Safe SSL Certs in a Hostile World: No Compromise

Jim Baker

jim.baker@rackspace.com

Overview

- Public key management
- Man-in-the-middle and other attacks
- Jython support - providing an OpenSSL-based API on top of Java

About me

- Software engineer at Rackspace
- Core developer of Jython
- Co-author of *Definitive Guide to Jython* from Apress
- Lecturer in CS at Univ of Colorado at Boulder - "Principles of Programming Languages"
- Leader, Boulder/Denver Storm Users meetup
- Formerly, part of original developer team of Ubuntu Juju - lots of experience with ZooKeeper

Background in SSL

- Knew fundamentals

Fundamentals

- Symmetric encryption - AES, formerly DES, 3DES
- Public key encryption - based on asymmetric functions
- Factorization vs multiplication, discrete log, elliptic curve
- SSL starts with public keys, negotiates a onetime symmetric key for session

Public keys

- Want everyone want to know your public keys
- Wrapped in a certificate, signed by. . .
  - Me
  - Others - works in a web of trust
  - Certificate authority

Background in SSL

- Knew fundamentals
- Man-in-the-middle, other possible attacks

Man-in-the-middle

Background in SSL

- Knew fundamentals
- Man-in-the-middle
- But really just an informed user
- Not paying attention to actual vulnerabilities

goto fail;

- Most prominent recent example
- Apple SSL flaw

goto fail;


```
static OSStatus
SSLVerifySignedServerKeyExchange(
    SSLContext *ctx, bool isRsa,
    SSLBuffer signedParams,
    uint8_t *signature, UInt16 signatureLen)
{
    OSStatus        err;
    ...
    if ((err = SSLHashSHA1.update(...)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(...)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(...)) != 0)
        goto fail;
    ...
```

Indented goto fail;


```
static OSStatus
SSLVerifySignedServerKeyExchange(
    SSLContext *ctx, bool isRsa,
    SSLBuffer signedParams,
    uint8_t *signature, UInt16 signatureLen)
{
    OSStatus err;
    ...
    if ((err = SSLHashSHA1.update(...)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(...)) != 0)
        goto fail;
    goto fail;  /* Not a problem in Python! */
    if ((err = SSLHashSHA1.final(...)) != 0)
        goto fail;
    ...
```

Jython SSL

- Need support now that Python ecosystem tools (pip, easy_install, etc) require it
- Cannot just say, use supporting Java API

Rabbit hole?

- Am I really the right person to work on this?
- I'm not a security expert
- I don't have time
- Really this is a question for **all of us**
- We do not need to re-implement algorithms - or devise new ones
- Need to use security carefully

Java security model

- Complex!
- Still doesn't suffice, so usually need to also use complementary Bouncy Castle
- `SSLContext`, `SSLEngine`
- `KeyManager`, `TrustManager`

KeyManager

- Manages key **material**
- Example: private key and corresponding public key certificate
- Can be looked up as necessary by `SSLEngine` / `SSLContext`

TrustManager

- Manages the trust of certificates
- Specific certificate validation (like `goto fail;` earlier)
- Certificate authorities and their root certificates

Don't do this!

A recipe that has circulated and I have even used for testing:

```
class TrustAllX509TrustManager(X509TrustManager):
    """Define a custom TrustManager to
       accept all certificates"""

    def checkClientTrusted(self, chain, auth):
        pass

    def checkServerTrusted(self, chain, auth):
        pass

    def getAcceptedIssuers(self):
        return None
```

Especially don't do this!

Making this be the default for all usages:

```
trust_managers = array(
    TrustManager, [TrustAllX509TrustManager()])
trust_all_context = SSLContext.getInstance("SSL")
trust_all_context.init(None, trust_managers, None)
SSLContext.setDefault(trust_all_context)
```

Equivalent code in Python 2.7

From the docs:

> *Warning The ssl module won't validate certificates by default. When used in client mode, this means you are vulnerable to man-in-the-middle attacks.*

```
ssl.wrap_socket(sock, cert_reqs=CERT_NONE, ...)
```

Proper trust management

- Your solution should be closing security holes
- Do not introduce them
- So delegate

Delegation

- Use `pip` or `easy_install` - they do the right thing
- Use `requests`
- Use the underlying security infrastructure, opening as necessary

Certificate pinning

- Trust as little as possible
- Certificate authorities can be compromised
- If possible, don't use them
- Register your certs in Chrome; support in progress in Firefox
- Register your certs in your mobile app, your Python app

In Jython

```
class CompositeX509TrustManager(X509TrustManager):
  def __init__(self, trust_managers):
    self.trust_managers = trust_managers

  def checkClientTrusted(self, chain, auth_type):
    for trust_manager in self.trust_managers:
      try:
        trustManager.checkClientTrusted(
          chain, auth_type)
        return
      except CertificateException:
        pass
    raise CertificateException(
      "None of the TrustManagers trust cert chain")
...
```

Questions

?