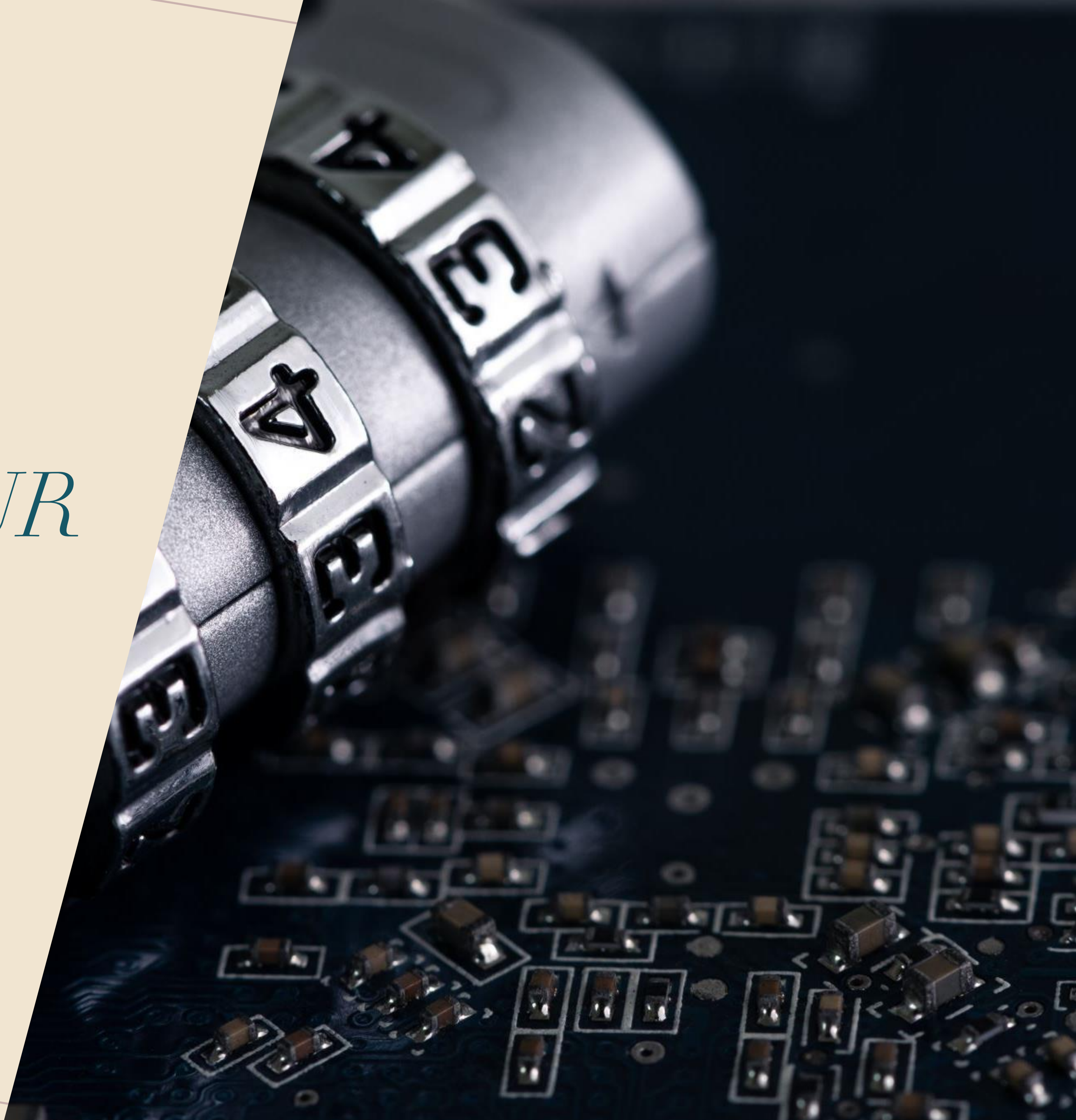
BYOR: BRING YOUR OWN ROOTKIT



Nischay Hegde
Malware Detection Researcher
SentinelOne

Agenda

- Introduction
- Why?
- Goals of this talk
- How?
- Make it
- Detect it

$whoami

- Malware Detection Researcher @ SentinelOne
- Ex-Security Researcher @ Uptycs
- Linux Security nerd
- Contributes to random FOSS projects when they have issues that aggravate me
- Helping fix bugs and push for a FOSS messenger (Matrix).

b10s
Meetups

What is a Rootkit?

- A rootkit is a piece of software that hides its existence and allows other (unprivileged) programs to execute as root.
- Essentially, a very specific type of backdoor
- Usually installed by escalating privileges and using various vulnerabilities.

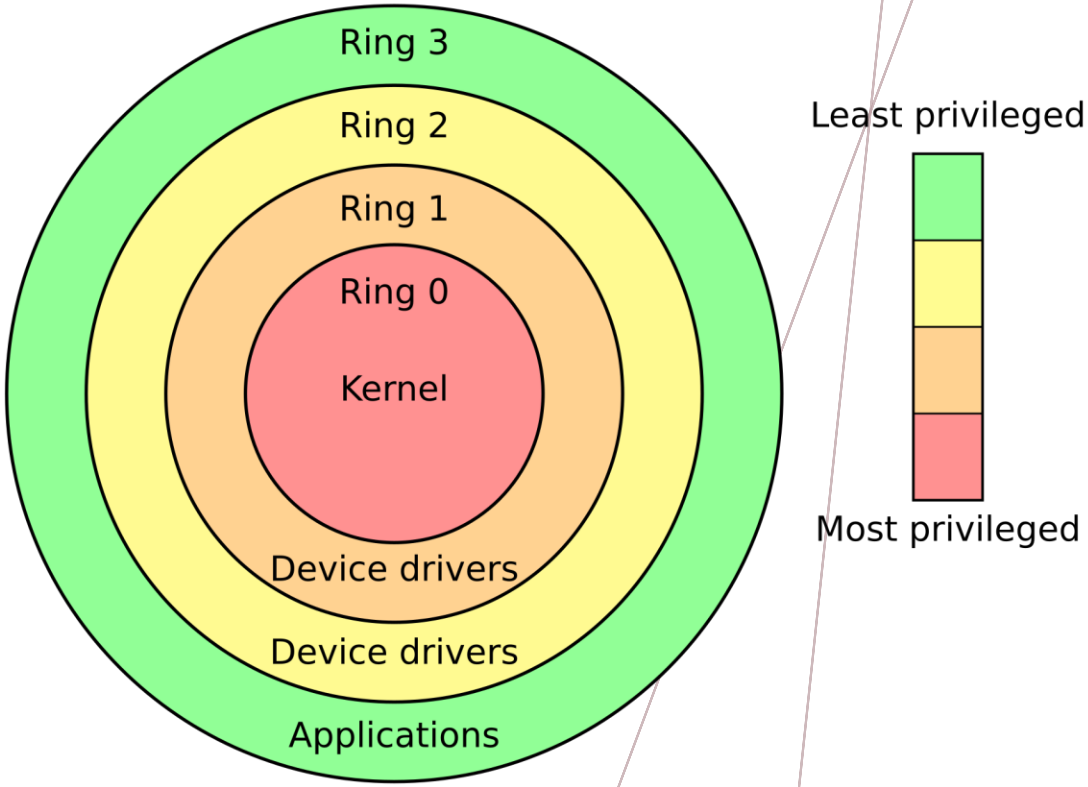
Why would you make your own?

- Because it’s interesting

b10s
Meetups

How?

- Rootkits are simply a subset of Linux Kernel Modules.
- LKMs are programs that get loaded as part of the kernel and run code in the kernel level.
- Can be listed with `lsmod`, inserted with `modprobe`, and removed by using `rmmod`.
- Therefore, we need to at least defeat these three to make it “invisible”.



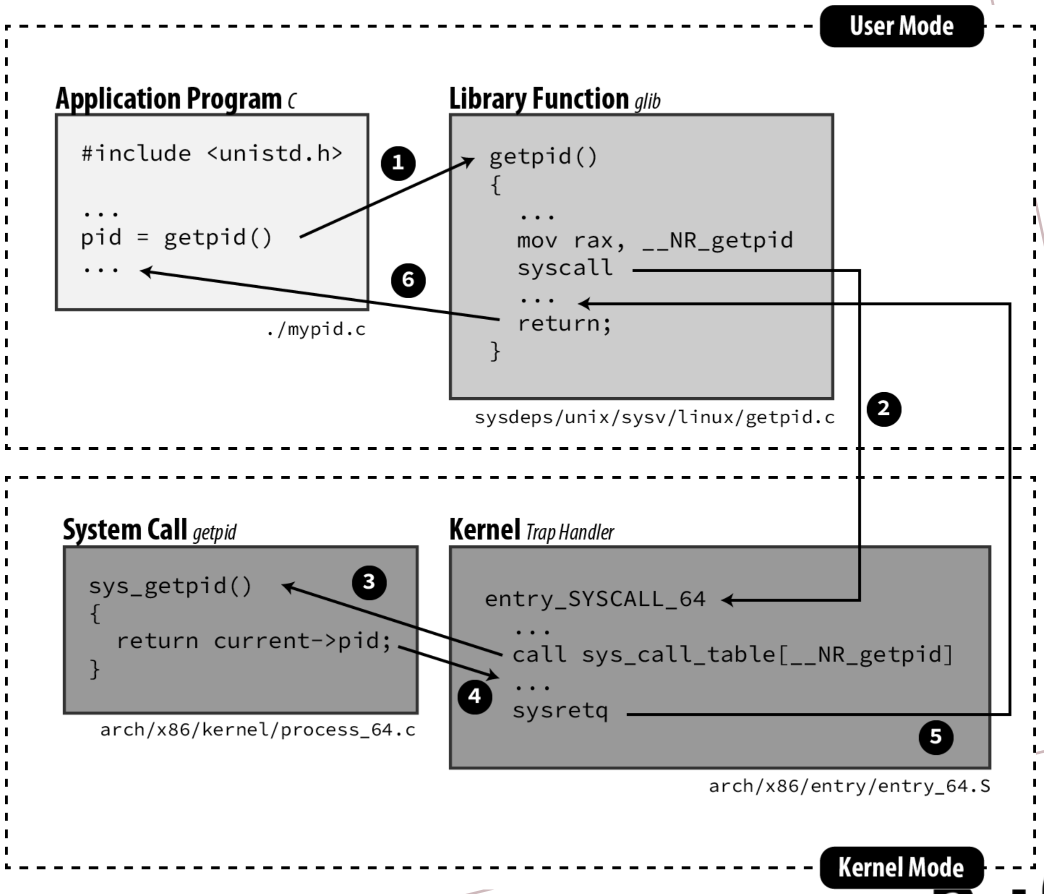
By Hertzsprung at English Wikipedia, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=8950144

Goals

- Write a kernel module that becomes invisible once you load it.
- Use it to do rootkit activities like giving root access to shells that do not have it.

Aside: How do syscalls work?

- Syscalls are an interface between the usermode and kernel-mode.
- When the syscall is called, the OS goes to the syscall table to find addresses associated with the syscall.
- In the syscall table, we redirect it to a function that does the required work.
- Therefore, our rootkit would have to hijack this process.



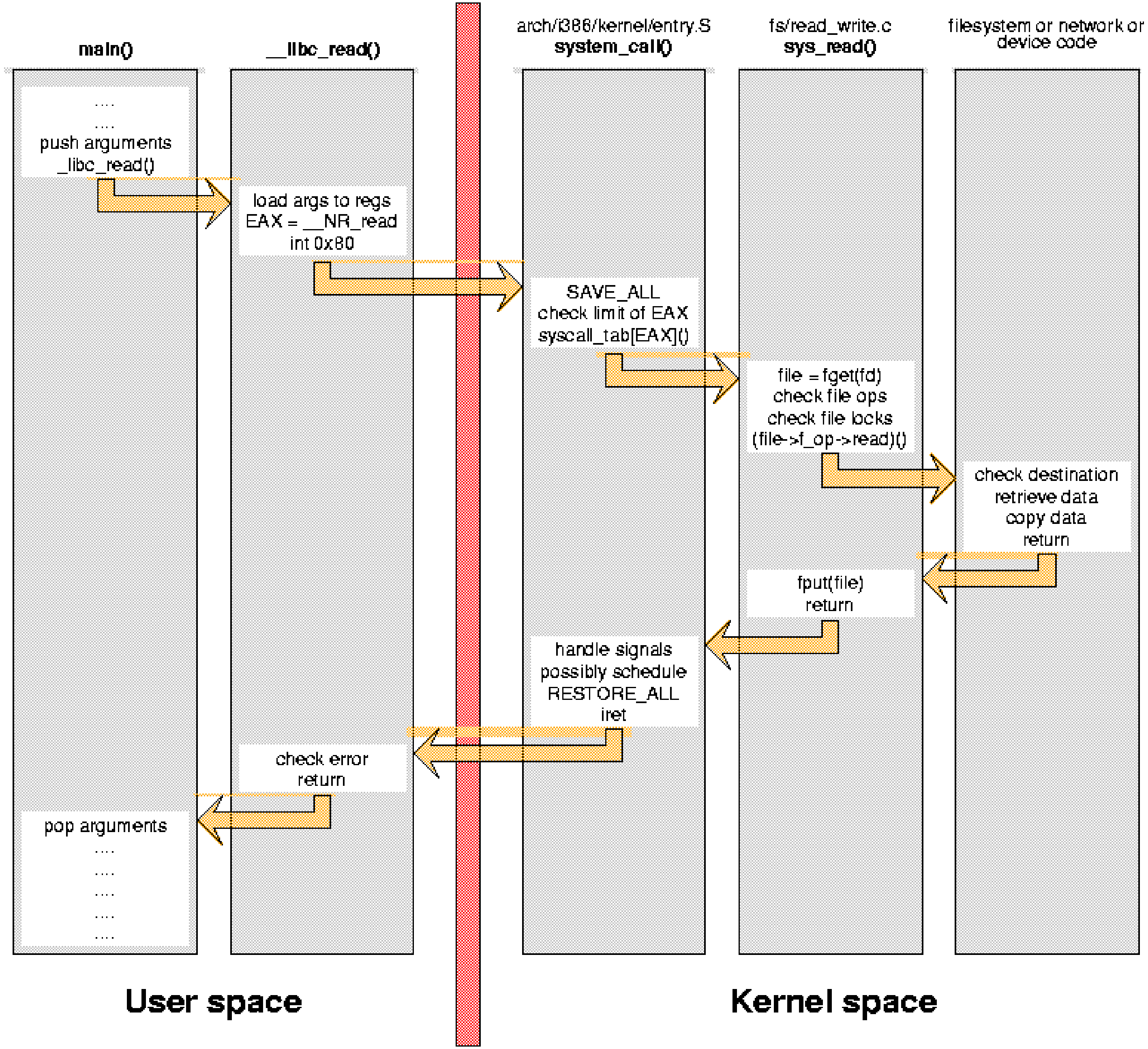
```
#include <unistd.h>

...
pid = getpid()
...
```

```
getpid()
{
  ...
  mov rax, __NR_getpid
  syscall
  ...
  return;
}
```

```
sys_getpid()
{
  return current->pid;
}
```

```
entry_SYSCALL_64
  ...
  call sys_call_table[__NR_getpid]
  ...
  sysretq
```

Aside: How do svscalls work?



b10s Meetups

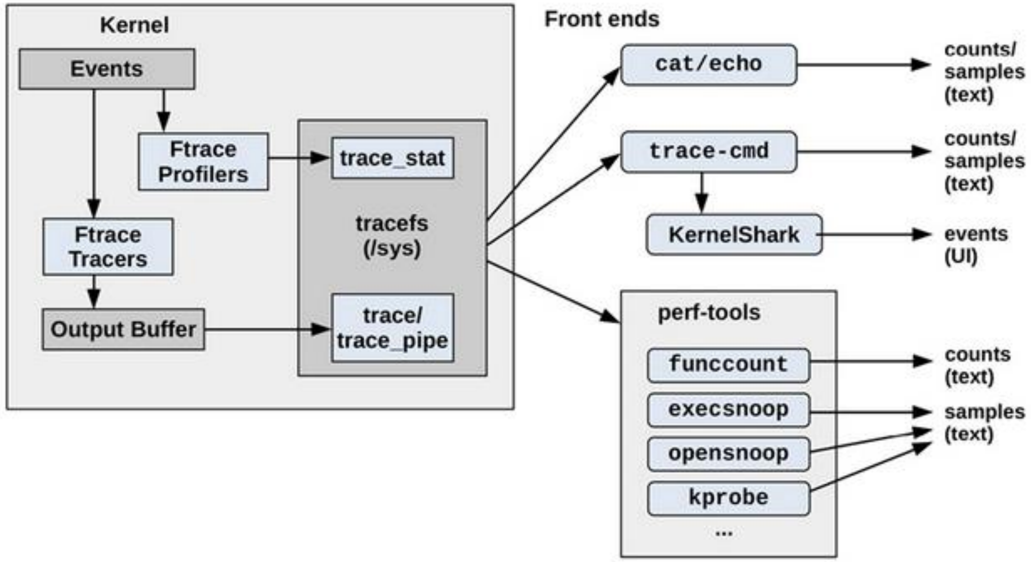
How would the rootkit work?

- Our hypothetical rootkit would need to modify (or “hook”) into certain syscalls, change their behaviour and then make it do the work that we need it to do.
- Two popular types of hooking:
  - Hook the syscall table with either the new function or an inline assembly that calls the new function
  - ftrace
  - eBPF
- We will be using ftrace because hooking syscall table is mostly useless now. (Linux 6.9+)

What is Ftrace?

- Linux kernel tracing mechanism

- Built into kernel, no extra installation necessary
- Can hook into various probes (i.e. places) in the kernel and outside (they're called kprobes, uprobes, tracepoints etc)



b10s Meetups

Aside: Linux kernel tracers

- Try to guess how many tracers exist.

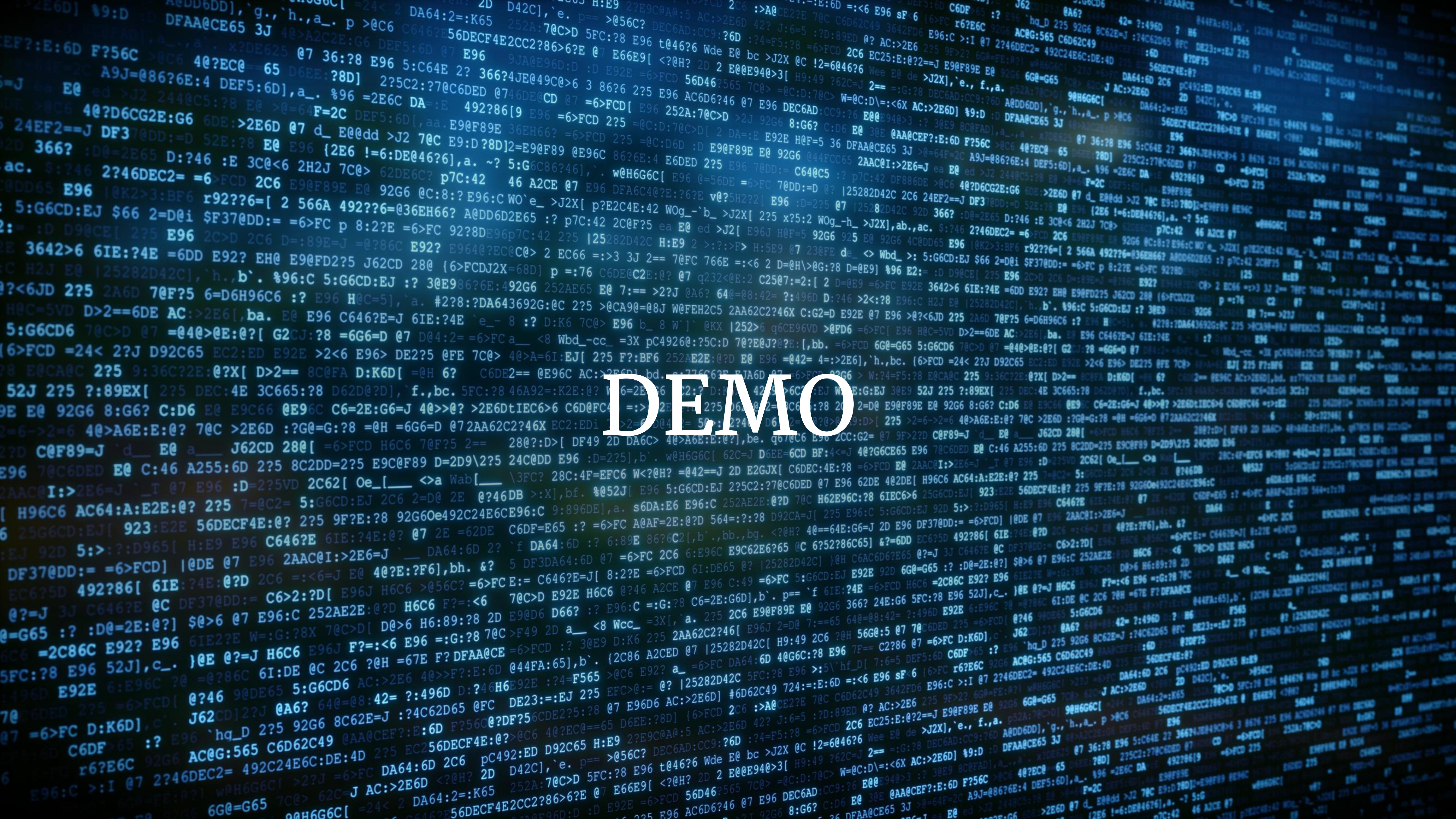
It’s 9!

- ftrace
- perf_events
- eBPF
- SystemTap
- LTTng
- ktap
- dtrace4linux
- OL DTrace
- sysdig

THIS IS FINE

Libraries that make it (slightly) easier

- https://github.com/milabs/khook
- https://github.com/WeiJiLab/kernel-inline-hook-framework
- Both are outdated (Linux 4.x mostly)



DEMO

Detection

- We will be using eBPF to detect this attack.
- Another kernel observability tool.
- Increasingly being used by EDRs because it's less code on the kernel than kernel modules.
- Has a high level language to write code in (bpftrace) as well as a lower-level libraries for C and Python (BCC).

References

- https://www.brendangregg.com/blog/2015-07-08/choosing-a-linux-tracer.html
- https://github.com/xcellerator/linux_kernel_hacking
  - XCellerator's blog is XCellent as well
- https://github.com/iovisor/bcc/blob/master/examples/tracing/stacksnoop.py - stacksnoop

THANK YOU

Nischay Hegde

mailto: contact@nischay.me

Mastodon: @thatloststudent@infosec.exchange

Twitter: @thatloststudent