afl-gcc.c源码分析 源码精注释版

原创

ret2ddme 于 2021-03-04 22:24:14 发布 241 收藏 2

这是对照着看雪一篇文章写的精注释版，基本一行源码一行注释，原本以为会写的很麻烦，但是没想到源码阅读起来并不困难，加上看雪那位大佬写的十分详细，基本就是一行源码一行源码写的解析，所以整个过程十分顺利，下篇继续afl-fuzz
更新：afl-fuzz继续不了了，csdn有字数限制发不上来，等我搞到github仓库里再发，下载下来看着方便

觉得看着源码不方便的可以看上面链接的那篇文章

请从main开始阅读，我事是直接对着源文件写的注释

```
/*
   american fuzzy lop - wrapper for GCC and clang
   ----------------------------------------------

   Written and maintained by Michal Zalewski <lcamtuf@google.com>

   Copyright 2013, 2014, 2015 Google Inc. All rights reserved.

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at:

     http://www.apache.org/licenses/LICENSE-2.0

   This program is a drop-in replacement for GCC or clang. The most common way
   of using it is to pass the path to afl-gcc or afl-clang via CC when invoking
   ./configure.

   (Of course, use CXX and point it to afl-g++ / afl-clang++ for C++ code.)

   The wrapper needs to know the path to afl-as (renamed to 'as'). The default
   is /usr/local/lib/afl/. A convenient way to specify alternative directories
   would be to set AFL_PATH.

   If AFL_HARDEN is set, the wrapper will compile the target app with various
   hardening options that may help detect memory management issues more
   reliably. You can also specify AFL_USE_ASAN to enable ASAN.

   If you want to call a non-default compiler as a next step of the chain,
   specify its location via AFL_CC or AFL_CXX.

 */

#define AFL_MAIN

#include "config.h"
#include "types.h"
#include "debug.h"
#include "alloc-inl.h"
```

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

static u8*  as_path;                /* Path to the AFL 'as' wrapper      */
static u8** cc_params;              /* Parameters passed to the real CC  */
static u32  cc_par_cnt = 1;         /* Param count, including argv0      */
static u8   be_quiet,               /* Quiet mode                        */
            clang_mode;             /* Invoked as afl-clang*?            */


/* Try to find our "fake" GNU assembler in AFL_PATH or at the location derived
   from argv[0]. If that fails, abort. */

static void find_as(u8* argv0) {
  //获取环境变量中的AFL_PATH变量
  u8 *afl_path = getenv("AFL_PATH");
  u8 *slash, *tmp;
  //如果获取AFL_PATH变量成功
  if (afl_path) {
    //动态分配一段空间存储对应的路径
    tmp = alloc_printf("%s/as", afl_path);
    //检查这个路径是否可以访问
    if (!access(tmp, X_OK)) {
      //可以访问就赋值给as_path
      as_path = afl_path;
      //然后free
      ck_free(tmp);
      return;
    }
    //不可以直接free
    ck_free(tmp);

  }
  //如果获取AFL_PATH变量失败
  //就提取当前路径dir
  slash = strrchr(argv0, '/');

  if (slash) {

    u8 *dir;

    *slash = 0;
    dir = ck_strdup(argv0);
    *slash = '/';

    tmp = alloc_printf("%s/afl-as", dir);
    //如果{dir}/afl可访问
    if (!access(tmp, X_OK)) {
      //赋值给as_path
      as_path = dir;
      //free
      ck_free(tmp);
      return;
    }
    //不可访问free
    ck_free(tmp);
    ck_free(dir);
```

```
  }
  //上两种情况都没有成功，就直接找as
  //找到了且可访问就赋值
  if (!access(AFL_PATH "/as", X_OK)) {
    as_path = AFL_PATH;
    return;
  }
  //没找到就输出错误信息然后exit(1)
  FATAL("Unable to find AFL wrapper binary for 'as'. Please set AFL_PATH");

}


/* Copy argv to cc_params, making the necessary edits. */

static void edit_params(u32 argc, char** argv) {
    //设置cc参数
  u8 fortify_set = 0, asan_set = 0;
  u8 *name;

#if defined(__FreeBSD__) && defined(__x86_64__)
  u8 m32_set = 0;
#endif
  //给cc_params分配空间
  cc_params = ck_alloc((argc + 128) * sizeof(u8*));
  //strrchr是查找'/'在字符从右数第一次出现的位置
  //strchr是从左边
  //此时name就是编译器名
  name = strrchr(argv[0], '/');
  if (!name) name = argv[0]; else name++;
  //如果是afl-clang(clang的封装)
  if (!strncmp(name, "afl-clang", 9)) {
    //是clang就clangmode设1，看是afl-clang还是afl-clang++
    clang_mode = 1;

    setenv(CLANG_ENV_VAR, "1", 1);
    if (!strcmp(name, "afl-clang++")) {
      //获取环境变量，如果是AFL_CXX那就是clang++，即c++
      u8* alt_cxx = getenv("AFL_CXX");
      //赋值给cc_params
      cc_params[0] = alt_cxx ? alt_cxx : (u8*)"clang++";
    } else {
      //否则看有没有AFL_CC环境变量
      u8* alt_cc = getenv("AFL_CC");
      //赋值
      cc_params[0] = alt_cc ? alt_cc : (u8*)"clang";
    }
    //如果没有afl-clang
  } else {

    /* With GCJ and Eclipse installed, you can actually compile Java! The
       instrumentation will work (amazingly). Alas, unhandled exceptions do
       not call abort(), so afl-fuzz would need to be modified to equate
       non-zero exit codes with crash conditions when working with Java
       binaries. Meh. */

  //没有afl-clang且是apple平台
#ifdef __APPLE__
    //是否是afl-g++
```

```
    //是否是afl-g++
    if (!strcmp(name, "afl-g++")) cc_params[0] = getenv("AFL_CXX");
    //是否是afl-gcj
    else if (!strcmp(name, "afl-gcj")) cc_params[0] = getenv("AFL_GCJ");
    //否则就赋值有没有AFL_CC环境变量
    else cc_params[0] = getenv("AFL_CC");
    //没get到，报错退出
    if (!cc_params[0]) {

      SAYF("\n" cLRD "[-] " cRST
           "On Apple systems, 'gcc' is usually just a wrapper for clang. Please use the\n"
           "    'afl-clang' utility instead of 'afl-gcc'. If you really have GCC installed,\n"
           "    set AFL_CC or AFL_CXX to specify the correct path to that compiler.\n");

      FATAL("AFL_CC or AFL_CXX required on MacOS X");

    }

#else  //不是apple平台
    //看是不是g++
    if (!strcmp(name, "afl-g++")) {
      //查环境变量
      u8* alt_cxx = getenv("AFL_CXX");
      cc_params[0] = alt_cxx ? alt_cxx : (u8*)"g++";
      //看是不是gcj
    } else if (!strcmp(name, "afl-gcj")) {
      u8* alt_cc = getenv("AFL_GCJ");
      cc_params[0] = alt_cc ? alt_cc : (u8*)"gcj";
    } else {
      //都不是，或环境变量AFL_CC，gcc的
      u8* alt_cc = getenv("AFL_CC");
      cc_params[0] = alt_cc ? alt_cc : (u8*)"gcc";
    }

#endif /* __APPLE__ */

  }
  //进入while循环，扫描argv数组，将参数放入cc_params
  while (--argc) {
    //目前处理的argv参数
    u8* cur = *(++argv);
    //如果扫描到了-B
    //-B 选项用于设置编译器的搜索路径
    if (!strncmp(cur, "-B", 2)) {
      //跳过，因为之前已经处理过as的path了
      if (!be_quiet) WARNF("-B is already set, overriding");

      if (!cur[2] && argc > 1) { argc--; argv++; }
      continue;

    }
    //扫到-integrated-as 跳过
    if (!strcmp(cur, "-integrated-as")) continue;
    //扫到-pipe跳过
    if (!strcmp(cur, "-pipe")) continue;

#if defined(__FreeBSD__) && defined(__x86_64__)
    //64位下设为32程序
    if (!strcmp(cur, "-m32")) m32_set = 1;
#endif
```

```
  //扫到-fsanitize=address或者-fsanitize=memory
  //告诉gcc检查内存访问的错误，比如数组越界
  //设置asan_set=1
  if (!strcmp(cur, "-fsanitize=address") ||
      !strcmp(cur, "-fsanitize=memory")) asan_set = 1;
  //扫到FORTIFY_SOURCE,就设置fortify_set=1
  //FORIFY_SOURE在使用各种字符串和内存操作功能时执行一些轻量级检查，以检查传冲去一处错误
  if (strstr(cur, "FORTIFY_SOURCE")) fortify_set = 1;
  //参数放入cc_params中
  cc_params[cc_par_cnt++] = cur;

}
//跳出循环后,向cc_params中加上-B以及对应的as_path
cc_params[cc_par_cnt++] = "-B";
cc_params[cc_par_cnt++] = as_path;
//检查clang_mode，之前测出是clang时设的值
if (clang_mode)
//如果是clang的话加入-no-integrated-as
  cc_params[cc_par_cnt++] = "-no-integrated-as";
//如果环境变量中有AFL_HARDEN
if (getenv("AFL_HARDEN")) {
  //加参数
  cc_params[cc_par_cnt++] = "-fstack-protector-all";
  //没有设置fortify_set，前面FORTIFY_SOURCE时设置的
  if (!fortify_set)
  //加参数
    cc_params[cc_par_cnt++] = "-D_FORTIFY_SOURCE=2";

}
//如果通过 -fsanitize=设置了asan_set
if (asan_set) {

  /* Pass this on to afl-as to adjust map density. */
  //设置环境变量 AFL_USE_ASAN = 1
  setenv("AFL_USE_ASAN", "1", 1);
  //如果设置了 AFL_USE_ASAN
} else if (getenv("AFL_USE_ASAN")) {
  //继续检测AFL_USE_MSAN，有就报错退出,因为他们互斥
  if (getenv("AFL_USE_MSAN"))
    FATAL("ASAN and MSAN are mutually exclusive");
  //继续检测AFL_HARDEN，有就报错退出
  if (getenv("AFL_HARDEN"))
    FATAL("ASAN and AFL_HARDEN are mutually exclusive");
  //如果没有以上两者，就添加这俩选项
  cc_params[cc_par_cnt++] = "-U_FORTIFY_SOURCE";
  cc_params[cc_par_cnt++] = "-fsanitize=address";
//如果设置了AFL_USE_MSAN
} else if (getenv("AFL_USE_MSAN")) {

  if (getenv("AFL_USE_ASAN"))
    FATAL("ASAN and MSAN are mutually exclusive");

  if (getenv("AFL_HARDEN"))
    FATAL("MSAN and AFL_HARDEN are mutually exclusive");
  //以上这俩如果在AFL_USE_MSAN存在时候存在就报错退出
  //没有就加参数
  cc_params[cc_par_cnt++] = "-U_FORTIFY_SOURCE";
  cc_params[cc_par_cnt++] = "-fsanitize=memory";
```

```
  }
//接下来对于优化选项进行判断
  //没有设置AFL_DONT_OPTIMIZE，也就是允许优化
  if (!getenv("AFL_DONT_OPTIMIZE")) {
//if中是freebsd系统中的事跳过
#if defined(__FreeBSD__) && defined(__x86_64__)

    /* On 64-bit FreeBSD systems, clang -g -m32 is broken, but -m32 itself
       works OK. This has nothing to do with us, but let's avoid triggering
       that bug. */

    if (!clang_mode || !m32_set)
      cc_params[cc_par_cnt++] = "-g";

#else

      cc_params[cc_par_cnt++] = "-g";

#endif

    //在编译选项数组中加入以下四个
    cc_params[cc_par_cnt++] = "-O3";
    cc_params[cc_par_cnt++] = "-funroll-loops";

    /* Two indicators that you're building for fuzzing; one of them is
       AFL-specific, the other is shared with libfuzzer. */

    cc_params[cc_par_cnt++] = "-D__AFL_COMPILER=1";
    cc_params[cc_par_cnt++] = "-DFUZZING_BUILD_MODE_UNSAFE_FOR_PRODUCTION=1";

  }
  //最后，如果设置了 AFL_NO_BUILTIN
  if (getenv("AFL_NO_BUILTIN")) {
    //那么设置以下编译选项
    cc_params[cc_par_cnt++] = "-fno-builtin-strcmp";
    cc_params[cc_par_cnt++] = "-fno-builtin-strncmp";
    cc_params[cc_par_cnt++] = "-fno-builtin-strcasecmp";
    cc_params[cc_par_cnt++] = "-fno-builtin-strncasecmp";
    cc_params[cc_par_cnt++] = "-fno-builtin-memcmp";
    cc_params[cc_par_cnt++] = "-fno-builtin-strstr";
    cc_params[cc_par_cnt++] = "-fno-builtin-strcasestr";

  }
  //最后末尾补一个NULL标值结束，return
  cc_params[cc_par_cnt] = NULL;

}


/* Main entry point */

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

  if (isatty(2) && !getenv("AFL_QUIET")) {

    SAYF(cCYA "afl-cc " cBRI VERSION cRST " by <lcamtuf@google.com>\n");

  } else be_quiet = 1;
```

```
  if (argc < 2) {

    SAYF("\n"
         "This is a helper application for afl-fuzz. It serves as a drop-in replacement\n"
         "for gcc or clang, letting you recompile third-party code with the required\n"
         "runtime instrumentation. A common use pattern would be one of the following:\n\n"

         "  CC=%s/afl-gcc ./configure\n"
         "  CXX=%s/afl-g++ ./configure\n\n"

         "You can specify custom next-stage toolchain via AFL_CC, AFL_CXX, and AFL_AS.\n"
         "Setting AFL_HARDEN enables hardening optimizations in the compiled code.\n\n",
         BIN_PATH, BIN_PATH);

    exit(1);

  }
  // 查找汇编器
  find_as(argv[0]);
  //通过传入编译器的参数来进行参数处理，确定好的放入cc_params数组中
  edit_params(argc, argv);
  //执行afl-gcc
  execvp(cc_params[0], (char**)cc_params);

  FATAL("Oops, failed to execute '%s' - check your PATH", cc_params[0]);

  return 0;

}
```

参考

https://bbs.pediy.com/thread-265936.htm