Ravi Nayyar
9/23/2021
Secure Coding

Fuzzing Assignment

Section 1: Finding the Heartbleed Vulnerability

1.

The Heartbleed Bug (CVE-2014-0160) was a vulnerability in the OpenSSL library. OpenSSL is used to encrypt communications on the network and is most commonly associated with https websites. The sockets that are used for communication between the website and the server timeout and close if no activity has been detected; to prevent this, a heartbeat packet, with a max size of 64 KB, is sent from the website. The server will then respond with a corresponding amount of data.

The problem occurs when an attacker sends a heartbeat packet and tells the server that it is 64 KB in size when actually it is much less. The server will respond with your original data but will also pad the response with data from the server's memory so that it is 64 KB in size. That padded data may be unintelligible, or it may contain sensitive data – anything from passwords and usernames to cryptographic keys, etc.
https://heartbleed.com/
https://www.youtube.com/watch?v=6Sz5wBBXzpc&t=1s

2.

a. N/A

b.

The main function in handshake.cc first initializes the SSL library and registers all the requisite algorithms, registers the error strings for the libcrypto functions, and then sets the certificate and private key variables to the contents of premade files. The aforementioned steps take place in the init function (called from main) and is essentially creating an SSL context, which allows the use of SSL and its functions in the program.

The main function then creates a server structure, and both read and write stream operations and links them all together. It then sets SLL to work in server mode. The main function then creates a statically allocated buffer of 100 bytes and attempts to fill that buffer with the contents of a file which is taken as a CLI input. The buffer is then sent to the write stream before the SSL handshake is conducted. Finally, the main program removes the server structure that was previously created
https://www.openssl.org/docs/manmaster/man3/SSL_new.html
https://www.openssl.org/docs/man1.0.2/man3/BIO_new.html
https://www.openssl.org/docs/man1.0.2/man3/SSL_set_bio.html
https://www.openssl.org/docs/man1.1.1/man3/BIO_write.html
https://linux.die.net/man/3/ssl_do_handshake
https://www.openssl.org/docs/man1.1.1/man3/SSL_free.html

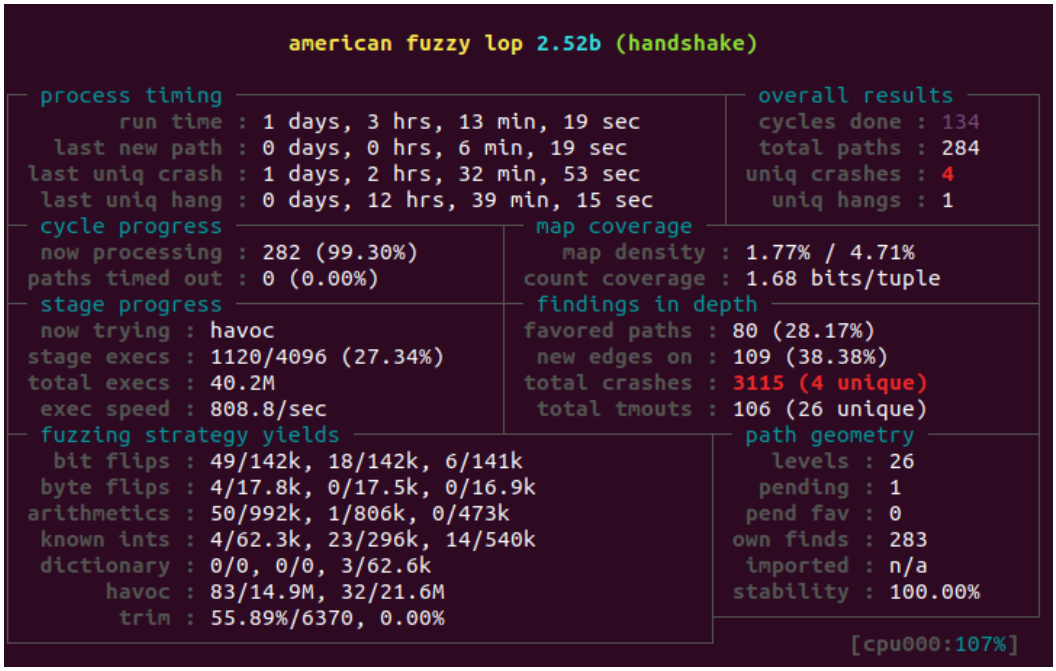
3. N/A

4.

We need to use Address Sanitizing when fuzzing Heartbleed because static analysis or test-driven development would not be sufficient. Oftentimes developers write bad incomplete tests or focus more on the quantity of code coverage rather than writing quality tests. Furthermore, even the top standard automated static analysis tools were unable to find Heartbleed in their current configuration due to OpenSSL not deallocating memory back to the system. Address sanitizing's main job is detecting buffer overflow vulnerabilities (such as over reads and over writes – on both the stack and the heap). When address sanitizers are combined with fuzzers like AFL, the crashes that they both create after finding hard to detect problems can then be interpreted.
https://dwheeler.com/essays/heartbleed.html

5. N/A

6.

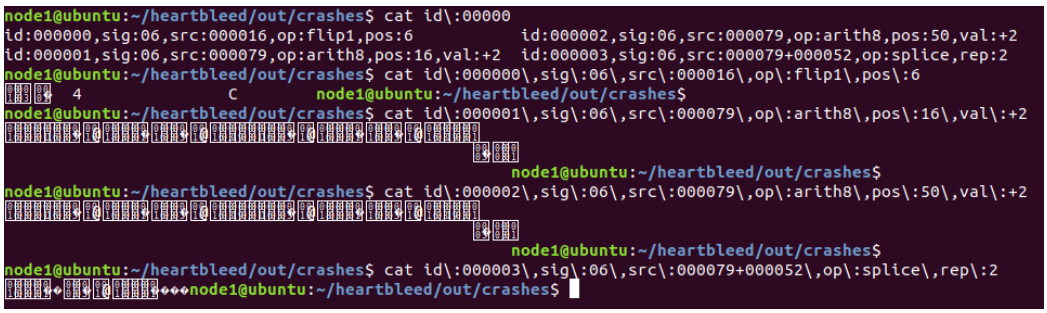


7.

a. N/A

b.

Over the course of 27 hours, AFL found 4 crashes (Though the first was found in around 10 minutes). Here are the contents of the 4 crash files.



c.

The outputted error was a heap-buffer-overflow error. This error occurs when a buffer overflow occurs in memory allocated in the heap. Address Sanitizer is not part of the error, but rather the open-source program that is used to find memory bugs in C and C++ program.

```
node1@ubuntu:~/heartbleed$ ./handshake < ./out/crashes/id\:000000\,sig\:06\,src\:000016\,op\:flip1\,pos\:6
=================================================================
==18630==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x629000009748 at pc 0x0000004bda21 bp 0x7ffe576f2610 sp 0x7ffe576f1dc0
READ of size 33536 at 0x629000009748 thread T0
    #0 0x4bda20  (/home/node1/heartbleed/handshake+0x4bda20)
    #1 0x51eef8  (/home/node1/heartbleed/handshake+0x51eef8)
    #2 0x59e1dd  (/home/node1/heartbleed/handshake+0x59e1dd)
    #3 0x5a34da  (/home/node1/heartbleed/handshake+0x5a34da)
    #4 0x5668a7  (/home/node1/heartbleed/handshake+0x5668a7)
    #5 0x562fa8  (/home/node1/heartbleed/handshake+0x562fa8)
    #6 0x538356  (/home/node1/heartbleed/handshake+0x538356)
    #7 0x510fff  (/home/node1/heartbleed/handshake+0x510fff)
    #8 0x7f262978abf6  (/lib/x86_64-linux-gnu/libc.so.6+0x21bf6)
    #9 0x41d379  (/home/node1/heartbleed/handshake+0x41d379)

0x629000009748 is located 0 bytes to the right of 17736-byte region [0x629000005200,0x629000009748)
allocated by thread T0 here:
    #0 0x4d29d0  (/home/node1/heartbleed/handshake+0x4d29d0)
    #1 0x5d7dad  (/home/node1/heartbleed/handshake+0x5d7dad)

SUMMARY: AddressSanitizer: heap-buffer-overflow (/home/node1/heartbleed/handshake+0x4bda20)
Shadow bytes around the buggy address:
  0x0c527fff9290: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c527fff92a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c527fff92b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c527fff92c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c527fff92d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c527fff92e0: 00 00 00 00 00 00 00 00[fa]fa fa fa fa fa fa
  0x0c527fff92f0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c527fff9300: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c527fff9310: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c527fff9320: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c527fff9330: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
==18630==ABORTING
```

8.

I learned that at the time, almost no automated tools used for finding memory bugs would have been able to find Heartbleed. Also, that the damage caused by Heartbleed is still unknown and that the vulnerability is still being exploited to this day due to the usage of the non-updated OpenSSL library.

Section 2

1.

The program that I chose to fuzz was taken directly from the first CTF problem. The code can be seen below.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

#define BUFSIZE 128
#define FLAGSIZE 128

void vuln(){
  char buf[BUFSIZE];
  gets(buf);
  puts(buf);
  fflush(stdout);
}

int main(int argc, char **argv){
  vuln();
  return 0;
}
```

The CTF problem uses the “gets()” program to take in input from a user and uses “puts()” to replay it out on the console.

2.

The initial seed file I gave to AFL was placed in the input file and contained the text string “AAAAAAAAAAAA”

```
node1@ubuntu:~/Desktop/Part2/in$ cat input
AAAAAAAAAAAA
node1@ubuntu:~/Desktop/Part2/in$
```

3.

The command I used to compile the object file was:
AFL_USE_ASAN=1 afl-clang-fast++ -g test.c -o test

The command I used to run AFL on the program was:
echo core | sudo tee /proc/sys/kernel/core_pattern
afl-fuzz -i ./in -o ./out -m none -- ./test

4.

Image of the AFL crash

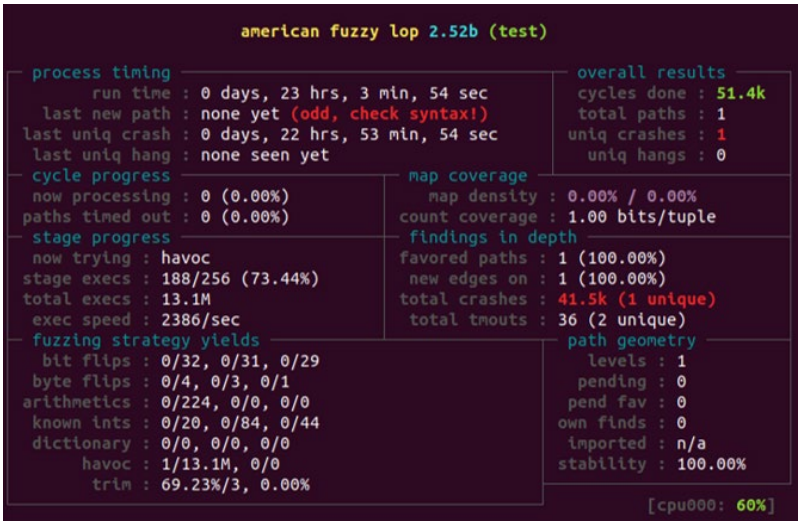


Image of crash file contents

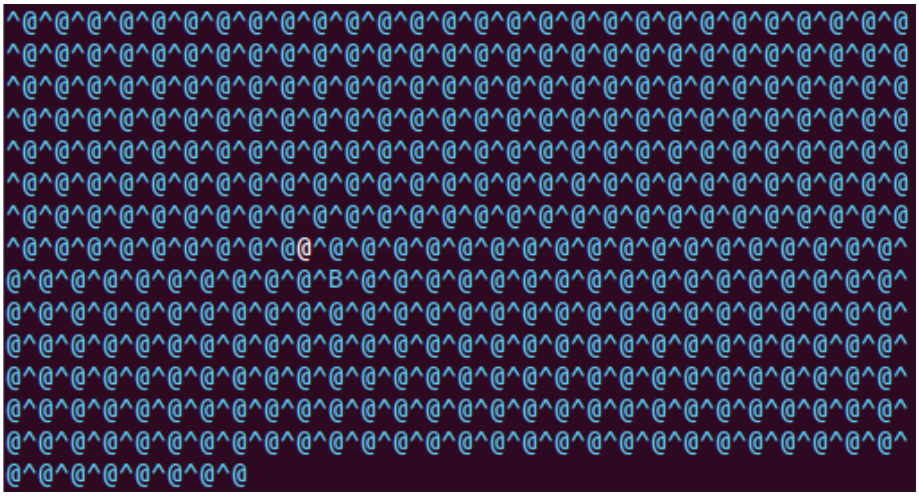
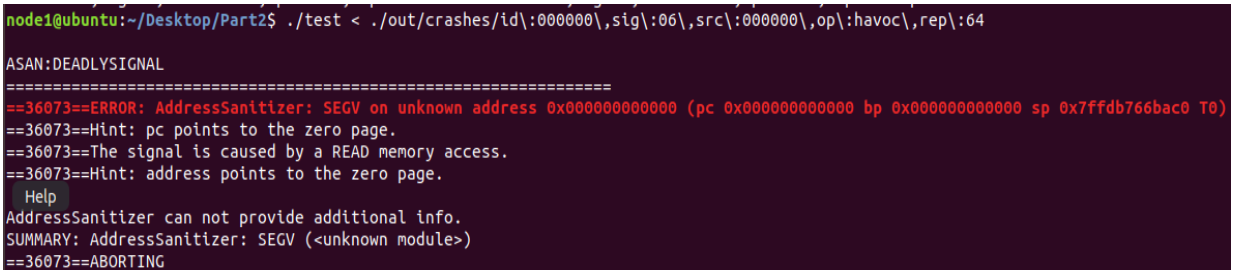


Image of crash file being used as an input



A potential reason for this error is that the input within the crash file is somehow referencing an address that is outside of the function's stack space. When it attempts to read that invalid memory address, it crashes and returns the SEGV error.