Developers Are Users Too:

Designing Crypto and Security APIs That Busy Engineers and Sysadmins Can Use Securely

Matthew Green, Johns Hopkins University
Matthew Smith, University of Bonn

Origins of Usable Security

- Three seminal ‘90 papers are seen as the origin of Usable Security and Privacy research*
  - Zurko and Simon’s: “User-Centered Security”
  - Adams and Sasse’s: “Users Are Not the Enemy”
  - Whitten and Tygar’s “Why Johnny Can’t Encrypt: A Usability Evaluation of PGP 5.0”
- All argued that users should not be seen as a problem to be dealt with,
  - but that security experts need to communicate more with users, and adopt user-centered design approaches.

* Garfinkel et al. Usable Security, History, Themes, and Challenges, 2014

Evaluation Techniques

Evaluating Without Users

E1 Literature Review
E2 Cognitive Walkthrough
E3 Heuristic Evaluation
E4 Model-Based Evaluation

Evaluating With Users

Qualitative

E5 Conceptual Model Extraction
E6 Silent Observation
E7 Think Aloud
E8 Constructive Interaction
E9 Retrospective Testing

+ Interviews, questionnaires,...

Quantitative

E10 Controlled Experiments

10 Usability Principles (Jakob Nielsen)

1. Keep the interface simple!
2. Speak the user’s language!
3. Minimize the user’s memory load!
4. Be consistent and predictable!
5. Provide feedback!
6. Design clear exits and closed dialogs!
7. Offer shortcuts for experts!
8. Help to recover from errors, offer Undo!
9. Prevent errors!
10. Include help and documentation!

7 Characteristics of good APIs by J. Bloch

1. Easy to learn
2. Easy to use, even without documentation
3. Hard to misuse
4. Easy to read and maintain code that uses it
5. Sufficiently powerful to satisfy requirements
6. Easy to extend
7. Appropriate to audience

OpenSSL Error handling

- Most OpenSSL functions will return an integer to indicate success or failure. Typically a function will return 1 on success or 0 on error. All return codes should be checked and handled as appropriate.
- Note that not all of the libcrypto functions return 0 for error and 1 for success.
- There are exceptions which can trip up the unwary.
- For example if you want to check a signature with some functions you get 1 if the signature is correct, 0 if it is not correct and -1 if something bad happened like a memory allocation failure.

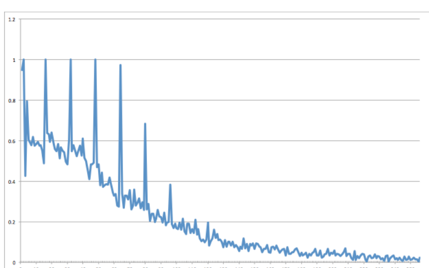
```
if (some_verify_function())
   /* signature successful */
```

```
if (1 != some_verify_function())
   /* signature successful */
```

source: wiki.openssl.org

Algorithm Choices 1/2

- Far too much developer responsibility for choosing and securely composing algorithms
  - Support for unauthenticated encryption (CBC/CTR)
  - RC4!
  - Generic composition of ciphers & MACs
  - Emphasis on legacy applications



How to Break XML Encryption*

Tibor Jager
Horst Görtz Institute for IT Security
Chair for Network- and Data Security
Ruhr-University Bochum
tibor.jager@rub.de

Juraj Somorovsky
Horst Görtz Institute for IT Security
Chair for Network- and Data Security
Ruhr-University Bochum
juraj.somorovsky@rub.de

Algorithm Choices 2/2

- RSA with PKCS #1v1.5 encryption
  - Provided as the only mandatory padding scheme in many software devices (e.g., PKCS11 tokens)
  - **It is conceivably possible to encrypt some types of data securely with PKCS#1v1.5 padding**
  - **Almost nobody knows how to do it (even OpenSSL has active timing vulns.)**

17.4. RSAES-PKCS1-v1_5

17.4.1. Description

The `"RSAES-PKCS1-v1_5"` algorithm identifier is used to perform encryption and decryption ordering to the RSAES-PKCS1-v1_5 algorithm specified in [RFC3447].

Non-intuitive interfaces

CertVerifyCertificateChainPolicy function

This topic has not yet been rated – Rate this topic

The **CertVerifyCertificateChainPolicy** function checks a certificate chain to verify its validity, including its compliance with any specified validity policy criteria.

Syntax

C++

```
BOOL WINAPI CertVerifyCertificateChainPolicy(
  _In_     LPCSTR pszPolicyOID,
  _In_     PCCERT_CHAIN_CONTEXT pChainContext,
  _In_     PCERT_CHAIN_POLICY_PARA pPolicyPara,
  _Inout_  PCERT_CHAIN_POLICY_STATUS pPolicyStatus
);
```

Parameters

Return value

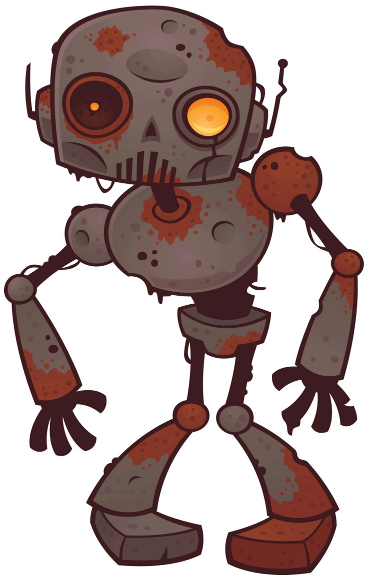
The return value indicates whether the function was able to check for the policy, it does not indicate whether the policy check failed or passed.

If the chain can be verified for the specified policy, **TRUE** is returned and the **dwError** member of the *pPolicyStatus* is updated. A **dwError** of 0 (ERROR_SUCCESS or S_OK) indicates the chain satisfies the specified policy.

If the chain cannot be validated, the return value is **TRUE** and you need to verify the *pPolicyStatus* parameter for the actual error.

TLS Static Code Analysis

- Analysis of 13,500 popular, free apps from Google's Play Market
  - 92.8 % of the apps use the Internet permission
  - 91.7 % of networking API calls are HTTP(S) related
  - 0.8 % exclusively HTTPS URLs
  - 46.2 % mix HTTP and HTTPS
- 17.28 % of all apps that use HTTPS API include code that fails in TLS certificate validation
  - 1070 include critical code
  - 790 accept all certificates
  - 284 accept all hostnames



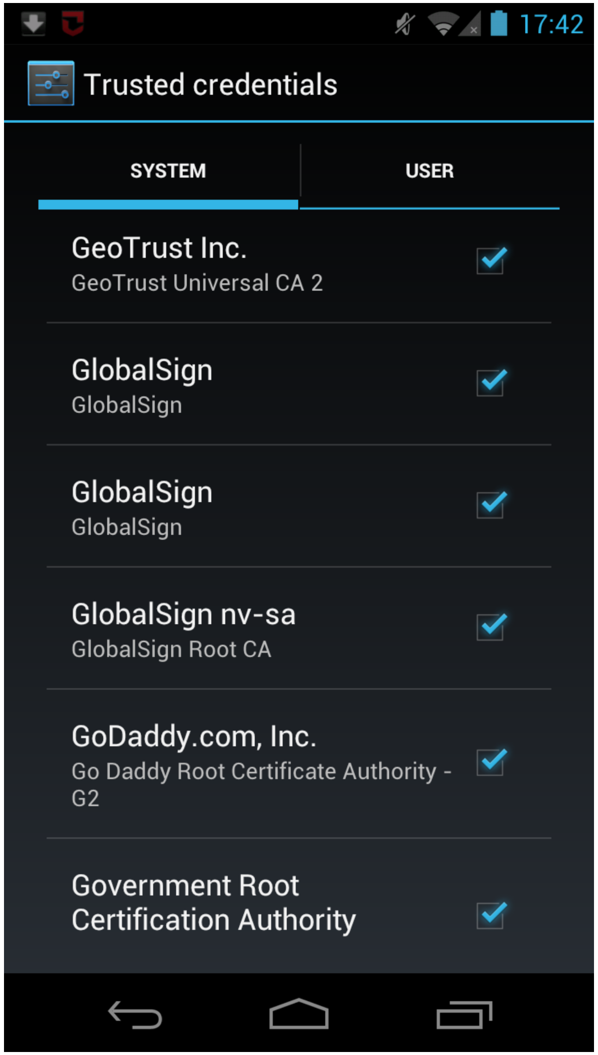
HTTPS on Android

The default Android HTTPS API
implements correct certificate validation.

What could possibly go wrong?

HTTPS Usage on Android and iOS

- A server needs a certificate that was signed by a trusted Certificate Authority
  - (~130 pre-installed CAs)
- For non-trusted certificates a custom workaround is needed
- Error handling requires custom code
- Additional security measures such as pinning or Certificate Transparency require custom code



Trust me I‘m an Engineer

Help?

Q: I am getting an error of „javax.net.ssl.SSLException: Not trusted server certificate“.

[...]

I have spent 40 hours researching and trying to figure out a workaround for this issue.

A: Look at this tutorial

http://blog.antoine.li/index.php/2010/10/android-trusting-ssl-certificates

stackoverflow.com

Trusting all Certificates

```java
// Create a trust manager that does not validate certificate chains
TrustManager[] trustAllCerts = new TrustManager[] { new X509TrustManager() {

        public java.security.cert.X509Certificate[] getAcceptedIssuers() {
                return null;
        }

        public void checkClientTrusted(X509Certificate[] chain, String authType) throws CertificateException {
                // do nothing
        }

        public void checkServerTrusted(X509Certificate[] chain, String authType) throws CertificateException {
                // do nothing
        }

} };
```

Common: Blaming Developers

"It's all the developers' fault!"

Talking To Developers

- Finding broken HTTPS in Android and iOS apps is good…

  …knowing what the root causes are is even better

- We contacted 80 developers of broken apps
  - informed them ✓
  - offered further assistance ✓
  - asked them for an interview ?

- 15 developers agreed ✓

Novice Developers

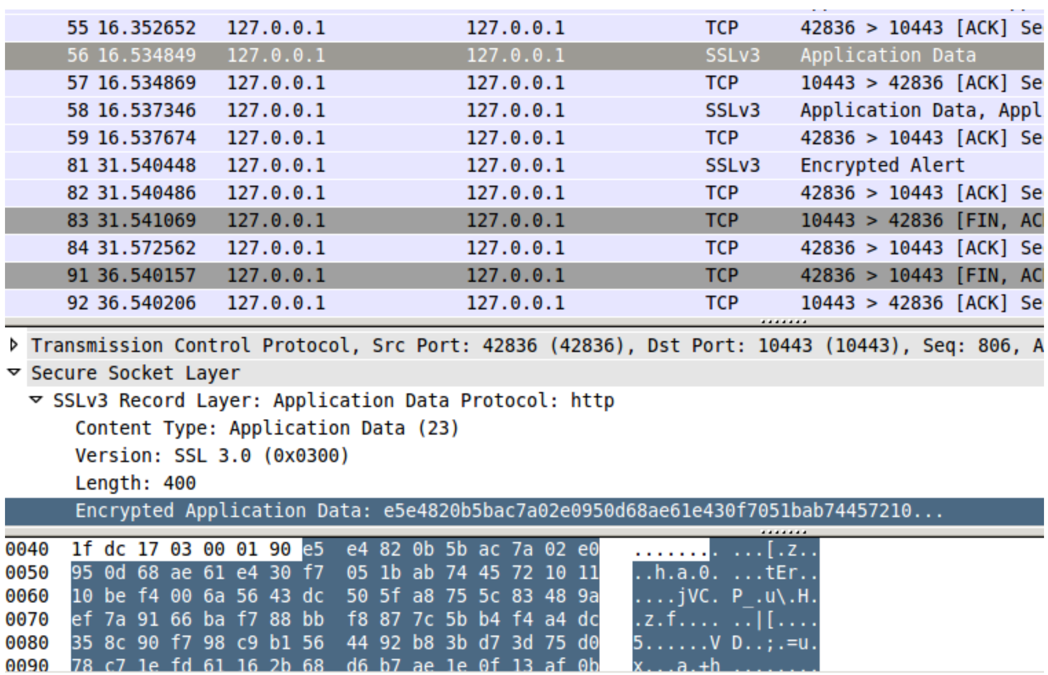
"This app was one of our first mobile apps and when we noticed that there were problems with the SSL certificate, we just implemented the first working solution we found on the Internet."

Intermediate Developers

“We use self-signed certificates for testing purposes and the easiest way to make them working is to remove certificate validation. Somehow we must have forgotten to remove that code again when we released our app.“

Expert Developers (kind of...)

"[...] When I used Wireshark to look at the traffic, Wireshark said that this is a proper SSL protected data stream and I could not see any cleartext information when I manually inspected the packets. So I really cannot see what the problem is here."



Expert Developers (time constrained)

universitätbonn

"The app accepts all SSL certificates because some users wanted to connect to their blogs with self-signed certs and […] because Android does not provide an easy-to-use SSL certificate warning message, it was a lot easier to simply accept all self-signed certificates."

vs.

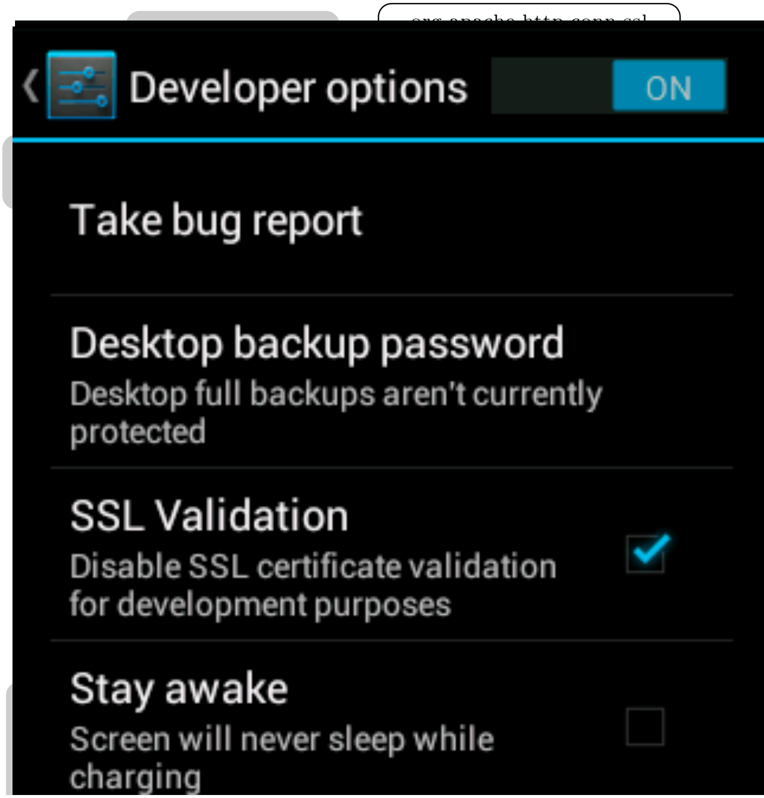
Developer Survey Summary

- Self-Signed Certificates – Development.
  - Developers commonly wish to use self-signed certificates for testing purposes and hence want to turn off certificate validation during testing.
- Self-Signed Certificates – Production.
  - A few developers wanted to use self-signed certificates in their production app for cost, effort and customer satisfaction reasons.
- Code Complexity.
  - Developers described the code-level customization features of HTTPS as too complex and requiring too much effort.
- Certificate Pinning / Trusted Roots.
  - Developers liked the idea of having an easy way to limit the number of trusted certificates and/or certificate authorities.
- Global Warning Message.
  - Developers requested global HTTPS warning messages since they described building their own warning messages as too challenging.

A new approach TLS on Android

Changed the TLS API on Android

- Removed TrustManager extension capabilities – no overriding of errors
- Support self-signed certificates
- Support certificate Pinning
- Offer default warning / user interaction
- Integration via configuration




```
<uses-ssl>
    <pins host="securessl.com">
        <pin type="ca" comment="Verisign Root CA">
            8F:57:5A:C8:5B:09:63:B0:24:2B:90...
        </pin>
        <pin type="cert" comment="Self-Signed">
            18:DA:D1:9E:26:7D:E8:BB:4A:21:58...
        </pin>
    </pins>
</uses-ssl>
```

10 Rules for a good Crypto API?

universitätbonn

1. Easy to learn, even without crypto background
2. Easy to use, even without documentation
3. Hard to misuse. Incorrect use should lead to visible errors
4. Hard to circumvent errors – except during testing/development
5. Easy to read and maintain code that uses it
6. Sufficiently powerful to satisfy non-security requirements
7. ~~Easy to extend~~ Hard to change/override core functionality
8. Appropriate to audience – this means people with no crypto experience
9. Assist with/handle end-user interaction
10. However, where possible integrate into standard APIs so normal developers never have to interact with crypto APIs in the first place

conduct developer studies

Evaluation Techniques

Evaluating Without Users

- E1 Literature Review
- E2 Cognitive Walkthrough
- E3 Heuristic Evaluation
- E4 Model-Based Evaluation

Evaluating With Users

Qualitative

- E5 Conceptual Model Extraction
- E6 Silent Observation
- E7 Think Aloud
- E8 Constructive Interaction
- E9 Retrospective Testing

+ Interviews, questionnaires,...

Quantitative

- E10 Controlled Experiments