Kernel Security: how2rootkit

Converting Text Programs into Executables

- What is a C/C++ compiler toolchain responsible for? Converting text (code) into an application that a CPU can run!
- This is accomplished in two main steps.
- **Compiling**
- **Linking**

Compiling: Seriously Oversimplified

- Converts source code (`.c`/`.cpp`) into object files (`.o`) containing machine code.
- The compiler performs various tasks:
    - Preprocessing (macros, includes)
    - Parsing and building an Abstract Syntax Tree
    - Generating machine code using its backend for AArch64
- The result of this stage: **Object files**

Linking: Seriously Oversimplified

- Once we have compiled object files, we need to **link** them together into an ELF executable.
- The linker resolves symbols (variables/functions) and stitches everything together:
  - References to undefined symbols are replaced with their correct addresses.
- We can share code in **libraries** to avoid duplication.
- Multiple ways to link against external code:

Linking

- **Static Linking**: External code gets included directly into your final executable.
    - Useful if you are unsure a needed library will be present on the target system.
    - Results in bigger binaries.
- **Dynamic Linking**: References to external libraries are stored symbolically in the binary.
    - At runtime, the loader (`ld.so`) loads these shared libraries.
    - Reduces binary size and promotes code reuse.

Shared Libraries (.so)

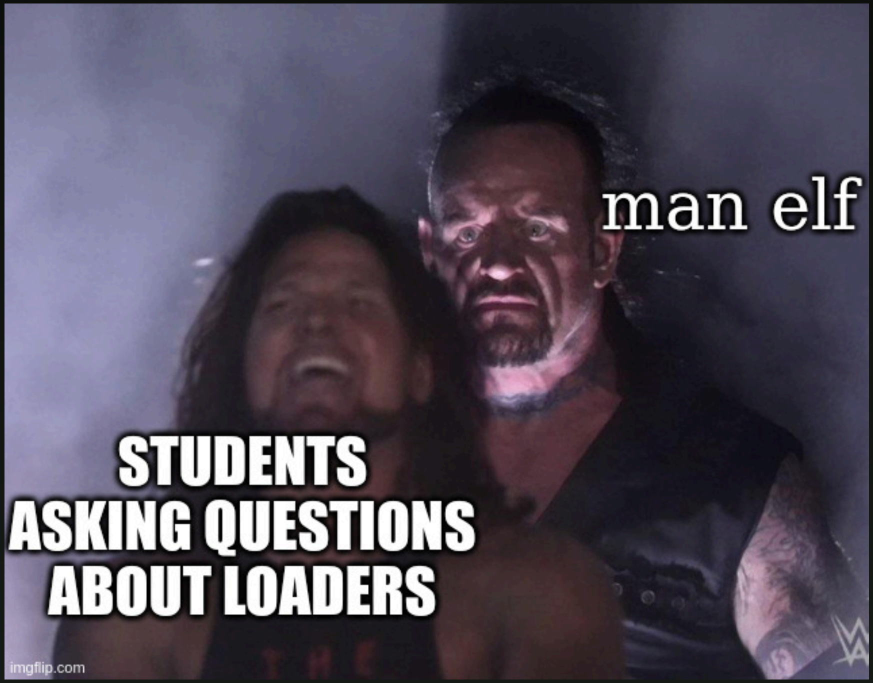
- In Linux, shared libraries are typically `.so` (shared object) files.
- They contain exported, callable functions loaded at runtime.
- Examples:
  - `libc.so`: Core C library for syscalls, memory management, etc.
  - `libm.so`: Math library.
  - Other specialized libraries: `libssl.so`, `libcrypto.so`, etc.

Dynamic Linking

- **Implicit Linking**
    - Your executable's ELF headers declare which `.so` libraries it depends on.
    - At load time, if the loader can't find them, the program can't start.
- **Explicit Linking**
    - Programs can manually load libraries at runtime with something like `dlopen()`.
    - If loading fails, the program can decide how to handle that gracefully.

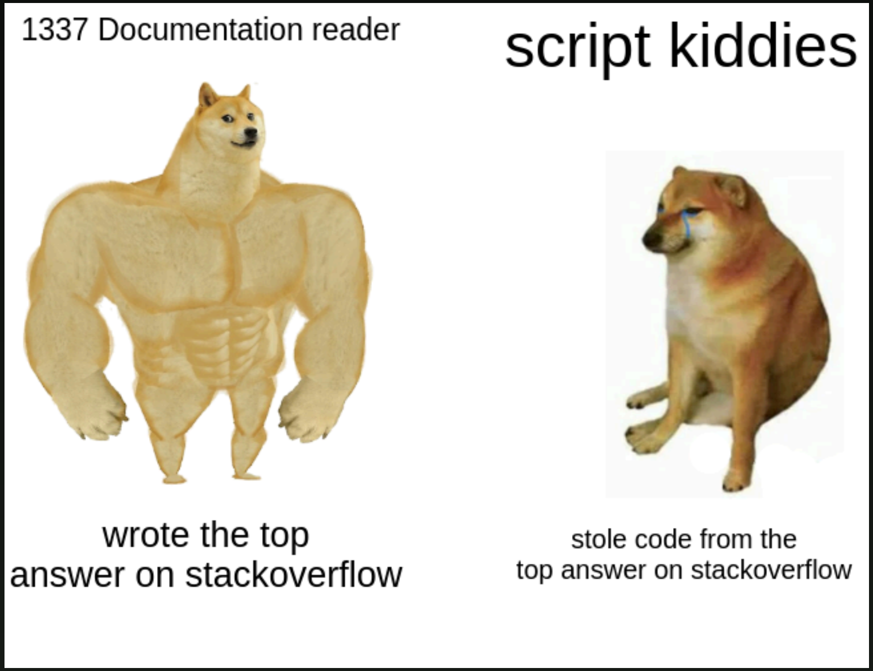
Reading the Docs

- “RTFM” (read the friendly manual) is vital for learning about Linux.
- Example: `man 2 open`, `man 3 printf`.
- to learn about the man pages,
  - `$ man man`



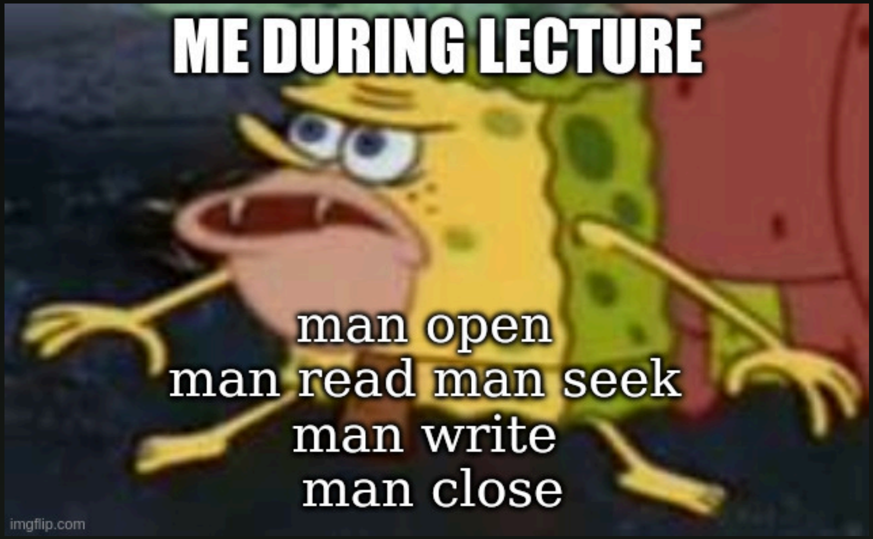
RTFM

- Debugging your code for 8 hours can save you 5 minutes of reading the docs
    - I myself, routinely don't read the documentation and suffer for it. Be better than me. Learn from my mistakes. RTFM


1337 Documentation reader
script kiddies
wrote the top answer on stackoverflow
stole code from the top answer on stackoverflow

Linux I/O on AArch64: Syscalls

- Focus: file I/O syscalls on AArch64
- I.e. "How do we read and filter through files"
- Use direct `syscall()`
- Explore kernel internals & structs
- Build efficient scanner for a 2GB file
- In-class assignment: structured raw file scan

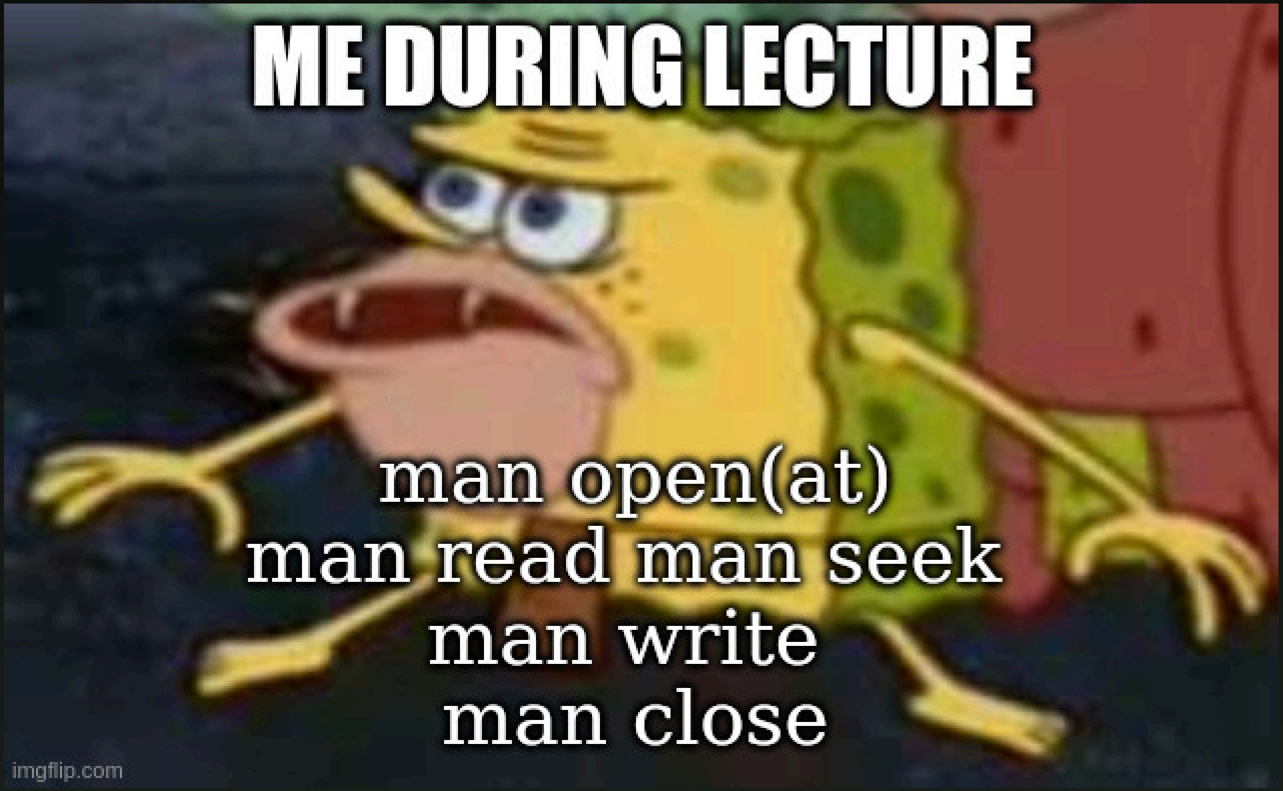


Finding Syscall values

```
cat /usr/include/asm-generic/unistd.h | grep openat
#define __NR_openat 56
__SYSCALL(__NR_openat, sys_openat)
#define __NR_openat2 437
__SYSCALL(__NR_openat2, sys_openat2)
```

```
grep -ho "__NR_[a-zA-Z0-9_]\+\s\+[0-9]\+" /usr/include/asm-generic/unistd.h | \
  sed 's/__NR_//' | column -t
```

Aarch64 IO


ME DURING LECTURE
man open(at)
man read man seek
man write
man close
imgflip.com

Syscall Interface (AArch64)

- Syscalls invoked with `svc #0`
- Registers:
  - `x8` = syscall number
  - `x0`–`x5` = up to 6 args
  - return value in `x0`
- Example:

```
int fd = syscall(SYS_openat, AT_FDCWD, "file.txt", O_RDONLY);
```

Kernel Objects Overview

```
task_struct
 ├─ files → files_struct
 │    └─ fd table → struct file *
 ├─ mm → mm_struct
 │    └─ vm_area_struct list/tree
```

Kernel Objects

- How to find kernel structs in linux
- `struct file` — https://github.com/torvalds/linux/blob/4ff71af020ae59ae2d83b174646fc2ad9f
- `struct file_operations` https://github.com/torvalds/linux/blob/4ff71af020ae59ae2d83b174646fc2ad9f
- `struct mm_struct` [https://github.com/torvalds/linux/blob/4ff71af020ae59ae2d83b174646fc2ad9 (https://github.com/torvalds/linux/blob/4ff71af020ae59ae2d83b174646fc2ad9
- `struct vm_area_struct` https://github.com/torvalds/linux/blob/4ff71af020ae59ae2d83b174646fc2ad9f

man openat

- Open file relative to directory (or AT_FDCWD for cwd)
    - Or create depending on arguments 0_ *
- On success, kernel creates struct file, updates task_struct->files
    - Inserts into fd table

Userland:

```
#include <syscall.h>
...
int fd = syscall(SYS_openat, AT_FDCWD, "/tmp/ch0nky.txt", O_RDONLY);
```

man read

- Reads bytes into user buffer
- Kernel uses page cache, `copy_to_user(...)`
- Updates `file->f_pos`

Userland:

```c
char buf[4096];
ssize_t n = syscall(SYS_read, fd, buf, sizeof(buf));
```

man write

- Copies data from user → kernel
- Updates page cache, marks pages dirty
- Advances `file->f_pos`
- Logically used to send data to an object manged by the kernel (file, pipe,..etc)

Userland:

```c
const char *msg = "Hello\n";
syscall(SYS_write, fd, msg, strlen(msg));
// example: writing data to stdout
syscall(SYS_write, 1, msg, strlen(msg));
```

stat / fstat

- Retrieves file metadata from inode
- No new file object created
- Useful for size, mode, timestamps

Userland:

```
struct stat st;
syscall(SYS_fstat, fd, &st);
printf("Size: %lld\n", (long long) st.st_size);
```

man `lseek`

- Moves file offset (file->f_pos)
- SEEK_SET, SEEK_CUR, SEEK_END
- Only for seekable fds

Userland:

```
off_t size = syscall(SYS_lseek, fd, 0, SEEK_END);
```

man `close`

- Releases fd from files_struct
- Decrements struct file refcount
- May free file object

Userland:

```
syscall(SYS_close, fd);
```

man pread / pwrite

- `pread(fd, buf, count, offset)`
- Reads from fd at offset
- Does not change file->f_pos
- Atomic (no race lseek+read)
- pwrite = write at offset
- Syscall: __NR_pread64 (AArch64 = 67)

Userland:

```
char buf[16];
ssize_t n = syscall(SYS_pread64, fd, buf, 16, 100);
```

man mmap

- Maps file region into process memory
- Creates new vm_area_struct in mm_struct
- Pages loaded lazily on fault

Userland:

```
char *map = syscall(SYS_mmap, NULL, size,
                    PROT_READ, MAP_PRIVATE, fd, 0);
```

man munmap

- munmap: removes VMA from mm_struct
- Kernel updates VMA flags + page tables

Userland:

```
syscall(SYS_munmap, map, size);
```

man mprotect

- mprotect: changes page protections
- Kernel updates VMA flags + page tables

Userland:

```
syscall(SYS_munmap, map, size);
```

Efficient Large File Scanning

Goal: find lines starting with "FLAG{" in 2GB file.

Steps:

- openat file
- fstat size
- mmap whole file (or chunked)
- Scan for prefix after newline/start
- munmap + close

Example Scanner

```c
char *data = mmap(NULL, size, PROT_READ, MAP_PRIVATE, fd, 0);
for (off_t i = 0; i < size; i++) {
    if (i == 0 || data[i-1] == '\n') {
        if (memcmp(&data[i], "FLAG{", 5) == 0) {
            // Found line
        }
    }
}
munmap(data, size);
```

Discussion

- Implant developer's perspective: what uses of file IO might we need?

Live demo

- Compiling

Rootkits and File IO

- Rootkits commonly want to "hide" artifacts associated with its existence
    - I.e. Hide files, hide directories
- Recall:
    - Userland Rootkit: inject payload into a process
        - hook common functions associated with action to detour
    - Kernel Land: inject/load into kernel
        - Somehow intercept responses to userland processes

Which Syscalls?

- Use `strace`

```
strace ls  /mnt/
execve("/usr/bin/ls", ["ls", "-la", "/mnt/"], 0xffffef43a910 /* 21 vars */) = 0
.....
statx(AT_FDCWD, "/mnt/", AT_STATX_SYNC_AS_STAT|AT_NO_AUTOMOUNT, STATX_MODE, {stx_mask=STATX_BA
openat(AT_FDCWD, "/mnt/", O_RDONLY|O_NONBLOCK|O_CLOEXEC|O_DIRECTORY) = 3
newfstatat(3, "", {st_mode=S_IFDIR|0755, st_size=4096, ...}, AT_EMPTY_PATH) = 0
getdents64(3, 0xaaab0db8fdf0 /* 3 entries */, 32768) = 80
getdents64(3, 0xaaab0db8fdf0 /* 0 entries */, 32768) = 0
close(3)                                = 0
newfstatat(1, "", {st_mode=S_IFCHR|0600, st_rdev=makedev(0xcc, 0x40), ...}, AT_EMPTY_PATH) = 0
ioctl(1, TCGETS, {c_iflag=ICRNL|IXON|IXOFF|IUTF8, c_oflag=NL0|CR0|TAB0|BS0|VT0|FF0|OPOST|ONLCR
write(1, "foobar\n", 7foobar
)                  = 7
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
```

- foo bar