PA193 Secure coding principles and practices

Seminar 4: Usability and usable security for cryptographic APIs

10. 10. 2019

Martin Ukrop, mukrop@mail.muni.cz

Ph.D. research cooperation

CRoCS, Faculty of Informatics, Masaryk University

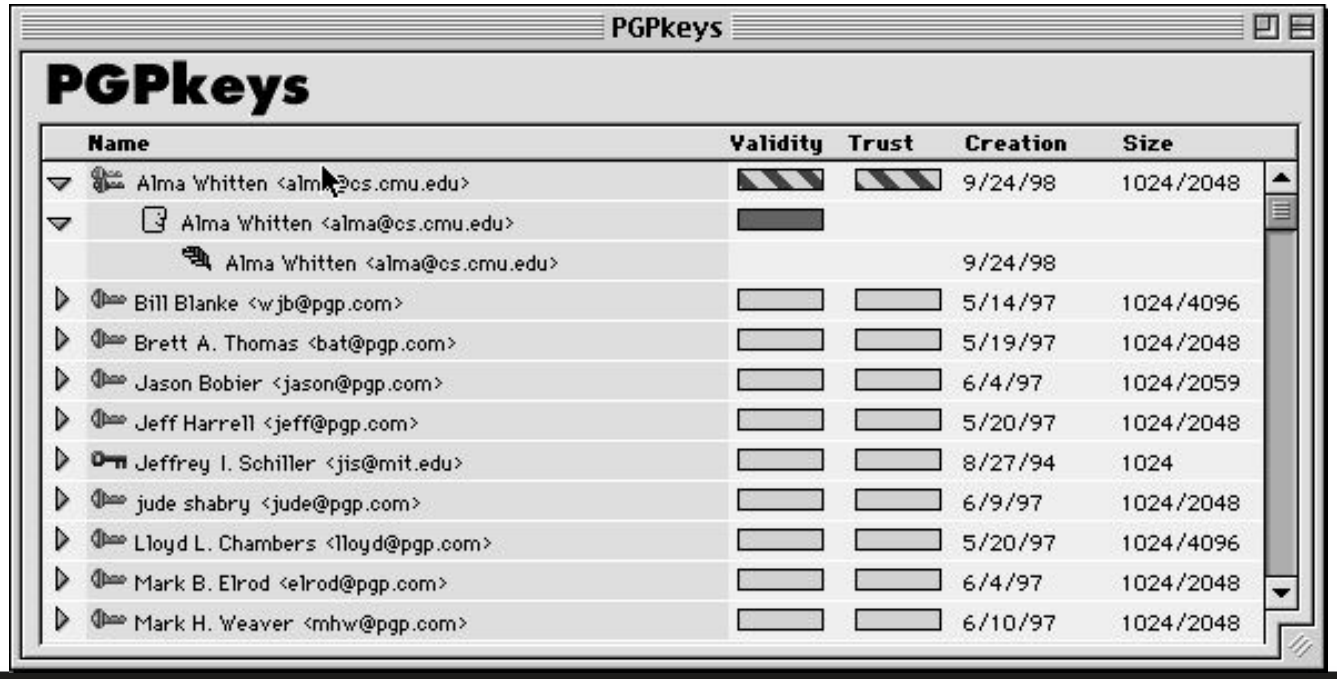
Centre for Research on Cryptography and Security

Seminar overview

- Usable security for end users
- Usable security for developers
- Examples of unusable APIs
- Designing unusable APIs
- Current usable security research at CRoCS
- Homework assignment

Usable security: How it all began

- "Why Johny can't encrypt" (A. Whitten and J. D. Tygar, 1999)
  - A usability study of PGP 5.0 (exchange secure email)
  - 4/12 success, 7/12 their keys, exposed private, ...



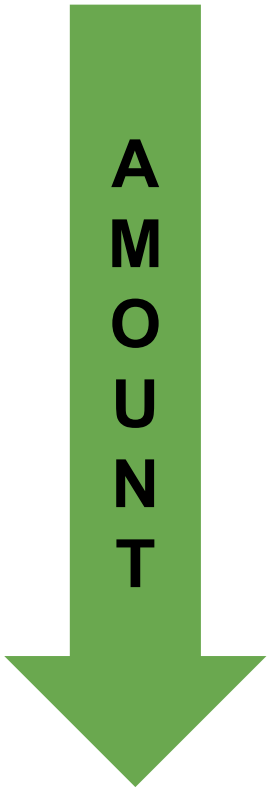
Users are not the enemy

- M. A. Sasse and A. Adams, 1999
  - Study of password authentication
- Consensus at the time:
  - Users are careless and unmotivated (security-wise)
  - Users lack security knowledge
- Promoting “user-centered approach”

Usable security research since then

- Email encryption (“The Johnny series”)
- SSL browser warnings (testing adherence)
- Authentication (passwords, tokens, ...)
- Security advice (passwords, ...)
- Android permissions
- Privacy issues in software
- ...

However… (The impact pyramid)



OS developers

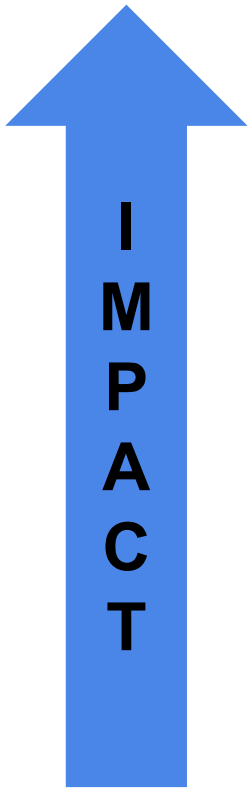
Library developers

Software developers

Administrators/IT support

IT professionals

End users



SSH: Authenticity can’t be established

```
[xukrop@styx ~]$ ssh aisa
The authenticity of host 'aisa.fi.muni.cz (147.251.48.1)' can't be established.
ECDSA key fingerprint is SHA256:QcU0hBKPumwmV4WFWf8OzReJc1lLtzr3wVJF5Cqlij8.
ECDSA key fingerprint is MD5:af:79:1b:77:ad:74:3c:35:e3:0b:60:78:f0:a4:3d:7f.
Are you sure you want to continue connecting (yes/no)?
```

```
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'aisa.fi.muni.cz' (ECDSA) to the list of known hosts.
Last login: Mon Nov 20 21:23:45 2017 from eduroam44-237.fi.muni.cz
aisa:/home/xukrop>$
```

SSH: Key changed

```
[xukrop@styx ~]$ ssh aisa
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@       WARNING: POSSIBLE DNS SPOOFING DETECTED!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The ED25519 host key for aisa.fi.muni.cz has changed,
and the key for the corresponding IP address 147.251.48.1
is unchanged. This could either mean that
DNS SPOOFING is happening or the IP address for the host
and its host key have changed at the same time.
Offending key for IP in /etc/ssh/ssh_known_hosts:960
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ED25519 key sent by the remote host is
SHA256:Js2Haw++eY49mzCoS8ZNMfAKrWXDqSKVpvQmidQvydw.
Please contact your system administrator.
Add correct host key in /home/xukrop/.ssh/known_hosts to get rid of this message.
Offending RSA key in /etc/ssh/ssh_known_hosts:71
ED25519 host key for aisa.fi.muni.cz has changed and you have requested strict checking.
Host key verification failed.
[xukrop@styx ~]$
```

(Un)usability of HTTPS deployment

- Empirical test of HTTPS deployment usability
  - 28 knowledgeable university students
  - Asked to deploy TLS on Apache to pass security audit
  - Security measured by Qualys SSL test

(Un)usability of HTTPS deployment

- Empirical test of HTTPS deployment usability
  - 28 knowledgeable university students
  - Asked to deploy TLS on Apache to pass security audit
  - Security measured by Qualys SSL test
- Only 4 deployed A-grade TLS
  - Challenges: Find the right information, generate CSR, choose appropriate cipher-suites, strict HTTPS, multiple config files, security/compatibility ballance

Research from *"I Have No Idea What I'm Doing" – On the Usability of Deploying HTTPS*, by K. Krombholz, W. Mayer, M. Schmiedecker and E. Weippl, 2017.

Authenticated encryption

- To get confidentiality + integrity
  - Encryption for confidentiality
  - MAC for integrity

Authenticated encryption

- To get confidentiality + integrity
  - Encryption for confidentiality
  - MAC for integrity
- Standard: In what order to combine them?
  - 4 possibilities
  - 1 almost always right, 1 depends, 2 always wrong

Authenticated encryption

- To get confidentiality + integrity
  - Encryption for confidentiality
  - MAC for integrity
- Standard: In what order to combine them?
  - 4 possibilities
  - 1 almost always secure, 1 depends, 2 always insecure
- NaCl/libsodium: crypto_box API
  - c = crypto_box(m, n, pk, sk);
  - m = crypto_box_open(c, n, pk, sk);
  - Usable API is not a silver bullet!

“Developer-resistant cryptography!”

K. Cairns and G. Steel, 2014

Example 1: CURL (2012)

- Libcurl
  - the multiprotocol file transfer library

- Two main directives for SSL validation
  - **`CURL_SSL_VERIFYPEER`** (checking certificate)
  - **`CURL_SSL_VERIFYHOST`** (checking hostname)

Example 1: CURL (2012)

- PayPal SDK:

```
curl_setopt($ch, CURL_SSL_VERIFYPEER, FALSE)
curl_setopt($ch, CURL_SSL_VERIFYHOST, FALSE)
```

Example 1: CURL (2012)

- PayPal SDK: version from 27th April 2012

```
curl_setopt($ch, CURL_SSL_VERIFYPEER, TRUE)
curl_setopt($ch, CURL_SSL_VERIFYHOST, TRUE)
```

Example 1: CURL (2012)

- PayPal SDK: version from 27th April 2012

```
curl_setopt($ch, CURL_SSL_VERIFYPEER, TRUE)
curl_setopt($ch, CURL_SSL_VERIFYHOST, TRUE)
```

- **Bool** `CURL_SSL_VERIFYPEER`
- **Int** `CURL_SSL_VERIFYHOST`
  - 0: no host verification
  - 1: debug (nearly no verification)
  - 2: verify hostname

CURL example from *The Most DangerousCode in the World: Validating SSL Certificates in Non-Browser Software*, 2012.

Example 2: OpenSSL Random API (2014)

OpenSSL functions for random bytes

```
int RAND_bytes(unsigned char *buf, int num);

int RAND_pseudo_bytes(unsigned char *buf, int num);
```

- Working as *providers*
  - *I.e. multiple implementations*
  - *I.e. multiple callers*

Example 2: OpenSSL Random API (2014)

Documentation: Description

- ***RAND_bytes(buf, num)*** puts *num* cryptographically strong pseudo-random bytes into *buf*. An error occurs if the PRNG has not been seeded with enough randomness to ensure an unpredictable byte sequence.
- ***RAND_pseudo_bytes(buf, num)*** puts *num* pseudo-random bytes into *buf*. Pseudo-random byte sequences generated by *RAND_pseudo_bytes()* will be unique if they are of sufficient length, but are not necessarily unpredictable. They can be used for non-cryptographic purposes and for certain purposes in cryptographic protocols, but usually not for key generation etc.

Example 2: OpenSSL Random API (2014)

Documentation: Return values

- ***RAND_bytes()*** returns *1* on success, *0* otherwise. The error code can be obtained by *ERR_get_error(3)*.
- ***RAND_pseudo_bytes()*** returns *1* if the bytes generated are cryptographically strong, *0* otherwise.
- Both functions return *-1* if they are not supported by the current RAND method.

Example 2: OpenSSL Random API (2014)

Implementations

- engines/e_aep.c:
  - *RAND_bytes()* same function as *RAND_pseudo_bytes()*.
  - Returns *0* on error, and adds an error.
  - Can fail without altering buffer.
- 20 similar other implementations in OpenSSL, Bind 9 and Heimdal
  - Always implemented by the same back-end function
  - Only a single implementation with semi-compliant error codes

Example 2: OpenSSL Random API (2014)

Callers

- Standard C error scheme incorrect:

```
if (!RAND_bytes(...)) /* handle error ... */
```

  - Debian code search: OpenSSL, Ruby, net-snmp, ZNC, DACS, dnsval/dnssec-tools, ...
  - GitHub code search: 1 456 results

RAND example by Joseph Birr-Pixton (http://jbp.io)

API issues

- Usability issues
  - *E.g. counterintuitive structure/names/arguments*
- Performance issues
  - *E.g. making copy of structures in arguments*
- Security issues

API issues

- Usability issues
  - *E.g. counterintuitive structure/names/arguments*
- Performance issues
  - *E.g. making copy of structures in arguments*
- Security issues
  - Due to bad technical design
    - *E.g. possible compromise by unexpected call order*
  - Due to bad usability
    - *E.g. prone to accidentally disable cert. verification*

Task: Intentional bad design

Design a C (or C-like) API for encryption/decryption routines with security issues due to bad usability.

- Work in groups of 2-3 people
- Get inspiration from existing API structure
  - OpenSSL AES header, OpenSSL enc/dec Wiki
  - GnuTLS encryption, API reference
  - NaCl encryption
  - mbedTLS encryption module

Usable security research in CRoCS

- Towards end users
  - Authentication (QR codes, tokens, ...)
  - PUA behavior in antivirus installation
- Towards IT professionals
  - Using CLI tools (OpenSSL, GnuTLS, NSS)
  - Understanding error messages
  - Comparing cryptographic APIs

Homework assignment

Experiencing different APIs, usability assessment

Homework overview

- Implement a simple TLS client in C connecting to a server, checking its certificate and performing a “HTTPS GET” for www.example.com.
- Do this for 3 different SSL libraries
  - OpenSSL, GnuTLS, mbedTLS (formerly PolarSSL)
  - Code skeletons provided for easy start in IS
- Compare relevant API parts of these libraries using the provided questions

TLS and terminology recap

- Transport Layer Security
  - Based on SSL 3.0 (prohibited in 2015)
  - Versions: 1.0 (1999), 1.1 (2006), 1.2 (2008), 1.3 (2018)
  - Different cipher suites available
- Certificate checking
  - Complicated, often not done (root, hostname!)
- Certificate revocation
  - Certificate Revocation Lists (CRL)
  - Online Certificate Status Protocol (OCSP)

Precise task

- Initiate a TLS handshake with www.example.com
- Validate the server certificate
  - Check at least expiration, server hostname, validity of the chain and the revocation status (CRL or OCSP)
  - Use the default OS trust store
  - *Tip1: Use badssl.com for debugging*
  - *Tip2: For revoked use revoked.grc.com*
- Force the minimal version of TLS 1.2.
- Perform a HTTPS GET on the root website.
- Correctly close the connection.

Homework technicalities

- Library version specifics
  - *OpenSSL >= 1.1.1* (!)
  - GnuTLS ~ 3.6.5
  - mbed TLS ~ 2.16.0
- Feel free to use your own machines
  - FI machines (aisa, nymfeXX, ...) are not suitable
  - Pre-set Ubuntu 19.04 virtual machine is available in IS (login: ‘vagrant’, password: ‘vagrant’)
- HW submission used also for research purposes at CRoCS (anonymous, opt-out possible)

Homework hand-in

- Deadline: 2019-10-17 23:59 CET
- Start from skeleton in IS
  - Stubs for all 3 libraries + Makefile for compilation
- Hand in 1 zip file containing
  - 3x source code (use skeleton skeleton structure)
  - reflection.pdf (at least 1 page, 5-10 specific points comparing the APIs)
    (*Tip: Get inspiration from the following questions.*)
- Partial points definitely possible!

- Do you find the API abstraction level appropriate to the task?
- Did you have to learn about the underlying architecture of the API and other conceptual information before starting to do anything useful related to your task?
- What are the information you had to maintain while completing the tasks?
- Does the amount of code required for this scenario seem just about right, too much, or too little? Why?
- How easy is it to stop in the middle of the scenario and check the progress of work so far?
- When you are working with the API, can you work on your programming task in any order you like, or does the system force you to think ahead and make certain decisions first?
- What are the places where you had to understand the intricate working details of the API while you work on your programming task?
- Did you have to extend types exposed by the API by providing their own implementation of custom behavior to accomplish task? What are the types you had to extend? Explain why you needed to extend the original type provided by the API in each case.

- When you need to make changes to previous work, how easy is it to make the change? Why?
- Were there different parts of the API that mean similar things, is the similarity clear from the way they appear? Please give examples.
- When reading code that uses the API, is it easy to tell what each section of code does? Why?
- Did the types of the API map directly onto the types and concepts you expected? If not, please mention the types you expected and how it was supported in the API.
- Have you came up with incidents where you incorrectly used the API and then identified the correct way of doing that? Did API give any help to identify that you used the API incorrectly? If there any similar incidents, please explain.
- Do you think the security of the end user of the application you developed, depends on how you completed the task? Or does it depend only on the security API you used?
- Did the API provide any guidance on how to test your application?

Selected from *A Generic Cognitive Dimensions Questionnaire to Evaluate the Usability of Security APIs* by Chamila Wijayarathna, Nalin A.G. Arachchilage and Jill Slay.