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.

sslKeyExchange.c

opensource.apple.com/source/Security/Security-55471/libsecurity_ssl/lib/sslKeyExchange.c

```
    hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;

if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

    err = sslRawVerify(ctx,
                       ctx->peerPubKey,
                       dataToSign,
                       dataToSignLen,
                       signature,
                       signatureLen);
    if(err) {
            sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRa
```

(Never Called)

Running a CA? Also hard.

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

TURKTRUST

COMODO
Creating Trust Online™

DigiNotar
Internet Trust Services

thawte
it's a trust thing

StartSSL

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

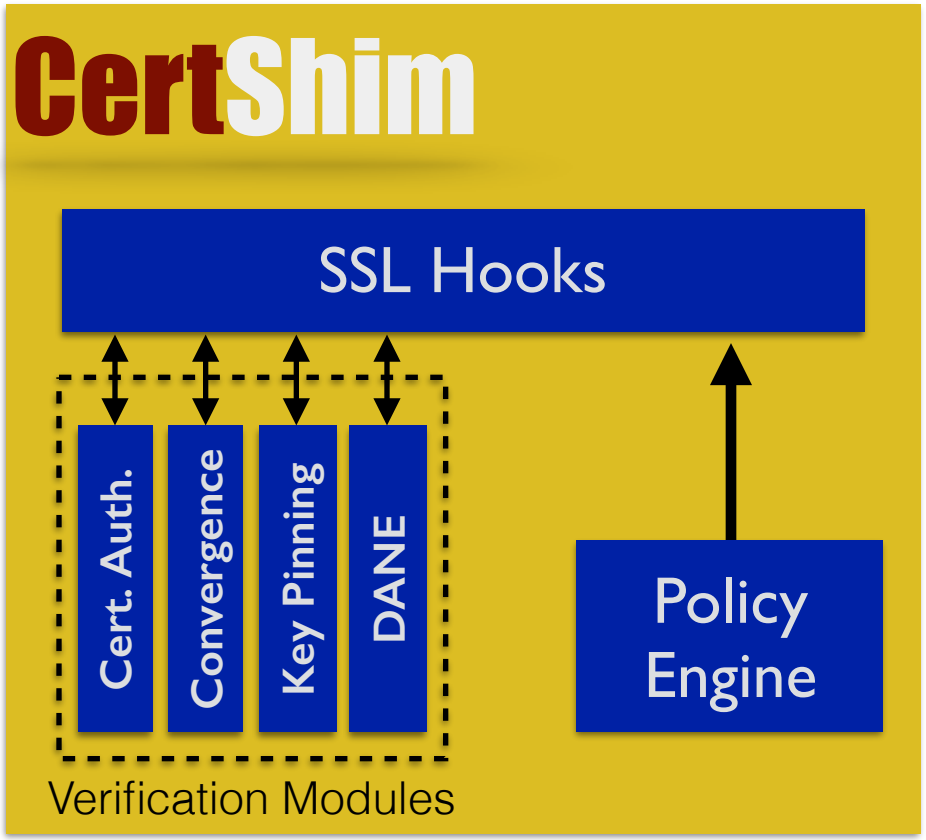
How can we change these applications' behavior without patching the applications?

Our Solution: CertShim

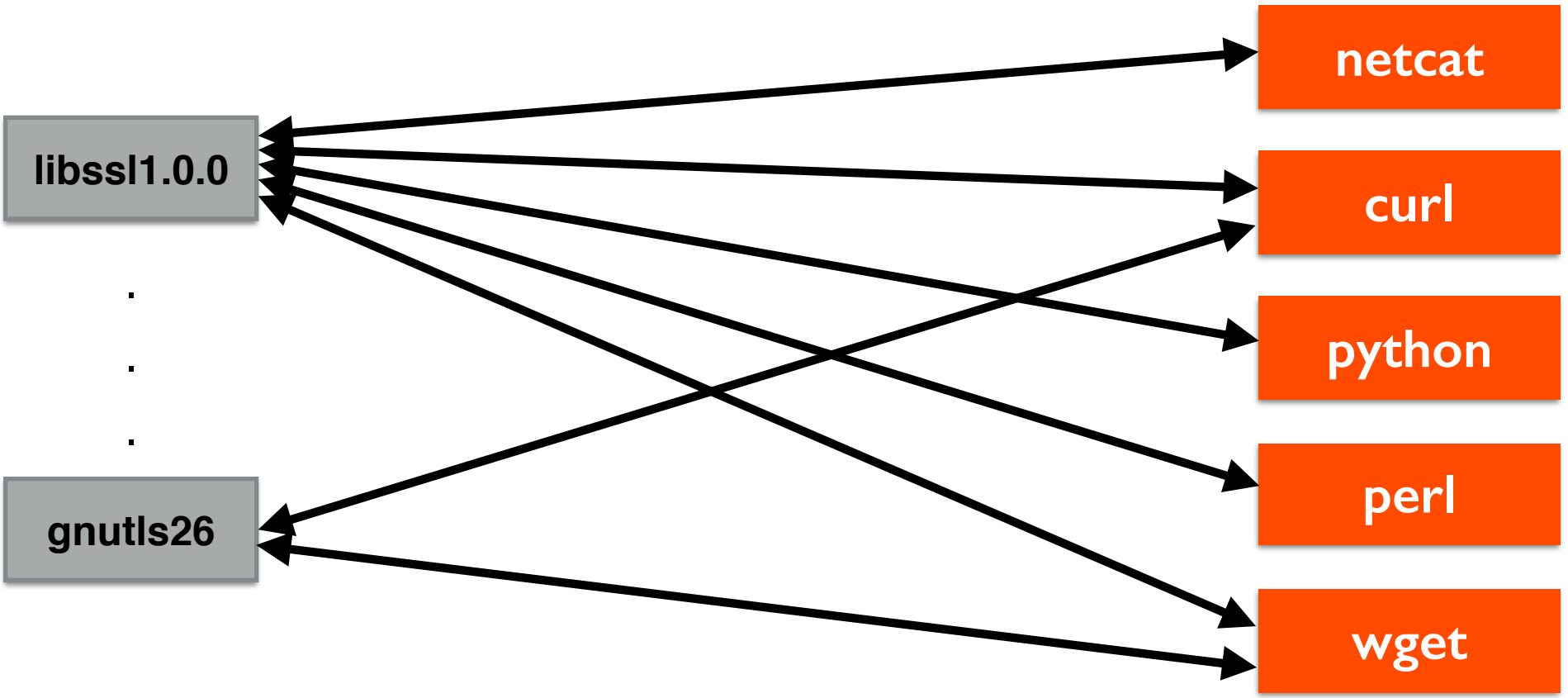
CertShim overrides SSL behavior in client applications, providing certificate handling with:

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

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



Design Overview



SSL Libraries

SSL Client Applications

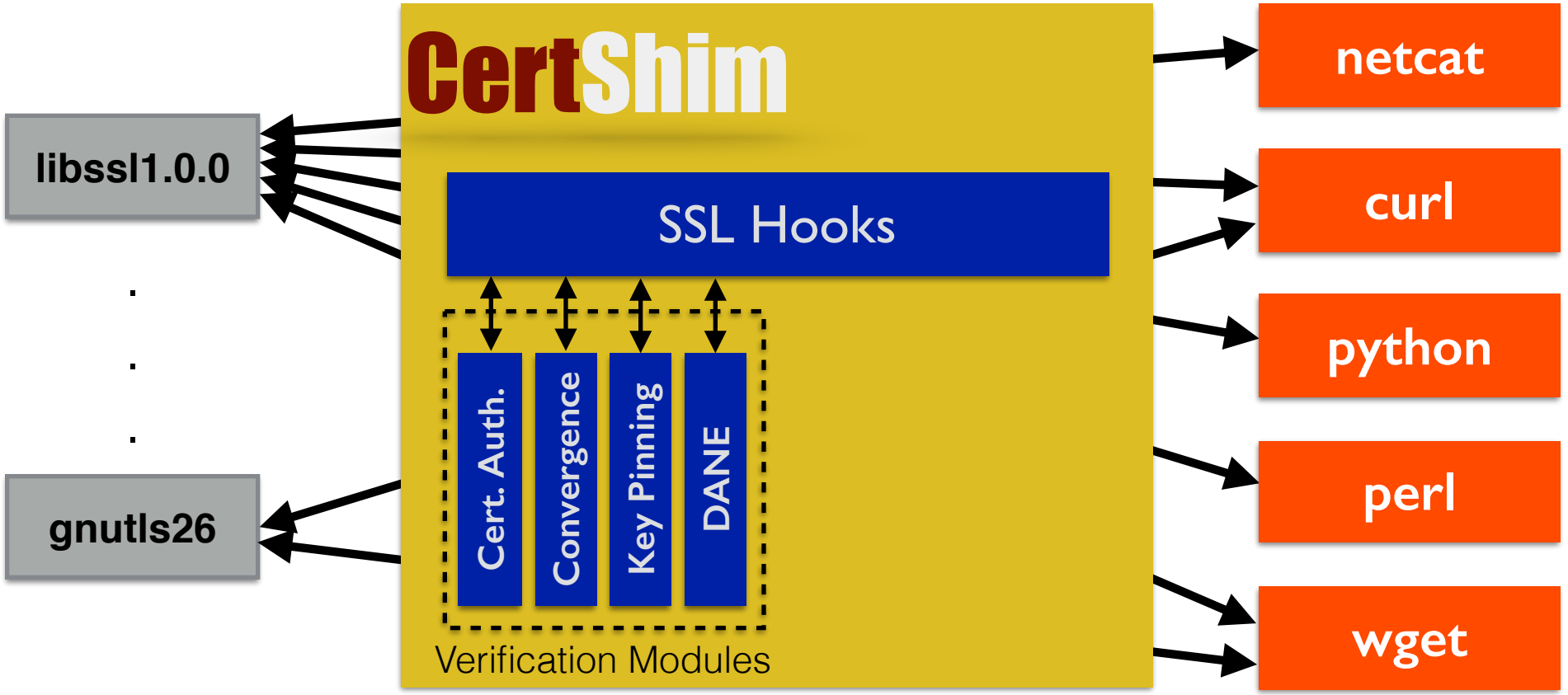
Design Overview

SSL Libraries

Preloaded Shared Object

SSL Client Applications

Design Overview

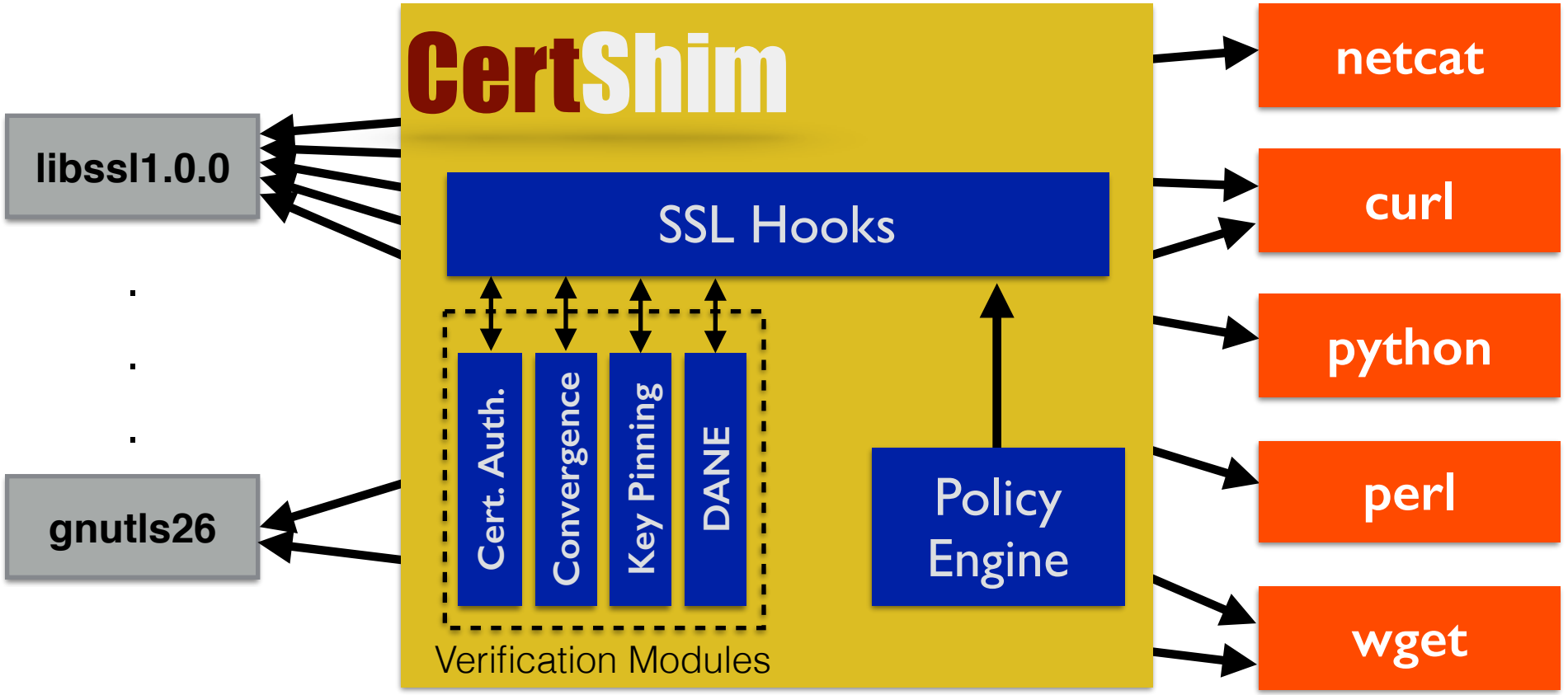


SSL Libraries

Preloaded Shared Object

SSL Client Applications

Design Overview



SSL Libraries

Preloaded Shared Object

SSL Client Applications

Assumptions

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

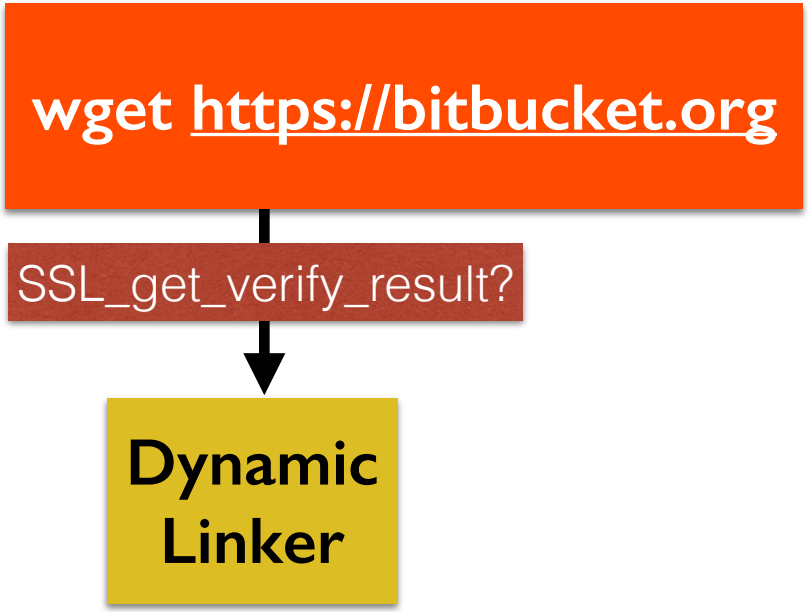
Normal dynamic linking to OpenSSL

wget https://bitbucket.org

LD_LIBRARY_PATH

SSL Hooks

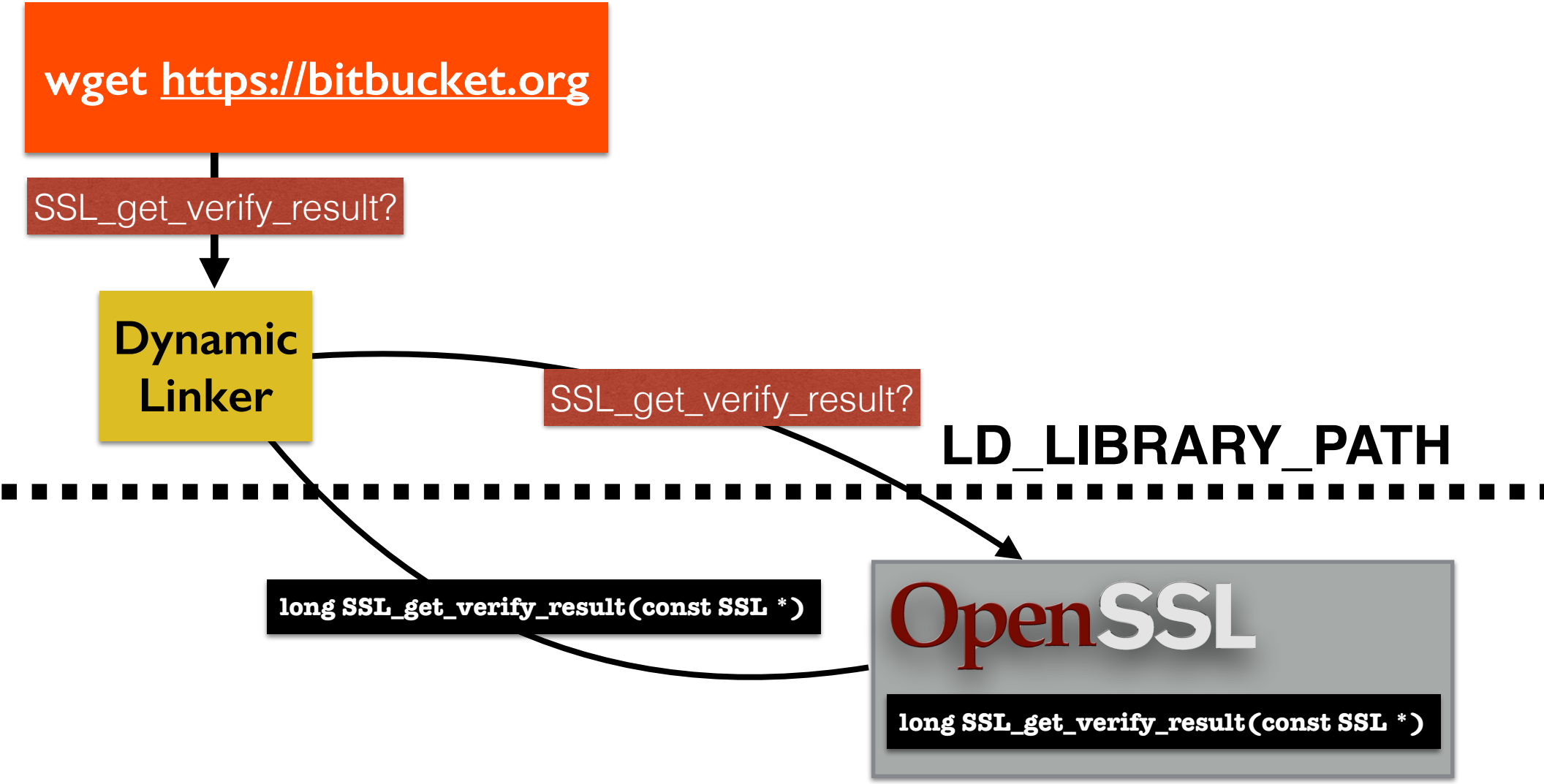
Normal dynamic linking to OpenSSL



LD_LIBRARY_PATH

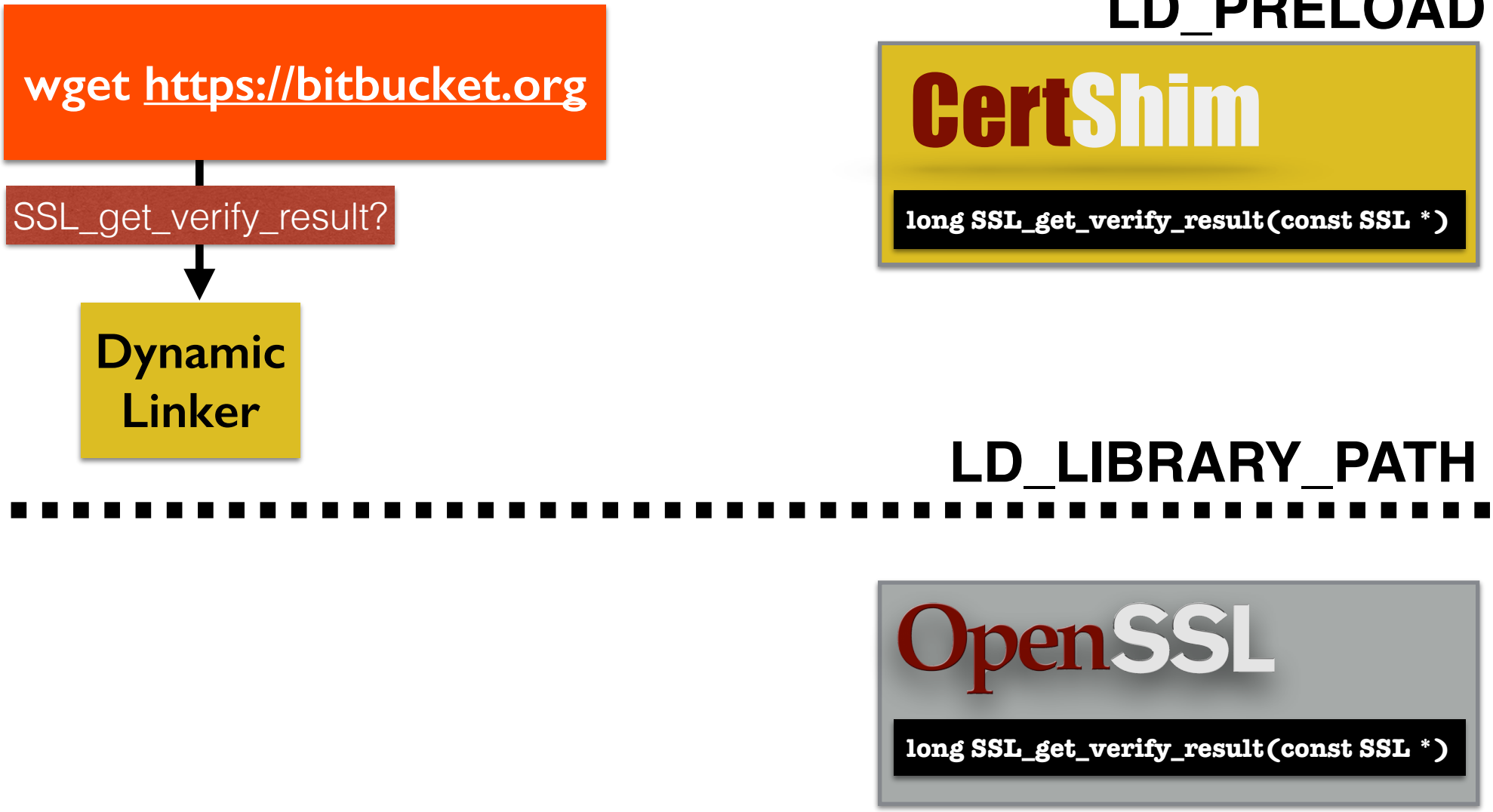
SSL Hooks

Normal dynamic linking to OpenSSL



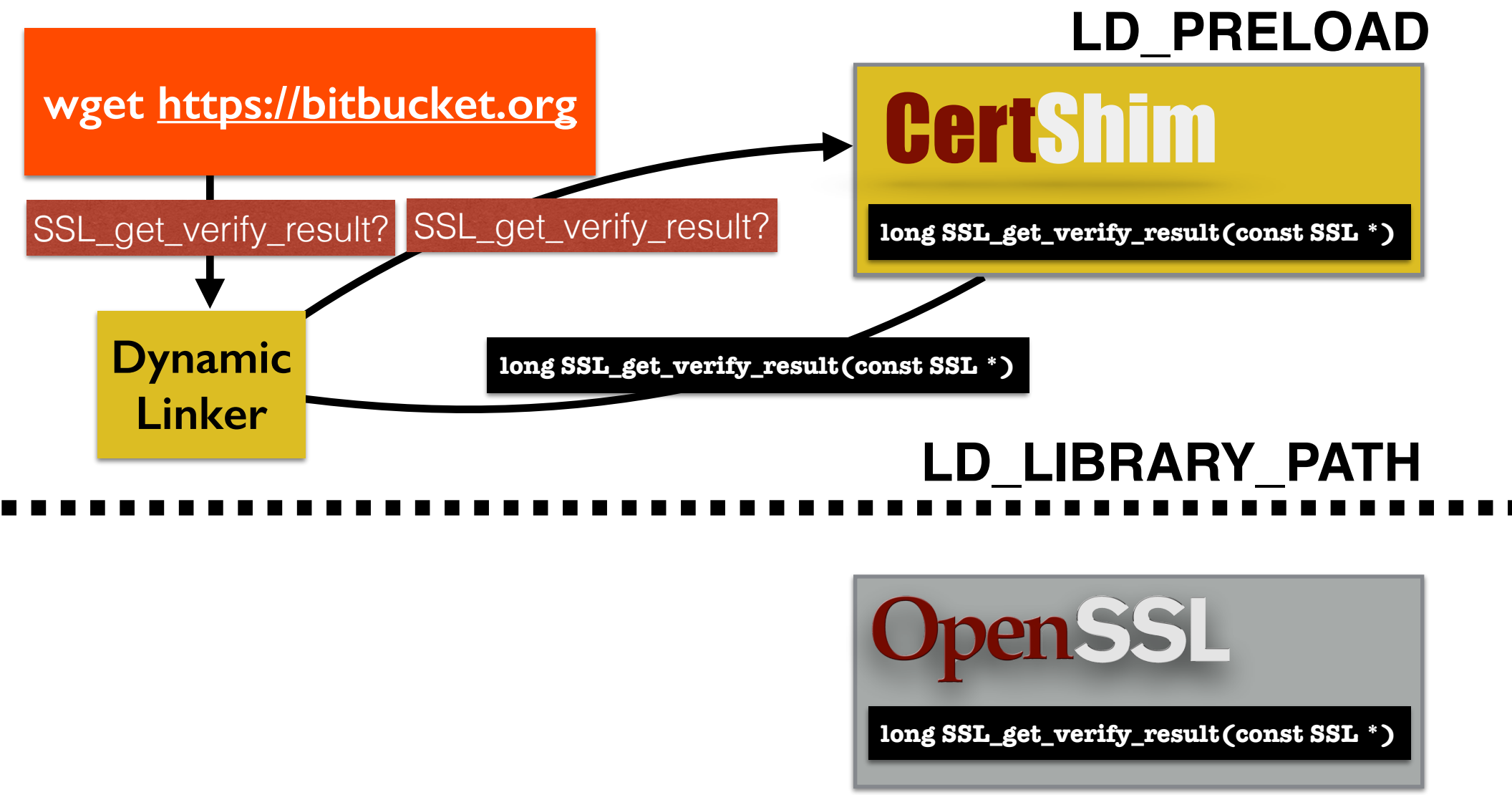
SSL Hooks

Dynamic Linking with CertShim



SSL Hooks

Dynamic Linking with CertShim



Other Network Hooks

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…

Verification Modules

CertShim supports 4 methods of certificate verification:

1. Traditional CA Verification [Hickman et al. 1995]
2. Convergence [Wendlandt et al. 2008, Marlinspike 2011]
3. DANE (RFC 6698) [Hoffman et al. 2012]
4. Client History Key Pinning [Soghoian et al. 2011]

More verification modules are in the works!

Verification Modules

20.5. urllib — Open arbitrary resources by URL

Note: The urllib module has been split into parts and renamed in Python 3 to urllib.request, urllib.parse, and urllib.error. The *2to3* tool will automatically adapt imports when converting your sources to Python 3. Also note that the urllib.request.urlopen() function in Python 3 is equivalent to urllib2.urlopen() and that urllib.urlopen() has been removed.

This module provides a high-level interface for fetching data across the World Wide Web. In particular, the urlopen() function is similar to the built-in function open(), but accepts Universal Resource Locators (URLs) instead of filenames. Some restrictions apply — it can only open URLs for reading, and no seek operations are available.

Warning: When opening HTTPS URLs, it does not attempt to validate the server certificate. Use at your own risk!

20.5.1. High-level interface ¶

urllib.**urlopen**(*url*[, *data*[, *proxies*]])

Open a network object denoted by a URL for reading. If the URL does not have a scheme identifier, or if it has file: as its scheme identifier, this opens a local file (without *universal newlines*); otherwise it opens a socket to a server somewhere on the network. If the connection cannot be made the IOError exception is raised. If all went well, a file-like object is returned. This supports the following methods: read(), readline(), readlines(), fileno(), close(), info(), getcode() and geturl(). It also has proper support for the *iterator* protocol. One caveat: the read() method, if the size argument is omitted or negative, may not read until the end of the data stream; there is no good way to determine that the entire stream from a socket has been read in the general case.

Except for the info(), getcode() and geturl() methods, these methods have the same interface as for file objects — see section *File Objects* in this manual. (It is not a built-in file object, however, so it can't be used at those few places where a true built-in file object is required.)

The info() method returns an instance of the class mimetools.Message containing meta-information associated with the URL. When the method is HTTP, these headers are those returned by the server at the head of the retrieved HTML page (including Content-Length and Content-Type). When the method is FTP, a Content-Length header will be present if (as is now usual) the server passed back a file length in response to the FTP retrieval request. A Content-Type header will be present if the MIME type can be guessed. When the method is

WARNING: When using HTTPS URLs, Urllib does not attempt to validate the server certificate. Use at your own risk!

Verification Modules

CertShim supports 4 methods of certificate verification:

1. Traditional CA Verification [Hickman et al. 1995]
2. Convergence [Wendlandt et al. 2008, Marlinspike 2011]
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.

```
global_policy: {
   cert_pinning = false;
   cert_authority = true;
   convergence = false;
   dane = false;
   vote = 1.00;};
```

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

```
host_policy: {
   host="www.torproject.org"
   vote=1.00;
   methods:{
      cert_authority=false;
      dane=true;};}
```

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

Analysis

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.

Library	Hook
connect	libssl1.0.0
do_handshake	libssl1.0.0
get_verify_result	libssl1.0.0
certificate_verify_peer_2	gnutls26
certificate_verify_peer_3	gnutls26
handshake	gnutls26
CheckIdentity	JDK6
CheckIdentity	JDK7
SetEndpointIdentifAlg	JDK7

Supported SSL API hooks

Override Insecure SSL Use

Library	Language
urllib, urllib2	Python
httplib	Python
pyCurl	Python
pyOpenSSL	Python
python ssl	Python
fsockopen	PHP
socket::ssl	Perl

Supported SSL Library Wrappers

Forces verification on SSL Library Wrappers that previously offered no support for certificate verification.

Enable Trust Enhancements

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?

```
dane=true; convergence=true; vote=0.5;
```

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

```
convergence=true; cert_pinning=true; 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 algother.

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.

Library/Wrapper	Covered?
libcurl	Yes
gnutls26	Yes
libssl1.0.0	Yes
SSLSocketFactory	Yes
perl socket::ssl	Yes
php_curl	NO!
fsockopen	Yes
pycurl	Yes
pyOpenSSL	Yes
python ssl	Yes
urllib, urllib2	Yes
gnutls-cli	Yes

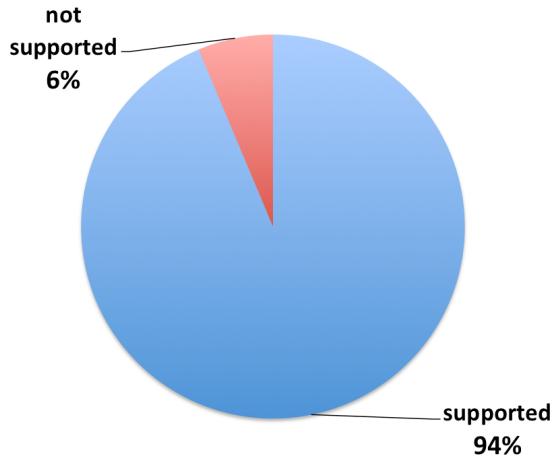
Evaluation: Coverage

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.

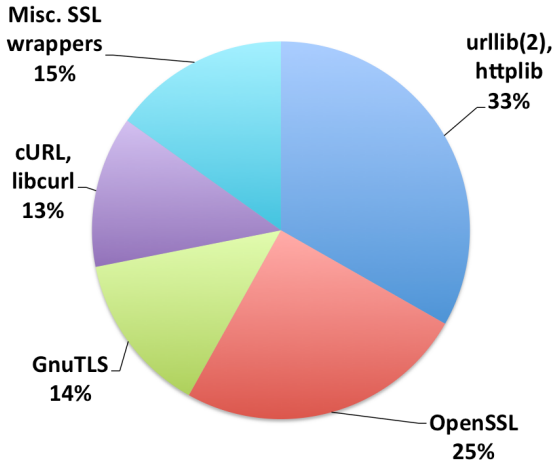
"Most Dangerous"	Covered?
cUrl	Yes
httplib	Yes
perl fsockopen	Yes
php_curl	NO!
python ssl	Yes
urllib	Yes
urllib2	Yes
Apache HttpClient	Yes
Weberknect	Yes

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.



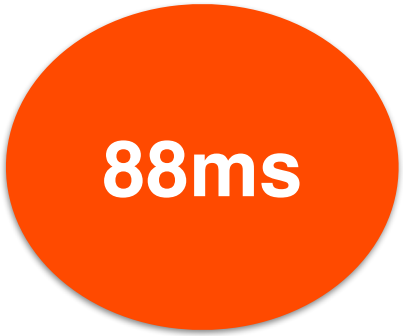
Total packages supported.



Breakdown of support by SSL Library/Wrapper.

Evaluation: Performance

Baseline



OpenSSL

CertShim Overheads

Traditional CA Verification	+ 20ms
Key Pinning (Cached)	+ 31ms
Key Pinning (Cache Miss)	+ 42ms
Convergence (Local Cache)	+ 31ms
Convergence (Notary Cache)	+ 655ms
Convergence (Cache Miss)	+ 918ms
DANE	*Misleadingly Bad (7 sec)*

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**
Under Research -> Internet Security

Thank you for your time.
adammbates@ufl.edu