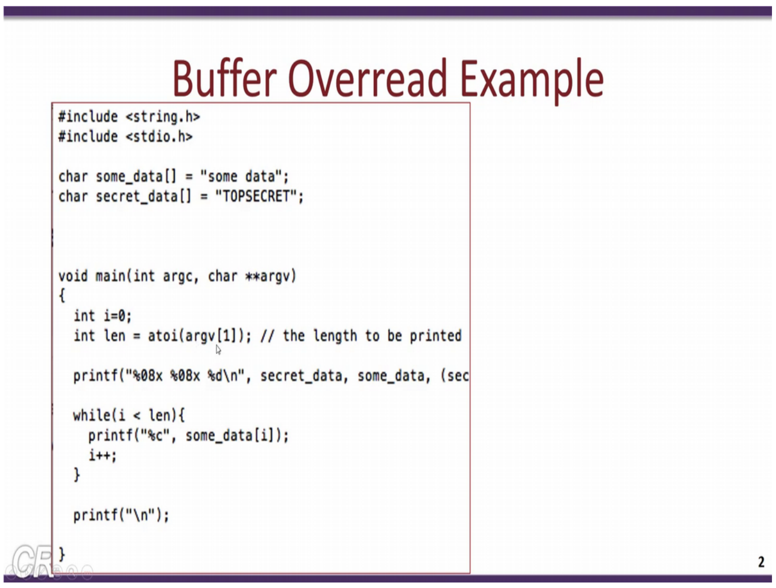
Information Security 5 Secure System Engineering
Professor Chester Rebeiro
Indian Institute of Technology Madras
Buffer Overreads
Mod03_Lec17

Hello and welcome to this lecture in the course for secure system engineering, so in the previous lectures we had actually looked at one particular vulnerability in programs specially in CC and C++ programs that vulnerability was caused due to buffer overflows in the stack, so what we seen was that an attacker could use this buffer overflows to inject code into a program and then forced that particular code to execute, so we called this as the subverting of the execution and then the execution of the payload and then in the previous lectures we had also seen a couple of phase where this buffer overflow type vulnerabilities could be invented.

Essentially we discussed a couple of countermeasures for the buffer overflow vulnerability, introduce lecture we will look at an other vulnerability based on buffers, here we are going to discuss about buffer overreads, so let start with a small example of what buffer overreads is.

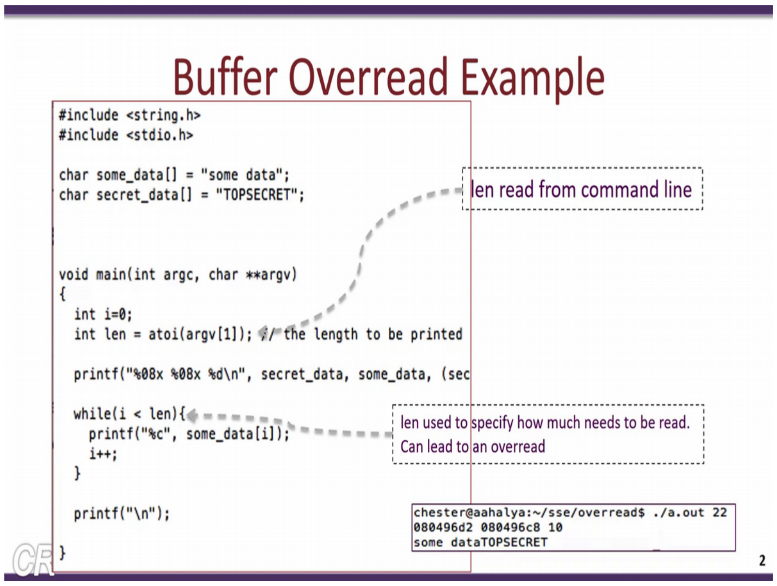
(Refer Slide Time: 1:26)



Buffer Overread Example

```
#include <string.h>
#include <stdio.h>

char some_data[] = "some data";
char secret_data[] = "TOPSECRET";



void main(int argc, char **argv)
{
  int i=0;
  int len = atoi(argv[1]); // the length to be printed

  printf("%08x %08x %d\n", secret_data, some_data, (sec

  while(i < len){
    printf("%c", some_data[i]);
    i++;
  }

  printf("\n");

}
```

2



Buffer Overread Example

```
#include <string.h>
#include <stdio.h>

char some_data[] = "some data";
char secret_data[] = "TOPSECRET";



void main(int argc, char **argv)
{
  int i=0;
  int len = atoi(argv[1]); // the length to be printed

  printf("%08x %08x %d\n", secret_data, some_data, (sec

  while(i < len){
    printf("%c", some_data[i]);
    i++;
  }

  printf("\n");

}
```

len read from command line

len used to specify how much needs to be read.
Can lead to an overread

```
chester@aahalya:~/sse/overread$ ./a.out 22
080496d2 080496c8 10
some dataTOPSECRET
```

So let us start with this particular example, so this particular program defines two global arrays, one is known as some data which is initialised to some arbitraries string and other array which is secret data which is initialised to topsecret, now in the main function there is length define which essentially obtained from the command line argument, the first command line argument is convert to an integer and it is used to initialise len and then we have this while loop over here which prints characters of some data, the number of characters that get printed depends on len and which in turn is specified by the user of this program in argv 1.

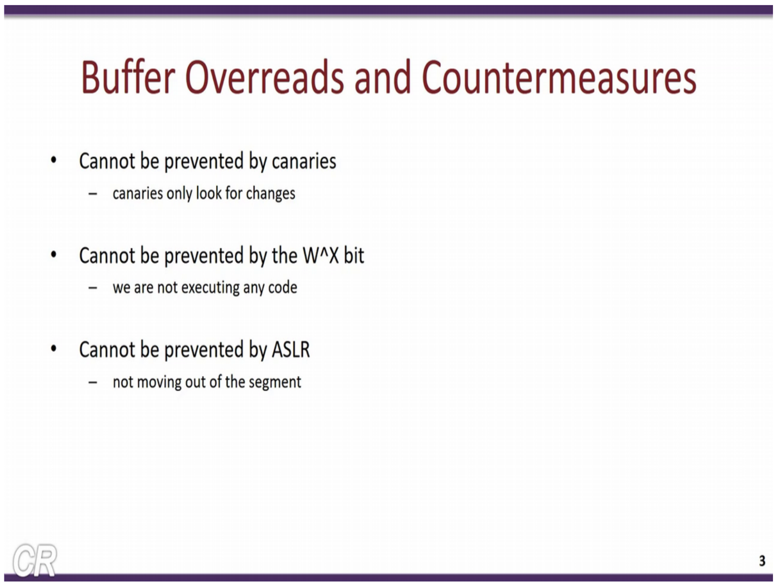
So there are two critical aspects in this particular program in order to understand the vulnerability of this program, firstly the user of this program can specify how many characters of some data he needs to print, if len is less than the size of this particular string then there is no problem and a few characters and at most all the characters of some data would get printed on the screen.

However the user specifies a very large number for len then these some data characters would get printed as well as the adjacent characters stored in the memory would also get printed, in this particular case the adjacent memory contains top-secret, so the value of len is large then not only is some data printed but also top-secret has printed on the.

Therefore we actually run this particular program, we compile it as usual and run it in the command line over here and specify as command line argument as 22, so len gets initialised to 22 and it prints 22 characters starting from some data, so what is get printed on the screen is not just some data but also the adjacent characters top-secret, so this we see is a very simple example of a vulnerability due to buffer overreads.

What has happened here is that the array has been initialised to some specific size but the user has managed to read more data than is required and essentially it is not just the array that gets printed on the screen but memory adjacent to that particular array would also get printed.

(Refer Slide Time: 4:13)



Buffer Overreads and Countermeasures

- Cannot be prevented by canaries
  - canaries only look for changes
- Cannot be prevented by the W^X bit
  - we are not executing any code
- Cannot be prevented by ASLR
  - not moving out of the segment

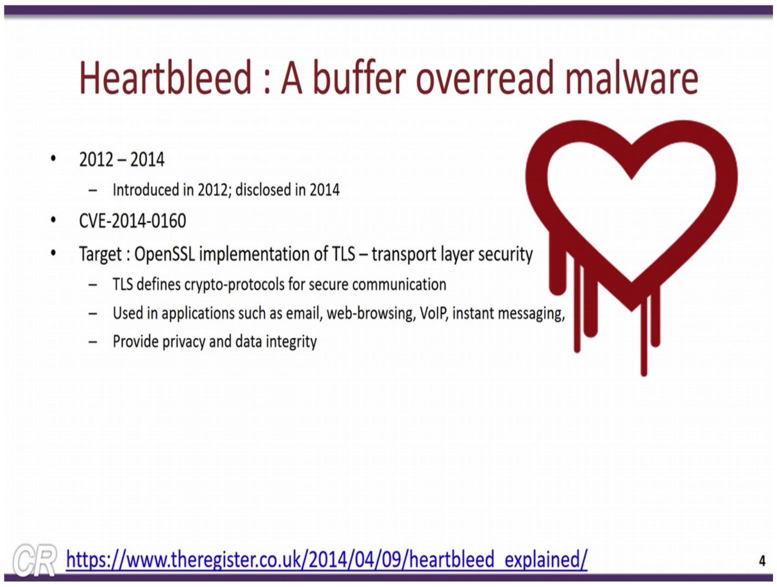
CR 3

So now if we look at the countermeasures that has been presented in the earlier lectures, we had actually studied the user canaries, the W xhor X bit as well as ASLR, I just place a layout randomization, with respect to the buffer overreads what we see over here is that the canaries and the W xhor X bit will not work to prevent buffer overreads, this is because the canaries essentially look for changes in the stack and with the buffer overread we are not writing or changing anything on the stack but we are just reading the contents of the stack.

Similarly with the W xhor X orbit protection mechanism we are not going to execute anything from the stack but rather since just we reading from the stack therefore the W xhor X orbit countermeasure will also not work, now the third countermeasure that we have studied the ASLR or the address space layout randomization will also not help to prevent the buffer overreads.

Essentially the reason is that we are restricting ourselves to reading from the stack or from a given segment of memory, now ASLR was useful to actually randomise the location of libraries, therefore making attack such as the ROP attack more difficult to actually mount, over here on the other hand since we are restricting the overreads to within a particular memory segment therefor the ASLR countermeasure will also not work to prevent the attack.

(Refer Slide Time: 5:49)



Heartbleed : A buffer overread malware

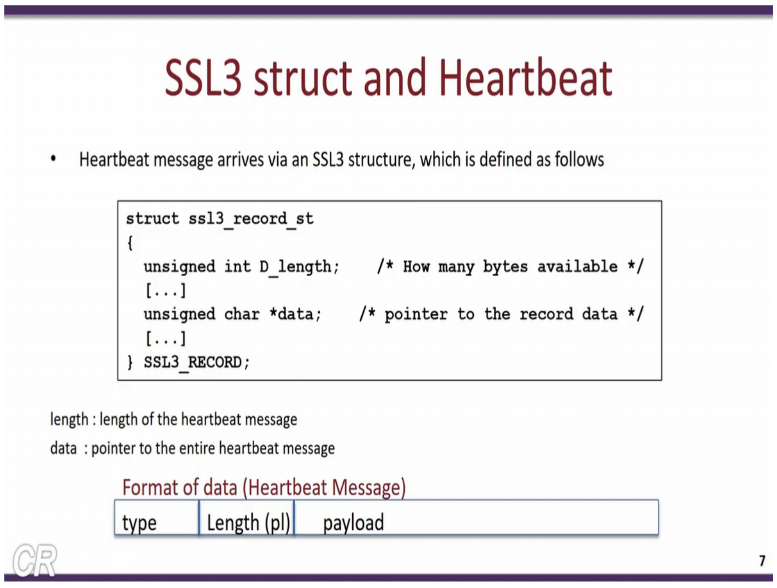
- 2012 – 2014
  - Introduced in 2012; disclosed in 2014
- CVE-2014-0160
- Target : OpenSSL implementation of TLS – transport layer security
  - TLS defines crypto-protocols for secure communication
  - Used in applications such as email, web-browsing, VoIP, instant messaging,
  - Provide privacy and data integrity

https://www.theregister.co.uk/2014/04/09/heartbleed_explained/ 4

Now let us take one particular example of buffer overread attack, so this particular attack is known as heart bleed, so it was a malware that was introduced in 2012 and it was disclosed in 2014, so the CVE for the malware is over here, CVE- 2014 – 0160 and this particular malware essentially used a buffer overread to steal critical information from a web server, so the target was an open SSL implementation of TLS, so TLS is the transport layer security.

Now this particular open SSL library is widely used essentially when you want cryptography or security in your web server applications, so the TLS or the transport layer security defines a crypto protocol for secure communication, it is widely used for applications such as email, web browsing, VoIP and instant messaging, it is used to provide both privacy as well as data integrity, now what we will discuss in this particular lecture is one particular vulnerability in this open SSL implementation which had got exploited by attackers to steal informations.

(Refer Slide Time: 7:10)



SSL3 struct and Heartbeat

- Heartbeat message arrives via an SSL3 structure, which is defined as follows

```
struct ssl3_record_st
{
  unsigned int D_length;    /* How many bytes available */
  [...]
  unsigned char *data;    /* pointer to the record data */
  [...]
} SSL3_RECORD;
```

length : length of the heartbeat message
data : pointer to the entire heartbeat message

Format of data (Heartbeat Message)

type	Length (pl)	payload

7

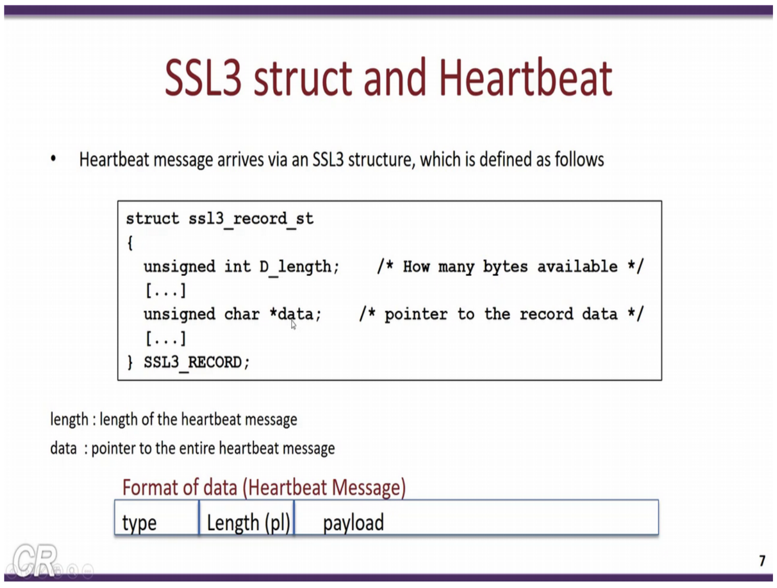
The vulnerability that was exploited in the heartbeat malware was specifically in something known as the heartbeat message that is accompanied of the TLS library, now this heartbeat message is sent between the client and the server and essentially use to keep the connection alive between these two systems, so what is that over here is that one of these systems would create something known as the heartbeat message and send that message to the other system.

Over here for example the client would create the heartbeat message and send that message to the server, now the server would determine that it is indeed the heartbeat message and replicate that particular message that it received and send it back the client, so this is kind of a ping-pong kind of thing, where one system would send a message and the other system would reply with the same message.

Now the heartbeat message look something like this, so it comprised of a 1 byte type over here which essentially specified that this particular message was the heart a beat message, second it had 2 bytes to specify the length of the payload, so therefore since its 2 bytes so the payload be anything from 1 byte ranging to 2 power 16 minus 1 bytes and then finally you have a payload over here and optionally there is extra padding that is also added.

So in our example over here the payload will be hello world, the length would be 12 bytes because hello world comprises of 12 bytes and the type would be as follows, TLS1 HB REQUEST.

(Refer Slide Time: 9:05)



SSL3 struct and Heartbeat

- Heartbeat message arrives via an SSL3 structure, which is defined as follows

```
struct ssl3_record_st
{
  unsigned int D_length;    /* How many bytes available */
  [...]
  unsigned char *data;    /* pointer to the record data */
  [...]
} SSL3_RECORD;
```

length : length of the heartbeat message
data : pointer to the entire heartbeat message

Format of data (Heartbeat Message)

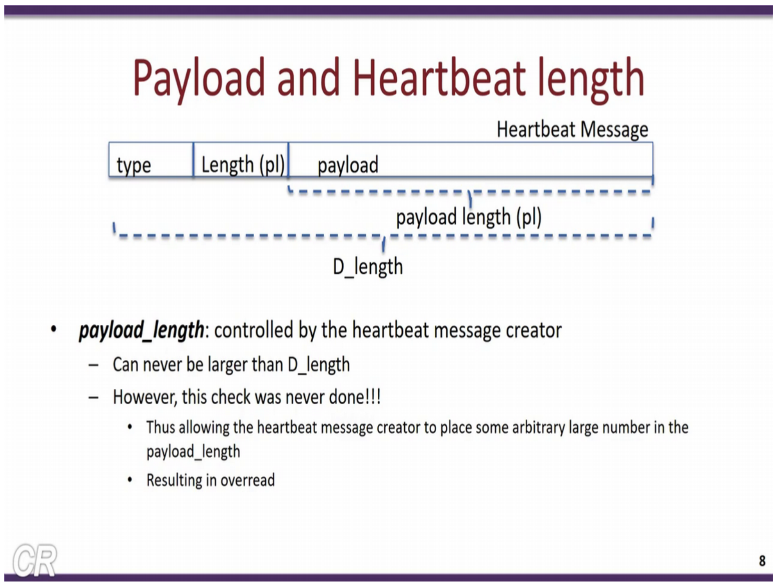
type	Length (pl)	payload

7

Now we also look as the SSL 3 structure which essentially holds the heartbeat message, now SSL 3 structure has a lot of elements but the important ones with respect to the heartbeat message are just two, one is the D length which is defined as unsigned int and the other one is data which is essentially a string, unsigned character pointer, now this data is essentially the heartbeat bit message, which we have discussed earlier comprising of the type, the length of the payload and the payload itself.

Now notice that we have two lengths that are involved in this particular heartbeat message, one is the D length which is present, so this D length is defined in the outer structure and the second one is the load length which is essentially part of the heartbeat message which is part of the data defined over here in the SSL 3 structure.

(Refer Slide Time: 10:07)


Heartbeat Message
type
Length (pl)
payload
payload length (pl)
D_length
CR
8

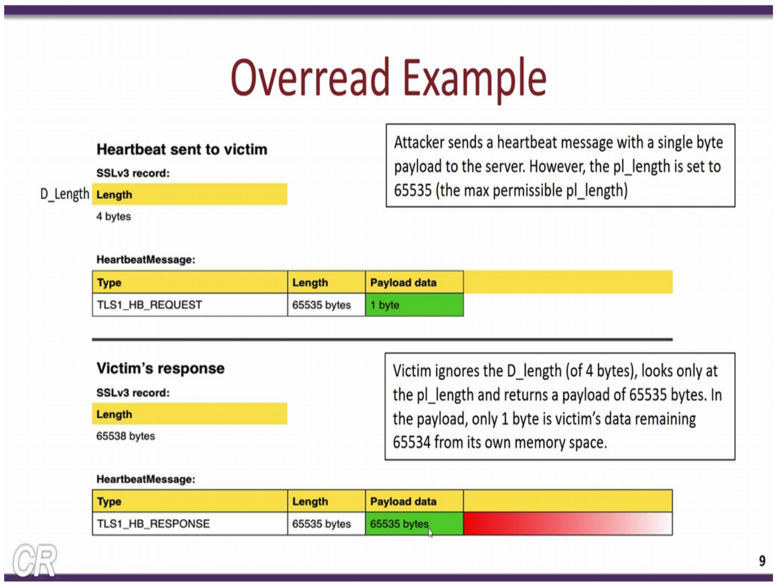
Payload and Heartbeat length

- ***payload_length***: controlled by the heartbeat message creator
  - Can never be larger than D_length
  - However, this check was never done!!!
    - Thus allowing the heartbeat message creator to place some arbitrary large number in the payload_length
    - Resulting in overread

So the D length specifies the entire size of the heartbeat message including the type, length as well as the payload but the length inside the heartbeat message that is the payload and just specifies the length of the payload, now the payload length that is this length is controlled by the creator of the heartbeat message, for example if the client system has creating the heartbeat message then the client system can decide on what is the length of the payload.

Another observation we see over here is that the payload length should be strictly less than the D length, however in the actual code of the in open SSL TLS library this check was never made, so as a result of this what could happen was that the client system which created this heartbeat message could create or set a particular length for the payload which is very large and since this length was never checked it could result to the buffer overread, since the payload length was never checked with respect to the D length it could result in a buffer overread, so let us look at the heart bleed attack with an example.

(Refer Slide Time: 11:25)



Overread Example

Heartbeat sent to victim

SSLv3 record:

D_Length	Length
	4 bytes

Attacker sends a heartbeat message with a single byte payload to the server. However, the pl_length is set to 65535 (the max permissible pl_length)

HeartbeatMessage:

Type	Length	Payload data	
TLS1_HB_REQUEST	65535 bytes	1 byte	

Victim's response

SSLv3 record:

Length
65538 bytes

Victim ignores the D_length (of 4 bytes), looks only at the pl_length and returns a payload of 65535 bytes. In the payload, only 1 byte is victim's data remaining 65534 from its own memory space.

HeartbeatMessage:

Type	Length	Payload data	
TLS1_HB_RESPONSE	65535 bytes	65535 bytes	🟥

9

So we have a client system over here which has created a malicious heartbeat message, so the heartbeat message would look something like this, it would, first is a type which is a 1 byte which specifies the TLS HB REQURST, the second one is the length where maliciously the client system has filled in the maximum length that is, since a length is of 2 bytes, so the maximum length could be 2 power 16 minus 1 which is 65535.

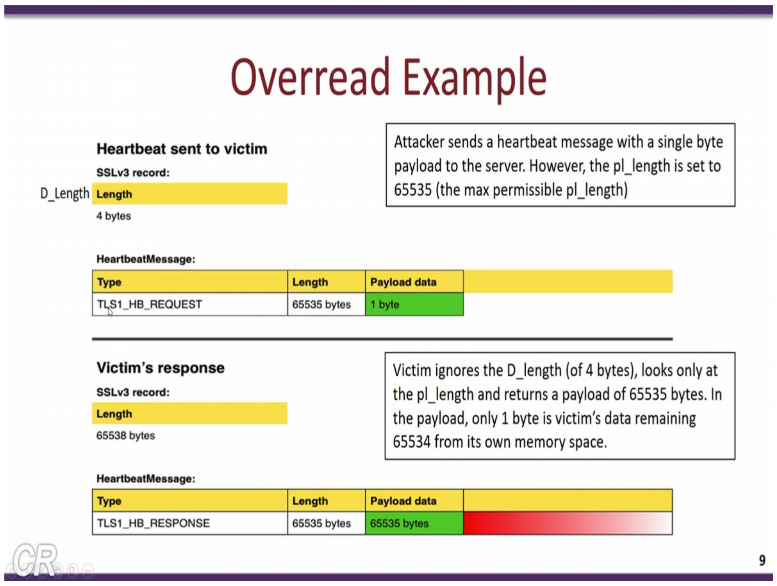
So now over here in the payload data the client system just fills in 1 byte even though it has specify that the length of the payload is 65535 bytes, now what happens during the communication is that this particular a heartbeat message is wrapped in SSL 3 structure, now the SSL 3 measures the length of this message as just 4 bytes, 1 for this type, 2 for length and 1 for this data which it has found, so thus it was specify the D length as 4 bytes.

So what happens in the victim that is our web server is that the web server completely ignores the D length part over here and just looks at the length within the heartbeat message, thus what it is going to do? It is going to copy the payload which is present over here that is 1 byte of payload and also it is going to because the length is specified as 65535 bytes, therefore it will also copy all the adjacent bytes that are present in its memory.

Thus the response from the victim would look like this way, it would comprise of 1 byte for type that is TLS HB RESPONSE, the length would be specify as before as 65535 bytes and the payload data would comprise of the 1 byte that was sent by the client machine as well as 65534 adjacent bytes, thus this particular packet gets wrapped in the SSL 3 response and therefore the length in the SSL 3 responses 65538 bytes, so D length in the response from the server to the client is 65538 bytes.

Now what is happening here is that there is a buffer overread and the servers data present in that particular segment gets transferred to the client and what was actually demonstrated in this heartbeat malware was that a lot of sensitive data held in the server was actually leaked to the client machine.

(Refer Slide Time: 14:10)



Overread Example

Heartbeat sent to victim

SSLv3 record:

D_Length **Length**

4 bytes

HeartbeatMessage:

Type	Length	Payload data
TLS1_HB_REQUEST	65535 bytes	1 byte

Attacker sends a heartbeat message with a single byte payload to the server. However, the pl_length is set to 65535 (the max permissible pl_length)

Victim's response

SSLv3 record:

Length

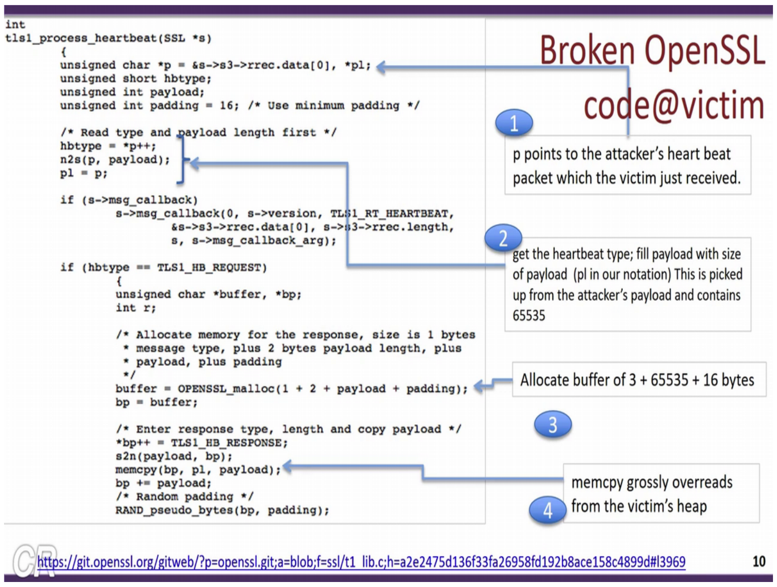
65538 bytes

HeartbeatMessage:

Type	Length	Payload data	
TLS1_HB_RESPONSE	65535 bytes	65535 bytes	

Victim ignores the D_length (of 4 bytes), looks only at the pl_length and returns a payload of 65535 bytes. In the payload, only 1 byte is victim's data remaining 65534 from its own memory space.

9



Broken OpenSSL code@victim

```
int
tls1_process_heartbeat(SSL *s)
        {
        unsigned char *p = &s->s3->rrec.data[0], *pl;
        unsigned short hbtype;
        unsigned int payload;
        unsigned int padding = 16; /* Use minimum padding */

        /* Read type and payload length first */
        hbtype = *p++;
        n2s(p, payload);
        pl = p;

        if (s->msg_callback)
                s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
                        &s->s3->rrec.data[0], s->s3->rrec.length,
                        s, s->msg_callback_arg);

        if (hbtype == TLS1_HB_REQUEST)
                {
                unsigned char *buffer, *bp;
                int r;

                /* Allocate memory for the response, size is 1 bytes
                 * message type, plus 2 bytes payload length, plus
                 * payload, plus padding
                 */
                buffer = OPENSSL_malloc(1 + 2 + payload + padding);
                bp = buffer;

                /* Enter response type, length and copy payload */
                *bp++ = TLS1_HB_RESPONSE;
                s2n(payload, bp);
                memcpy(bp, pl, payload);
                bp += payload;
                /* Random padding */
                RAND_pseudo_bytes(bp, padding);
```

1 p points to the attacker's heart beat packet which the victim just received.

2 get the heartbeat type; fill payload with size of payload (pl in our notation) This is picked up from the attacker's payload and contains 65535

Allocate buffer of 3 + 65535 + 16 bytes

3

4 memcpy grossly overreads from the victim's heap

https://git.openssl.org/gitweb/?p=openssl.git;a=blob;f=ssl/t1_lib.c;h=a2e2475d136f33fa26958fd192b8ace158c4899d#l3969

10

So we will take dig a little more deeper into how the heartbeat actually works and we will look at the open SSL code which actually have the vulnerability, so the important function for us is this particular function that is the TLS process heartbeat okay, so we have here a pointer

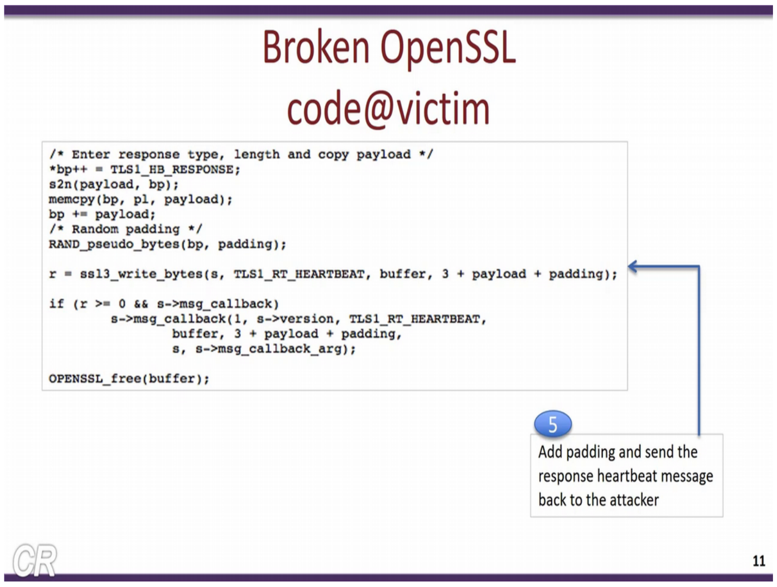
P which is define and this pointer is initialised to the heartbeat message that is the data present in the SSL 3 packet which was obtained.

The next important part is these three statements over here where at the server end, the server is trying to evaluate the hard packet that it has obtained, first thing that server would do is determine the heartbeat type which as we have seen before is the TLS HB REQUEST, so then it would extract the payload length which is then stored in the load, so this is the 2 byte length which gets stored in this payload variable present here and finally there is a pointer P1 defined which points to the actual payload data.

So the next thing will look at is this particular response over here which is a call to the malloc function, so this malloc is call to essentially create the response which is sent back from the server to the client, note the critical aspect over here is that the size requested to malloc comprises of 3 bytes plus the payload length plus the padding and the padding is as specified 16 bytes and payload since it is the payload length and as we seen over here the payload length is 65535 bytes therefore the size of the response is going to be 3 plus 65535 plus 16 bytes.

The fourth critical point in this function is this invocation to memcpy, so this invocation to memcpy essentially creates the response from the server back to the client, it essentially fills the recently created a buffer with the payload data that client has sent, so note that even though the client has sent 1 byte, since the payload specified was 65535 therefore the buffer would actually contain data of 65535 bytes, so out of these 65535 bytes there is only one byte which contains what the client had actually sent and the remaining bytes is due to a buffer overread where 65534 byte adjacent to that one byte is copied into the buffer.

(Refer Slide Time: 16:51)



Broken OpenSSL code@victim

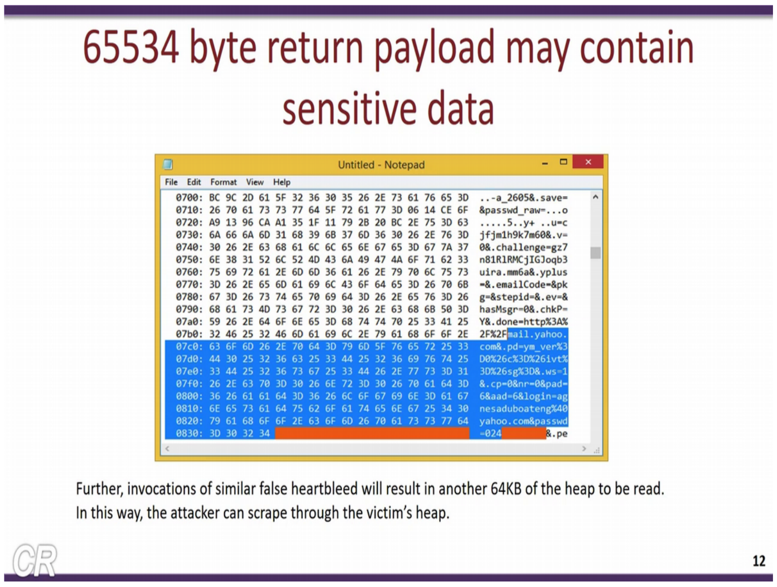
```
/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
bp += payload;
/* Random padding */
RAND_pseudo_bytes(bp, padding);

r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, 3 + payload + padding);

if (r >= 0 && s->msg_callback)
        s->msg_callback(1, s->version, TLS1_RT_HEARTBEAT,
                buffer, 3 + payload + padding,
                s, s->msg_callback_arg);

OPENSSL_free(buffer);
```

5 Add padding and send the response heartbeat message back to the attacker

11

Now this buffer which has just created is wrapped in the SSL 3 structure and sent back to the client, note that we are sending the buffer here and the size of this particular packet is 3 plus payload plus the paddle, thus what the client obtains is the response to its heartbeat message comprising of just one byte which is actually sent and the remaining almost 64K bytes of data which it has gleaned from the server.

(Refer Slide Time: 17:24)



65534 byte return payload may contain sensitive data

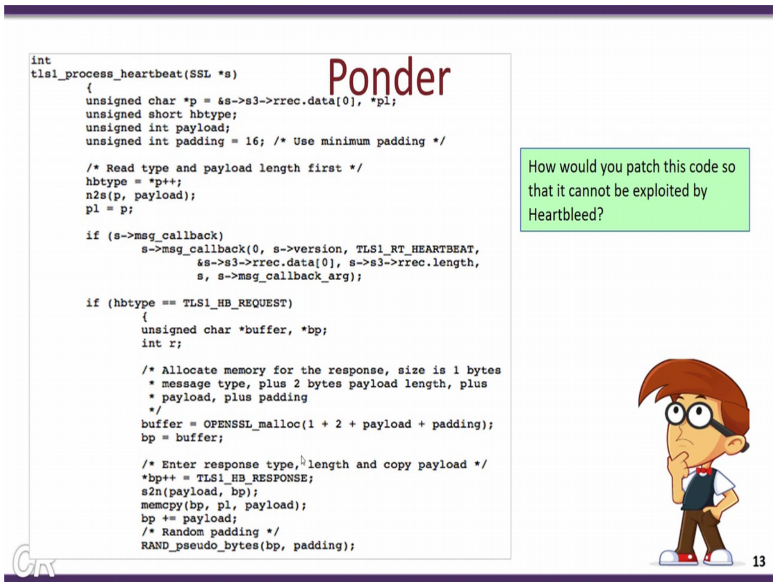
```
0700: BC 9C 2D 61 5F 32 36 30 35 26 2E 73 61 76 65 3D   ..-a_2605&.save=
0710: 26 70 61 73 73 77 64 5F 72 61 77 3D 06 14 CE 6F   &passwd_raw=...o
0720: A9 13 96 CA A1 35 1F 11 79 2B 20 BC 2E 75 3D 63   .....5..y+ ..u=c
0730: 6A 66 6A 6D 31 68 39 6B 37 6D 36 30 26 2E 76 3D   jfjm1h9k7m60&.v=
0740: 30 26 2E 63 68 61 6C 6C 65 6E 67 65 3D 67 7A 37   0&.challenge=gz7
0750: 6E 38 31 52 6C 52 4D 43 6A 49 47 4A 6F 71 62 33   n81RlRMCjIGJoqb3
0760: 75 69 72 61 2E 6D 6D 36 61 26 2E 79 70 6C 75 73   uira.mm6a&.yplus
0770: 3D 26 2E 65 6D 61 69 6C 43 6F 64 65 3D 26 70 6B   =&.emailCode=&pk
0780: 67 3D 26 73 74 65 70 69 64 3D 26 2E 65 76 3D 26   g=&stepid=&.ev=&
0790: 68 61 73 4D 73 67 72 3D 30 26 2E 63 68 6B 50 3D   hasMsgr=0&.chkP=
07a0: 59 26 2E 64 6F 6E 65 3D 68 74 74 70 25 33 41 25   Y&.done=http%3A%
07b0: 32 46 25 32 46 6D 61 69 6C 2E 79 61 68 6F 6F 2E   2F%2Fmail.yahoo.
07c0: 63 6F 6D 26 2E 70 64 3D 79 6D 5F 76 65 72 25 33   com&.pd=ym_ver%3
07d0: 44 30 25 32 36 63 25 33 44 25 32 36 69 76 74 25   D0%26c%3D%26ivt%
07e0: 33 44 25 32 36 73 67 25 33 44 26 2E 77 73 3D 31   3D%26sg%3D&.ws=1
07f0: 26 2E 63 70 3D 30 26 6E 72 3D 30 26 70 61 64 3D   &.cp=0&nr=0&pad=
0800: 36 26 61 61 64 3D 36 26 6C 6F 67 69 6E 3D 61 67   6&aad=6&login=ag
0810: 6E 65 73 61 64 75 62 6F 61 74 65 6E 67 25 34 30   nesaduboateng%40
0820: 79 61 68 6F 6F 2E 63 6F 6D 26 70 61 73 73 77 64   yahoo.com&passwd
0830: 3D 30 32 34 [redacted]   =024[redacted]&.pe
```

Further, invocations of similar false heartbleed will result in another 64KB of the heap to be read.
In this way, the attacker can scrape through the victim's heap.

12

This particular slide shows the dump of the data which is obtain from some server, so what we see over here is that a lot of information present in the server gets leak to the client, many of them are critical information such as the login account and so on, aspect such as the password and so on get heap from the server to the client, so each invocation of a malicious heartbeat message would allow the client to read about 64K of server data, by repeatedly creating the such malicious heartbeat packets over a period of locations the client machine would able to glean almost the entire heaps space of the server.

(Refer Slide Time: 18:07)



Ponder

```
int
tls1_process_heartbeat(SSL *s)
        {
        unsigned char *p = &s->s3->rrec.data[0], *pl;
        unsigned short hbtype;
        unsigned int payload;
        unsigned int padding = 16; /* Use minimum padding */

        /* Read type and payload length first */
        hbtype = *p++;
        n2s(p, payload);
        pl = p;

        if (s->msg_callback)
                s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
                        &s->s3->rrec.data[0], s->s3->rrec.length,
                        s, s->msg_callback_arg);

        if (hbtype == TLS1_HB_REQUEST)
                {
                unsigned char *buffer, *bp;
                int r;

                /* Allocate memory for the response, size is 1 bytes
                 * message type, plus 2 bytes payload length, plus
                 * payload, plus padding
                 */
                buffer = OPENSSL_malloc(1 + 2 + payload + padding);
                bp = buffer;

                /* Enter response type, length and copy payload */
                *bp++ = TLS1_HB_RESPONSE;
                s2n(payload, bp);
                memcpy(bp, pl, payload);
                bp += payload;
                /* Random padding */
                RAND_pseudo_bytes(bp, padding);
```

How would you patch this code so that it cannot be exploited by Heartbleed?

13

So the heart bleed vulnerability was known to the public in 2014 and it took just a few days, some 2 or 3 days to actually fix this vulnerability and patch the open SSL code as you would understand by now the flaw or the vulnerability in the code was very minor, all that was the problem was that there was that the D length could be much smaller than the actual payload length, so what I would like you to think of right now is to look at this code and figure out how or what statements to be added in this code so that the heart bleed vulnerability can be fixed. Thank you.