Buffer Overreads

Chester Rebeiro

Indian Institute of Technology Madras

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
```

Buffer Overreads and Countermeasures

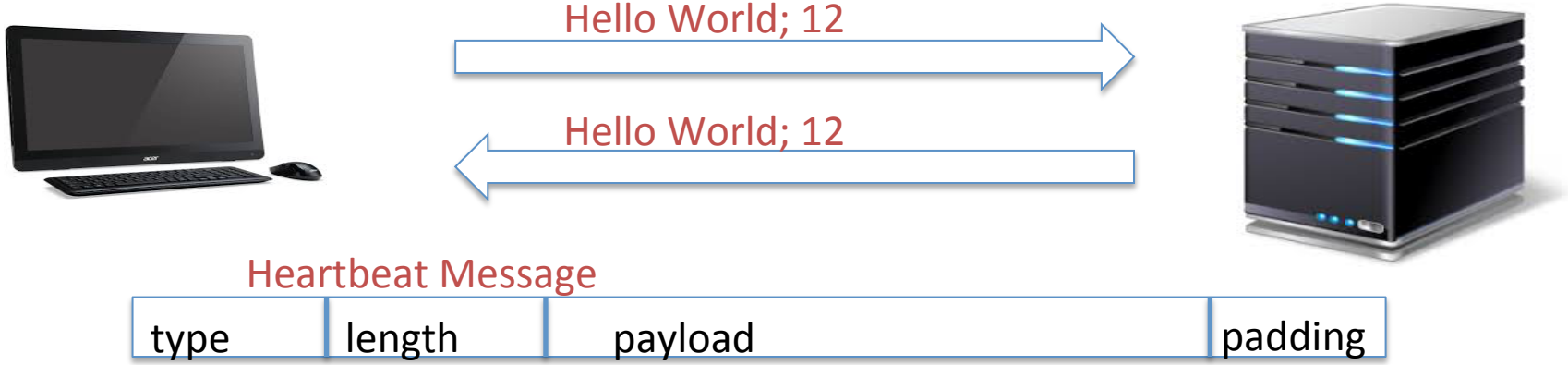
- Cannot be prevented by canaries
  - canaries only look for changes
- Cannot be prevented by the W^X bit
  - we are not executing any code
- Cannot be prevented by ASLR
  - not moving out of the segment

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/

Heartbeat



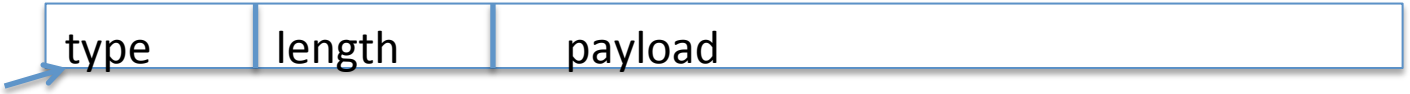
- A component of TLS that provides a means to keep alive secure communication links
  - This avoids closure of connections due to some firewalls
  - Also ensures that the peer is still alive

Heartbeat

Hello World; 12

Hello World; 12

Heartbeat Message



TLS1_HB_REQUEST

- Client sends a heart beat message with some payload
- Server replies with the same payload to signal that everything is OK

SSL3 struct and Heartbeat

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

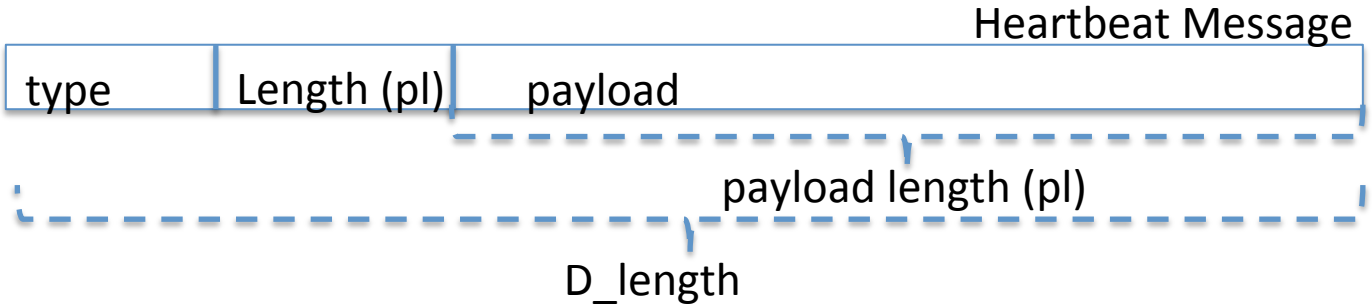
```
struct ssl3_record_st
{
  unsigned int D_length;     /* How many bytes available */
  [...]
  unsigned char *data;     /* pointer to the record data */
  [...]
} SSL3_RECORD;
```

length : length of the heartbeat message

data : pointer to the entire heartbeat message

Format of data (Heartbeat Message)

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

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.

Broken OpenSSL code@victim

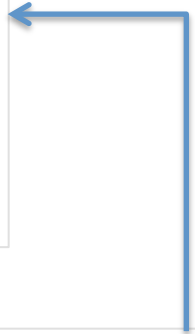
```
int
tls1_process_heartbeat(SSL *s)
        {
        unsigned char *p = &s->s3->rrec.data[0], *pl;
        unsigned short hbtype;
        unsigned int payload;
        unsigned int padding = 16; /* Use minimum padding */

        /* 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
3. Allocate buffer of 3 + 65535 + 16 bytes
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

Broken OpenSSL code@victim

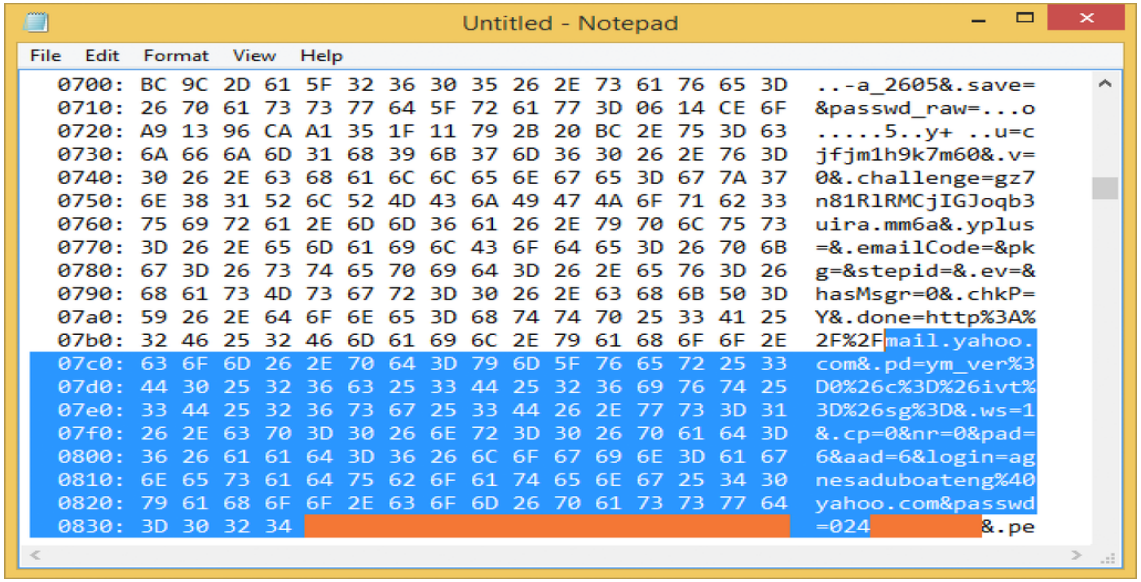
```
/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
bp += payload;
/* Random padding */
RAND_pseudo_bytes(bp, padding);

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

65534 byte return payload may contain sensitive data



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.

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?