Don’t Rush Ahead in Lab05

- The fuzzing tool can make a mess of your Linux account if you are not careful
- ***Follow the instructions provided as written unless directed otherwise by the instructor***

CPE 455/555
Secure Software Development
Lab05 – Fuzzing with AFL

Dr. David J. Coe
Software Safety and Security Engineering Lab
Electrical and Computer Engineering Department

Sponsored by National Security Agency Grant H98230-17-1-0344

Outline

- Introduction to Fuzzing
- Fuzzing with AFL
  - Example #1: Simple Example – Crashy
  - **Quiz-Lab05**
  - Example #2: Common Utility – SQLite (Demo)
- Discussion
- Summary
- References

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

Introduction to Fuzzing

Sponsored by National Security Agency Grant H98230-17-1-0344

Introduction to Fuzzing - 1

- What is **`Fuzzing`**?
  - At its core, Fuzzing is a technique for testing a program by supplying ‘random’ inputs and seeing if the program crashes or hangs.
  - This is a form of dynamic testing that evaluates a program’s performance at run-time.
  - Because the inputs are often unexpected values, they can find crashes that are not seen with dynamic analysis edge cases.

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Introduction to Fuzzing - 2

- The term “fuzzing” was coined by Professor Barton Miller at the University of Wisconsin-Madison.
  - It was a project assignment for his CS736 class in 1988.
  - The assignment was to create a “Fuzz Generator” to supply various types of random ASCII inputs to UNIX utilities and try to break them.
    - They broke about 30% of them.

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Introduction to Fuzzing - 3

- Fuzzing programs can be dumb or intelligent.
- The dumb programs are brute force methods that just randomly try inputs.
  - If the program takes complicated input formats or has a lot of conditional paths, the random inputs are unlikely to be formatted well enough to make it very far through the program's control flow.
- The intelligent programs supply inputs that are usually not completely random, but mutated forms of valid input structures.
  - This allows for inputs that reach certain paths to be held static with other values changed to test out different paths more thoroughly and discover more paths.
    - Refered to as guided fuzzing.

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Introduction to Fuzzing - 4

- Software developers and testers utilize fuzzing to improve the quality of products under development

- ***Hackers use fuzzing techniques to discover defects that may prove to be exploitable***

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Fuzzing with AFL

Sponsored by National Security Agency Grant H98230-17-1-0344

Fuzzing with AFL - 0

- The ***American Fuzzy Lop*** is a rabbit.

https://zepafarm.club/american-fuzzy-lop-rabbit/

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Fuzzing with AFL - 1

- The American Fuzzy Lop is a rabbit.
- AFL, named for the rabbit, is an open-source fuzzer that can make use of instrumentation inserted into the compiled code for the unit under test to keep track of paths found.
- It also uses genetic algorithms to trim the test cases to the smallest size possible.
  - The developer also cautions against supplying extra inputs where one with the proper format will suffice.
    - e.g. an image processing program only needs one picture of each type it accepts as input for the fuzzer to learn the format.

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Fuzzing with AFL - 2

- The developer describes AFL's overall process as follows:
  - Load user test cases into a queue
  - Take next file from the queue
  - Try to trim the test case so that it doesn't alter the measured behavior of the program
  - Mutate the file using a variety of strategies
  - If a new path is found, add that mutated input as a new entry in the queue
  - Repeat

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Fuzzing with AFL - 3

- The latest version of AFL is available from the developer's web site and can be obtained by using the command:
  - **wget http://lcamtuf.coredump.cx/afl/releases/afl-latest.tgz**
- Then you can extract and build it with the supplied makefile
- **NOTE: For Lab05, the AFL tool is already installed on the ENG 246 Linux lab machines which are remotely accessible**

Sponsored by National Security Agency Grant H98230-17-1-0344

Access to AFL

- SSH to **blackhawk.ece.uah.edu**
- Then from **blackhawk**, SSH to an ENG 246 lab machine

  ssh –Y username@172.21.246.X

  where **1 <= X <= 35**

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Example #1
Simple Example – Crashy

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

Example #1 - Crashy - 1

- Crashy.c is a sample program that reads and prints out a custom file format.

```
xray@xray:~/NSA/Fuzz/crashy$ ls
crashy  crashy.c  crashyGDB  exampleCrash01  inputFiles  Makefile  outputResults
xray@xray:~/NSA/Fuzz/crashy$ cd inputFiles/
xray@xray:~/NSA/Fuzz/crashy/inputFiles$ ls
example1  example2  example3  example4  example5
xray@xray:~/NSA/Fuzz/crashy/inputFiles$ ../crashy example1
i: 0x78563412
c: 0xcc
s: hello
xray@xray:~/NSA/Fuzz/crashy/inputFiles$ ../crashy example2
i: 0x00000001
i: 0x00000002
i: 0x00000003
i: 0x00000004
i: 0x00000005
xray@xray:~/NSA/Fuzz/crashy/inputFiles$ ../crashy example3
s: this
s: is
s: an
s: example
```

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Crashy Input File Format

- Input file format: Block-Type Value
  - 0x01: 32-bit integer
  - 0x02: 8-bit character
  - 0x03: string (length including null terminator)
  - 0x04: comment (null terminated string)
- Terminated with 0xff

```
-bash-4.2$ hexdump -C example1
00000000  01 12 34 56 78 02 cc 03  06 00 00 00 68 65 6c 6c  |..4Vx.......hell|
00000010  6f 00 ff                                          |o..|
00000013
-bash-4.2$
-bash-4.2$ ./crashy example1
i: 0x78563412
c: 0xcc
s: hello
-bash-4.2$
```

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Example #1

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char *argv[]) {
        FILE *fp;
        unsigned int sz, len, ptr;
        char *buffer;
        char type;
        char tmp_char;
        int tmp_int;
        char *tmp_string;
        if(argc != 2) {
          fprintf(stderr, "Usage: %s <inputfile>\n",
                  argv[0]);
          return 1;
        }
        fp = fopen(argv[1], "r");
        if(!fp) {
          fprintf(stderr, "Failed to open '%s'\n", argv[1]);
          return 1;
        }
        fseek(fp, 0, SEEK_END);
        sz = ftell(fp);
        fseek(fp, 0, SEEK_SET);
        buffer = malloc(sz);
        if(!buffer) {
          fprintf(stderr, "Failed to allocate %d bytes of
                  memory\n", sz);
          return 1;
        }
        fread(buffer, sizeof(char), sz, fp);
        ptr = 0;
```

Crashy - 2

```
while(ptr < sz) {
  type = buffer[ptr++];
  switch(type) {
        case '\x01': // integer
                tmp_int = *(int *)(buffer + ptr);
                ptr += sizeof(int);
                printf("i: 0x%08x\n", tmp_int);
                break;
        case '\x02': // char
                tmp_char = buffer[ptr++];
                printf("c: 0x%02x\n", (unsigned char)tmp_char);
                break;
        case '\x03': // string
                len = *(int *)(buffer + ptr);
                ptr += sizeof(int);
                tmp_string = malloc(len);
                if(!tmp_string) {
                        fprintf(stderr, "Failed to allocate
                          %d bytes of memory\n", len);
                        return 1;
                }
                strcpy(tmp_string, (buffer + ptr));
                printf("s: %s\n", tmp_string);
                ptr += len;
                break;
        case '\x04': // comment
                printf("#: %s\n", (buffer + ptr));
                while(*(buffer + ptr++) != '\x00');
                break;
        case '\xff': // END
                return 0;
        default:
                fprintf(stderr, "Unknown data type '%c'\n", type);
  }
}
}
```

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

AFL Compilation

```
-bash-4.2$ cat Makefile
CC       := afl-gcc
CFLAGS   := -Wno-unused-result

all: crashy.c
        $(CC) $(CFLAGS) -o crashy crashy.c

clean:
        rm crashy
-bash-4.2$
-bash-4.2$ make
afl-gcc -Wno-unused-result -o crashy crashy.c
afl-cc 2.52b by <lcamtuf@google.com>
afl-as 2.52b by <lcamtuf@google.com>
[+] Instrumented 23 locations (32-bit, non-hardened mode,
ratio 100%).
-bash-4.2$
```

Sponsored by National Security Agency Grant H98230-17-1-0344

AFL Input File Analysis

- Input file format: Block-Type Value
  - 0x01: 32-bit integer
  - 0x02: 8-bit character
  - 0x03: string (length including null terminator)
  - 0x04: comment (null terminated string)
- Terminated with 0xff

```
-bash-4.2$ afl-analyze -i example1 ./crashy @@
afl-analyze 2.52b by <lcamtuf@google.com>

[+] Read 19 bytes from 'example1'.
[*] Performing dry run (mem limit = 50 MB, timeout = 1000 ms)...
[*] Analyzing input file (this may take a while)...

    01  - no-op block              01  - suspected length field
    01  - superficial content      01  - suspected cksum or magic int
    01  - critical stream          01  - suspected checksummed block
    01  - "magic value" section

[000000] #01 #12  4   V   x  #02 #cc #03 #06 #00 #00 #00  h   e   l   l  >
[000016]  o  #00 #ff

[+] Analysis complete. Interesting bits: 42.11% of the input file.
[+] We're done here. Have a nice day!

-bash-4.2$
```

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Example #1 - Crashy - 3

- Given the source code and a few example input files, we can fuzz the program using AFL.
- First, we extract the compressed **crashy** files from the supplied zip file.

unzip crashy.zip

Sponsored by National Security Agency Grant H98230-17-1-0344

Example #1 - Crashy - 4

- Then, we use AFL's modified version of **gcc** to compile the code with the instrumentation it needs for intelligent fuzzing.

```
xray@xray:~/NSA/Fuzz/crashy$ make CC=$HOME/NSA/Fuzz/afl-2.52b/afl-gcc CFLAGS=-Wno-unused-re
sult
/home/xray/NSA/Fuzz/afl-2.52b/afl-gcc -Wno-unused-result -o crashy crashy.c
afl-cc 2.52b by <lcamtuf@google.com>
afl-as 2.52b by <lcamtuf@google.com>
[+] Instrumented 22 locations (64-bit, non-hardened mode, ratio 100%).
```

- **NOTE:** The provided **Makefile** has already been altered, replacing **gcc** with **afl-gcc** so there is no need to supply make with command line arguments as shown above
- To compile, type

  make

Sponsored by National Security Agency Grant H98230-17-1-0344

Example #1 - Crashy - 5

- Before we can start the fuzzer, we need to set up the directory structure so that we have an input directory with the example files and an empty output directory to hold the findings and progress information AFL uses.

```
xray@xray:~/NSA/Fuzz$ cd crashy
xray@xray:~/NSA/Fuzz/crashy$ ls
crashy.c  example1  example2  example3  example4  example5  Makefile
xray@xray:~/NSA/Fuzz/crashy$ mkdir inputFiles outputResults
xray@xray:~/NSA/Fuzz/crashy$ mv example* inputFiles/
xray@xray:~/NSA/Fuzz/crashy$ cd inputFiles/
xray@xray:~/NSA/Fuzz/crashy/inputFiles$ ls
example1  example2  example3  example4  example5
```

  - We just made subdirectories in the **crashy/** folder called **inputFiles** and **outputResults**.

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Example #1 - Crashy - 6

- Finally, we can start AFL while in the directory with crashy's executable with the command:

  afl-fuzz -i inputFiles -o outputResults ./crashy @@

- The **-i** flag tells AFL where to find input examples and the **-o** flag is where the results will be placed.
  - If the process is interrupted, you can continue by passing a dash as the input directory (**-i -**) and using the same output directory.
- The **@@** is a placeholder for a filename which AFL will supply from the input examples it finds.

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Example #1 - Crashy - 7

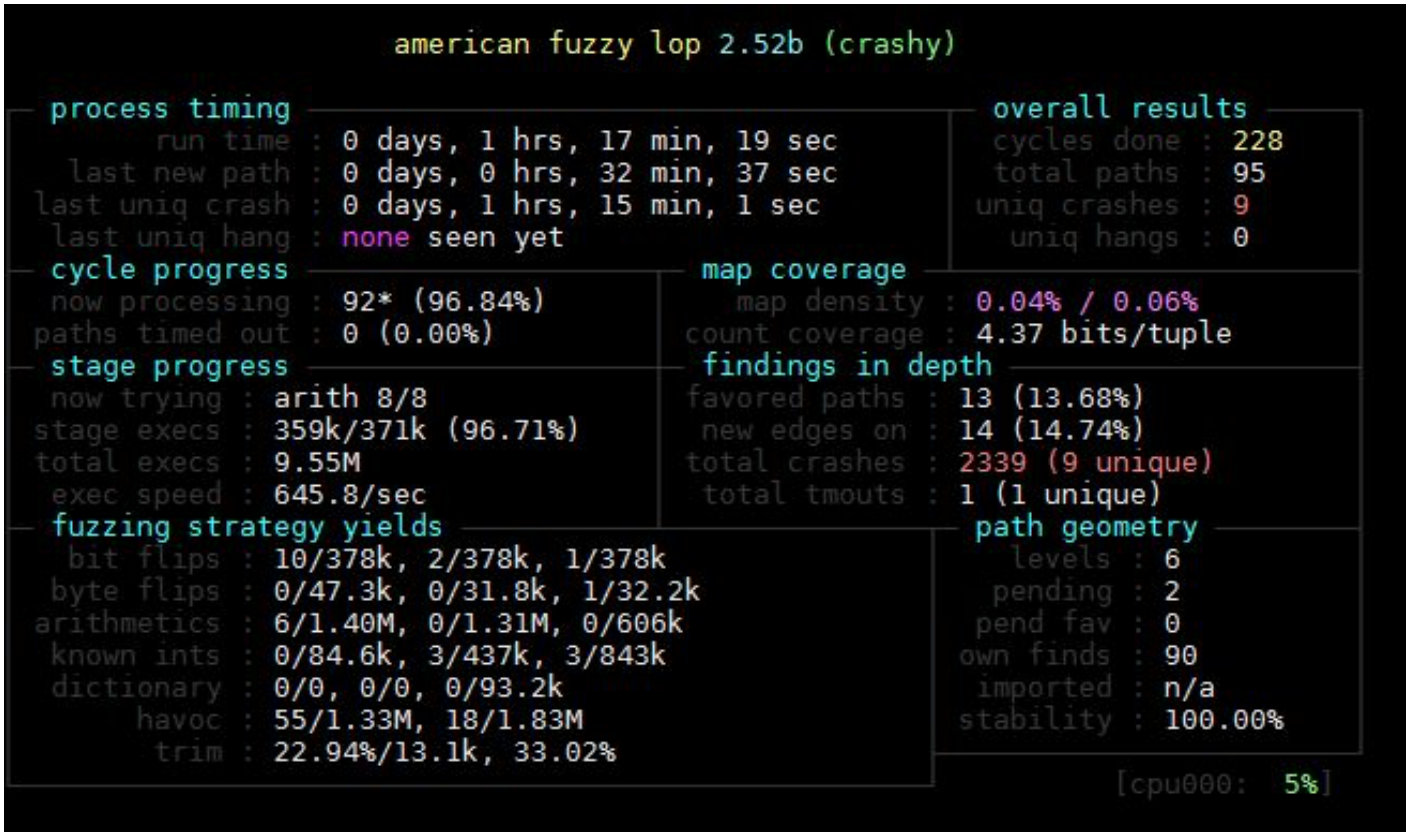
```
xray@xray:~/NSA/Fuzz/crashy$ $HOME/NSA/Fuzz/afl-2.52b/afl-fuzz -i inputFiles -o outputResults ./cras
hy @@
afl-fuzz 2.52b by <lcamtuf@google.com>
[+] You have 20 CPU cores and 2 runnable tasks (utilization: 10%).
[+] Try parallel jobs - see docs/parallel_fuzzing.txt.
[*] Checking CPU core loadout...
[+] Found a free CPU core, binding to #0.
[*] Checking core_pattern...
[*] Setting up output directories...
[+] Output directory exists but deemed OK to reuse.
[*] Deleting old session data...
[+] Output dir cleanup successful.
[*] Scanning 'inputFiles'...
[+] No auto-generated dictionary tokens to reuse.
[*] Creating hard links for all input files...
[*] Validating target binary...
[*] Attempting dry run with 'id:000000,orig:example1'...
    len = 19, map size = 16, exec speed = 199 us
[*] Attempting dry run with 'id:000001,orig:example2'...
    len = 26, map size = 11, exec speed = 169 us
[*] Attempting dry run with 'id:000002,orig:example3'...
    len = 40, map size = 12, exec speed = 174 us
[*] Attempting dry run with 'id:000003,orig:example4'...
    len = 7, map size = 11, exec speed = 169 us
[*] Attempting dry run with 'id:000004,orig:example5'...
    len = 13, map size = 16, exec speed = 172 us
[+] All test cases processed.

[+] Here are some useful stats:

    Test case count : 3 favored, 0 variable, 5 total
       Bitmap range : 11 to 16 bits (average: 13.20 bits)
        Exec timing : 169 to 199 us (average: 177 us)

[*] No -t option specified, so I'll use exec timeout of 20 ms.
[+] All set and ready to roll!
```

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Example #1 - Crashy - 8

- After it starts, AFL will continue until you use **ctrl-c**.



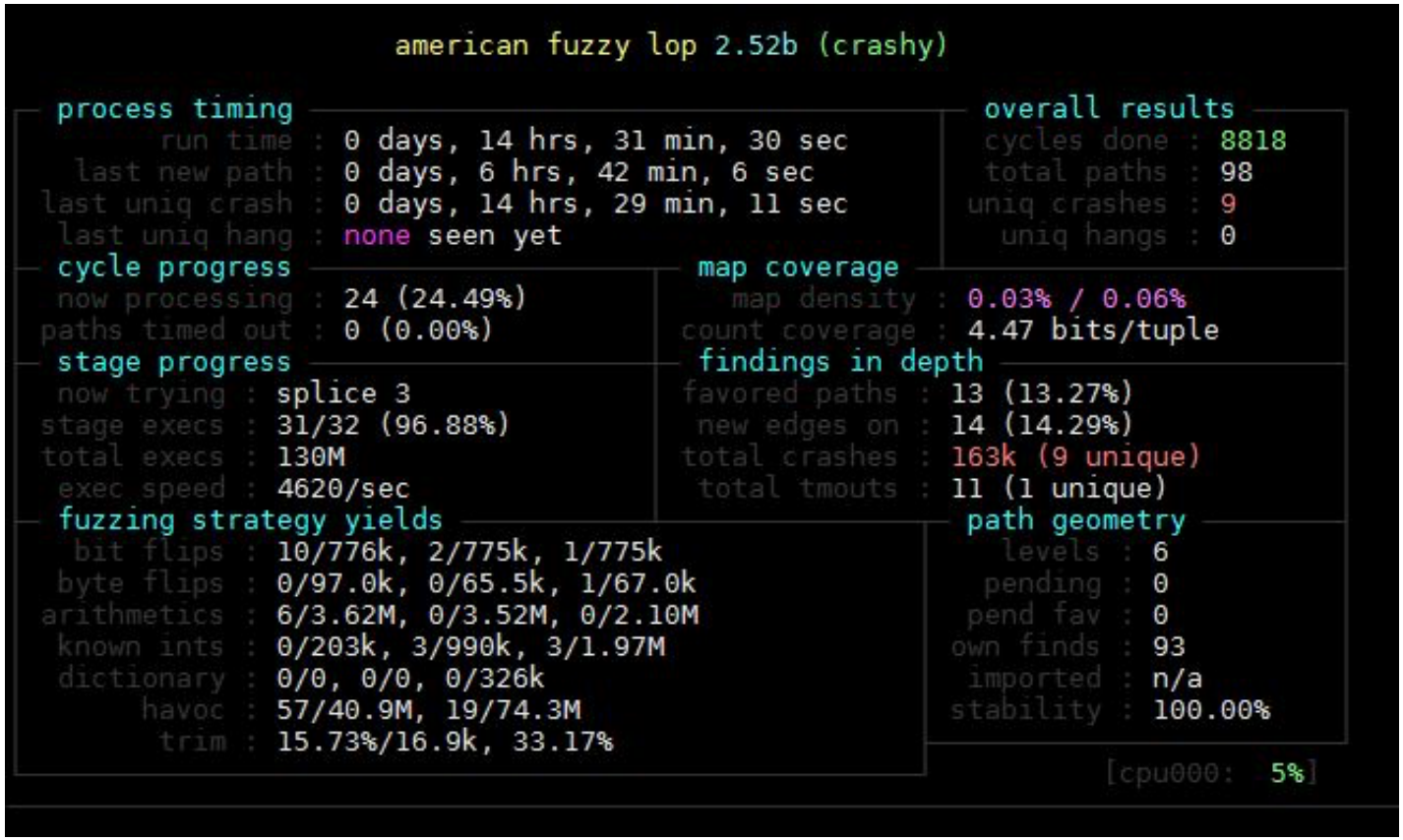
- Crashy after 1 hour and 228 cycles had found 9 unique crashes.

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Example #1 - Crashy - 9

- 14 hours later there were no additional unique crashes.



- Note it had only been half that time since the last new path was found.

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Quiz-Lab05

- Continue running **AFL** on the **crashy** program until you see **unique crashes** identified

- With AFL displaying in **RED** that unique crashes have been identified, **TAKE A SCREENSHOT** for submission to the **Quiz-Lab05**

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Example #1 - Crashy - 10

- Now we review the findings by looking in the output directory.

```
xray@xray:~/NSA/Fuzz/crashy$ cd outputResults/
xray@xray:~/NSA/Fuzz/crashy/outputResults$ ls
crashes  fuzz_bitmap  fuzzer_stats  hangs  plot_data  queue
xray@xray:~/NSA/Fuzz/crashy/outputResults$ cd crashes
xray@xray:~/NSA/Fuzz/crashy/outputResults/crashes$ ls
id:000001,sig:06,src:000005,op:havoc,rep:16
id:000002,sig:11,src:000021,op:havoc,rep:2
id:000003,sig:11,src:000023,op:havoc,rep:2
id:000004,sig:06,src:000002,op:havoc,rep:8
id:000005,sig:11,src:000034,op:int32,pos:2,val:+1000
id:000006,sig:06,src:000051,op:havoc,rep:8
id:000007,sig:11,src:000060+000072,op:splice,rep:2
id:000008,sig:11,src:000021+000072,op:splice,rep:8
README.txt
```

- The inputs that caused crashes are in the **crashes/** folder and are named accordingly.

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Example #1 - Crashy - 11

- We can then confirm that input example will cause a crash by running it directly.

```
xray@xray:~/NSA/Fuzz/crashy$ ./crashy exampleCrash01
Unknown data type '▒'
Unknown data type ''
Unknown data type '4'
Unknown data type 'V'
Unknown data type 'x'
c: 0xcc
s: hello4▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒Vx▒
Unknown data type '4'
Unknown data type '▒'
```

```
Unknown data type '▒'
Unknown data type 'V'
Unknown data type 'x'
c: 0xcc
Segmentation fault
```

- Then debugging can begin using these samples to find the root cause and hopefully correct it.

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Example #2
Common Utility - SQLite

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

Example #2 – SQLite - 1

- The previous example was designed to have faults and to be simple enough for the fuzzer to evaluate quickly
- This example will take a more popular utility, **SQLite**, and show how the fuzzer can be used to test it as well
- To save time, the instructor will demonstrate parallel fuzz testing of **sqlite**

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Discussion

- Once you have sample input files that cause **crashes** and/or **hangs**, you may use **gdb** to determine ***(1) the nature of the defects*** and ***(2) if the defects are exploitable***
- The random nature of **fuzzing** makes it necessary to use other code analysis tools to ensure complete code coverage.
- Still, **fuzzing** is a good addition to a tester's toolkit because the unexpected semi-valid inputs can help find bugs that would have been otherwise missed.

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Summary

- Fuzzing
  - A software testing method that alters inputs to a program in order to see how it performs with unexpected, invalid, and random input.
- AFL
  - A good open-source input file fuzzer that instruments code to enable guided fuzzing for better results.

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

References

- Fuzz Testing of Application Reliability
  http://pages.cs.wisc.edu/~bart/fuzz/
- American Fuzzy Lop (2.52b)
  http://lcamtuf.coredump.cx/afl/
- Crashy - Fuzzing Toy Program Example
  https://?????
- SQLite - version 3.7.17
  https://www.sqlite.org/2013/sqlite-autoconf-3071700.tar.gz

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE