Using Frankencerts for Automated Adversarial Testing of Certificate Validation in SSL/TLS Implementations

Chad Brubaker\textsuperscript{1}  Suman Jana\textsuperscript{1}  Baishakhi Ray\textsuperscript{2}
Sarfraz Khurshid\textsuperscript{1}  Vitaly Shmatikov\textsuperscript{1}
\textsuperscript{1}University of Texas at Austin
\textsuperscript{2}University of California at Davis
Internet security = SSL/TLS
SSL/TLS security objectives

• End-to-end security even if the network is insecure
  – Authentication  = certificate validation!!
  – Confidentiality
  – Integrity
Certificate validation in SSL/TLS implementations
How to check if implementations are correct?

```c
bool is_cert_valid (cert_t *cert)
{
    return true;
}
```

How do people test SSL/TLS implementations?
## Current state of the art

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Test certificate count</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSS</td>
<td>54</td>
</tr>
<tr>
<td>GnuTLS</td>
<td>51</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>44</td>
</tr>
<tr>
<td>PolarSSL</td>
<td>18</td>
</tr>
<tr>
<td>CyaSSL</td>
<td>9</td>
</tr>
<tr>
<td>MatrixSSL</td>
<td>9</td>
</tr>
</tbody>
</table>

Most of these are just well-formed certificates!
Testing certificate validation code

• Test input generation
  – Fuzzing - huge input space, a fuzzed string won't even parse as an X.509 cert
Interpreting test results

- **test certificate**
- **SSL/TLS implementation**
- **accept/reject**

How do you know that the result is correct?
We tackle both of these problems in this work.
How to generate test certificates?

X.509 standards...ugh!
How to generate test certificates?

**Requirements**
- Must generate “semantically bad” certificates
- Should be syntactically correct, otherwise won’t exercise most of the cert validation code
- Must scale to millions of certs

**Idea**
- X.509 certs contain structured data, can we exploit that?
X.509 certificate structure

• Multilayered structured data
• **Syntactic constraints** for each piece
  – Ex: Version must be an integer
• **Semantic constraints** for individual piece or across multiple pieces
  – Ex: Version must be 0, 1, or 2
  – Ex: if version!=2, extensions must be NULL

<table>
<thead>
<tr>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial Number</td>
</tr>
<tr>
<td>Signature Algorithm Identifier</td>
</tr>
<tr>
<td>Issuer Name</td>
</tr>
<tr>
<td>Validity Period</td>
</tr>
<tr>
<td>Subject Name</td>
</tr>
<tr>
<td>Public Key Information</td>
</tr>
<tr>
<td>Issuer Unique ID</td>
</tr>
<tr>
<td>Subject Unique ID</td>
</tr>
<tr>
<td>Extensions</td>
</tr>
</tbody>
</table>
How to generate test certificates?

Create X.509 certs using randomly picked syntactically valid pieces

Likely to violate some semantic constraints i.e. will generate “bad” test certs just as we wanted

Wait, but how can we generate a large set of such syntactically valid pieces without reading X.509 specs?
Scan the internet for certificates

Collect 243,246 X.509 server certificates
Extract syntactically valid pieces

keyUsage extension from cert3

keyUsage extension from cert2

version from cert 1

ExtendedkeyUsage extension extension from cert4
Generate 8 million frankencerts from random combinations of certificate pieces
Interpret frankencert test results

• Differential testing of SSL/TLS implementations

• Multiple implementations of SSL/TLS should implement the same certificate validation logic

• If a certificate is accepted by some and rejected by others, what does this mean?
Which one is rotten?

No false positives though some instances might be different interpretations of X.509
Test results summary

- Tested 14 different SSL/TLS implementations
- 208 discrepancies due to 15 root causes
- Multiple bugs
  - Accepting fake and unauthorized intermediate Certificate Authorities (CAs)
  - Attacker can impersonate any website!
Some test results

<table>
<thead>
<tr>
<th>Problem</th>
<th>OpenSSL</th>
<th>PolarSSL</th>
<th>GnuTLS</th>
<th>CyaSSL</th>
<th>MatrixSSL</th>
<th>NSS</th>
<th>OpenJDK, Bouncy Castle</th>
<th>Browsers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Untrusted version 1 intermediate CA certificate</td>
<td>No</td>
<td>reject</td>
<td></td>
<td></td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Untrusted version 2 intermediate CA certificate</td>
<td>No</td>
<td>reject</td>
<td></td>
<td></td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Version 1 certificate with valid basic constraints</td>
<td>No</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>Firefox: reject Opera, Chrome: accept</td>
</tr>
<tr>
<td>Intermediate CA not authorized to issue further intermediate CA certificates, but followed in the chain by an intermediate CA certificate... followed by a leaf CA certificate</td>
<td>No</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Intermediate CA not authorized to issue certificates for server's hostname</td>
<td>No</td>
<td>reject</td>
<td></td>
<td></td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Certificate not yet valid</td>
<td>Yes</td>
<td>reject</td>
<td></td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Certificate expired in its timezone</td>
<td>Yes</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Certificate not authorized for signing other certificates</td>
<td>No</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Server certificate not authorized for use in SSL/TLS handshake</td>
<td>Yes</td>
<td>accept</td>
<td></td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Server certificate not authorized for server authentication</td>
<td>Yes</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Certificate with unknown critical extension</td>
<td>No</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Certificate with malformed extension value</td>
<td>No</td>
<td>accept</td>
<td></td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Certificate with the same issuer and subject and a valid chain of trust</td>
<td>No</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Issuer name does not match AKI</td>
<td>No</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Issuer serial number does not match AKI</td>
<td>No</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
</tr>
</tbody>
</table>
Exhibits
Version 1 CA certificates

*If an SSL/TLS implementation encounters a version 1 (v1) CA certificate that cannot be validated out of band, it must reject it*

RFC 5280 Section 6.1.4(k)

v1 CA certs do not support the CA bit: anybody with a valid v1 certificate can pretend to be a CA
Exhibit 1: GnuTLS

/* Disable V1 CA flag to prevent version 1 certificates in a supplied chain. */
flags &= ~(GNUTLS_VERIFY_ALLOW_X509_V1_CA_CRT);
ret = _gnutls_verify_certificate2 (flags,..))
Exhibit 2: Google Chrome

The site's security certificate has expired!

You attempted to reach www.google.com, but the server presented an expired certificate. No information is available to indicate whether that certificate has been compromised since its expiration. This means Google Chrome cannot guarantee that you are communicating with www.google.com and not an attacker. Your computer's clock is currently set to Wednesday, May 7, 2014 8:33:18 PM. Does that look right? If not, you should correct the error and refresh this page.

You should not proceed, especially if you have never seen this warning before for this site.

Proceed anyway  Back to safety

Help me understand

OK to click through?
Exhibit 2: Google Chrome untrusted CA
Exhibit 2: underlying cause

- Chrome uses a modified version of NSS for SSL certificate validation
- If a certificate is issued by a untrusted CA and is expired, the validation code only returns the expired error
- Firefox uses a glue layer called Personal Security Manager (PSM) over NSS and thus is not affected
Check the paper for more such goodies!!
Conclusions

• Differential testing with frankencerts is an effective technique for finding flaws in SSL/TLS implementations.

• Start integrating frankencerts with the test harness of your SSL/TLS implementation. The code is available at: https://github.com/sumanj/frankencert
Backup Slides
Frankencert features

- Frankencerts are random, yet syntactically correct X.509 certificates with ...
  - Unusual extensions
  - Rare and malformed values for these extensions
  - Strange key usage constraints
  - Rare combination of extensions
  - ... and many other unusual features
Mutate a few pieces randomly
/* Certificate authority constraint only available in version 3 certs */

if ((ic->version > 1) && (ic->extensions.bc.ca<= 0)) {
    psTraceCrypto("no CA permissions\n");
    sc->authStatus = PS_CERT_AUTH_FAIL_BC;
    return PS_CERT_AUTH_FAIL_BC;
}