

软件安全实验报告

学号: 2310764 姓名: 王亦辉 班次: 计科一班

1 实验名称 :

AFL 模糊测试

2 实验要求 :

根据课本 7.4.5 章节, 复现 AFL 在 Kali 下的安装、应用, 查阅资料理解覆盖引导和文件变异的概念和含义。

3 实验过程 :

3.1 安装 Kali

首先在官网下载给虚拟机预构建的 Kali 操作系统, 我们选择的是 VMware 的版本。[Get Kali | Kali Linux](#)

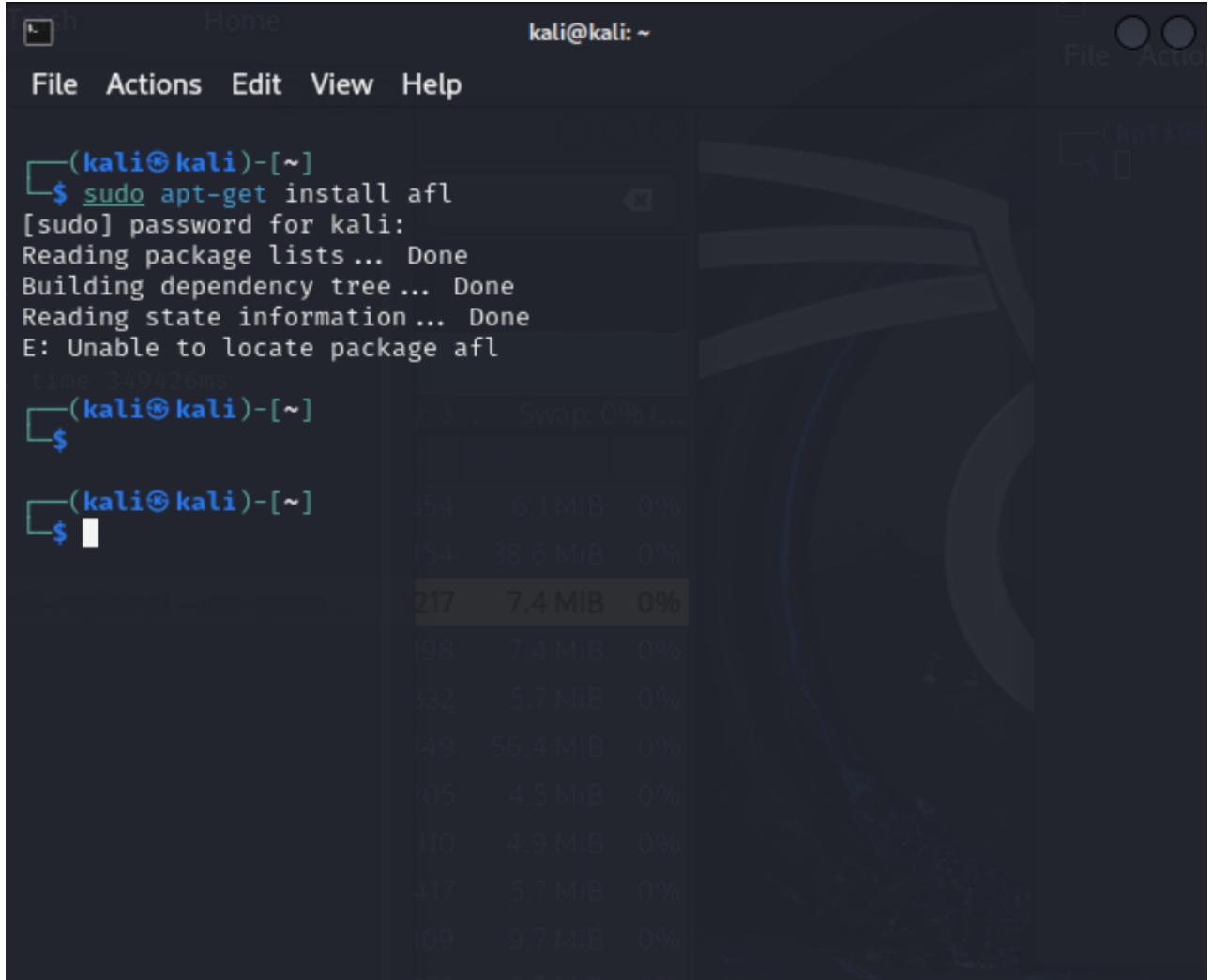
随后按照官网的教程进行操作系统的导入。[Import Pre-Made Kali VMware VM | Kali Linux Documentation](#)

使用 kali 作为用户名和密码登录

3.2 复现 AFL 在 KALI 下的安装、应用

3.2.1 在安装之前, 配置网络

由于下载 AFL 需要使用 `sudo apt-get install afl` 命令, 直接运行, 可以看到无法获取到包, 这是网络问题, 接下来我们尝试连接主机的代理。



```
(kali㉿kali)-[~]
$ sudo apt-get install afl
[sudo] password for kali:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
E: Unable to locate package afl
time 349426ms
```

```
(kali㉿kali)-[~]
$
```

```
(kali㉿kali)-[~]
$
```

	Size	Filesystem	Used	Available	Capacity	% Used
217	7.4 MiB	0%				
198	7.4 MiB	0%				
332	5.7 MiB	0%				
149	56.4 MiB	0%				
105	4.5 MiB	0%				
110	4.9 MiB	0%				
417	5.7 MiB	0%				
109	9.7 MiB	0%				

我下载的 Kali 虚拟机中已经内置了 [proxychains4](#) 工具，我们使用此工具配置代理。

我们运行 `sudo vim /etc/proxchains4.conf` 打开该工具的配置文件，然后在 ProxyList 最后，将 `socks4` 那行注释掉(没用)。再加上一行，`socks5 192.168.23.1 7897`。其中，`192.168.23.1` 是在主机上用 `ipconfig` 查到的 VMnet1 的 `IPV4` 地址；`7897` 是主机上启用的代理的监听端口。使用 socks5 代理协议。参考[教程1](#) [教程2](#)

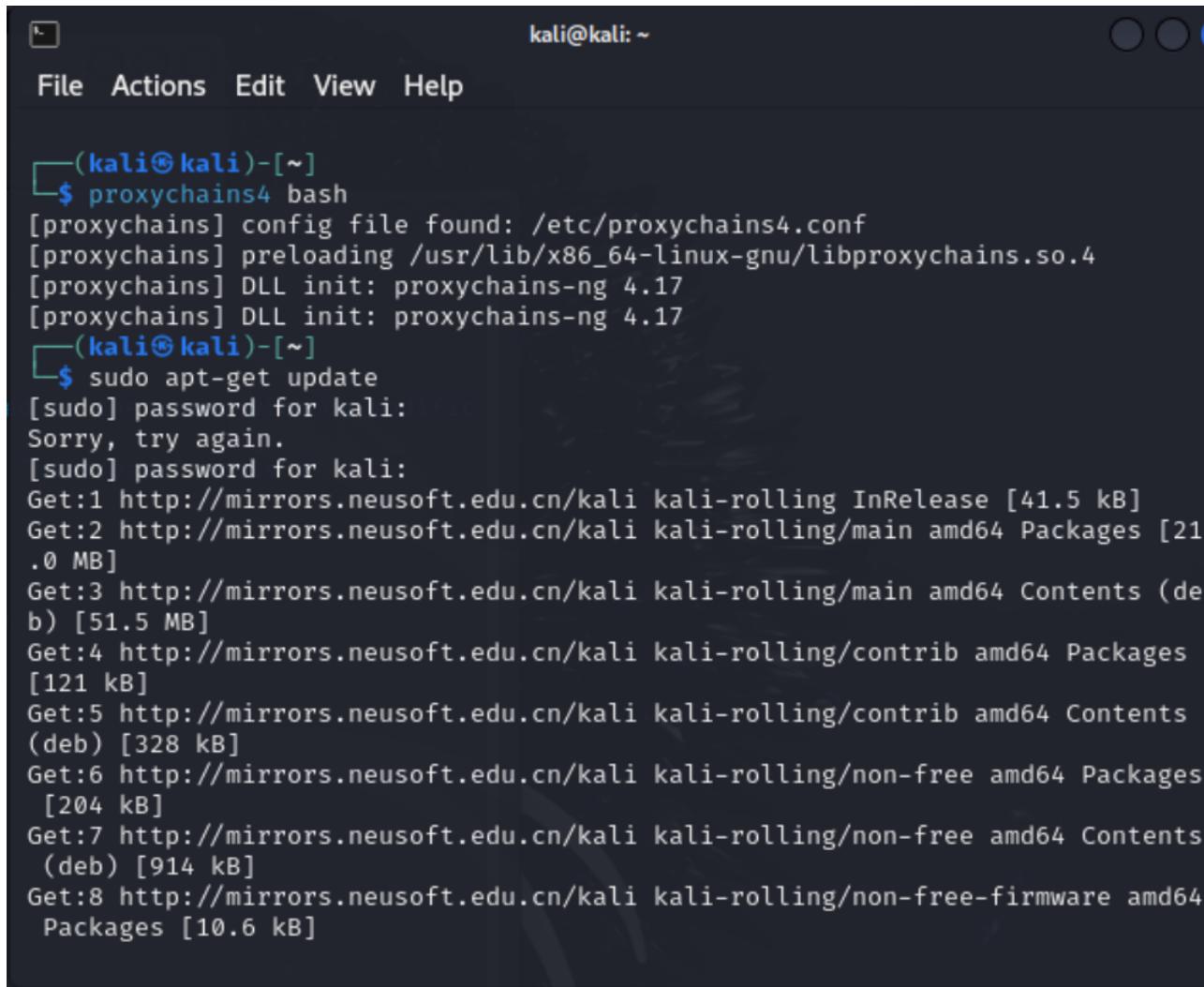
```
kali@kali: ~
File Actions Edit View Help
#
#      [~]
#      get update
#      proxy types: http, socks4, socks5, raw
#          * raw: The traffic is simply forwarded to the proxy without modification.
#          ( auth types supported: "basic"-http-r"user/pass"-socks ()1.5 KB)
#          mirrors.neusoft.edu.cn/kali kali-rolling/main amd64 Packages [21
[ProxyList]
# add proxy here .. [neusoft.edu.cn/kali kali-rolling/main amd64 Contents (de
# meanwhile
# defaults set to "tor" [neusoft.edu.cn/kali kali-rolling/contrib amd64 Packages
#socks4      127.0.0.1 9050
socks5 192.168.23.1 7897 [neusoft.edu.cn/kali kali-rolling/contrib amd64 Contents
~          [mirrors.neusoft.edu.cn/kali kali-rolling/non-free amd64 Packages
~          [mirrors.neusoft.edu.cn/kali kali-rolling/non-free amd64 Contents
~          [B]
~          [mirrors.neusoft.edu.cn/kali kali-rolling/non-free-firmware amd64
~          [.6 KB]
~          [mirrors.neusoft.edu.cn/kali kali-rolling/non-free-firmware amd64
~          [b) [24.3 KB]
~          [MB in 2min 51s (435 KB/s)
~          ge lists ... Done
~          [~]
1)-[~]           158,20           Bot
```

以太网适配器 VMware Network Adapter VMnet1:

连接特定的 DNS 后缀 :	
本地链接 IPv6 地址 :	fe80::b3d1:c455:cf5e:6613%7
IPv4 地址 :	192.168.23.1
子网掩码 :	255.255.255.0
默认网关 :	

`proxychains4` 的使用方法是，在需要通过代理的命令之前加上 `proxychains4`。

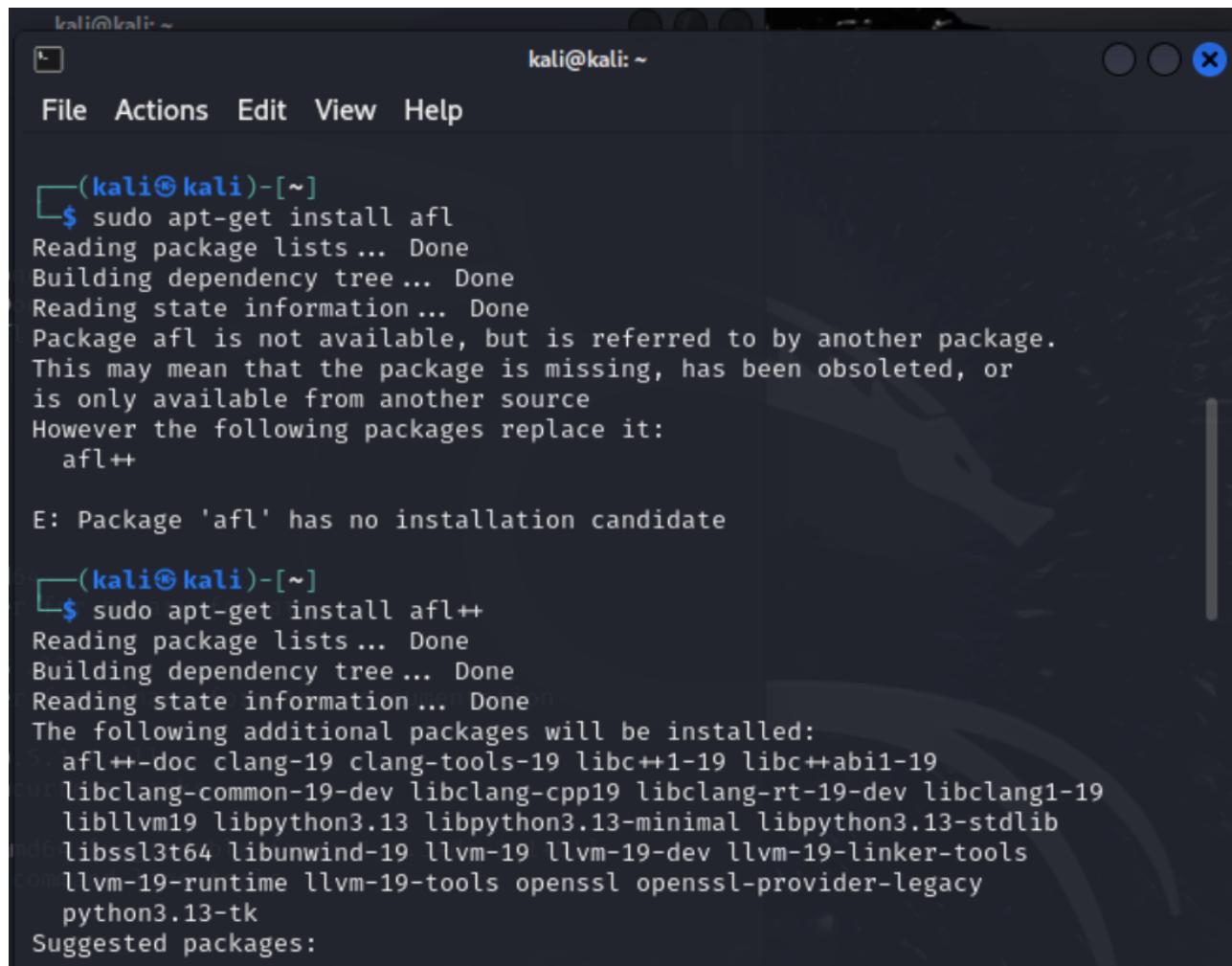
可以使用 `proxychains4 bash` 启动一个流量全部由代理接管的终端，避免重复输入 `proxychains4`。在 `install` 之前，先使用 `update` 更新一下



```
kali㉿kali: ~
File Actions Edit View Help
└─(kali㉿kali)-[~]
$ proxychains4 bash
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.17
[proxychains] DLL init: proxychains-ng 4.17
└─(kali㉿kali)-[~]
$ sudo apt-get update
[sudo] password for kali: 
Sorry, try again.
[sudo] password for kali:
Get:1 http://mirrors.neusoft.edu.cn/kali kali-rolling InRelease [41.5 kB]
Get:2 http://mirrors.neusoft.edu.cn/kali kali-rolling/main amd64 Packages [21
.0 MB]
Get:3 http://mirrors.neusoft.edu.cn/kali kali-rolling/main amd64 Contents (de
b) [51.5 MB]
Get:4 http://mirrors.neusoft.edu.cn/kali kali-rolling/contrib amd64 Packages
[121 kB]
Get:5 http://mirrors.neusoft.edu.cn/kali kali-rolling/contrib amd64 Contents
(deb) [328 kB]
Get:6 http://mirrors.neusoft.edu.cn/kali kali-rolling/non-free amd64 Packages
[204 kB]
Get:7 http://mirrors.neusoft.edu.cn/kali kali-rolling/non-free amd64 Contents
(deb) [914 kB]
Get:8 http://mirrors.neusoft.edu.cn/kali kali-rolling/non-free-firmware amd64
Packages [10.6 kB]
```

3.2.2 安装

运行命令，可以看到，afl 已经被更先进的 afl++取代了。并且 afl 的 [github 仓库](#)已经被 google 归档了。于是尝试安装 afl++.



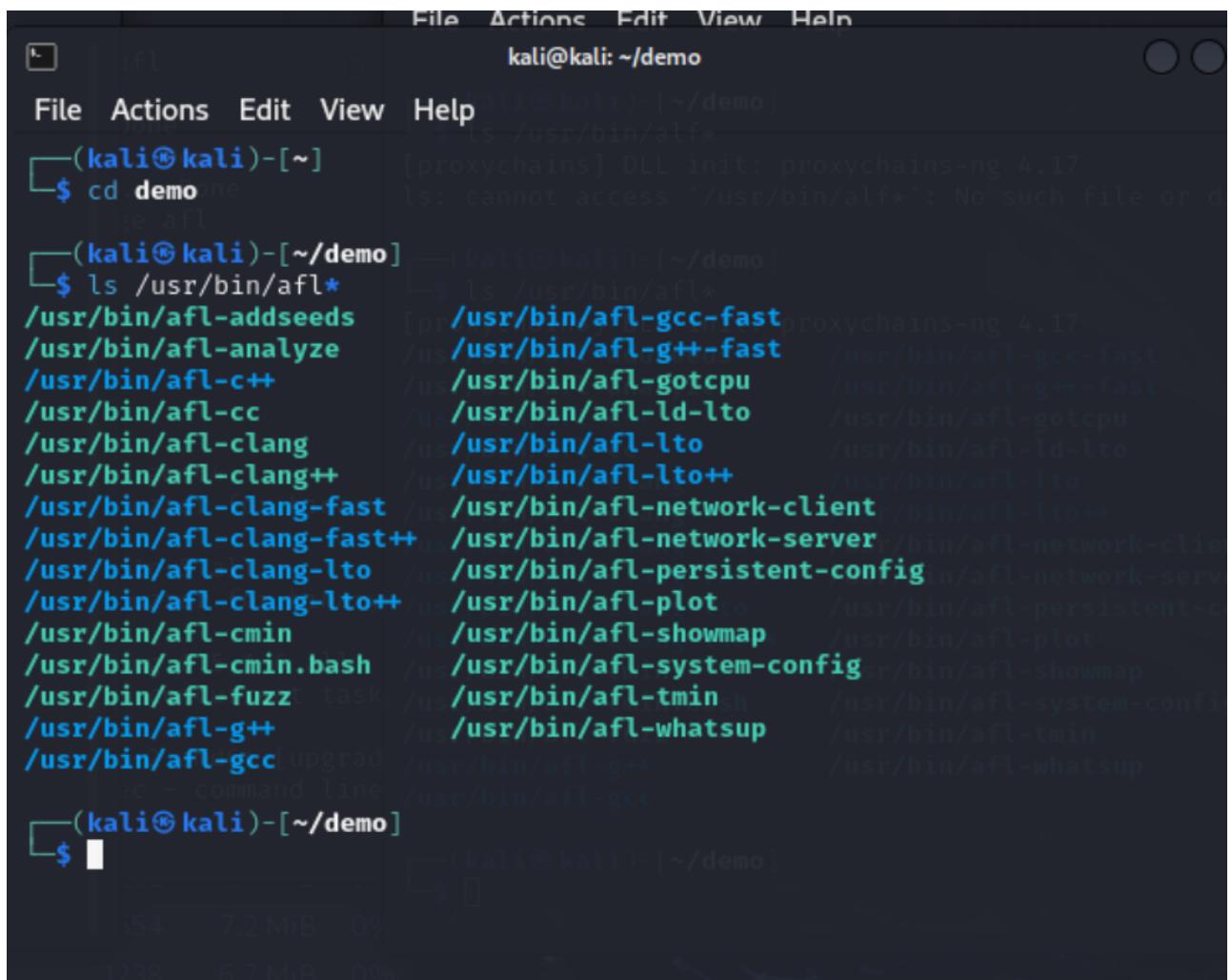
The screenshot shows a terminal window titled 'kali@kali: ~'. The window has a dark background with light-colored text. It displays two separate command-line sessions. The first session shows the attempt to install 'afl' via 'sudo apt-get install afl', which fails because 'afl' is not available but is referred to by 'afl++'. The second session shows the successful installation of 'afl++' via 'sudo apt-get install afl++', along with the list of additional packages installed, including various LLVM and Clang components.

```
(kali㉿kali)-[~]
$ sudo apt-get install afl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Package afl is not available, but is referred to by another package.
This may mean that the package is missing, has been obsoleted, or
is only available from another source
However the following packages replace it:
  afl++

E: Package 'afl' has no installation candidate

(kali㉿kali)-[~]
$ sudo apt-get install afl++
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  afl++-doc clang-19 clang-tools-19 libc++1-19 libc++abi1-19
  libclang-common-19-dev libclang-cpp19 libclang-rt-19-dev libclang1-19
  libllvm19 libpython3.13 libpython3.13-minimal libpython3.13-stdlib
  libssl3t64 libunwind-19 llvm-19 llvm-19-dev llvm-19-linker-tools
  llvm-19-runtime llvm-19-tools openssl openssl-provider-legacy
  python3.13-tk
Suggested packages:
```

可以看到，安装完成后，`/usr/bin/` 下有了这些文件。



```
kali@kali:~/demo
File Actions Edit View Help
(kali㉿kali)-[~] ls /usr/bin/afl*
ls: cannot access '/usr/bin/afl*': No such file or directory
$ cd demo
$ ls /usr/bin/afl*
/usr/bin/afl-addseeds  /usr/bin/afl-gcc-fast  /usr/bin/afl-gotcpu
/usr/bin/afl-analyze   /usr/bin/afl-g++-fast  /usr/bin/afl-g++-fast
/usr/bin/afl-c++       /usr/bin/afl-gotcpu   /usr/bin/afl-g++-fast
/usr/bin/afl-cc        /usr/bin/afl-ld-lto   /usr/bin/afl-gotcpu
/usr/bin/afl-clang     /usr/bin/afl-lto    /usr/bin/afl-ld-lto
/usr/bin/afl-clang++   /usr/bin/afl-lto++   /usr/bin/afl-lto
/usr/bin/afl-clang-fast /usr/bin/afl-network-client /usr/bin/afl-lto++
/usr/bin/afl-clang-fast++ /usr/bin/afl-network-server /usr/bin/afl-network-client
/usr/bin/afl-clang-lto   /usr/bin/afl-persistent-config /usr/bin/afl-network-server
/usr/bin/afl-clang-lto++ /usr/bin/afl-plot   /usr/bin/afl-persistent-config
/usr/bin/afl-cmin      /usr/bin/afl-showmap /usr/bin/afl-plot
/usr/bin/afl-cmin.bash /usr/bin/afl-system-config /usr/bin/afl-showmap
/usr/bin/afl-fuzz      /usr/bin/afl-tmin   /usr/bin/afl-system-config
/usr/bin/afl-g++        /usr/bin/afl-whatsup /usr/bin/afl-tmin
/usr/bin/afl-gcc        /usr/bin/afl-g++      /usr/bin/afl-whatsup
$
```

3.2.3 应用

接下来我们使用 `afl++` 进行测试，看看它的能力，顺便感受下模糊测试。

- 创建本次实验的程序：新建文件夹 demo，并创建实验的程序 Test.c，该代码编译后得到的程序如果被传入“deadbeef”则会终止，如果传入其他字符会原样输出。

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv) {
    char ptr[20];
    FILE *fp = fopen(argv[1], "r");
    fgets(ptr, sizeof(ptr), fp);
    if(argc>1){
        if(ptr[0] == 'd'){
            if(ptr[1] == 'e'){
                if(ptr[2] == 'a'){
                    if(ptr[3] == 'b'){
                        if(ptr[4] == 'e'){
                            if(ptr[5] == 'f'){
                                if(ptr[6] == 'e'){
                                    if(ptr[7] == 'f'){
                                        abort();
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    printf("%c", ptr);
    return 0;
}
```

- 使用 afl 的编译器编译，可以使模糊测试过程更加高效。命令： `afl-gcc -o test`

`test.c`。编译出可执行文件，可以看到提示说有更好的编译方式。

```
/usr/bin/afl-clang-lto  /usr/bin/afl-persistent-config
/usr/bin/afl-clang-lto++ /usr/bin/afl-plot
/usr/bin/afl-cmin       /usr/bin/afl-showmap
/usr/bin/afl-cmin.bash  /usr/bin/afl-system-config
/usr/bin/afl-fuzz        /usr/bin/afl-tmin
/usr/bin/afl-g++         /usr/bin/afl-whatsup
/usr/bin/afl-gcc

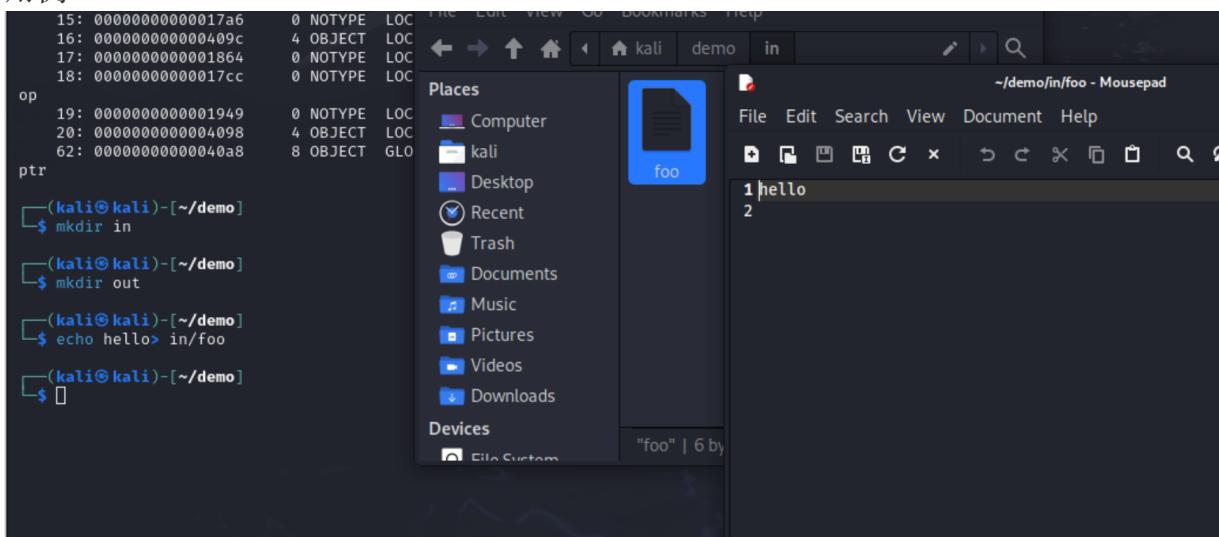
(kali㉿kali)-[~/demo]
$ afl-gcc -o test test.c
afl-gcc: command not found

(kali㉿kali)-[~/demo]
$ ./test
[!] WARNING: You are using outdated instrumentation, install LLVM and/or
plugin and use afl-clang-fast/afl-clang-lto/afl-gcc-fast instead!
[+] Instrumented 14 locations (64-bit, non-hardened mode, ratio 100%).
[+] Instrumen[~] 2 files: 21.8 KiB (22,308 bytes) | F
```

3. 编译后会有插桩符号，使用下面的命令可以验证这一点。命令：`readelf -s ./test | grep afl`

```
afl-as++4.21c by Michal Zalewski
[+] Instrumented 14 locations (64-bit, non-hardened mode, ratio 100%).
File System
[(kali㉿kali)-[~/demo]]
$ readelf -s ./test | grep afl
 4: 0000000000000001630    0 NOTYPE  LOCAL  DEFAULT  15 __afl_maybe_log
 6: 00000000000004088    8 OBJECT  LOCAL  DEFAULT  26 __afl_area_ptr
 7: 0000000000000001668    0 NOTYPE  LOCAL  DEFAULT  15 __afl_setup
 8: 0000000000000001640    0 NOTYPE  LOCAL  DEFAULT  15 __afl_store
 9: 00000000000004090    8 OBJECT  LOCAL  DEFAULT  26 __afl_prev_loc
10: 000000000000000165d    0 NOTYPE  LOCAL  DEFAULT  15 __afl_return
11: 000000000000040a0    1 OBJECT  LOCAL  DEFAULT  26 __afl_setup_failur
e
12: 0000000000000001689    0 NOTYPE  LOCAL  DEFAULT  15 __afl_setup_first
14: 0000000000000001951    0 NOTYPE  LOCAL  DEFAULT  15 __afl_setup_abort
15: 00000000000000017a6    0 NOTYPE  LOCAL  DEFAULT  15 __afl_forkserver
16: 0000000000000409c    4 OBJECT  LOCAL  DEFAULT  26 __afl_temp
17: 0000000000000001864    0 NOTYPE  LOCAL  DEFAULT  15 __afl_fork_resume
18: 00000000000000017cc    0 NOTYPE  LOCAL  DEFAULT  15 __afl_fork_wait_lo
op
19: 0000000000000001949    0 NOTYPE  LOCAL  DEFAULT  15 __afl_die
20: 00000000000004098    4 OBJECT  LOCAL  DEFAULT  26 __afl_fork_pid
62: 000000000000040a8    8 OBJECT  GLOBAL DEFAULT  26 __afl_global_area_
ptr
[(kali㉿kali)-[~/demo]]
$
```

4. 创建测试用例。首先，创建两个文件夹 in 和 out，分别存储模糊测试所需的输入和输出相关的内容。然后，在输入文件夹中创建一个包含字符串“hello”的文件。就是我们的测试用例，里面包含初步字符串 hello。AFL 会通过这个语料进行变异，构造更多的测试用例。



The screenshot shows a terminal window and a desktop environment. The terminal window displays the creation of directory structures and a file named 'foo' containing the string 'hello'. The desktop environment shows a file manager window with a 'Places' sidebar and a 'Mousepad' application window showing the same 'foo' file content.

```
15: 00000000000000017a6    0 NOTYPE  LOC
16: 0000000000000409c    4 OBJECT  LOC
17: 0000000000000001864    0 NOTYPE  LOC
18: 00000000000000017cc    0 NOTYPE  LOC
op
19: 0000000000000001949    0 NOTYPE  LOC
20: 00000000000004098    4 OBJECT  LOCAL
62: 000000000000040a8    8 OBJECT  GLO
ptr
[(kali㉿kali)-[~/demo]]
$ mkdir in
[(kali㉿kali)-[~/demo]]
$ mkdir out
[(kali㉿kali)-[~/demo]]
$ echo hello> in/foo
[(kali㉿kali)-[~/demo]]
$
```

5. 启动模糊测试。运行如下命令，开始启动模糊测试（@@表示目标程序需要从文件读取输入）： 命令： `afl-fuzz -i in -o out -- ./test @@`

```
(kali㉿kali)-[~/demo]
$ afl-fuzz -i in -o out -- ./test @@
afl-fuzz++4.21c based on afl by Michał Zalewski and a large online community
[+] AFL++ is maintained by Marc "van Hauser" Heuse, Dominik Maier, Andrea Fioraldi and Heiko "hexcoder" Eißfeldt
[+] AFL++ is open source, get it at https://github.com/AFLplusplus/AFLplusplus
[+] NOTE: AFL++ > v3 has changed defaults and behaviours - see README.md
[+] No -M/-S set, autoconfiguring for "-S default"
[*] Getting to work ...
[*] Using exploration-based constant power schedule (EXPLORE)
[*] Enabled testcache with 50 MB
[*] Generating fuzz data with a length of min=1 max=1048576
[*] Checking core_pattern ...
[!] WARNING: Could not check CPU scaling governor
[+] You have 4 CPU cores and 1 runnable tasks (utilization: 25%).
[+] Try parallel jobs - see docs/fuzzing_in_depth.md#c-using-multiple-cores
[*] Setting up output directories ...
[*] Checking CPU core loadout ...
[*] Found a free CPU core, try binding to #0.
[*] Scanning 'in' ...
[*] Loaded a total of 1 seeds.
[*] Creating hard links for all input files ...
[*] Validating target binary ...
[*] Spinning up the fork server ...
[!] WARNING: Old fork server model is used by the target, this still works though.
[+] All right - old fork server is up.
[*] No auto-generated dictionary tokens to reuse.
[*] Attempting dry run with 'id:000000,time:0,execs:0,orig:foo' ...
  len = 6, map size = 3, exec speed = 4021 us, hash = 9e3cbb6cdf9e04e3
[+] All test cases processed.
[+] Here are some useful stats:

  Test case count : 1 favored, 0 variable, 0 ignored, 1 total
  Bitmap range : 3 to 3 bits (average: 3.00 bits)
  Exec timing : 4021 to 4021 us (average: 4021 us)

[*] No -t option specified, so I'll use an exec timeout of 40 ms.
[+] All set and ready to roll!

      american fuzzy lop ++4.21c {default} (./test) [explore]
      process timing          overall results
      run time : 0 days, 0 hrs, 1 min, 13 sec   cycles done : 37

[+] All test cases processed.
[+] Here are some useful stats:

  Test case count : 1 favored, 0 variable, 0 ignored, 1 total
  Bitmap range : 3 to 3 bits (average: 3.00 bits)
  Exec timing : 4021 to 4021 us (average: 4021 us)

[*] No -t option specified, so I'll use an exec timeout of 40 ms.
[+] All set and ready to roll!

      american fuzzy lop ++4.21c {default} (./test) [explore]
      process timing          overall results
      run time : 0 days, 0 hrs, 1 min, 32 sec   cycles done : 47
      last new find : 0 days, 0 hrs, 1 min, 30 sec   corpus count : 5
      last saved crash : none seen yet   saved crashes : 0
      last saved hang : none seen yet   saved hangs : 0
      cycle progress          map coverage
      now processing : 4.65 (80.0%)   map density : 0.00% / 0.00%
      runs timed out : 0 (0.00%)   count coverage : 1.00 bits/tuple
      stage progress          findings in depth
      now trying : havoc   favored items : 5 (100.00%)
      stage execs : 252/600 (42.00%)   new edges on : 5 (100.00%)
      total execs : 88.2k   total crashes : 0 (0 saved)
      exec speed : 947.4/sec   total tmouts : 0 (0 saved)
      fuzzing strategy yields          item geometry
      bit flips : 0/0, 0/0, 0/0   levels : 5
      byte flips : 0/0, 0/0, 0/0   pending : 0
      arithmetics : 0/0, 0/0, 0/0   pend fav : 0
      known ints : 0/0, 0/0, 0/0   own finds : 4
      dictionary : 0/0, 0/0, 0/0, 0/0   imported : 0
      havoc/splice : 4/87.8k, 0/0   stability : 100.00%
      py/custom/rq : unused, unused, unused, unused
      trim/eff : n/a, n/a
      strategy: explore          state: started :-)
                                         [cpu000: 25%]
                                         ^[^\A]
```

6. 分析 crash 观察 fuzzing 结果，如有 crash，定位问题。
 - (a) 在 out 文件夹的 crashes 子文件夹里面是产生 crash 的样例，hangs 里面是产生超时的样例。
 - (b) 通常，得到 crash 样例后，可以将这些样例作为目标测试程序的输入，重新触发异常并跟踪运行状态，进行分析、定位程序出错的原因或确认存在的漏洞类型。

7. 可以看到，前 8 个字符确实是 deadbeef，分析原始程序可以发现，只要前面 8 个字符正确，就会引发错误。经过手动测试也可以验证确实是这样。

The terminal window displays Valgrind's 'explore' command results:

```
+4.21c {default} (./test) [explore]
overall results
cycles done : 154
corpus count : 8
saved crashes : 1
saved hangs : 0

map coverage
map density : 0.00% / 0.00%
count coverage : 1.00 bits/tuple
findings in depth
favored items : 8 (100.00%)
new edges on : 8 (100.00%)
total crashes : 10 (1 saved)
total tmouts : 0 (0 saved)

item geometry
levels : 6
pending : 0
pend fav : 0
own finds : 7
imported : 0
stability : 100.00%

[cpu000: 75%]
```

The status bar at the bottom shows: state: finished ... ^[A]

To the right, a hex editor window shows the byte sequence: 1 deadbeef 00 01 dbbbá

The screenshot shows a terminal window titled '(kali㉿kali)-[~/demo]' with the following command history:

```
$ ./test  
deadbeefSOHNUdbbb  
deadbeefSOHNUdbbb  
zsh: IOT instruction ./test  
$ ./test  
jiowioe  
jiowioe  
j  
$ ./test  
deadbeef  
deadbeef  
zsh: IOT instruction ./test  
$  
$
```

3.3 覆盖引导的概念和含义

覆盖引导模糊测试是一种智能模糊测试技术，它会监控目标程序的代码覆盖情况，并基于覆盖率的反馈来生成新的测试用例，从而不断探索程序中尚未被测试到的路径。

思想：

1. Code Coverage（代码覆盖）：观察测试用例触发了哪些代码路径。
2. Guided（引导）：用新的测试用例尝试覆盖更多路径。

优点：相比传统“随机乱测”的模糊测试，它是“有目的地乱测”，效率高出很多倍。

3.3.1 文件变异的概念和含义

文件变异就是指对原始的测试输入文件（seed）做出各种扰动、修改、变形，以便触发程序新的行为甚至发现 bug。

- 基本变异方式 (bit/byte level)

1. 翻转一个 bit/byte
2. 插入随机字节
3. 删除某一段内容
4. 重复一个字段或块
5. 改变文件长度

4 心得体会：

1. 熟悉了一些 Linux 下的操作，面对命令行和 vim 显得不那么生疏了。
2. Kali 的下载，让我对 `.torrent` 文件的使用方式更了解。
3. AFL，是一款模糊测试工具，通过安装、尝试使用，我对模糊测试的基本流程和此工具更了解了。
4. 配置网络环境的过程，让我对有了更多网络方面的实践经验，可能对计算机网络的学习有一定帮助。了解到代理协议分为 `http (s)` 和 `socks`，后者功能更丰富；了解了虚拟机里的一些网络结构，比如 NAT、桥接的概念以及它们大概是怎么运作的。