

AFL下载

在microsoft store上，用WSL Ubuntu上下载AFL。

有个坑：不要直接下载Ubuntu,应该下载它的LTS版本。（直接下载后，连make命令都用不了）



下载AFL:

```
wget https://lcamtuf.coredump.cx/afl/releases/afl-latest.tgz
```

这个命令下得太慢了。推荐直接去github的镜像站下载

```
sudo apt install gcc
sudo apt install git
git clone https://gitcode.com/gh_mirrors/af/AFL.git
#克隆成功后，安装
cd AFL
make
sudo make install
#安装完成后，检测
afl-gcc --version
```

出现下面这个就是成功了。

```
xiaoyu@LAPTOP-256NC7NO:~/AFL$ afl-gcc --version
afl-cc 2.57b by <lcamtuf@google.com>
gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

测试代码

在ubuntu系统中用nano创建测试代码

```
xiaoyu@LAPTOP-256NC7N0:~$ cd Work/afl-project/  
xiaoyu@LAPTOP-256NC7N0:~/Work/afl-project$ nano test.c  
xiaoyu@LAPTOP-256NC7N0:~/Work/afl-project$
```

参考：[我的AFL入门之路 - 知乎 \(zhihu.com\)](https://zhuanlan.zhihu.com/p/100000000)

```
#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
#include <string.h>  
#include <signal.h>  
  
int vuln(char *str) {  
    int len = strlen(str);  
    if (str[0] == 'A' && len == 66) {  
        // 如果输入的字符串的首字符为A并且长度为66，则异常退出  
        raise(SIGSEGV);  
    } else if (str[0] == 'F' && len == 6) {  
        // 如果输入的字符串的首字符为F并且长度为6，则异常退出  
        raise(SIGSEGV);  
    } else {  
        printf("it is good!\n");  
    }  
    return 0;  
}  
  
int main(int argc, char *argv[]) {  
    char buf[100] = {0};  
    gets(buf); // 存在栈溢出漏洞。如果输入过长，则会导致栈溢出  
    printf(buf); // 存在格式化字符串漏洞。没有指定格式字符串，如果buf包含格式化指令，则会导致未定义行为  
    vuln(buf);  
    return 0;  
}
```

- 如果输入的字符串的首字符为A并且长度为66，则异常退出
- 如果输入的字符串的首字符为F并且长度为6，则异常退出
- 使用gets函数从标准输入获取buf内容，存在栈溢出漏洞
- 使用printf输出buf内容，存在格式化字符串漏洞

流程概述：

1. 首先是用afl-gcc编译源代码，然后以文件(最好小于1K)为输入。
2. 然后启动afl-fuzz程序，将testcase(输入的测试文件)作为程序的输入执行程序，afl会在这个testcase的基础上进行自动变异输入，使得程序产生crash，产生了crash就会被记录起来。

1. 编译与模糊测试

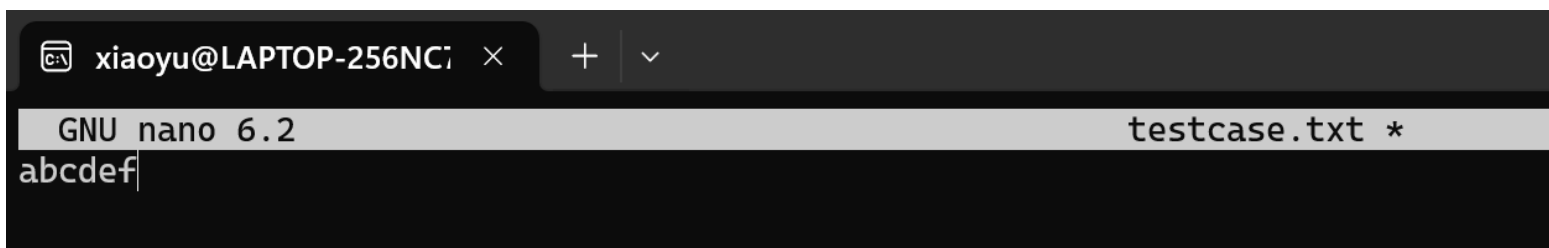
对test.c进行编译。（如果是编译一个c的源码，那就需要用afl-g）

```
afl-gcc -g -o ./afl_example ./test.c
```

```
xiaoyu@LAPTOP-256NC7N0:~/Work/afl-project$ mkdir fuzz_in fuzz_out
xiaoyu@LAPTOP-256NC7N0:~/Work/afl-project$ ls
fuzz_in  fuzz_out  test.c
xiaoyu@LAPTOP-256NC7N0:~/Work/afl-project$ afl-gcc -g -o ./afl_example ./test.c
afl-cc 2.57b by <lcamtuf@google.com>
./test.c: In function 'main':
./test.c:23:5: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
 23 |     gets(buf); // 存在栈溢出漏洞。如果输入过长，则会导致栈溢出
    |     ^~~~~
    |     fgets
./test.c:24:12: warning: format not a string literal and no format arguments [-Wformat-security]
 24 |     printf(buf); // 存在格式化字符串漏洞。没有指定格式字符串，如果buf包含格式化指令，则会导致未定义行为
    |
afl-as 2.57b by <lcamtuf@google.com>
[+] Instrumented 8 locations (64-bit, non-hardened mode, ratio 100%).
/usr/bin/ld: /tmp/ccwTTw4m.o: in function 'main':
/home/xiaoyu/Work/afl-project/./test.c:23: warning: the 'gets' function is dangerous and should not be used.
```

建立两个文件夹：fuzz_in和fuzz_out，用来存放程序的输入和fuzz的输出结果。

其中，从fuzz_in中读取输入，输出放入fuzz_out中，afl_example是要进行模糊测试的程序。



```
#进行模糊测试
afl-fuzz -i fuzz_in -o fuzz_out ./afl_example -f
```

可能会出现这个错误：

```
xiaoyu@LAPTOP-256NC7N0:~/Work/afl-project$ afl-fuzz -i fuzz_in -o fuzz_out ./afl_example -f
afl-fuzz 2.57b by <lcamtuf@google.com>
[+] You have 20 CPU cores and 2 runnable tasks (utilization: 10%).
[+] Try parallel jobs - see /usr/local/share/doc/afl/parallel_fuzzing.txt.
[*] Checking CPU core loadout...
[+] Found a free CPU core, binding to #0.
[*] Checking core_pattern...

[-] Hmm, your system is configured to send core dump notifications to an
external utility. This will cause issues: there will be an extended delay
between stumbling upon a crash and having this information relayed to the
fuzzer via the standard waitpid() API.

To avoid having crashes misinterpreted as timeouts, please log in as root
and temporarily modify /proc/sys/kernel/core_pattern, like so:

echo core >/proc/sys/kernel/core_pattern

[-] PROGRAM ABORT : Pipe at the beginning of 'core_pattern'
    Location : check_crash_handling(), afl-fuzz.c:7347
```

解决办法：

```
sudo su
echo core >/proc/sys/kernel/core_pattern
#检查是否解决 如果输出core 则已经设置成功
```

```
cat /proc/sys/kernel/core_pattern  
#ctrl d 退出后 再次执行之前的命令即可
```

开始进行模糊测试

```
xiaoyu@LAPTOP-256NC7NO:~/Work/afl-project$ afl-fuzz -i fuzz_in -o fuzz_out ./afl_example -f  
afl-fuzz 2.57b by <lcamtuf@google.com>  
[+] You have 20 CPU cores and 2 runnable tasks (utilization: 10%).  
[+] Try parallel jobs - see /usr/local/share/doc/afl/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 'fuzz_in'...  
[+] No auto-generated dictionary tokens to reuse.  
[*] Creating hard links for all input files...  
[*] Validating target binary...  
[*] Attempting dry run with 'id:000000,orig:testcase.txt'...  
[*] Spinning up the fork server...  
[+] All right - fork server is up.  
    len = 7, map size = 6, exec speed = 195 us  
[+] All test cases processed.  
  
[+] Here are some useful stats:  
  
    Test case count : 1 favored, 0 variable, 1 total  
    Bitmap range   : 6 to 6 bits (average: 6.00 bits)  
    Exec timing    : 195 to 195 us (average: 195 us)  
  
[*] No -t option specified, so I'll use exec timeout of 20 ms.  
[+] All set and ready to roll!
```

american fuzzy lop 2.57b (afl_example)

process timing		overall results	
run time : 0 days, 0 hrs, 3 min, 17 sec		cycles done : 1684	
last new path : 0 days, 0 hrs, 3 min, 17 sec		total paths : 3	
last uniq crash : 0 days, 0 hrs, 1 min, 41 sec		uniq crashes : 5	
last uniq hang : none seen yet		uniq hangs : 0	
cycle progress		map coverage	
now processing : 2 (66.67%)		map density : 0.01% / 0.01%	
paths timed out : 0 (0.00%)		count coverage : 1.00 bits/tuple	
stage progress		findings in depth	
now trying : havoc		favored paths : 3 (100.00%)	
stage execs : 150/256 (58.59%)		new edges on : 3 (100.00%)	
total execs : 1.51M		total crashes : 2368 (5 unique)	
exec speed : 7471/sec		total tmouts : 2 (1 unique)	
fuzzing strategy yields		path geometry	
bit flips : 2/120, 0/117, 0/111		levels : 2	
byte flips : 0/15, 0/12, 0/6		pending : 0	
arithmetics : 1/840, 0/25, 0/0		pend fav : 0	
known ints : 0/77, 0/332, 0/264		own finds : 2	
dictionary : 0/0, 0/0, 0/0		imported : n/a	
havoc : 4/1.51M, 0/0		stability : 100.00%	
trim : 28.57%/3, 0.00%			
[cpu000: 14%]			

等待运行一段时间，收集到一定的错误后进行分析。

process timing 展示当前fuzzer的运行时间、最近一次发现新执行路径的时间、最近一次崩溃的时间、最近一次超时的时间。值得注意的是第2项，最近一次发现新路径的时间。如果由于目标二进制文件或者命令行参数出错，那么其执行路径应该是一直不变的，所以如果从fuzzing开始一直没有发现新的执行路径，那么就要考虑是否有二进制或者命令行参数错误的问题了。对于此状况，AFL也会智能地进行提醒。

overall results 这里包括运行的总周期数、总路径数、崩溃次数、超时次数。其中，总周期数可以用来作为何时停止fuzzing的参考。随着不断地fuzzing，周期数会不断增大，其颜色也会由洋红色，逐步变为黄色、蓝色、绿色。一般来说，当其变为绿色时，代表可执行的内容已经很少了，继续fuzzing下去也不会有什么新的发现了。此时，我们便可以通过Ctrl-C，中止当前的fuzzing。

stage progress 这里包括正在测试的fuzzing策略、进度、目标的执行总次数、目标的执行速度。执行速度可以直观地反映当前跑的快不快，如果速度过慢，比如低于500次每秒，那么测试时间会变得非常漫长。如果发生了这种情况，我们需要进一步优化我们的Fuzzing。以上是简单的介绍，如果要看完整的可以查看[官方文档](#)。

american fuzzy lop 2.57b (afl_example)

process timing		overall results	
run time : 0 days, 0 hrs, 11 min, 55 sec		cycles done : 6164	
last new path : 0 days, 0 hrs, 11 min, 55 sec		total paths : 3	
last uniq crash : 0 days, 0 hrs, 10 min, 19 sec		uniq crashes : 5	
last uniq hang : none seen yet		uniq hangs : 0	
cycle progress		map coverage	
now processing : 0 (0.00%)		map density : 0.01% / 0.01%	
paths timed out : 0 (0.00%)		count coverage : 1.00 bits/tuple	
stage progress		findings in depth	
now trying : havoc		favored paths : 3 (100.00%)	
stage execs : 294/384 (76.56%)		new edges on : 3 (100.00%)	
total execs : 5.53M		total crashes : 113k (5 unique)	
exec speed : 7406/sec		total touts : 2 (1 unique)	
fuzzing strategy yields		path geometry	
bit flips : 2/120, 0/117, 0/111		levels : 2	
byte flips : 0/15, 0/12, 0/6		pending : 0	
arithmetics : 1/840, 0/25, 0/0		pend fav : 0	
known ints : 0/77, 0/332, 0/264		own finds : 2	
dictionary : 0/0, 0/0, 0/0		imported : n/a	
havoc : 4/5.53M, 0/0		stability : 100.00%	
trim : 28.57%/3, 0.00%			
^C		[cpu000: 15%]	

+++ Testing aborted by user +++

[+] We're done here. Have a nice day!

运行这段时间后 产生了5个crashes

```
xiaoyu@LAPTOP-256NC7N0:~/Work/afl-project/fuzz_out/crashes$ ls
README.txt                                id:000002,sig:11,src:000001,op:havoc,rep:4
id:000000,sig:11,src:000000,op:arith8,pos:0,val:-27 id:000003,sig:06,src:000001,op:havoc,rep:128
id:000001,sig:06,src:000000,op:havoc,rep:32      id:000004,sig:11,src:000001,op:havoc,rep:8
```

用XXD工具分析：

```
xiaoyu@LAPTOP-256NC7N0:~/Work/afl-project/fuzz_out/crashes$ xxd id:000002,sig:11,src:000001,op:havoc,rep:4
00000000: 4177 2573 6364 6364                Aw%scdcd
xiaoyu@LAPTOP-256NC7N0:~/Work/afl-project/fuzz_out/crashes$ xxd id:000000,sig:11,src:000000,op:arith8,pos:0,val:-27
00000000: 4662 6364 6566 0a                  Fbcdef.
xiaoyu@LAPTOP-256NC7N0:~/Work/afl-project/fuzz_out/crashes$ xxd id:000001,sig:06,src:000000,op:havoc,rep:32
00000000: 2c7f ef2c 2c4b 2c2c 2c2c 2cff 2c2c 402c      ,...K,.....@,
00000010: 2c2c 2c2c 2c2c 422c 2c2c 2c2c 2c7f ffff      ,...B,.....
00000020: 2c2c 2c7f ff2c 2c7f ffff 2c2c 2c7f 4b2c      ,...K,
00000030: 2c4b 2c2c 2c2c 2cff 2c2c 2c2c 2c38 122c      ,K,.....8.,
00000040: 2c2c 5200 ff2c 2c2c ff2c 2c2c 2c2c 2c2e      ,R.....
00000050: 2c2c 2c2c 2c2c 2c2c 2c2c 2c2c 2c2c 2c2c      ,.....
00000060: 2c2c 2c2c 2c2c 2c2c 402c 2c2c 2c2c 2c7f      ,.....@,.....
xiaoyu@LAPTOP-256NC7N0:~/Work/afl-project/fuzz_out/crashes$ xxd id:000003,sig:06,src:000001,op:havoc,rep:128
00000000: e7e7 e7e0 e726 00e7 e7e8 0700 e7e7 e7f6      .....&.....
00000010: 0000 ffff e7e7 e7e7 e7e7 e7e7 e7e7 dbef      .....
00000020: e7e7 e7e7 67e7 e7e7 e7e7 e7db e7e7 e7e7      ....g.....
00000030: e7e7 ffff e7db e7e7 e7e7 e7e7 e7e7 dbef      .....
00000040: e7e7 e7e7 67e7 e7e7 e7e7 e7db e7e7 e7e7      ....g.....
00000050: e7e7 e7e7 e7e7 e7e7 e7e7 ffe7 e700 7fe7      .....
00000060: e7e7 e7e7 e7e7 e7e7 a9              .....
xiaoyu@LAPTOP-256NC7N0:~/Work/afl-project/fuzz_out/crashes$ xxd id:000004,sig:11,src:000001,op:havoc,rep:8
00000000: 4162 6341 4141 4141 4141 4141 4141 4141      AbcAAAAAAAAAAAA
00000010: 4141 4131 4150 0202 0202 0202 0202 0202      AAA1AP.....
00000020: 0202 0202 0202 0202 0202 0202 0202 0202      .....
00000030: 0241 4141 4141 4141 4141 4141 4141 4110      .AAAAAAAAAAAAAA.
00000040: 4141                                AA
```

000000 :

这种情况符合我们在vuln中规定的：如果输入的字符串的首字符为F并且长度为6，则异常退出。

000001 :

这种情况的输入数据长度为 $9*16+3=147$ 字节，超出了buf的长度从而导致了栈溢出错误。

000002 :

这种情况的输入中包含%S这一printf输入控制符，导致了printf的错误，存在格式化字符串漏洞。

000003 :

这种情况的输入数据长度为 $7*16=112$ 字节，超出了buf的长度从而导致了栈溢出错误。

000004 :

这种情况符合我们在vuln中规定的：如果输入的字符串的首字符为A并且长度为66，则异常退出。

AFL的模糊测试基本找出了我们给定代码中的一些漏洞。