We ❤ SSL

Emilia Käsper
OpenSSL / Google
Let’s start with a guessing game...

What is this graph about?
Myth: Heartbleed broke the Internet
Fact: Internet-breaking bugs are common

- CVE-2011-0014 - infoleak, true impact unknown
- CVE-2012-2110 - possibly arbitrary code execution on reading certificates
- CVE-2012-2333 - buffer over-read, true impact unknown
- CVE-2014-1266 - “goto fail” server spoofing (Apple)
- CVE-2014-0160 - Heartbleed
- CVE-2014-0224 - “early CCS” disables encryption
- CVE-2014-1568 - RSA signature forgery (NSS)
In this talk...

● A history of OpenSSL: the good, the bad and the ugly
● Heartbleed in the sea of exploits: why the hype, and what can we learn from this?
● The future of OpenSSL: what we’re doing, and how you can help.
Heartbleed - why the attention?

NIST (NVD) vulnerability scores

- Impact
- Exploitability
- Score

- goto fail
- Heartbleed
- Early CCS
- RSA signature forgery
Heartbleed - why the attention?

- Branding => press coverage, pop culture
- Changed awareness: Snowden
- Simplicity of exploit
- Remote code executions aren’t concrete enough
- Offensive institutions are much better at judging bug impact. Recall…
  - CVE-2011-0014 - infoleak, true impact unknown
  - CVE-2012-2333 - buffer over-read, true impact unknown
Lesson #1: we need code review
Add support for TLS/DTLS heartbeats.

## Description
Add support for TLS/DTLS heartbeats.

## Patch Set 1

<table>
<thead>
<tr>
<th>Unified diffs</th>
<th>Side-by-side diffs</th>
<th>Delta from patch set</th>
<th>Stats (+650 lines, -19 lines)</th>
</tr>
</thead>
<tbody>
<tr>
<td>M CHANGES</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M apps/s_cb.c</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M apps/s_client.c</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M apps/s_server.c</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M crypto/objects/obj_det.h</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M crypto/objects/obj_mac.h</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M crypto/objects/obj_mac.num</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M crypto/objects/objects.bt</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M crypto/rsa/rsa_pmeth.c</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M ssl/d1_both.c</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M ssl/d1_clnt.c</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M ssl/d1_lib.c</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M ssl/d1_pkt.c</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M ssl/d1_srvr.c</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M ssl/dlssl1.h</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M ssl/s3_clnt.c</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M ssl/s3_lib.c</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M ssl/s3_pkt.c</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M ssl/s3_srvr.c</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M ssl/ssl.h</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M ssl/ssl3.h</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M ssl/ssl_err.c</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M ssl/ssl_loc1.h</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M ssl/t1_lib.c</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M ssl/tls1.h</td>
<td>View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M util/mkdef.pl</td>
<td>View</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
#ifndef OPENSSL_NO_HEARTBEATS
int tls1_process_heartbeat(SSL *s)
{
    unsigned char *p = s->s3->rrec.data[0], *pl;
    unsigned short htype;
    unsigned int payload;
    unsigned int padding = 16; /* Use minimum padding */

    /* Read type and payload length first */
    htype = *p++;
    n2s(p, payload);
    pl = p;

    if (s->msg_callback)
        s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
                       &s->s3->rrec.data[0], s->s3->rrec.length,
                       s, s->msg_callback_arg);

    if (htype == TLS1_HB_REQUEST)
        {
            unsigned char *buffer, *bp;
            int r;

            /* Allocate memory for the response, size is 1 bytes
             * message type, plus 2 bytes payload length, plus
             * payload, plus padding */
            buffer = OPENSSL_malloc(1 + 2 + payload + padding);
            bp = buffer;

            /* Enter response type, length and copy payload */
            *bp++ = TLS1_HB_RESPONSE;
            s2n(payload, bp);
            memcpy(bp, pl, payload);

            r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, 3 + payload + padding);

            OPENSSL_free(buffer);
        }

    return r;
}
#endif
Lesson #2: review != audit

- Code reviewers are not trained to find complex bugs.
- Few people are paid to audit critical codebases defensively.
- Fewer people are paid to turn vulnerabilities into exploits defensively.
- Offensive industry will routinely do this => huge edge in finding full exploit chains.
- You get what you pay for => we need to fix this are fixing this.
Changes in the OpenSSL team

● Expanded development team (3 FTE* + 12 volunteers)
● Mandatory code reviews
● New security policy
● New release strategy
● New blog :)  

*https://www.openssl.org/support/acknowledgments.html
New OpenSSL release today!

- Security updates for 1.0.1/1.0.0./0.9.8
- Fixing 8 security vulnerabilities
- We get a lot of reports from academia & industry
- 5th security release since Heartbleed - this is a good thing!
How can the community help?

- Formal verification of crypto code
  - Hitting $< 2^{-64}$ corner cases with unit testing is difficult.
  - New-ish elliptic curve implementations: P-224, P-256, P-521 - fast and constant-time. But are they correct?
  - Regression testing (again!) for bug attacks and oracle attacks.
How can the community help?

- State machine analysis
  - Very old code, not written with adversarial behaviour in mind
  - Individual reports from different research groups…
  - ... => continuous regression testing?
How can the community help?

- Record/message/ASN.1 object layer fuzzing
  - Some open-source tools already available to help:
    - American Fuzzy Lop
    - Frankencert
- Smarter tools for finding/building exploits
How can the community help?

- Constant-time crypto
  - AES, RSA, P-256 quite well covered across platforms
  - But how about a library for implementing common operations \((x = \text{condition} ? a : b)\)?
  - … or a constant-time code generator for field operations?
  - Authenticated encryption is brittle \(\Rightarrow\) need new primitives.
Questions?

The OpenSSL development team:

Come talk to us!