AFL——覆盖引导的模糊测试技术

孙以晨 2016级信息安全 201611123009

一. 实验目的

- AFL（American Fuzzy Lop）通过记录输入样本的代码覆盖率，从而调整输入样本以提高覆盖率，增加发现漏洞的概率。
- 其工作流程大致如下：

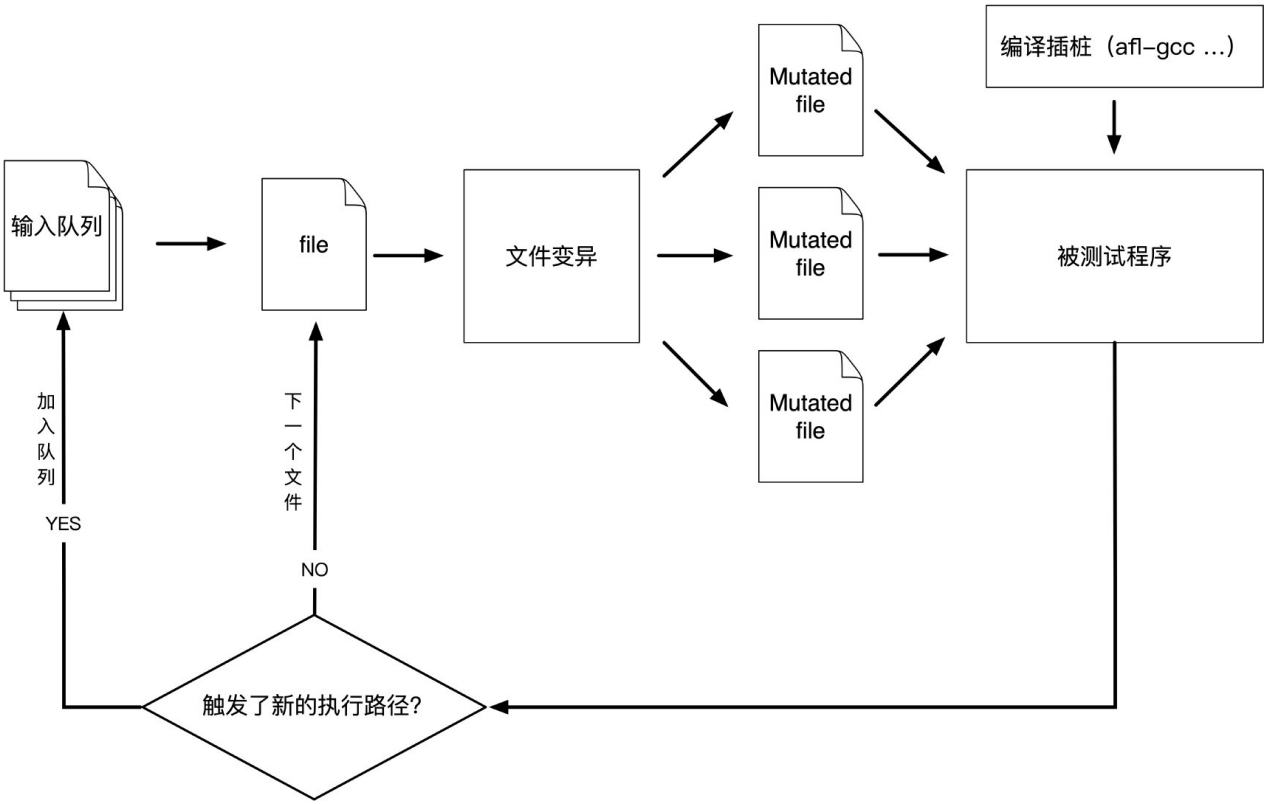
  ①从源码编译程序时进行插桩，以记录代码覆盖率（Code Coverage）；

  ②选择一些输入文件，作为初始测试集加入输入队列（queue）；

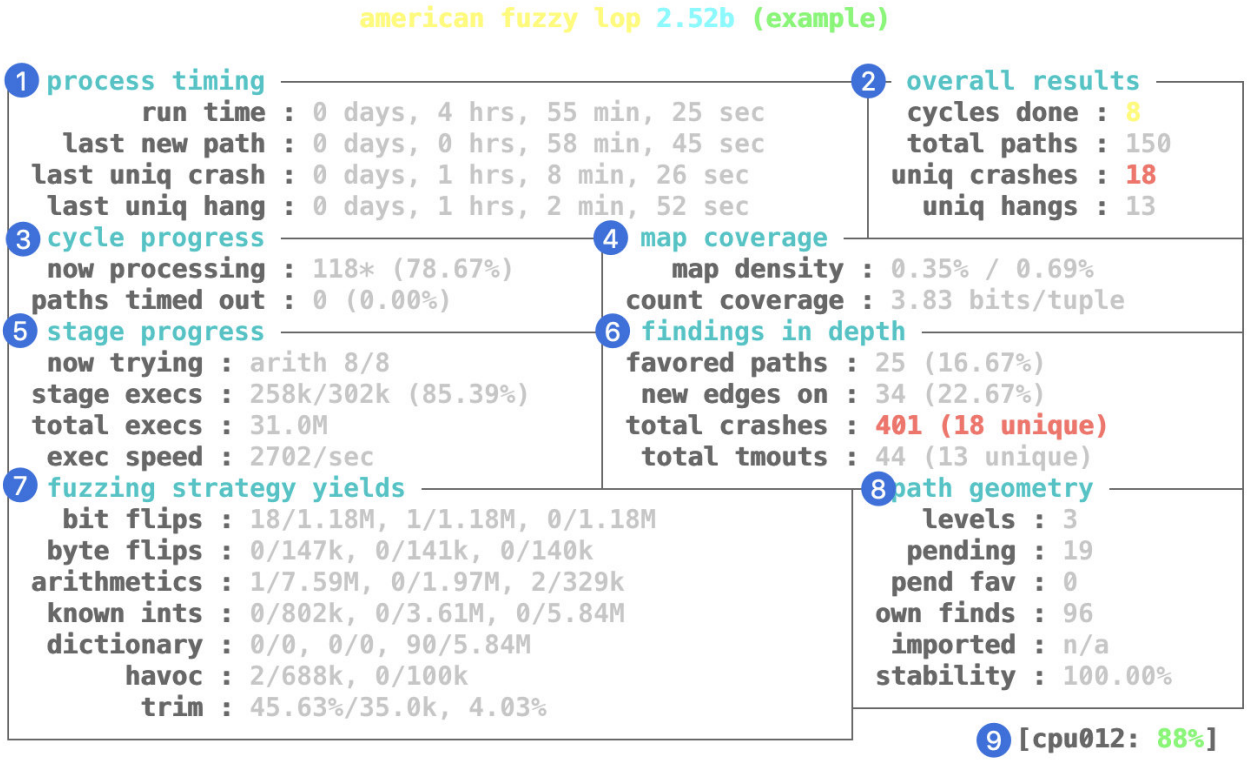
  ③将队列中的文件按一定的策略进行“突变”；

  ④如果经过变异文件更新了覆盖范围，则将其保留添加到队列中;

  ⑤上述过程会一直循环进行，期间触发了crash的文件会被记录下来。



- AFL语料库的构建。AFL需要一些初始输入数据（也叫种子文件）作为Fuzzing的起点，这些输入甚至可以是毫无意义的数据，AFL可以通过启发式算法自动确定文件格式结构。
- AFL状态窗口。



① Process timing:Fuzzer运行时长、以及距离最近发现的路径、崩溃和挂起经过了多长时间。

② Overall results：Fuzzer当前状态的概述。

③ Cycle progress：我们输入队列的距离。

④ Map coverage：目标二进制文件中的插桩代码所观察到覆盖范围的细节。

⑤ Stage progress：Fuzzer现在正在执行的文件变异策略、执行次数和执行速度。

⑥ Findings in depth：有关我们找到的执行路径，异常和挂起数量的信息。

⑦ Fuzzing strategy yields：关于突变策略产生的最新行为和结果的详细信息。

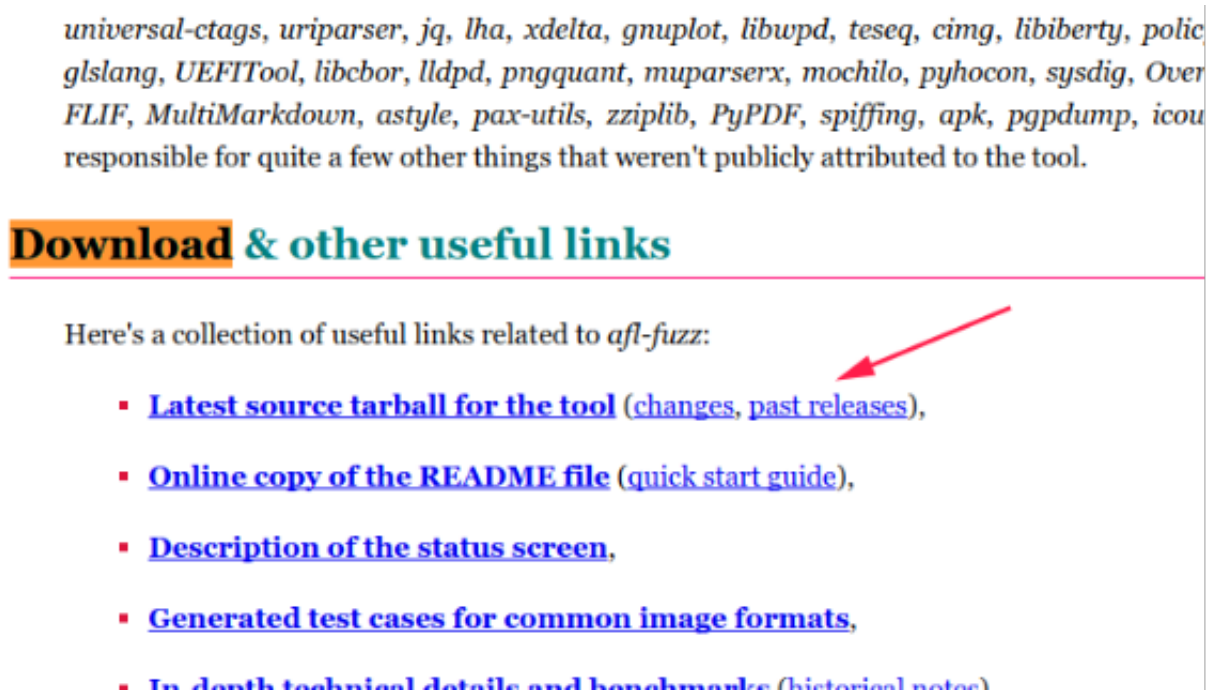
⑧ Path geometry：有关Fuzzer找到的执行路径的信息。

⑨ CPU load：CPU利用率

二. 用AFL进行模糊测试

2.1 安装AFL

2.1.1 下载最新源码



2.1.2 解压并安装

```
make
sudo make install
```

```
cookie@cookie-VirtualBox:/$ cd '/home/cookie/afl-2.52b'
cookie@cookie-VirtualBox:~/afl-2.52b$ make
[*] Checking for the ability to compile x86 code...
afl-cc 2.52b by <lcamtuf@google.com>
afl-as 2.52b by <lcamtuf@google.com>
[+] Instrumented 1 locations (64-bit, non-hardened mode, ratio 100%).
[+] Everything seems to be working, ready to compile.
afl-gcc -O3 -funroll-loops -Wall -D_FORTIFY_SOURCE=2 -g -Wno-pointer-sign -
DAFL_PATH=\"/usr/local/lib/afl\" -DDOC_PATH=\"/usr/local/share/doc/afl\" -D
BIN_PATH=\"/usr/local/bin\" afl-gcc.c -o afl-gcc -ldl
afl-cc 2.52b by <lcamtuf@google.com>
afl-as 2.52b by <lcamtuf@google.com>
[+] Instrumented 100 locations (64-bit, non-hardened mode, ratio 100%).
set -e; for i in afl-g++ afl-clang afl-clang++; do ln -sf afl-gcc $i; done
afl-gcc -O3 -funroll-loops -Wall -D_FORTIFY_SOURCE=2 -g -Wno-pointer-sign -
DAFL_PATH=\"/usr/local/lib/afl\" -DDOC_PATH=\"/usr/local/share/doc/afl\" -D
BIN_PATH=\"/usr/local/bin\" afl-fuzz.c -o afl-fuzz -ldl
afl-cc 2.52b by <lcamtuf@google.com>
afl-as 2.52b by <lcamtuf@google.com>
[+] Instrumented 3923 locations (64-bit, non-hardened mode, ratio 100%).
afl-gcc -O3 -funroll-loops -Wall -D_FORTIFY_SOURCE=2 -g -Wno-pointer-sign -
DAFL_PATH=\"/usr/local/lib/afl\" -DDOC_PATH=\"/usr/local/share/doc/afl\" -D
BIN_PATH=\"/usr/local/bin\" afl-showmap.c -o afl-showmap -ldl
afl-cc 2.52b by <lcamtuf@google.com>
afl-as 2.52b by <lcamtuf@google.com>
[+] Instrumented 247 locations (64-bit, non-hardened mode, ratio 100%).
afl-gcc -O3 -funroll-loops -Wall -D_FORTIFY_SOURCE=2 -g -Wno-pointer-sign -
```

```
cookie@cookie-VirtualBox:~/afl-2.52b$ sudo make install
[*] Checking for the ability to compile x86 code...
[+] Everything seems to be working, ready to compile.
[*] Testing the CC wrapper and instrumentation output...
unset AFL_USE_ASAN AFL_USE_MSAN; AFL_QUIET=1 AFL_INST_RATIO=100 AFL_PATH=.
./afl-gcc -O3 -funroll-loops -Wall -D_FORTIFY_SOURCE=2 -g -Wno-pointer-sign
 -DAFL_PATH=\"/usr/local/lib/afl\" -DDOC_PATH=\"/usr/local/share/doc/afl\"
-DBIN_PATH=\"/usr/local/bin\" test-instr.c -o test-instr -ldl
echo 0 | ./afl-showmap -m none -q -o .test-instr0 ./test-instr
echo 1 | ./afl-showmap -m none -q -o .test-instr1 ./test-instr
[+] All right, the instrumentation seems to be working!
[+] All done! Be sure to review README - it's pretty short and useful.
NOTE: If you can read this, your terminal probably uses white background.
This will make the UI hard to read. See docs/status_screen.txt for advice.
mkdir -p -m 755 ${DESTDIR}/usr/local/bin ${DESTDIR}/usr/local/lib/afl ${DES
TDIR}/usr/local/share/doc/afl ${DESTDIR}/usr/local/share/afl
rm -f ${DESTDIR}/usr/local/bin/afl-plot.sh
install -m 755 afl-gcc afl-fuzz afl-showmap afl-tmin afl-gotcpu afl-analyze
 afl-plot afl-cmin afl-whatsup ${DESTDIR}/usr/local/bin
rm -f ${DESTDIR}/usr/local/bin/afl-as
```

如果不报错，afl-fuzz就安装成功了

```
cookie@cookie-VirtualBox:~$ afl-fuzz
afl-fuzz 2.52b by <lcamtuf@google.com>

afl-fuzz [ options ] -- /path/to/fuzzed_app [ ... ]

Required parameters:

  -i dir        - input directory with test cases
  -o dir        - output directory for fuzzer findings

Execution control settings:

  -f file       - location read by the fuzzed program (stdin)
  -t msec       - timeout for each run (auto-scaled, 50-1000 ms)
  -m megs       - memory limit for child process (50 MB)
  -Q            - use binary-only instrumentation (QEMU mode)

Fuzzing behavior settings:

  -d            - quick & dirty mode (skips deterministic steps)
  -n            - fuzz without instrumentation (dumb mode)
  -x dir        - optional fuzzer dictionary (see README)

Other stuff:
```

2.1.3 AFL Fuzz UPX

有源码的afl-fuzz，这里以fuzz upx为例进行测试。

编译upx

- upx项目地址(https://github.com/upx/upx)
- 因为afl会对有源码的程序进行重新编译,因此需要修改upx的Makefile

```
$git clone https://github.com/upx/upx.git
$cd upx
$vim Makefile
CC = /usr/local/bin/afl-gcc #添加此句
```

```
#
# UPX toplevel Makefile - needs GNU make 3.81 or better
#
# Copyright (C) 1996-2018 Markus Franz Xaver Johannes Oberhumer
#

MAKEFLAGS += -rR
.SUFFIXES:
export SHELL = /bin/sh
CC = /usr/local/bin/afl-gcc #添加此句

srcdir = .
top_srcdir = .
include $(wildcard $(top_srcdir)/Makevars.global ./Makevars.local)

# info: src/stub needs special build tools from https://github.com/upx/upx-
stubtools
BUILD_STUB = 0
ifneq ($(wildcard $(HOME)/local/bin/bin-upx/upx-stubtools-check-version),)
BUILD_STUB = 1
endif
ifneq ($(wildcard $(HOME)/bin/bin-upx/upx-stubtools-check-version),)
BUILD_STUB = 1
-- INSERT --                                              11,1          Top
```

```
$cd src
$vim Makefile
CXX    ?= /usr/local/bin/afl-g++ #将CXX改成afl-g++
```

```
override ee = $($1) $(EXTRA_$1) $(upx_$1) $(upx_EXTRA_$1) $($(basename $(notdir
$@)).$1)

ifndef srcdir
srcdir := $(dir $(lastword $(MAKEFILE_LIST)))
srcdir := $(shell echo '$(srcdir)' | sed 's,/*$$,,' || echo 'ERROR')
endif
ifndef top_srcdir
top_srcdir := $(srcdir)/..
endif
include $(wildcard $(top_srcdir)/Makevars.global ./Makevars.local)
ifneq ($(srcdir),.)
vpath %.cpp .:$(srcdir)
vpath %.h   .:$(srcdir)
endif

# toolchain
CXX    ?= /usr/local/bin/afl-g++ #将CXX改成afl-g++
CXXLD   = $(CXX)
exeext ?= .out
libext ?= .a
objext ?= .o

"Makefile" 176L, 6102C                                      31,54-51      9%
```

通过upx的文档,还需要安装三个库

- 安装lzma-sdk

```
$git submodule update --init --recursive
```

- 安装ucl

```
# 下载ucl
bash wget http://www.oberhumer.com/opensource/ucl/download/ucl-1.03.tar.gz
# 编译
$cd ucl-1.03
$./configure
$make
$sudo make install

$export UPX_UCCLDIR="~/ucl-1.03"
```

- 安装zlib

```
  $wget http://pkgs.fedoraproject.org/repo/pkgs/zlib/zlib-
1.2.11.tar.xz/sha512/b7f50ada138c7f93eb7eb1631efccd1d9f03a5e77b6c13c8b757017b2
d462e19d2d3e01c50fad60a4ae1bc86d431f6f94c72c11ff410c25121e571953017cb67/zlib-
1.2.11.tar.xz
  $cd zlib-1.2.11/
  $./configure
  $sudo make install
```

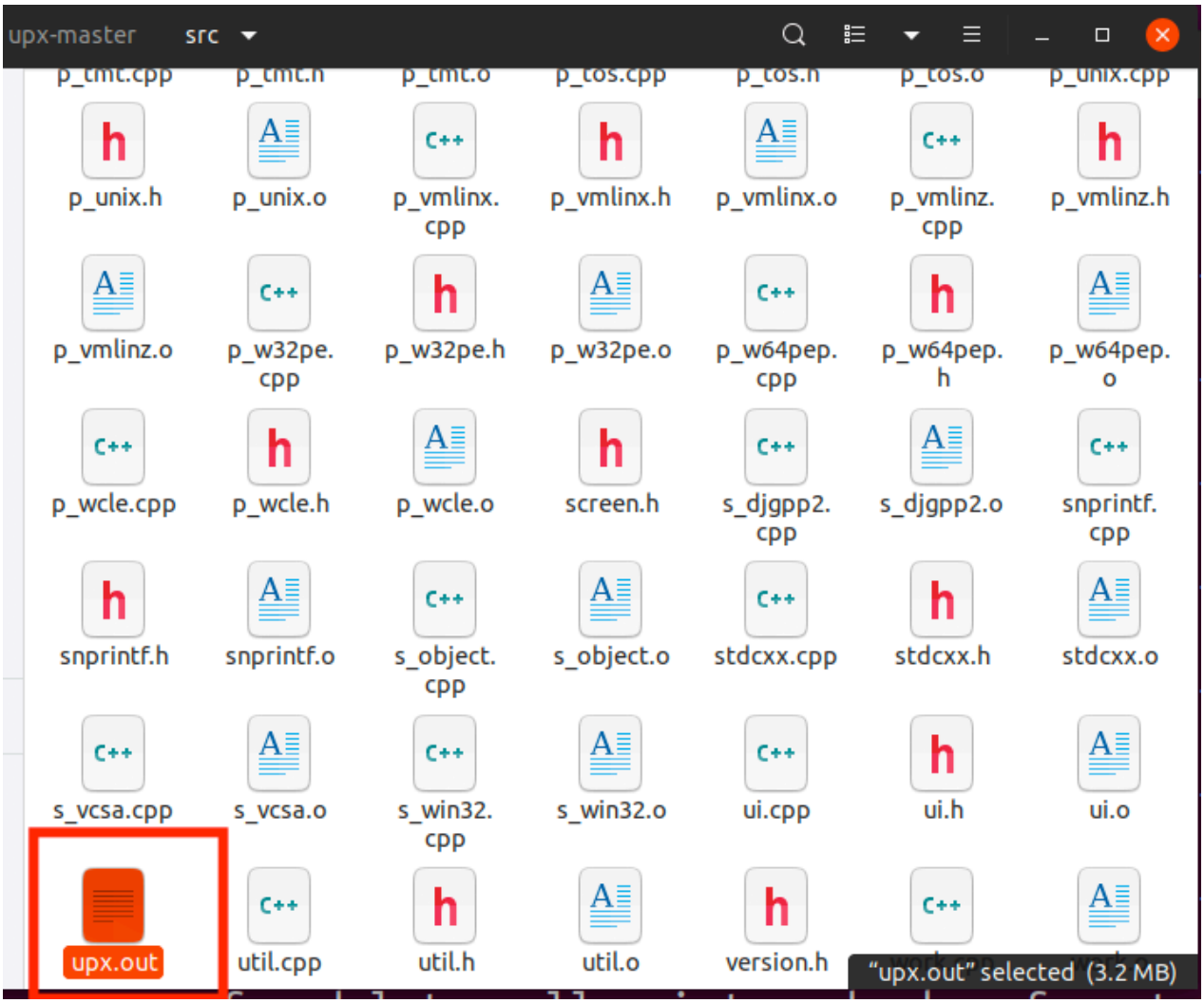
```
cookie@cookie-VirtualBox:~/upx-master/src$ sudo apt install zlib1g-dev
[sudo] password for cookie:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  fonts-liberation2 fonts-opensymbol gir1.2-geocodeglib-1.0 gir1.2-gst-plugins-base-1.0 gir1.2-gstreamer-1.0
  gir1.2-gudev-1.0 gir1.2-udisks-2.0 grilo-plugins-0.3-base gstreamer1.0-gtk3 guile-2.2-libs libboost-date-time1.67.0
  libboost-filesystem1.67.0 libboost-iostreams1.67.0 libboost-locale1.67.0 libcdr-0.1-1 libclucene-contribs1v5
  libclucene-core1v5 libcmis-0.5-5v5 libcolamd2 libcurl4 libdazzle-1.0-0 libe-book-0.1-1 libeot0 libepubgen-0.1-1
  libetonyek-0.1-1 libevent-2.1-6 libfreerdp-client2-2 libfreerdp2-2 libgee-0.8-2 libgom-1.0-0 libgpgmepp6
  libgpod-common libgpod4 liblangtag-common liblangtag1 liblirc-client0 liblua5.3-0 libmediaart-2.0-0 libminiupnpc17
  libmspub-0.1-1 libodfgen-0.1-1 liborcus-0.14-0 libqqwing2v5 libraw19 librevenge-0.0-0 libsgutils2-2
  libsuitesparseconfig5 libvncclient1 libwinpr2-2 libxmlsec1-nss lp-solve media-player-info python3-mako
  python3-markupsafe syslinux syslinux-common syslinux-legacy usb-creator-common
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  zlib1g-dev
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 172 kB of archives.
After this operation, 437 kB of additional disk space will be used.
Get:1 http://cn.archive.ubuntu.com/ubuntu disco/main amd64 zlib1g-dev amd64 1:1.2.11.dfsg-1ubuntu2 [172 kB]
Fetched 172 kB in 9s (19.3 kB/s)
Selecting previously unselected package zlib1g-dev:amd64.
(Reading database ... 165638 files and directories currently installed.)
Preparing to unpack .../zlib1g-dev_1%3a1.2.11.dfsg-1ubuntu2_amd64.deb ...
Unpacking zlib1g-dev:amd64 (1:1.2.11.dfsg-1ubuntu2) ...
Setting up zlib1g-dev:amd64 (1:1.2.11.dfsg-1ubuntu2) ...
Processing triggers for man-db (2.8.5-2) ...
```

编译upx

```
$cd ~/upx
$make all
```

若没有报错,则编译成功

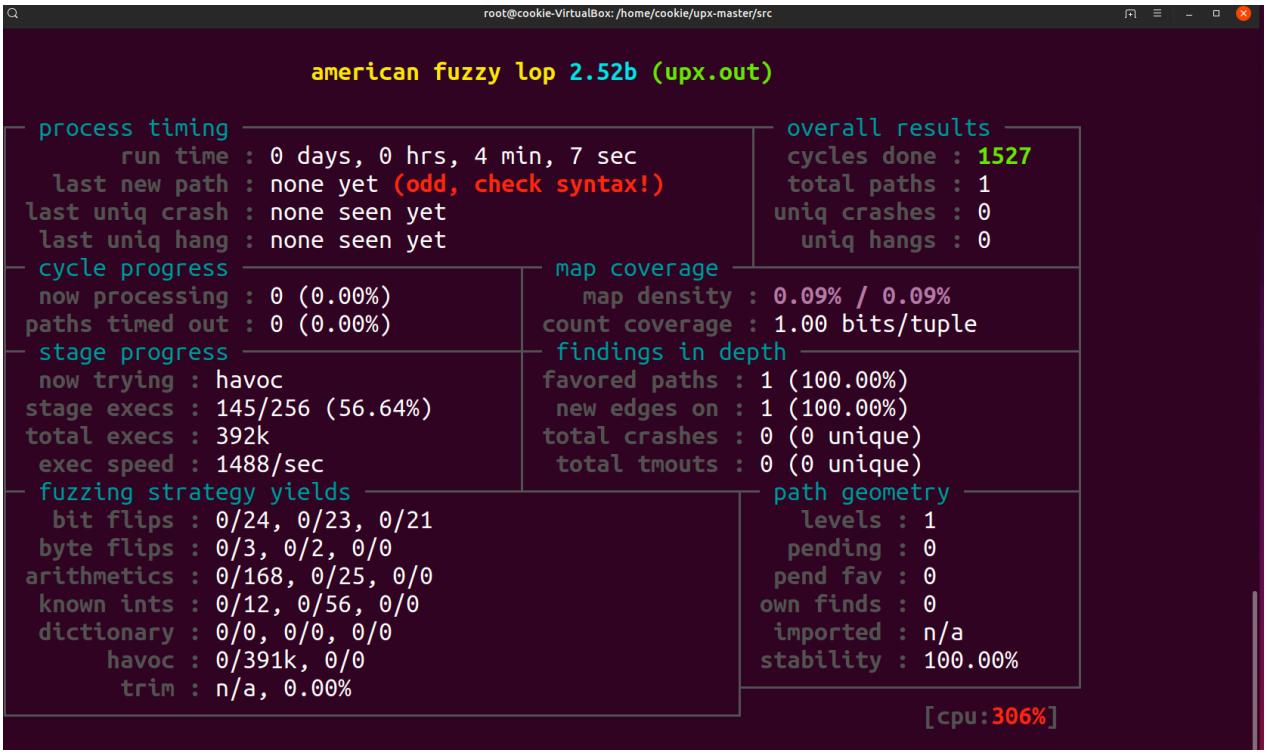
此时可在/src目录下找到upx.out文件



对upx进行fuzz测试

```
cookie@cookie-VirtualBox:~/upx-master/src$ afl-fuzz -i fuzz_input -o fuzz_output ./upx.out -1 @@ -o /dev/null
afl-fuzz 2.52b by <lcamtuf@google.com>
[+] You have 1 CPU core and 1 runnable tasks (utilization: 100%).
[*] 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:7275
```

AFL运行界面



三. AFL源码分析

代码覆盖率是一种度量代码的覆盖程度的方式，也就是指源代码中的某行代码是否已执行；对二进制程序可将此概念理解为汇编代码中的某条指令是否已执行。对模糊测试来说希望每句代码都能被检测到，覆盖率越高越好。

覆盖率的计量方式主要为三种：**函数**，**基本块**，**边界**。（插桩是为了覆盖率而实行的方法。）

afl-gcc.c

afl-gcc是gcc的一个封装，主要有以下三个功能：

```
find_as(argv[0]);                       //找到gcc/clang/llvm编译器
edit_params(argc, argv);                 //处理参数
execvp(cc_params[0], (char**)cc_params);//执行
```

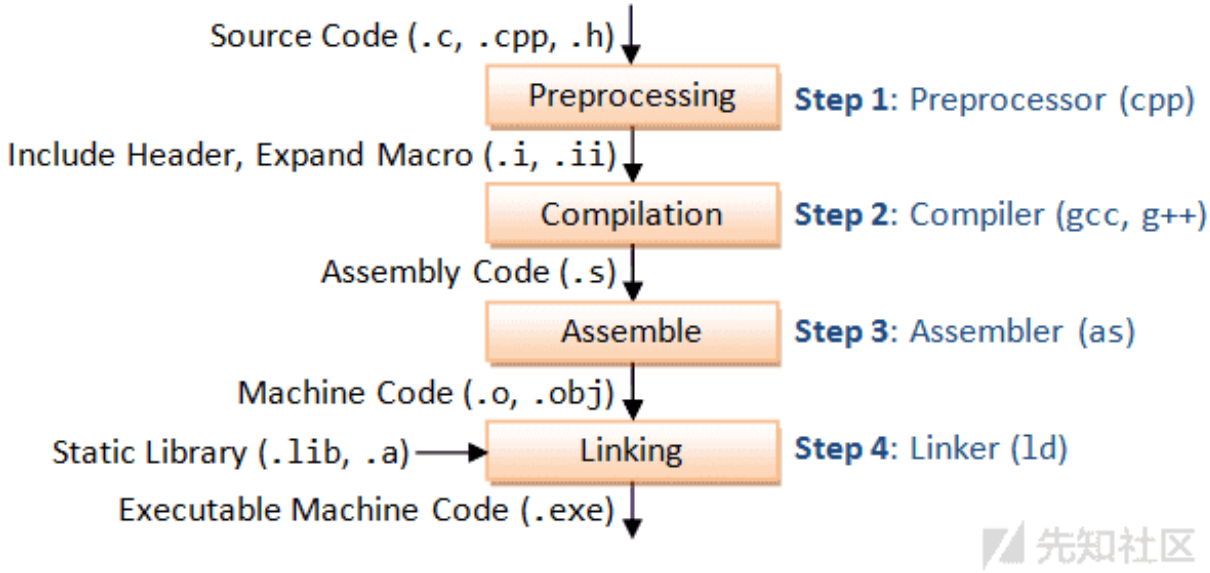
打印出`cc_params`,看看真正的参数是什么

```
gcc -o test test.c -B /usr/local/lib/afl -g -O3 -funroll-loops -
D__AFL_COMPILER=1 -DFUZZING_BUILD_MODE_UNSAFE_FOR_PRODUCTION=1
```

看看参数的意思。用了编译优化，指定了编译的标志，最终要的是`-B`指定了**编译器**(Assembler)

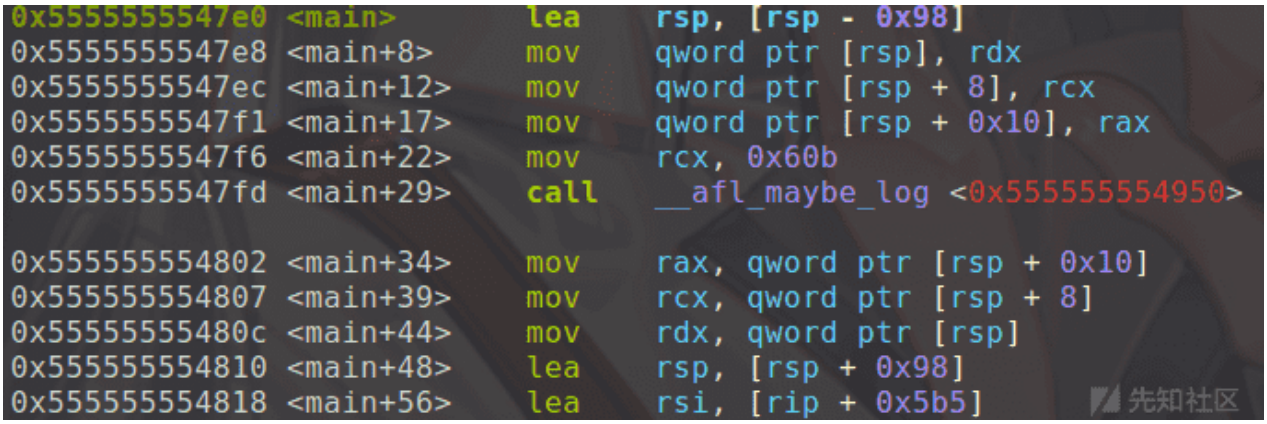
```
-funroll-loops          执行循环强度消除并消除在循环内部使用的变量。这是用简单而快速的操作（如加法和减法）替代耗时操作（如乘法和除法）的过程
-B <目录>               将 <目录> 添加到编译器的搜索路径中
#ifdef FUZZING_BUILD_MODE_UNSAFE_FOR_PRODUCTION     /* a flag also shared with libfuzzer) or */
#ifdef __AFL_COMPILER                               /* (this one is just for AFL). */
```

这一步正是汇编文件通过as进一步编译成二进制文件，这里替换了Assembler，当然为了插桩



afl-as.c和afl-as.h

反汇编刚才生成的test,会发现插了一些插入了额外的汇编指令



这两个文件被单独提出来可以来解释这里是怎么操作的

```
The sole purpose of this wrapper is to preprocess assembly files generated by GCC / clang and inject the instrumentation bits included from afl-as.h. It is automatically invoked by the toolchain when compiling programs using afl-gcc / afl-clang.
```

主要是处理不同平台设置标志，处理参数等等.重要函数`add_instrumentation`

fprintf将插桩用的汇编用`fprintf`插如合适的地方

```
static  void  add_instrumentation (void) {
    ......
    while (fgets(line, MAX_LINE, inf)) {//读取每行
    ......
      fprintf(outf, use_64bit ? trampoline_fmt_64 : trampoline_fmt_32,
              R(MAP_SIZE));//插入，注意R(MAP_SIZE)
    ......
        //下面有意思是怎么判断合适的地方插入，选择分支，有兴趣自己看看
    }
}
```

下面分别是32位和64位的，和调试看的一样

```
static const u8* trampoline_fmt_32 =

  "\n"
  "/* --- AFL TRAMPOLINE (32-BIT) --- */\n"
  "\n"
  ".align 4\n"
  "\n"
  "leal -16(%%esp), %%esp\n"     //太高栈
  "movl %%edi,  0(%%esp)\n"      //保存寄存器
  "movl %%edx,  4(%%esp)\n"
  "movl %%ecx,  8(%%esp)\n"
  "movl %%eax, 12(%%esp)\n"
  "movl $0x%08x, %%ecx\n"        //保存随机数
  "call __afl_maybe_log\n"       //调用__afl_maybe_log
  "movl 12(%%esp), %%eax\n"
  "movl  8(%%esp), %%ecx\n"
  "movl  4(%%esp), %%edx\n"
  "movl  0(%%esp), %%edi\n"
  "leal 16(%%esp), %%esp\n"
  "\n"
  "/* --- END --- */\n"
  "\n";

static const u8* trampoline_fmt_64 =

  "\n"
  "/* --- AFL TRAMPOLINE (64-BIT) --- */\n"
  "\n"
  ".align 4\n"
  "\n"
  "leaq -(128+24)(%%rsp), %%rsp\n"
  "movq %%rdx,  0(%%rsp)\n"
  "movq %%rcx,  8(%%rsp)\n"
```

```
  "movq %%rax, 16(%%rsp)\n"
  "movq $0x%08x, %%rcx\n"
  "call __afl_maybe_log\n"
  "movq 16(%%rsp), %%rax\n"
  "movq  8(%%rsp), %%rcx\n"
  "movq  0(%%rsp), %%rdx\n"
  "leaq (128+24)(%%rsp), %%rsp\n"
  "\n"
  "/* --- END --- */\n"
  "\n";
```

所以能看到，插桩是为了统计覆盖率。