Toward Semantic Cryptography APIs

Tudor Dumitraș
Assistant Professor, ECE & MC2
University of Maryland, College Park

Work with Soumya Indela,
Mukul Kulkarni, and Kartik Nayak (UMD)
Why Johnny Developer Can’t Encrypt?

• User mistakes in using cryptography
  – Why Johnny Can’t Encrypt: A Usability Evaluation of PGP 5.0 [Witten & Tygar, 1999]
  – Stimulated usability research in security

• Developer mistakes in using cryptography [Georgiev+, 2012; Egele+, 2013; Fahl+, 2013; Reaves+, 2015]
  – Examples of mistakes
    • Insecure parameters (ECB mode, constant encryption keys)
    • No authentication (e.g. incorrect SSL certificate validation)
    • DIY crypto
  – Severe impact
    • Leak of personal information (e.g. financial data), can steal money!
Can you find a bug?

[https://docs.python.org/2/library/ssl.html]

Establishing a secure SSL connection (code from Python documentation)

import socket, ssl

c = ssl.SSLContext(ssl.PROTOCOL_TLSv1)
c.verify_mode = ssl.CERT_REQUIRED
c.load_default_certs()

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ssl_sock = c.wrap_socket(s, server_hostname='www.verisign.com')
ssl_sock.connect(('www.verisign.com', 443))

T. Dumitraș :: Toward Semantic Cryptography APIs
Examples of Crypto Mistakes

[https://docs.python.org/2/library/ssl.html]

import socket, ssl

context = ssl.SSLContext(ssl.PROTOCOL_TLSv1)
context.verify_mode = ssl.CERT_REQUIRED
context.load_default_certs()

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ssl_sock = context.wrap_socket(s, server_hostname='www.verisign.com')
ssl_sock.connect(('www.verisign.com', 443))
import socket, ssl

context = ssl.SSLContext(ssl.PROTOCOL_TLSv1)
context.verify_mode = ssl.CERT_REQUIRED
context.check_hostname = True
context.load_default_certs()

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ssl_sock = context.wrap_socket(s, server_hostname='www.verisign.com')
ssl_sock.connect(('www.verisign.com', 443))
import socket, ssl

context = ssl.SSLContext(ssl.PROTOCOL_TLSv1)
context.verify_mode = ssl.CERT_REQUIRED
context.check_hostname = True
context.load_default_certs()
context.verify_flags = ssl.VERIFY_CRL_CHECK_LEAF
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ssl_sock = context.wrap_socket(s, server_hostname='www.verisign.com')
ssl_sock.connect(('www.verisign.com', 443))
Examples of Crypto Mistakes – cont’d

[https://docs.python.org/2/library/ssl.html]

```python
import socket, ssl

cert = ssl.SSLContext(ssl.PROTOCOL_TLSv1)
cert.verify_mode = ssl.CERT_REQUIRED
cert.check_hostname = True
cert.load_default_certs()
cert.verify_flags = ssl.VERIFY_CRL_CHAIN
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ssl_sock = cert.wrap_socket(s, server_hostname='www.verisign.com')
ssl_sock.connect(('www.verisign.com', 443))
```
Summary of Problems

• Default parameters do not ensure security
  – Example: default certs

• Must perform additional security checks
  – Checks are sometimes disabled during development & testing
    (e.g. self-signed certs)

• Security depends on external information
  – Example: certificate revocation status

• Need an understanding of attacks and defenses to use
  crypto APIs correctly
  – No separation of concerns
4 Tips for Better Crypto Code

1. Don’t expect too much from simplified APIs and secure defaults

2. Provide a mechanism to integrate security-critical external information at runtime

3. Confine security decision to code components written by security engineers

4. Provide compile-time mechanisms to turn off security workarounds
1. Secure Defaults and Simplified APIs
Secure Defaults Are Insufficient

• Default values do not ensure security
  – ECB mode is default for AES in PyCrypto, Java JSSE and derived Android libraries (CWE-326)
  – Uninitialized variables may be controlled by an adversary (CWE-453)

• Default values may vary across libraries

$url = "https://example.com";
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, TRUE);
curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 2);
$data = curl_exec($ch);
curl_close($ch);
print_r($data);
Developers Have Diverse Needs

• Various developer needs
  – Simple communication over the Internet
  – Libraries for managing cloud resources
  – Secure financial transactions

• Application specific choices
  – No consensus on best way to check for certificate revocations [Liu+, 2015]
    • Mobile browsers don’t check CRLs
    • OCSP not widely adopted

• Developers must make security decisions
  – The simplified APIs adopted by recent crypto frameworks (e.g. NaCL, cryptography.io) are insufficient
2. Security Engineers vs. Functionality Engineers
How to Use Crypto APIs Correctly?

• Need understanding of attacks and defenses
  – Common mistakes
    • Accept expired certificates (CWE-298, CWE-324)
    • Skip hostname verification (CWE-297, CWE-295)
    • Store secret keys unencrypted (CWE-311, CWE-312)
    • Use hard-coded keys (CWE-321, CWE-259)

• Need understanding of implementation
  – Hostname verification in JSSE: only if algorithm field is set to “HTTPS”
  – Expired certificate: handshake fail (IBM JSSE) vs. handshake succeeds (Oracle JSSE)
    • Same Java code!

No Separation of Concerns!
Semantic API

• **Security Engineers**: write security sensitive code
  – Authentication and authorization
  – Secure communication
  – Secure storage

• **Functionality Engineers**: focus on application logic
  – Use APIs that do not require security knowledge, e.g. secureSend()

• Encapsulate security decisions in code written by security engineers
Decoupling Functionality and Security

Functionality

// Send
send(msg1)

// Send msg securely
secureSend(msg2)

Security

AbstractMessage

isConfidential()

ConcreteMessage

isConfidential()

Communicate

connect()
secureSend()
send()

if (isConfidential (msg))
throw Exception("Use secureSend()!");

...
3. Integrating External Information
Need for External Information

• Security depends on information external to the system
  – Crypto libraries implement algorithms no longer secure
    • MD5 (CWE-327), SHA1 (CWE-759), DES (CWE-328), …
  – Some algorithm parameters are known to be insecure
    • ECB mode, small key sizes, …
  – Accepting revoked certificates is a security threat (CWE-299)

• This information evolves over time
  – In some cases the updates are frequent (e.g. certificate revocation)

Integrate external info at runtime
Regulator Pattern

- Regulator pattern for integrating external security information
  - **Push Model**: receive updates periodically (e.g. CRLs)
  - **Pull Model**: request update when needed (e.g. OCSP)
  - **Selective Pull Model**: receive unforgeable info from server (e.g. OCSP Stapling)
4. Managing Security Workarounds
Need for Security Workarounds

• Security mechanisms prevent certain operations
  – Cannot test all code paths
  – Some functionality is developed before the key infrastructure is in place

• Programmers disable security checks during development and testing
  – Some of these workarounds remain in the public release
  – Example: Using self-signed certificates in production (CWE-296)

Need to manage security workarounds
Compile-Time Workaround Management

• Security workaround management
  – Isolate workarounds in code
  – Disable them with compile-time switch

• Implementation
  – Projects typically maintain separate debug and release build profiles
  – May disable some checks in debug build
  – Example: specify certificate store in Maven pom.xml
    • Debug store may use self-signed certificates
    • Certificate stores populated by security engineers
Alternative Approaches

• Restrict decision choice for developers
  – Simplified cryptography APIs: keyczar, cryptography.io, OpenCCE, NaCL

• Prove certain security properties
  – Static analysis and type systems
    [Hunt & Sands, 2006; van Delft+, 2015]
  – Security API analysis [Anderson, Security Engineering], International Workshop on Analysis of Security APIs

• Follow best practices in API design [Bloch, 2006]
Conclusion

• Developers need the flexibility to select the most appropriate implementation of a security mechanism
  – Code organization: security vs. functionality engineers

• Security often depends on external information
  – Regulator pattern to retrieve such information at runtime

• Security workarounds serve legitimate purpose
  – Isolate them to development build environment with compile time checks
Thank you!

Tudor Dumitraș

tud0rcand3crypt@gmail.com

http://ter.ps/cryptoapi

@tudor_dumitras
Backup Slides
No Separation of Concerns

• Security specific knowledge is required
  – Knowledge of algorithms for example setting “https” algorithm field in JSSE?
  – Knowledge that certificates should be verified back to the root CA

... PKIXParameters params = new PKIXParameters(trustAnchor);
params.setRevocationEnabled(true);
params.addCertPathChecker(new ExtendedKeyUsagePKIXCertPathChecker(clientAuth, newChain[0]));
...
Need for External Information

Comparing algorithms supported by a web browser

- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_RC4_128_MD5
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_DSS_WITH_AES_128_CBC_SHA
- SSL_RSA_WITH_RC4_128_SHA
- TLS_ECDH_ECDSA_WITH_RC4_128_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_EMPTY_RENEGOTIATION_INFO_SCSV
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_DSS_WITH_AES_128_CBC_SHA
- SSL_RSA_WITH_RC4_128_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
- SSL_RSA_WITH_RC4_128_SHA
- TLS_ECDH_ECDSA_WITH_RC4_128_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_EMPTY_RENEGOTIATION_INFO_SCSV
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA

Thus, algorithms need to be updated
Regulator Pattern - Models

Application | Regulator | External Source

[At Regular Intervals]
- Pushes Update
- Check for Update
- Receive Update

[When required or when update not received]
- App. Operation
  - Request Update
  - Receive Update
- App. Response
Selective Pull Model

App-Regulator
Application
Server
Server-regulator
External Source

[At Regular Intervals]
- Request Certificate
- Stapled Response

Signed time stamp
- Initiate Connection
- Signed timestamp

[Signed timestamp not received]
- Certificate
- Certificate validation
- Valid/Invalid Certificate
- Pull validity
- Finalize Connection