hb.c 리컴파일 방법 : $ gcc -L/home/sec/12345/openssl/lib -I/home/sec/12345/openssl/include

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

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

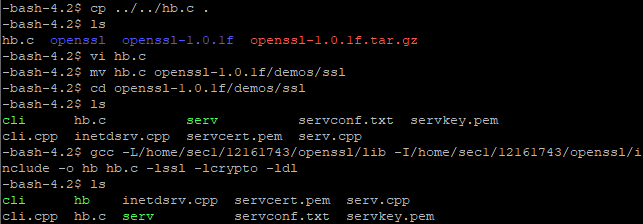
Run server and hb.

$ ./serv

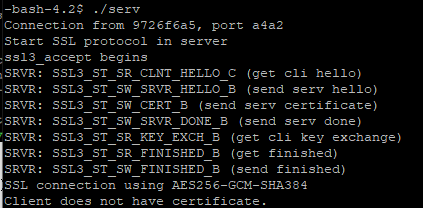
In another window (use your SSL server's port number)

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

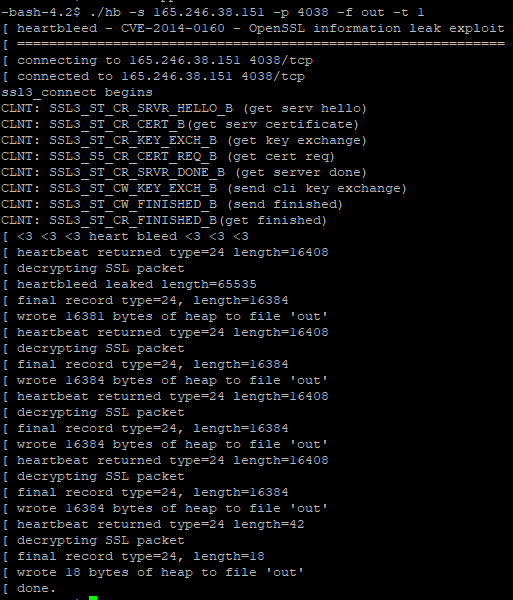
The result should be in file "out". See "out" with xxd and find the server certificate information.

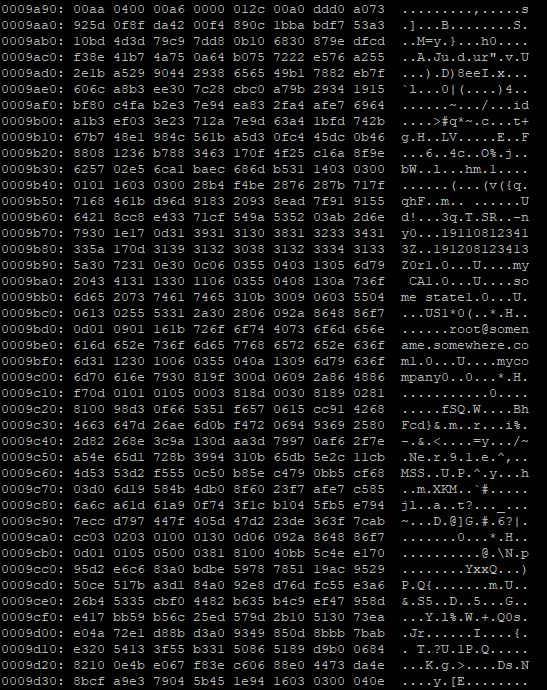


serv



hb



certificate information  


openssl-1.0.1f/demos/ssl에 out 파일 있음. xxd out > l 한다음 l 파일을 보면 됨.

서버를 실행시키고 hb를 타입 1로 실행시킨 뒤 out 파일을 확인해 봤을 때, certificate information 정보를 그대로 확인할 수 있었다.

2) Modify hb.c

void \* heartbleed(...){

........

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

...........

switch(type){

..........

}

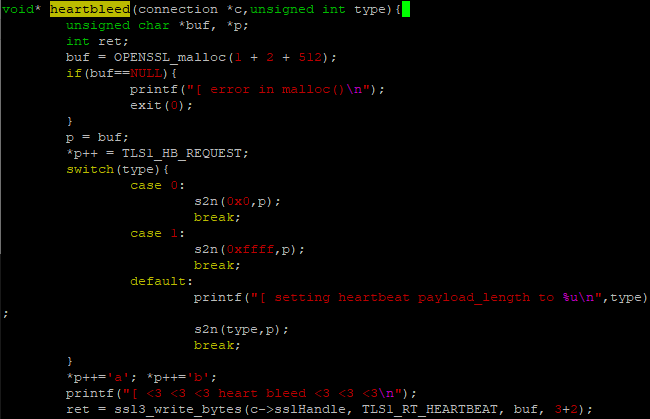
\*p++='a'; \*p++='b'; // 2 byte payload ("ab")

....

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

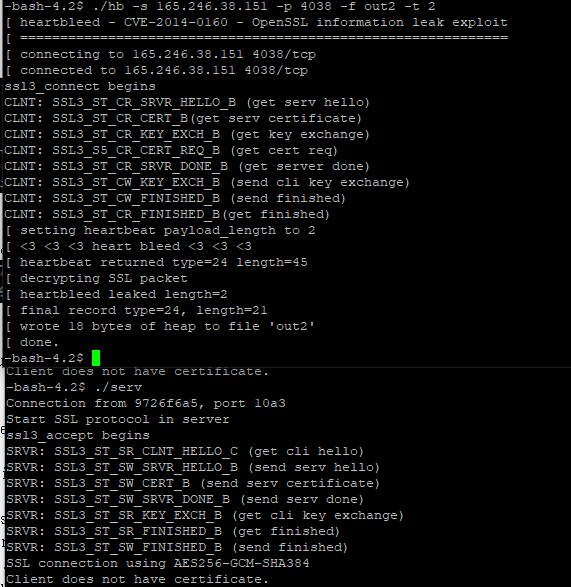
.......

}

Recompile hb.c and run with type 2

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

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

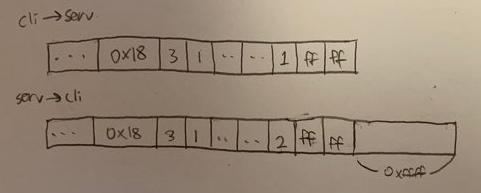




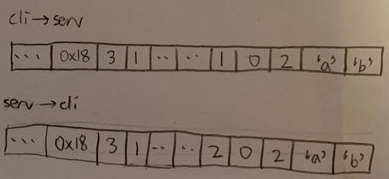
적당히 512만큼의 공간을 준뒤 payload로 ab를 넣어주고 ab가 출력되도록 buf에 3+2를 넣어주었다. serv와 hb를 실행시키고 아웃풋 파일을 확인했을 때 ab가 그대로 파일에 들어있는걸 알 수 있었다.

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

hw1



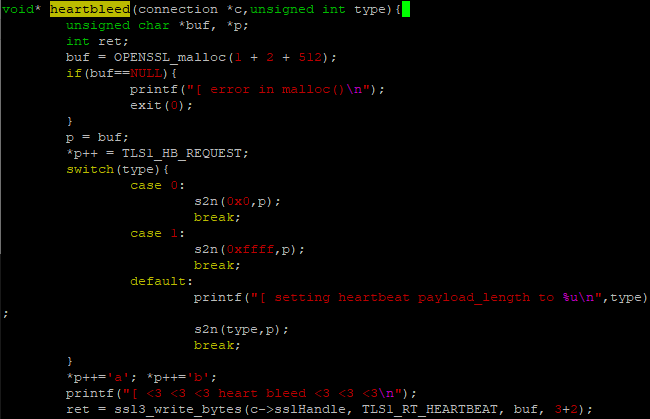
hw2



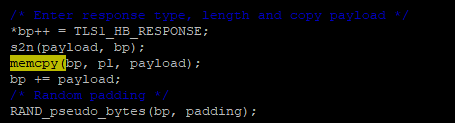
hw1에서는 서버가 클라이언트에게서 받은 payload 길이와 실제 payload 길이가 같은지 확인하지 않기 때문에 만약 클라이언트로부터 받은 payload 길이가 0xffff라면 msglen도 0xffff로 지정하고 그만큼의 공간에 서버에 원래 있던 정보들로 채워서 클라이언트에게 다시 보내게 된다. hw2에서는 payload의 길이도 2바이트이고 실제 payload도 2바이트여서 클라이언트가 보내온 패킷을 그대로 다시 클라이언트에게 보낼 수 있게 된다.

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

hb.c/heartbleed()



tls1\_process\_heartbeat()



hw1에서는 타입이 1이기 때문에 heartbleed 함수에서 case 1에 해당하게 된다. SSL 서버가 패킷을 받으면 그것을 s->s3->rrec->data[]에 저장하게 된다. data[0]에는 TLS1\_HB\_REQUEST, 즉 1이 있고, data[1]과 data[2]에는 0xffff가 있다. 그러고나서 tls1\_process\_heartbeat 함수를 부르고, tls1\_process\_heartbeat 함수는 payload 데이터를 가리키고 있는 pl에서 payload의 길이인 0xffff만큼을 bp에 복사시킨다. 그렇기 때문에 결과파일을 보면 서버의 certificate 정보까지 다 들어있는 것이다. 반면 hw2는 타입이 2이기 때문에 default 케이스에 속하게 된다. 이 때는 payload의 길이를 2로 세팅해주고 또한 실제 payload의 길이도 2이기 때문에 serv와 hb를 실행시키고 파일을 확인해보면 ab 외에 서버의 다른 추가적인 정보는 포함되지 않는단걸 알 수 있다.

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

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" {

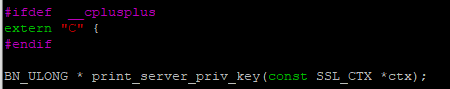
#endif

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;

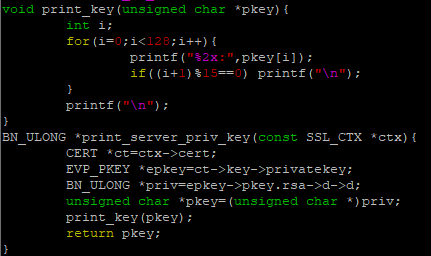
BN\_ULONG \*priv=epkey->pkey.rsa->d->d;

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

print\_key(pkey);

return pkey;

}



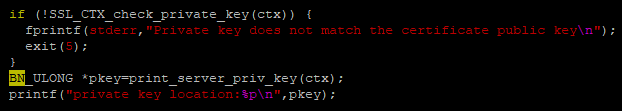
서버의 private key 정보를 출력해내도록 코딩해주는데 이를 위해 라이브러리를 고치게 됐기 때문에 make와 make install을 해주었다.

4.3) Call this function in serv.cpp.

.............after SSL\_CTX\_check\_private\_key(ctx)){...}

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

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

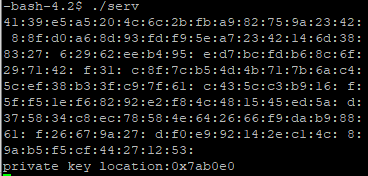


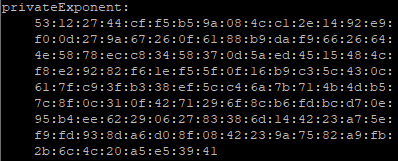
serv.cpp에 private key의 정보를 출력해내도록 하고 그 위치도 출력하도록 만들어줬다.

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





출력된 private key의 정보는 실제 private key 정보와 똑같았다. 서버의 private key의 메모리 주소는 0x7ab0e0이라는걸 알 수 있다.

5) Perform Heartbleed attack 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

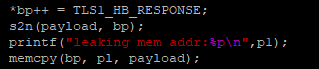
.............

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

memcpy(bp, pl, payload);

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

}



라이브러리에서 leaked memory의 주소를 알 수 있도록 printf절을 하나 추가해주고 make와 make install을 해주었다.

5.2) Run server and hb.

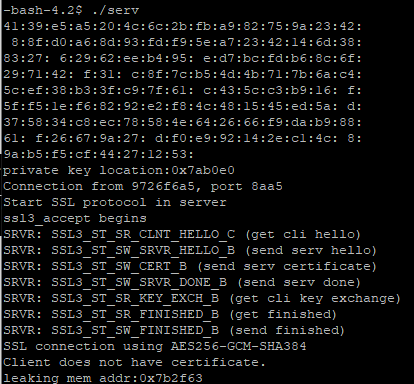
$ ./serv

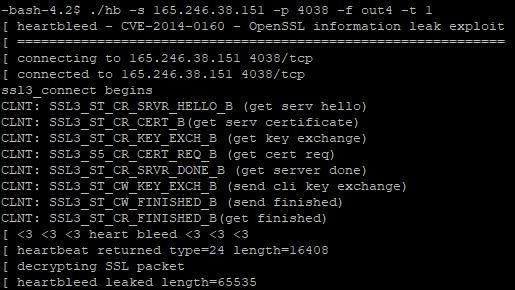
$ ./hb –s 165.246.38.151 –p 12345 –f out –t 1

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.





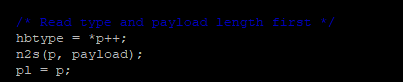




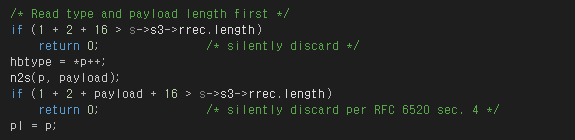
serv와 hb도 다시 컴파일 해주고 실행시켰을 때, leaking memory의 주소는 0x7b2f63이고 private key의 위치는 0x7ab0e0라는걸 확인할 수 있었다. 둘중에 leaking address가 더 크기 때문에 아웃풋 파일에는 private key의 정보가 포함되지 않는단걸 알 수 있다.

6) How can you fix SSL to prevent Heartbleed attack? (밥먹고 여기부터 다시)

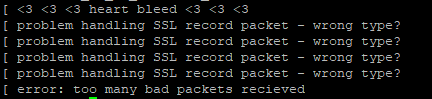
original :



openssl-1.0.1f/ssl/t1\_lib.c/tls1\_heartbeat\_process



최신 버전의 SSL의 tls1\_heartbeat\_process 함수에서는 위와 같은 코드가 추가됐다. 만약 패킷의 전체 길이가 payload의 길이를 합친 1+2+payload+16보다 작으면 payload가 잘못됐다는 뜻이기 때문에 0를 리턴하도록 코딩한 것 같다.



실제로 코드를 위와 같이 고쳐보고 실행시켜본 결과, heartbleed attack에 성공하지 못한걸 확인할 수 있었다.

(라이브러리 고친 후 항상 serv랑 hb 둘다 재컴파일 해줘야함