validation of the test setup.

2. If it is not possible to add a custom CA into the list of CAs trusted by the client, try to get hold of any valid non-CA certificate (and its private key) issued by CA trusted by the client. If such a certificate is passed to sslcaudit via --user-cert/--user-key, it will be used to produce certificates used for basicConstraints validation exercise.

3. Sslcaudit should be provided with CN of the server the client communicates with. It can be given explicitly via --user-cn option, or by specifying the server address and port with --server option. In the latter case sslcaudit will try to fetch certificate information from the server.

Sslcaudit does not (yet) do any risk assessment. Instead it displays information about what certificate configurations have been tried and how the client has been behaving. It is up to the user to make conclusions, which are obvious in most cases anyway.

Below we will consider four examples showing how sslcaudit helps testing the behavior of SSL clients.

5.3 Test if client trusts an arbitrary self-signed certificate (example 1)

Open two terminal windows, run sslcaudit in one of them.

```
$ ./sslcaudit
```

When launched sslcaudit starts listening on all interfaces on port 8443.

In another terminal let's run openssl to connect to sslcaudit.

```
$ openssl s_client -connect localhost:8443
CONNECTED(00000003)
depth=0 /CN=www.example.com/C=BE/O=Gremwell bvba
verify error:num=18:self signed certificate
verify return:1
depth=0 /CN=www.example.com/C=BE/O=Gremwell bvba
verify return:1
```

In the first terminal you will see a the result of the test.

```
$ ./sslcaudit
127.0.0.1:38849  selfsigned(www.example.com)          connected, read timeout (in 3.0s)
```

The output says:

- a connection was received from 127.0.0.1:38849
- the connection was handled with a self-signed certificate with CN=www.example.com

- SSL connection was established successfully, but client has sent no data in 3 sec

The client establishes SSL session with a server presenting a self-signed certificate and does not close it immediately. It appears the client verifies nothing at all and therefore vulnerable to MITM attack.

5.4 Test if client trusts an arbitrary self-signed certificate (example 2)

Now do the same as above, but use socat instead of openssl. Socat validates server certificates by default and will not connect to an arbitrary peer. Now run sslaudit as in the previous example, then start socat:

```
$ socat - OPENSSL:localhost:8443
2012/05/01 10:50:50 socat[18692] E SSL_connect(): error:14090086:SSL
routines:SSL3_GET_SERVER_CERTIFICATE:certificate verify failed
```

On the side of sslcaudit we will see:

```
127.0.0.1:38889  selfsigned(www.example.com)                    tlsv1 alert unknown ca
```

Again:

- a connection was received from 127.0.0.1:38889
- the connection was handled with a self-signed certificate with CN=www.example.com
- SSL connection setup has failed

The client refuses connecting to the server presenting self-signed certificate for an arbitrary CN. Based on this we can only conclude that the client is not completely broken and refuses connecting to obviously insecure server.

Let's assume we cannot tamper with client's list of trusted CAs nor find any certificate issued by a CA already trusted by the client. One thing we should do under the circumstances is point sslcaudit towards the real server and let it mimic server's certificate. Connecting with socat again, but now run it in an infinite loop:

```
$ while true ; do socat - OPENSSL:localhost:8443; sleep .5; done
```

The following appears on the side of sslcaudit:

```
$ ./sslcaudit --server 62.213.200.252:443
...
127.0.0.1:41264  selfsigned(www.example.com)                    tlsv1 alert unknown ca
127.0.0.1:41265  selfsigned(brufeprd1.hackingmachines.com)       tlsv1 alert unknown ca
```

We can see that the client rejects self-signed certificate even if its subject matches the one fetched from the server. So far there are no evidences of an insecure behavior. Still no tests done confirming the client does proper basicConstraints validation.

5.5 Re-purpose a certificate (example 3)

If we don't have a chance to alter the list of CAs trusted by the client, there is only one additional

thing we can do: supply sslcaudit with some certificate issued by CA already trusted by the client. If such a certificate is passed to sslcaudit (via --user-cert/--user-key), it will be used to produce certificates for basicConstraints validation exercises.

To simulate client we use socat again, but pass extra parameter to make it trust CA which has issued the certificate.

```
$ while true ; do socat - OPENSSL:localhost:8443,cafile=test/certs/test-ca-cacert.pem ;
sleep .5; done
```

Running sslcaudit, passing it the certificate and the key:

```
$ ./sslcaudit --server 62.213.200.252:443 \
      --user-cert test/certs/www.example.com-cert.pem
      --user-key test/certs/www.example.com-key.pem
127.0.0.1:41764  user-supplied(www.example.com)        connected, read timeout (in 3.0s)
127.0.0.1:41765  selfsigned(www.example.com)                      tlsv1 alert unknown ca
127.0.0.1:41766  selfsigned(brufeprd1.hackingmachines.com)        tlsv1 alert unknown ca
127.0.0.1:41767  signed1(www.example.com, www.example.com)        tlsv1 alert unknown ca
127.0.0.1:41768  signed1(brufeprd1.hackingmachines.com, www.example.com)
                                                                  tlsv1 alert unknown ca
```

The result of the first test indicates the client has established connection and didn't close it right away. This suggests that the client validates CA, but does ignores CN mismatch. This weakness is exploitable if an attacker can get hold of a certificate (and private key) issued by any CA trusted by the client.

The last two lines correspond to attempts to produce a certificate by signing it with user-supplied certificate. The client under test has rejected those certificates, which suggests the client validates basicConstraints of the certificate in the chain of trust.

5.6 Tests with user-supplied CA (example 4)

As mentioned earlier, for most comprehensive testing it is necessary to add test CA to the client configuration. Here we assume it was done and CA certificate (test/certs/test-ca-cacert.pem) is already added to the list of CAs trusted by the client.

To simulate client side we will use socat again.

```
$ while true ; do socat - OPENSSL:localhost:8443,cafile=test/certs/test-ca-cacert.pem ;
sleep .5; done
```

```
$ ./sslcaudit --server 62.213.200.252:443 \
      --user-ca-cert test/certs/test-ca-cacert.pem \
      --user-ca-key test/certs/test-ca-cakey.pem
127.0.0.1:41907  selfsigned(www.example.com)                    tlsv1 alert unknown ca
127.0.0.1:41908  selfsigned(brufeprd1.hackingmachines.com)      tlsv1 alert unknown ca
127.0.0.1:41909  signed1(www.example.com, test-ca)     connected, read timeout (in 3.0s)
127.0.0.1:41911  signed1(brufeprd1.hackingmachines.com, test-ca)
                                                     connected, read timeout (in 3.0s)
127.0.0.1:41912  signed2(www.example.com, ca-none, test-ca)     tlsv1 alert unknown ca
127.0.0.1:41913  signed2(www.example.com, ca-false, test-ca)    tlsv1 alert unknown ca
127.0.0.1:41914  signed2(www.example.com, ca-true, test-ca)
                                                     connected, read timeout (in 3.0s)
127.0.0.1:41916  signed2(brufeprd1.hackingmachines.com, ca-none, test-ca)
                                                                  tlsv1 alert unknown ca
127.0.0.1:41917  signed2(brufeprd1.hackingmachines.com, ca-false, test-ca)
```

```
                                                                  tlsv1 alert unknown ca
127.0.0.1:41918  signed2(brufeprd1.hackingmachines.com, ca-true, test-ca)
                                                        connected, read timeout (in 3.0s)
```

From the output of sslcaudit above it is apparent that the client properly validates the trust of chain, but accepts certificate with any CN. This is consistent with what is expected from socat. Additionally this proves that sslcaudit produces well formatted “trustable” certificates.

5.7 Tests with user-supplied CA (example 5)

Finally we repeat the last test, but against a proper SSL client, curl, invoked as following:

```
$ while true ; do curl --cacert test/certs/test-ca-cacert.pem https://localhost:8443/ ;
sleep .5 ; done
```

Now we run sslcaudit. Here we assume we somehow know what CN the client expects and specify it directly via --user-cn parameter.

```
$ ./sslcaudit --user-cn localhost \
      --user-ca-cert test/certs/test-ca-cacert.pem \
      --user-ca-key test/certs/test-ca-cakey.pem
127.0.0.1:42028  selfsigned(www.example.com)                     tlsv1 alert unknown ca
127.0.0.1:42029  selfsigned(localhost)                           tlsv1 alert unknown ca
127.0.0.1:42030  signed1(www.example.com, test-ca)
                                          connected, EOF before timeout (in 0.001s)
127.0.0.1:42031  signed1(localhost, test-ca)          connected, got 155 octets in 0.0s
127.0.0.1:42032  signed2(www.example.com, ca-none, test-ca)      tlsv1 alert unknown ca
127.0.0.1:42033  signed2(www.example.com, ca-false, test-ca)     tlsv1 alert unknown ca
127.0.0.1:42034  signed2(www.example.com, ca-true, test-ca)
                                          connected, EOF before timeout (in 0.001s)
127.0.0.1:42035  signed2(localhost, ca-none, test-ca)            tlsv1 alert unknown ca
127.0.0.1:42036  signed2(localhost, ca-false, test-ca)           tlsv1 alert unknown ca
127.0.0.1:42037  signed2(localhost, ca-true, test-ca)  connected, got 155 octets in 0.0s
```

Here we can see that the client only establishes connection with servers having certificate signed by a trusted CA. Self-signed certificates and certificate signed by an intermediate CA with unsafe basicConstraints are rejected. Also, the client closes the connection right away if there is a CN mismatch.

This kind of output in general means that server certificate validation is implemented correctly. (The behavior of the client towards servers with an expired certificate or a certificate with NUL-characters in CN remains untested.)

6 SUPPORTED SYSTEMS

Sslcaudit is written in Python. It is tested on Python 2.7. Requires M2Crypto (http://chandlerproject.org/bin/view/Projects/MeTooCrypto) library which provides binding to OpenSSL.

It is developed and tested on Ubuntu Natty 11.04, with stock python-m2crypto-0.20.1-1ubuntu5 package installed. Partially tested on BackTrack 5 R2.

OpenSSL library shipped with recent Linux distributions does not support SSLv2. This does not affect this version of sslcaudit. The next version of sslcaudit will feature protocol level tests and will require OpenSSL

library supporting SSLv2.

7 COMMAND LINE PARAMETERS

```
$ ./sslcaudit -h
Usage: sslcaudit [OPTIONS]

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -l LISTEN_ON          Specify IP address and TCP PORT to listen on, in
                        format of [HOST:]PORT
  -m MODULES            Launch specific modules. For now the only functional
                        module is 'sslcert'. There is also 'dummy' module used
                        for internal testing or as a template code for new
                        modules. Default is sslcert
  -v VERBOSE            Increase verbosity level. Default is 0. Try 1.
  -d DEBUG_LEVEL        Set debug level. Default is 0, which disables
                        debugging output. Try 1 to enable it.
  -c NCLIENTS           Number of clients to handle before quitting. By
                        default sslcaudit will quit as soon as it gets one
                        client fully processed.
  -N TEST_NAME          Set the name of the test. If specified will appear in
                        the leftmost column in the output.
  --user-cn=USER_CN     Set user-specified CN.
  --server=SERVER       Where to fetch the server certificate from, in
                        HOST:PORT format.
  --user-cert=USER_CERT_FILE
                        Set path to file containing the user-supplied
                        certificate.
  --user-key=USER_KEY_FILE
                        Set path to file containing the user-supplied key.
  --user-ca-cert=USER_CA_CERT_FILE
                        Set path to file containing certificate for user-
                        supplied CA.
  --user-ca-key=USER_CA_KEY_FILE
                        Set path to file containing key for user-supplied CA.
  --no-default-cn       Do not use default CN
  --no-self-signed      Don't try self-signed certificates
  --no-user-cert-signed
                        Do not sign server certificates with user-supplied one
```

8 LICENSE AND AUTHORS

The tool is released under GPLv3 license.

Most of the sslcaudit code is written by Alexandre Bezroutchko, abb@gremwell.com. Code handling keyboard interrupts contributed by Raf Somers raf.somers@telenet.be.

9 SUPPORT

If you have a question, post it to the forum dedicated to sslcaudit support available at http://www.gremwell.com/forum/117 . You can also send an email to info@gremwell.com if your matter is confidential.

10 ABOUT GREMWELL

Gremwell (http://www.gremwell.com/) offers security consulting services in the area of penetration testing, ethical hacking, vulnerability assessments and security code and configuration reviews. We are located in the neighbourhood of Brussels, and service clients in Belgium and abroad. Gremwell's consultants have more than 10 years experience in IT security.

Gremwell develops MagicTree - a data management tool for penetration testers.

11 REFERENCES

SSL/TLS security - the server side
[WIKI-TLS] http://en.wikipedia.org/wiki/Transport_Layer_Security
[SCANIT-SSL] http://www.scanit.be/uploads/ssl%20security%20in%20be%20-%2003-2008.pdf
[OWASP-TLS] https://www.owasp.org/index.php/Testing_for_SSL-TLS_%28OWASP-CM-001%29
[SSL-RATING] https://www.ssllabs.com/projects/rating-guide/index.html
[SSLAUDIT] http://code.google.com/p/sslaudit/
[TLS-RENEG] http://www.g-sec.lu/practicaltls.pdf

SSL/TLS security - the client side
[SSLSNIFF] http://www.thoughtcrime.org/software/sslsniff/
[SSLSTRIP] http://www.thoughtcrime.org/software/sslstrip/
[BH-SSL-STRIP] http://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf
[BH-SSL-TTRUST] https://media.blackhat.com/bh-eu-12/Jarmoc/bh-eu-12-Jarmoc-SSL_TLS_Interception-Slides.pdf
[SSL-TTRUST] http://www.secureworks.com/research/threats/transitive-trust/
[SSLTEST] https://ssltest.offenseindepth.com/
[BS-BEAST] http://www.schneier.com/blog/archives/2011/09/man-in-the-midd_4.html
[OPERA-BEAST] http://my.opera.com/securitygroup/blog/2011/09/28/the-beast-ssl-tls-issue
[OSCP-ATTACK] http://www.thoughtcrime.org/papers/ocsp-attack.pdf

IE5 SSL Spoofing vulnerability
[IE-SSL-CHAIN] http://www.thoughtcrime.org/ie-ssl-chain.txt
[BID-2737] http://www.securityfocus.com/bid/2737
[MS01-027] http://technet.microsoft.com/en-us/security/bulletin/ms01-027

Multiple Vendor Invalid X.509 Certificate Chain Vulnerability
[BID-5410] http://www.securityfocus.com/bid/5410

Apple iOS Data Security Certificate Chain Validation Security Vulnerability
[TWSL2011-007] https://www.trustwave.com/spiderlabs/advisories/TWSL2011-007.txt
[CVE-2011-0228] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-0228

[MARVIN] http://www.gremwell.com/marvin-mitm-tapping-dot1x-links