UNIVERSITÄT DUISBURG ESSEN

About the Heartbleed Bug

Matthäus Wander

<matthaeus.wander@uni-due.de>

Internet-Technology & Web Engineering

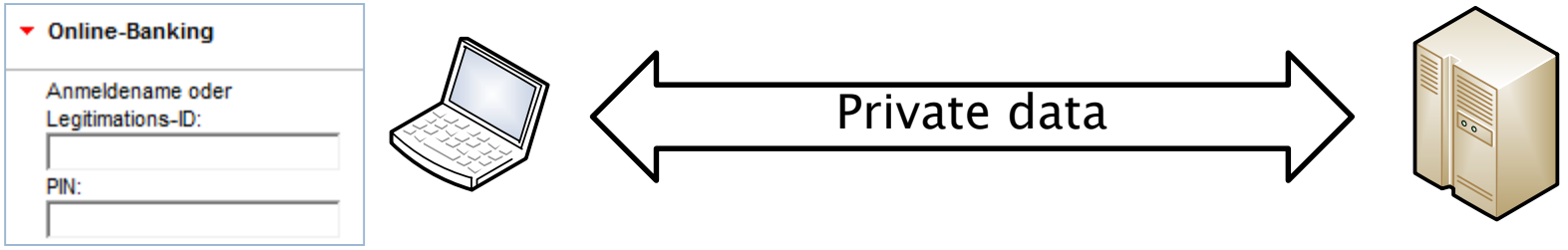
Heartbleed Bug

- Software bug in OpenSSL
  - In implementation of the TLS **Heartbeat** extension
  - Heartbleed is a software bug, not a broken protocol

OpenSSL

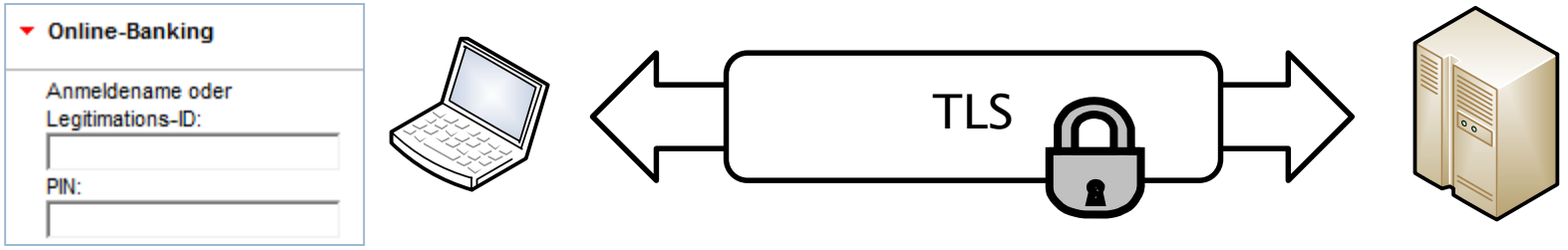
- Open source software library
  - Provides crypto primitives (RSA, AES, SHA-1, ...)
  - Provides TLS functionality to application

Transport Layer Security (TLS) 1/2



- Application sends and receives private data
- Talk to the right server (authenticity)
- Encrypt data (confidentiality)
- Make sure data is not manipulated (integrity)

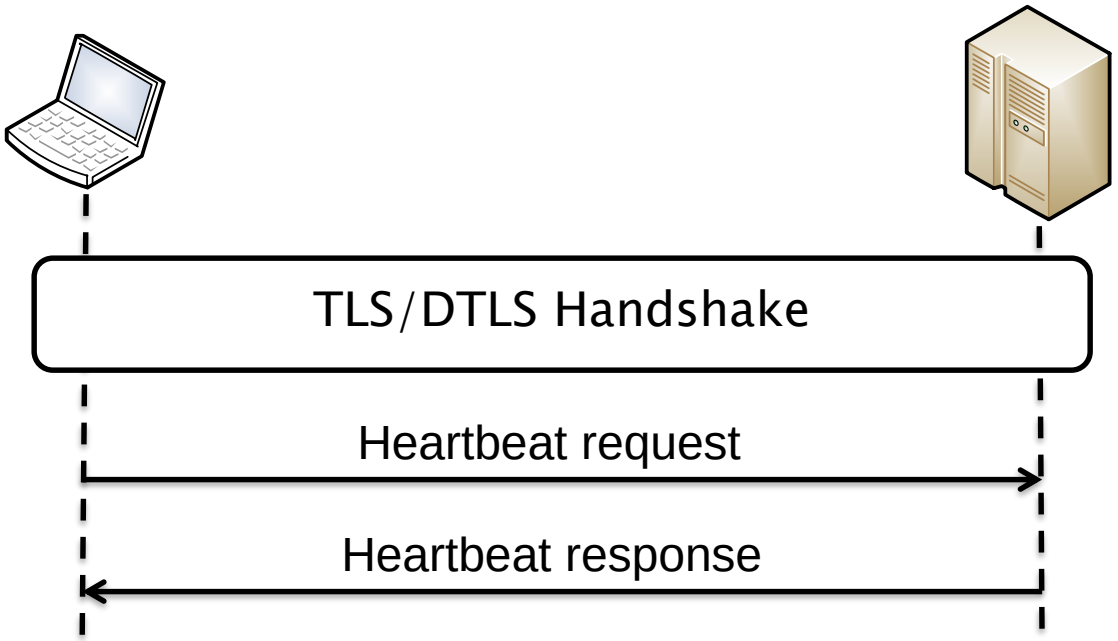
Transport Layer Security (TLS) 2/2



- Cryptographic network protocol
  - TLS is what we use today over TCP
  - SSL are old versions of TLS (deprecated, don‘t use)
  - DTLS is an adaptation of TLS for connectionless transports like UDP (rarely used)
- Extensible with new protocol features

Heartbeat TLS Extension

- Keep-alive mechanism
  - Heartbeat request/response (works like Ping)
  - Specified in RFC 6520, Feb 2012



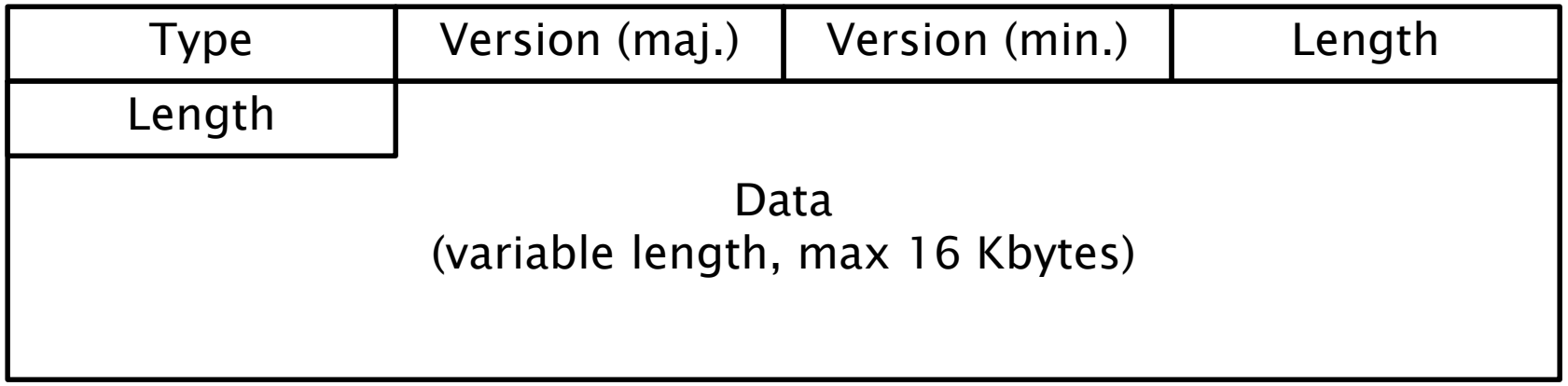
Use Cases

- Keep-alive
  - NAT routers hold a state for TCP/UDP data flows
  - Cheap routers quickly remove idle UDP flows
- Path MTU Discovery
  - Max datagram size is unknown and limited

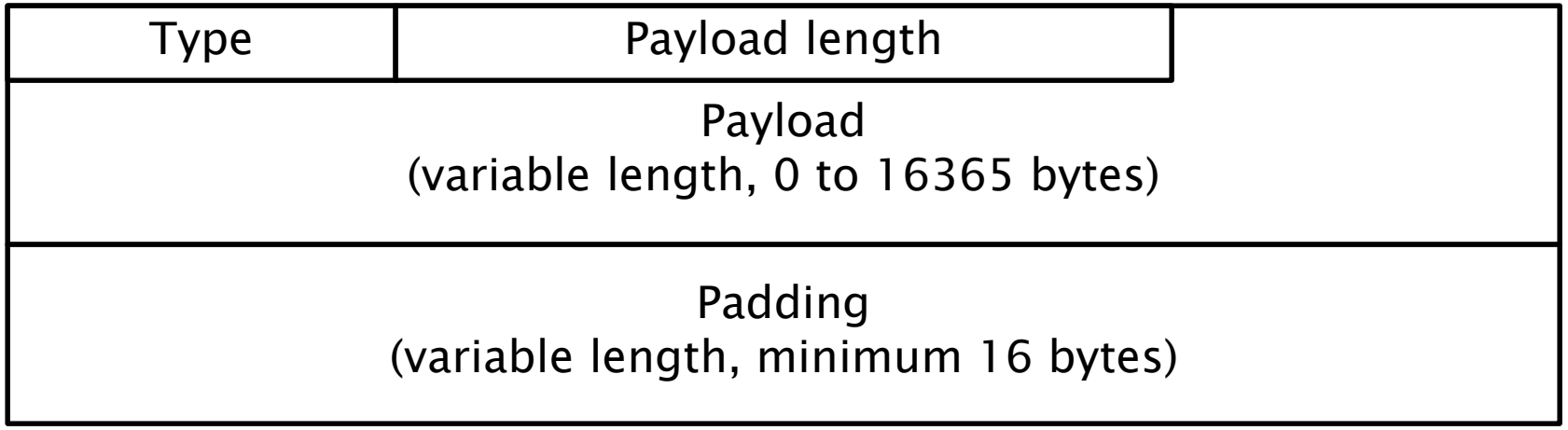
  - Send variable-sized Heartbeats to determine MTU
- Important for UDP-based DTLS
  - Keep-alive can be useful for TCP-based TLS, too

TLS Record Protocol

- Heartbeat messages are sent within TLS records
- „Type“ indicates Heartbeat or other message
  - Skip unknown message types if not implemented
- „Data“ contains the actual Heartbeat message



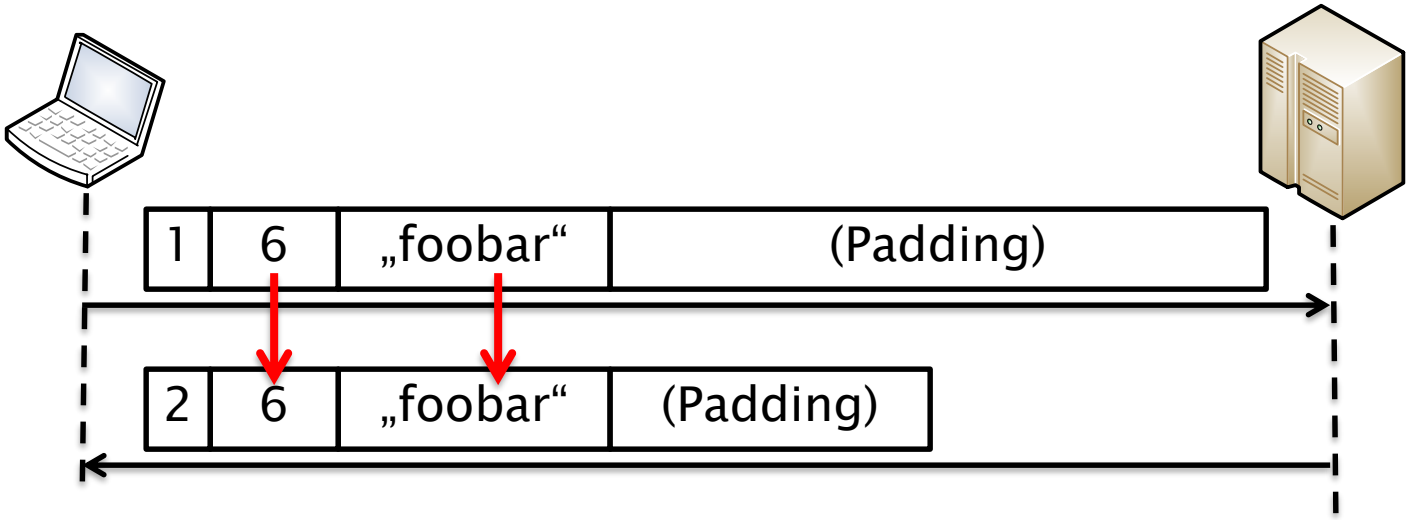
TLS Heartbeat Protocol 1/2



- „Type“: Heartbeat message or response?
- „Payload length“: length of payload field
  - Length of padding is known from total record size
- Payload, padding: arbitrary data

TLS Heartbeat Protocol 2/2

- Application can send any data as payload
  - Receiver copies payload from request into response



- Padding is random data, not copied
  - Variable-length padding for Path MTU Discovery

OpenSSL Implementation 1/2

```
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;


[...]
```

OpenSSL Implementation 1/2

```
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;


[...]
```

Pointer to data field in TLS record

OpenSSL Implementation 1/2

```
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;


[...]
```

Read Heartbeat type
(and increment pointer)

OpenSSL Implementation 1/2

```
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;


[...]
```

Read payload length

OpenSSL Implementation 1/2

```
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;


[...]
```

Pointer to payload

OpenSSL Implementation 1/2

```
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;


[...]
```

No input validation of payload length

OpenSSL Implementation 2/2

```
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);

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

OpenSSL Implementation 2/2

```
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);

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

Initialize output buffer

OpenSSL Implementation 2/2

```
unsigned char *buffer, *bp;
int r;

/* Allocate memory for the response, size is 1 bytes
 * message t                    payload length, plus
 * payload,
 */
buffer = OPENS_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);

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

Pointer to output buffer

OpenSSL Implementation 2/2

```
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);

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

Heartbeat response type

OpenSSL Implementation 2/2

```
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);

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

Write payload length

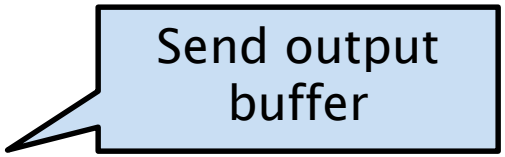
OpenSSL Implementation 2/2

```
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);

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

Copy payload from input buffer to output buffer

OpenSSL Implementation 2/2

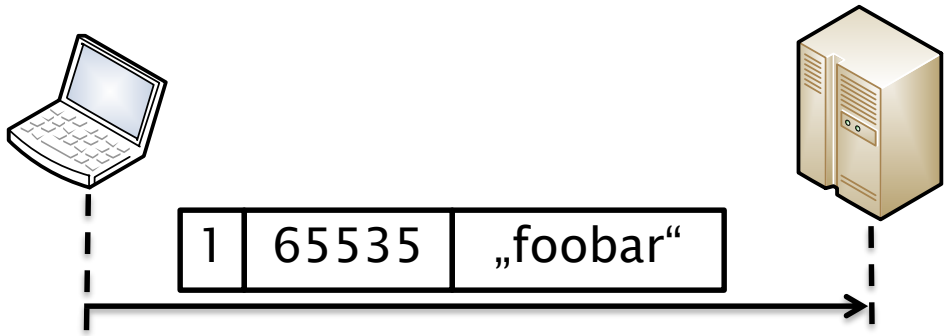
```
unsigned char *buffer, *bp;
int r;

/* 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);

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



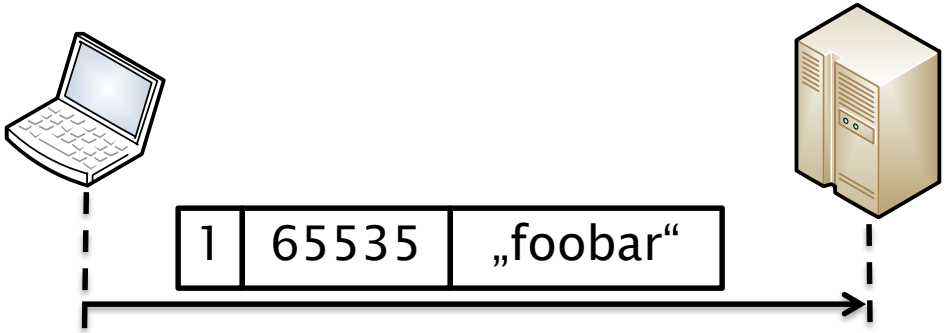
OpenSSL Implementation 2/2

```
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);

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

Exploit

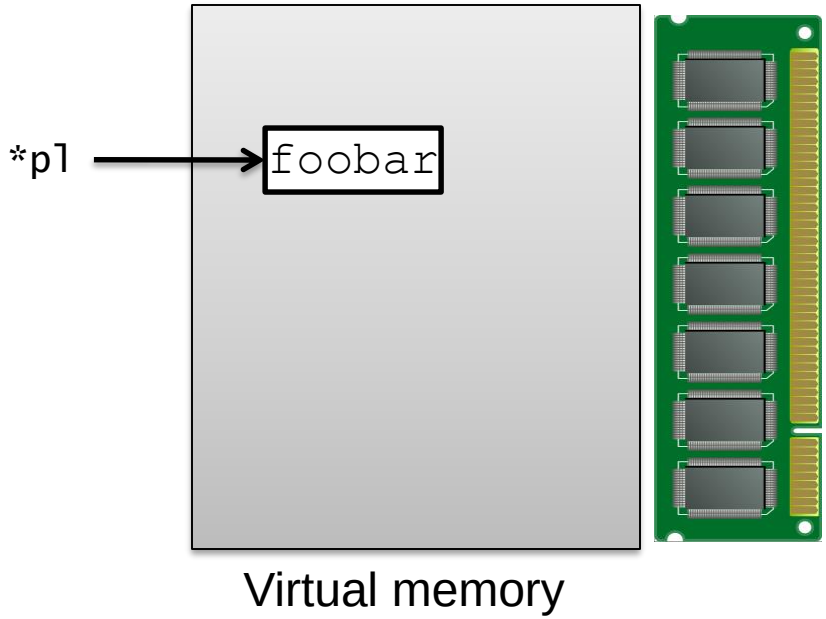


Exploit



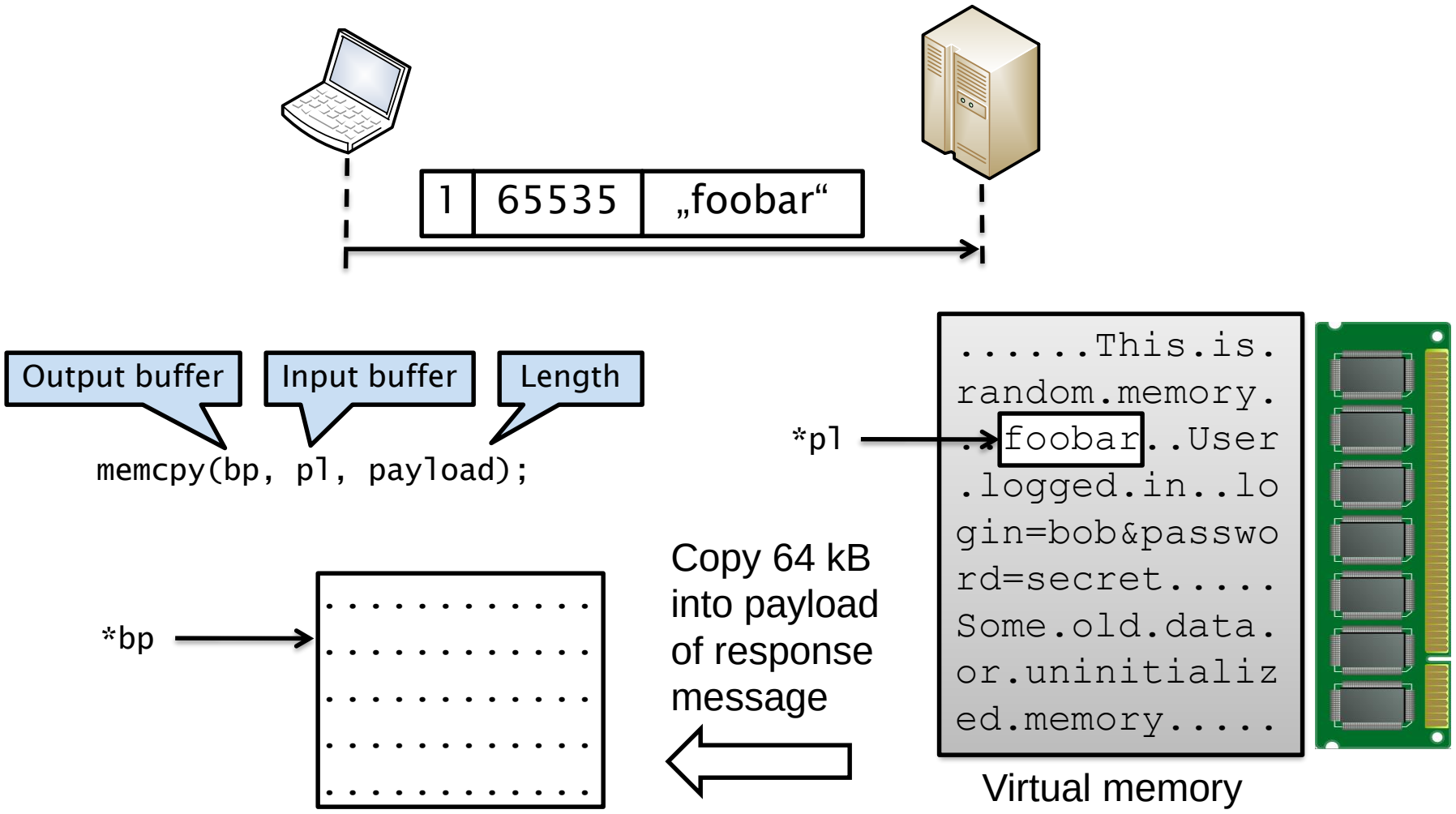
Output buffer | Input buffer | Length

```
memcpy(bp, pl, payload);
```



Virtual memory

Exploit


1
65535
„foobar“
Output buffer
Input buffer
Length
memcpy(bp, pl, payload);
*bp
*pl
.....This.is.
random.memory.
..foobar..User
.logged.in..lo
gin=bob&passwo
rd=secret.....
Some.old.data.
or.uninitializ
ed.memory.....
Copy 64 kB
into payload
of response
message
Virtual memory

Conclusion

- Heartbleed is a software bug in OpenSSL
  - Buffer over-read beyond its boundary
- In Heartbeat extension for TLS/DTLS
  - Enabled by default in OpenSSL
- Affected millions of devices (clients and servers)
- Bug discovered after 2 years
- Cause: missing input validation
- Data transmitted over network is not trusted