Securing SSL Certificate Verification through Dynamic Linking

Adam Bates
University of Florida

Joe Pletcher
University of Oregon

Tyler Nichols
University of Oregon

Braden Hollembaek
iSEC Partners

Dave Tian
University of Oregon

Kevin R.B. Butler
University of Florida

CCS'14, Scottsdale, AZ, USA. 4 November, 2014.
Writing SSL code is hard.

KEEP CALM AND goto fail;

Developers misunderstand and even intentionally disable certificate validation in their SSL/TLS client software.
Running a CA? Also hard.

Certificate Authorities are constantly in the news for high-profile compromises and blunders. Can they be trusted?
The Problem

We now have decades of legacy code that is…

1. Vulnerable to Man in the Middle attacks
2. Married to a trust model that isn’t working.

This code isn’t getting patched anytime soon…
The Problem

How can we change these applications’ behavior without patching the applications?
CertShim overrides SSL behavior in client applications, providing certificate handling with:

- Safe Defaults
- CA Trust Enhancements
- Multi-Factor Verification

CertShim is also configurable, enabling special handling for different applications/domains.
Design Overview

SSL Libraries

libssl1.0.0

.

.

.

gnutls26

SSL Client Applications

netcat

curl

python

perl

wget

SSL Libraries

SSL Client Applications
Design Overview

CertShim

SSL Libraries

libssl1.0.0

gnutls26

SSL Hooks

Preloaded Shared Object

netcat

curl

python

perl

wget

SSL Client Applications
Design Overview

SSL Libraries

CertShim

SSL Hooks

Cert. Auth.  Convergence  Key Pinning  DANE

Verification Modules

Preloaded Shared Object

SSL Client Applications

netcat  curl  python  perl  wget

SSL Libraries

libssl1.0.0

gnutls26
Design Overview

SSL Libraries  Preloaded Shared Object  SSL Client Applications

libssl1.0.0  ...  gnutls26

SSL Hooks

Cert. Auth.  Convergence  Key Pinning  DANE

Policy Engine

CertShim

Verification Modules

netcat  curl  python  perl  wget
We assume…

1. SSL Libraries are correctly implemented.

We assume, and later demonstrate…

2. Software vendors write lazy SSL code.
   • Because it’s easier, they use (or misuse) the most popular SSL libraries and the standard APIs.
   • They also don’t take active countermeasures to evade our system.
Threat Model

• Our adversary is (somewhere) in the network, but our host is not under attacker control.

• Each verification module makes different assumptions about the adversary’s abilities:
  
  • **DANE** — Access to correct DNS resolvers and CAs.
  
  • **Convergence** — Access to correct notaries with diverse network paths to the target server.

  • **Client History Key Pinning** — Trust On First Use

  • **Traditional CA Verification** — No trusted CA has ever lost, leased, or given away its private key to the adversary.
SSL Hooks

Normal dynamic linking to OpenSSL

wget https://bitbucket.org

LD_LIBRARY_PATH

long SSL_get_verify_result(const SSL *)
SSL Hooks

Normal dynamic linking to OpenSSL

wget https://bitbucket.org

SSL_get_verify_result?

Dynamic Linker

LD_LIBRARY_PATH

OpenSSL

long SSL_get_verify_result(const SSL *)
SSL Hooks

Normal dynamic linking to OpenSSL

wget https://bitbucket.org

Dynamic Linker

SSL_get_verify_result?

LD_LIBRARY_PATH

long SSL_get_verify_result(const SSL *)

OpenSSL

long SSL_get_verify_result(const SSL *)
SSL Hooks

Dynamic Linking with CertShim

LD_PRELOAD
CertShim
long SSL_get_verify_result(const SSL *)

LD_LIBRARY_PATH
OpenSSL
long SSL_get_verify_result(const SSL *)

wget https://bitbucket.org
Dynamic Linker
SSL_get_verify_result?
SSL Hooks

Dynamic Linking with CertShim

LD_PRELOAD

wget https://bitbucket.org

SSL_get_verify_result?

CertShim

long SSL_get_verify_result(const SSL *)

LD_LIBRARY_PATH

OpenSSL

long SSL_get_verify_result(const SSL *)
Some verification modules required the domain name or port, which was not always contained in the SSL Context.

Our workaround was to instrument the `getaddrinfo()`, `gethostname()`, and `connect()` system calls.

This allowed us to create the mapping...
CertShim supports 4 methods of certificate verification:

1. Traditional CA Verification [Hickman et al. 1995]
3. DANE (RFC 6698) [Hoffman et al. 2012]
4. Client History Key Pinning [Soghoian et al. 2011]

More verification modules are in the works!
CertShim supports 4 methods of certificate verification:

1. Traditional CA Verification [Hickman et al. 1995]
3. DANE (RFC 6698) [Hoffman et al. 2012]
4. Client History Key Pinning [Soghoian et al. 2011]

More verification modules are in the works!

**WARNING:** When using HTTPS URLs, Urllib does not attempt to validate the server certificate. Use at your own risk!
CertShim supports 4 methods of certificate verification:

1. Traditional CA Verification [Hickman et al. 1995]
3. DANE (RFC 6698) [Hoffman et al. 2012]
4. Client History Key Pinning [Soghoian et al. 2011]

More verification modules are in the works!
Policy Engine

Enables ensemble votes for verification modules:

- Global policy sets CertShim’s default behavior.

- Command policies set application-specific behavior.

- Host policies set domain-specific behavior.

Argument-based policies, e.g., `wget --no-check-certificate`, are also in the works.

```c
// Global policy
global_policy: {
    cert_pinning = false;
    cert_authority = true;
    convergence = false;
    dane = false;
    vote = 1.00;
};

// Command policy
command_policy: {
    cmd= "/usr/bin/git";
    vote=1.00;
    methods:{
        cert_authority=false;
        convergence=true;
    };
};

// Host policy
host_policy: {
    host="www.torproject.org"
    vote=1.00;
    methods:{
        cert_authority=false;
        dane=true;
    };
};
```
In designing CertShim, we set out with the following goals:

1. *Override* insecure SSL usage

2. *Enable* SSL Trust Enhancements

3. *Maximize* compatibility
Override Insecure SSL Use

- Hook 9 total entry points to 3 SSL Libraries.
- Default policy enforces standard CA verification.
- Ensure safe certificate handling in applications that misuse SSL APIs.
- Forces verification on broken by design apps.

<table>
<thead>
<tr>
<th>Library</th>
<th>Hook</th>
</tr>
</thead>
<tbody>
<tr>
<td>connect</td>
<td>libssl1.0.0</td>
</tr>
<tr>
<td>do_handshake</td>
<td>libssl1.0.0</td>
</tr>
<tr>
<td>get_verify_result</td>
<td>libssl1.0.0</td>
</tr>
<tr>
<td>certificate_verify_peer_2</td>
<td>gnutls26</td>
</tr>
<tr>
<td>certificate_verify_peer_3</td>
<td>gnutls26</td>
</tr>
<tr>
<td>handshake</td>
<td>gnutls26</td>
</tr>
<tr>
<td>CheckIdentity</td>
<td>JDK6</td>
</tr>
<tr>
<td>CheckIdentity</td>
<td>JDK7</td>
</tr>
<tr>
<td>SetEndpointIdentifAlg</td>
<td>JDK7</td>
</tr>
</tbody>
</table>

Supported SSL API hooks
Override Insecure SSL Use

<table>
<thead>
<tr>
<th>Library</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>urllib, urllib2</td>
<td>Python</td>
</tr>
<tr>
<td>httplib</td>
<td>Python</td>
</tr>
<tr>
<td>pyCurl</td>
<td>Python</td>
</tr>
<tr>
<td>pyOpenSSL</td>
<td>Python</td>
</tr>
<tr>
<td>python ssl</td>
<td>Python</td>
</tr>
<tr>
<td>fsockopen</td>
<td>PHP</td>
</tr>
<tr>
<td>socket::ssl</td>
<td>Perl</td>
</tr>
</tbody>
</table>

Supported SSL Library Wrappers

Forces verification on SSL Library Wrappers that previously offered no support for certificate verification.
CertShim enables system-wide use of CA alternatives, but…

**Practical Obstacle for CA Trust Enhancements:** Due to inherent limitations or incremental deployment, none of these systems can validate *all* domain certificates:

Examples of Failures:

- Convergence: Domains on private networks
- DANE: Domains without a TLSA DNS record.
- Key Pinning: “Is this cert change malicious or benign?”
Enable Trust Enhancements

Our policy engine mitigates these shortcomings!

*Like DANE, but want a back-up for domains w/o TLSA?*

\[
\text{dane}=\text{true}; \text{convergence}=\text{true}; \text{vote}=0.5;
\]

*Like Convergence, but worried about an adversary taking control of all paths to a target server?*

\[
\text{convergence}=\text{true}; \text{cert_pinning}=\text{true}; \text{vote}=1.0;
\]

The case for ensemble verification will continue to grow more compelling as we develop more verification modules.
Maximize Compatibility

CertShim’s policy engine minimizes application breakage.

- You can change CertShim’s behavior to meet the needs of your application or domain.

- If your trust that an application is secure, e.g., it has a hardcoded cert, you can disable CertShim altogether.

We are investigating tools to aid in policy management, and even the release of policy modules for popular applications.
Evaluation: Coverage

1. Manually confirmed to work on 12 of 13 inspected SSL libraries and wrappers.

2. Confirmed to fix 8 of the 9 data transport vulnerabilities in [Georgiev et al. 2012].

3. Static analysis and inspection revealed that CertShim covers 94% of SSL usage in Ubuntu’s Top 10,000 pkgs.

<table>
<thead>
<tr>
<th>Library/Wrapper</th>
<th>Covered?</th>
</tr>
</thead>
<tbody>
<tr>
<td>libcurl</td>
<td>Yes</td>
</tr>
<tr>
<td>gnuTLS26</td>
<td>Yes</td>
</tr>
<tr>
<td>libssl1.0.0</td>
<td>Yes</td>
</tr>
<tr>
<td>SSLSocketFactory</td>
<td>Yes</td>
</tr>
<tr>
<td>perl socket::ssl</td>
<td>Yes</td>
</tr>
<tr>
<td>php_curl</td>
<td>NO!</td>
</tr>
<tr>
<td>fsockopen</td>
<td>Yes</td>
</tr>
<tr>
<td>pycurl</td>
<td>Yes</td>
</tr>
<tr>
<td>pyOpenSSL</td>
<td>Yes</td>
</tr>
<tr>
<td>python ssl</td>
<td>Yes</td>
</tr>
<tr>
<td>urllib, urllib2</td>
<td>Yes</td>
</tr>
<tr>
<td>gnutls-cli</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Evaluation: Coverage

1. Manually confirmed to work on 12 of 13 inspected SSL libraries and wrappers.

2. Confirmed to fix 8 of the 9 data transport vulnerabilities in [Georgiev et al. 2012].

3. Static analysis and inspection revealed that CertShim covers 94% of SSL usage in Ubuntu’s Top 10,000 pkgs.

<table>
<thead>
<tr>
<th>“Most Dangerous”</th>
<th>Covered?</th>
</tr>
</thead>
<tbody>
<tr>
<td>cUrl</td>
<td>Yes</td>
</tr>
<tr>
<td>httplib</td>
<td>Yes</td>
</tr>
<tr>
<td>perl fsockopen</td>
<td>Yes</td>
</tr>
<tr>
<td>php_curl</td>
<td>NO!</td>
</tr>
<tr>
<td>python ssl</td>
<td>Yes</td>
</tr>
<tr>
<td>urllib</td>
<td>Yes</td>
</tr>
<tr>
<td>urllib2</td>
<td>Yes</td>
</tr>
<tr>
<td>Apache HttpClient</td>
<td>Yes</td>
</tr>
<tr>
<td>Weberknect</td>
<td>Yes</td>
</tr>
</tbody>
</table>
1. Manually confirmed to work on 12 of 13 inspected SSL libraries and wrappers.

2. Confirmed to fix 8 of the 9 data transport vulnerabilities in [Georgiev et al. 2012].

3. Static analysis and inspection revealed that CertShim covers 94% of SSL usage in Ubuntu’s Top 10,000 pkgs.
Evaluation: Performance

<table>
<thead>
<tr>
<th>Baseline</th>
<th>CertShim Overheads</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenSSL</td>
<td>Traditional CA Verification</td>
</tr>
<tr>
<td></td>
<td>Key Pinning (Cached)</td>
</tr>
<tr>
<td>88ms</td>
<td>Key Pinning (Cache Miss)</td>
</tr>
<tr>
<td></td>
<td>Convergence (Local Cache)</td>
</tr>
<tr>
<td></td>
<td>Convergence (Notary Cache)</td>
</tr>
<tr>
<td></td>
<td>Convergence (Cache Miss)</td>
</tr>
<tr>
<td></td>
<td>DANE</td>
</tr>
<tr>
<td></td>
<td>+ 20ms</td>
</tr>
<tr>
<td></td>
<td>+ 31ms</td>
</tr>
<tr>
<td></td>
<td>+ 42ms</td>
</tr>
<tr>
<td></td>
<td>+ 31ms</td>
</tr>
<tr>
<td></td>
<td>+ 655ms</td>
</tr>
<tr>
<td></td>
<td>+ 918ms</td>
</tr>
<tr>
<td></td>
<td>Misleadingly Bad (7 sec)</td>
</tr>
</tbody>
</table>
Discussion

• **Continued Development**: future support for new SSL libraries, CA alternatives, and policy types.

• **Support for Java**: In the paper, we show that JSSE can be hooked with Java Instrumentation objects.

• **Support for Windows, OS X**: Other OS’ have comparable mechanisms to LD_PRELOAD.

• **Static Linking**: Vulnerable applications *cannot* be secured with our current approach (See paper).
Conclusion

- CertShim provides *system-wide protection* from SSL client vulnerabilities.

- CertShim’s Policy Engine enables *ensemble verification*, a new way of thinking about certificate handling.

- CertShim promotes *independent analysis* of SSL Trust Enhancements, and simplifies the process of prototyping of new enhancements.
Questions?

CertShim is available at [http://eng.ufl.edu/sensei](http://eng.ufl.edu/sensei)
Under Research -> Internet Security

Thank you for your time.

adammbates@ufl.edu