如何获取一个crash的复现程序

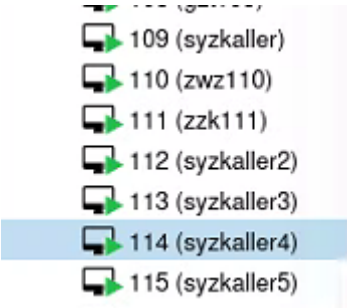
1. 将log程序转移至操作用的虚拟机中

所有syzkaller相关虚拟机（VM109，VM112~115）的密码皆为：qinjiaji

syzkaller：VM115

向日葵特征码：663511178

验证码：qaz123



2. 利用syz-repro从log中获取可复现程序

2.1 获取可复现的系统调用序列

```
1 ./bin/syz-repro -config=my613.cfg -output=repro.txt -crepro=repro.c -
  title=bug_title.txt  log0
```

该命令将执行以下操作：

1. 加载 `manager.cfg` 配置文件。
2. 读取 `crash.log` 崩溃日志文件。
3. 创建虚拟机池并开始重现过程。
4. 将重现的程序保存到 `repro.txt` 。
5. 生成 C 语言的重现程序并保存到 `repro.c` 。
6. 保存 bug 的标题到 `bug_title.txt` 。

syz-repro会不断尝试最小化复现程序，过程如下：

但值得注意的是，最终的复现程序由于过于简短，反而不能稳定触发crash。此时可选取中间POC作为最终的复现程序。

```
1 # V1
```

```
perf_event_open(&(0x7f00000003c0)={0x2, 0x80, 0xe7, 0x0, 0x0, 0x0, 0x0, 0x1,
0x44c21d, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x203, 0x0,
@perf_config_ext={0x202ece72, 0xfffffffffffffe05}, 0x2802, 0x3, 0x91, 0x6,
0x8, 0x0, 0x0, 0x0, 0x0, 0x0, 0x14}, 0x0, 0xffffffffffffffff,
0xffffffffffffffff, 0x0) (async, rerun: 64)
r0 = socket$pppoe(0x18, 0x1, 0x0) (rerun: 64)
connect$pppoe(r0, &(0x7f0000000400)={0x18, 0x0, {0x2, @dev=
{'\xaa\xaa\xaa\xaa\xaa', 0xa}, 'lo\x00'}}, 0x1e) (async)
r1 = socket$nl_rdma(0x10, 0x3, 0x14)
openat$rdma_cm(0xffffffffffffff9c, &(0x7f0000000280), 0x2, 0x0) (async)
sendmsg$RDMA_NLDEV_CMD_NEWLINK(r1, &(0x7f0000000440)={0x0, 0x0, &
(0x7f0000000400)={&(0x7f0000000280)=ANY=[@ANYBLOB="380000000141401"], 0x38},
0x1, 0x0, 0x0, 0x4000}, 0x10)

# V2
perf_event_open(&(0x7f00000003c0)={0x2, 0x80, 0xe7, 0x0, 0x0, 0x0, 0x0, 0x1,
0x44c21d, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x203, 0x0,
@perf_config_ext={0x202ece72, 0xfffffffffffffe05}, 0x2802, 0x3, 0x91, 0x6,
0x8, 0x0, 0x0, 0x0, 0x0, 0x0, 0x14}, 0x0, 0xffffffffffffffff,
0xffffffffffffffff, 0x0) (async, rerun: 64)
r0 = socket$pppoe(0x18, 0x1, 0x0) (rerun: 64)
connect$pppoe(r0, &(0x7f0000000400)={0x18, 0x0, {0x2, @dev=
{'\xaa\xaa\xaa\xaa\xaa', 0xa}, 'lo\x00'}}, 0x1e) (async)
socket$nl_rdma(0x10, 0x3, 0x14)
openat$rdma_cm(0xffffffffffffff9c, &(0x7f0000000280), 0x2, 0x0) (async)

# V3
perf_event_open(&(0x7f00000003c0)={0x2, 0x80, 0xe7, 0x0, 0x0, 0x0, 0x0, 0x1,
0x44c21d, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x203, 0x0,
@perf_config_ext={0x202ece72, 0xfffffffffffffe05}, 0x2802, 0x3, 0x91, 0x6,
0x8, 0x0, 0x0, 0x0, 0x0, 0x0, 0x14}, 0x0, 0xffffffffffffffff,
0xffffffffffffffff, 0x0) (async, rerun: 64)
r0 = socket$pppoe(0x18, 0x1, 0x0) (rerun: 64)
connect$pppoe(r0, &(0x7f0000000400)={0x18, 0x0, {0x2, @dev=
{'\xaa\xaa\xaa\xaa\xaa', 0xa}, 'lo\x00'}}, 0x1e) (async)
socket$nl_rdma(0x10, 0x3, 0x14)
# 红色部分是syz-repro提供的最终的POC，红色部分+绿色部分是在不断最小化的中间POC。经测试，中间POC可以稳定触发该crash，而最终POC则不能。
# 分析：按照sykaller文档所说，syz-repro并不完美，所以可能功能存在一定问题
```

```
 </TASK>
extracting prog: 1m58.787654136s
minimizing prog: 21m57.530880135s
simplifying prog options: 7m25.791778345s
extracting C: 5m5.936792302s
simplifying C: 0s
opts: {Threaded:true Repeat:true RepeatTimes:0 Procs:8 Slowdown:1 Sandbox:none S
andboxArg:0 Leak:false NetInjection:true NetDevices:true NetReset:true Cgroups:t
rue BinfmtMisc:true CloseFDs:true KCSAN:false DevlinkPCI:true NicVF:true USB:tru
e VhciInjection:true Wifi:true IEEE802154:true Sysctl:true Swap:true UseTmpDir:t
rue HandleSegv:true Trace:false LegacyOptions:{Collide:false Fault:false FaultCa
ll:0 FaultNth:0}} crepro: false

perf_event_open(&(0x7f00000003c0)={0x2, 0x80, 0xe7, 0x0, 0x0, 0x0, 0x0, 0x1, 0x4
4c21d, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x203, 0x0, @perf_config_ext={0x202ece
72, 0xfffffffffffffe05}, 0x2802, 0x3, 0x91, 0x6, 0x8, 0x0, 0x0, 0x0, 0x0, 0x0, 0
x14}, 0x0, 0xffffffffffffffff, 0xffffffffffffffff, 0x0) (rerun: 64)

program saved to repro.txt
```

2.2 将系统调用序列转换为C程序

```
./bin/syz-prog2c -prog=repro.syz -os=linux -arch=amd64 -build > repro

./bin/syz-prog2c -prog=repro.syz > generated_program.c

./generated_program
```

`-build`：如果设置为 `true`，工具将在生成 C 源代码后尝试编译它。

`-threaded`：生成多线程程序。

`-repeat`：指定程序重复执行的次数（`<=0` 表示无限重复）。

`-procs`：并行进程的数量。

`-slowdown`：模拟执行时的慢速（例如，由于仿真或插装引起的执行减慢）。

`-prog`：指定要转换的 Syzkaller 程序文件。

```
x68\x76\x86\x5d\x89\x5b\x25\x82\xef\x04\x9d\xf3\x9c\xb3\xb0\xe6\x66\x4a\x7b\xdf\
x2b\x3f\x69\xc6\x04\xa9\x8b\xb9\xf7\x34\xde\x33\x47\x93\x67\x09\xa2\x7d\x8a\x34\
xc5\x89\xbd\x72\x6e\xaf\xee\x7e\x13\x8f\x4a\x6c\xc6\x89\xf6\xff\x69\x6f\x79\xc0\
xda\x26\x73\xf2\xa0\xb1\x31\x29\xea\xee\x1e\xb0\x8e\xae\xf1\x44\xcd\x15\xb6\x78\
xa5\xde\xb5\x14\x1d\x23\x1d\xbb\x99\xfb\xbd\x35\x95\x8e\x5c\xf8\xb2", 290);
        syscall(__NR_write, /*fd=*/r[0], /*buffer=*/0x20000140ul, /*count=*/8ul)
;
        for (int i = 0; i < 64; i++) {
        syscall(__NR_write, /*fd=*/r[0], /*buffer=*/0x20000140ul, /*count=*/8ul)
;
        }
        return 0;
}
binary build OK
(base) qjj@syzkaller109:~/go1.22.1_projects/go_projects/syzkaller$
```

3. 利用syz-execprog检验可复现程序的有效性

syzkaller官方文档：reproducing_crashes.md

以下以1220_6.13rc_KASAN中的KASAN_ slab-out-of-bounds Write in snd_seq_oss_synth_sysex为例进行说明：

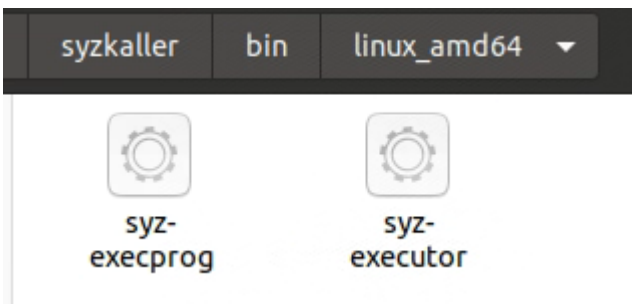
注意reproducing_crashes.md 中复现的是POC中列出的crash，以下均基于Syz Reproducer实现本地发现的crash。

3.1 编译syz-execprog，syz-executor，syz-exec2prog

在syzkaller路径下运行：

```
1 make execprog
2 make executor
3 make prog2c
```

编译结果存放在syzkaller/bin/linux_amd64中，注意该路径



3.2 确定内核版本并编译

```
本地一般已经具有编译后的内核，记录路径即可
```

Kernel Commit：导致该crash 的内核提交版本。

```
# 6.13rc3 - 2024-12-15
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/log/?id=78d4f34e2115b517bcbfe7ec0d018bbbb6f9b0b8
```

Kernel Config：用于构建内核的配置文件。此例选择本地的6.13rc3所采用配置文件

3.3 启动虚拟机

```
需要一个可启动磁盘镜像。Syzbot目前使用基于Buildroot的小型镜像，可以在本地构建或下载.NET镜像。
本地的crash可以直接使用syzkaller的启动镜像。
```

Disk Image启动命令：

```
# 用debug指令获取镜像的启动命令
./bin/syz-manager -config=my613.cfg -debug
```

```
qemu-system-x86_64 \
  -m 2048 \
  -smp 4 \
  -chardev socket,id=SOCKSYZ,server=on,wait=off,host=localhost,port=58191 \
  -mon chardev=SOCKSYZ,mode=control \
  -display none \
  -serial stdio \
  -no-reboot \
  -name VM-0 \
  -device virtio-rng-pci \
  -enable-kvm \
  -cpu host,migratable=off \
```

```
  -device e1000,netdev=net0 \
  -netdev user,id=net0,restrict=on,hostfwd=tcp:127.0.0.1:64317-:22 \
  -hda /home/qjj/go1.22.1_projects/go_projects/image/bullseye.img \
  -snapshot \
  -kernel
/home/qjj/go1.22.1_projects/go_projects/Newlinux/arch/x86/boot/bzImage \
  -append "root=/dev/sda console=ttyS0"
```

3.4 修改虚拟机配置

1.新增密码

```
passwd
```

2.允许外部访问

打开config文件：

```
nano /etc/ssh/sshd_config
```

在文件中新增以下字段：

```
PermitRootLogin yes
PasswordAuthentication yes
```

重启服务

```
systemctl restart ssh
systemctl status ssh
```

3.5 ~~准备Syz Reproducer：准备用于复现 crash 的程序。~~

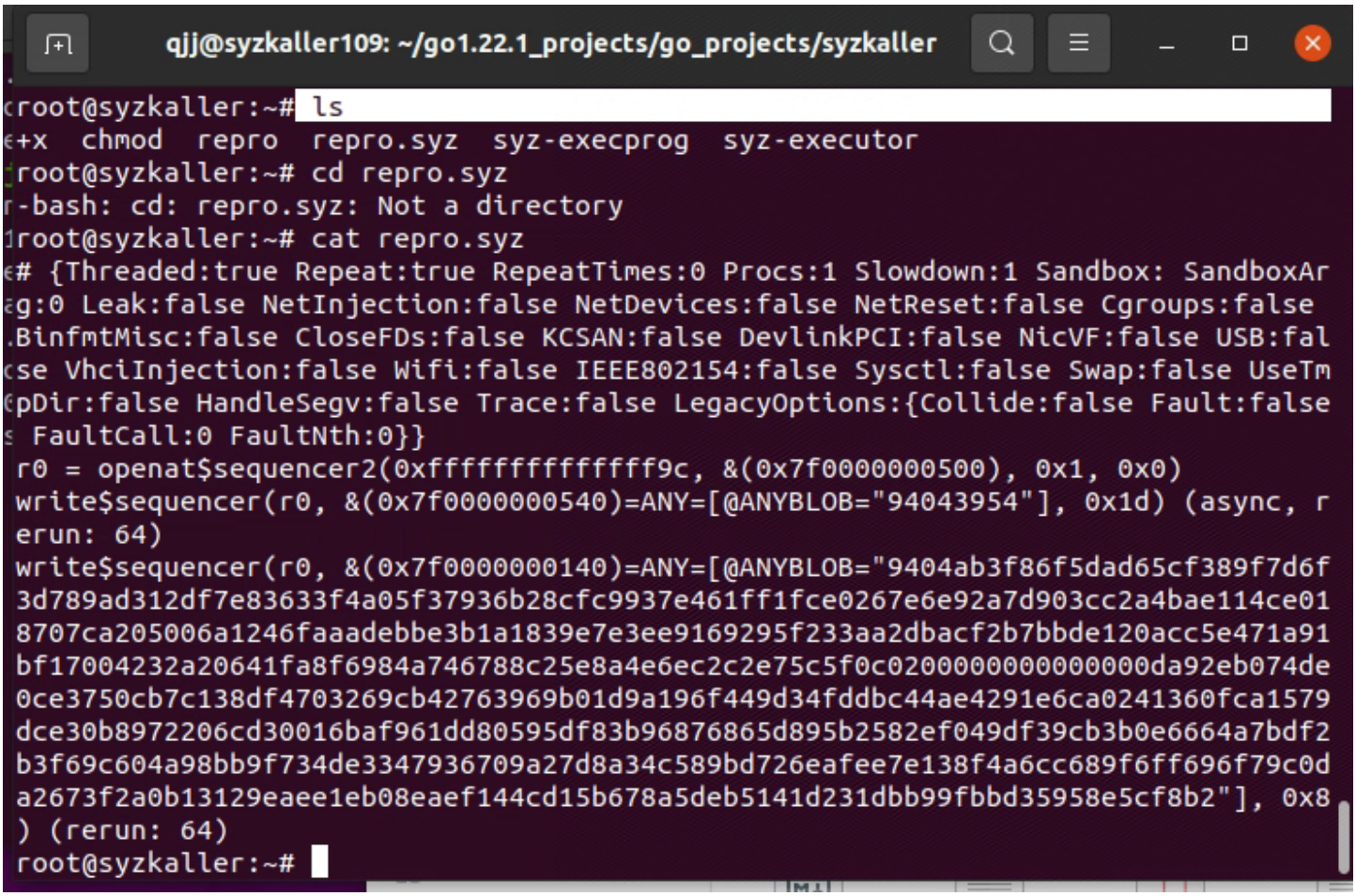
```
# 复现syzbot报告的crash时用如下指令获取复现程序
wget -O 'repro.c' 'https://syzkaller.appspot.com/x/repro.c?x=177beac0580000'
# 复现本地发现的crash时则直接将对应复现程序转移至syzkaller目录中
```

3.6 将 Reproducer 传输到虚拟机并运行：

使用 `scp` 将编译好的 `repro` 程序传输到虚拟机：使用 `scp` 命令将 syzkaller 的二进制文件（syzkaller 目录下 `bin/linux_amd64` 目录中的所有文件）和 Reproducer 程序（当前目录下的 `repro.syz` 文件）传输到远程虚拟机（通过 SSH 端口转发到本地的 127.0.0.1:64317），注意端口应与第三步的vm启动命令中内容相符。

```
#上传复现工具及复现程序到虚拟机中
scp -P 64317 -o UserKnownHostsFile=/dev/null  -o StrictHostKeyChecking=no -o IdentitiesOnly=yes /home/qjj/go1.22.1_projects/go_projects/syzkaller/bin/linux_amd64/* ./repro.syz root@127.0.0.1:/root/

#执行复现程序
ssh -p 64317 -o UserKnownHostsFile=/dev/null  -o StrictHostKeyChecking=no -o IdentitiesOnly=yes root@127.0.0.1 './syz-execprog -enable=all -repeat=0 -procs=8 -cover=0 repro.syz'
```

```
export SYZKALLER_PATH="~/syzkaller"
scp -P 64317 -o UserKnownHostsFile=/dev/null  -o StrictHostKeyChecking=no -o IdentitiesOnly=yes $SYZKALLER_PATH/bin/linux_amd64/* ./repro.syz root@127.0.0.1:/root/

# 以下是绝对路径
scp -P 64317 -o UserKnownHostsFile=/dev/null  -o StrictHostKeyChecking=no -o IdentitiesOnly=yes /home/qjj/go1.22.1_projects/go_projects/syzkaller/bin/linux_amd64/* ./repro.syz root@127.0.0.1:/root/
```



然后通过 `ssh` 或者虚拟机中手动执行程序：

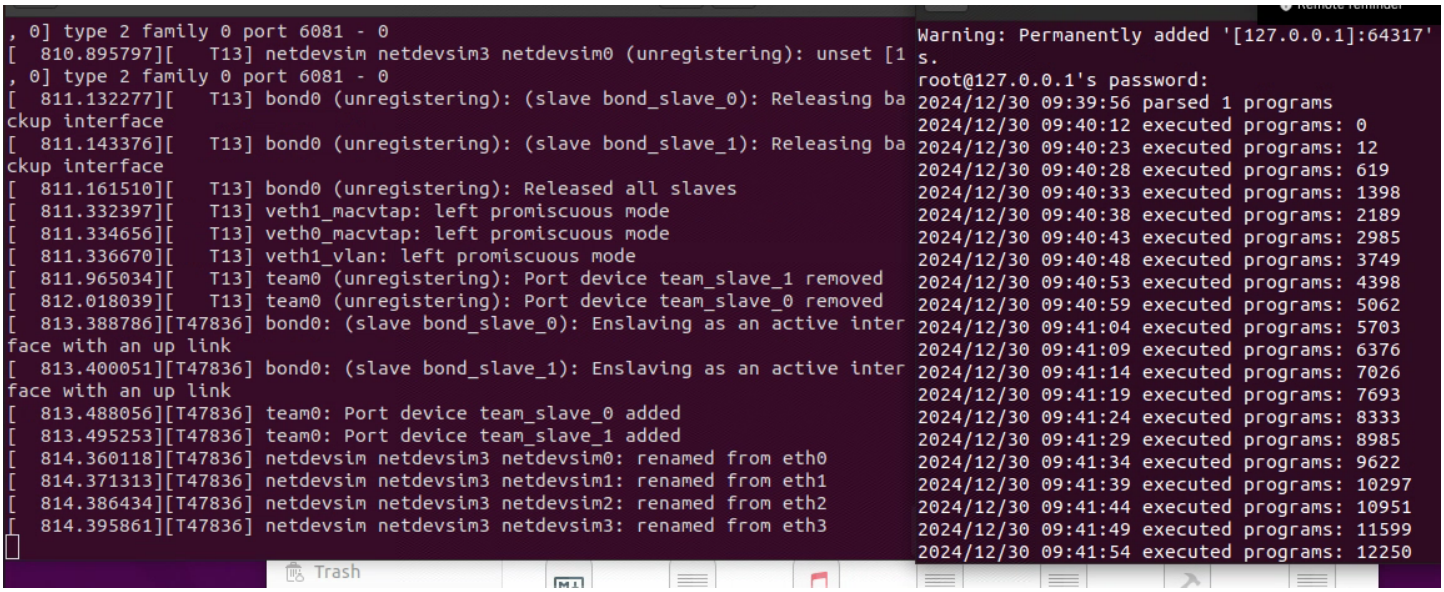
```
ssh -p 10022 -o UserKnownHostsFile=/dev/null \     -o StrictHostKeyChecking=no \     -o IdentitiesOnly=yes root@127.0.0.1 'chmod +x ./repro && ./repro'

# 在虚拟机中给予执行权限
chmod +x /root/repro.syz

# 在另一个终端中运行syz-execprog，推荐采用此项目，否则不便观察当前的运行次数
ssh -p 64317 -o UserKnownHostsFile=/dev/null  -o StrictHostKeyChecking=no -o IdentitiesOnly=yes root@127.0.0.1 './syz-execprog -enable=all -repeat=0 -procs=6 -cover=0 repro.syz'

# 在虚拟机中手动运行yz-execprog
sudo ./syz-execprog -executor=./syz-executor -repeat=16 -procs=16 -cover=0 repro.syz
```

3.7 `syz-execprog` 的参数说明：

```
-executor string: 指定执行程序的路径（你已经使用了 -executor=./syz-executor，这应该是正确的）。
```

```
-repeat int：指定重复执行次数（你传递了 -repeat=0，表示无限次执行）。
-procs int：指定并行进程数（你传递了 -procs=16）。
-cover int：是否收集覆盖率信息（你传递了 -cover=0，表示不收集）。
-sandbox string：指定沙箱模式（默认为 none，可以选择 setuid、namespace 或 android 等）。
```

成功开始执行后如图所示，左侧终端为虚拟机的串行启动框，右侧为另一个终端框，通过ssh调用虚拟机中的syz-executor进行复现。



末端跳转