



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

**Chad Brubaker<sup>1</sup> Suman Jana<sup>1</sup> Baishakhi Ray<sup>2</sup>**

**Sarfraz Khurshid<sup>1</sup> Vitaly Shmatikov<sup>1</sup>**

<sup>1</sup>University of Texas at Austin

<sup>2</sup>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

MatrixSSL™

OpenSSL™  
Cryptography and SSL/TLS Toolkit



# How to check if implementations are correct?

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

How do people test SSL/TLS implementations?

# Current state of the art

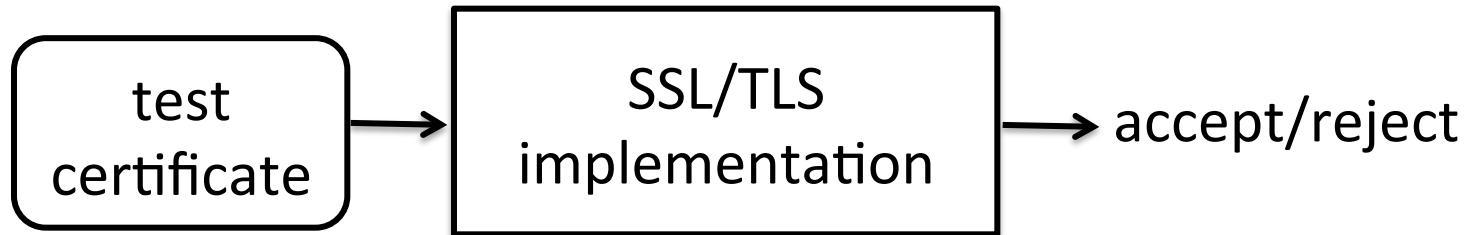
<b>Implementation</b>	<b>Test certificate count</b>
NSS	54
GnuTLS	51
OpenSSL	44
PolarSSL	18
CyaSSL	9
MatrixSSL	9

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

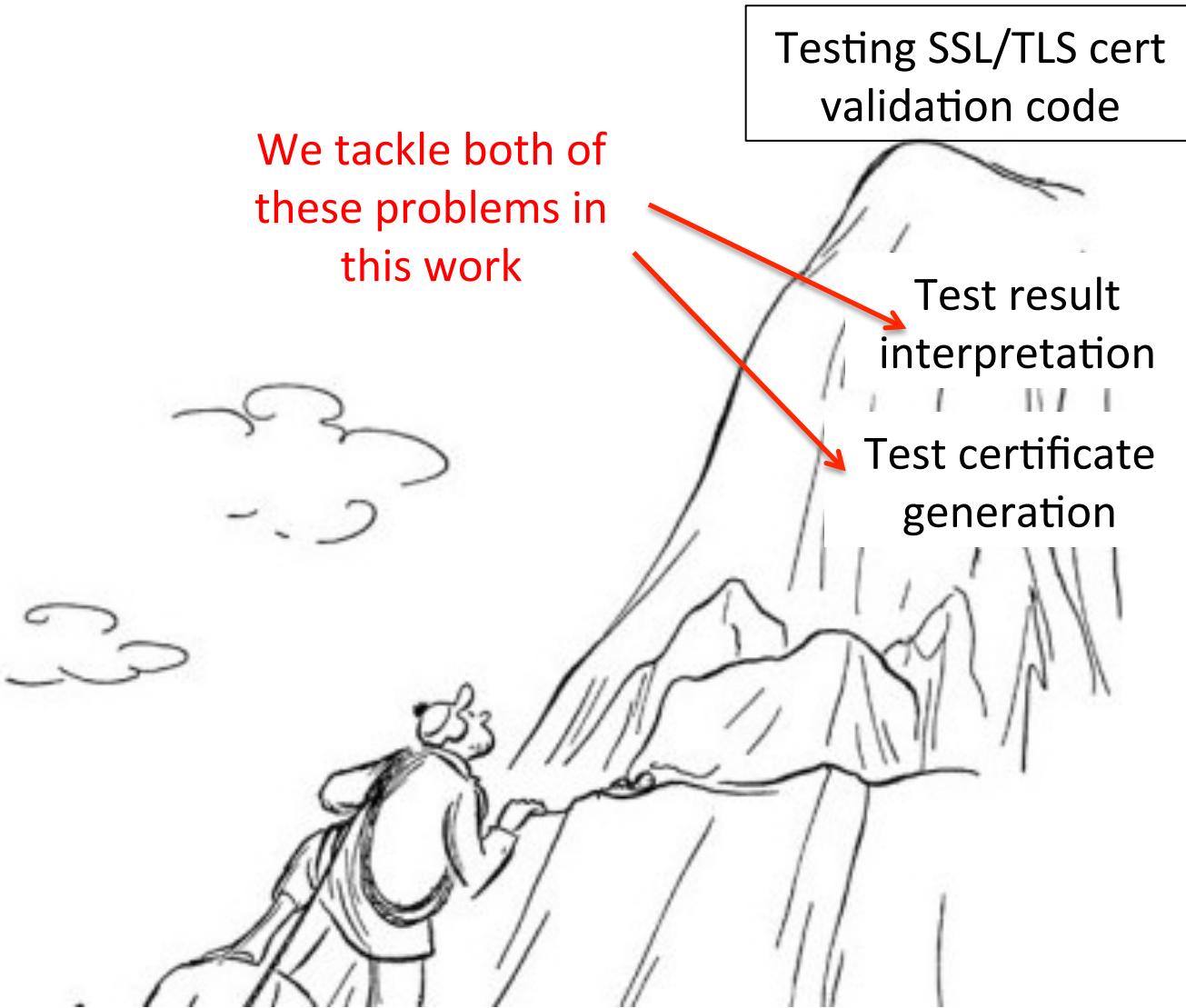
# Interpreting test results



Now What?!



How do you know that the result is correct?



A black and white cartoon illustration of a person with a backpack and a staff climbing a steep, rocky mountain peak. The person is looking up at the top of the mountain. In the background, there are some clouds.

Testing SSL/TLS cert validation code

We tackle both of these problems in this work

Test result interpretation

Test certificate generation

# 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

Version
Serial Number
Signature Algorithm Identifier
Issuer Name
Validity Period
Subject Name
Public Key Information
Issuer Unique ID
Subject Unique ID
Extensions

# 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

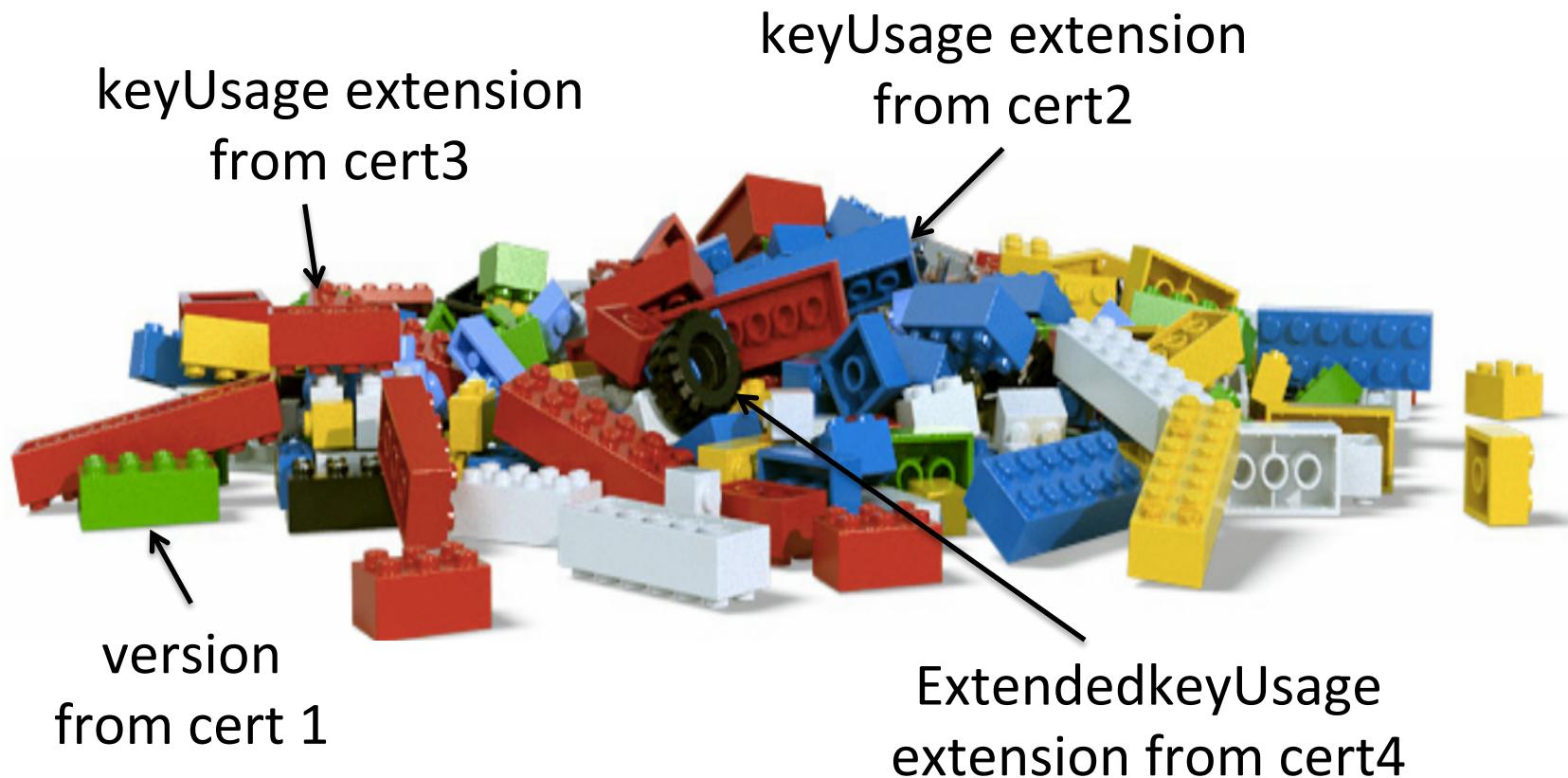
Wait, but how can we generate a large set  
of such syntactically valid pieces without  
reading X.509 specs?

# Scan the internet for certificates

Collect 243,246 X.509 server certificates



# Extract syntactically valid pieces



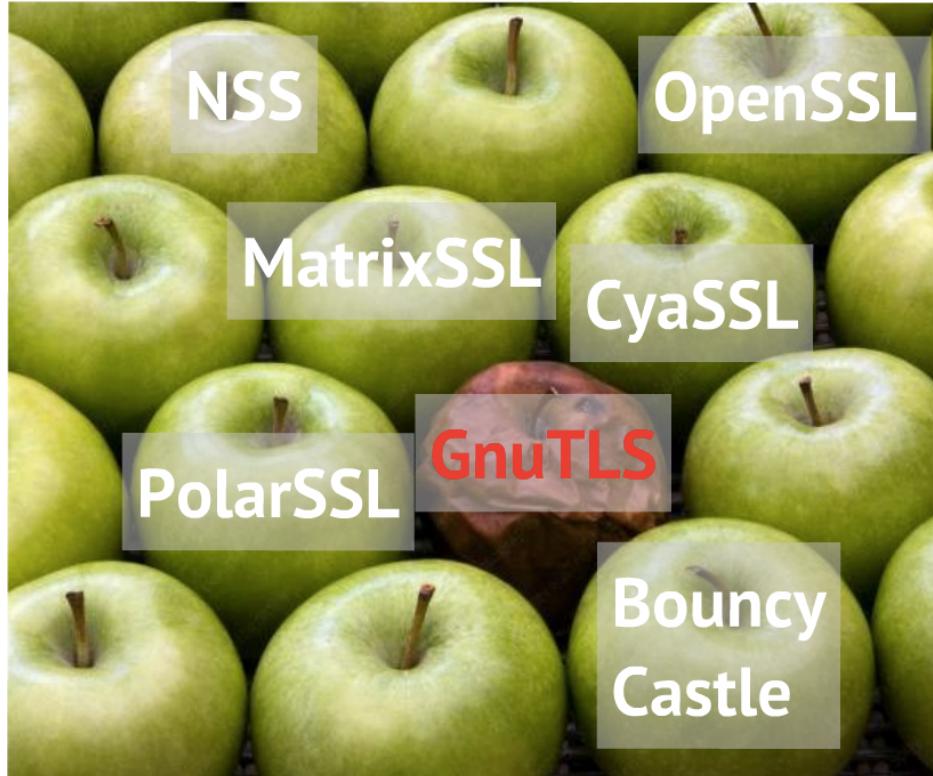
Generate 8 million frankencerts from  
random combinations of certificate pieces



# 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

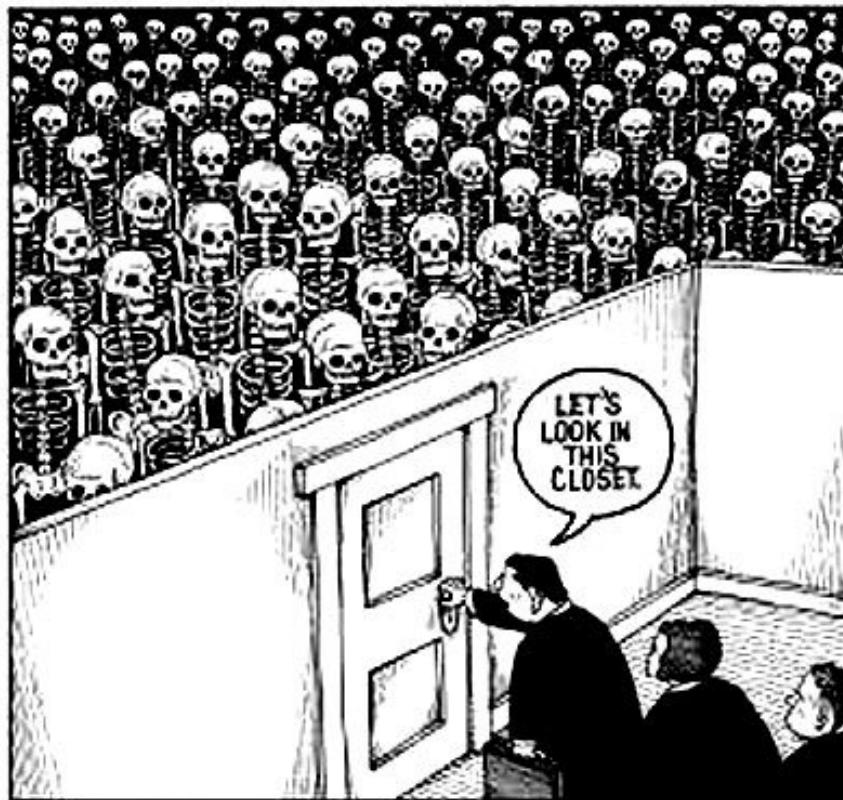
- Tested 14 different SSL/TLS implementations
- 208 discrepancies due to 15 root causes
- Multiple bugs
  - Accepting fake and unauthorized intermediate Certificate Authorities (CAs)

attacker can impersonate  
any website!

# Some test results

Problem	Certificates triggering the problem occur in the original corpus	OpenSSL	PolarSSL	GnuTLS	CyaSSL	MatrixSSL	NSS	OpenJDK, Bouncy Castle	Browsers
Untrusted version 1 intermediate CA certificate	No	reject	reject	accept	reject	accept	reject	reject	reject
Untrusted version 2 intermediate CA certificate	No	reject	reject	reject	reject	accept	reject	reject	reject
Version 1 certificate with valid basic constraints	No	accept	reject	accept	accept	accept	reject	reject	Firefox: reject Opera, Chrome: accept
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	No	reject	reject	reject	reject	accept	reject	reject	reject
Intermediate CA not authorized to issue certificates for server's hostname	No	reject	reject	accept	accept	accept	reject	reject	reject
Certificate not yet valid	Yes	reject	accept	reject	reject	reject	reject	reject	reject
Certificate expired in its timezone	Yes	reject	accept	reject	reject	accept	reject	reject	reject
CA certificate not authorized for signing other certificates	No	reject	reject	accept	accept	accept	reject	reject	reject
Server certificate not authorized for use in SSL/TLS handshake	Yes	reject	accept	accept	accept	accept	reject	reject	reject
Server certificate not authorized for server authentication	Yes	reject	accept	accept	accept	accept	reject	reject	reject
Certificate with unknown critical extension	No	reject	reject	accept	accept	accept	reject	reject	reject
Certificate with malformed extension value	No	accept	reject	accept	accept	accept	reject	reject	reject
Certificate with the same issuer and subject and a valid chain of trust	No	reject	reject	accept	reject	accept	reject	reject	reject
Issuer name does not match AKI	No	reject	accept	accept	accept	accept	reject	reject	reject
Issuer serial number does not match AKI	No	reject	accept	reject	accept	accept	reject	reject	reject

# 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 screenshot shows a Google Chrome browser window with a yellow warning box. The address bar at the top shows a red 'X' icon followed by `https://www.google.com`. The main content area has a yellow background with a white warning box. Inside the box, there is a yellow triangle icon with a black exclamation mark. The text reads: "The site's security certificate has expired!" Below this, a paragraph explains the issue: "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." A note below says: "You should not proceed, especially if you have never seen this warning before for this site." At the bottom of the box are two buttons: "Proceed anyway" and "Back to safety". Below the box is a link: "▶ [Help me understand](#)".

OK to click through?

# Exhibit 2: Google Chrome

The screenshot shows a browser window for <https://www.google.com>. A yellow warning bar on the left indicates that the site is untrusted because it was signed by an untrusted CA (Certificate Authority). The bar includes a large exclamation mark icon and text about proceeding anyway or getting help.

On the right, a "Certificate Viewer" window is open for the same URL. It displays the certificate details:

- General** tab selected.
- This certificate has been verified for the following usages:**
- Issued To:**
  - Common Name (CN): www.google.com
  - Organization (O): Google Inc.
  - Organizational Unit (OU): <Not Part Of Certificate>
  - Serial Number: 00:BC:BA:57:5A:51:B4:D5:31
- Issued By:**
  - Common Name (CN): **www.foobar.com** (circled in red)
  - Organization (O): Foobar Inc.
  - Organizational Unit (OU): <Not Part Of Certificate>
- Validity Period:**
  - Issued On: 2/5/12
  - Expires On: 2/5/14
- Fingerprints:**
  - SHA-256 Fingerprint: B9 4B 94 80 9F 99 B3 90 CD DC BA FF 4F E4 06 8B 0E AC 26 81 A9 A2 04 15 0C 18 22 71 7E EB AD

A red annotation highlights the "Issued By" section, specifically the Common Name (CN) "www.foobar.com".

## 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



## Exhibit 2: MatrixSSL

```
/* 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;  
}
```