# 修改runtarget

**//if (child\_timed\_out) return FAULT\_TMOUT;改为 //liu 6/18**

**// if (child\_timed\_out) return FAULT\_NONE;就可以不总是hang了**

## 0.1 自己写的例子，验证

#include<stdio.h>

#include<stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

int main**(**int argc**,** char **\*\***argv**)**

**{**

int fd**;**

char buff**[**260**]** **=** **{**0**};**

//fd = open("serial.txt", O\_RDONLY);

printf**(**"%s\n%s\n"**,** argv**[**0**],** argv**[**1**]);**

fd **=** open**(**argv**[**1**],** O\_RDONLY**);**

read**(**fd**,** buff**,** 256**);**

close**(**fd**);**

**if** **(**buff**[**0**]** **!=** 'a'**)**

**{**

printf**(**"Bad boy:a\n"**);**

**}**

**if** **(**buff**[**1**]** **!=** 'b'**)**

**{**

printf**(**"Bad boy:b\n"**);**

**}**

**if** **(**buff**[**2**]** **!=** 'c'**)**

**{**

printf**(**"Bad boy:c\n"**);**

**}**

**if** **(**buff**[**3**]** **!=** 'd'**)**

**{**

printf**(**"Bad boy:d\n"**);**

**}**

**if** **(**buff**[**4**]** **!=** 'e'**)**

**{**

printf**(**"Bad boy:e\n"**);**

**}**

printf**(**"Good boy\n"**);**

**while(**1**);**

**return** 0**;**

**}**

# Winafl

## 1.1 run\_target

/\* Execute target application, monitoring for timeouts. Return status

information. The called program will update trace\_bits[]. \*/

static u8 run\_target**(**char**\*\*** argv**)** **{**

//todo watchdog timer to detect hangs

char command**[]** **=** "F"**;**

DWORD num\_read**;**

char result **=** 0**;**

**if(**sinkhole\_stds **&&** devnul\_handle **==** INVALID\_HANDLE\_VALUE**)** **{**

devnul\_handle **=** CreateFile**(**

"nul"**,**

GENERIC\_READ **|** GENERIC\_WRITE**,**

FILE\_SHARE\_READ **|** FILE\_SHARE\_WRITE**,**

**NULL,**

OPEN\_EXISTING**,**

0**,**

**NULL);**

**if(**devnul\_handle **==** INVALID\_HANDLE\_VALUE**)** **{**

PFATAL**(**"Unable to open the nul device."**);**

**}**

**}**

**if(!**is\_child\_running**())** **{**

destroy\_target\_process**(**0**);**

create\_target\_process**(**argv**);**

fuzz\_iterations\_current **=** 0**;**

**}**

child\_timed\_out **=** 0**;**

memset**(**trace\_bits**,** 0**,** MAP\_SIZE**);**

WriteFile**(**

pipe\_handle**,** // handle to pipe

command**,** // buffer to write from

1**,** // number of bytes to write

**&**num\_read**,** // number of bytes written

**NULL);** // not overlapped I/O

watchdog\_timeout\_time **=** get\_cur\_time**()** **+** exec\_tmout**;**

watchdog\_enabled **=** 1**;**

ReadFile**(**pipe\_handle**,** **&**result**,** 1**,** **&**num\_read**,** **NULL);//读取测试结果**

watchdog\_enabled **=** 0**;**

total\_execs**++;**

fuzz\_iterations\_current**++;**

**if(**fuzz\_iterations\_current **==** fuzz\_iterations\_max**)** **{**

destroy\_target\_process**(**2000**);**

**}**

//printf("total\_execs: %lld\n", total\_execs);

#ifdef \_\_x86\_64\_\_

classify\_counts**((**u64**\*)**trace\_bits**);**

#else

classify\_counts**((**u32**\*)**trace\_bits**);**

#endif /\* ^\_\_x86\_64\_\_ \*/

**if** **(**result **==** 'K'**)** **return** FAULT\_NONE**;**

**if** **(**result **==** 'C'**)** **{**

destroy\_target\_process**(**2000**);**

**return** FAULT\_CRASH**;**

**}**

destroy\_target\_process**(**0**);**

**return** FAULT\_HANG**;//修改为**//return FAULT\_HANG; //liu 6/17

**}**

## 1.2 watchdog\_timer

setup\_watchdog\_timer**();**

static void setup\_watchdog\_timer**()** **{**

watchdog\_enabled **=** 0**;**

InitializeCriticalSection**(&**critical\_section**);**

CreateThread**(NULL,** 0**,** watchdog\_timer**,** 0**,** 0**,** **NULL);**

**}**

DWORD WINAPI watchdog\_timer**(** LPVOID lpParam **)** **{**

u64 current\_time**;**

**while(**1**)** **{**

Sleep**(**1000**);**

current\_time **=** get\_cur\_time**();**

**if(**watchdog\_enabled **&&** **(**current\_time **>** watchdog\_timeout\_time**))** **{**

destroy\_target\_process**(**0**);//超时销毁进程**

**}**

**}**

**}**

## 1.3 测试程序，mywps,windows

// mywps.cpp : 定义控制台应用程序的入口点。

//

#include "stdafx.h"

#include <windows.h>

#include <iostream>

#include <exception>

using namespace std**;**

// crt\_settrans.cpp

// compile with: /EHa

#include <stdio.h>

#include <eh.h>

void stackoverflow**(**int n**)**

**{**

char tmp**[**4**];**

**if(**n**<**65530**)**

**{**

memset**(**tmp**,**'a'**,**n**);**

**}**

//int a=n/0;

//\*(int \*)(0)=0;

printf**(**"%s\n"**,**tmp**);**

//throw "Division by zero condition!";

**}**

int main**(**int argc**,** char **\***argv**[])**

**{**

//char ch;

HANDLE h**;**

//char fname[50]; //用于存放文件名

**if(**argc**!=**2**)**

**{**

printf**(**"argc should be 2!!!"**);**

exit**(**0**);**

**}**

**else**

**{**

//fp=fopen(argv[1],"r"); //只供读取

//\_\_asm int 3

h **=** CreateFileA**(**argv**[**1**],**

GENERIC\_READ**,**

0 **,**

**NULL,**

OPEN\_EXISTING**,**

FILE\_ATTRIBUTE\_NORMAL**,**

**NULL);**

**if** **(**h **!=** INVALID\_HANDLE\_VALUE**)**

**{**

**}**

**else**

**{**

printf**(**"creatfile fail!!!\n"**);**

exit**(**1**);**

**}**

int bRet**;**

DWORD dwReadSize**;**

char buff**[**256**];**

//\_\_asm int 3

bRet **=** ReadFile**(**h**,**buff**,**20**,&**dwReadSize**,NULL);**

**if** **(**buff**[**0**]** **!=** 'a'**)**

**{**

printf**(**"Bad boy:a\n"**);**

**}**

**if** **(**buff**[**1**]** **!=** 'b'**)**

**{**

printf**(**"Bad boy:b\n"**);**

**}**

**if** **(**buff**[**2**]** **!=** 'c'**)**

**{**

printf**(**"Bad boy:c\n"**);**

**}**

**if** **(**buff**[**3**]** **!=** 'd'**)**

**{**

printf**(**"Bad boy:d\n"**);**

**}**

**if** **(**buff**[**4**]** **!=** 'e'**)**

**{**

printf**(**"Bad boy:e\n"**);**

**}**

printf**(**"Good boy\n"**);**

**while(**1**);**

**return** 0**;**

//fclose(fp); //关闭文件

CloseHandle**(**h**);**

**}**

**return** 0**;**

**}**

## 1.4 编译winafl

Winafl从GitHub下载，DynamoRIO从官网下载。

在windows7 x64环境下，安装有vs2010

需要安装cmake，已经上传到百度网盘，cmake-3.9.0-rc2-win64-x64.msi

cd C:\winafl-master\build32

cmake .. -DDynamoRIO\_DIR=C:\DynamoRIO\cmake

Cmake生成了WinAFL.sln工程，可以使用vs 2010打开。

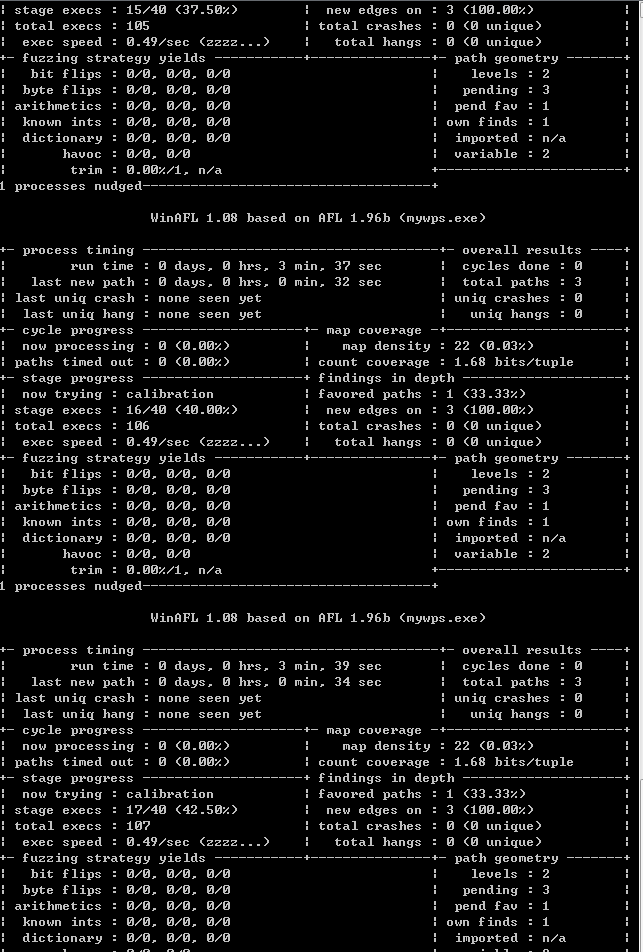
修改了run\_target函数以后，重新编译。

## 1.5 运行winafl

cd C:\winafl-master\build32\Debug

afl-fuzz.exe -i C:\Fuzzing\mywps\in -o C:\Fuzzing\mywps\out -D C:\DynamoRIO\bin32 -t 1000 -- -coverage\_module mywps.exe -fuzz\_iterations 10 -target\_module mywps.exe -target\_offset 0x11480 -nargs 2 -- "C:\source\2.mywps\mywps\Debug\mywps.exe" @@

注意：in文件夹里面放1.txt，内容为abcdefg



运行正确！

## 1.6 运行wps程序

afl-fuzz.exe -i C:\Fuzzing\RTF\tmp -o C:\Fuzzing\RTF\out -D C:\DynamoRIO\bin32 -t 20000 -- -coverage\_module wpsmain.dll -fuzz\_iterations 10 -target\_module wpsmain.dll -target\_offset 0x382C3 -nargs 2 -- "C:\Users\paper\AppData\Local\Kingsoft\WPS Office\10.1.0.6490\office6\wps.exe" @@

afl-fuzz.exe -i C:\Fuzzing\mywps\in -o C:\Fuzzing\mywps\out -D C:\DynamoRIO\bin32 -t 1000 -- -coverage\_module mywps.exe -fuzz\_iterations 10 -target\_module mywps.exe -target\_offset 0x11480 -nargs 2 -- "C:\source\2.mywps\mywps\Debug\mywps.exe" @@

# 调试

GitHub版本

l@ubuntu:~/Downloads/afl$ git log

commit feef3d49d11e734907d332517ba6d3b3d3577d44

Author: Martin Carpenter <mcarpenter@free.fr>

Date: Fri Jun 16 20:03:38 2017 +0200

2.43b

编译

afl-gcc -o mywps -g mywps.c

测试命令

afl-fuzz -i in -o out -t 1000 -- ./mywps @@

3. afl-fuzz初始化阶段，gdb调试

```

gdb -q afl-fuzz

set args -i in -o out -t 1000 -- ./mywps @@

set follow-fork-mode parent

show follow-fork-mode

b afl-fuzz.c:pivot\_inputs

b afl-fuzz.c:2571

```

查找命中的trace\_bits

find /b trace\_bits,+0x10000, 0x01

# 3. bitmap

EXP\_ST u8**\*** trace\_bits**;** /\* SHM with instrumentation bitmap \*/

EXP\_ST u8 virgin\_bits**[**MAP\_SIZE**],** /\* Regions yet untouched by fuzzing \*/

virgin\_hang**[**MAP\_SIZE**],** /\* Bits we haven't seen in hangs \*/

virgin\_crash**[**MAP\_SIZE**];** /\* Bits we haven't seen in crashes \*/

static u8 var\_bytes**[**MAP\_SIZE**];** /\* Bytes that appear to be variable \*/

# 4. queue

struct queue\_entry **{**

u8**\*** fname**;** /\* File name for the test case \*/

u32 len**;** /\* Input length \*/

u8 cal\_failed**,** /\* Calibration failed? \*/

trim\_done**,** /\* Trimmed? \*/

was\_fuzzed**,** /\* Had any fuzzing done yet? \*/

passed\_det**,** /\* Deterministic stages passed? \*/

has\_new\_cov**,** /\* Triggers new coverage? \*/

var\_behavior**,** /\* Variable behavior? \*/

favored**,** /\* Currently favored? \*/

fs\_redundant**;** /\* Marked as redundant in the fs? \*/

u32 bitmap\_size**,** /\* Number of bits set in bitmap \*/

exec\_cksum**;** /\* Checksum of the execution trace \*/

u64 exec\_us**,** /\* Execution time (us) \*/

handicap**,** /\* Number of queue cycles behind \*/

depth**;** /\* Path depth \*/

u8**\*** trace\_mini**;** /\* Trace bytes, if kept \*/

u32 tc\_ref**;** /\* Trace bytes ref count \*/

struct queue\_entry **\***next**,** /\* Next element, if any \*/

**\***next\_100**;** /\* 100 elements ahead \*/

**};**

static struct queue\_entry **\***queue**,** /\* Fuzzing queue (linked list) \*/

**\***queue\_cur**,** /\* Current offset within the queue \*/

**\***queue\_top**,** /\* Top of the list \*/

**\***q\_prev100**;** /\* Previous 100 marker \*/

static struct queue\_entry**\***

top\_rated**[**MAP\_SIZE**];** /\* Top entries for bitmap bytes \*/

# 5./init\_count\_class16()

static u16 count\_class\_lookup16**[**65536**];//数组**

EXP\_ST void init\_count\_class16**(**void**)** **{**

u32 b1**,** b2**;**

**for** **(**b1 **=** 0**;** b1 **<** 256**;** b1**++)**

**for** **(**b2 **=** 0**;** b2 **<** 256**;** b2**++)**

count\_class\_lookup16**[(**b1 **<<** 8**)** **+** b2**]** **=** //数组赋值

**(**count\_class\_lookup8**[**b1**]** **<<** 8**)** **|**

count\_class\_lookup8**[**b2**];**

**}**

static const u8 count\_class\_lookup8**[**256**]** **=** **{**

**[**0**]** **=** 0**,**

**[**1**]** **=** 1**,**

**[**2**]** **=** 2**,**

**[**3**]** **=** 4**,**

**[**4 **...** 7**]** **=** 8**,//这种数组赋值方式，a[4]=a[5]=a[6]=a[7]=8**

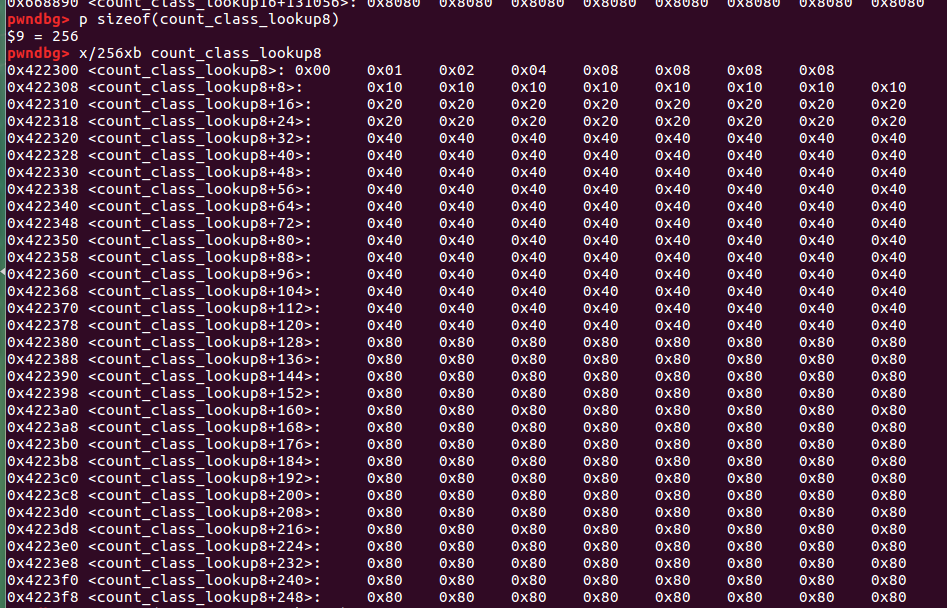
**[**8 **...** 15**]** **=** 16**,**

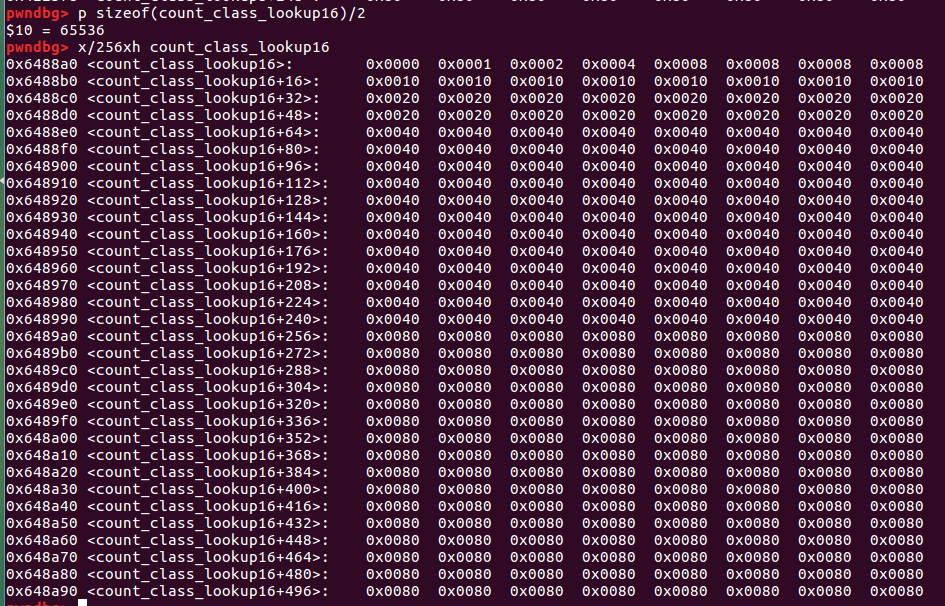
**[**16 **...** 31**]** **=** 32**,**

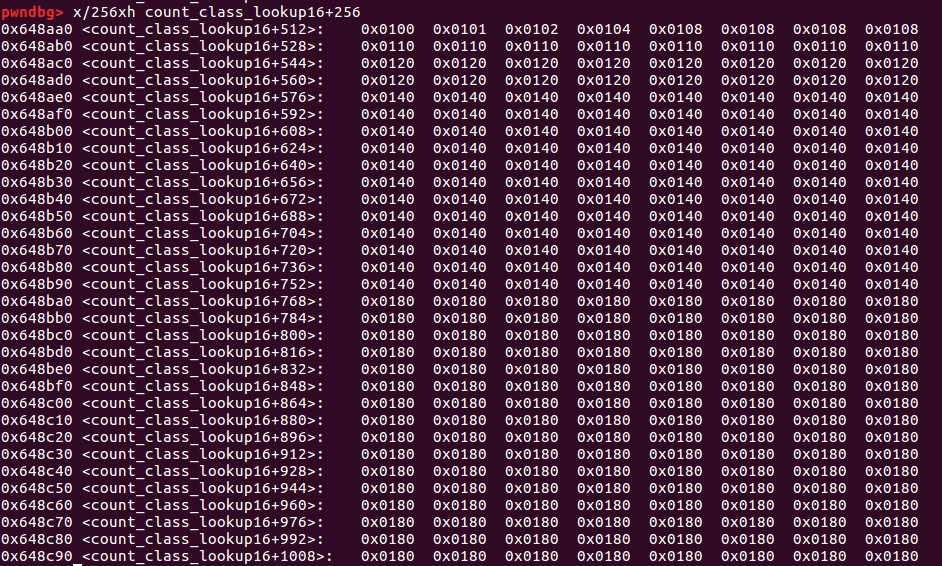
**[**32 **...** 127**]** **=** 64**,**

**[**128 **...** 255**]** **=** 128

**};**







# 6. setup\_shm建立bitmap

/\* Map size for the traced binary (2^MAP\_SIZE\_POW2). Must be greater than

2; you probably want to keep it under 18 or so for performance reasons

(adjusting AFL\_INST\_RATIO when compiling is probably a better way to solve

problems with complex programs). You need to recompile the target binary

after changing this - otherwise, SEGVs may ensue. \*/

#define MAP\_SIZE\_POW2 16

#define MAP\_SIZE (1 << MAP\_SIZE\_POW2)

/\* Configure shared memory and virgin\_bits. This is called at startup. \*/

EXP\_ST void setup\_shm**(**void**)** **{**

u8**\*** shm\_str**;**

**if** **(!**in\_bitmap**)** memset**(**virgin\_bits**,** 255**,** MAP\_SIZE**); //初始化**

memset**(**virgin\_hang**,** 255**,** MAP\_SIZE**);**

memset**(**virgin\_crash**,** 255**,** MAP\_SIZE**);**

shm\_id **=** shmget**(**IPC\_PRIVATE**,** MAP\_SIZE**,** IPC\_CREAT **|** IPC\_EXCL **|** 0600**);//申请共享内存，64K**

**if** **(**shm\_id **<** 0**)** PFATAL**(**"shmget() failed"**);**

atexit**(**remove\_shm**);**

shm\_str **=** alloc\_printf**(**"%d"**,** shm\_id**);**

/\* If somebody is asking us to fuzz instrumented binaries in dumb mode,

we don't want them to detect instrumentation, since we won't be sending

fork server commands. This should be replaced with better auto-detection

later on, perhaps? \*/

**if** **(!**dumb\_mode**)** setenv**(**SHM\_ENV\_VAR**,** shm\_str**,** 1**);//设置环境变量，子进程可以通过环境变量获得共享内存标识符，见ida**

ck\_free**(**shm\_str**);**

trace\_bits **=** shmat**(**shm\_id**,** **NULL,** 0**); //在afl-fuzz进程中，共享内存（bitmap）为trace\_bits，测试子进程也同步修改该bitmap**

**if** **(!**trace\_bits**)** PFATAL**(**"shmat() failed"**);**

**}**

# 6. -t timeout超时处理

## 5.1 timeout signal

**超时处理如下：**

/\* Exec timeout notifications. \*/

sa**.**sa\_handler **=** handle\_timeout**;**

sigaction**(**SIGALRM**,** **&**sa**,** **NULL);**

/\* Handle timeout (SIGALRM). \*/

static void handle\_timeout**(**int sig**)** **{**

**if** **(**child\_pid **>** 0**)** **{**

child\_timed\_out **=** 1**;** //设置全局标记

kill**(**child\_pid**,** SIGKILL**);//kill 孙子进程**

**}** **else** **if** **(**child\_pid **==** **-**1 **&&** forksrv\_pid **>** 0**)** **{**

child\_timed\_out **=** 1**;**

kill**(**forksrv\_pid**,** SIGKILL**);//将forkserver子进程给kill**

**}**

**}**

## 5.2 -t选项设置

**case** 't'**:** **{** /\* timeout \*/

u8 suffix **=** 0**;**

**if** **(**timeout\_given**)** FATAL**(**"Multiple -t options not supported"**);**

**if** **(**sscanf**(**optarg**,** "%u%c"**,** **&**exec\_tmout**,** **&**suffix**)** **<** 1 **||**

optarg**[**0**]** **==** '-'**)** FATAL**(**"Bad syntax used for -t"**);**

**if** **(**exec\_tmout **<** 5**)** FATAL**(**"Dangerously low value of -t"**);**

**if** **(**suffix **==** '+'**)** timeout\_given **=** 2**;** **else** timeout\_given **=** 1**;**

**break;**

**}**

# 7. /has\_new\_bits 新的发现

/\* Check if the current execution path brings anything new to the table.

Update virgin bits to reflect the finds. Returns 1 if the only change is

the hit-count for a particular tuple; 2 if there are new tuples seen.

Updates the map, so subsequent calls will always return 0.

This function is called after every exec() on a fairly large buffer, so

it needs to be fast. We do this in 32-bit and 64-bit flavors. \*/

static inline u8 has\_new\_bits**(**u8**\*** virgin\_map**)** **{**

#ifdef \_\_x86\_64\_\_

u64**\*** current **=** **(**u64**\*)**trace\_bits**;**

u64**\*** virgin **=** **(**u64**\*)**virgin\_map**;**

u32 i **=** **(**MAP\_SIZE **>>** 3**);//64位，8个字节**

#else

u32**\*** current **=** **(**u32**\*)**trace\_bits**;**

u32**\*** virgin **=** **(**u32**\*)**virgin\_map**;**

u32 i **=** **(**MAP\_SIZE **>>** 2**);**

#endif /\* ^\_\_x86\_64\_\_ \*/

u8 ret **=** 0**;**

**while** **(**i**--)** **{**

/\* Optimize for (\*current & \*virgin) == 0 - i.e., no bits in current bitmap

that have not been already cleared from the virgin map - since this will

almost always be the case. \*/

**if** **(**unlikely**(\***current**)** **&&** unlikely**(\***current **&** **\***virgin**))** **{//有新发现**

**if** **(**likely**(**ret **<** 2**))** **{**

u8**\*** cur **=** **(**u8**\*)**current**;**

u8**\*** vir **=** **(**u8**\*)**virgin**;**

/\* Looks like we have not found any new bytes yet; see if any non-zero

bytes in current[] are pristine in virgin[]. \*/

#ifdef \_\_x86\_64\_\_

**if** **((**cur**[**0**]** **&&** vir**[**0**]** **==** 0xff**)** **||** **(**cur**[**1**]** **&&** vir**[**1**]** **==** 0xff**)** **||**

**(**cur**[**2**]** **&&** vir**[**2**]** **==** 0xff**)** **||** **(**cur**[**3**]** **&&** vir**[**3**]** **==** 0xff**)** **||**

**(**cur**[**4**]** **&&** vir**[**4**]** **==** 0xff**)** **||** **(**cur**[**5**]** **&&** vir**[**5**]** **==** 0xff**)** **||**

**(**cur**[**6**]** **&&** vir**[**6**]** **==** 0xff**)** **||** **(**cur**[**7**]** **&&** vir**[**7**]** **==** 0xff**))** ret **=** 2**;//有新发现，新的tuple**

**else** ret **=** 1**;//只是hit count的变化**

#else

**if** **((**cur**[**0**]** **&&** vir**[**0**]** **==** 0xff**)** **||** **(**cur**[**1**]** **&&** vir**[**1**]** **==** 0xff**)** **||**

**(**cur**[**2**]** **&&** vir**[**2**]** **==** 0xff**)** **||** **(**cur**[**3**]** **&&** vir**[**3**]** **==** 0xff**))** ret **=** 2**;**

**else** ret **=** 1**;**

#endif /\* ^\_\_x86\_64\_\_ \*/

**}**

**\***virgin **&=** **~\***current**;//把virgin\_bits清零。virgin\_bits[MAP\_SIZE], /\* Regions yet untouched by fuzzing \*/**

**}**

current**++;**

virgin**++;**

**}**

**if** **(**ret **&&** virgin\_map **==** virgin\_bits**)** bitmap\_changed **=** 1**;**

**return** ret**;**

**}**

## 7.1 count\_bytes 统计数量

/\* Count the number of bytes set in the bitmap. Called fairly sporadically,

mostly to update the status screen or calibrate and examine confirmed

new paths. \*/

static u32 count\_bytes**(**u8**\*** mem**)** **{**

u32**\*** ptr **=** **(**u32**\*)**mem**;**

u32 i **=** **(**MAP\_SIZE **>>** 2**);**

u32 ret **=** 0**;**

**while** **(**i**--)** **{**

u32 v **=** **\*(**ptr**++);**

**if** **(!**v**)** **continue;**

**if** **(**v **&** FF**(**0**))** ret**++;**

**if** **(**v **&** FF**(**1**))** ret**++;**

**if** **(**v **&** FF**(**2**))** ret**++;**

**if** **(**v **&** FF**(**3**))** ret**++;**

**}**

**return** ret**;**

**}**

## 7.2 minimize\_bits

/\* Compact trace bytes into a smaller bitmap. We effectively just drop the

count information here. This is called only sporadically, for some

new paths. \*/

static void minimize\_bits**(**u8**\*** dst**,** u8**\*** src**)** **{**

u32 i **=** 0**;**

**while** **(**i **<** MAP\_SIZE**)** **{**

**if** **(\*(**src**++))** dst**[**i **>>** 3**]** **|=** 1 **<<** **(**i **&** 7**);**

i**++;**

**}**

**}**

# 8. ida 插桩结果

char \_\_usercall \_afl\_maybe\_log@**<**al**>(**char a1@**<**of**>,** \_\_int64 \_RAX@**<**rax**>,** \_\_int64 a3@**<**rcx**>,** \_\_m128i a4@**<**xmm0**>,** \_\_m128i a5@**<**xmm1**>,** \_\_m128i a6@**<**xmm2**>,** \_\_m128i a7@**<**xmm3**>,** \_\_m128i a8@**<**xmm4**>,** \_\_m128i a9@**<**xmm5**>,** \_\_m128i a10@**<**xmm6**>,** \_\_m128i a11@**<**xmm7**>,** \_\_m128i a12@**<**xmm8**>,** \_\_m128i a13@**<**xmm9**>,** \_\_m128i a14@**<**xmm10**>,** \_\_m128i a15@**<**xmm11**>,** \_\_m128i a16@**<**xmm12**>,** \_\_m128i a17@**<**xmm13**>,** \_\_m128i a18@**<**xmm14**>,** \_\_m128i a19@**<**xmm15**>,** \_\_int64 a20**,** \_\_int64 a21**,** \_\_int64 a22**,** int a23**,** int a24**,** int a25**,** int a26**,** int a27**,** int a28**,** \_\_int64 a29**,** int a30**,** int a31**,** \_\_int64 a32**,** int a33**,** int a34**,** \_\_int64 a35**,** int a36**,** int a37**,** \_\_int64 a38**,** int a39**,** int a40**,** \_\_int64 a41**,** int a42**,** int a43**,** \_\_int64 a44**,** int a45**,** int a46**,** \_\_int64 a47**,** int a48**,** int a49**,** \_\_int64 a50**,** int a51**,** int a52**,** \_\_int64 a53**,** int a54**,** int a55**,** \_\_int64 a56**,** int a57**,** int a58**,** \_\_int64 a59**,** int a60**,** int a61**)**

**{**

\_\_int64 v61**;** // rdx@1

\_\_int64 v62**;** // rcx@2

char **\***v64**;** // rax@7

int v65**;** // eax@8

void **\***v66**;** // rax@8

int v67**;** // edi@10

\_\_int64 v68**;** // rax@11

\_\_int64 v69**;** // rax@13

\_\_int64 v70**;** // [sp-10h] [bp-20h]@9

\_\_int64 retaddr**;** // [sp+10h] [bp+0h]@7

\_\_int64 v72**;** // [sp+18h] [bp+8h]@7

\_\_asm **{** lahf **}**

LOBYTE**(**\_RAX**)** **=** a1**;**

v61 **=** \_afl\_area\_ptr**;**

**if** **(** **!**\_afl\_area\_ptr **)**

**{**

**if** **(** \_afl\_setup\_failure **)**

**return** \_RAX **+** 127**;**

v61 **=** \_afl\_global\_area\_ptr**;**

**if** **(** \_afl\_global\_area\_ptr **)**

**{**

\_afl\_area\_ptr **=** \_afl\_global\_area\_ptr**;**

**}**

**else**

**{**

retaddr **=** \_RAX**;**

v72 **=** a3**;**

\_mm\_storel\_epi64**((**\_\_m128i **\*)&**a31**,** a4**);**

\_mm\_storel\_epi64**((**\_\_m128i **\*)&**a33**,** a5**);**

\_mm\_storel\_epi64**((**\_\_m128i **\*)&**a35**,** a6**);**

\_mm\_storel\_epi64**((**\_\_m128i **\*)&**a37**,** a7**);**

\_mm\_storel\_epi64**((**\_\_m128i **\*)&**a39**,** a8**);**

\_mm\_storel\_epi64**((**\_\_m128i **\*)&**a41**,** a9**);**

\_mm\_storel\_epi64**((**\_\_m128i **\*)&**a43**,** a10**);**

\_mm\_storel\_epi64**((**\_\_m128i **\*)&**a45**,** a11**);**

\_mm\_storel\_epi64**((**\_\_m128i **\*)&**a47**,** a12**);**

\_mm\_storel\_epi64**((**\_\_m128i **\*)&**a49**,** a13**);**

\_mm\_storel\_epi64**((**\_\_m128i **\*)&**a51**,** a14**);**

\_mm\_storel\_epi64**((**\_\_m128i **\*)&**a53**,** a15**);**

\_mm\_storel\_epi64**((**\_\_m128i **\*)&**a55**,** a16**);**

\_mm\_storel\_epi64**((**\_\_m128i **\*)&**a57**,** a17**);**

\_mm\_storel\_epi64**((**\_\_m128i **\*)&**a59**,** a18**);**

\_mm\_storel\_epi64**((**\_\_m128i **\*)&**a61**,** a19**);**

v64 **=** getenv**(**"\_\_AFL\_SHM\_ID"**); //获取共享内存map的地址，通过环境变量**

**if** **(** **!**v64 **||** **(**v65 **=** atoi**(**v64**),** v66 **=** shmat**(**v65**,** 0LL**,** 0**),** v66 **==** **(**void **\*)-**1**)** **)**

**{**

**++**\_afl\_setup\_failure**;**

LOBYTE**(**\_RAX**)** **=** retaddr**;**

**return** \_RAX **+** 127**;**

**}**

\_afl\_area\_ptr **=** **(**\_\_int64**)**v66**;//共享内存map赋值**

\_afl\_global\_area\_ptr **=** v66**;**

v70 **=** **(**\_\_int64**)**v66**;**

**if** **(** write**(**199**,** **&**\_afl\_temp**,** 4uLL**)** **==** 4 **) //通知afl-fuzz父进程init\_forkserver函数，已经启动forkserver**

**{**

**while** **(** 1 **)**

**{**

v67 **=** 198**;**

**if** **(** read**(**198**,** **&**\_afl\_temp**,** 4uLL**)** **!=** 4 **)//等待启动命令，run\_target**

**break;**

LODWORD**(**v68**)** **=** fork**();**

**if** **(** v68 **<** 0 **)**

**break;**

**if** **(** **!**v68 **)**

**goto** \_\_afl\_fork\_resume**;//孙子进程继续跑测试软件**

\_afl\_fork\_pid **=** v68**;**

write**(**199**,** **&**\_afl\_fork\_pid**,** 4uLL**);//forkserver将孙子进程pid发给afl-fuzz**

v67 **=** \_afl\_fork\_pid**;**

LODWORD**(**v69**)** **=** waitpid**(**\_afl\_fork\_pid**,** **&**\_afl\_temp**,** 0**);//<https://linux.die.net/man/2/waitpid> afl\_temp保存exit status**

**if** **(** v69 **<=** 0 **)**

**break;**

write**(**199**,** **&**\_afl\_temp**,** 4uLL**);//forkserver将孙子进程的退出状态发给afl-fuzz**

**}**

\_exit**(**v67**);**

**}**

\_\_afl\_fork\_resume**:**

close**(**198**);**

close**(**199**);**

v61 **=** v70**;**

LOBYTE**(**\_RAX**)** **=** retaddr**;**

a3 **=** v72**;**

**}**

**}**

v62 **=** \_afl\_prev\_loc **^** a3**;**

\_afl\_prev\_loc **^=** v62**;**

\_afl\_prev\_loc **=** **(**unsigned \_\_int64**)**\_afl\_prev\_loc **>>** 1**;**

**++\*(**\_BYTE **\*)(**v61 **+** v62**); // map count ++**

**return** \_RAX **+** 127**;**

**}**

**覆盖率计算**

**通过在编译期间instrument一些指令来捕获branch (edge) coverage和运行时分支执行计数**

**在分支点插入的指令大概如下：**

**cur\_location = <COMPILE\_TIME\_RANDOM>;**

**shared\_mem[cur\_location ^ prev\_location]++;**

**prev\_location = cur\_location >> 1;**

**为了简化连接复杂对象的过程和保持XOR输出平均分布，当前位置是随机产生的。**

**share\_mem[]数组是一个调用者传给被instrument程序的64KB的共享内存区域，数组的元素是Byte。数组中的每个元素，都被编码成一个(branch\_src, branch\_dst)，相当于存储路径的bitmap。这个数组的大小要应该能存2K到10K个分支节点，这样即可以减少冲突，也可以实现毫秒级别的分析。**

**这种形式的覆盖率，相对于简单的基本块覆盖率来说，对程序运行路径提供了一个更好的描述。以下面两个路径产生的tupes为例：**

**A -> B -> C -> D -> E (tuples: AB, BC, CD, DE)**

**A -> B -> D -> C -> E (tuples: AB, BD, DC, CE)**

**这更有助于发现代码的漏洞，因为大多数安全漏洞经常是一些没有预料到的状态转移，而不是因为没有覆盖那一块代码。**

**最后一行右移操作是用来保持tuples的定向性。如果没有右移操作，A ^ B和B ^ A就没办法区别了，同样A ^ A和B ^ B也是一样的。Intel CPU缺少算数指令，左移可能会会导致级数重置为0，但是这种可能性很小，用左移纯粹是为了效率。**

# 9. Main初始化

## 9.1 setup\_dirs\_fds 建立out文件目录

/\* Prepare output directories and fds. \*/

EXP\_ST void setup\_dirs\_fds**(**void**)** **{**

### 9.1.1 创建out文件夹里面的各种文件目录，

/\* Queue directory for any starting & discovered paths. \*/

tmp **=** alloc\_printf**(**"%s/queue"**,** out\_dir**);**

**if** **(**mkdir**(**tmp**,** 0700**))** PFATAL**(**"Unable to create '%s'"**,** tmp**);**

9.2.2 maybe\_delete\_out\_dir，如果存在内容就删除

/\* Prepare output directories and fds. \*/

EXP\_ST void setup\_dirs\_fds**(**void**)** **{**

u8**\*** tmp**;**

s32 fd**;**

ACTF**(**"Setting up output directories..."**);**

**if** **(**sync\_id **&&** mkdir**(**sync\_dir**,** 0700**)** **&&** errno **!=** EEXIST**)**

PFATAL**(**"Unable to create '%s'"**,** sync\_dir**);**

**if** **(**mkdir**(**out\_dir**,** 0700**))** **{**

**if** **(**errno **!=** EEXIST**)** PFATAL**(**"Unable to create '%s'"**,** out\_dir**);**

maybe\_delete\_out\_dir**();//如果存在就删除**

**}** **else** **{**

**if** **(**in\_place\_resume**)// -i - 这种参数形式的时候**

FATAL**(**"Resume attempted but old output directory not found"**);**

out\_dir\_fd **=** open**(**out\_dir**,** O\_RDONLY**);**

#ifndef \_\_sun

**if** **(**out\_dir\_fd **<** 0 **||** flock**(**out\_dir\_fd**,** LOCK\_EX **|** LOCK\_NB**))**

PFATAL**(**"Unable to flock() output directory."**);**

#endif /\* !\_\_sun \*/

**}**

**如果out里面已经存在内容了**

**maybe\_delete\_out\_dir();**

### 9.1.2 函数实现，删除，或者再用（resume）

/\* Delete fuzzer output directory if we recognize it as ours, if the fuzzer

is not currently running, and if the last run time isn't too great. \*/

static void maybe\_delete\_out\_dir**(**void**)** **{**

FILE**\*** f**;**

u8 **\***fn **=** alloc\_printf**(**"%s/fuzzer\_stats"**,** out\_dir**);//保存上次fuzz的信息**

/\* See if the output directory is locked. If yes, bail out. If not,

create a lock that will persist for the lifetime of the process

(this requires leaving the descriptor open).\*/

out\_dir\_fd **=** open**(**out\_dir**,** O\_RDONLY**);**

**if** **(**out\_dir\_fd **<** 0**)** PFATAL**(**"Unable to open '%s'"**,** out\_dir**);**

#ifndef \_\_sun

**if** **(**flock**(**out\_dir\_fd**,** LOCK\_EX **|** LOCK\_NB**)** **&&** errno **==** EWOULDBLOCK**)** **{**

SAYF**(**"\n" cLRD "[-] " cRST

"Looks like the job output directory is being actively used by another\n"

" instance of afl-fuzz. You will need to choose a different %s\n"

" or stop the other process first.\n"**,**

sync\_id **?** "fuzzer ID" **:** "output location"**);**

FATAL**(**"Directory '%s' is in use"**,** out\_dir**);**

**}**

#endif /\* !\_\_sun \*/

f **=** fopen**(**fn**,** "r"**);**

**if** **(**f**)** **{**

u64 start\_time**,** last\_update**;**

**if** **(**fscanf**(**f**,** "start\_time : %llu\n"

"last\_update : %llu\n"**,** **&**start\_time**,** **&**last\_update**)** **!=** 2**)//读取上次fuzz开始的时间和结束的时间**

FATAL**(**"Malformed data in '%s'"**,** fn**);**

fclose**(**f**);**

/\* Let's see how much work is at stake. \*/

**if** **(!**in\_place\_resume **&&** last\_update **-** start\_time **>** OUTPUT\_GRACE **\*** 60**)** **{//如果结束很长时间了，建议用户保存数据，终止fuzz**

SAYF**(**"\n" cLRD "[-] " cRST

"The job output directory already exists and contains the results of more\n"

" than %u minutes worth of fuzzing. To avoid data loss, afl-fuzz will \*NOT\*\n"

" automatically delete this data for you.\n\n"

" If you wish to start a new session, remove or rename the directory manually,\n"

" or specify a different output location for this job. To resume the old\n"

" session, put '-' as the input directory in the command line ('-i -') and\n"

" try again.\n"**,** OUTPUT\_GRACE**);**

FATAL**(**"At-risk data found in '%s'"**,** out\_dir**);**

**}**

**}**

ck\_free**(**fn**);**

/\* The idea for in-place resume is pretty simple: we temporarily move the old

queue/ to a new location that gets deleted once import to the new queue/

is finished. If \_resume/ already exists, the current queue/ may be

incomplete due to an earlier abort, so we want to use the old \_resume/

dir instead, and we let rename() fail silently. \*/

**if** **(**in\_place\_resume**)** **{**

u8**\*** orig\_q **=** alloc\_printf**(**"%s/queue"**,** out\_dir**);**

in\_dir **=** alloc\_printf**(**"%s/\_resume"**,** out\_dir**);//该模式下in\_dir的值**

rename**(**orig\_q**,** in\_dir**);** /\* Ignore errors \*///里面的内容就是上次fuzz的out/queue的内容

OKF**(**"Output directory exists, will attempt session resume."**);**

ck\_free**(**orig\_q**);**

**}** **else** **{**

OKF**(**"Output directory exists but deemed OK to reuse."**);**

**}**

ACTF**(**"Deleting old session data..."**);**

/\* Okay, let's get the ball rolling! First, we need to get rid of the entries

in <out\_dir>/.synced/.../id:\*, if any are present. \*/

fn **=** alloc\_printf**(**"%s/.synced"**,** out\_dir**);**

**if** **(**delete\_files**(**fn**,** **NULL))** **goto** dir\_cleanup\_failed**;**

ck\_free**(**fn**);**

/\* Next, we need to clean up <out\_dir>/queue/.state/ subdirectories: \*/

fn **=** alloc\_printf**(**"%s/queue/.state/deterministic\_done"**,** out\_dir**);//删除**

**if** **(**delete\_files**(**fn**,** CASE\_PREFIX**))** **goto** dir\_cleanup\_failed**;**

ck\_free**(**fn**);**

fn **=** alloc\_printf**(**"%s/queue/.state/auto\_extras"**,** out\_dir**);**

**if** **(**delete\_files**(**fn**,** "auto\_"**))** **goto** dir\_cleanup\_failed**;**

ck\_free**(**fn**);**

fn **=** alloc\_printf**(**"%s/queue/.state/redundant\_edges"**,** out\_dir**);**

**if** **(**delete\_files**(**fn**,** CASE\_PREFIX**))** **goto** dir\_cleanup\_failed**;**

ck\_free**(**fn**);**

fn **=** alloc\_printf**(**"%s/queue/.state/variable\_behavior"**,** out\_dir**);**

**if** **(**delete\_files**(**fn**,** CASE\_PREFIX**))** **goto** dir\_cleanup\_failed**;**

ck\_free**(**fn**);**

/\* Then, get rid of the .state subdirectory itself (should be empty by now)

and everything matching <out\_dir>/queue/id:\*. \*/

fn **=** alloc\_printf**(**"%s/queue/.state"**,** out\_dir**);**

**if** **(**rmdir**(**fn**)** **&&** errno **!=** ENOENT**)** **goto** dir\_cleanup\_failed**;**

ck\_free**(**fn**);**

fn **=** alloc\_printf**(**"%s/queue"**,** out\_dir**);**

**if** **(**delete\_files**(**fn**,** CASE\_PREFIX**))** **goto** dir\_cleanup\_failed**;**

ck\_free**(**fn**);**

/\* All right, let's do <out\_dir>/crashes/id:\* and <out\_dir>/hangs/id:\*. \*/

**if** **(!**in\_place\_resume**)** **{**

fn **=** alloc\_printf**(**"%s/crashes/README.txt"**,** out\_dir**);**

unlink**(**fn**);** /\* Ignore errors \*/

ck\_free**(**fn**);**

**}**

fn **=** alloc\_printf**(**"%s/crashes"**,** out\_dir**);**

/\* Make backup of the crashes directory if it's not empty and if we're

doing in-place resume. \*/

**if** **(**in\_place\_resume **&&** rmdir**(**fn**))** **{**

time\_t cur\_t **=** time**(**0**);**

struct tm**\*** t **=** localtime**(&**cur\_t**);**

#ifndef SIMPLE\_FILES

u8**\*** nfn **=** alloc\_printf**(**"%s.%04u-%02u-%02u-%02u:%02u:%02u"**,** fn**,**

t**->**tm\_year **+** 1900**,** t**->**tm\_mon **+** 1**,** t**->**tm\_mday**,**

t**->**tm\_hour**,** t**->**tm\_min**,** t**->**tm\_sec**);**

#else

u8**\*** nfn **=** alloc\_printf**(**"%s\_%04u%02u%02u%02u%02u%02u"**,** fn**,**

t**->**tm\_year **+** 1900**,** t**->**tm\_mon **+** 1**,** t**->**tm\_mday**,**

t**->**tm\_hour**,** t**->**tm\_min**,** t**->**tm\_sec**);**

#endif /\* ^!SIMPLE\_FILES \*/

rename**(**fn**,** nfn**);** /\* Ignore errors. \*/

ck\_free**(**nfn**);**

**}**

**if** **(**delete\_files**(**fn**,** CASE\_PREFIX**))** **goto** dir\_cleanup\_failed**;**

ck\_free**(**fn**);**

fn **=** alloc\_printf**(**"%s/hangs"**,** out\_dir**);**

/\* Backup hangs, too. \*/

**if** **(**in\_place\_resume **&&** rmdir**(**fn**))** **{**

time\_t cur\_t **=** time**(**0**);**

struct tm**\*** t **=** localtime**(&**cur\_t**);**

#ifndef SIMPLE\_FILES

u8**\*** nfn **=** alloc\_printf**(**"%s.%04u-%02u-%02u-%02u:%02u:%02u"**,** fn**,**

t**->**tm\_year **+** 1900**,** t**->**tm\_mon **+** 1**,** t**->**tm\_mday**,**

t**->**tm\_hour**,** t**->**tm\_min**,** t**->**tm\_sec**);**

#else

u8**\*** nfn **=** alloc\_printf**(**"%s\_%04u%02u%02u%02u%02u%02u"**,** fn**,**

t**->**tm\_year **+** 1900**,** t**->**tm\_mon **+** 1**,** t**->**tm\_mday**,**

t**->**tm\_hour**,** t**->**tm\_min**,** t**->**tm\_sec**);**

#endif /\* ^!SIMPLE\_FILES \*/

rename**(**fn**,** nfn**);** /\* Ignore errors. \*/

ck\_free**(**nfn**);**

**}**

**if** **(**delete\_files**(**fn**,** CASE\_PREFIX**))** **goto** dir\_cleanup\_failed**;**

ck\_free**(**fn**);**

/\* And now, for some finishing touches. \*/

fn **=** alloc\_printf**(**"%s/.cur\_input"**,** out\_dir**);**

**if** **(**unlink**(**fn**)** **&&** errno **!=** ENOENT**)** **goto** dir\_cleanup\_failed**;**

ck\_free**(**fn**);**

fn **=** alloc\_printf**(**"%s/fuzz\_bitmap"**,** out\_dir**);**

**if** **(**unlink**(**fn**)** **&&** errno **!=** ENOENT**)** **goto** dir\_cleanup\_failed**;**

ck\_free**(**fn**);**

**if** **(!**in\_place\_resume**)** **{**

fn **=** alloc\_printf**(**"%s/fuzzer\_stats"**,** out\_dir**);**

**if** **(**unlink**(**fn**)** **&&** errno **!=** ENOENT**)** **goto** dir\_cleanup\_failed**;**

ck\_free**(**fn**);**

**}**

fn **=** alloc\_printf**(**"%s/plot\_data"**,** out\_dir**);**

**if** **(**unlink**(**fn**)** **&&** errno **!=** ENOENT**)** **goto** dir\_cleanup\_failed**;**

ck\_free**(**fn**);**

OKF**(**"Output dir cleanup successful."**);**

/\* Wow... is that all? If yes, celebrate! \*/

**return;**

dir\_cleanup\_failed**:**

SAYF**(**"\n" cLRD "[-] " cRST

"Whoops, the fuzzer tried to reuse your output directory, but bumped into\n"

" some files that shouldn't be there or that couldn't be removed - so it\n"

" decided to abort! This happened while processing this path:\n\n"

" %s\n\n"

" Please examine and manually delete the files, or specify a different\n"

" output location for the tool.\n"**,** fn**);**

FATAL**(**"Output directory cleanup failed"**);**

**}**

## 5.2 read\_testcases();

/\* Read all testcases from the input directory, then queue them for testing.

Called at startup. \*/

static void read\_testcases**(**void**)** **{**

**<http://man7.org/linux/man-pages/man3/scandir.3.html>**

### 5.2.1 获取in样本文件名

/\* We use scandir() + alphasort() rather than readdir() because otherwise,

the ordering of test cases would vary somewhat randomly and would be

difficult to control. \*/

nl\_cnt **=** scandir**(**in\_dir**,** **&**nl**,** **NULL,** alphasort**);**

### 5.2.2 遍历文件夹

**for** **(**i **=** 0**;** i **<** nl\_cnt**;** i**++)** **{**

struct stat st**;**

u8**\*** fn **=** alloc\_printf**(**"%s/%s"**,** in\_dir**,** nl**[**i**]->**d\_name**);**

u8**\*** dfn **=** alloc\_printf**(**"%s/.state/deterministic\_done/%s"**,** in\_dir**,** nl**[**i**]->**d\_name**);**

### 5.2.3 检测样本文件是否太大

**if** **(**st**.**st\_size **>** MAX\_FILE**)**

FATAL**(**"Test case '%s' is too big (%s, limit is %s)"**,** fn**,**

DMS**(**st**.**st\_size**),** DMS**(**MAX\_FILE**));**

### 5.2.4 添加到队列

add\_to\_queue**(**fn**,** st**.**st\_size**,** passed\_det**);**

struct queue\_entry**\*** q **=** ck\_alloc**(sizeof(**struct queue\_entry**));**

q**->**fname **=** fname**;**

q**->**len **=** len**;**

q**->**depth **=** cur\_depth **+** 1**;**

q**->**passed\_det **=** passed\_det**;**

**if** **(**q**->**depth **>** max\_depth**)** max\_depth **=** q**->**depth**;**

**if** **(**queue\_top**)** **{**

queue\_top**->**next **=** q**;**

queue\_top **=** q**; //queue\_top感觉是队列的结尾**

**}** **else** q\_prev100 **=** queue **=** queue\_top **=** q**;**

queued\_paths**++;**

pending\_not\_fuzzed**++;**

## 5.3 Load\_auto加载之前fuzz识别的字典

如果是恢复fuzz，要加载之前识别的字典。

/\* Load automatically generated extras. \*/

static void load\_auto**(**void**)** **{**

u32 i**;**

**for** **(**i **=** 0**;** i **<** USE\_AUTO\_EXTRAS**;** i**++)** **{**

u8 tmp**[**MAX\_AUTO\_EXTRA **+** 1**];**

u8**\*** fn **=** alloc\_printf**(**"%s/.state/auto\_extras/auto\_%06u"**,** in\_dir**,** i**);//in\_dir为out/queue**

s32 fd**,** len;

fd **=** open**(**fn**,** O\_RDONLY**,** 0600**);**

**if** **(**fd **<** 0**)** **{**

**if** **(**errno **!=** ENOENT**)** PFATAL**(**"Unable to open '%s'"**,** fn**);**

ck\_free**(**fn**);**

**break;**

**}**

/\* We read one byte more to cheaply detect tokens that are too

long (and skip them). \*/

len **=** read**(**fd**,** tmp**,** MAX\_AUTO\_EXTRA **+** 1**);**

**if** **(**len **<** 0**)** PFATAL**(**"Unable to read from '%s'"**,** fn**);**

**if** **(**len **>=** MIN\_AUTO\_EXTRA **&&** len **<=** MAX\_AUTO\_EXTRA**)**

maybe\_add\_auto**(**tmp**,** len**);**

close**(**fd**);**

ck\_free**(**fn**);**

**}**

**if** **(**i**)** OKF**(**"Loaded %u auto-discovered dictionary tokens."**,** i**);**

**else** OKF**(**"No auto-generated dictionary tokens to reuse."**);**

**}**

/\* Maybe add automatic extra. \*/

static void maybe\_add\_auto**(**u8**\*** mem**,** u32 len**)** **{**

/\* At this point, looks like we're dealing with a new entry. So, let's

append it if we have room. Otherwise, let's randomly evict some other

entry from the bottom half of the list. \*/

**if** **(**a\_extras\_cnt **<** MAX\_AUTO\_EXTRAS**)** **{**

a\_extras **=** ck\_realloc\_block**(**a\_extras**,** **(**a\_extras\_cnt **+** 1**)** **\***

**sizeof(**struct extra\_data**));**

a\_extras**[**a\_extras\_cnt**].**data **=** ck\_memdup**(**mem**,** len**);//加载到**a\_extras数组

a\_extras**[**a\_extras\_cnt**].**len **=** len**;**

a\_extras\_cnt**++;**

**}**

## 5.4. pivot\_inputs 输入重新起名id

/\* Create hard links for input test cases in the output directory, choosing

good names and pivoting accordingly. \*/

static void pivot\_inputs**(**void**)** **{**

**重新起名为**

u8**\*** use\_name **=** strstr**(**rsl**,** ",orig:"**);**

**if** **(**use\_name**)** use\_name **+=** 6**;** **else** use\_name **=** rsl**;**

nfn **=** alloc\_printf**(**"%s/queue/id:%06u,orig:%s"**,** out\_dir**,** id**,** use\_name**);//id开头的orig结尾的文件名，放在queue文件夹下**

link\_or\_copy**(**q**->**fname**,** nfn**);**

## 5.5. detect\_file\_args

/\* Detect @@ in args. \*/

EXP\_ST void detect\_file\_args**(**char**\*\*** argv**)** **{**

u32 i **=** 0**;**

u8**\*** cwd **=** getcwd**(NULL,** 0**);**

**if** **(!**cwd**)** PFATAL**(**"getcwd() failed"**);**

**while** **(**argv**[**i**])** **{**

u8**\*** aa\_loc **=** strstr**(**argv**[**i**],** "@@"**);**

**if** **(**aa\_loc**)** **{**

u8 **\***aa\_subst**,** **\***n\_arg**;**

/\* If we don't have a file name chosen yet, use a safe default. \*/

**if** **(!**out\_file**)**

out\_file **=** alloc\_printf**(**"%s/.cur\_input"**,** out\_dir**);//赋值为**.cur\_input

/\* Be sure that we're always using fully-qualified paths. \*/

**if** **(**out\_file**[**0**]** **==** '/'**)** aa\_subst **=** out\_file**;**

**else** aa\_subst **=** alloc\_printf**(**"%s/%s"**,** cwd**,** out\_file**);**

/\* Construct a replacement argv value. \*/

**\***aa\_loc **=** 0**;**

n\_arg **=** alloc\_printf**(**"%s%s%s"**,** argv**[**i**],** aa\_subst**,** aa\_loc **+** 2**);**

argv**[**i**]** **=** n\_arg**;//@@替换为**.cur\_input

**\***aa\_loc **=** '@'**;**

**if** **(**out\_file**[**0**]** **!=** '/'**)** ck\_free**(**aa\_subst**);**

**}**

i**++;**

**}**

free**(**cwd**);** /\* not tracked \*/}

## 5.6. Resume 恢复fuzzing

**命令如下：**

**afl-fuzz -i - -o out -- ./mywps @@**

**/\* The idea for in-place resume is pretty simple: we temporarily move the old**

**queue/ to a new location that gets deleted once import to the new queue/**

**is finished. If \_resume/ already exists, the current queue/ may be**

**incomplete due to an earlier abort, so we want to use the old \_resume/**

**dir instead, and we let rename() fail silently. \*/**

**if** **(**in\_place\_resume**)** **{**

u8**\*** orig\_q **=** alloc\_printf**(**"%s/queue"**,** out\_dir**);**

in\_dir **=** alloc\_printf**(**"%s/\_resume"**,** out\_dir**);**

rename**(**orig\_q**,** in\_dir**);** /\* Ignore errors \*/

OKF**(**"Output directory exists, will attempt session resume."**);**

ck\_free**(**orig\_q**);**

**}** **else** **{**

OKF**(**"Output directory exists but deemed OK to reuse."**);**

**}**

# 10. 加载字典文件。。。待续

## 9.1 选项

**case** 'x'**:** /\* dictionary \*/

**if** **(**extras\_dir**)** FATAL**(**"Multiple -x options not supported"**);**

extras\_dir **=** optarg**;**

**break;**

## 9.2 加载

**if** **(**extras\_dir**)** load\_extras**(**extras\_dir**);**

# 11. 超时时间选项-t

**case** 't'**:** **{** /\* timeout \*/

u8 suffix **=** 0**;**

**if** **(**timeout\_given**)** FATAL**(**"Multiple -t options not supported"**);**

**if** **(**sscanf**(**optarg**,** "%u%c"**,** **&**exec\_tmout**,** **&**suffix**)** **<** 1 **||**

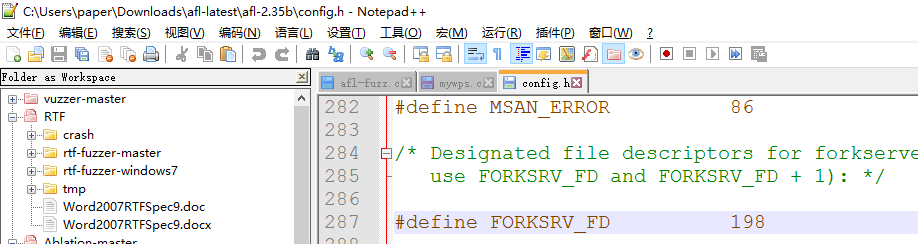
optarg**[**0**]** **==** '-'**)** FATAL**(**"Bad syntax used for -t"**);**

**if** **(**exec\_tmout **<** 5**)** FATAL**(**"Dangerously low value of -t"**);**

**if** **(**suffix **==** '+'**)** timeout\_given **=** 2**;** **else** timeout\_given **=** 1**;**

**break;**

# 12.init\_forkserver



## 4.1 父进程创建管道

**if** **(**pipe**(**st\_pipe**)** **||** pipe**(**ctl\_pipe**))** PFATAL**(**"pipe() failed"**);**

forksrv\_pid **=** fork**(); //获取forkserver pid**

## 4.2 子进程dup2管道

/\* Designated file descriptors for forkserver commands (the application will

use FORKSRV\_FD and FORKSRV\_FD + 1): \*/

#define FORKSRV\_FD 198

**if** **(**dup2**(**ctl\_pipe**[**0**],** FORKSRV\_FD**)** **<** 0**)** PFATAL**(**"dup2() failed"**);//重定向到198 199**

**if** **(**dup2**(**st\_pipe**[**1**],** FORKSRV\_FD **+** 1**)** **<** 0**)** PFATAL**(**"dup2() failed"**);**

dup2**(**dev\_null\_fd**,** 1**);**

dup2**(**dev\_null\_fd**,** 2**);**

**if** **(**out\_file**)** **{**

dup2**(**dev\_null\_fd**,** 0**);**

**}** **else** **{**

dup2**(**out\_fd**,** 0**);**

close**(**out\_fd**);**

**}**

execv**(**target\_path**,** argv**);**

## 4.3 父进程设置超时时间 alarm

it**.**it\_value**.**tv\_sec **=** **((**exec\_tmout **\*** FORK\_WAIT\_MULT**)** **/** 1000**);**

it**.**it\_value**.**tv\_usec **=** **((**exec\_tmout **\*** FORK\_WAIT\_MULT**)** **%** 1000**)** **\*** 1000**;**

setitimer**(**ITIMER\_REAL**,** **&**it**,** **NULL);**

**超时处理如下：**

/\* Exec timeout notifications. \*/

sa**.**sa\_handler **=** handle\_timeout**;**

sigaction**(**SIGALRM**,** **&**sa**,** **NULL);**

/\* Handle timeout (SIGALRM). \*/

static void handle\_timeout**(**int sig**)** **{**

**if** **(**child\_pid **>** 0**)** **{**

child\_timed\_out **=** 1**;**

kill**(**child\_pid**,** SIGKILL**);**

**}** **else** **if** **(**child\_pid **==** **-**1 **&&** forksrv\_pid **>** 0**)** **{**

child\_timed\_out **=** 1**;**

kill**(**forksrv\_pid**,** SIGKILL**);**

**}**

**}**

## 4.4 父进程等待forkserver启动，然后返回

rlen **=** read**(**fsrv\_st\_fd**,** **&**status**,** 4**);**

it**.**it\_value**.**tv\_sec **=** 0**;**

it**.**it\_value**.**tv\_usec **=** 0**;**

setitimer**(**ITIMER\_REAL**,** **&**it**,** **NULL);**

/\* If we have a four-byte "hello" message from the server, we're all set.

Otherwise, try to figure out what went wrong. \*/

**if** **(**rlen **==** 4**)** **{**

OKF**(**"All right - fork server is up."**);**

**return;**

**}**

对应于ida的结果：

**if** **(** write**(**199**,** **&**\_afl\_temp**,** 4uLL**)** **==** 4 **) //通知afl-fuzz父进程，已经启动forkserver**

# 13. perform\_dry\_run(use\_argv);

/\* Perform dry run of all test cases to confirm that the app is working as

expected. This is done only for the initial inputs, and only once. \*/

static void perform\_dry\_run**(**char**\*\*** argv**)** **{**

## 13.1 读取输入样本文件（queue）

fd **=** open**(**q**->**fname**,** O\_RDONLY**);**

**if** **(**fd **<** 0**)** PFATAL**(**"Unable to open '%s'"**,** q**->**fname**);**

use\_mem **=** ck\_alloc\_nozero**(**q**->**len**);**

**if** **(**read**(**fd**,** use\_mem**,** q**->**len**)** **!=** q**->**len**)**

FATAL**(**"Short read from '%s'"**,** q**->**fname**);**

close**(**fd**);**

## 13.2 /calibrate\_case(argv, q, use\_mem, 0, 1);

/\* Calibrate a new test case. This is done when processing the input directory

to warn about flaky or otherwise problematic test cases early on; and when

new paths are discovered to detect variable behavior and so on. \*/

static u8 calibrate\_case**(**char**\*\*** argv**,** struct queue\_entry**\*** q**,** u8**\*** use\_mem**,**

u32 handicap**,** u8 from\_queue**)** **{**

static u8 first\_trace**[**MAP\_SIZE**];**

u8 fault **=** 0**,** new\_bits **=** 0**,** var\_detected **=** 0**,**

first\_run **=** **(**q**->**exec\_cksum **==** 0**);**

u64 start\_us**,** stop\_us**;**

s32 old\_sc **=** stage\_cur**,** old\_sm **=** stage\_max**;**

u32 use\_tmout **=** exec\_tmout**;**

u8**\*** old\_sn **=** stage\_name**;**

/\* Be a bit more generous about timeouts when resuming sessions, or when

trying to calibrate already-added finds. This helps avoid trouble due

to intermittent latency. \*/

**if** **(!**from\_queue **||** resuming\_fuzz**)**

use\_tmout **=** MAX**(**exec\_tmout **+** CAL\_TMOUT\_ADD**,**

exec\_tmout **\*** CAL\_TMOUT\_PERC **/** 100**);**

q**->**cal\_failed**++;**

stage\_name **=** "calibration"**;**

stage\_max **=** CAL\_CYCLES**;//8**

/\* Make sure the forkserver is up before we do anything, and let's not

count its spin-up time toward binary calibration. \*/

**if** **(**dumb\_mode **!=** 1 **&&** **!**no\_forkserver **&&** **!**forksrv\_pid**)**

init\_forkserver**(**argv**);**

**if** **(**q**->**exec\_cksum**)** memcpy**(**first\_trace**,** trace\_bits**,** MAP\_SIZE**);**

start\_us **=** get\_cur\_time\_us**();**

**for** **(**stage\_cur **=** 0**;** stage\_cur **<** stage\_max**;** stage\_cur**++)** **{**

u32 cksum**;**

**if** **(!**first\_run **&&** **!(**stage\_cur **%** stats\_update\_freq**))** show\_stats**();**

write\_to\_testcase**(**use\_mem**,** q**->**len**);**

fault **=** run\_target**(**argv**,** use\_tmout**);**

/\* stop\_soon is set by the handler for Ctrl+C. When it's pressed,

we want to bail out quickly. \*/

**if** **(**stop\_soon **||** fault **!=** crash\_mode**)** **goto** abort\_calibration**;**

**if** **(!**dumb\_mode **&&** **!**stage\_cur **&&** **!**count\_bytes**(**trace\_bits**))** **{**

fault **=** FAULT\_NOINST**;**

**goto** abort\_calibration**;**

**}**

cksum **=** hash32**(**trace\_bits**,** MAP\_SIZE**,** HASH\_CONST**);**

**if** **(**q**->**exec\_cksum **!=** cksum**)** **{**

u8 hnb **=** has\_new\_bits**(**virgin\_bits**);**

**if** **(**hnb **>** new\_bits**)** new\_bits **=** hnb**;**

**if** **(**q**->**exec\_cksum**)** **{**

u32 i**;**

**for** **(**i **=** 0**;** i **<** MAP\_SIZE**;** i**++)** **{**

**if** **(!**var\_bytes**[**i**]** **&&** first\_trace**[**i**]** **!=** trace\_bits**[**i**])** **{**

var\_bytes**[**i**]** **=** 1**;**

stage\_max **=** CAL\_CYCLES\_LONG**;**

**}**

**}**

var\_detected **=** 1**;**

**}**

**else** **{**

q**->**exec\_cksum **=** cksum**;**

memcpy**(**first\_trace**,** trace\_bits**,** MAP\_SIZE**);**

**}**

**}**

**}**

stop\_us **=** get\_cur\_time\_us**();**

total\_cal\_us **+=** stop\_us **-** start\_us**;**

total\_cal\_cycles **+=** stage\_max**;**

/\* OK, let's collect some stats about the performance of this test case.

This is used for fuzzing air time calculations in calculate\_score(). \*/

q**->**exec\_us **=** **(**stop\_us **-** start\_us**)** **/** stage\_max**;**

q**->**bitmap\_size **=** count\_bytes**(**trace\_bits**);**

q**->**handicap **=** handicap**;**

q**->**cal\_failed **=** 0**;**

total\_bitmap\_size **+=** q**->**bitmap\_size**;**

total\_bitmap\_entries**++;**

update\_bitmap\_score**(**q**);**

/\* If this case didn't result in new output from the instrumentation, tell

parent. This is a non-critical problem, but something to warn the user

about. \*/

**if** **(!**dumb\_mode **&&** first\_run **&&** **!**fault **&&** **!**new\_bits**)** fault **=** FAULT\_NOBITS**;**

abort\_calibration**:**

**if** **(**new\_bits **==** 2 **&&** **!**q**->**has\_new\_cov**)** **{**

q**->**has\_new\_cov **=** 1**;**

queued\_with\_cov**++;**

**}**

/\* Mark variable paths. \*/

**if** **(**var\_detected**)** **{**

var\_byte\_count **=** count\_bytes**(**var\_bytes**);**

**if** **(!**q**->**var\_behavior**)** **{**

mark\_as\_variable**(**q**);**

queued\_variable**++;**

**}**

**}**

stage\_name **=** old\_sn**;**

stage\_cur **=** old\_sc**;**

stage\_max **=** old\_sm**;**

**if** **(!**first\_run**)** show\_stats**();**

**return** fault**;}**

### 13.2.1 开启forkserver

/\* Make sure the forkserver is up before we do anything, and let's not

count its spin-up time toward binary calibration. \*/

**if** **(**dumb\_mode **!=** 1 **&&** **!**no\_forkserver **&&** **!**forksrv\_pid**)**

init\_forkserver**(**argv**);**

### 13.2.2 /write\_to\_testcase(use\_mem, q->len);

/\* Write modified data to file for testing. If out\_file is set, the old file

is unlinked and a new one is created. Otherwise, out\_fd is rewound and

truncated. \*/

static void write\_to\_testcase**(**void**\*** mem**,** u32 len**)** **{**

s32 fd **=** out\_fd**;**

**if** **(**out\_file**)** **{**

unlink**(**out\_file**);** /\* Ignore errors. \*///detect\_file\_args中赋值为.cur\_input

fd **=** open**(**out\_file**,** O\_WRONLY **|** O\_CREAT **|** O\_EXCL**,** 0600**);**

**if** **(**fd **<** 0**)** PFATAL**(**"Unable to create '%s'"**,** out\_file**);**

**}** **else** lseek**(**fd**,** 0**,** SEEK\_SET**);**

ck\_write**(**fd**,** mem**,** len**,** out\_file**);//写入新文件**

**if** **(!**out\_file**)** **{**

**if** **(**ftruncate**(**fd**,** len**))** PFATAL**(**"ftruncate() failed"**);**

lseek**(**fd**,** 0**,** SEEK\_SET**);**

**}** **else** close**(**fd**);**

**}**

### 13.2.3 fault = run\_target(argv, use\_tmout);

运行测试进程，并且返回测试状态。见单独专题。

### 13.2.4 hash trace\_bits

cksum **=** hash32**(**trace\_bits**,** MAP\_SIZE**,** HASH\_CONST**);**

#define ROL64(\_x, \_r) ((((u64)(\_x)) << (\_r)) | (((u64)(\_x)) >> (64 - (\_r))))

static inline u32 hash32**(**const void**\*** key**,** u32 len**,** u32 seed**)** **{**

const u64**\*** data **=** **(**u64**\*)**key**;**

u64 h1 **=** seed **^** len**;**

len **>>=** 3**;**

**while** **(**len**--)** **{**

u64 k1 **=** **\***data**++;**

k1 **\*=** 0x87c37b91114253d5ULL**;**

k1 **=** ROL64**(**k1**,** 31**);**

k1 **\*=** 0x4cf5ad432745937fULL**;**

h1 **^=** k1**;**

h1 **=** ROL64**(**h1**,** 27**);**

h1 **=** h1 **\*** 5 **+** 0x52dce729**;**

**}**

h1 **^=** h1 **>>** 33**;**

h1 **\*=** 0xff51afd7ed558ccdULL**;**

h1 **^=** h1 **>>** 33**;**

h1 **\*=** 0xc4ceb9fe1a85ec53ULL**;**

h1 **^=** h1 **>>** 33**;**

**return** h1**;**

**}**

### 13.2.5 更新q->exec\_cksum

**if** **(**q**->**exec\_cksum **!=** cksum**)** **{**

u8 hnb **=** has\_new\_bits**(**virgin\_bits**);//检查是否有新发现，见另一专题**

**if** **(**hnb **>** new\_bits**)** new\_bits **=** hnb**;**

**if** **(**q**->**exec\_cksum**)** **{**

u32 i**;**

**for** **(**i **=** 0**;** i **<** MAP\_SIZE**;** i**++)** **{**

**if** **(!**var\_bytes**[**i**]** **&&** first\_trace**[**i**]** **!=** trace\_bits**[**i**])** **{**

var\_bytes**[**i**]** **=** 1**;**

stage\_max **=** CAL\_CYCLES\_LONG**;**

**}**

**}**

var\_detected **=** 1**;**

**}** **else** **{**

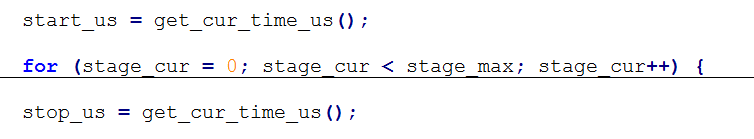
q**->**exec\_cksum **=** cksum**;//更新q->exec\_cksum**

memcpy**(**first\_trace**,** trace\_bits**,** MAP\_SIZE**);**

**}**

**}**

### 13.2.6 统计信息



q**->**exec\_us **=** **(**stop\_us **-** start\_us**)** **/** stage\_max**;**

q**->**bitmap\_size **=** count\_bytes**(**trace\_bits**);//遍历的touple个数**

total\_bitmap\_size **+=** q**->**bitmap\_size**;**

total\_bitmap\_entries**++;**

### 13.2.9 /update\_bitmap\_score 找到favorite，赋值给top\_rated

/\* When we bump into a new path, we call this to see if the path appears

more "favorable" than any of the existing ones. The purpose of the

"favorables" is to have a minimal set of paths that trigger all the bits

seen in the bitmap so far, and focus on fuzzing them at the expense of

the rest.

The first step of the process is to maintain a list of top\_rated[] entries

for every byte in the bitmap. We win that slot if there is no previous

contender, or if the contender has a more favorable speed x size factor. \*/

static void update\_bitmap\_score**(**struct queue\_entry**\*** q**)** **{**

u32 i**;**

u64 fav\_factor **=** q**->**exec\_us **\*** q**->**len**;//只与执行时间和文件长度有关**

/\* For every byte set in trace\_bits[], see if there is a previous winner,

and how it compares to us. \*/

**for** **(**i **=** 0**;** i **<** MAP\_SIZE**;** i**++)**

**if** **(**trace\_bits**[**i**])** **{**

**if** **(**top\_rated**[**i**])** **{**

/\* Faster-executing or smaller test cases are favored. \*/

**if** **(**fav\_factor **>** top\_rated**[**i**]->**exec\_us **\*** top\_rated**[**i**]->**len**)** **continue;//没有原来的好**

/\* Looks like we're going to win. Decrease ref count for the

previous winner, discard its trace\_bits[] if necessary. \*/

**if** **(!--**top\_rated**[**i**]->**tc\_ref**)** **{**

ck\_free**(**top\_rated**[**i**]->**trace\_mini**);**

top\_rated**[**i**]->**trace\_mini **=** 0**;**

**}**

**}**

/\* Insert ourselves as the new winner. \*/

top\_rated**[**i**]** **=** q**;//当前的q是最好的，赋值给**top\_rated**[**i**]**

q**->**tc\_ref**++;**

**if** **(!**q**->**trace\_mini**)** **{**

q**->**trace\_mini **=** ck\_alloc**(**MAP\_SIZE **>>** 3**);**

minimize\_bits**(**q**->**trace\_mini**,** trace\_bits**);**

**}**

score\_changed **=** 1**;**

**}**

**}**

### 13.2.8 返回fault

fault **=** run\_target**(**argv**,** use\_tmout**);**

**return** fault**;**

## 13.3 结果FAULT\_NONE

**switch** **(**res**)** **{**

**case** FAULT\_NONE**:**

**if** **(**q **==** queue**)** check\_map\_coverage**();**

**if** **(**crash\_mode**)** FATAL**(**"Test case '%s' does \*NOT\* crash"**,** fn**);**

**break;**

## 13.4 结果FAULT\_TMOUT

**case** FAULT\_TMOUT**:**

**if** **(**timeout\_given**)** **{**

/\* The -t nn+ syntax in the command line sets timeout\_given to '2' and

instructs afl-fuzz to tolerate but skip queue entries that time

out. \*/

**if** **(**timeout\_given **>** 1**)** **{**

WARNF**(**"Test case results in a timeout (skipping)"**);//如果有+，跳过这个样例**

q**->**cal\_failed **=** CAL\_CHANCES**;**

cal\_failures**++;**

**break;**

**}**

SAYF**(**"\n" cLRD "[-] " cRST

"The program took more than %u ms to process one of the initial test cases.\n"

" Usually, the right thing to do is to relax the -t option - or to delete it\n"

" altogether and allow the fuzzer to auto-calibrate. That said, if you know\n"

" what you are doing and want to simply skip the unruly test cases, append\n"

" '+' at the end of the value passed to -t ('-t %u+').\n"**,** exec\_tmout**,**

exec\_tmout**);**

FATAL**(**"Test case '%s' results in a timeout"**,** fn**);//否则报错**

**}** **else** **{**

SAYF**(**"\n" cLRD "[-] " cRST

"The program took more than %u ms to process one of the initial test cases.\n"

" This is bad news; raising the limit with the -t option is possible, but\n"

" will probably make the fuzzing process extremely slow.\n\n"

" If this test case is just a fluke, the other option is to just avoid it\n"

" altogether, and find one that is less of a CPU hog.\n"**,** exec\_tmout**);**

FATAL**(**"Test case '%s' results in a timeout"**,** fn**);**

**}**

## 13.5 结果crash

**case** FAULT\_CRASH**:**

**if** **(**crash\_mode**)** **break;**

**if** **(**skip\_crashes**)** **{**

WARNF**(**"Test case results in a crash (skipping)"**);**

q**->**cal\_failed **=** CAL\_CHANCES**;**

cal\_failures**++;**

**break;**

**}**

# 14. show\_stats 显示状态

/\* A spiffy retro stats screen! This is called every stats\_update\_freq

execve() calls, plus in several other circumstances. \*/

static void show\_stats**(**void**)** **{**

## 14.1 write\_stats\_file更新fuzzer\_stats文件

static void write\_stats\_file**(**double bitmap\_cvg**,** double stability**,** double eps**)** **{**

static double last\_bcvg**,** last\_stab**,** last\_eps**;**

u8**\*** fn **=** alloc\_printf**(**"%s/fuzzer\_stats"**,** out\_dir**);**

s32 fd**;**

FILE**\*** f**;**

fd **=** open**(**fn**,** O\_WRONLY **|** O\_CREAT **|** O\_TRUNC**,** 0600**);**

**if** **(**fd **<** 0**)** PFATAL**(**"Unable to create '%s'"**,** fn**);**

ck\_free**(**fn**);**

f **=** fdopen**(**fd**,** "w"**);**

**if** **(!**f**)** PFATAL**(**"fdopen() failed"**);**

/\* Keep last values in case we're called from another context

where exec/sec stats and such are not readily available. \*/

**if** **(!**bitmap\_cvg **&&** **!**stability **&&** **!**eps**)** **{**

bitmap\_cvg **=** last\_bcvg**;**

stability **=** last\_stab**;**

eps **=** last\_eps**;**

**}** **else** **{**

last\_bcvg **=** bitmap\_cvg**;**

last\_stab **=** stability**;**

last\_eps **=** eps**;**

**}**

fprintf**(**f**,** "start\_time : %llu\n"

"last\_update : %llu\n"

"fuzzer\_pid : %u\n"

"cycles\_done : %llu\n"

"execs\_done : %llu\n"

"execs\_per\_sec : %0.02f\n"

"paths\_total : %u\n"

"paths\_favored : %u\n"

"paths\_found : %u\n"

"paths\_imported : %u\n"

"max\_depth : %u\n"

"cur\_path : %u\n"

"pending\_favs : %u\n"

"pending\_total : %u\n"

"variable\_paths : %u\n"

"stability : %0.02f%%\n"

"bitmap\_cvg : %0.02f%%\n"

"unique\_crashes : %llu\n"

"unique\_hangs : %llu\n"

"last\_path : %llu\n"

"last\_crash : %llu\n"

"last\_hang : %llu\n"

"execs\_since\_crash : %llu\n"

"exec\_timeout : %u\n"

"afl\_banner : %s\n"

"afl\_version : " VERSION "\n"

"command\_line : %s\n"**,**

start\_time **/** 1000**,** get\_cur\_time**()** **/** 1000**,** getpid**(),**

queue\_cycle **?** **(**queue\_cycle **-** 1**)** **:** 0**,** total\_execs**,** eps**,**

queued\_paths**,** queued\_favored**,** queued\_discovered**,** queued\_imported**,**

max\_depth**,** current\_entry**,** pending\_favored**,** pending\_not\_fuzzed**,**

queued\_variable**,** stability**,** bitmap\_cvg**,** unique\_crashes**,**

unique\_hangs**,** last\_path\_time **/** 1000**,** last\_crash\_time **/** 1000**,**

last\_hang\_time **/** 1000**,** total\_execs **-** last\_crash\_execs**,**

exec\_tmout**,** use\_banner**,** orig\_cmdline**);**

/\* ignore errors \*/

fclose**(**f**);**

**}**

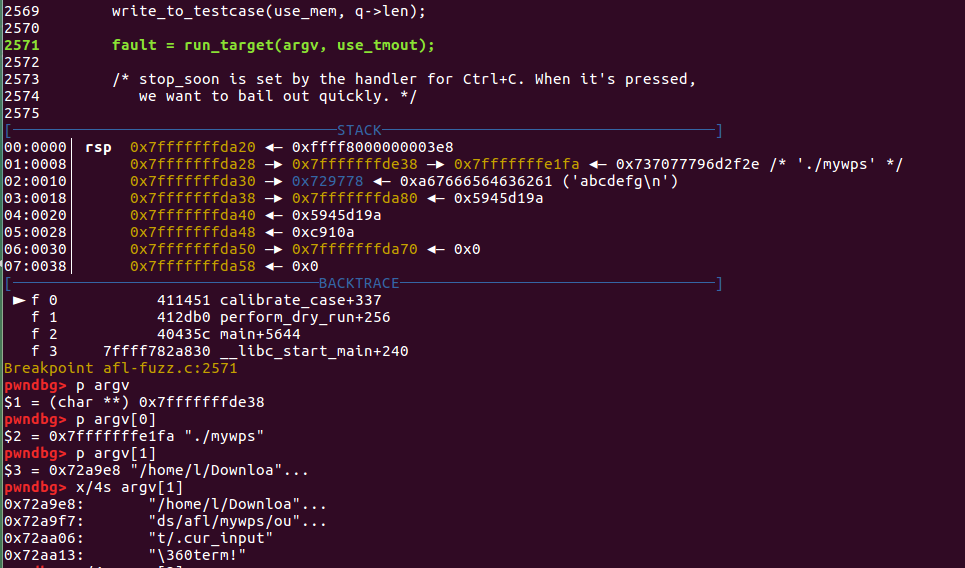
# 15. run\_target

/\* Execute target application, monitoring for timeouts. Return status

information. The called program will update trace\_bits[]. \*/

static u8 run\_target**(**char**\*\*** argv**,** u32 timeout**)** **{**

## 15.1 run\_target的参数



## 15.2 forkserver模式

**else** **{**

s32 res**;**

/\* In non-dumb mode, we have the fork server up and running, so simply

tell it to have at it, and then read back PID. \*/

**if** **((**res **=** write**(**fsrv\_ctl\_fd**,** **&**prev\_timed\_out**,** 4**))** **!=** 4**)** **{//启动测试**

**if** **(**stop\_soon**)** **return** 0**;**

RPFATAL**(**res**,** "Unable to request new process from fork server (OOM?)"**);**

**}**

**if** **((**res **=** read**(**fsrv\_st\_fd**,** **&**child\_pid**,** 4**))** **!=** 4**)** **{//读取child\_pid，结合ida**

**if** **(**stop\_soon**)** **return** 0**;**

RPFATAL**(**res**,** "Unable to request new process from fork server (OOM?)"**);**

**}**

**if** **(**child\_pid **<=** 0**)** FATAL**(**"Fork server is misbehaving (OOM?)"**);**

**}**

## 15.3 设置超时

/\* Configure timeout, as requested by user, then wait for child to terminate. \*/

it**.**it\_value**.**tv\_sec **=** **(**timeout **/** 1000**);**

it**.**it\_value**.**tv\_usec **=** **(**timeout **%** 1000**)** **\*** 1000**;**

setitimer**(**ITIMER\_REAL**,** **&**it**,** **NULL);**

/\* The SIGALRM handler simply kills the child\_pid and sets child\_timed\_out. \*/

**if** **(**dumb\_mode **==** 1 **||** no\_forkserver**)** **{**

**if** **(**waitpid**(**child\_pid**,** **&**status**,** 0**)** **<=** 0**)** PFATAL**(**"waitpid() failed"**);**

**}**

**else** **{**

s32 res**;**

**if** **((**res **=** read**(**fsrv\_st\_fd**,** **&**status**,** 4**))** **!=** 4**)** **{ //读取forkserver传回来的etxit status**

**if** **(**stop\_soon**)** **return** 0**;**

RPFATAL**(**res**,** "Unable to communicate with fork server (OOM?)"**);**

**}**

**}**

**在这期间如果超时：**

child\_timed\_out **=** 1**;** //设置全局标记

kill**(**child\_pid**,** SIGKILL**);//kill 孙子进程**

## 15.4 查询本次测试，bitmap的变化

tb4 **=** **\*(**u32**\*)**trace\_bits**;**

#ifdef \_\_x86\_64\_\_

classify\_counts**((**u64**\*)**trace\_bits**);**

#else

classify\_counts**((**u32**\*)**trace\_bits**);**

#endif /\* ^\_\_x86\_64\_\_ \*/

static inline void classify\_counts**(**u64**\*** mem**)** **{**

u32 i **=** MAP\_SIZE **>>** 3**;**

**while** **(**i**--)** **{**

/\* Optimize for sparse bitmaps. \*/

**if** **(**unlikely**(\***mem**))** **{**

u16**\*** mem16 **=** **(**u16**\*)**mem**;**

mem16**[**0**]** **=** count\_class\_lookup16**[**mem16**[**0**]];//trace\_bits命中次数重新赋值，规约，1,2,4,8...**

mem16**[**1**]** **=** count\_class\_lookup16**[**mem16**[**1**]];**

mem16**[**2**]** **=** count\_class\_lookup16**[**mem16**[**2**]];**

mem16**[**3**]** **=** count\_class\_lookup16**[**mem16**[**3**]];**

**}**

mem**++;**

**}**

**}**

## 15.5 返回运行status

prev\_timed\_out **=** child\_timed\_out**;**

/\* Report outcome to caller. \*/

**if** **(**child\_timed\_out**)** **return** FAULT\_TMOUT**;//超时返回**

**//如果将此处的返回值改为**

**//if (child\_timed\_out) return FAULT\_TMOUT;改为 //liu 6/18**

**// if (child\_timed\_out) return FAULT\_NONE;就可以不总是hang了**

**if** **(**WIFSIGNALED**(**status**)** **&&** **!**stop\_soon**)** **{**

kill\_signal **=** WTERMSIG**(**status**);**

**return** FAULT\_CRASH**;//返回崩溃**

**}**

/\* A somewhat nasty hack for MSAN, which doesn't support abort\_on\_error and

must use a special exit code. \*/

**if** **(**uses\_asan **&&** WEXITSTATUS**(**status**)** **==** MSAN\_ERROR**)** **{**

kill\_signal **=** 0**;**

**return** FAULT\_CRASH**;**

**}**

**if** **((**dumb\_mode **==** 1 **||** no\_forkserver**)** **&&** tb4 **==** EXEC\_FAIL\_SIG**)**

**return** FAULT\_ERROR**;**

**return** FAULT\_NONE**;//返回正常**

**}**

# show\_init\_stats()显示原始样本结果

/\* Display quick statistics at the end of processing the input directory,

plus a bunch of warnings. Some calibration stuff also ended up here,

along with several hardcoded constants. Maybe clean up eventually. \*/

static void show\_init\_stats**(**void**)** **{**

struct queue\_entry**\*** q **=** queue**;**

u32 min\_bits **=** 0**,** max\_bits **=** 0**;**

u64 min\_us **=** 0**,** max\_us **=** 0**;**

u64 avg\_us **=** 0**;**

u32 max\_len **=** 0**;**

**if** **(**total\_cal\_cycles**)** avg\_us **=** total\_cal\_us **/** total\_cal\_cycles**;//矫正了8次，平均时间**

**while** **(**q**)** **{**

**if** **(!**min\_us **||** q**->**exec\_us **<** min\_us**)** min\_us **=** q**->**exec\_us**;**

**if** **(**q**->**exec\_us **>** max\_us**)** max\_us **=** q**->**exec\_us**;**

**if** **(!**min\_bits **||** q**->**bitmap\_size **<** min\_bits**)** min\_bits **=** q**->**bitmap\_size**;**

**if** **(**q**->**bitmap\_size **>** max\_bits**)** max\_bits **=** q**->**bitmap\_size**;**

**if** **(**q**->**len **>** max\_len**)** max\_len **=** q**->**len**;//寻找最值**

q **=** q**->**next**;**

**}**

SAYF**(**"\n"**);**

**if** **(**avg\_us **>** **(**qemu\_mode **?** 50000 **:** 10000**))**

WARNF**(**cLRD "The target binary is pretty slow! See %s/perf\_tips.txt."**,**

doc\_path**);**

/\* Let's keep things moving with slow binaries. \*/

**if** **(**avg\_us **>** 50000**)** havoc\_div **=** 10**;** /\* 0-19 execs/sec \*/

**else** **if** **(**avg\_us **>** 20000**)** havoc\_div **=** 5**;** /\* 20-49 execs/sec \*/

**else** **if** **(**avg\_us **>** 10000**)** havoc\_div **=** 2**;** /\* 50-100 execs/sec \*/

**if** **(!**resuming\_fuzz**)** **{**

**if** **(**max\_len **>** 50 **\*** 1024**)**

WARNF**(**cLRD "Some test cases are huge (%s) - see %s/perf\_tips.txt!"**,**

DMS**(**max\_len**),** doc\_path**);**

**else** **if** **(**max\_len **>** 10 **\*** 1024**)**

WARNF**(**"Some test cases are big (%s) - see %s/perf\_tips.txt."**,**

DMS**(**max\_len**),** doc\_path**);**

**if** **(**useless\_at\_start **&&** **!**in\_bitmap**)**

WARNF**(**cLRD "Some test cases look useless. Consider using a smaller set."**);**

**if** **(**queued\_paths **>** 100**)**

WARNF**(**cLRD "You probably have far too many input files! Consider trimming down."**);**

**else** **if** **(**queued\_paths **>** 20**)**

WARNF**(**"You have lots of input files; try starting small."**);**

**}**

OKF**(**"Here are some useful stats:\n\n"

cGRA " Test case count : " cRST "%u favored, %u variable, %u total\n"

cGRA " Bitmap range : " cRST "%u to %u bits (average: %0.02f bits)\n"

cGRA " Exec timing : " cRST "%s to %s us (average: %s us)\n"**,**

queued\_favored**,** queued\_variable**,** queued\_paths**,** min\_bits**,** max\_bits**,** //发现新tuple的数量范围range

**((**double**)**total\_bitmap\_size**)** **/** **(**total\_bitmap\_entries **?** total\_bitmap\_entries **:** 1**),**

DI**(**min\_us**),** DI**(**max\_us**),** DI**(**avg\_us**));**

**if** **(!**timeout\_given**)** **{**

/\* Figure out the appropriate timeout. The basic idea is: 5x average or

1x max, rounded up to EXEC\_TM\_ROUND ms and capped at 1 second.

If the program is slow, the multiplier is lowered to 2x or 3x, because

random scheduler jitter is less likely to have any impact, and because

our patience is wearing thin =) \*/

**if** **(**avg\_us **>** 50000**)** exec\_tmout **=** avg\_us **\*** 2 **/** 1000**;**

**else** **if** **(**avg\_us **>** 10000**)** exec\_tmout **=** avg\_us **\*** 3 **/** 1000**;**

**else** exec\_tmout **=** avg\_us **\*** 5 **/** 1000**;**

exec\_tmout **=** MAX**(**exec\_tmout**,** max\_us **/** 1000**);**

exec\_tmout **=** **(**exec\_tmout **+** EXEC\_TM\_ROUND**)** **/** EXEC\_TM\_ROUND **\*** EXEC\_TM\_ROUND**;**

**if** **(**exec\_tmout **>** EXEC\_TIMEOUT**)** exec\_tmout **=** EXEC\_TIMEOUT**;**

ACTF**(**"No -t option specified, so I'll use exec timeout of %u ms."**,**

exec\_tmout**);**

timeout\_given **=** 1**;**

**}** **else** **if** **(**timeout\_given **==** 3**)** **{**

ACTF**(**"Applying timeout settings from resumed session (%u ms)."**,** exec\_tmout**);**

**}**

/\* In dumb mode, re-running every timing out test case with a generous time

limit is very expensive, so let's select a more conservative default. \*/

**if** **(**dumb\_mode **&&** **!**getenv**(**"AFL\_HANG\_TMOUT"**))**

hang\_tmout **=** MIN**(**EXEC\_TIMEOUT**,** exec\_tmout **\*** 2 **+** 100**);**

OKF**(**"All set and ready to roll!"**);**

**}**

# 18.find\_start\_position

seek\_to **=** find\_start\_position**();**

/\* When resuming, try to find the queue position to start from. This makes sense

only when resuming, and when we can find the original fuzzer\_stats. \*/

static u32 find\_start\_position**(**void**)** **{**

static u8 tmp**[**4096**];** /\* Ought to be enough for anybody. \*/

u8 **\***fn**,** **\***off**;**

s32 fd**,** i**;**

u32 ret**;**

**if** **(!**resuming\_fuzz**)** **return** 0**;**

**if** **(**in\_place\_resume**)** fn **=** alloc\_printf**(**"%s/fuzzer\_stats"**,** out\_dir**);**

**else** fn **=** alloc\_printf**(**"%s/../fuzzer\_stats"**,** in\_dir**);**

fd **=** open**(**fn**,** O\_RDONLY**);**

ck\_free**(**fn**);**

**if** **(**fd **<** 0**)** **return** 0**;**

i **=** read**(**fd**,** tmp**,** **sizeof(**tmp**)** **-** 1**);** **(**void**)**i**;** /\* Ignore errors \*/

close**(**fd**);**

off **=** strstr**(**tmp**,** "cur\_path : "**);**

**if** **(!**off**)** **return** 0**;**

ret **=** atoi**(**off **+** 17**);**

**if** **(**ret **>=** queued\_paths**)** ret **=** 0**;**

**return** ret**;**

**}**

## 18.1 resuming\_fuzz

如果文件名是id开头的

**if** **(!**strncmp**(**rsl**,** CASE\_PREFIX**,** 3**)** **&&**

sscanf**(**rsl **+** 3**,** "%06u"**,** **&**orig\_id**)** **==** 1 **&&** orig\_id **==** id**)** **{**

u8**\*** src\_str**;**

u32 src\_id**;**

resuming\_fuzz **=** 1**;**

## 18.2 in\_place\_resume

如果输入参数是-i -

**case** 'i'**:** /\* input dir \*/

**if** **(**in\_dir**)** FATAL**(**"Multiple -i options not supported"**);**

in\_dir **=** optarg**;**

**if** **(!**strcmp**(**in\_dir**,** "-"**))** in\_place\_resume **=** 1**;**

# write\_stats\_file(0, 0, 0);

/\* Update stats file for unattended monitoring. \*/

static void write\_stats\_file**(**double bitmap\_cvg**,** double stability**,** double eps**)** **{**

static double last\_bcvg**,** last\_stab**,** last\_eps**;**

u8**\*** fn **=** alloc\_printf**(**"%s/fuzzer\_stats"**,** out\_dir**);**

s32 fd**;**

FILE**\*** f**;**

fd **=** open**(**fn**,** O\_WRONLY **|** O\_CREAT **|** O\_TRUNC**,** 0600**);**

**if** **(**fd **<** 0**)** PFATAL**(**"Unable to create '%s'"**,** fn**);**

ck\_free**(**fn**);**

f **=** fdopen**(**fd**,** "w"**);**

**if** **(!**f**)** PFATAL**(**"fdopen() failed"**);**

/\* Keep last values in case we're called from another context

where exec/sec stats and such are not readily available. \*/

**if** **(!**bitmap\_cvg **&&** **!**stability **&&** **!**eps**)** **{**

bitmap\_cvg **=** last\_bcvg**;**

stability **=** last\_stab**;**

eps **=** last\_eps**;**

**}** **else** **{**

last\_bcvg **=** bitmap\_cvg**;**

last\_stab **=** stability**;**

last\_eps **=** eps**;**

**}**

fprintf**(**f**,** "start\_time : %llu\n"

"last\_update : %llu\n"

"fuzzer\_pid : %u\n"

"cycles\_done : %llu\n"

"execs\_done : %llu\n"

"execs\_per\_sec : %0.02f\n"

"paths\_total : %u\n"

"paths\_favored : %u\n"

"paths\_found : %u\n"

"paths\_imported : %u\n"

"max\_depth : %u\n"

"cur\_path : %u\n"

"pending\_favs : %u\n"

"pending\_total : %u\n"

"variable\_paths : %u\n"

"stability : %0.02f%%\n"

"bitmap\_cvg : %0.02f%%\n"

"unique\_crashes : %llu\n"

"unique\_hangs : %llu\n"

"last\_path : %llu\n"

"last\_crash : %llu\n"

"last\_hang : %llu\n"

"execs\_since\_crash : %llu\n"

"exec\_timeout : %u\n"

"afl\_banner : %s\n"

"afl\_version : " VERSION "\n"

"command\_line : %s\n"**,**

start\_time **/** 1000**,** get\_cur\_time**()** **/** 1000**,** getpid**(),**

queue\_cycle **?** **(**queue\_cycle **-** 1**)** **:** 0**,** total\_execs**,** eps**,**

queued\_paths**,** queued\_favored**,** queued\_discovered**,** queued\_imported**,**

max\_depth**,** current\_entry**,** pending\_favored**,** pending\_not\_fuzzed**,**

queued\_variable**,** stability**,** bitmap\_cvg**,** unique\_crashes**,**

unique\_hangs**,** last\_path\_time **/** 1000**,** last\_crash\_time **/** 1000**,**

last\_hang\_time **/** 1000**,** total\_execs **-** last\_crash\_execs**,**

exec\_tmout**,** use\_banner**,** orig\_cmdline**);**

/\* ignore errors \*/

fclose**(**f**);**

**}**

## 19.1 start\_time

Main初始化的时候获取的，本次fuzz的起始时间。

start\_time **=** get\_cur\_time**();**

/\* Get unix time in milliseconds \*/

static u64 get\_cur\_time**(**void**)** **{**

struct timeval tv**;**

struct timezone tz**;**

gettimeofday**(&**tv**,** **&**tz**);**

**return** **(**tv**.**tv\_sec **\*** 1000ULL**)** **+** **(**tv**.**tv\_usec **/** 1000**);**

**}**

## 19.2 /queue\_cycle Queue round counter

queue\_cycle**,** /\* Queue round counter \*/

在每一轮开始的时候，加一,main函数中

**while** **(**1**)** **{**

u8 skipped\_fuzz**;**

cull\_queue**();**

**if** **(!**queue\_cur**)** **{**

queue\_cycle**++;**

## 19.3 total\_execs Total execve() calls

total\_execs**,** /\* Total execve() calls \*/

**else** **{**

s32 res**;**

**if** **((**res **=** read**(**fsrv\_st\_fd**,** **&**status**,** 4**))** **!=** 4**)** **{**

**if** **(**stop\_soon**)** **return** 0**;**

RPFATAL**(**res**,** "Unable to communicate with fork server (OOM?)"**);**

**}**

**}**

**if** **(!**WIFSTOPPED**(**status**))** child\_pid **=** 0**;**

it**.**it\_value**.**tv\_sec **=** 0**;**

it**.**it\_value**.**tv\_usec **=** 0**;**

setitimer**(**ITIMER\_REAL**,** **&**it**,** **NULL);**

total\_execs**++;//在run\_target里面，返回status以后**

## 19.4 eps 平均每秒钟exec次数

avg\_exec **=** **((**double**)**total\_execs**)** **\*** 1000 **/** **(**cur\_ms **-** start\_time**);参数传递过来的**

## 19.5 /queued\_paths queue队列的大小

static void add\_to\_queue**(**u8**\*** fname**,** u32 len**,** u8 passed\_det**)** **{**

struct queue\_entry**\*** q **=** ck\_alloc**(sizeof(**struct queue\_entry**));**

q**->**fname **=** fname**;**

q**->**len **=** len**;**

q**->**depth **=** cur\_depth **+** 1**;**

q**->**passed\_det **=** passed\_det**;**

**if** **(**q**->**depth **>** max\_depth**)** max\_depth **=** q**->**depth**;**

**if** **(**queue\_top**)** **{**

queue\_top**->**next **=** q**;**

queue\_top **=** q**;**

**}** **else** q\_prev100 **=** queue **=** queue\_top **=** q**;**

queued\_paths**++;**

## 19.6 queued\_favored top\_rated数组中非零值的个数

static void cull\_queue**(**void**)** **{**

struct queue\_entry**\*** q**;**

static u8 temp\_v**[**MAP\_SIZE **>>** 3**];**

u32 i**;**

**if** **(**dumb\_mode **||** **!**score\_changed**)** **return;**

score\_changed **=** 0**;**

memset**(**temp\_v**,** 255**,** MAP\_SIZE **>>** 3**);**

queued\_favored **=** 0**;**

pending\_favored **=** 0**;**

q **=** queue**;**

**while** **(**q**)** **{**

q**->**favored **=** 0**;**

q **=** q**->**next**;**

**}**

/\* Let's see if anything in the bitmap isn't captured in temp\_v.

If yes, and if it has a top\_rated[] contender, let's use it. \*/

**for** **(**i **=** 0**;** i **<** MAP\_SIZE**;** i**++)**

**if** **(**top\_rated**[**i**]** **&&** **(**temp\_v**[**i **>>** 3**]** **&** **(**1 **<<** **(**i **&** 7**))))** **{**

u32 j **=** MAP\_SIZE **>>** 3**;**

/\* Remove all bits belonging to the current entry from temp\_v. \*/

**while** **(**j**--)**

**if** **(**top\_rated**[**i**]->**trace\_mini**[**j**])**

temp\_v**[**j**]** **&=** **~**top\_rated**[**i**]->**trace\_mini**[**j**];**

top\_rated**[**i**]->**favored **=** 1**;**

queued\_favored**++;**

## 19.7 queued\_discovered, /\* Items discovered during this run \*/

## 19.8 total\_cal\_us

Calibrate 总时间，在calibrate\_case函数里

total\_cal\_us **+=** stop\_us **-** start\_us**;**

## 19.9 total\_cal\_cycles

Calibrate 次数，也在calibrate\_case函数里

total\_cal\_cycles **+=** stage\_max**;**

## 19.10 total\_bitmap\_size total\_bitmap\_entries

也在calibrate\_case函数里

static u64 total\_bitmap\_size**,** /\* Total bit count for all bitmaps \*/

total\_bitmap\_entries**;** /\* Number of bitmaps counted \*/

q**->**bitmap\_size **=** count\_bytes**(**trace\_bits**);**

total\_bitmap\_size **+=** q**->**bitmap\_size**;**

total\_bitmap\_entries**++;**

## 19.11 q->depth

static void add\_to\_queue**(**u8**\*** fname**,** u32 len**,** u8 passed\_det**)** **{**

struct queue\_entry**\*** q **=** ck\_alloc**(sizeof(**struct queue\_entry**));**

q**->**fname **=** fname**;**

q**->**len **=** len**;**

q**->**depth **=** cur\_depth **+** 1**;**

## 19.12 cur\_depth

cur\_depth**,** /\* Current path depth \*/

cur\_depth **=** queue\_cur**->**depth**; //fuzz\_one函数变异前获取**

## 19.13 current\_entry

current\_entry**,** /\* Current queue entry ID \*/

### 19.13.1 main while循环起始处赋值0

**while** **(**1**)** **{**

u8 skipped\_fuzz**;**

cull\_queue**();**

**if** **(!**queue\_cur**)** **{**

queue\_cycle**++;**

current\_entry **=** 0**;**

### 19.13.2 在while循环结尾加一

current\_entry**++;**

## 19.14 /unique\_crashes

unique\_crashes**,** /\* Crashes with unique signatures \*/

unique\_crashes**++;//在save\_if\_interesting函数中加一**

## 19.15 /kill\_signal

在run\_target函数中

**if** **(**WIFSIGNALED**(**status**)** **&&** **!**stop\_soon**)** **{**

kill\_signal **=** WTERMSIG**(**status**);**

**return** FAULT\_CRASH**;**

**}**

## 19.16 map density

SAYF**(**" map density : %s%-21s " bSTG bV "\n"**,** t\_byte\_ratio **>** 70 **?** cLRD **:**

**((**t\_bytes **<** 200 **&&** **!**dumb\_mode**)** **?** cPIN **:** cRST**),** tmp**);**

# /add\_to\_queue

/\* Append new test case to the queue. \*/

static void add\_to\_queue**(**u8**\*** fname**,** u32 len**,** u8 passed\_det**)** **{**

struct queue\_entry**\*** q **=** ck\_alloc**(sizeof(**struct queue\_entry**));**

q**->**fname **=** fname**;**

q**->**len **=** len**;**

q**->**depth **=** cur\_depth **+** 1**;**

q**->**passed\_det **=** passed\_det**;**

**if** **(**q**->**depth **>** max\_depth**)** max\_depth **=** q**->**depth**;**

**if** **(**queue\_top**)** **{**

queue\_top**->**next **=** q**;//添加到队列**

queue\_top **=** q**;**

**}** **else** q\_prev100 **=** queue **=** queue\_top **=** q**;**

queued\_paths**++;**

pending\_not\_fuzzed**++;**

cycles\_wo\_finds **=** 0**;**

**if** **(!(**queued\_paths **%** 100**))** **{**

q\_prev100**->**next\_100 **=** q**;**

q\_prev100 **=** q**;**

**}**

last\_path\_time **=** get\_cur\_time**();**

**}**

# /calculate\_score

/\* Calculate case desirability score to adjust the length of havoc fuzzing.

A helper function for fuzz\_one(). Maybe some of these constants should

go into config.h. \*/

static u32 calculate\_score**(**struct queue\_entry**\*** q**)** **{**

u32 avg\_exec\_us **=** total\_cal\_us **/** total\_cal\_cycles**;**

u32 avg\_bitmap\_size **=** total\_bitmap\_size **/** total\_bitmap\_entries**;**

u32 perf\_score **=** 100**;**

/\* Adjust score based on execution speed of this path, compared to the

global average. Multiplier ranges from 0.1x to 3x. Fast inputs are

less expensive to fuzz, so we're giving them more air time. \*/

**if** **(**q**->**exec\_us **\*** 0.1 **>** avg\_exec\_us**)** perf\_score **=** 10**; //执行时间越长，分数越低**

**else** **if** **(**q**->**exec\_us **\*** 0.25 **>** avg\_exec\_us**)** perf\_score **=** 25**;**

**else** **if** **(**q**->**exec\_us **\*** 0.5 **>** avg\_exec\_us**)** perf\_score **=** 50**;**

**else** **if** **(**q**->**exec\_us **\*** 0.75 **>** avg\_exec\_us**)** perf\_score **=** 75**;**

**else** **if** **(**q**->**exec\_us **\*** 4 **<** avg\_exec\_us**)** perf\_score **=** 300**;**

**else** **if** **(**q**->**exec\_us **\*** 3 **<** avg\_exec\_us**)** perf\_score **=** 200**;**

**else** **if** **(**q**->**exec\_us **\*** 2 **<** avg\_exec\_us**)** perf\_score **=** 150**;**

/\* Adjust score based on bitmap size. The working theory is that better

coverage translates to better targets. Multiplier from 0.25x to 3x. \*/

**if** **(**q**->**bitmap\_size **\*** 0.3 **>** avg\_bitmap\_size**)** perf\_score **\*=** 3**;//**bitmap\_size越大越好

**else** **if** **(**q**->**bitmap\_size **\*** 0.5 **>** avg\_bitmap\_size**)** perf\_score **\*=** 2**;**

**else** **if** **(**q**->**bitmap\_size **\*** 0.75 **>** avg\_bitmap\_size**)** perf\_score **\*=** 1.5**;**

**else** **if** **(**q**->**bitmap\_size **\*** 3 **<** avg\_bitmap\_size**)** perf\_score **\*=** 0.25**;**

**else** **if** **(**q**->**bitmap\_size **\*** 2 **<** avg\_bitmap\_size**)** perf\_score **\*=** 0.5**;**

**else** **if** **(**q**->**bitmap\_size **\*** 1.5 **<** avg\_bitmap\_size**)** perf\_score **\*=** 0.75**;**

/\* Adjust score based on handicap. Handicap is proportional to how late

in the game we learned about this path. Latecomers are allowed to run

for a bit longer until they catch up with the rest. \*/

**if** **(**q**->**handicap **>=** 4**)** **{**

perf\_score **\*=** 4**;**

q**->**handicap **-=** 4**;**

**}** **else** **if** **(**q**->**handicap**)** **{**

perf\_score **\*=** 2**;**

q**->**handicap**--;**

**}**

/\* Final adjustment based on input depth, under the assumption that fuzzing

deeper test cases is more likely to reveal stuff that can't be

discovered with traditional fuzzers. \*/

**switch** **(**q**->**depth**)** **{**

**case** 0 **...** 3**:** **break;**

**case** 4 **...** 7**:** perf\_score **\*=** 2**;** **break;**

**case** 8 **...** 13**:** perf\_score **\*=** 3**;** **break;**

**case** 14 **...** 25**:** perf\_score **\*=** 4**;** **break;//越深发现漏洞可能性越高**

**default:** perf\_score **\*=** 5**;**

**}**

/\* Make sure that we don't go over limit. \*/

**if** **(**perf\_score **>** HAVOC\_MAX\_MULT **\*** 100**)** perf\_score **=** HAVOC\_MAX\_MULT **\*** 100**;**

**return** perf\_score**;**

**}**

# /common\_fuzz\_stuff

/\* Write a modified test case, run program, process results. Handle

error conditions, returning 1 if it's time to bail out. This is

a helper function for fuzz\_one(). \*/

EXP\_ST u8 common\_fuzz\_stuff**(**char**\*\*** argv**,** u8**\*** out\_buf**,** u32 len**)** **{**

u8 fault**;**

**if** **(**post\_handler**)** **{**

out\_buf **=** post\_handler**(**out\_buf**,** **&**len**);**

**if** **(!**out\_buf **||** **!**len**)** **return** 0**;**

**}**

write\_to\_testcase**(**out\_buf**,** len**);//写到文件中**

fault **=** run\_target**(**argv**,** exec\_tmout**);//运行被测试软件**

**if** **(**stop\_soon**)** **return** 1**;**

**if** **(**fault **==** FAULT\_TMOUT**)** **{**

**if** **(**subseq\_tmouts**++** **>** TMOUT\_LIMIT**)** **{**

cur\_skipped\_paths**++;**

**return** 1**;**

**}**

**}** **else** subseq\_tmouts **=** 0**;**

/\* Users can hit us with SIGUSR1 to request the current input

to be abandoned. \*/

**if** **(**skip\_requested**)** **{**

skip\_requested **=** 0**;**

cur\_skipped\_paths**++;**

**return** 1**;**

**}**

/\* This handles FAULT\_ERROR for us: \*/

queued\_discovered **+=** save\_if\_interesting**(**argv**,** out\_buf**,** len**,** fault**);//如果有新发现就连入队列**

**if** **(!(**stage\_cur **%** stats\_update\_freq**)** **||** stage\_cur **+** 1 **==** stage\_max**)**

show\_stats**();//以一定的时间间隔，显示状态**

**return** 0**;**

**}**

## 22.1 /save\_if\_interesting

保存fuzz获得的新样本

/\* Check if the result of an execve() during routine fuzzing is interesting,

save or queue the input test case for further analysis if so. Returns 1 if

entry is saved, 0 otherwise. \*/

static u8 save\_if\_interesting**(**char**\*\*** argv**,** void**\*** mem**,** u32 len**,** u8 fault**)** **{**

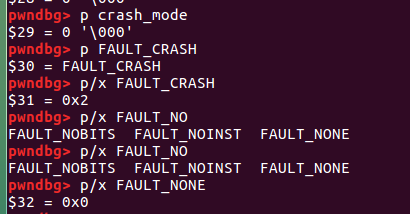
u8 **\***fn **=** ""**;**

u8 hnb**;**

s32 fd**;**

u8 keeping **=** 0**,** res**;**

**if** **(**fault **==** crash\_mode**)** **{**



**/////////////////**

**case** 'C'**:** /\* crash mode \*/

**if** **(**crash\_mode**)** FATAL**(**"Multiple -C options not supported"**);**

crash\_mode **=** FAULT\_CRASH**;**

**break;**

**////////////////**

/\* Keep only if there are new bits in the map, add to queue for

future fuzzing, etc. \*/

**if** **(!(**hnb **=** has\_new\_bits**(**virgin\_bits**)))** **{//如果没有新发现，就直接返回0**

**if** **(**crash\_mode**)** total\_crashes**++;**

**return** 0**;**

**}**

#ifndef SIMPLE\_FILES

fn **=** alloc\_printf**(**"%s/queue/id:%06u,%s"**,** out\_dir**,** queued\_paths**,**

describe\_op**(**hnb**));//保存的文件名**

#else

fn **=** alloc\_printf**(**"%s/queue/id\_%06u"**,** out\_dir**,** queued\_paths**);**

#endif /\* ^!SIMPLE\_FILES \*/

add\_to\_queue**(**fn**,** len**,** 0**);//添加到队列**

**if** **(**hnb **==** 2**)** **{//有新的发现**

queue\_top**->**has\_new\_cov **=** 1**;**

queued\_with\_cov**++;**

**}**

queue\_top**->**exec\_cksum **=** hash32**(**trace\_bits**,** MAP\_SIZE**,** HASH\_CONST**);**

/\* Try to calibrate inline; this also calls update\_bitmap\_score() when

successful. \*/

res **=** calibrate\_case**(**argv**,** queue\_top**,** mem**,** queue\_cycle **-** 1**,** 0**);**

**if** **(**res **==** FAULT\_ERROR**)**

FATAL**(**"Unable to execute target application"**);**

fd **=** open**(**fn**,** O\_WRONLY **|** O\_CREAT **|** O\_EXCL**,** 0600**);**

**if** **(**fd **<** 0**)** PFATAL**(**"Unable to create '%s'"**,** fn**);**

ck\_write**(**fd**,** mem**,** len**,** fn**);//写文件**

close**(**fd**);**

keeping **=** 1**;**

**}**

**switch** **(**fault**)** **{**

**case** FAULT\_TMOUT**:**

/\* Timeouts are not very interesting, but we're still obliged to keep

a handful of samples. We use the presence of new bits in the

hang-specific bitmap as a signal of uniqueness. In "dumb" mode, we

just keep everything. \*/

total\_tmouts**++;**

**if** **(**unique\_hangs **>=** KEEP\_UNIQUE\_HANG**)** **return** keeping**;**

**if** **(!**dumb\_mode**)** **{**

#ifdef \_\_x86\_64\_\_

simplify\_trace**((**u64**\*)**trace\_bits**);**

#else

simplify\_trace**((**u32**\*)**trace\_bits**);**

#endif /\* ^\_\_x86\_64\_\_ \*/

**if** **(!**has\_new\_bits**(**virgin\_tmout**))** **return** keeping**;**

**}**

unique\_tmouts**++;**

/\* Before saving, we make sure that it's a genuine hang by re-running

the target with a more generous timeout (unless the default timeout

is already generous). \*/

**if** **(**exec\_tmout **<** hang\_tmout**)** **{**

u8 new\_fault**;**

write\_to\_testcase**(**mem**,** len**);**

new\_fault **=** run\_target**(**argv**,** hang\_tmout**);**

**if** **(**stop\_soon **||** new\_fault **!=** FAULT\_TMOUT**)** **return** keeping**;**

**}**

#ifndef SIMPLE\_FILES

fn **=** alloc\_printf**(**"%s/hangs/id:%06llu,%s"**,** out\_dir**,**

unique\_hangs**,** describe\_op**(**0**));**

#else

fn **=** alloc\_printf**(**"%s/hangs/id\_%06llu"**,** out\_dir**,**

unique\_hangs**);**

#endif /\* ^!SIMPLE\_FILES \*/

unique\_hangs**++;**

last\_hang\_time **=** get\_cur\_time**();**

**break;**

**case** FAULT\_CRASH**:**

/\* This is handled in a manner roughly similar to timeouts,

except for slightly different limits and no need to re-run test

cases. \*/

total\_crashes**++;**

**if** **(**unique\_crashes **>=** KEEP\_UNIQUE\_CRASH**)** **return** keeping**;**

**if** **(!**dumb\_mode**)** **{**

#ifdef \_\_x86\_64\_\_

simplify\_trace**((**u64**\*)**trace\_bits**);**

#else

simplify\_trace**((**u32**\*)**trace\_bits**);**

#endif /\* ^\_\_x86\_64\_\_ \*/

**if** **(!**has\_new\_bits**(**virgin\_crash**))** **return** keeping**;**

**}**

**if** **(!**unique\_crashes**)** write\_crash\_readme**();**

#ifndef SIMPLE\_FILES

fn **=** alloc\_printf**(**"%s/crashes/id:%06llu,sig:%02u,%s"**,** out\_dir**,**

unique\_crashes**,** kill\_signal**,** describe\_op**(**0**));//保存crash文件名**

#else

fn **=** alloc\_printf**(**"%s/crashes/id\_%06llu\_%02u"**,** out\_dir**,** unique\_crashes**,**

kill\_signal**);**

#endif /\* ^!SIMPLE\_FILES \*/

unique\_crashes**++;**

last\_crash\_time **=** get\_cur\_time**();**

last\_crash\_execs **=** total\_execs**;**

**break;**

**case** FAULT\_ERROR**:** FATAL**(**"Unable to execute target application"**);**

**default:** **return** keeping**;**

**}**

/\* If we're here, we apparently want to save the crash or hang

test case, too. \*/

fd **=** open**(**fn**,** O\_WRONLY **|** O\_CREAT **|** O\_EXCL**,** 0600**);**

**if** **(**fd **<** 0**)** PFATAL**(**"Unable to create '%s'"**,** fn**);**

ck\_write**(**fd**,** mem**,** len**,** fn**);**

close**(**fd**);**

ck\_free**(**fn**);**

**return** keeping**;**

**}**

### 22.1.1 /describe\_op 标记文件名

/\* Construct a file name for a new test case, capturing the operation

that led to its discovery. Uses a static buffer. \*/

static u8**\*** describe\_op**(**u8 hnb**)** **{**

static u8 ret**[**256**];**

**if** **(**syncing\_party**)** **{**

sprintf**(**ret**,** "sync:%s,src:%06u"**,** syncing\_party**,** syncing\_case**);**

**}** **else** **{**

sprintf**(**ret**,** "src:%06u"**,** current\_entry**);//由哪个entry变异产生**

**if** **(**splicing\_with **>=** 0**)**

sprintf**(**ret **+** strlen**(**ret**),** "+%06u"**,** splicing\_with**);**

sprintf**(**ret **+** strlen**(**ret**),** ",op:%s"**,** stage\_short**);//当前的变异阶段**



**if** **(**stage\_cur\_byte **>=** 0**)** **{**

sprintf**(**ret **+** strlen**(**ret**),** ",pos:%u"**,** stage\_cur\_byte**);//变异的位置**

**if** **(**stage\_val\_type **!=** STAGE\_VAL\_NONE**)**

sprintf**(**ret **+** strlen**(**ret**),** ",val:%s%+d"**,**

**(**stage\_val\_type **==** STAGE\_VAL\_BE**)** **?** "be:" **:** ""**,**

stage\_cur\_val**);**

**}** **else** sprintf**(**ret **+** strlen**(**ret**),** ",rep:%u"**,** stage\_cur\_val**);**

**}**

**if** **(**hnb **==** 2**)** strcat**(**ret**,** ",+cov"**);//**2 if there are new tuples seen

**return** ret**;**

**}**

# /UR 产生随机数

/\* Generate a random number (from 0 to limit - 1). This may

have slight bias. \*/

static inline u32 UR**(**u32 limit**)** **{**

**if** **(**unlikely**(!**rand\_cnt**--))** **{**

u32 seed**[**2**];**

ck\_read**(**dev\_urandom\_fd**,** **&**seed**,** **sizeof(**seed**),** "/dev/urandom"**);**

srandom**(**seed**[**0**]);**

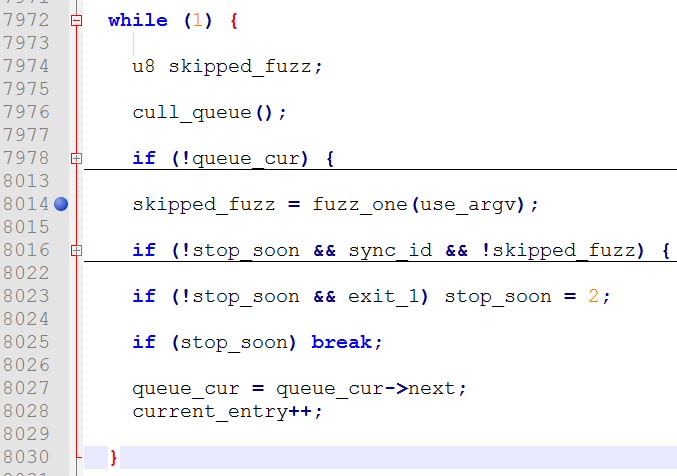
rand\_cnt **=** **(**RESEED\_RNG **/** 2**)** **+** **(**seed**[**1**]** **%** RESEED\_RNG**);**

**}**

**return** random**()** **%** limit**;**

**}**

# fuzz loop



**while** **(**1**)** **{**

u8 skipped\_fuzz**;**

cull\_queue**();//找到queue中最合适的，赋值给top\_rated，并且设置q->favored**

**if** **(!**queue\_cur**)** **{//初始值为0**

queue\_cycle**++;//轮数加一**

current\_entry **=** 0**;**

cur\_skipped\_paths **=** 0**;**

queue\_cur **=** queue**;//指向当前的q**

**while** **(**seek\_to**)** **{//如果不是resume，seek\_to为0**

current\_entry**++;**

seek\_to**--;**

queue\_cur **=** queue\_cur**->**next**;**

**}**

show\_stats**();**

**if** **(**not\_on\_tty**)** **{**

ACTF**(**"Entering queue cycle %llu."**,** queue\_cycle**);**

fflush**(**stdout**);**

**}**

/\* If we had a full queue cycle with no new finds, try

recombination strategies next. \*/

**if** **(**queued\_paths **==** prev\_queued**)** **{//注意？？？**

**if** **(**use\_splicing**)** cycles\_wo\_finds**++;** **else** use\_splicing **=** 1**;**

**}** **else** cycles\_wo\_finds **=** 0**;**

prev\_queued **=** queued\_paths**;//记录本轮的queue队列大小**

**if** **(**sync\_id **&&** queue\_cycle **==** 1 **&&** getenv**(**"AFL\_IMPORT\_FIRST"**))**

sync\_fuzzers**(**use\_argv**);**

**}**

skipped\_fuzz **=** fuzz\_one**(**use\_argv**);**

**if** **(!**stop\_soon **&&** sync\_id **&&** **!**skipped\_fuzz**)** **{**

**if** **(!(**sync\_interval\_cnt**++** **%** SYNC\_INTERVAL**))**

sync\_fuzzers**(**use\_argv**);**

**}**

**if** **(!**stop\_soon **&&** exit\_1**)** stop\_soon **=** 2**;**

**if** **(**stop\_soon**)** **break;**

queue\_cur **=** queue\_cur**->**next**;**

current\_entry**++;**

**}**

## 24.1 /cull\_queue()设置q->favored

注意结合前面的/update\_bitmap\_score，找到top\_rated**[**i**]**

/\* The second part of the mechanism discussed above is a routine that

goes over top\_rated[] entries, and then sequentially grabs winners for

previously-unseen bytes (temp\_v) and marks them as favored, at least

until the next run. The favored entries are given more air time during

all fuzzing steps. \*/

static void cull\_queue**(**void**)** **{**

struct queue\_entry**\*** q**;**

static u8 temp\_v**[**MAP\_SIZE **>>** 3**];**

u32 i**;**

**if** **(**dumb\_mode **||** **!**score\_changed**)** **return;**

score\_changed **=** 0**;**

memset**(**temp\_v**,** 255**,** MAP\_SIZE **>>** 3**);//初始化0xff**

queued\_favored **=** 0**;**

pending\_favored **=** 0**;**

q **=** queue**;**

**while** **(**q**)** **{**

q**->**favored **=** 0**;//事先清空标志位**

q **=** q**->**next**;**

**}**

/\* Let's see if anything in the bitmap isn't captured in temp\_v.

If yes, and if it has a top\_rated[] contender, let's use it. \*/

**for** **(**i **=** 0**;** i **<** MAP\_SIZE**;** i**++)**

**if** **(**top\_rated**[**i**]** **&&** **(**temp\_v**[**i **>>** 3**]** **&** **(**1 **<<** **(**i **&** 7**))))** **{**

u32 j **=** MAP\_SIZE **>>** 3**;**

/\* Remove all bits belonging to the current entry from temp\_v. \*/

**while** **(**j**--)**

**if** **(**top\_rated**[**i**]->**trace\_mini**[**j**])**

temp\_v**[**j**]** **&=** **~**top\_rated**[**i**]->**trace\_mini**[**j**];//清除该q覆盖到的tuple**

top\_rated**[**i**]->**favored **=** 1**;//将找到的q->favored 赋值为1**

queued\_favored**++;**

**if** **(!**top\_rated**[**i**]->**was\_fuzzed**)** pending\_favored**++;**

**}**

q **=** queue**;**

**while** **(**q**)** **{**

mark\_as\_redundant**(**q**,** **!**q**->**favored**);**

q **=** q**->**next**;**

**}**

**}**

## 24.2 /fuzz\_one

/\* Take the current entry from the queue, fuzz it for a while. This

function is a tad too long... returns 0 if fuzzed successfully, 1 if

skipped or bailed out. \*/

static u8 fuzz\_one**(**char**\*\*** argv**)** **{**

s32 len**,** fd**,** temp\_len**,** i**,** j**;**

u8 **\***in\_buf**,** **\***out\_buf**,** **\***orig\_in**,** **\***ex\_tmp**,** **\***eff\_map **=** 0**;**

u64 havoc\_queued**,** orig\_hit\_cnt**,** new\_hit\_cnt**;**

u32 splice\_cycle **=** 0**,** perf\_score **=** 100**,** orig\_perf**,** prev\_cksum**,** eff\_cnt **=** 1**;**

u8 ret\_val **=** 1**,** doing\_det **=** 0**;**

u8 a\_collect**[**MAX\_AUTO\_EXTRA**];**

u32 a\_len **=** 0**;**

#ifdef IGNORE\_FINDS

/\* In IGNORE\_FINDS mode, skip any entries that weren't in the

initial data set. \*/

**if** **(**queue\_cur**->**depth **>** 1**)** **return** 1**;**

### 20.1.1 queue\_cur->depth

初始值为1

/\* Append new test case to the queue. \*/

static void add\_to\_queue**(**u8**\*** fname**,** u32 len**,** u8 passed\_det**)** **{**

struct queue\_entry**\*** q **=** ck\_alloc**(sizeof(**struct queue\_entry**));**

q**->**fname **=** fname**;**

q**->**len **=** len**;**

q**->**depth **=** cur\_depth **+** 1**;//初始化赋值**

**在fuzz\_one中，后面有：**

cur\_depth **=** queue\_cur**->**depth**;**

queue\_cur赋值

queue\_cur **=** queue**; 初始值**

queue\_cur **=** queue\_cur**->**next**; 遍历队列**

///////////////////////////////

#else

**if** **(**pending\_favored**)** **{**

### 20.1.2 pending\_favored, /\* Pending favored paths

由cull\_queue来负责更新

static void cull\_queue**(**void**)** **{**

struct queue\_entry**\*** q**;**

static u8 temp\_v**[**MAP\_SIZE **>>** 3**];**

u32 i**;**

**if** **(**dumb\_mode **||** **!**score\_changed**)** **return;**

score\_changed **=** 0**;**

memset**(**temp\_v**,** 255**,** MAP\_SIZE **>>** 3**);**

queued\_favored **=** 0**;**

pending\_favored **=** 0**;**

q **=** queue**;**

**while** **(**q**)** **{**

q**->**favored **=** 0**;**

q **=** q**->**next**;**

**}**

/\* Let's see if anything in the bitmap isn't captured in temp\_v.

If yes, and if it has a top\_rated[] contender, let's use it. \*/

**for** **(**i **=** 0**;** i **<** MAP\_SIZE**;** i**++)**

**if** **(**top\_rated**[**i**]** **&&** **(**temp\_v**[**i **>>** 3**]** **&** **(**1 **<<** **(**i **&** 7**))))** **{**

u32 j **=** MAP\_SIZE **>>** 3**;**

/\* Remove all bits belonging to the current entry from temp\_v. \*/

**while** **(**j**--)**

**if** **(**top\_rated**[**i**]->**trace\_mini**[**j**])**

temp\_v**[**j**]** **&=** **~**top\_rated**[**i**]->**trace\_mini**[**j**];**

top\_rated**[**i**]->**favored **=** 1**;**

queued\_favored**++;**

**if** **(!**top\_rated**[**i**]->**was\_fuzzed**)** pending\_favored**++;**

**}**

////////////////////////////

/\* If we have any favored, non-fuzzed new arrivals in the queue,

possibly skip to them at the expense of already-fuzzed or non-favored

cases. \*/

**if** **((**queue\_cur**->**was\_fuzzed **||** **!**queue\_cur**->**favored**)** **&&**

UR**(**100**)** **<** SKIP\_TO\_NEW\_PROB**)** **return** 1**;//如果已经fuzz过，或者favored为0，则跳过该样本的变异**

**}** **else** **if** **(!**dumb\_mode **&&** **!**queue\_cur**->**favored **&&** queued\_paths **>** 10**)** **{**

/\* Otherwise, still possibly skip non-favored cases, albeit less often.

The odds of skipping stuff are higher for already-fuzzed inputs and

lower for never-fuzzed entries. \*/

**if** **(**queue\_cycle **>** 1 **&&** **!**queue\_cur**->**was\_fuzzed**)** **{**

**if** **(**UR**(**100**)** **<** SKIP\_NFAV\_NEW\_PROB**)** **return** 1**;**

**}** **else** **{**

**if** **(**UR**(**100**)** **<** SKIP\_NFAV\_OLD\_PROB**)** **return** 1**;**

**}**

**}**

#endif /\* ^IGNORE\_FINDS \*/

**if** **(**not\_on\_tty**)** **{**

ACTF**(**"Fuzzing test case #%u (%u total, %llu uniq crashes found)..."**,**

current\_entry**,** queued\_paths**,** unique\_crashes**);**

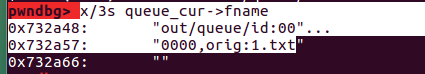
fflush**(**stdout**);**

**}**

/\* Map the test case into memory. \*/

fd **=** open**(**queue\_cur**->**fname**,** O\_RDONLY**);**

### 20.1.3 打开样本文件，映射内存



//////////

**if** **(**fd **<** 0**)** PFATAL**(**"Unable to open '%s'"**,** queue\_cur**->**fname**);**

orig\_in **=** in\_buf **=** mmap**(**0**,** len**,** PROT\_READ **|** PROT\_WRITE**,** MAP\_PRIVATE**,** fd**,** 0**);**

**if** **(**orig\_in **==** MAP\_FAILED**)** PFATAL**(**"Unable to mmap '%s'"**,** queue\_cur**->**fname**);**

close**(**fd**);**

/\* We could mmap() out\_buf as MAP\_PRIVATE, but we end up clobbering every

single byte anyway, so it wouldn't give us any performance or memory usage

benefits. \*/

out\_buf **=** ck\_alloc\_nozero**(**len**);**

subseq\_tmouts **=** 0**;**

cur\_depth **=** queue\_cur**->**depth**;**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* CALIBRATION (only if failed earlier on) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**if** **(**queue\_cur**->**cal\_failed**)** **{**

u8 res **=** FAULT\_TMOUT**;**

**if** **(**queue\_cur**->**cal\_failed **<** CAL\_CHANCES**)** **{**

res **=** calibrate\_case**(**argv**,** queue\_cur**,** in\_buf**,** queue\_cycle **-** 1**,** 0**);**

**if** **(**res **==** FAULT\_ERROR**)**

FATAL**(**"Unable to execute target application"**);**

**}**

**if** **(**stop\_soon **||** res **!=** crash\_mode**)** **{**

cur\_skipped\_paths**++;**

**goto** abandon\_entry**;**

**}**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*

\* TRIMMING \*

\*\*\*\*\*\*\*\*\*\*\*\*/

**if** **(!**dumb\_mode **&&** **!**queue\_cur**->**trim\_done**)** **{**

u8 res **=** trim\_case**(**argv**,** queue\_cur**,** in\_buf**);**

**if** **(**res **==** FAULT\_ERROR**)**

FATAL**(**"Unable to execute target application"**);**

**if** **(**stop\_soon**)** **{**

cur\_skipped\_paths**++;**

**goto** abandon\_entry**;**

**}**

/\* Don't retry trimming, even if it failed. \*/

queue\_cur**->**trim\_done **=** 1**;**

**if** **(**len **!=** queue\_cur**->**len**)** len **=** queue\_cur**->**len**;**

**}**

### 20.1.4 将输入文件缓冲区拷贝到输出缓冲区

memcpy**(**out\_buf**,** in\_buf**,** len**);**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* PERFORMANCE SCORE \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

### 20.1.5 记录之前的分数calculate\_score

orig\_perf **=** perf\_score **=** calculate\_score**(**queue\_cur**);**

**//////////////////////////////////////**

/\* Skip right away if -d is given, if we have done deterministic fuzzing on

this entry ourselves (was\_fuzzed), or if it has gone through deterministic

testing in earlier, resumed runs (passed\_det). \*/

**if** **(**skip\_deterministic **||** queue\_cur**->**was\_fuzzed **||** queue\_cur**->**passed\_det**)**

**goto** havoc\_stage**;**

/\* Skip deterministic fuzzing if exec path checksum puts this out of scope

for this master instance. \*/

**if** **(**master\_max **&&** **(**queue\_cur**->**exec\_cksum **%** master\_max**)** **!=** master\_id **-** 1**)**

**goto** havoc\_stage**;**

doing\_det **=** 1**;**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* SIMPLE BITFLIP (+dictionary construction) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define FLIP\_BIT(\_ar, \_b) do { \

u8\* \_arf = (u8\*)(\_ar); \

u32 \_bf = (\_b); \

\_arf[(\_bf) >> 3] ^= (128 >> ((\_bf) & 7)); \

} while (0)

### 20.1.6 flip1

/\* Single walking bit. \*/

stage\_short **=** "flip1"**;**

stage\_max **=** len **<<** 3**;**

stage\_name **=** "bitflip 1/1"**;**

stage\_val\_type **=** STAGE\_VAL\_NONE**;**

orig\_hit\_cnt **=** queued\_paths **+** unique\_crashes**;**

prev\_cksum **=** queue\_cur**->**exec\_cksum**;**

**for** **(**stage\_cur **=** 0**;** stage\_cur **<** stage\_max**;** stage\_cur**++)** **{//每一个bit翻转**

stage\_cur\_byte **=** stage\_cur **>>** 3**;**

FLIP\_BIT**(**out\_buf**,** stage\_cur**);//变异**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;//测试变异样本**

FLIP\_BIT**(**out\_buf**,** stage\_cur**);//恢复**

/\* While flipping the least significant bit in every byte, pull of an extra

trick to detect possible syntax tokens. In essence, the idea is that if

you have a binary blob like this:

xxxxxxxxIHDRxxxxxxxx

...and changing the leading and trailing bytes causes variable or no

changes in program flow, but touching any character in the "IHDR" string

always produces the same, distinctive path, it's highly likely that

"IHDR" is an atomically-checked magic value of special significance to

the fuzzed format.

We do this here, rather than as a separate stage, because it's a nice

way to keep the operation approximately "free" (i.e., no extra execs).

Empirically, performing the check when flipping the least significant bit

is advantageous, compared to doing it at the time of more disruptive

changes, where the program flow may be affected in more violent ways.

The caveat is that we won't generate dictionaries in the -d mode or -S

mode - but that's probably a fair trade-off.

This won't work particularly well with paths that exhibit variable

behavior, but fails gracefully, so we'll carry out the checks anyway.

\*/

**if** **(!**dumb\_mode **&&** **(**stage\_cur **&** 7**)** **==** 7**)** **{**

u32 cksum **=** hash32**(**trace\_bits**,** MAP\_SIZE**,** HASH\_CONST**);**

**if** **(**stage\_cur **==** stage\_max **-** 1 **&&** cksum **==** prev\_cksum**)** **{**

/\* If at end of file and we are still collecting a string, grab the

final character and force output. \*/

**if** **(**a\_len **<** MAX\_AUTO\_EXTRA**)** a\_collect**[**a\_len**]** **=** out\_buf**[**stage\_cur **>>** 3**];**

a\_len**++;**

**if** **(**a\_len **>=** MIN\_AUTO\_EXTRA **&&** a\_len **<=** MAX\_AUTO\_EXTRA**)**

maybe\_add\_auto**(**a\_collect**,** a\_len**);**

**}** **else** **if** **(**cksum **!=** prev\_cksum**)** **{**

/\* Otherwise, if the checksum has changed, see if we have something

worthwhile queued up, and collect that if the answer is yes. \*/

**if** **(**a\_len **>=** MIN\_AUTO\_EXTRA **&&** a\_len **<=** MAX\_AUTO\_EXTRA**)**

maybe\_add\_auto**(**a\_collect**,** a\_len**);**

a\_len **=** 0**;**

prev\_cksum **=** cksum**;**

**}**

/\* Continue collecting string, but only if the bit flip actually made

any difference - we don't want no-op tokens. \*/

**if** **(**cksum **!=** queue\_cur**->**exec\_cksum**)** **{**

**if** **(**a\_len **<** MAX\_AUTO\_EXTRA**)** a\_collect**[**a\_len**]** **=** out\_buf**[**stage\_cur **>>** 3**];**

a\_len**++;**

**}**

**}**

**}**

new\_hit\_cnt **=** queued\_paths **+** unique\_crashes**;**

stage\_finds**[**STAGE\_FLIP1**]** **+=** new\_hit\_cnt **-** orig\_hit\_cnt**;**

stage\_cycles**[**STAGE\_FLIP1**]** **+=** stage\_max**;**

### 20.1.7 bitflip 2/1

/\* Two walking bits. \*/

stage\_name **=** "bitflip 2/1"**;**

stage\_short **=** "flip2"**;**

stage\_max **=** **(**len **<<** 3**)** **-** 1**;**

orig\_hit\_cnt **=** new\_hit\_cnt**;**

**for** **(**stage\_cur **=** 0**;** stage\_cur **<** stage\_max**;** stage\_cur**++)** **{**

stage\_cur\_byte **=** stage\_cur **>>** 3**;**

FLIP\_BIT**(**out\_buf**,** stage\_cur**);**

FLIP\_BIT**(**out\_buf**,** stage\_cur **+** 1**);**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

FLIP\_BIT**(**out\_buf**,** stage\_cur**);**

FLIP\_BIT**(**out\_buf**,** stage\_cur **+** 1**);**

**}**

new\_hit\_cnt **=** queued\_paths **+** unique\_crashes**;**

stage\_finds**[**STAGE\_FLIP2**]** **+=** new\_hit\_cnt **-** orig\_hit\_cnt**;**

stage\_cycles**[**STAGE\_FLIP2**]** **+=** stage\_max**;**

### 20.1.8 bitflip 4/1

/\* Four walking bits. \*/

stage\_name **=** "bitflip 4/1"**;**

stage\_short **=** "flip4"**;**

stage\_max **=** **(**len **<<** 3**)** **-** 3**;**

orig\_hit\_cnt **=** new\_hit\_cnt**;**

**for** **(**stage\_cur **=** 0**;** stage\_cur **<** stage\_max**;** stage\_cur**++)** **{**

stage\_cur\_byte **=** stage\_cur **>>** 3**;**

FLIP\_BIT**(**out\_buf**,** stage\_cur**);**

FLIP\_BIT**(**out\_buf**,** stage\_cur **+** 1**);**

FLIP\_BIT**(**out\_buf**,** stage\_cur **+** 2**);**

FLIP\_BIT**(**out\_buf**,** stage\_cur **+** 3**);**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

FLIP\_BIT**(**out\_buf**,** stage\_cur**);**

FLIP\_BIT**(**out\_buf**,** stage\_cur **+** 1**);**

FLIP\_BIT**(**out\_buf**,** stage\_cur **+** 2**);**

FLIP\_BIT**(**out\_buf**,** stage\_cur **+** 3**);**

**}**

new\_hit\_cnt **=** queued\_paths **+** unique\_crashes**;**

stage\_finds**[**STAGE\_FLIP4**]** **+=** new\_hit\_cnt **-** orig\_hit\_cnt**;**

stage\_cycles**[**STAGE\_FLIP4**]** **+=** stage\_max**;**

/\* Effector map setup. These macros calculate:

EFF\_APOS - position of a particular file offset in the map.

EFF\_ALEN - length of a map with a particular number of bytes.

EFF\_SPAN\_ALEN - map span for a sequence of bytes.

\*/

#define EFF\_APOS(\_p) ((\_p) >> EFF\_MAP\_SCALE2)

#define EFF\_REM(\_x) ((\_x) & ((1 << EFF\_MAP\_SCALE2) - 1))

#define EFF\_ALEN(\_l) (EFF\_APOS(\_l) + !!EFF\_REM(\_l))

#define EFF\_SPAN\_ALEN(\_p, \_l) (EFF\_APOS((\_p) + (\_l) - 1) - EFF\_APOS(\_p) + 1)

/\* Initialize effector map for the next step (see comments below). Always

flag first and last byte as doing something. \*/

eff\_map **=** ck\_alloc**(**EFF\_ALEN**(**len**));**

eff\_map**[**0**]** **=** 1**;**

**if** **(**EFF\_APOS**(**len **-** 1**)** **!=** 0**)** **{**

eff\_map**[**EFF\_APOS**(**len **-** 1**)]** **=** 1**;**

eff\_cnt**++;**

**}**

### 20.1.9 bitflip 8/8

/\* Walking byte. \*/

stage\_name **=** "bitflip 8/8"**;**

stage\_short **=** "flip8"**;**

stage\_max **=** len**;**

orig\_hit\_cnt **=** new\_hit\_cnt**;**

**for** **(**stage\_cur **=** 0**;** stage\_cur **<** stage\_max**;** stage\_cur**++)** **{**

stage\_cur\_byte **=** stage\_cur**;**

out\_buf**[**stage\_cur**]** **^=** 0xFF**;**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

/\* We also use this stage to pull off a simple trick: we identify

bytes that seem to have no effect on the current execution path

even when fully flipped - and we skip them during more expensive

deterministic stages, such as arithmetics or known ints. \*/

**if** **(!**eff\_map**[**EFF\_APOS**(**stage\_cur**)])** **{**

u32 cksum**;**

/\* If in dumb mode or if the file is very short, just flag everything

without wasting time on checksums. \*/

**if** **(!**dumb\_mode **&&** len **>=** EFF\_MIN\_LEN**)**

cksum **=** hash32**(**trace\_bits**,** MAP\_SIZE**,** HASH\_CONST**);**

**else**

cksum **=** **~**queue\_cur**->**exec\_cksum**;**

**if** **(**cksum **!=** queue\_cur**->**exec\_cksum**)** **{**

eff\_map**[**EFF\_APOS**(**stage\_cur**)]** **=** 1**;**

eff\_cnt**++;**

**}**

**}**

out\_buf**[**stage\_cur**]** **^=** 0xFF**;**

**}**

/\* If the effector map is more than EFF\_MAX\_PERC dense, just flag the

whole thing as worth fuzzing, since we wouldn't be saving much time

anyway. \*/

**if** **(**eff\_cnt **!=** EFF\_ALEN**(**len**)** **&&**

eff\_cnt **\*** 100 **/** EFF\_ALEN**(**len**)** **>** EFF\_MAX\_PERC**)** **{**

memset**(**eff\_map**,** 1**,** EFF\_ALEN**(**len**));**

blocks\_eff\_select **+=** EFF\_ALEN**(**len**);**

**}** **else** **{**

blocks\_eff\_select **+=** eff\_cnt**;**

**}**

blocks\_eff\_total **+=** EFF\_ALEN**(**len**);**

new\_hit\_cnt **=** queued\_paths **+** unique\_crashes**;**

stage\_finds**[**STAGE\_FLIP8**]** **+=** new\_hit\_cnt **-** orig\_hit\_cnt**;**

stage\_cycles**[**STAGE\_FLIP8**]** **+=** stage\_max**;**

### 20.1.10 bitflip 16/8

/\* Two walking bytes. \*/

**if** **(**len **<** 2**)** **goto** skip\_bitflip**;**

stage\_name **=** "bitflip 16/8"**;**

stage\_short **=** "flip16"**;**

stage\_cur **=** 0**;**

stage\_max **=** len **-** 1**;**

orig\_hit\_cnt **=** new\_hit\_cnt**;**

**for** **(**i **=** 0**;** i **<** len **-** 1**;** i**++)** **{**

/\* Let's consult the effector map... \*/

**if** **(!**eff\_map**[**EFF\_APOS**(**i**)]** **&&** **!**eff\_map**[**EFF\_APOS**(**i **+** 1**)])** **{**

stage\_max**--;**

**continue;**

**}**

stage\_cur\_byte **=** i**;**

**\*(**u16**\*)(**out\_buf **+** i**)** **^=** 0xFFFF**;**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

stage\_cur**++;**

**\*(**u16**\*)(**out\_buf **+** i**)** **^=** 0xFFFF**;**

**}**

new\_hit\_cnt **=** queued\_paths **+** unique\_crashes**;**

stage\_finds**[**STAGE\_FLIP16**]** **+=** new\_hit\_cnt **-** orig\_hit\_cnt**;**

stage\_cycles**[**STAGE\_FLIP16**]** **+=** stage\_max**;**

**if** **(**len **<** 4**)** **goto** skip\_bitflip**;**

### 20.1.11 bitflip 32/8

/\* Four walking bytes. \*/

stage\_name **=** "bitflip 32/8"**;**

stage\_short **=** "flip32"**;**

stage\_cur **=** 0**;**

stage\_max **=** len **-** 3**;**

orig\_hit\_cnt **=** new\_hit\_cnt**;**

**for** **(**i **=** 0**;** i **<** len **-** 3**;** i**++)** **{**

/\* Let's consult the effector map... \*/

**if** **(!**eff\_map**[**EFF\_APOS**(**i**)]** **&&** **!**eff\_map**[**EFF\_APOS**(**i **+** 1**)]** **&&**

**!**eff\_map**[**EFF\_APOS**(**i **+** 2**)]** **&&** **!**eff\_map**[**EFF\_APOS**(**i **+** 3**)])** **{**

stage\_max**--;**

**continue;**

**}**

stage\_cur\_byte **=** i**;**

**\*(**u32**\*)(**out\_buf **+** i**)** **^=** 0xFFFFFFFF**;**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

stage\_cur**++;**

**\*(**u32**\*)(**out\_buf **+** i**)** **^=** 0xFFFFFFFF**;**

**}**

new\_hit\_cnt **=** queued\_paths **+** unique\_crashes**;**

stage\_finds**[**STAGE\_FLIP32**]** **+=** new\_hit\_cnt **-** orig\_hit\_cnt**;**

stage\_cycles**[**STAGE\_FLIP32**]** **+=** stage\_max**;**

skip\_bitflip**:**

**if** **(**no\_arith**)** **goto** skip\_arith**;**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ARITHMETIC INC/DEC \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

### 20.1.12 arith 8/8

/\* 8-bit arithmetics. \*/

stage\_name **=** "arith 8/8"**;**

stage\_short **=** "arith8"**;**

stage\_cur **=** 0**;**

stage\_max **=** 2 **\*** len **\*** ARITH\_MAX**;**

stage\_val\_type **=** STAGE\_VAL\_LE**;**

orig\_hit\_cnt **=** new\_hit\_cnt**;**

**for** **(**i **=** 0**;** i **<** len**;** i**++)** **{**

u8 orig **=** out\_buf**[**i**];**

/\* Let's consult the effector map... \*/

**if** **(!**eff\_map**[**EFF\_APOS**(**i**)])** **{**

stage\_max **-=** 2 **\*** ARITH\_MAX**;**

**continue;**

**}**

stage\_cur\_byte **=** i**;**

**for** **(**j **=** 1**;** j **<=** ARITH\_MAX**;** j**++)** **{**

u8 r **=** orig **^** **(**orig **+** j**);**

/\* Do arithmetic operations only if the result couldn't be a product

of a bitflip. \*/

**if** **(!**could\_be\_bitflip**(**r**))** **{**

stage\_cur\_val **=** j**;**

out\_buf**[**i**]** **=** orig **+** j**;**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

stage\_cur**++;**

**}** **else** stage\_max**--;**

r **=** orig **^** **(**orig **-** j**);**

**if** **(!**could\_be\_bitflip**(**r**))** **{**

stage\_cur\_val **=** **-**j**;**

out\_buf**[**i**]** **=** orig **-** j**;**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

stage\_cur**++;**

**}** **else** stage\_max**--;**

out\_buf**[**i**]** **=** orig**;**

**}**

**}**

new\_hit\_cnt **=** queued\_paths **+** unique\_crashes**;**

stage\_finds**[**STAGE\_ARITH8**]** **+=** new\_hit\_cnt **-** orig\_hit\_cnt**;**

stage\_cycles**[**STAGE\_ARITH8**]** **+=** stage\_max**;**

/\* 16-bit arithmetics, both endians. \*/

**if** **(**len **<** 2**)** **goto** skip\_arith**;**

### 20.1.13 arith 16/8

stage\_name **=** "arith 16/8"**;**

stage\_short **=** "arith16"**;**

stage\_cur **=** 0**;**

stage\_max **=** 4 **\*** **(**len **-** 1**)** **\*** ARITH\_MAX**;**

orig\_hit\_cnt **=** new\_hit\_cnt**;**

**for** **(**i **=** 0**;** i **<** len **-** 1**;** i**++)** **{**

u16 orig **=** **\*(**u16**\*)(**out\_buf **+** i**);**

/\* Let's consult the effector map... \*/

**if** **(!**eff\_map**[**EFF\_APOS**(**i**)]** **&&** **!**eff\_map**[**EFF\_APOS**(**i **+** 1**)])** **{**

stage\_max **-=** 4 **\*** ARITH\_MAX**;**

**continue;**

**}**

stage\_cur\_byte **=** i**;**

**for** **(**j **=** 1**;** j **<=** ARITH\_MAX**;** j**++)** **{**

u16 r1 **=** orig **^** **(**orig **+** j**),**

r2 **=** orig **^** **(**orig **-** j**),**

r3 **=** orig **^** SWAP16**(**SWAP16**(**orig**)** **+** j**),**

r4 **=** orig **^** SWAP16**(**SWAP16**(**orig**)** **-** j**);**

/\* Try little endian addition and subtraction first. Do it only

if the operation would affect more than one byte (hence the

& 0xff overflow checks) and if it couldn't be a product of

a bitflip. \*/

stage\_val\_type **=** STAGE\_VAL\_LE**;**

**if** **((**orig **&** 0xff**)** **+** j **>** 0xff **&&** **!**could\_be\_bitflip**(**r1**))** **{**

stage\_cur\_val **=** j**;**

**\*(**u16**\*)(**out\_buf **+** i**)** **=** orig **+** j**;**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

stage\_cur**++;**

**}** **else** stage\_max**--;**

**if** **((**orig **&** 0xff**)** **<** j **&&** **!**could\_be\_bitflip**(**r2**))** **{**

stage\_cur\_val **=** **-**j**;**

**\*(**u16**\*)(**out\_buf **+** i**)** **=** orig **-** j**;**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

stage\_cur**++;**

**}** **else** stage\_max**--;**

/\* Big endian comes next. Same deal. \*/

stage\_val\_type **=** STAGE\_VAL\_BE**;**

**if** **((**orig **>>** 8**)** **+** j **>** 0xff **&&** **!**could\_be\_bitflip**(**r3**))** **{**

stage\_cur\_val **=** j**;**

**\*(**u16**\*)(**out\_buf **+** i**)** **=** SWAP16**(**SWAP16**(**orig**)** **+** j**);**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

stage\_cur**++;**

**}** **else** stage\_max**--;**

**if** **((**orig **>>** 8**)** **<** j **&&** **!**could\_be\_bitflip**(**r4**))** **{**

stage\_cur\_val **=** **-**j**;**

**\*(**u16**\*)(**out\_buf **+** i**)** **=** SWAP16**(**SWAP16**(**orig**)** **-** j**);**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

stage\_cur**++;**

**}** **else** stage\_max**--;**

**\*(**u16**\*)(**out\_buf **+** i**)** **=** orig**;**

**}**

**}**

new\_hit\_cnt **=** queued\_paths **+** unique\_crashes**;**

stage\_finds**[**STAGE\_ARITH16**]** **+=** new\_hit\_cnt **-** orig\_hit\_cnt**;**

stage\_cycles**[**STAGE\_ARITH16**]** **+=** stage\_max**;**

/\* 32-bit arithmetics, both endians. \*/

### 20.1.14 arith 32/8

**if** **(**len **<** 4**)** **goto** skip\_arith**;**

stage\_name **=** "arith 32/8"**;**

stage\_short **=** "arith32"**;**

stage\_cur **=** 0**;**

stage\_max **=** 4 **\*** **(**len **-** 3**)** **\*** ARITH\_MAX**;**

orig\_hit\_cnt **=** new\_hit\_cnt**;**

**for** **(**i **=** 0**;** i **<** len **-** 3**;** i**++)** **{**

u32 orig **=** **\*(**u32**\*)(**out\_buf **+** i**);**

/\* Let's consult the effector map... \*/

**if** **(!**eff\_map**[**EFF\_APOS**(**i**)]** **&&** **!**eff\_map**[**EFF\_APOS**(**i **+** 1**)]** **&&**

**!**eff\_map**[**EFF\_APOS**(**i **+** 2**)]** **&&** **!**eff\_map**[**EFF\_APOS**(**i **+** 3**)])** **{**

stage\_max **-=** 4 **\*** ARITH\_MAX**;**

**continue;**

**}**

stage\_cur\_byte **=** i**;**

**for** **(**j **=** 1**;** j **<=** ARITH\_MAX**;** j**++)** **{**

u32 r1 **=** orig **^** **(**orig **+** j**),**

r2 **=** orig **^** **(**orig **-** j**),**

r3 **=** orig **^** SWAP32**(**SWAP32**(**orig**)** **+** j**),**

r4 **=** orig **^** SWAP32**(**SWAP32**(**orig**)** **-** j**);**

/\* Little endian first. Same deal as with 16-bit: we only want to

try if the operation would have effect on more than two bytes. \*/

stage\_val\_type **=** STAGE\_VAL\_LE**;**

**if** **((**orig **&** 0xffff**)** **+** j **>** 0xffff **&&** **!**could\_be\_bitflip**(**r1**))** **{**

stage\_cur\_val **=** j**;**

**\*(**u32**\*)(**out\_buf **+** i**)** **=** orig **+** j**;**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

stage\_cur**++;**

**}** **else** stage\_max**--;**

**if** **((**orig **&** 0xffff**)** **<** j **&&** **!**could\_be\_bitflip**(**r2**))** **{**

stage\_cur\_val **=** **-**j**;**

**\*(**u32**\*)(**out\_buf **+** i**)** **=** orig **-** j**;**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

stage\_cur**++;**

**}** **else** stage\_max**--;**

/\* Big endian next. \*/

stage\_val\_type **=** STAGE\_VAL\_BE**;**

**if** **((**SWAP32**(**orig**)** **&** 0xffff**)** **+** j **>** 0xffff **&&** **!**could\_be\_bitflip**(**r3**))** **{**

stage\_cur\_val **=** j**;**

**\*(**u32**\*)(**out\_buf **+** i**)** **=** SWAP32**(**SWAP32**(**orig**)** **+** j**);**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

stage\_cur**++;**

**}** **else** stage\_max**--;**

**if** **((**SWAP32**(**orig**)** **&** 0xffff**)** **<** j **&&** **!**could\_be\_bitflip**(**r4**))** **{**

stage\_cur\_val **=** **-**j**;**

**\*(**u32**\*)(**out\_buf **+** i**)** **=** SWAP32**(**SWAP32**(**orig**)** **-** j**);**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

stage\_cur**++;**

**}** **else** stage\_max**--;**

**\*(**u32**\*)(**out\_buf **+** i**)** **=** orig**;**

**}**

**}**

new\_hit\_cnt **=** queued\_paths **+** unique\_crashes**;**

stage\_finds**[**STAGE\_ARITH32**]** **+=** new\_hit\_cnt **-** orig\_hit\_cnt**;**

stage\_cycles**[**STAGE\_ARITH32**]** **+=** stage\_max**;**

skip\_arith**:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* INTERESTING VALUES \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

### 20.1.15 interest 8/8

stage\_name **=** "interest 8/8"**;**

stage\_short **=** "int8"**;**

stage\_cur **=** 0**;**

stage\_max **=** len **\*** **sizeof(**interesting\_8**);**

stage\_val\_type **=** STAGE\_VAL\_LE**;**

orig\_hit\_cnt **=** new\_hit\_cnt**;**

/\* Setting 8-bit integers. \*/

**for** **(**i **=** 0**;** i **<** len**;** i**++)** **{**

u8 orig **=** out\_buf**[**i**];**

/\* Let's consult the effector map... \*/

**if** **(!**eff\_map**[**EFF\_APOS**(**i**)])** **{**

stage\_max **-=** **sizeof(**interesting\_8**);**

**continue;**

**}**

stage\_cur\_byte **=** i**;**

**for** **(**j **=** 0**;** j **<** **sizeof(**interesting\_8**);** j**++)** **{**

/\* Skip if the value could be a product of bitflips or arithmetics. \*/

**if** **(**could\_be\_bitflip**(**orig **^** **(**u8**)**interesting\_8**[**j**])** **||**

could\_be\_arith**(**orig**,** **(**u8**)**interesting\_8**[**j**],** 1**))** **{**

stage\_max**--;**

**continue;**

**}**

stage\_cur\_val **=** interesting\_8**[**j**];**

out\_buf**[**i**]** **=** interesting\_8**[**j**];**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

out\_buf**[**i**]** **=** orig**;**

stage\_cur**++;**

**}**

**}**

new\_hit\_cnt **=** queued\_paths **+** unique\_crashes**;**

stage\_finds**[**STAGE\_INTEREST8**]** **+=** new\_hit\_cnt **-** orig\_hit\_cnt**;**

stage\_cycles**[**STAGE\_INTEREST8**]** **+=** stage\_max**;**

/\* Setting 16-bit integers, both endians. \*/

**if** **(**no\_arith **||** len **<** 2**)** **goto** skip\_interest**;**

### 20.1.16 interest 16/8

stage\_name **=** "interest 16/8"**;**

stage\_short **=** "int16"**;**

stage\_cur **=** 0**;**

stage\_max **=** 2 **\*** **(**len **-** 1**)** **\*** **(sizeof(**interesting\_16**)** **>>** 1**);**

orig\_hit\_cnt **=** new\_hit\_cnt**;**

**for** **(**i **=** 0**;** i **<** len **-** 1**;** i**++)** **{**

u16 orig **=** **\*(**u16**\*)(**out\_buf **+** i**);**

/\* Let's consult the effector map... \*/

**if** **(!**eff\_map**[**EFF\_APOS**(**i**)]** **&&** **!**eff\_map**[**EFF\_APOS**(**i **+** 1**)])** **{**

stage\_max **-=** **sizeof(**interesting\_16**);**

**continue;**

**}**

stage\_cur\_byte **=** i**;**

**for** **(**j **=** 0**;** j **<** **sizeof(**interesting\_16**)** **/** 2**;** j**++)** **{**

stage\_cur\_val **=** interesting\_16**[**j**];**

/\* Skip if this could be a product of a bitflip, arithmetics,

or single-byte interesting value insertion. \*/

**if** **(!**could\_be\_bitflip**(**orig **^** **(**u16**)**interesting\_16**[**j**])** **&&**

**!**could\_be\_arith**(**orig**,** **(**u16**)**interesting\_16**[**j**],** 2**)** **&&**

**!**could\_be\_interest**(**orig**,** **(**u16**)**interesting\_16**[**j**],** 2**,** 0**))** **{**

stage\_val\_type **=** STAGE\_VAL\_LE**;**

**\*(**u16**\*)(**out\_buf **+** i**)** **=** interesting\_16**[**j**];**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

stage\_cur**++;**

**}** **else** stage\_max**--;**

**if** **((**u16**)**interesting\_16**[**j**]** **!=** SWAP16**(**interesting\_16**[**j**])** **&&**

**!**could\_be\_bitflip**(**orig **^** SWAP16**(**interesting\_16**[**j**]))** **&&**

**!**could\_be\_arith**(**orig**,** SWAP16**(**interesting\_16**[**j**]),** 2**)** **&&**

**!**could\_be\_interest**(**orig**,** SWAP16**(**interesting\_16**[**j**]),** 2**,** 1**))** **{**

stage\_val\_type **=** STAGE\_VAL\_BE**;**

**\*(**u16**\*)(**out\_buf **+** i**)** **=** SWAP16**(**interesting\_16**[**j**]);**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

stage\_cur**++;**

**}** **else** stage\_max**--;**

**}**

**\*(**u16**\*)(**out\_buf **+** i**)** **=** orig**;**

**}**

new\_hit\_cnt **=** queued\_paths **+** unique\_crashes**;**

stage\_finds**[**STAGE\_INTEREST16**]** **+=** new\_hit\_cnt **-** orig\_hit\_cnt**;**

stage\_cycles**[**STAGE\_INTEREST16**]** **+=** stage\_max**;**

**if** **(**len **<** 4**)** **goto** skip\_interest**;**

/\* Setting 32-bit integers, both endians. \*/

### 20.1.17 interest 32/8

stage\_name **=** "interest 32/8"**;**

stage\_short **=** "int32"**;**

stage\_cur **=** 0**;**

stage\_max **=** 2 **\*** **(**len **-** 3**)** **\*** **(sizeof(**interesting\_32**)** **>>** 2**);**

orig\_hit\_cnt **=** new\_hit\_cnt**;**

**for** **(**i **=** 0**;** i **<** len **-** 3**;** i**++)** **{**

u32 orig **=** **\*(**u32**\*)(**out\_buf **+** i**);**

/\* Let's consult the effector map... \*/

**if** **(!**eff\_map**[**EFF\_APOS**(**i**)]** **&&** **!**eff\_map**[**EFF\_APOS**(**i **+** 1**)]** **&&**

**!**eff\_map**[**EFF\_APOS**(**i **+** 2**)]** **&&** **!**eff\_map**[**EFF\_APOS**(**i **+** 3**)])** **{**

stage\_max **-=** **sizeof(**interesting\_32**)** **>>** 1**;**

**continue;**

**}**

stage\_cur\_byte **=** i**;**

**for** **(**j **=** 0**;** j **<** **sizeof(**interesting\_32**)** **/** 4**;** j**++)** **{**

stage\_cur\_val **=** interesting\_32**[**j**];**

/\* Skip if this could be a product of a bitflip, arithmetics,

or word interesting value insertion. \*/

**if** **(!**could\_be\_bitflip**(**orig **^** **(**u32**)**interesting\_32**[**j**])** **&&**

**!**could\_be\_arith**(**orig**,** interesting\_32**[**j**],** 4**)** **&&**

**!**could\_be\_interest**(**orig**,** interesting\_32**[**j**],** 4**,** 0**))** **{**

stage\_val\_type **=** STAGE\_VAL\_LE**;**

**\*(**u32**\*)(**out\_buf **+** i**)** **=** interesting\_32**[**j**];**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

stage\_cur**++;**

**}** **else** stage\_max**--;**

**if** **((**u32**)**interesting\_32**[**j**]** **!=** SWAP32**(**interesting\_32**[**j**])** **&&**

**!**could\_be\_bitflip**(**orig **^** SWAP32**(**interesting\_32**[**j**]))** **&&**

**!**could\_be\_arith**(**orig**,** SWAP32**(**interesting\_32**[**j**]),** 4**)** **&&**

**!**could\_be\_interest**(**orig**,** SWAP32**(**interesting\_32**[**j**]),** 4**,** 1**))** **{**

stage\_val\_type **=** STAGE\_VAL\_BE**;**

**\*(**u32**\*)(**out\_buf **+** i**)** **=** SWAP32**(**interesting\_32**[**j**]);**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

stage\_cur**++;**

**}** **else** stage\_max**--;**

**}**

**\*(**u32**\*)(**out\_buf **+** i**)** **=** orig**;**

**}**

new\_hit\_cnt **=** queued\_paths **+** unique\_crashes**;**

stage\_finds**[**STAGE\_INTEREST32**]** **+=** new\_hit\_cnt **-** orig\_hit\_cnt**;**

stage\_cycles**[**STAGE\_INTEREST32**]** **+=** stage\_max**;**

skip\_interest**:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DICTIONARY STUFF \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**if** **(!**extras\_cnt**)** **goto** skip\_user\_extras**;**

/\* Overwrite with user-supplied extras. \*/

### 20.1.18 user extras (over)

stage\_name **=** "user extras (over)"**;**

stage\_short **=** "ext\_UO"**;**

stage\_cur **=** 0**;**

stage\_max **=** extras\_cnt **\*** len**;**

stage\_val\_type **=** STAGE\_VAL\_NONE**;**

orig\_hit\_cnt **=** new\_hit\_cnt**;**

**for** **(**i **=** 0**;** i **<** len**;** i**++)** **{**

u32 last\_len **=** 0**;**

stage\_cur\_byte **=** i**;**

/\* Extras are sorted by size, from smallest to largest. This means

that we don't have to worry about restoring the buffer in

between writes at a particular offset determined by the outer

loop. \*/

**for** **(**j **=** 0**;** j **<** extras\_cnt**;** j**++)** **{**

/\* Skip extras probabilistically if extras\_cnt > MAX\_DET\_EXTRAS. Also

skip them if there's no room to insert the payload, if the token

is redundant, or if its entire span has no bytes set in the effector

map. \*/

**if** **((**extras\_cnt **>** MAX\_DET\_EXTRAS **&&** UR**(**extras\_cnt**)** **>=** MAX\_DET\_EXTRAS**)** **||**

extras**[**j**].**len **>** len **-** i **||**

**!**memcmp**(**extras**[**j**].**data**,** out\_buf **+** i**,** extras**[**j**].**len**)** **||**

**!**memchr**(**eff\_map **+** EFF\_APOS**(**i**),** 1**,** EFF\_SPAN\_ALEN**(**i**,** extras**[**j**].**len**)))** **{**

stage\_max**--;**

**continue;**

**}**

last\_len **=** extras**[**j**].**len**;**

memcpy**(**out\_buf **+** i**,** extras**[**j**].**data**,** last\_len**);**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

stage\_cur**++;**

**}**

/\* Restore all the clobbered memory. \*/

memcpy**(**out\_buf **+** i**,** in\_buf **+** i**,** last\_len**);**

**}**

new\_hit\_cnt **=** queued\_paths **+** unique\_crashes**;**

stage\_finds**[**STAGE\_EXTRAS\_UO**]** **+=** new\_hit\_cnt **-** orig\_hit\_cnt**;**

stage\_cycles**[**STAGE\_EXTRAS\_UO**]** **+=** stage\_max**;**

/\* Insertion of user-supplied extras. \*/

### 20.1.19 user extras (insert)

stage\_name **=** "user extras (insert)"**;**

stage\_short **=** "ext\_UI"**;**

stage\_cur **=** 0**;**

stage\_max **=** extras\_cnt **\*** len**;**

orig\_hit\_cnt **=** new\_hit\_cnt**;**

ex\_tmp **=** ck\_alloc**(**len **+** MAX\_DICT\_FILE**);**

**for** **(**i **=** 0**;** i **<=** len**;** i**++)** **{**

stage\_cur\_byte **=** i**;**

**for** **(**j **=** 0**;** j **<** extras\_cnt**;** j**++)** **{**

**if** **(**len **+** extras**[**j**].**len **>** MAX\_FILE**)** **{**

stage\_max**--;**

**continue;**

**}**

/\* Insert token \*/

memcpy**(**ex\_tmp **+** i**,** extras**[**j**].**data**,** extras**[**j**].**len**);**

/\* Copy tail \*/

memcpy**(**ex\_tmp **+** i **+** extras**[**j**].**len**,** out\_buf **+** i**,** len **-** i**);**

**if** **(**common\_fuzz\_stuff**(**argv**,** ex\_tmp**,** len **+** extras**[**j**].**len**))** **{**

ck\_free**(**ex\_tmp**);**

**goto** abandon\_entry**;**

**}**

stage\_cur**++;**

**}**

/\* Copy head \*/

ex\_tmp**[**i**]** **=** out\_buf**[**i**];**

**}**

ck\_free**(**ex\_tmp**);**

new\_hit\_cnt **=** queued\_paths **+** unique\_crashes**;**

stage\_finds**[**STAGE\_EXTRAS\_UI**]** **+=** new\_hit\_cnt **-** orig\_hit\_cnt**;**

stage\_cycles**[**STAGE\_EXTRAS\_UI**]** **+=** stage\_max**;**

skip\_user\_extras**:**

**if** **(!**a\_extras\_cnt**)** **goto** skip\_extras**;**

### 20.1.20 auto extras (over)

stage\_name **=** "auto extras (over)"**;**

stage\_short **=** "ext\_AO"**;**

stage\_cur **=** 0**;**

stage\_max **=** MIN**(**a\_extras\_cnt**,** USE\_AUTO\_EXTRAS**)** **\*** len**;**

stage\_val\_type **=** STAGE\_VAL\_NONE**;**

orig\_hit\_cnt **=** new\_hit\_cnt**;**

**for** **(**i **=** 0**;** i **<** len**;** i**++)** **{**

u32 last\_len **=** 0**;**

stage\_cur\_byte **=** i**;**

**for** **(**j **=** 0**;** j **<** MIN**(**a\_extras\_cnt**,** USE\_AUTO\_EXTRAS**);** j**++)** **{**

/\* See the comment in the earlier code; extras are sorted by size. \*/

**if** **(**a\_extras**[**j**].**len **>** len **-** i **||**

**!**memcmp**(**a\_extras**[**j**].**data**,** out\_buf **+** i**,** a\_extras**[**j**].**len**)** **||**

**!**memchr**(**eff\_map **+** EFF\_APOS**(**i**),** 1**,** EFF\_SPAN\_ALEN**(**i**,** a\_extras**[**j**].**len**)))** **{**

stage\_max**--;**

**continue;**

**}**

last\_len **=** a\_extras**[**j**].**len**;**

memcpy**(**out\_buf **+** i**,** a\_extras**[**j**].**data**,** last\_len**);**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** len**))** **goto** abandon\_entry**;**

stage\_cur**++;**

**}**

/\* Restore all the clobbered memory. \*/

memcpy**(**out\_buf **+** i**,** in\_buf **+** i**,** last\_len**);**

**}**

new\_hit\_cnt **=** queued\_paths **+** unique\_crashes**;**

stage\_finds**[**STAGE\_EXTRAS\_AO**]** **+=** new\_hit\_cnt **-** orig\_hit\_cnt**;**

stage\_cycles**[**STAGE\_EXTRAS\_AO**]** **+=** stage\_max**;**

skip\_extras**:**

/\* If we made this to here without jumping to havoc\_stage or abandon\_entry,

we're properly done with deterministic steps and can mark it as such

in the .state/ directory. \*/

**if** **(!**queue\_cur**->**passed\_det**)** mark\_as\_det\_done**(**queue\_cur**);**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* RANDOM HAVOC \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

havoc\_stage**:**

stage\_cur\_byte **=** **-**1**;**

/\* The havoc stage mutation code is also invoked when splicing files; if the

splice\_cycle variable is set, generate different descriptions and such. \*/

**if** **(!**splice\_cycle**)** **{**

### 20.1.21 havoc

stage\_name **=** "havoc"**;**

stage\_short **=** "havoc"**;**

stage\_max **=** **(**doing\_det **?** HAVOC\_CYCLES\_INIT **:** HAVOC\_CYCLES**)** **\***

perf\_score **/** havoc\_div **/** 100**;**

**}** **else** **{**

static u8 tmp**[**32**];**

perf\_score **=** orig\_perf**;**

### 20.1.22 splice

sprintf**(**tmp**,** "splice %u"**,** splice\_cycle**);**

stage\_name **=** tmp**;**

stage\_short **=** "splice"**;**

stage\_max **=** SPLICE\_HAVOC **\*** perf\_score **/** havoc\_div **/** 100**;**

**}**

**if** **(**stage\_max **<** HAVOC\_MIN**)** stage\_max **=** HAVOC\_MIN**;**

temp\_len **=** len**;**

orig\_hit\_cnt **=** queued\_paths **+** unique\_crashes**;**

havoc\_queued **=** queued\_paths**;**

/\* We essentially just do several thousand runs (depending on perf\_score)

where we take the input file and make random stacked tweaks. \*/

**for** **(**stage\_cur **=** 0**;** stage\_cur **<** stage\_max**;** stage\_cur**++)** **{**

u32 use\_stacking **=** 1 **<<** **(**1 **+** UR**(**HAVOC\_STACK\_POW2**));**

stage\_cur\_val **=** use\_stacking**;**

**for** **(**i **=** 0**;** i **<** use\_stacking**;** i**++)** **{**

**switch** **(**UR**(**15 **+** **((**extras\_cnt **+** a\_extras\_cnt**)** **?** 2 **:** 0**)))** **{**

**case** 0**:**

/\* Flip a single bit somewhere. Spooky! \*/

FLIP\_BIT**(**out\_buf**,** UR**(**temp\_len **<<** 3**));**

**break;**

**case** 1**:**

/\* Set byte to interesting value. \*/

out\_buf**[**UR**(**temp\_len**)]** **=** interesting\_8**[**UR**(sizeof(**interesting\_8**))];**

**break;**

**case** 2**:**

/\* Set word to interesting value, randomly choosing endian. \*/

**if** **(**temp\_len **<** 2**)** **break;**

**if** **(**UR**(**2**))** **{**

**\*(**u16**\*)(**out\_buf **+** UR**(**temp\_len **-** 1**))** **=**

interesting\_16**[**UR**(sizeof(**interesting\_16**)** **>>** 1**)];**

**}** **else** **{**

**\*(**u16**\*)(**out\_buf **+** UR**(**temp\_len **-** 1**))** **=** SWAP16**(**

interesting\_16**[**UR**(sizeof(**interesting\_16**)** **>>** 1**)]);**

**}**

**break;**

**case** 3**:**

/\* Set dword to interesting value, randomly choosing endian. \*/

**if** **(**temp\_len **<** 4**)** **break;**

**if** **(**UR**(**2**))** **{**

**\*(**u32**\*)(**out\_buf **+** UR**(**temp\_len **-** 3**))** **=**

interesting\_32**[**UR**(sizeof(**interesting\_32**)** **>>** 2**)];**

**}** **else** **{**

**\*(**u32**\*)(**out\_buf **+** UR**(**temp\_len **-** 3**))** **=** SWAP32**(**

interesting\_32**[**UR**(sizeof(**interesting\_32**)** **>>** 2**)]);**

**}**

**break;**

**case** 4**:**

/\* Randomly subtract from byte. \*/

out\_buf**[**UR**(**temp\_len**)]** **-=** 1 **+** UR**(**ARITH\_MAX**);**

**break;**

**case** 5**:**

/\* Randomly add to byte. \*/

out\_buf**[**UR**(**temp\_len**)]** **+=** 1 **+** UR**(**ARITH\_MAX**);**

**break;**

**case** 6**:**

/\* Randomly subtract from word, random endian. \*/

**if** **(**temp\_len **<** 2**)** **break;**

**if** **(**UR**(**2**))** **{**

u32 pos **=** UR**(**temp\_len **-** 1**);**

**\*(**u16**\*)(**out\_buf **+** pos**)** **-=** 1 **+** UR**(**ARITH\_MAX**);**

**}** **else** **{**

u32 pos **=** UR**(**temp\_len **-** 1**);**

u16 num **=** 1 **+** UR**(**ARITH\_MAX**);**

**\*(**u16**\*)(**out\_buf **+** pos**)** **=**

SWAP16**(**SWAP16**(\*(**u16**\*)(**out\_buf **+** pos**))** **-** num**);**

**}**

**break;**

**case** 7**:**

/\* Randomly add to word, random endian. \*/

**if** **(**temp\_len **<** 2**)** **break;**

**if** **(**UR**(**2**))** **{**

u32 pos **=** UR**(**temp\_len **-** 1**);**

**\*(**u16**\*)(**out\_buf **+** pos**)** **+=** 1 **+** UR**(**ARITH\_MAX**);**

**}** **else** **{**

u32 pos **=** UR**(**temp\_len **-** 1**);**

u16 num **=** 1 **+** UR**(**ARITH\_MAX**);**

**\*(**u16**\*)(**out\_buf **+** pos**)** **=**

SWAP16**(**SWAP16**(\*(**u16**\*)(**out\_buf **+** pos**))** **+** num**);**

**}**

**break;**

**case** 8**:**

/\* Randomly subtract from dword, random endian. \*/

**if** **(**temp\_len **<** 4**)** **break;**

**if** **(**UR**(**2**))** **{**

u32 pos **=** UR**(**temp\_len **-** 3**);**

**\*(**u32**\*)(**out\_buf **+** pos**)** **-=** 1 **+** UR**(**ARITH\_MAX**);**

**}** **else** **{**

u32 pos **=** UR**(**temp\_len **-** 3**);**

u32 num **=** 1 **+** UR**(**ARITH\_MAX**);**

**\*(**u32**\*)(**out\_buf **+** pos**)** **=**

SWAP32**(**SWAP32**(\*(**u32**\*)(**out\_buf **+** pos**))** **-** num**);**

**}**

**break;**

**case** 9**:**

/\* Randomly add to dword, random endian. \*/

**if** **(**temp\_len **<** 4**)** **break;**

**if** **(**UR**(**2**))** **{**

u32 pos **=** UR**(**temp\_len **-** 3**);**

**\*(**u32**\*)(**out\_buf **+** pos**)** **+=** 1 **+** UR**(**ARITH\_MAX**);**

**}** **else** **{**

u32 pos **=** UR**(**temp\_len **-** 3**);**

u32 num **=** 1 **+** UR**(**ARITH\_MAX**);**

**\*(**u32**\*)(**out\_buf **+** pos**)** **=**

SWAP32**(**SWAP32**(\*(**u32**\*)(**out\_buf **+** pos**))** **+** num**);**

**}**

**break;**

**case** 10**:**

/\* Just set a random byte to a random value. Because,

why not. We use XOR with 1-255 to eliminate the

possibility of a no-op. \*/

out\_buf**[**UR**(**temp\_len**)]** **^=** 1 **+** UR**(**255**);**

**break;**

**case** 11 **...** 12**:** **{**

/\* Delete bytes. We're making this a bit more likely

than insertion (the next option) in hopes of keeping

files reasonably small. \*/

u32 del\_from**,** del\_len**;**

**if** **(**temp\_len **<** 2**)** **break;**

/\* Don't delete too much. \*/

del\_len **=** choose\_block\_len**(**temp\_len **-** 1**);**

del\_from **=** UR**(**temp\_len **-** del\_len **+** 1**);**

memmove**(**out\_buf **+** del\_from**,** out\_buf **+** del\_from **+** del\_len**,**

temp\_len **-** del\_from **-** del\_len**);**

temp\_len **-=** del\_len**;**

**break;**

**}**

**case** 13**:**

**if** **(**temp\_len **+** HAVOC\_BLK\_XL **<** MAX\_FILE**)** **{**

/\* Clone bytes (75%) or insert a block of constant bytes (25%). \*/

u8 actually\_clone **=** UR**(**4**);**

u32 clone\_from**,** clone\_to**,** clone\_len**;**

u8**\*** new\_buf**;**

**if** **(**actually\_clone**)** **{**

clone\_len **=** choose\_block\_len**(**temp\_len**);**

clone\_from **=** UR**(**temp\_len **-** clone\_len **+** 1**);**

**}** **else** **{**

clone\_len **=** choose\_block\_len**(**HAVOC\_BLK\_XL**);**

clone\_from **=** 0**;**

**}**

clone\_to **=** UR**(**temp\_len**);**

new\_buf **=** ck\_alloc\_nozero**(**temp\_len **+** clone\_len**);**

/\* Head \*/

memcpy**(**new\_buf**,** out\_buf**,** clone\_to**);**

/\* Inserted part \*/

**if** **(**actually\_clone**)**

memcpy**(**new\_buf **+** clone\_to**,** out\_buf **+** clone\_from**,** clone\_len**);**

**else**

memset**(**new\_buf **+** clone\_to**,**

UR**(**2**)** **?** UR**(**256**)** **:** out\_buf**[**UR**(**temp\_len**)],** clone\_len**);**

/\* Tail \*/

memcpy**(**new\_buf **+** clone\_to **+** clone\_len**,** out\_buf **+** clone\_to**,**

temp\_len **-** clone\_to**);**

ck\_free**(**out\_buf**);**

out\_buf **=** new\_buf**;**

temp\_len **+=** clone\_len**;**

**}**

**break;**

**case** 14**:** **{**

/\* Overwrite bytes with a randomly selected chunk (75%) or fixed

bytes (25%). \*/

u32 copy\_from**,** copy\_to**,** copy\_len**;**

**if** **(**temp\_len **<** 2**)** **break;**

copy\_len **=** choose\_block\_len**(**temp\_len **-** 1**);**

copy\_from **=** UR**(**temp\_len **-** copy\_len **+** 1**);**

copy\_to **=** UR**(**temp\_len **-** copy\_len **+** 1**);**

**if** **(**UR**(**4**))** **{**

**if** **(**copy\_from **!=** copy\_to**)**

memmove**(**out\_buf **+** copy\_to**,** out\_buf **+** copy\_from**,** copy\_len**);**

**}** **else** memset**(**out\_buf **+** copy\_to**,**

UR**(**2**)** **?** UR**(**256**)** **:** out\_buf**[**UR**(**temp\_len**)],** copy\_len**);**

**break;**

**}**

/\* Values 15 and 16 can be selected only if there are any extras

present in the dictionaries. \*/

**case** 15**:** **{**

/\* Overwrite bytes with an extra. \*/

**if** **(!**extras\_cnt **||** **(**a\_extras\_cnt **&&** UR**(**2**)))** **{**

/\* No user-specified extras or odds in our favor. Let's use an

auto-detected one. \*/

u32 use\_extra **=** UR**(**a\_extras\_cnt**);**

u32 extra\_len **=** a\_extras**[**use\_extra**].**len**;**

u32 insert\_at**;**

**if** **(**extra\_len **>** temp\_len**)** **break;**

insert\_at **=** UR**(**temp\_len **-** extra\_len **+** 1**);**

memcpy**(**out\_buf **+** insert\_at**,** a\_extras**[**use\_extra**].**data**,** extra\_len**);**

**}** **else** **{**

/\* No auto extras or odds in our favor. Use the dictionary. \*/

u32 use\_extra **=** UR**(**extras\_cnt**);**

u32 extra\_len **=** extras**[**use\_extra**].**len**;**

u32 insert\_at**;**

**if** **(**extra\_len **>** temp\_len**)** **break;**

insert\_at **=** UR**(**temp\_len **-** extra\_len **+** 1**);**

memcpy**(**out\_buf **+** insert\_at**,** extras**[**use\_extra**].**data**,** extra\_len**);**

**}**

**break;**

**}**

**case** 16**:** **{**

u32 use\_extra**,** extra\_len**,** insert\_at **=** UR**(**temp\_len **+** 1**);**

u8**\*** new\_buf**;**

/\* Insert an extra. Do the same dice-rolling stuff as for the

previous case. \*/

**if** **(!**extras\_cnt **||** **(**a\_extras\_cnt **&&** UR**(**2**)))** **{**

use\_extra **=** UR**(**a\_extras\_cnt**);**

extra\_len **=** a\_extras**[**use\_extra**].**len**;**

**if** **(**temp\_len **+** extra\_len **>=** MAX\_FILE**)** **break;**

new\_buf **=** ck\_alloc\_nozero**(**temp\_len **+** extra\_len**);**

/\* Head \*/

memcpy**(**new\_buf**,** out\_buf**,** insert\_at**);**

/\* Inserted part \*/

memcpy**(**new\_buf **+** insert\_at**,** a\_extras**[**use\_extra**].**data**,** extra\_len**);**

**}** **else** **{**

use\_extra **=** UR**(**extras\_cnt**);**

extra\_len **=** extras**[**use\_extra**].**len**;**

**if** **(**temp\_len **+** extra\_len **>=** MAX\_FILE**)** **break;**

new\_buf **=** ck\_alloc\_nozero**(**temp\_len **+** extra\_len**);**

/\* Head \*/

memcpy**(**new\_buf**,** out\_buf**,** insert\_at**);**

/\* Inserted part \*/

memcpy**(**new\_buf **+** insert\_at**,** extras**[**use\_extra**].**data**,** extra\_len**);**

**}**

/\* Tail \*/

memcpy**(**new\_buf **+** insert\_at **+** extra\_len**,** out\_buf **+** insert\_at**,**

temp\_len **-** insert\_at**);**

ck\_free**(**out\_buf**);**

out\_buf **=** new\_buf**;**

temp\_len **+=** extra\_len**;**

**break;**

**}**

**}**

**}**

**if** **(**common\_fuzz\_stuff**(**argv**,** out\_buf**,** temp\_len**))**

**goto** abandon\_entry**;**

/\* out\_buf might have been mangled a bit, so let's restore it to its

original size and shape. \*/

**if** **(**temp\_len **<** len**)** out\_buf **=** ck\_realloc**(**out\_buf**,** len**);**

temp\_len **=** len**;**

memcpy**(**out\_buf**,** in\_buf**,** len**);**

/\* If we're finding new stuff, let's run for a bit longer, limits

permitting. \*/

**if** **(**queued\_paths **!=** havoc\_queued**)** **{**

**if** **(**perf\_score **<=** HAVOC\_MAX\_MULT **\*** 100**)** **{**

stage\_max **\*=** 2**;**

perf\_score **\*=** 2**;**

**}**

havoc\_queued **=** queued\_paths**;**

**}**

**}**

new\_hit\_cnt **=** queued\_paths **+** unique\_crashes**;**

**if** **(!**splice\_cycle**)** **{**

stage\_finds**[**STAGE\_HAVOC**]** **+=** new\_hit\_cnt **-** orig\_hit\_cnt**;**

stage\_cycles**[**STAGE\_HAVOC**]** **+=** stage\_max**;**

**}** **else** **{**

stage\_finds**[**STAGE\_SPLICE**]** **+=** new\_hit\_cnt **-** orig\_hit\_cnt**;**

stage\_cycles**[**STAGE\_SPLICE**]** **+=** stage\_max**;**

**}**

#ifndef IGNORE\_FINDS

/\*\*\*\*\*\*\*\*\*\*\*\*

\* SPLICING \*

\*\*\*\*\*\*\*\*\*\*\*\*/

/\* This is a last-resort strategy triggered by a full round with no findings.

It takes the current input file, randomly selects another input, and

splices them together at some offset, then relies on the havoc

code to mutate that blob. \*/

retry\_splicing**:**

**if** **(**use\_splicing **&&** splice\_cycle**++** **<** SPLICE\_CYCLES **&&**

queued\_paths **>** 1 **&&** queue\_cur**->**len **>** 1**)** **{**

struct queue\_entry**\*** target**;**

u32 tid**,** split\_at**;**

u8**\*** new\_buf**;**

s32 f\_diff**,** l\_diff**;**

/\* First of all, if we've modified in\_buf for havoc, let's clean that

up... \*/

**if** **(**in\_buf **!=** orig\_in**)** **{**

ck\_free**(**in\_buf**);**

in\_buf **=** orig\_in**;**

len **=** queue\_cur**->**len**;**

**}**

/\* Pick a random queue entry and seek to it. Don't splice with yourself. \*/

**do** **{** tid **=** UR**(**queued\_paths**);** **}** **while** **(**tid **==** current\_entry**);**

splicing\_with **=** tid**;**

target **=** queue**;**

**while** **(**tid **>=** 100**)** **{** target **=** target**->**next\_100**;** tid **-=** 100**;** **}**

**while** **(**tid**--)** target **=** target**->**next**;**

/\* Make sure that the target has a reasonable length. \*/

**while** **(**target **&&** **(**target**->**len **<** 2 **||** target **==** queue\_cur**))** **{**

target **=** target**->**next**;**

splicing\_with**++;**

**}**

**if** **(!**target**)** **goto** retry\_splicing**;**

/\* Read the testcase into a new buffer. \*/

fd **=** open**(**target**->**fname**,** O\_RDONLY**);**

**if** **(**fd **<** 0**)** PFATAL**(**"Unable to open '%s'"**,** target**->**fname**);**

new\_buf **=** ck\_alloc\_nozero**(**target**->**len**);**

ck\_read**(**fd**,** new\_buf**,** target**->**len**,** target**->**fname**);**

close**(**fd**);**

/\* Find a suitable splicing location, somewhere between the first and

the last differing byte. Bail out if the difference is just a single

byte or so. \*/

locate\_diffs**(**in\_buf**,** new\_buf**,** MIN**(**len**,** target**->**len**),** **&**f\_diff**,** **&**l\_diff**);**

**if** **(**f\_diff **<** 0 **||** l\_diff **<** 2 **||** f\_diff **==** l\_diff**)** **{**

ck\_free**(**new\_buf**);**

**goto** retry\_splicing**;**

**}**

/\* Split somewhere between the first and last differing byte. \*/

split\_at **=** f\_diff **+** UR**(**l\_diff **-** f\_diff**);**

/\* Do the thing. \*/

len **=** target**->**len**;**

memcpy**(**new\_buf**,** in\_buf**,** split\_at**);**

in\_buf **=** new\_buf**;**

ck\_free**(**out\_buf**);**

out\_buf **=** ck\_alloc\_nozero**(**len**);**

memcpy**(**out\_buf**,** in\_buf**,** len**);**

**goto** havoc\_stage**;**

**}**

#endif /\* !IGNORE\_FINDS \*/

ret\_val **=** 0**;**

abandon\_entry**:**

splicing\_with **=** **-**1**;**

### 20.1.23 结束

/\* Update pending\_not\_fuzzed count if we made it through the calibration

cycle and have not seen this entry before. \*/

**if** **(!**stop\_soon **&&** **!**queue\_cur**->**cal\_failed **&&** **!**queue\_cur**->**was\_fuzzed**)** **{**

queue\_cur**->**was\_fuzzed **=** 1**;**

pending\_not\_fuzzed**--;**

**if** **(**queue\_cur**->**favored**)** pending\_favored**--;**

**}**

munmap**(**orig\_in**,** queue\_cur**->**len**);**

**if** **(**in\_buf **!=** orig\_in**)** ck\_free**(**in\_buf**);**

ck\_free**(**out\_buf**);**

ck\_free**(**eff\_map**);**

**return** ret\_val**;**

#undef FLIP\_BIT

**}**