THE HEARTBLEED BUG AND ITS BUDDIES:
Where they come from and how to get rid of them

RAULI KAKSONEN, CTO
Codenomicon Ltd.
Heartbleed bug
http://heartbleed.com
OpenSSL

• The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade, full-featured, and Open Source toolkit... The project is managed by a worldwide community of volunteers that use the Internet to communicate, plan, and develop the OpenSSL toolkit and its related documentation.

https://www.openssl.org/

• Widely used, believed to be reliable and efficient
#RUNFORTHEHILLS

Computer Bug 'Heartbleed' Could Allow Hackers To Devastate Internet
THE HEARTBLEED BUG

• Programming mistake which caused OpenSSL to leak memory into response messages
  • Fast to harvest memory
  • Hard to detect
  • Easy to collect a lot of memory
• As a result, OpenSSL leaks all secrets it has
  • Including private keys, which is a bad thing

• So, what happened..?
TLS/SSL PROTOCOL

Client

- Client Hello
- Client Certificate *
- Client Key Exchange
- Certificate Verify *
- Change Cipher Spec
- Client Finished

Server

- Server Hello
- Server Certificate *
- Server Key Exchange *
- Certificate Request *
- Server Hello Done
- Change Cipher Spec
- Server Finished

- Heartbeat Request *
- Heartbeat Response *
- Close Notify (Alert)
- Close Notify (Alert)

Application data

© 2014 All Rights Reserved
The Heartbeat protocol messages consist of their type and an arbitrary payload and padding.

```
struct {
    HeartbeatMessageType type;
    uint16 payload_length;
    opaque payload[HeartbeatMessage.payload_length];
    opaque padding[padding_length];
} HeartbeatMessage;
```

When a HeartbeatRequest message is received and sending a HeartbeatResponse is not prohibited as described elsewhere in this document, the receiver MUST send a corresponding HeartbeatResponse message carrying an exact copy of the payload of the received HeartbeatRequest.
HEARTBEAT EXCHANGE

1 7 ‘Payload’

2 7 ‘Payload’

1 16 ‘A-longer-payload’

2 16 ‘A-longer-payload’
HEARTBEAT ADDED TO OpenSSL

commit 4817504d069b4c5082161b02a22116ad75f822b1
Author: xxxx
Date: Sat Dec 31 22:59:57 2011 +0000

PR: 2658
Submitted by: xxx
Reviewed by: xxx

Support for TLS/DTLS heartbeats.
unsigned short hbtype;
unsigned int payload;
unsigned int padding = 16; /* Use minimum padding */

/* Read type and payload length first */
hbtype = *p++;
unsigned int payload = n2s(p, payload);
...
if (hbtype == TLS1_HB_REQUEST)
{
    unsigned char *buffer, *bp;
    int r;

    /* Allocate memory for the response, size is 1 byte 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);

    /* Random padding */
    RAND_pseudo_bytes(p, padding);

    r = dtls1_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, 3 + payload + padding);

Payload length is read and trusted without validation
“HEARTBLEED EXCHANGE”
WHAT WAS LEAKED?

• Anything from OpenSSL memory
  • Login information, including passwords
  • Other user data
  • Private keys used in cryptography
  • All cleartext

• How to exploit
  • Use a simple script to collect memory of vulnerable server
  • Harvest the data for secrets
LOOKING AT THE LEAK

04dfe90: 0000 0000 0000 0000 0000 0000 0000 0000
04dfeaa: 0000 0000 0000 0000 0000 0000 0000 0000
04dfeb0: 0000 0000 0000 0000 0000 0000 0000 0000
04dfec0: 0000 0000 0000 0000 0000 0000 0000 0000
04dfed0: 0000 0000 0000 0000 0000 0000 0000 0000
04dfee0: 0000 0000 0000 0000 0000 0000 0000 0000
04dfef0: 0000 0000 0000 0000 0000 0000 0000 0000
04dfff0: 0000 0000 0000 0000 0000 0000 0000 0000
04dfff10: 0000 0000 0000 0000 36a0 0414 3af5 3cf9 b05e
04dfff20: dbc4 844a f7d7 0992 acc7 e300 0000 006e
04dfff30: 8093 a700 2e07 ac65 3aed d64a 10b2 737e
04dfff40: ff22 c180 f589 e78b 8018 fba8 752a 2f2a
04dfff50: 0d0a 4175 7468 6f72 697a 6174 696f 6e3a
04dfff60: 2042 6173 6963 2062 576c 7261 3238 3662
04dfff70: 576c 7261 3238 3d0d 0a48 6f73 743a 2031
04dfff80: 3732 2e31 362e 3536 2e31 3033 0d0a 436f
04dfff90: 6e6e 6563 7469 6174 6565 6570 2d41
04dffffff0: 6c69 7665 0d0a 0d0a 4049 8d2a 6558 ce6e
04dff0: 06bd 8043 c560 9789 0005 0004 0015 0012
04dff10: 0009 0014 0011 0008 0006 0003 00ff 0021
04dff20: 0000 6d00 0b00 0403 0001 0200 0a00 3400

...6...<...^...J............n...e:J...s'".."A*/*

Authorization: Basic bWlra286b
Wlra28=,:Host: 172.16.56.101,:Co
nection: Keep-A
live...@I.*e...
...C:\...
...m...........4.

© 2014 All Rights Reserved
HOW WAS HEARTBLEED FOUND?

• Codenomicon
  • We were conducting some new security tests, using advanced fuzzing
  • Noticed a discrepancy between the heartbeat request size and the response size
  • Reported this to CERT-FI

• Google (Neel Mehta)
  • Apparently noticed the bug in code review
YOU SAID “FUZZING”?
WHO PUBLISHED THE VULNERABILITY?

• The OpenSSL team provided a fix with information about the vulnerability 2014-04-07
  • After they learned that there were two independent vulnerability discoveries
• A few companies were notified earlier
MITIGATION OF HEARTBLEED

• Fix the bug
  • Recompile with heartbeat disabled
  • Or fix the code, not difficult once you realize the problem
  • Or upgrade to a patched version of OpenSSL

• Change certificate & private key
  • Only changing the certificate is not enough
  • Revoke old certificate

• Change passwords

• Any other secrets through TLS/SSL tunnel – change!
ATTACKS USING HEARTBLEED

• Estimated 24-55% of all web servers were vulnerable at time of the public announcement
• Scans for vulnerable servers started within first 24 hours
• Most popular servers were patched during first 48 hours

VULNERABLE DEVICES

• Not just Web servers...
• Network scans identified vulnerable device/SW package categories:
  • Communication services
  • Software control panels
  • Network attached storage
  • Firewall and VPN devices
  • Printers
  • Other

HEARTBLEED TIMELINE

- Bug committed 2011-12-31
- Google discovery 2014-03-21
- Google notifies OpenSSL 2014-04-01
- Codenomicon discovery, reports to CERT-FI 2014-04-03
- OpenSSL fix and advisory 2014-04-07
- First confirmed scans to find vulnerable devices 2014-04-08
HEARTBLEED FOLLOW-UP

• News, news, and news...
• Patching...
• Unseen amount of public lessons-learned discussion
  • Many government-sponsored Heartbleed related event and seminars

Best Server-Side Bug 2014
StrongSwan bug
CVE-2012-2388
strongSWAN

• OpenSource IPsec-based VPN Solution

https://www.strongswan.org/
POSESSION OF PRIVATE KEY PROVES IDENTITY

1. Certificate with public key

2. Check certificate using CA

3. Signature by private key: ”A4F30ADC”

4. Check signature, use the public key

5. Access ok
THE strongSWAN BUG

“If the gmp plugin is used for RSA signature verification (the default on many platforms) an empty or zeroed signature is handled as a legitimate one.”

• This means that user is not properly authenticated
  • Present somebody else’s certificate (which is public)
  • Provide an invalid signature
  • strongSwan accepts the certificate as yours
THE BUG VISUALLY

1. Certificate with public key

2. Check certificate using CA

3. Signature by private key: "0"

   Invalid value

4. Check signature, use the public key

   Oops...

5. Access ok

   © 2014 All Rights Reserved
/**
 * Verification of an EMPSA PKCS1 signature
 */
static bool verify_emsa_pkcs1_signature(private_gmp_rsa_public_key_t *this,
                                        hash_algorithm_t algorithm,
                                        chunk_t data, chunk_t signature)
{
    ... 
    bool success = FALSE;
    ...

    /* remove any preceding 0-bytes from signature */
    while (signature.len && *(signature.ptr) == 0x00)
    {
        signature = chunk_skip(signature, 1);
    }

    if (signature.len == 0 || signature.len > this->k)
    {
        return INVALID_ARG;
    }
}
SOME OTHER SECURITY BUGS

- OpenSSL (CVE-2014-5139, -3512, -3511)
- Polar TLS (CVE-2014-4911)
- Bind (CVE-2014-3859)
- GNU TLS (CVE-2014-3466)
- SCTP Linux Kernel (CVE-2014-0101)

- **XML libraries** in 2009
- **ASN.1 / SNMP** (CVE-2002-0013)
Squishing the bugs
KNOWN AND UNKNOWN BUGS

• Unknown vulnerabilities
  • Dormant bugs

• Zero-day vulnerability
  • Unknown vulnerability unveiled
  • No fix available

• Known vulnerabilities
  • Publicly known issues
  • Fix or mitigation is available

→ Testing, resilient coding

→ Incident response

→ Scan and mitigate, PR
SOFTWARE LIFECYCLE

Design  Programming

Specification  Integration

Maintenance  Testing

Deployment
TOP 10 SOFTWARE SECURITY DESIGN FLAWS

1. Validate data from untrusted source
2. Use a proper authentication mechanism
3. Authorize after you authenticate
4. Separate data and control, never take controls from an untrusted source
5. Ensure that all data is explicitly validated
6. Use cryptography correctly
7. Identify and handle sensitive data
8. Always consider the users
9. Understand the impact of external components
10. Be flexible for future changes

Adapted from http://cybersecurity.ieee.org
# TOP 25 MOST DANGEROUS SOFTWARE ERRORS

<table>
<thead>
<tr>
<th>Rank</th>
<th>Score</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>93.8</td>
<td>CWE-89</td>
<td>Improper Neutralization of Special Elements used In an SQL Command ('SQL Injection')</td>
</tr>
<tr>
<td>[2]</td>
<td>83.3</td>
<td>CWE-78</td>
<td>Improper Neutralization of Special Elements used In an OS Command ('OS Command Injection')</td>
</tr>
<tr>
<td>[3]</td>
<td>79.0</td>
<td>CWE-120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
</tr>
<tr>
<td>[4]</td>
<td>77.7</td>
<td>CWE-79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
</tr>
<tr>
<td>[6]</td>
<td>76.8</td>
<td>CWE-862</td>
<td>Missing Authorization</td>
</tr>
<tr>
<td>[7]</td>
<td>75.0</td>
<td>CWE-798</td>
<td>Use of Hard-coded Credentials</td>
</tr>
<tr>
<td>[8]</td>
<td>75.0</td>
<td>CWE-311</td>
<td>Missing Encryption of Sensitive Data</td>
</tr>
<tr>
<td>[9]</td>
<td>74.0</td>
<td>CWE-434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>[10]</td>
<td>73.8</td>
<td>CWE-807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
</tr>
<tr>
<td>[11]</td>
<td>73.1</td>
<td>CWE-250</td>
<td>Execution with Unnecessary Privileges</td>
</tr>
<tr>
<td>[12]</td>
<td>70.1</td>
<td>CWE-352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>[13]</td>
<td>69.3</td>
<td>CWE-22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
</tr>
<tr>
<td>[14]</td>
<td>68.5</td>
<td>CWE-494</td>
<td>Download of Code Without Integrity Check</td>
</tr>
<tr>
<td>[16]</td>
<td>66.0</td>
<td>CWE-829</td>
<td>Inclusion of Functionality from Untrusted Control Sphere</td>
</tr>
<tr>
<td>[17]</td>
<td>65.5</td>
<td>CWE-732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
</tr>
<tr>
<td>[18]</td>
<td>64.6</td>
<td>CWE-676</td>
<td>Use of Potentially Dangerous Function</td>
</tr>
<tr>
<td>[19]</td>
<td>64.1</td>
<td>CWE-327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
</tr>
<tr>
<td>[20]</td>
<td>62.4</td>
<td>CWE-131</td>
<td>Incorrect Calculation of Buffer Size</td>
</tr>
<tr>
<td>[21]</td>
<td>61.5</td>
<td>CWE-307</td>
<td>Improper Restriction of Excessive Authentication Attempts</td>
</tr>
<tr>
<td>[22]</td>
<td>61.1</td>
<td>CWE-601</td>
<td>URL Redirection to Untrusted Site ('Open Redirect')</td>
</tr>
<tr>
<td>[23]</td>
<td>61.0</td>
<td>CWE-134</td>
<td>Uncontrolled Format String</td>
</tr>
<tr>
<td>[24]</td>
<td>60.3</td>
<td>CWE-190</td>
<td>Integer Overflow or Wraparound</td>
</tr>
<tr>
<td>[25]</td>
<td>59.9</td>
<td>CWE-759</td>
<td>Use of a One-Way Hash without a Salt</td>
</tr>
</tbody>
</table>

Source: [http://cwe.mitre.org](http://cwe.mitre.org)
SECURITY REQUIREMENTS

• Identify sensitive data and state its criticality
• Design security requirements for end users
• Risk management
• Plan for disaster recovery, including patching
  • I believe this is quite different in ICS compared to usual IT
SECURE DESIGN

• Identify data flows from external sources, validate
• Use cryptography properly
• Caveat emperor
  • Complexity is the enemy of security
  • The more components there are, the more bugs there are
SOFTWARE ABSTRACTION AND COMPLEXITY

Vulnerabilities...

...from programming

...from external components

Abstraction, complexity
SECURE PROGRAMMING

• Programmer training
• Some programming languages are more susceptible to vulnerabilities than others
• Source code scanning
  • Enforces good programming practises
  • Helps to mitigate impact of programming mistakes
INTEGRATION

• Modern software is built from internal and external components
• Outsourcing development
  • Quality systems traditionally measure the process, not the product
  • Flaws are usually not visible from the process
• Security of external components
  • Quality may be unknown
  • Open source vs. closed source
  • Who is responsible for updates
• Manage components and risks
SECURITY TESTING

• Traditionally, testing...
  • Tends to be positive
  • Checks that enumerated requirements are met

• Fuzzing
  • Generates a large number of test cases automatically
  • Tests for invalid and/or unexpected cases
  • Effective in locating security flaws

• Both Heartbleed and strongSWAN were found by fuzzing with advanced logic checks
DEPLOYMENT

- Authentication/access control
- VPN
- Firewalls
  - Partition the network
  - Limit the attack surface internally and externally
- IDS/IPS
MAINTENANCE

• Configuration options
  • Passwords – the static, the default and the bad
  • Bad default configuration

• Network scanning
  • Know you network
  • Detect vulnerable network services

• Penetration testing

• Patching
INCIDENT RESPONSE

• Mistakes happen, even the best software made using the best process can have bugs

• Incident response
  1. Accept bug reports
  2. Estimate their criticality and impact
  3. Inform customers ()
  4. Create the fix
  5. Test and validate the fix (ICS!)
  6. Distribute the fix
  7. Inform customers
CONCLUSION

• Heartbleed was a contradiction
  • A small bug, simple to understand
  • Just one of many others
  • But the impact is huge

• Lessons to learn
  • Software contains vulnerabilities
    • Unknown vulnerabilities
    • Known vulnerabilities
  • Prepare to mitigate
HEARTBLEED HOMEWORK

• Have you identified all client-side code that uses OpenSSL?
• Have you checked if your VPN or mesh networks use OpenSSL?
• Have you checked if any of your connected devices are using OpenSSL?
• Have you checked if your back-end systems are using OpenSSL?
• Have you checked if any of your embedded systems are using OpenSSL (many embedded OS use OpenSSL)?
• Do you have browser based customer interfaces for situation awareness, billing or reporting over SSL/TLS?
• Have you checked third-party binaries/firmware for the existence of OpenSSL?
• Are you an early adopter of DTLS? (OpenSSL DTLS was equally flawed)
• When you go and check in practice, be a bit careful: Some embedded devices with SSL/TLS are rumored to crash with the Heartbleed probe

http://heartbleed.com/
http://appcheck.codenomicon.com/
Rauli Kaksonen
CTO
rauli.kaksonen@codenomicon.com