Lab Exercise oob1

This is a lab exercise on developing secure software. For more information, see the introduction to the labs.

1. Task

Please change the code below to fix this simplified version of the "Heartbleed" bug in OpenSSL, an example of a buffer overread.

2. Background

In almost all programming languages, if program attempts to read or write outside of a buffer, the default is always either an attempt to resize the buffer or an error of some kind (e.g., by raising an exception). That's because it's extremely easy to accidentally attempt to read or write outside of a buffer.

However, C and C++ are different. C++ has evolved to become somewhat safer (e.g., through smart pointers), In C and C++, attempting to do actions like read or write outside a buffer is in many cases *undefined behavior*, and when undefined behavior occurs, *anything* is allowed to happen without any kind of protection. In practice, what often happens is a read or write (respectively) of other data. There are proposals to improve C++ memory safety, but currently many C++ built-in constructs (like arrays) are not memory safe, so we will treat the two languages together here.

The 2014 revelation of the Heartbleed vulnerability (CVE-2014-0160) is an example of a buffer overread vulnerability. Heartbleed was a vulnerability in OpenSSL, a widely-used toolkit written in C that implements the cryptographic protocol Secure Sockets Layer (SSL) and its successor the Transport Layer Security (TLS). Heartbleed affected a huge number of popular websites, including Google, YouTube, Yahoo!, Pinterest, Blogspot, Instagram, Tumblr, Reddit, Netflix, Stack Overflow, Slate, GitHub, Yelp, Etsy, the U.S. Postal Service (USPS), Blogger, Dropbox, Wikipedia, and the Washington Post. See here for more.

3. Task Information

We're going re-create the fix in OpenSSF for the Heartbleed vulnerability by modifying function `dtls1_process_heartbeat` .

At this point in the code, the construct `s->s3->rrec.length` indicates how many bytes are available. If we don't check for the maximum sizes, we could easily cause reading beyond a buffer. Modify the code below in two places to fix this.

First, modify the code so that if the minimum length of a response `(1 + 2 + 16)` is more than the length claimed by `s->s3->rrec.length`, then return 0 without sending a heartbeat, This will prevent trying to create a heartbeat when there's not enough room to create a heartbeat at all.

Second, modify the code so that if the minimum length of a response with a payload `(1 + 2 + payload + 16)` is more than the total length for a response given in `s->s3->rrec.length` then again return 0 without sending a heartbeat.

This will prevent trying to create a heartbeat when there's not enough room to create one and there was a payload to return as a heartbeat.

Note that this is not terribly difficult to fix. The code we add is short. The problem is that reading and writing buffers is extremely common, but by default such accesses are unsafe in C and C++. In practice it is difficult to *always* check all ranges in all possible cases, which is why memory safety vulnerabilities are so common in programs written in C or C++.

Use the “hint” and “give up” buttons if necessary.

4. Interactive Lab (COMPLETE!)

```
int
dtls1_process_heartbeat(SSL *s)
  {
  unsigned char *p = &s->s3->rrec.data[0], *pl;
  unsigned short hbtype;
  unsigned int payload;

  // ... Some details omitted here
```

```
 if(1+2+16>s->s3->rrec.length)
    return 0;
```

```
  hbtype = *p++;
  n2s(p, payload);
```

```
if(1+2+payload+16>s->s3->rrec.length)
  return 0;
```

```
  // ... Later on there will be a memory copy ("memcpy")
  // to copy the payload data into a new buffer. If we
  // had not checked, it would not be long enough:
  //   memcpy(bp, pl, payload);
```

Hint Reset Give up

This lab was developed by David A. Wheeler at The Linux Foundation.

▶ Source: This example was extracted from OpenSSL file `ssl/d1_both.c` commit 731f431497f4; you can see its full header by expanding details here.

Completed 2025-02-13T21:46:17.825Z 704ecd52-202a-4020-a2c2-a36ae5d62fca 1crn6vx0x4aj74 (GA)