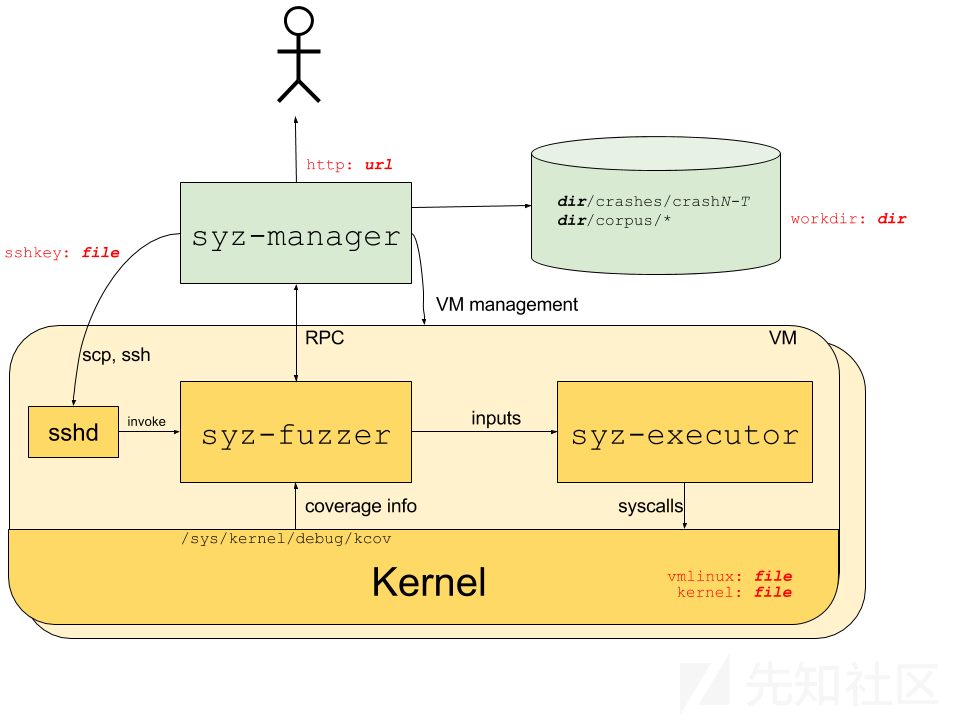
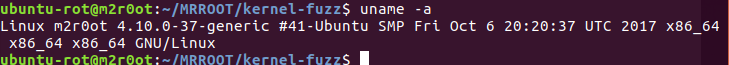
Syzkaller fuzz Demo



Manger format: 不要用kali~~



syz-manager通过ssh调用syz-fuzzer，syz-fuzzer和syz-manager之间通过RPC进行通信。syz-fuzzer将输入传给syz-executor，从kernel中读取代码覆盖率信息。syz-executor执行syscall系统调用。我们再看一下代码的整体目录。

syz-execprog和syz-prog2c可以帮助我们找到引发crash的程序。

# 前驱条件

apt-get -o Acquire::ForceIPv4=true update

* apt-get install subversion #开放源代码的版本控制系统
* apt-get install g++ #编译器
* apt-get install vim #文本编辑器
* apt-get install git #git
* apt-get install kvm qemu-kvm #qemu用于运行内核对象
* apt install libssl-dev libelf-dev #运行库

飞机：

$ sudo add-apt-repository ppa:hzwhuang/ss-qt5

$ sudo apt-get update

$ sudo apt-get install shadowsocks-qt5

VM虚拟机设置网络-全局代理-IP为物理机IP

物理机SS开启局域网连接设置

# 安装gcc

不要装最新版本GCC，会出现许多错误~我用的是GCC4.9

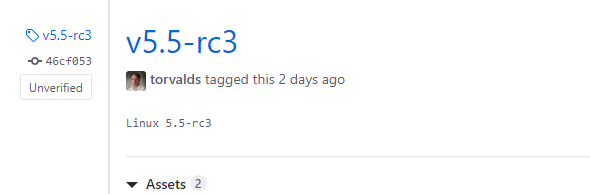
sudo apt-get install flex bison libc6-dev libc6-dev-i386 linux-libc-dev linux-libc-dev:i386 libgmp3-dev libmpfr-dev libmpc-dev

# 内核编译

|  |  |
| --- | --- |
| **命令** | **描述** |
| make config | 普通文本接口 |
| make menuconfig | 基于文本的彩色菜单、选项列表、对话框 |
| make nconfig | 增强的基于文本的彩色菜单 |
| make xconfig | 基于 QT 的配置工具 |
| make gconfig | 基于 GTK+ 的配置工具 |
| make oldconfig | 使用已存在的 ./.config 文件里的配置，然后再询问新版本的配置 |
| make olddefconfig | 和上面一样，但会把新配置设为默认值，而不进行提示 |
| make defconfig | 使用默认值来创建 ./.config 文件，这些值来源于 arch/$ARCH/defconfig 或 arch/$ARCH/configs/${PLATFORM}\_defconfig |
| make ${PLATFORM}\_defconfig | 使用来自于 arch/$ARCH/configs/${PLATFORM}\_defconfig 的默认值创建 ./.config 文件 |
| make allyesconfig | 创建一个 ./.config 文件，其中所有值都是 y |
| make allmodconfig | 创建一个值全是 m 的 ./.config 文件 |
| make allnoconfig | 创建一个所有值为 n 的 ./.config 文件 |
| make randconfig` | 创建一个具有随即值的 ./.config 文件 |
| make localmodconfig | 创建一个基于当前 config 和已加载的模块的 config 文件。禁用那些没必要加载的模块 |
| make localyesconfig | 和 localmodconfig 相似，除了所有模块选项都是 =y（内置） |
| make kvmconfig | 使能用于 kvm guest 内核的额外选项 |
| make xenconfig | 是能用于 xen dom0 guest 内核的额外选项 |
| make tinyconfig | 尽可能配置最小的内核 |

## 下载内核文件

<https://github.com/torvalds/linux/releases/tag/v5.5-rc3>



## 生成内核配置

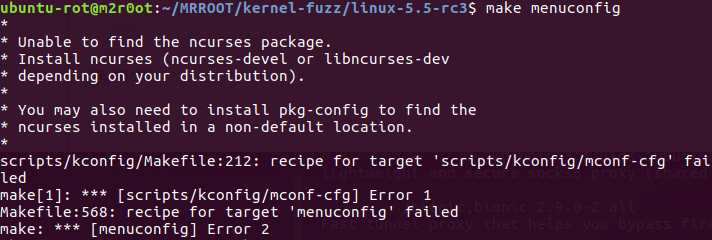
利用上述表格进行.config配置生成。

make mrproper

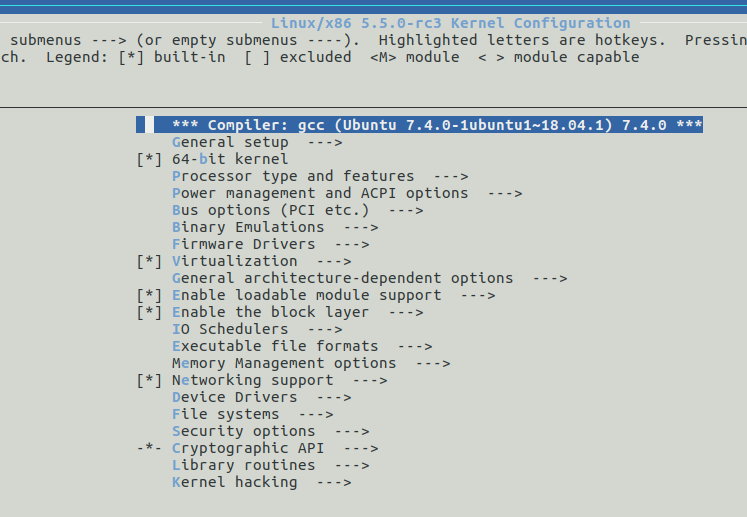
make CC="使用的gcc " defconfig

make CC="使用的gcc " kvmconfig

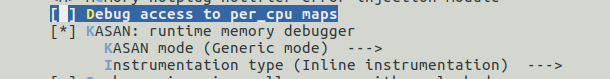
这里我是用 make menuconfig可视化配置



装缺少的库



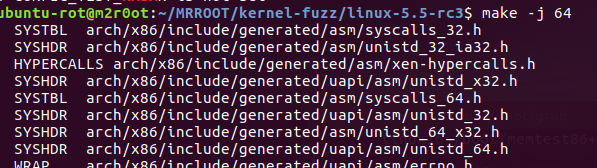
主要开启





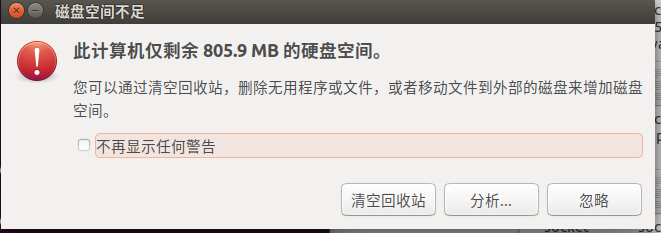
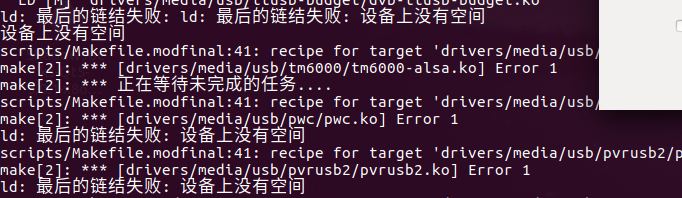
## 编译内核

make CC=‘你的GCC’ -j64



###用-j64会出现bug，这里我们选择-j8

###中间件太大了~



出现vmlinuxz和arch/x86/boot/bzImage即成功。

# 安装Syzkaller

## 安装Go

**wget** [**https://storage.googleapis.com/golang/go1.8.1.linux-amd64.tar.gz**](https://storage.googleapis.com/golang/go1.8.1.linux-amd64.tar.gz)

**tar -xf go1.8.1.linux-amd64.tar.gz**

**#mv go goroot**

**export GOROOT=`pwd`/goroot**

**export PATH=$PATH:$GOROOT/bin**

**mkdir gopath**

**export GOPATH=`pwd`/gopath**

## 安装Syzkaller

go get -u -d github.com/google/syzkaller/...

cd gopath/src/github.com/google/syzkaller/

mkdir workdir

make CC=”交叉编译”

# Debian-wheezy制作镜像

系统=kernel+rootfs

要能使用运行的Linux系统，必须具备内核和根文件系统。

sudo apt-get install debootstrap

# debootstrap是debian/ubuntu下的一个工具，用来构建一套基本的系统(根文件系统)。生成的目录符合Linux文件系统标准(FHS)，即包含了/boot、/etc、/bin、/usr等等目录，但它比发行版本的Linux体积小很多，当然功能也没那么强大，因此，只能说是“基本的系统”。

cd gopath/src/github.com/google/syzkaller/tools/

./create-image.sh

成功！

# ~~Busybox制作根目录~~

~~下载：~~[~~http://www.busybox.net/~~](http://www.busybox.net/)

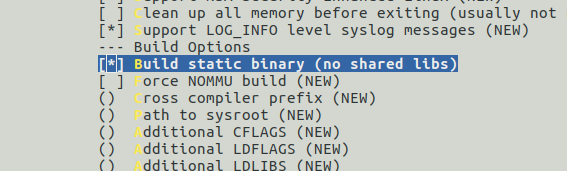
~~编译：make menuconfig~~

~~因为Linux运行环境当中是不带动态库的，所以必须以静态方式来编译BusyBox。修改~~

~~Busybox Settings --->~~

~~Build Options --->~~

~~[\*] Build BusyBox as a static binary(no shared libs)~~



# Qemu启动内核

export KERNEL=/home/ubuntu-rot/MRROOT/kernel-fuzz/linux

export IMG=/home/ubuntu-rot/MRROOT/kernel-fuzz/gopath/src/github.com/google/syzkaller/tools

qemu-system-x86\_64 \

-kernel $KERNEL/arch/x86\_64/boot/bzImage \

-append "console=ttyS0 root=/dev/sda debug earlyprintk=serial slub\_debug=QUZ"\

-hda $IMG/stretch.img \

-net user,hostfwd=tcp::10021-:22 \

-net nic \

-nographic \

-enable-kvm \

-m 2G \

-smp 2 \

-pidfile vm.pid \

2>&1 | tee vm.log

注释：

模拟x86硬件平台

-kernel：指定内核

-m：指定内存大小

-append：内核启动参数。（控制端，根文件系统所在设备，开启调试，内存检测技术）

-had：指定硬盘

-smp ：CPU核数

-enable-kvm：KVM进行加速

-net nic：创建虚拟网卡

-net user,hostfwd=tcp::10021-:22 ：客户机（虚拟机）与主机之间通讯可通过hostfwd参数指定转发端口来实现

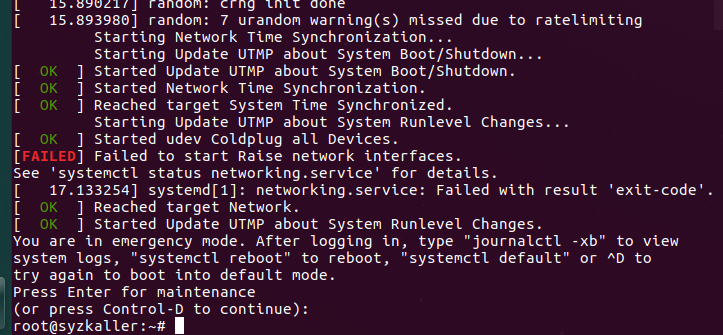
[tcp|udp]:[主机ip]:主机端口-[客户机ip]:客户机端口

- nographic：无图形界面运行

-pidfile：存储qemu进程PID到文件，用于脚本启动



VMware中启动上述虚拟化技术，因为Qemu要用到KVM

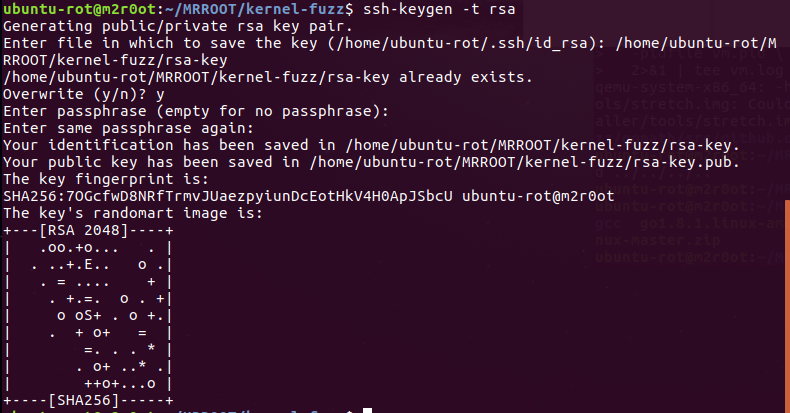


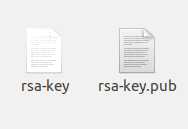
出现上述，表示成功~

要结束当前的qemu实例用下面命令：

sudo kill $(cat vm.pid)

# 配置免密登录SSH



生产rsa密钥

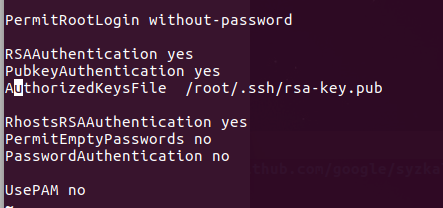
将公钥复制到Qemy虚拟机上

Cd /root/.ssh

Touch rsa-key.pub

复制

修改虚拟机上ssh的配置：/etc/ssh/sshd\_config



重启SSH服务

/etc/init.d/ssh restart

本机测试登录

ssh -p 10021 -i rsa-key [root@127.0.0.1](mailto:root@127.0.0.1)

scp -P 10021 -i rsa-key -r $(GOPATH)/bin root@127.0.0.1



# 配置

## 添加配置文件

编写syzkaller配置文件

{

"target": "linux/amd64",

"http": "127.0.0.1:10233",

"workdir": "$GOPATH/src/github.com/google/syzkaller/workdir",

"kernel\_obj": "$KERNEL ",

"image": "$IMG/stretch.img ",

"kernel\_obj": "/linux/",

"sshkey": " rsa-key",

"syzkaller": "$GOPATH/src/github.com/google/syzkaller",

"enable\_syscalls": [

"open$proc",

"read$proc",

"write$proc",

"close$proc"

],

"procs": 4,

"type": "qemu",

"vm": {

"count": 4,

"kernel": "$KERNEL/arch/x86\_64/boot/bzImage",

"cpu": 2,

"mem": 2048

}

}

touch my.cfg

./bin/syz-manager -config=my.cfg

{##参数翻译

"target": 目标框架,

"http": fuzzer仪表盘,

"workdir": syz-manager进程的工作目录的位置,

"kernel\_obj": 包含目标文件的目录，例如linux中的vmlinux,

"image": qemu实例的磁盘镜像文件的位置,

"sshkey": 用于与虚拟机通信的SSH密钥的位置,

"syzkaller": syzkaller的位置,

"procs": 每个VM中的并行测试进程数,

"enable\_syscalls": 测试的系统调用列表,[“syscall$type”,…]

"disable\_syscalls"：禁用的系统调用列表

"suppressions": 已知错误的正则表达式列表,

"type": 要使用的虚拟机类型，例如qemu,

"vm": {

"count": 并行运行的VM数,

"kernel": 要测试的内核的bzImage文件的位置,

"cpu": 要在VM中模拟的CPU数,

"mem": 2048

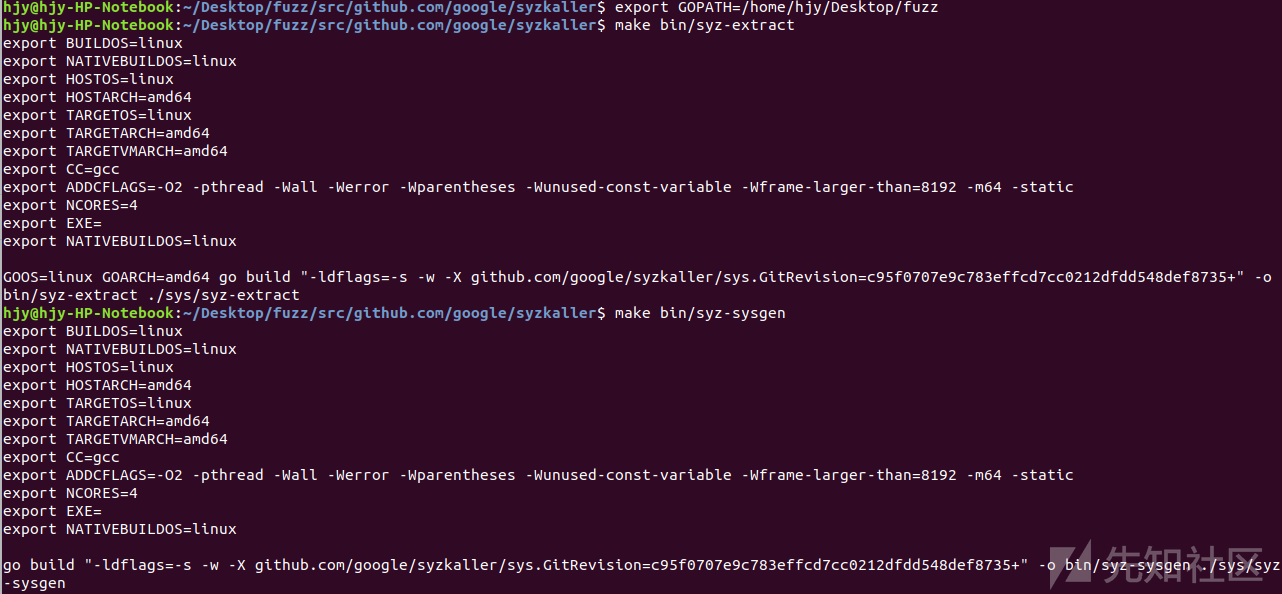
}

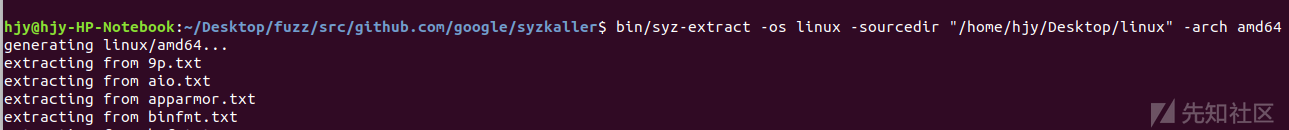
}

<https://github.com/google/syzkaller/blob/master/pkg/mgrconfig/config.go>

## fuzz特定驱动

1. 第一步是使用syz-extract从linux源代码中提取符号常量的值，结果被存储在.const文件中，例如/sys/linux/tty.txt被转换为sys/linux/tty\_amd64.const
2. 第二步是根据系统调用模板和第一步中生成的const文件使用syz-sysgen生成syzkaller用的go代码。可以在/sys/linux/gen/amd64.go和/executor/syscalls.h中看到结果。





**手动添加驱动调用：即编写xxx.txt文件**

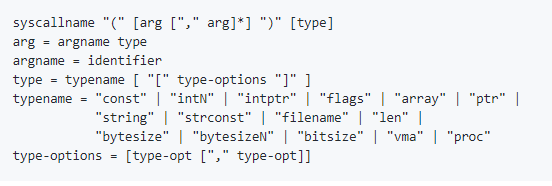
---------------------------xxx.txt--------------------------------

|  |
| --- |
| include <linux/fs.h>  open$proc(file ptr[in, string["/proc/test"]], flags flags[proc\_open\_flags], mode flags[proc\_open\_mode]) fd  read$proc(fd, buf buffer[out], count len[buf]) len[buf]  write$proc(fd fd, buf buffer[in], count len[buf]) len[buf]  close$proc(fd fd)  proc\_open\_flags = O\_RDONLY, O\_WRONLY, O\_RDWR, O\_APPEND, FASYNC, O\_CLOEXEC, O\_CREAT, O\_DIRECT, O\_DIRECTORY, O\_EXCL, O\_LARGEFILE, O\_NOATIME, O\_NOCTTY, O\_NOFOLLOW, O\_NONBLOCK, O\_PATH, O\_SYNC, O\_TRUNC, \_\_O\_TMPFILE  proc\_open\_mode = S\_IRUSR, S\_IWUSR, S\_IXUSR, S\_IRGRP, S\_IWGRP, S\_IXGRP, S\_IROTH, S\_IWOTH, S\_IXOTH |

---------------------------xxx.txt--------------------------------

resource sock\_packet[sock]

* 声明一个新类型，继承sock



**SyscallName$Type（ArgumentName ArgumentType[Limit,opt]…）re-arg**

* **SyscallName**：系统调用名，是内核提供的接口，在源码的$(SYZKALLER\_SOURCE)/sys/sys.txt中有通用的调用的形式申明可以参考,告诉fuzz使用哪种系统调用~。
* **Type**：指特定类型的系统调用，这个名字是由规则编写者确定的，具体行为靠的是后面的参数去确定。
* **ArgumentName**：参数名称
* [**ArgumentType**](https://github.com/google/syzkaller/blob/master/docs/syscall_descriptions_syntax.md)：参数类型

|  |
| --- |
| * "const": integer constant, type-options:   value, underlying type (one of "intN", "intptr")  ## f1 const[0x42, int16be] –f1取值为0x0042，宽度为16be大段   * "intN"/"intptr": an integer without a particular meaning, type-options:   optional range of values (e.g. "5:10", or "100:200"),  optionally followed by an alignment parameter  ## f3 int32[1:10, 2] -f3取值范围{1, 3, 5, 7, 9}，2为字节对齐宽度~  ## f2 int32[0:100] -f2 0 to 100随机值  ## f4 int64:20 -f4 20-bit随机值   * "flags": a set of flags, type-options:   reference to flags description (see below), underlying int type (e.g. "int32")  ## f5 flags[argv] -f5从argv罗列的数据取值   * "array": a variable/fixed-length array, type-options:   type of elements, optional size (fixed "5", or ranged "5:10", boundaries inclusive)   * "ptr"/"ptr64": a pointer to an object, type-options:   direction (in/out/inout); type of the object  ptr64 has size of 8 bytes regardless of target pointer size   * "string": a zero-terminated memory buffer (no pointer indirection implied), type-options:   either a string value in quotes for constant strings (e.g. "foo"),  or a reference to string flags (special value `filename` produces file names),  optionally followed by a buffer size (string values will be padded with \x00 to that size)   * "stringnoz": a non-zero-terminated memory buffer (no pointer indirection implied), type-options:   either a string value in quotes for constant strings (e.g. "foo"),  or a reference to string flags,   * "fmt": a string representation of an integer (not zero-terminated), type-options:   format (one of "dec", "hex", "oct") and the value (a resource, int, flags, const or proc)  the resulting data is always fixed-size (formatted as "%020llu", "0x%016llx" or "%023llo", respectively)   * "len": length of another field (for array it is number of elements), type-options:   argname of the object   * "bytesize": similar to "len", but always denotes the size in bytes, type-options:   argname of the object   * "bitsize": similar to "len", but always denotes the size in bits, type-options:   argname of the object   * "offsetof": offset of the field from the beginning of the parent struct, type-options:   field   * "vma"/"vma64": a pointer to a set of pages (used as input for mmap/munmap/mremap/madvise), type-options:   optional number of pages (e.g. vma[7]), or a range of pages (e.g. vma[2-4])  vma64 has size of 8 bytes regardless of target pointer size   * "proc": per process int (see description below), type-options:   value range start, how many values per process, underlying type   * "text": machine code of the specified type, type-options:   text type (x86\_real, x86\_16, x86\_32, x86\_64, arm64)   * "void": type with static size 0   mostly useful inside of templates and varlen unions, can't be syscall argument |

* [**Limit,opt**]:[]中是取值，Limit为取值，’,’分割后面为opt
* **re-arg**:函数返回值

## 重编译syzkaller

make clean

make bin/syz-extract

bin/syz-extract -arch amd64 -linux /PATH/TO/LINUX/SOURCE sys/xxx.txt

## -linux待测内核的编译目录

make all

# 拷贝重编译成功的二进制进虚机

scp -P $(SSH\_PORT) -i ~/.ssh/id\_rsa -r syzkaller/bin [root@127.0.0.1:/root/bin](mailto:root@127.0.0.1:/root/bin)

# 运行Fuzz

./bin/syz-manager -config=my.cfg -v 10

用浏览器打开127.0.0.1：10233

# 参考

<https://www.freebuf.com/sectool/142969.html>

<http://embedsec.systems/zh/gnulinux-security/2017/06/05/syzkaller-demo.html>

<https://github.com/hardenedlinux/Debian-GNU-Linux-Profiles/tree/master/docs/harbian_qa/fuzz_testing>

https://github.com/google/syzkaller/blob/master/docs/syscall\_descriptions\_syntax.md

https://xz.aliyun.com/t/5079

<https://xz.aliyun.com/t/5154>

<https://xz.aliyun.com/t/5098>

<https://xz.aliyun.com/t/2714#toc-10>

https://xz.aliyun.com/t/5098#toc-0

|  |
| --- |
| ##xxx.txt  include <linux/fs.h>  open$proc(file ptr[in, string["/proc/test"]], flags flags[proc\_open\_flags], mode flags[proc\_open\_mode]) fd  read$proc(fd, buf buffer[out], count len[buf]) len[buf]  write$proc(fd, buf buffer[in], count len[buf]) len[buf]  close$proc(fd fd)  proc\_open\_flags = O\_RDONLY, O\_WRONLY, O\_RDWR, O\_APPEND, FASYNC, O\_CLOEXEC, O\_CREAT, O\_DIRECT, O\_DIRECTORY, O\_EXCL, O\_LARGEFILE, O\_NOATIME, O\_NOCTTY, O\_NOFOLLOW, O\_NONBLOCK, O\_PATH, O\_SYNC, O\_TRUNC, \_\_O\_TMPFILE  proc\_open\_mode = S\_IRUSR, S\_IWUSR, S\_IXUSR, S\_IRGRP, S\_IWGRP, S\_IXGRP, S\_IROTH, S\_IWOTH, S\_IXOTH |
| //test.c  #include <linux/init.h>  #include <linux/module.h>  #include <linux/proc\_fs.h>  #include <linux/uaccess.h>  #include <linux/slab.h>  #define MY\_DEV\_NAME "test"  #define DEBUG\_FLAG "PROC\_DEV"  static ssize\_t proc\_read **(**struct file **\***proc\_file**,** char \_\_user **\***proc\_user**,** size\_t n**,** loff\_t **\***loff**);**  static ssize\_t proc\_write **(**struct file **\***proc\_file**,** const char \_\_user **\***proc\_user**,** size\_t n**,** loff\_t **\***loff**);**  static int proc\_open **(**struct inode **\***proc\_inode**,** struct file **\***proc\_file**);**  static struct file\_operations a **=** **{**  **.**open **=** proc\_open**,**  **.**read **=** proc\_read**,**  **.**write **=** proc\_write**,**  **};**  static int \_\_init mod\_init**(**void**)**  **{**  struct proc\_dir\_entry **\***test\_entry**;**  const struct file\_operations **\***proc\_fops **=** **&**a**;**  printk**(**DEBUG\_FLAG":proc init start!\n"**);**  test\_entry **=** proc\_create**(**MY\_DEV\_NAME**,** S\_IRUGO**|**S\_IWUGO**,** **NULL,** proc\_fops**);**  **if(!**test\_entry**)**  printk**(**DEBUG\_FLAG":there is somethings wrong!\n"**);**    printk**(**DEBUG\_FLAG":proc init over!\n"**);**  **return** 0**;**  **}**  static ssize\_t proc\_read **(**struct file **\***proc\_file**,** char \_\_user **\***proc\_user**,** size\_t n**,** loff\_t **\***loff**)**  **{**  printk**(**DEBUG\_FLAG":finish copy\_from\_use,the string of newbuf is"**);**  **return** 0**;**  **}**  static ssize\_t proc\_write **(**struct file **\***proc\_file**,** const char \_\_user **\***proc\_user**,** size\_t n**,** loff\_t **\***loff**)**  **{**  char **\***c **=** kmalloc**(**512**,** GFP\_KERNEL**);**  copy\_from\_user**(**c**,** proc\_user**,** 4096**);**  printk**(**DEBUG\_FLAG":into write!\n"**);**  **return** 0**;**  **}**  int proc\_open **(**struct inode **\***proc\_inode**,** struct file **\***proc\_file**)**  **{**  printk**(**DEBUG\_FLAG":into open!\n"**);**  **return** 0**;**  **}**  module\_init**(**mod\_init**);** |