GRAYHAT

Fuzzing and finding vulnerabilities using AFL/WinAFL

Hardik Shah

@hardik05

McAfee™

Fuzzing with AFL

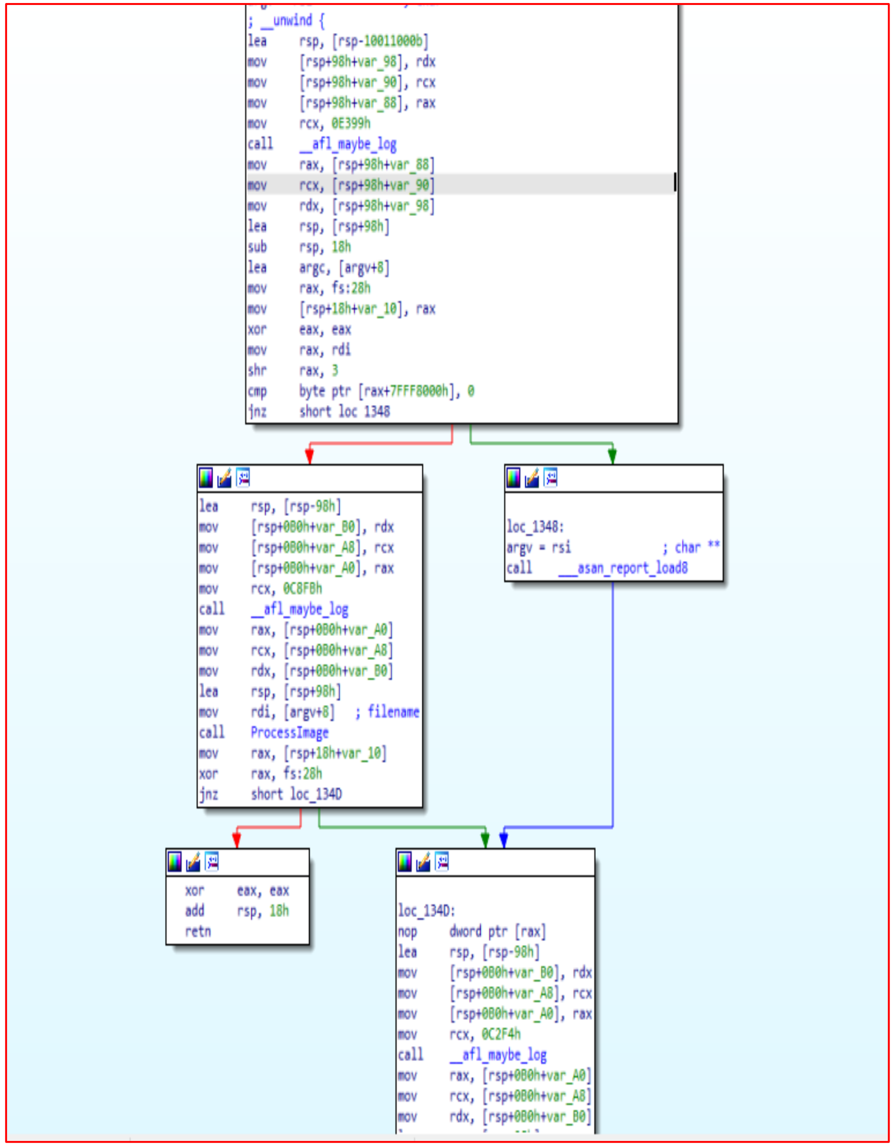
- What is AFL?
- How it works?
- Fork Server vs Persistent mode.
- Fuzzing strategies
- Sanitizers – ASAN,UBSAN, MSAN,TSAN
- Using AFL
    - Hands on: How to compile Sample C program
- Hands on: Fuzzing sample C program with AFL
- Hands on: Root cause analysis
- Hands on: Crash Triage
- Fuzzing real world programs
    - Hands on: Fuzzing open source software with AFL
- Conclusion

What Is AFL?

- American Fuzzy Lop
- Created by Michael Zelwaski
- Fuzzer with instrumentation-guided genetic algorithm.
- Comes with set of utilities:
  - afl-fuzz, afl-cmin, afl-tmin, afl-showmap etc..
- Fork server/Persistent mode.
  - Fork server mode – creates copy of the process
  - Persistent mode – loop around the function.
- Mutate the files based on various strategies.
  - Bitflip, byteflip, havoc, splice etc.

How it works?

- Adds Compile time instrumentation.
- Provides compiler wrappers
    - afl-gcc,afl-g++, afl-clang, afl-clang++, afl-clang-fast, afl-clang-fast++
- uses binary rewriting technique.
    - Add instrumentation at each basic block
        - Each basic block will have a unique random id.
    - Done by assembly equivalent of the following pseudo code:
        - cur_location = <COMPILE_TIME_RANDOM >;
        - shared_mem[cur_location ^ prev_location]++;
        - prev_location = cur_location >> 1;
    - A → B →C → D → E vs A → B → D → C → E



```
; __unwind {
lea     rsp, [rsp-10011000b]
mov     [rsp+98h+var_98], rdx
mov     [rsp+98h+var_90], rcx
mov     [rsp+98h+var_88], rax
mov     rcx, 0E399h
call    __afl_maybe_log
mov     rax, [rsp+98h+var_88]
mov     rcx, [rsp+98h+var_90]
mov     rdx, [rsp+98h+var_98]
lea     rsp, [rsp+98h]
sub     rsp, 18h
lea     argc, [argv+8]
mov     rax, fs:28h
mov     [rsp+18h+var_10], rax
xor     eax, eax
mov     rax, rdi
shr     rax, 3
cmp     byte ptr [rax+7FFF8000h], 0
jnz     short loc 1348
```

```
lea     rsp, [rsp-98h]
mov     [rsp+0B0h+var_B0], rdx
mov     [rsp+0B0h+var_A8], rcx
mov     [rsp+0B0h+var_A0], rax
mov     rcx, 0C8FBh
call    __afl_maybe_log
mov     rax, [rsp+0B0h+var_A0]
mov     rcx, [rsp+0B0h+var_A8]
mov     rdx, [rsp+0B0h+var_B0]
lea     rsp, [rsp+98h]
mov     rdi, [argv+8]   ; filename
call    ProcessImage
mov     rax, [rsp+18h+var_10]
xor     rax, fs:28h
jnz     short loc_134D
```

```
loc_1348:
argv = rsi              ; char **
call    __asan_report_load8
```

```
xor     eax, eax
add     rsp, 18h
retn
```

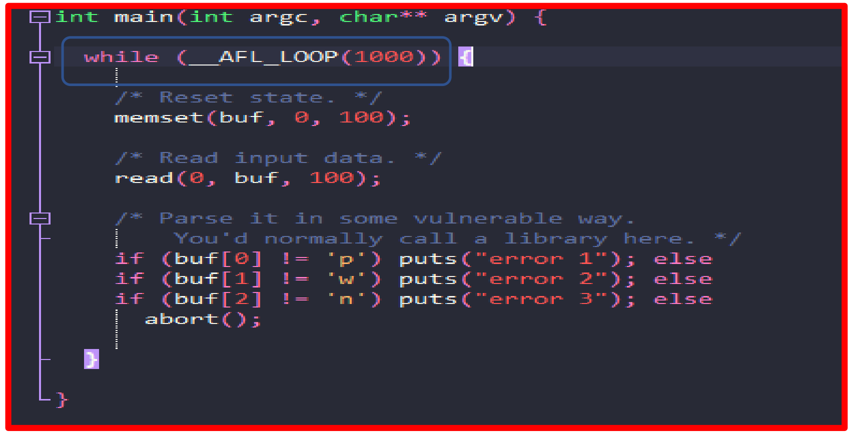
```
loc_134D:
nop     dword ptr [rax]
lea     rsp, [rsp-98h]
mov     [rsp+0B0h+var_B0], rdx
mov     [rsp+0B0h+var_A8], rcx
mov     [rsp+0B0h+var_A0], rax
mov     rcx, 0C2F4h
call    __afl_maybe_log
mov     rax, [rsp+0B0h+var_A0]
mov     rcx, [rsp+0B0h+var_A8]
mov     rdx, [rsp+0B0h+var_B0]
```

Fork Server Vs Persistent Mode

- Fork Server Mode
  - Stop at main().
  - Uses fork to create clone of the program.
  - Process input and create another clone.
  - Saves time in initializing program and thus offer speed improvements.

Ref: https://lcamtuf.blogspot.com/2014/10/fuzzing-binaries-without-execve.html

- Persistent Mode
  - Fork is still costly.
  - Don’t really need to kill child process after each run.
  - Uses in process Fuzzing.
  - Need to write a harness program.
  - Ex:



```
int main(int argc, char** argv) {
  while (__AFL_LOOP(1000)) {
    /* Reset state. */
    memset(buf, 0, 100);

    /* Read input data. */
    read(0, buf, 100);

    /* Parse it in some vulnerable way.
       You'd normally call a library here. */
    if (buf[0] != 'p') puts("error 1"); else
    if (buf[1] != 'w') puts("error 2"); else
    if (buf[2] != 'n') puts("error 3"); else
      abort();

  }
}
```

- Ref: https://lcamtuf.blogspot.com/2015/06/new-in-afl-persistent-mode.html

Fuzzing Strategies

- **Bitflip** – flips a bit i.e. 1 becomes 0, 0 becomes 1
    - 1/1,2/1,4/1,8/8 ….32/8
- **Byte Flip** – flips a byte
- **Arithmetic** –random arithmetic like plus/minus
- **Havoc** – random things with bit/bytes/addition/subtraction
- **Dictionary** – user provided dictionary or auto discovered tokens.
    - Over/insert/over(autodetected)
- **Interest** - replace content in original file with interesting values
    - 0xff,0x7f etc – 8/8,16/8..
- **Splice** – split and combine two or more files to get a new file.
- Ref: https://github.com/google/AFL/blob/master/docs/technical_details.txt

Sanitizers

Sanitizers

- Tools based on compiler instrumentation.
- Helpful for identifying bugs.
- Can discover bugs like large memory allocations, heap overflow, use after free etc.
- Different types of sanitizers
  - ASAN
  - MSAN
  - UBSAN
  - TSAN

Address Sanitizer (ASAN)

- Compiler directive : **-fsanitize=address**
- Can detect various issues like UAF, Heap Buffer overflow, Memory Leaks etc..
- ASAN + Fuzzer = More bugs!

Ref:https://clang.llvm.org/docs/AddressSanitizer.html

- Use After Free vulnerabilities
- Heap Buffer Overflows
- Stack Buffer Overflows
- Initialization order bugs
- Memory Leaks
- Use after scope

Undefined Behavior Sanitizer (UBSAN)

- Detects undefined behavior in the program
  - Divide by zero, integer overflow, uninitialized reads etc.
- Compiler directive : **-fsanitize=undefined**
- improved bug finding capabilities

Ref: https://clang.llvm.org/docs/UndefinedBehaviorSanitizer.html

Null Pointer Dereferences

Signed Integer Overflows

Typecast Overflows

Divide by Zero errors

Memory Sanitizer(MSAN) and Thread Sanitizer(TSAN)

- Memory Sanitizer (MSAN)
  - Compiler directive : **-fsanitize=memory**
  - Detects uninitialized reads etc.
- Thread Sanitizer (TSAN)
  - Compiler directive : **-fsanitize=thread**
  - Detects data races etc.

Ref: https://clang.llvm.org/docs/MemorySanitizer.html

Ref: https://clang.llvm.org/docs/ThreadSanitizer.html

What we have learned So far?

What is AFL, How it works?
Fuzzing strategies.
Different sanitizers and how to enable them.

Hands on

Hands on : Compiling and installing AFL

- **git clone https://github.com/google/AFL.git**
- **make**
- **cd llvm_mode**
- **make** → need clang installed
- **cd ..**
- **sudo make install**

Hands on: How to compile program with AFL?

- afl-clang -fsanitize=address imgRead.c -g -o imgReadafl
  - afl-clang -> compiler wrapper for gcc, this will compile and instrument the binary.
  - -fsanitize address -> enables asan[can also use AFL_USE_ASAN=1]
  - -g -> debugging symbols support
  - imgRead.c -> source file.
  - imgReadafl -> generated executable file which will be fuzzed.

Hands on: Fuzzing Sample C program with AFL

- Generate Input
  - **echo “IMG” > input/1.img**
- **afl-fuzz –i input –o output –m none -- ./imgRead @@**
  - **afl-fuzz** -> fuzzer binary.
  - **-i** -> directory containing input seed files.
  - **-o** -> directory containing output data from fuzzer
    - Crashes -> contains input files which crashes target program.
    - Hangs -> contains input files which causes hangs for target program.
  - **-m** -> memory limit, if ASAN and 64 bit, set it to none
    - Else compile it in 32 bit using compiler flag –m32
    - Set memory limit as –m 800
    - Find more crashes.
  - **-M** -> Master instance, in case you have multiple CPU core.
  - **-S** -> Slave instance, can be n- number depending on the cores you have.

Hands on: Root Cause analysis

- Lets use GDB to analyze crashes.
- Commands:
  - **Gdb <exe file name>**
  - **r** -> run the program
  - s -> step over
  - **Next/fi** -> execute till return
  - **b** <filename.c:linenumber> -> puts breakpoint in filename.c at linenumber
- In our case **gdb ./imgread**
  - r <output/crashes/filename>

Hands on: Crash Triage

- Crashwalk is a useful tool to triage crashes if you get lot of crashes.
- Installing crashwalk
  - **sudo apt-get install golang**
  - **go get -u github.com/bnagy/crashwalk/cmd/...**
  - **~/go/bin**
- Installing exploitable
  - **~/src/exploitable/exploitable/exploitable.py**
  - **mkdir ~/src**
  - **cd ~/src**
  - **git clone https://github.com/jfoote/exploitable.git**

Hands on: Crash Triage

- Cwtriage – utility to triage crashes
  - **ASAN_OPTIONS="abort_on_error=1:symbolize=0" Cwtriage –afl –root output**
  - Analyzes each crash file and saves results in crashwalk.db.
  - Run with ASAN else crash will not get replicate.
- Cwdump – utility to dump crash info from crashwalk.db
  - Cwdump crashwalk.db

What we have learned So far?

How to fuzz simple program using AFL on linux.
Root cause analysis using gdb.
Crash triaging.

Fuzzing open source softwares

Hands on: Fuzzing tcpdump

- Get the source code of tcpdump and libpcap.
  - **git clone https://github.com/the-tcpdump-group/tcpdump.git**
  - **cd tcpdump**
  - **git clone https://github.com/the-tcpdump-group/libpcap.git**
  - **cd libpcap**
- Compile it using AFL
  - **CC=afl-gcc CFLAGS="-g -fsanitize=address -fno-omit-frame-pointer" LDFLAGS="-g -fsanitize=address -fno-omit-frame-pointer" ./configure**
  - **sudo make && make install**
- Corpus?
  - Check tests folder ☺
  - Minimise it: **afl-cmin –i tests –o mincorpus –m none -- ./tcpdump –vv –ee –nnr @@**
- Fuzz it
  - **afl-fuzz –I mincorpus –o fuzzoutput –m none -- ./tcpdump –vv –ee –nnr @@**

Hands on: Fuzzing libtiff

- Get the source code from here:
  - https://gitlab.com/libtiff/libtiff
- Compile it using AFL
  - **./autogen.sh**
  - **CC=afl-gcc CXX=afl-g++ CFLAGS="-g -fsanitize=address -fno-omit-frame-pointer" CXXFLAGS="-g -fsanitize=address -fno-omit-frame-pointer" LDFLAGS="-g -fsanitize=address -fno-omit-frame-pointer" ./configure**
  - **sudomake && make install**
- get the corpus
  - https://lcamtuf.coredump.cx/afl/demo/
- Minimize it
  - **afl-cmin –i tiff –o mintiff -- ./tiff2rgba @@ test.tiff**
- Fuzz it
  - **afl-fuzz –i <input> -o fuzzoutput –m none -- tiff2rgba @@ test.tiff**

Conclusion

- Fuzzing on linux with AFL is simple.
- Source code is available for most of the libs/software.
- Compile time instrumentation is available.
- Use ASAN,MSAN,UBSAN for fuzzing.
- Sometime requires efforts in compilation.
    - Missing libraries
    - Compilation errors
    - Worth learning.

GRAYHAT