FOSS crypto

JP Aumasson (@veorq)

This talk:

Get you to know common **FOSS crypto libs**

What they can do for you

Not a howto

Role of crypto libraries and APIs:

Allow you to use third-party code
for crypto **protocols and algorithms**

“Don’t roll your own crypto implementations”

OpenSSL™
Cryptography and SSL/TLS Toolkit

ARM® mbed™

NaCl: Networking and Cryptography library

OPEN SOURCE, OPEN ACADEMIA, OPEN MINDS
LibTom PROJECTS

Crypto++® Library 5.6.2

libsodium

Botan

KRYPT
SEMPER FI
platform- and library-independent cryptography for Ruby

The Legion of the Bouncy Castle

Network Security Services

cryptlib

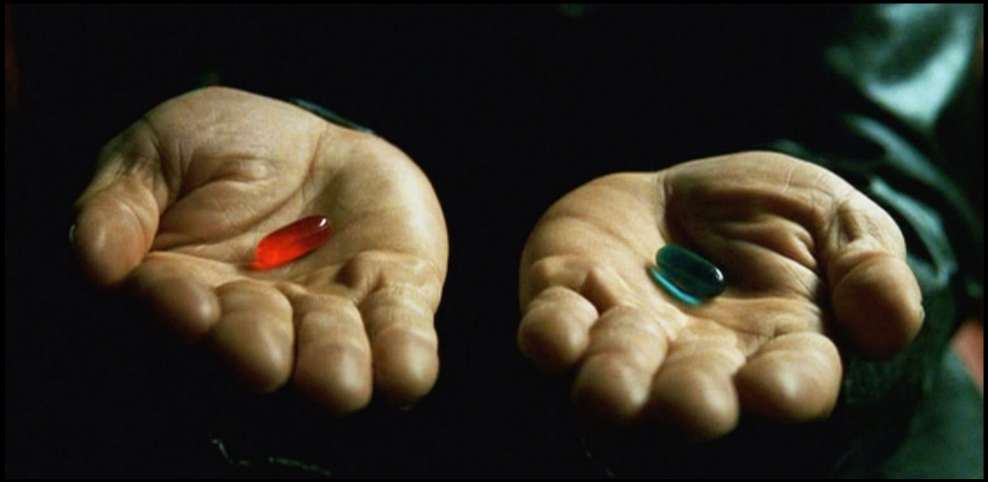
TLS Lite

crypto-js
JavaScript implementations of standard and secure cryptographic algorithms

Many more...

Choosing the right lib is difficult

Define your **requirements**



Differentiators:

Language
License
Functionality
Algorithms and protocols
API level
Security
Performance

Language:

Most libs written in C(++)

C# and Java for Bouncy Castle

JavaScript libs, pure JS or Emscripten’d

Popular libs already have **bindings** for most common languages; you may write your own

License:

Often permissive

OpenSSL: Apache 1.0 and 4-clause BSD

both permissive, no copyleft, not GPL-compatible

mbed TLS: GPLv2 with possible exceptions

NaCl: “Public domain”

LibTomCrypt: WTFPL (Do What the Fuck You Want to PL)

BouncyCastle: MIT (permissive, no copyleft, OSI, GPL compatible)

Crypto++: Boost 1.0 (MIT-like)

Functionality:

Do you need a whole TLS or just an AES?

Or a more specific protocol, like OTR chat?

Algorithms and protocols:

Established standards vs. state-of-the-art

Single algorithm vs. a collection of algorithms

Crypto++: AES, Blowfish, Camellia, CAST-256, DES, DESX, 3DES, GOST, IDEA, MARS, Panama, RC2, RC4, RC5, Salsa20, SEED, Serpent, SHACAL-2, Skipjack, Sosemanuk, Square, TEA, XTEA

in modes CBC, CCM, CFB, CTR, CTS, EAX, GCM, OFB

NaCl: Salsa20, AES-128-CTR

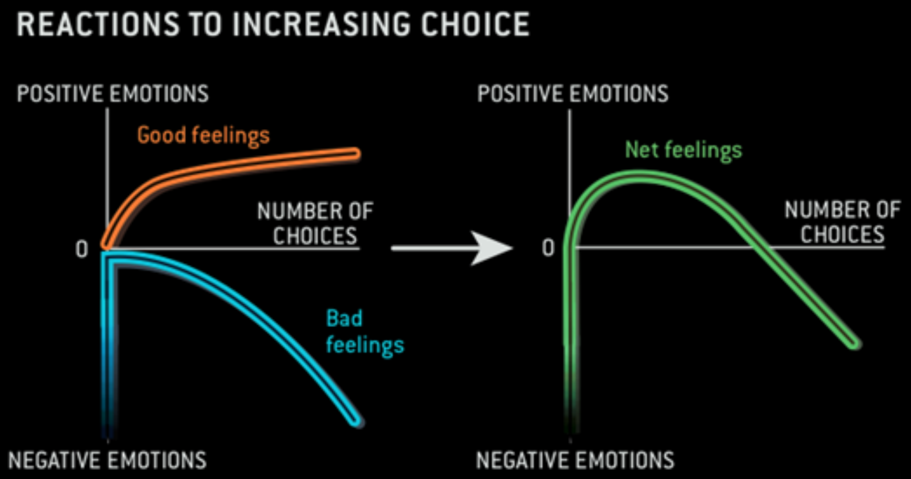
Secure session = key agreement followed by authenticated encryption

OpenSSL implements most TLS standards, cipher suites, features and options, etc.

NaCl only implements its custom algorithms, without all the session establishment

API level:

The fewer choices/freedom/options, the fewer chances to get it wrong



Example of a **high-level** API: NaCl

```
/* key generation */
pk = crypto_box_keypair( &sk )

/* authenticated encryption */
c = crypto_box( m, n, pk, sk )

/* decryption and verification */
m = crypto_box_open( c, n, pk, sk )
```

Example of a **low-level** API: OpenSSL

```
/* RSA key generation */
EVP_PKEY_CTX_set_rsa_keygen_bits(kctx, 2048);
EVP_PKEY_keygen(kctx, &key);

/* omitting generation of a symmetric key... */

/* encrypting one message with AES-256-CBC */
EVP_EncryptInit(&ctx, EVP_aes_256_cbc(), key,
iv); EVP_EncryptUpdate(&ctx, out, &outlen1, in,
sizeof(in)); EVP_EncryptFinal(&ctx, out +
outlen1, &outlen2);

/* (...) */
```

Security:

Most important criteria: if crypto doesn’t do its job, why bother?

“Usual” software bugs: logical bugs, memory corruptions, memory leaks, etc.

Crypto bugs: incorrect implementations, oracles, timing leaks, fault attacks, etc.

Most of the popular libraries sport **complex and non-intuitive APIs** that present the developer with numerous choices, many of of which are insecure. The result is that even experienced developers routinely select dangerous combinations. The visible consequence is a superabundance of security vulnerabilities in recent cryptographic software (...)

Matthew Green

https://www.usenix.org/conference/hotsec13/crypto-apis

OpenSSL:

Many LoCs => more bugs (not good)

Many eyeballs => more bug reports (good)

Often prioritized speed and functionality

Fragile against cache-timing and oracle attacks

NaCl:

Few LoCs, DJB-quality code => fewer bugs

No major bug reported

Only inherently safe primitives

Time-constant, no secret branchings, etc.

Performance (speed):

Sometimes crucial, sometimes unimportant

OpenSSL: fast implementations of algorithms, CPU-specific, using assembly optimizations

NaCl: choice of fast algorithms, suited for fast implementations

A closer look at popular and unique libs...

OpenSSL

Obviously

libcrypto, EVP API + command-line toolkit

More than 460,000 lines of code

https://openssl.org https://wiki.openssl.org

ASN.1 parsing, CA/CRL management

crypto: RSA, DSA, DH*, ECDH*; AES, CAMELLIA, CAST, DES, IDEA, RC2, RC4, RC5; MD2, MD5, RIPEMD160, SHA*; SRP, CCM, GCM, HMAC, GOST*, PKCS*,

PRNG, password hashing, S/MIME

X.509 certificate management, timestamping

some crypto accelerators, hardware tokens

clients and servers for SSL2, SSL3, TLS1.0, TLS1.1, TLS1.2, DTLS1.0, DTLS1.2

SNI, session tickets, etc. etc.

*nix

BeOS

DOS

HP-UX

Mac OS Classic

NetWare

OpenVMS

ULTRIX

VxWorks

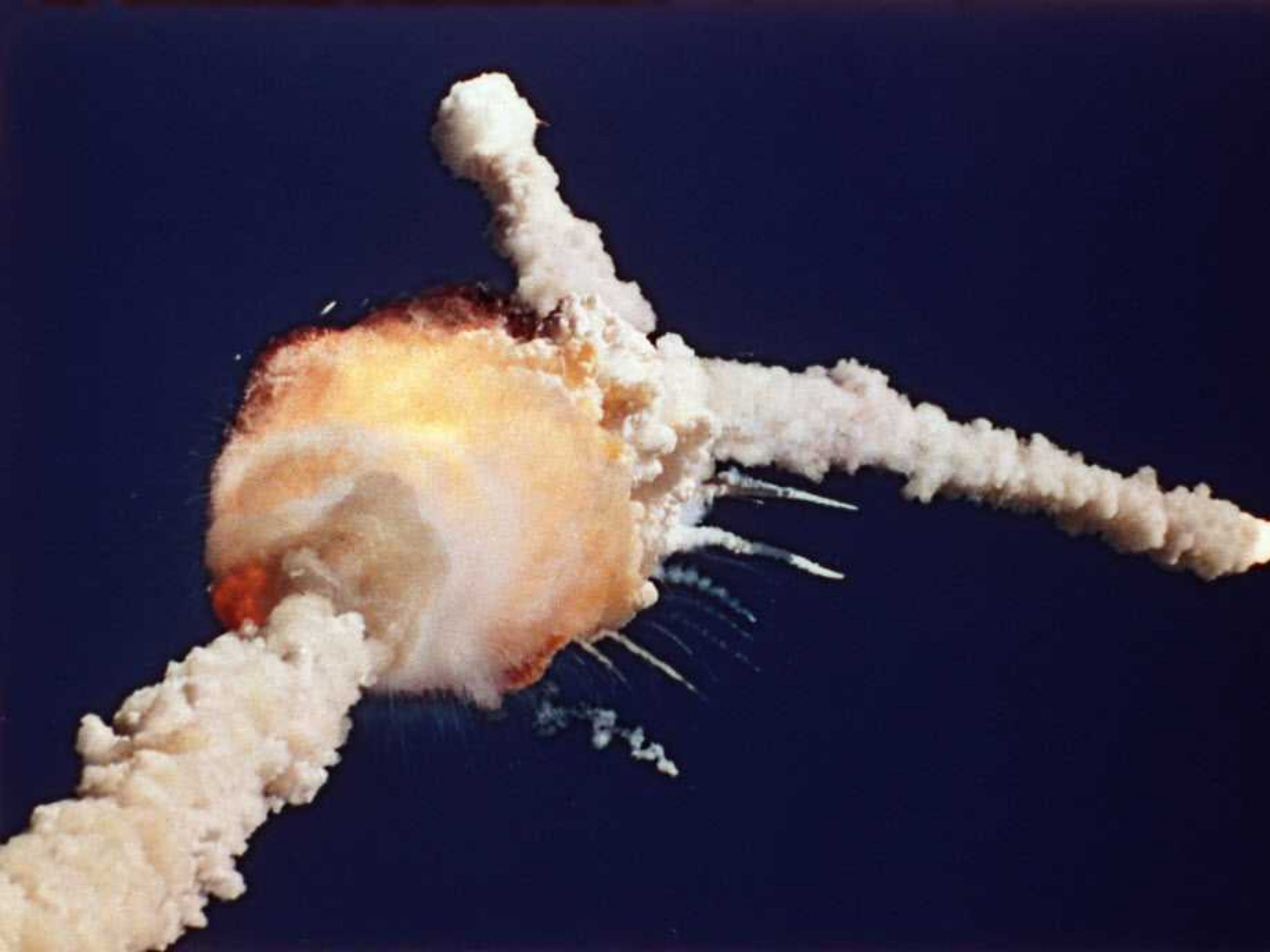
Win* (including 16-bit, CE)

OpenSSL is the space shuttle of crypto libraries. It will get you to space, provided you have a team of people to push the ten thousand buttons required to do so.

Matthew Green

I promise nothing complete; because any human thing supposed to be complete, must not for that very reason infallibly be faulty.

Herman Melville, in Moby Dick



```
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, \
                  3 + payload + padding);
```

```
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, \
                  3 + payload + padding);
```

payload is not the payload but its length (pl is the payload)

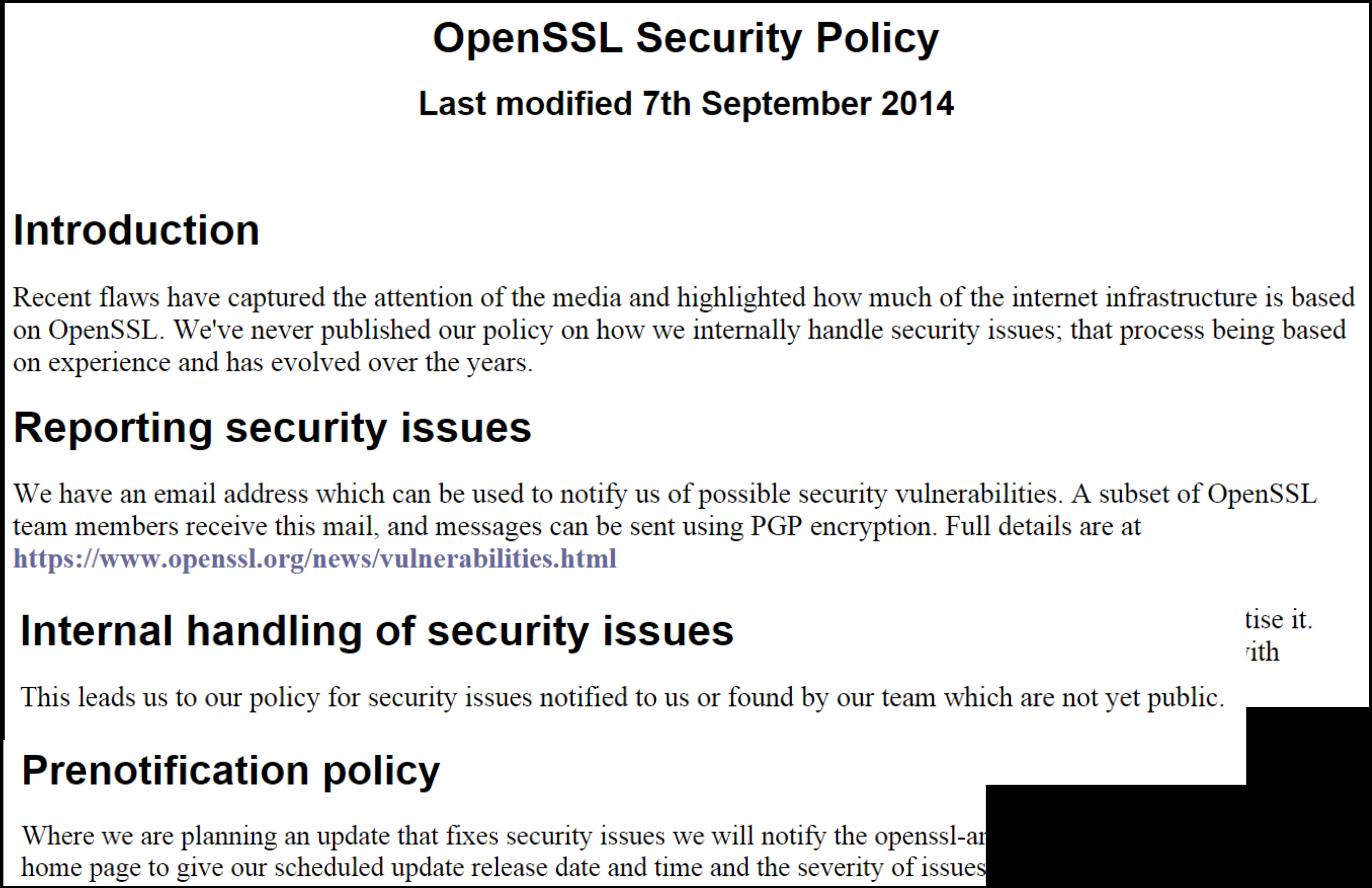
Easy to criticize OpenSSL’s code…

Source code and API complex, often confusing

Large codebase, many contributors

Few quality- and security-control processes

Recent effort: https://www.openssl.org/about/secpolicy.html



OpenSSL Security Policy

Last modified 7th September 2014

Introduction

Recent flaws have captured the attention of the media and highlighted how much of the internet infrastructure is based on OpenSSL. We've never published our policy on how we internally handle security issues; that process being based on experience and has evolved over the years.

Reporting security issues

We have an email address which can be used to notify us of possible security vulnerabilities. A subset of OpenSSL team members receive this mail, and messages can be sent using PGP encryption. Full details are at https://www.openssl.org/news/vulnerabilities.html

tise it.
ith

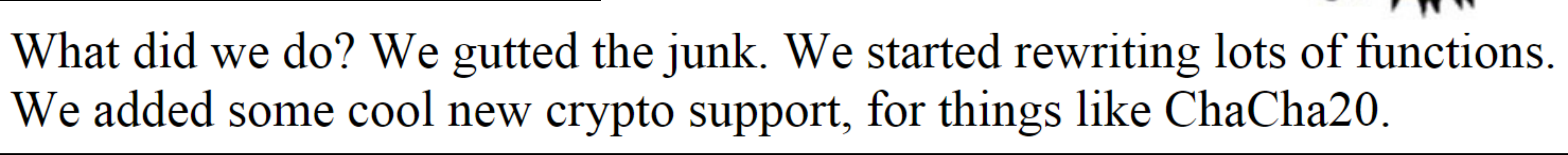
Internal handling of security issues

This leads us to our policy for security issues notified to us or found by our team which are not yet public.

Prenotification policy

Where we are planning an update that fixes security issues we will notify the openssl-ar
home page to give our scheduled update release date and time and the severity of issues

LibreSSL


What did we do? We gutted the junk. We started rewriting lots of functions. We added some cool new crypto support, for things like ChaCha20.

Initiative of the **OpenBSD** community

Big progress in little time

Portable version and OpenBSD version

libtls library for simpler TLS clients and servers

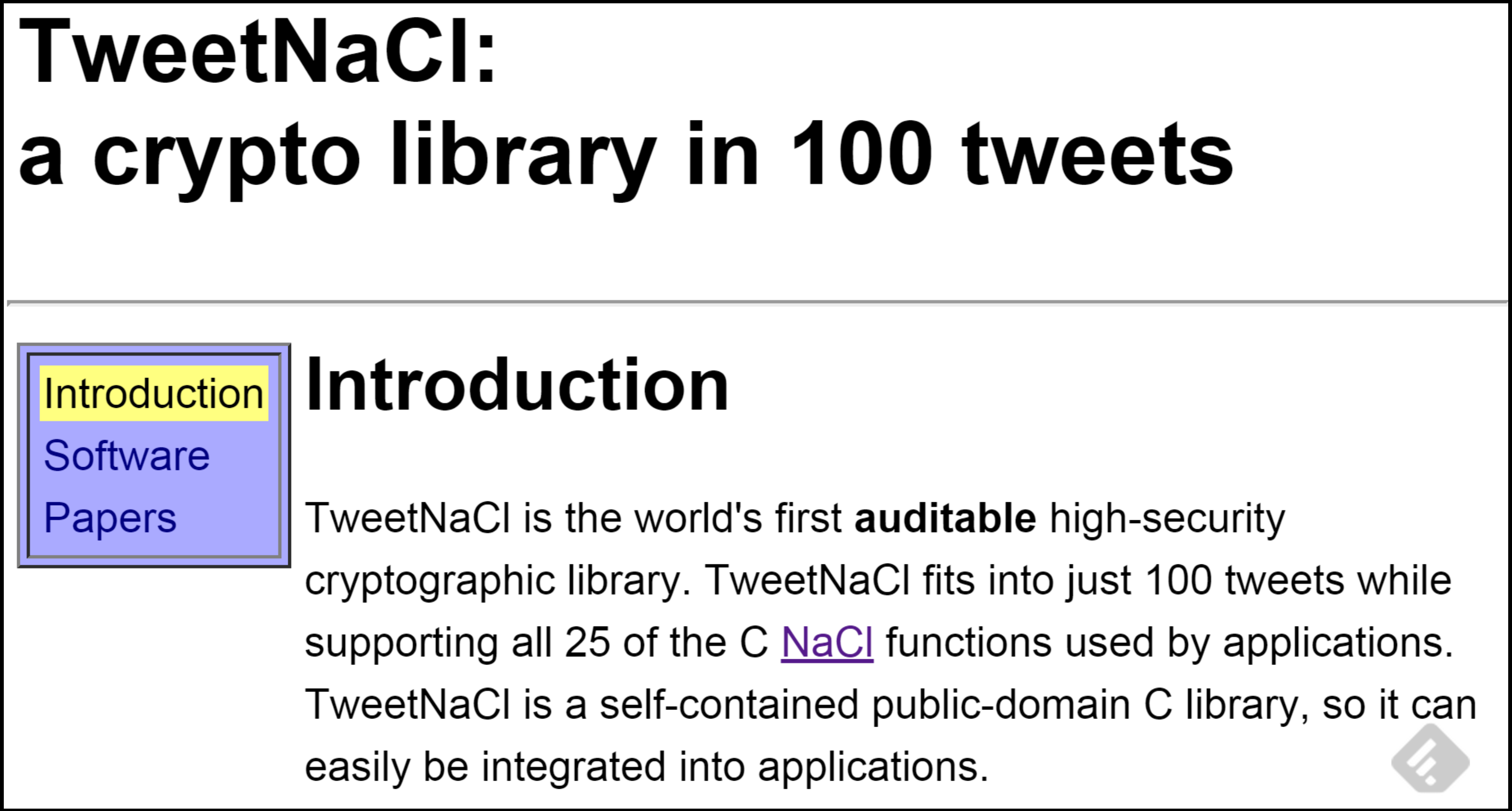
NaCl (“salt”)

The anti-OpenSSL

High-security and high-speed {primitives, code}

About 15,000 lines of code

http://nacl.cr.yp.to



TweetNaCl: a crypto library in 100 tweets

Introduction

TweetNaCl is the world's first **auditable** high-security cryptographic library. TweetNaCl fits into just 100 tweets while supporting all 25 of the C NaCl functions used by applications. TweetNaCl is a self-contained public-domain C library, so it can easily be integrated into applications.

975 lines of code!

NaCl is more like an elevator — you just press a button and it takes you there. No frills or options.

Matthew Green

The other side of the coin:

Restricted set of algorithms and functionalities
Limited portability, non-standard build system
Irregularly updated (some bugs remain unfixed)

libsodium

“a portable, cross-compilable, installable, packageable fork of NaCl, with a compatible API, and an extended API to improve usability even further.” https://download.libsodium.org/doc/

Builds on Windows, OS X, iOS, Android, etc.

Bindings for all common languages

Compiled to pure JavaScript: libsodium.js

```
    prompt_input("a key", (char*)key, sizeof key, 0);
    message_len = prompt_input("a message", (char*)message,
sizeof message, 1);

    printf("Generating %s authentication...\n",
crypto_auth_primitive());
    crypto_auth(mac, message, message_len, key);

    printf("Authentication tag: ");
    print_hex(mac, sizeof mac);

    puts("Verifying authentication tag...");
    ret = crypto_auth_verify(mac, message, message_len, key);
    print_verification(ret);

    sodium_memzero(key, sizeof key); /* wipe sensitive data */
```

An even more specific library...

libotr

Implements the off-the-record (OTR) protocol

Runs on top of instant messaging systems

https://github.com/off-the-record/libotr https://otr.cypherpunks.ca/

libotr is not a travesty of confusion and neglect like openssl. In fact, it shows encouraging signs of being competently written.

Joseph Birr-Pixton

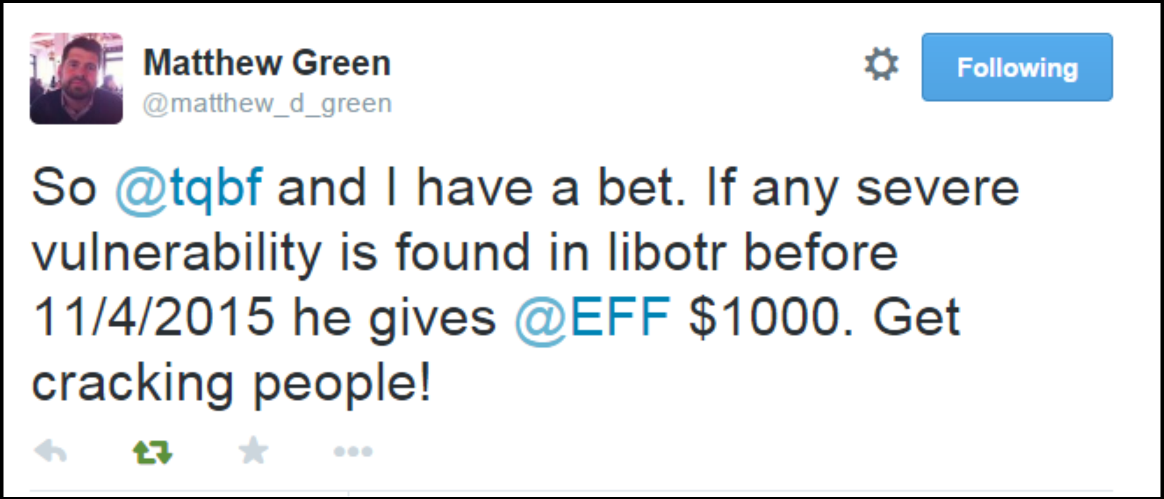
http://jbp.io/2014/08/28/libotr-code-review/

libotr

Quality code, consistent, commented

Does one thing and does it well

Good security track record



So @tqbf and I have a bet. If any severe vulnerability is found in libotr before 11/4/2015 he gives @EFF $1000. Get cracking people!

Conclusions

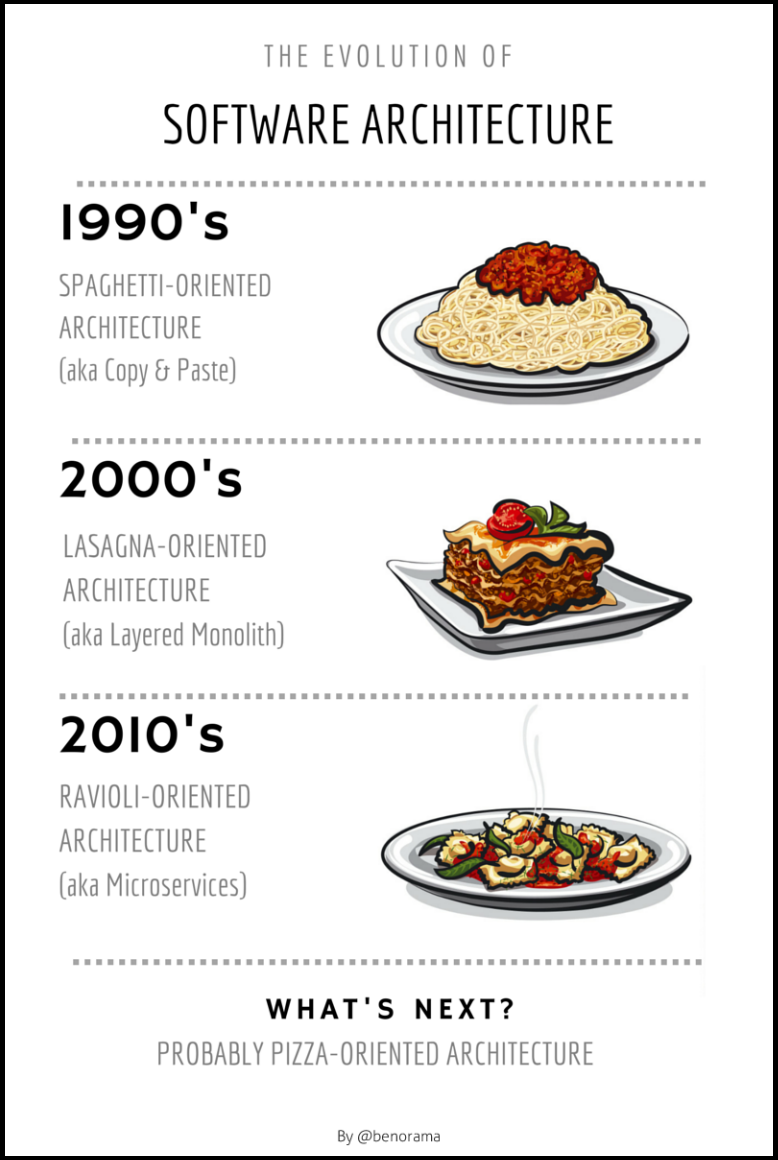
There’s probably a crypto library matching your needs, no need to write your own

Identify your requirements and search for the lib that best matches

Prefer high-level to low-level APIs, reduces the risk of error and the code on your side

Will we move towards crypto microservices?

Multiple high-level libs for specific applications, rather than one low-level lib misused by developers?


THE EVOLUTION OF
SOFTWARE ARCHITECTURE
1990's
SPAGHETTI-ORIENTED ARCHITECTURE (aka Copy & Paste)
2000's
LASAGNA-ORIENTED ARCHITECTURE (aka Layered Monolith)
2010's
RAVIOLI-ORIENTED ARCHITECTURE (aka Microservices)
WHAT'S NEXT?
PROBABLY PIZZA-ORIENTED ARCHITECTURE
By @benorama

Merci!

List of crypto libs: http://tinyurl.com/cryptolibs