EPF: Evil Packet Filter

Di Jin Vaggelis Atlidakis Vasileios P. Kemerlis

Secure Systems Laboratory (SSL)
Department of Computer Science
Brown University

Overview

- Background
  - Kernel security and exploitation
  - Berkeley Packet Filter (BPF)
- EPF attacks
  - BPF-Reuse
  - BPF-ROP
- EPF defenses
- Evaluation

BROWN

Kernel Security

OS kernels are central to modern systems

- Providing abstractions
  - Virtual memory
  - File systems
  - Namespaces
  - ...
- Managing devices
  - CPUs, GPUs, RAM, ...
  - SSDs, HDDs, ...
  - NICs, ...
- ...

BROWN

Kernel Security

OS kernels are central to modern systems

- Providing abstractions
  - Virtual memory
  - File systems
  - Namespaces
  - ...
- Managing devices
  - CPUs, GPUs, RAM, ...
  - SSDs, HDDs, ...
  - NICs, ...
- ...

- Privilege management, access control, and policy enforcement

BROWN

Kernel Security (cont'd)

By breaking kernel security, the attacker can

- ✘ Compromise other users on the system
- ✘ Gain control over physical devices
- ✘ Bypass protection mechanisms and policy enforcement

Kernel Security (cont'd)

Linux kernel ➜ More than 25 MLoC

- Hundreds of security vulnerabilities every year
  - Syzkaller found 3736 bugs in 2017–2020[1]
  - Bugs take on average 66 days to be fixed[2]

[1] *An In-depth Analysis of Duplicated Linux Kernel Bug Reports.* NDSS 2022
[2] *Undo Workarounds for Kernel Bugs.* USENIX SEC 2021

BROWN

Kernel Exploitation

- Attacker
  - Unprivileged user ➜ Privilege escalation
- Kernel
  - Memory error(s)➜ Buffer overflow, use-after-free, ...

[3] *kGuard: Lightweight Kernel Prot. agst. Return-to-User Attacks.* USENIX SEC 2012

BROWN

Kernel Exploitation

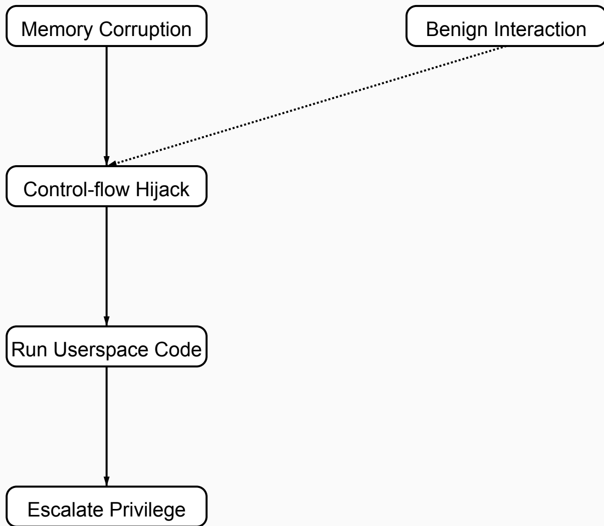
- Attacker
  - Unprivileged user ➜ Privilege escalation
- Kernel
  - Memory error(s)➜ Buffer overflow, use-after-free, ...

Traditionally, attackers employed a strategy called `ret2usr`[3]

[3] *kGuard: Lightweight Kernel Prot. agst. Return-to-User Attacks.* USENIX SEC 2012

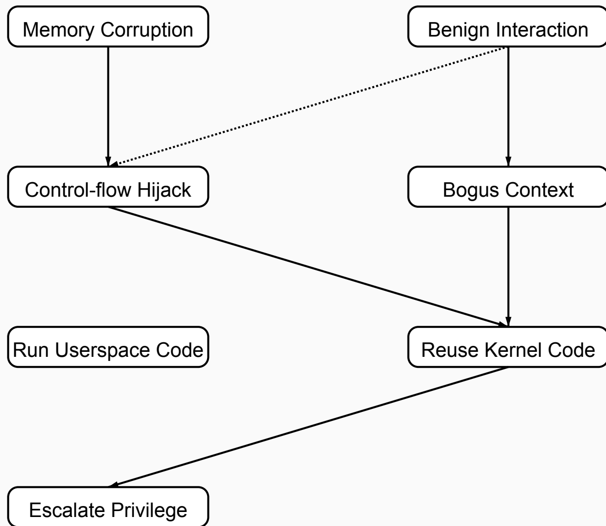
BROWN

Kernel Exploitation (cont'd)



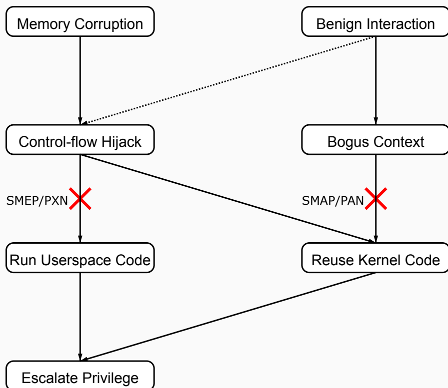
BROWN

Kernel Exploitation (cont'd)



ret2usr Defenses

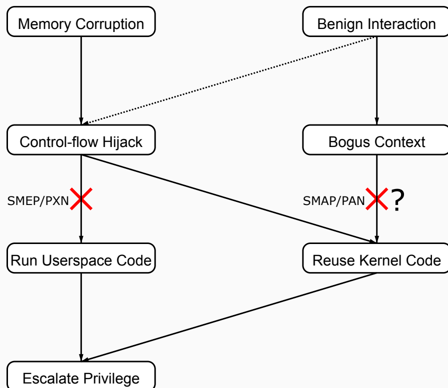
- SMEP/PXN
  - ✔ Prevents control-flow from being redirected to userspace code
- SMAP/PAN
  - ✔ Prevents unintended access of userspace data



BROWN

ret2usr Defenses

- SMEP/PXN
  - ✔ Prevents control-flow from being redirected to userspace code
- SMAP/PAN
  - ✔ Prevents unintended access of userspace data



BROWN

Kernel Exploitation (cont'd)

Bypassing `ret2usr` defenses

- ✖ `ret2dir` ➜ Implicit memory sharing btw. kernel and user space[4]
- ✖ Control register(s) overwrite ➜ Early stage(s) of the exploit[5]
  - Vulnerability and implementation dependent
  - `native_write_cr4` ➜ Disable SMEP and SMAP

[4] *ret2dir: Rethinking Kernel Isolation.* USENIX SEC 2014

[5] *FUZE: Towards Facilitating Exploit Generation for Kernel Use-After-Free Vuln.* USENIX SEC 2018

BROWN

Berkeley Packet Filter (BPF)

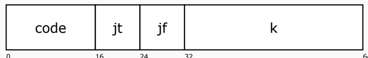
- User apps ➜ Safely delegate computations to kernel[6]
- A small virtual architecture with a RISC-like instruction set
- Many applications
  - Packet filtering
  - Network routing (Cilium)
  - System call filtering (Android, Docker, Chrome, OpenSSH, Tor, ...)
  - Kernel profiling
  - FUSE (Filesystem in Userspace)[7]
  - High-performance storage[8]
  - ...

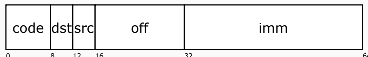
[6] *The BSD Pkt. Filter: A New Arch. for User-level Pkt. Capture.* USENIX Winter 1993
[7] *Extension Framework for File Systems in User space.* USENIX ATC 2019
[8] *XRP: In-Kernel Storage Functions with eBPF.* OSDI 2022

BROWN

Berkeley Packet Filter (cont'd)

- Classic BPF (cBPF)
  - Older, legacy ISA
  - Always available
- Extended BPF (eBPF)
  - Newer, modern ISA
  - Additional functionality
    1. Maps
    2. Helpers
  - Actively developed





Berkeley Packet Filter (cont'd)

- Instructions ➜ ALU, memory access, conditional branching, **...**
- Safety ➜ Statically verified (termination, pointer-access safety, **...**)
  - Conservative pointer and value-range analysis
- Runtime ➜ Interpreted and JIT-compiled execution
  - cBPF is translated to eBPF internally

✪ EPF Attacks

Threat Model

Unprivileged attacker

- ✖ Execute userland programs
- ✖ Access any (pseudo-)file to which they have access
- ✖ Trigger memory error(s) in kernel code

Threat Model

Unprivileged attacker

- ✖ Execute userland programs
- ✖ Access any (pseudo-)file to which they have access
- ✖ Trigger memory error(s) in kernel code

OS Kernel

- ✔ cBPF is enabled
- ✔ SMEP/PXN and SMAP/PAN are enabled
- ✔ W^X, KASLR, ... (standard, deployed protection mechanisms)

Weak Kernel-Userland Isolation

BPF programs ➜ Great for injecting code-reuse payloads

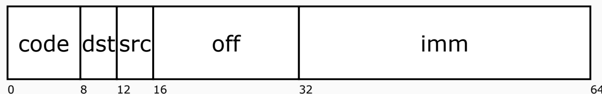
Weak Kernel-Userland Isolation

BPF programs ➜ Great for injecting code-reuse payloads

- ✖ Allow a user to push huge amounts of content into kernel space
- ✖ Relative freedom on the injected content
- ✖ "Live" inside the kernel and are "consumed" by the kernel
  - Naturally bypasses existing isolation mechanisms (including any defense for `ret2dir`)

Challenges

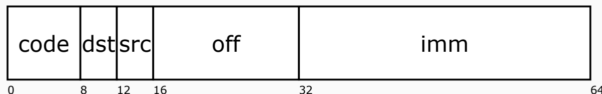
- BPF instruction-space is sparse ➜ Control every other four bytes
- BPF programs are read-only in memory



Challenges

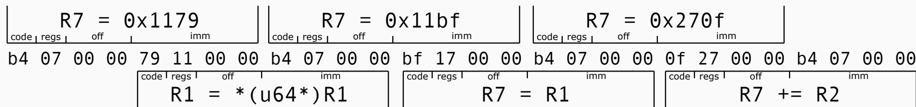
- BPF instruction-space is sparse ➜ Control every other four bytes
- BPF programs are read-only in memory

If this is our code-reuse payload, *what* can we reuse?



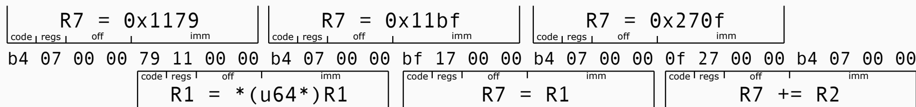
✪ BPF-Reuse Attack (EPF v1)

- BPF programs ➜ Verified only when loaded into the kernel
- BPF code ➜ Semantically different when starting from an offset



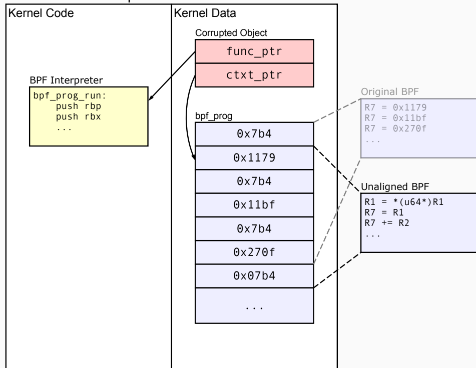
✪ BPF-Reuse Attack (EPF v1)

- BPF programs ➜ Verified only when loaded into the kernel
- BPF code ➜ Semantically different when starting from an offset



➥ Embed an unsafe BPF program inside a safe one

✪ BPF-Reuse Attack (cont'd)



- Target ➜ BPF interpreter
- Payload ➜ Unalign. BPF code

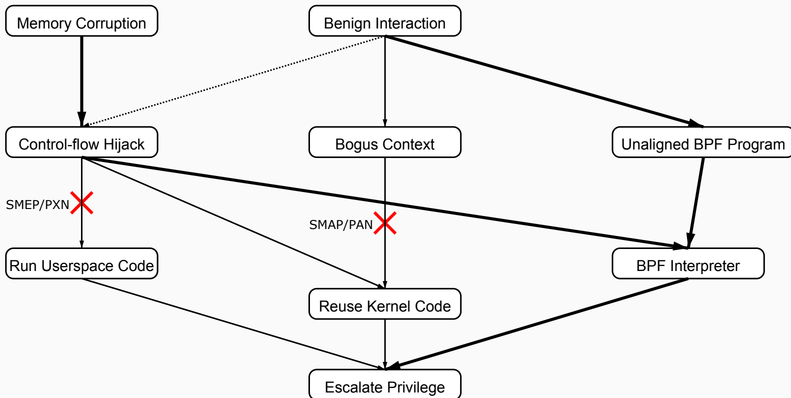
✪ BPF-Reuse Attack (cont'd)

BPF-Reuse (EPF v1) expressiveness

- ✔ Arithmetics between registers
- ✔ Memory load/store using registers
- ✔ Conditional branch and loop
- ✔ Load arbitrary imm. values into registers (with a bit of effort)

✪ BPF-Reuse Attack (cont'd)

- BPF interpreter ➜ Turing-complete computations, arb. `R/W`
  - ✔ Overwrite the credentials of the attacker's proc. ➜ Escalate priv.



BROWN

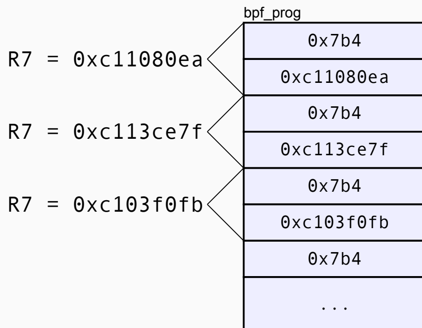
✪ BPF-ROP Attack (EPF v2)

On 32-bit arch. BPF code can encode ROP (or code-reuse) payloads

✪ BPF-ROP Attack (EPF v2)

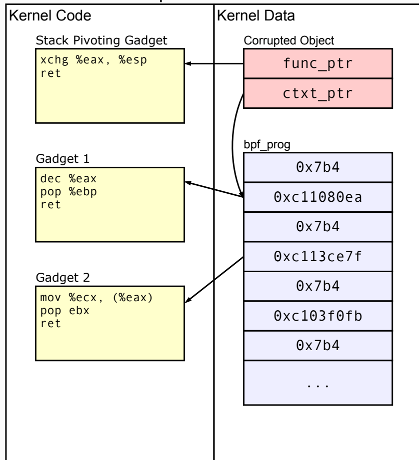
On 32-bit arch. BPF code can encode ROP (or code-reuse) payloads

1. Use special code gadgets that work with the restrictions
   - …; `ret` ➜ …; `pop` …; `ret`
2. Bootstrap a stage-2 ROP-payload execution in writable mem.



BROWN

✪ BPF-ROP Attack (cont'd)



- Target ➜ Code gadgets
  - …;`ret` ➜ …;`pop` …;`ret`
- Payload ➜ Fake stack
  - Encoded using BPF code

BROWN

✪ EPF Defenses

EPF Defenses

Runtime isolation between BPF and regular kernel data

- ✔ **BPF-ISR** ➜ Randomize the BPF prog. in-memory representation
- ✔ **BPF-NX** ➜ Designated region for BPF instructions
  - Instruction fetches are limited to allowed region(s) only
- ✔ **BPF-CFI** ➜ BPF program exec. starts from the “beginning” only

EPF Defenses

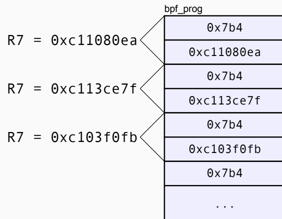
- Prevent mismatches between mem. accesses and data types

Access Type \ Data Type	Regular Data	BPF Programs	
		Aligned	Unaligned
Data Access	Allowed	BPF-ISR	BPF-ISR
BPF Interpretation	BPF-NX	Allowed	BPF-CFI

BROWN

✪ BPF-ISR

- Adaptation of the ISR (instruction-set randomization)[9] concept
  - ✖ BPF-ROP (EPF v2)

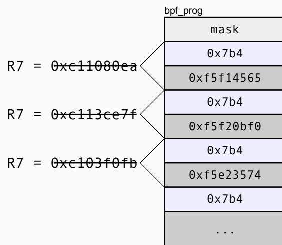


- ■ Random mask per BPF program
- ✔ Mask instructions (load time)
- ✔ Unmask during interpretation

[9] *Countering Code-Injection Attacks With Instruction-Set Randomization. ACM CCS 2003*

✪ BPF-ISR

- Adaptation of the ISR (instruction-set randomization)[9] concept
  - ✖ BPF-ROP (EPF v2)

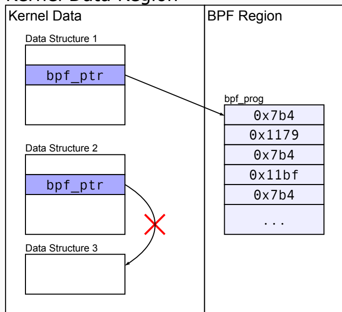


- ■ Random mask per BPF program
- ✔ Mask instructions (load time)
- ✔ Unmask during interpretation

[9] *Countering Code-Injection Attacks With Instruction-Set Randomization. ACM CCS 2003*

✪ BPF-NX

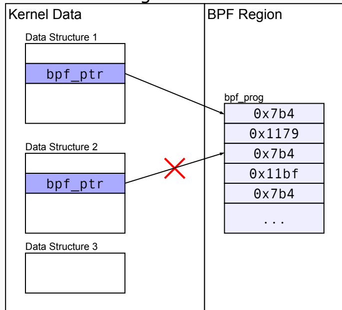
- Inspired by the W^X policy



- ✔ Sub-address space for BPF code
- ✔ Address-range check on instruction fetches (during interpretation)

✪ BPF-CFI

- Based on the CFI (control-flow integrity)[10] concept
  - ✖ BPF-Reuse (EPF v1)



- ✔ Entry-point check (during interp.)
- ✔ Sentinel variable for protecting the check ➜ IRM (inlined ref. monitor)

[10] *Control-Flow Integrity.* ACM CCS 2005

BROWN

Evaluation

✪ EPF Effectiveness

- Bypass `ret2usr` defenses ➜ Escalate privilege

CVE	Vulnerability Type	Method
CVE-2021-43267	Heap overflow	BPF-Reuse
CVE-2017-7308	Heap overflow	BPF-Reuse
CVE-2016-8655	Use-after-free	BPF-Reuse
CVE-2017-7308	Heap overflow	BPF-ROP
CVE-2017-6074	Use-after-free	BPF-ROP
CVE-2016-8655	Use-after-free	BPF-ROP
CVE-2013-2094	Arbitrary write	BPF-ROP

✪ EPF Effectiveness (cont'd)

- Payload injection using different system calls
  - `setsockopt` and `seccomp` represent BPF spraying

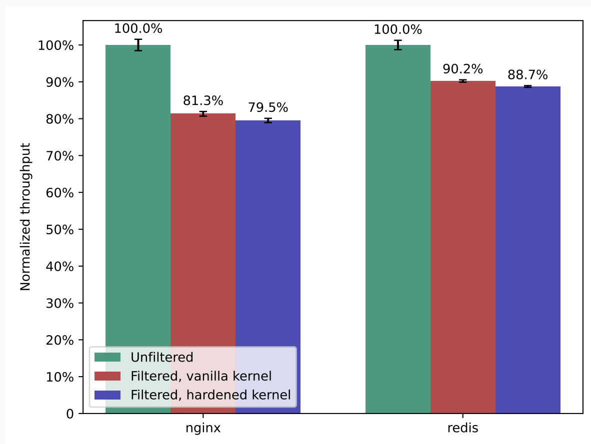
Sprayed memory size (in GB)

RAM size (in GB)

setsockopt
pipe
seccomp
socket (tcp)
socket (udp)
semget
msgsnd
mq
add_key
readv

✪ BPF-{ISR, NX, CFI} Performance

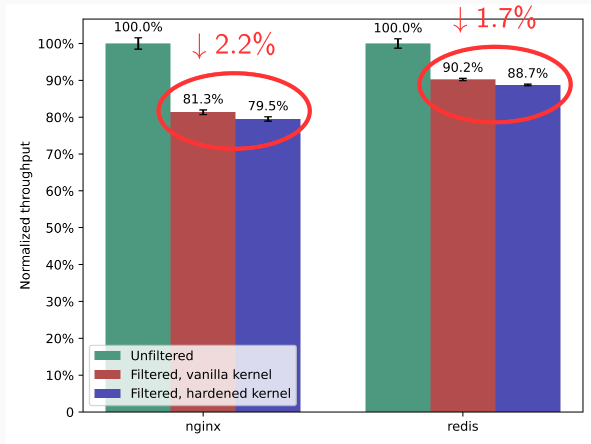
- Impact on system call filtering (`seccomp-BPF`)
  - System call set(s) extracted with `sysfilter`[11]



[11] *sysfilter: Automated System Call Filtering for Commodity Software.* RAID 2020

✪ BPF-{ISR, NX, CFI} Performance

- Impact on system call filtering (seccomp-BPF)
  - System call set(s) extracted with sysfilter[11]



BROWN

[11] *sysfilter: Automated System Call Filtering for Commodity Software.* RAID 2020

✪ BPF-{ISR, NX, CFI} Performance (cont'd)

- Impact on packet filtering (`setsockopt`)

Raw Socket Filter	Vanilla Kernel	Hardened Kernel
bpf1	0.625	0.611
bpf2	0.580	0.568
bpf3	0.528	0.517
bpf4	0.520	0.509
bpf5	0.439	0.425
bpf6	0.330	0.326

Throughput (Million Packets per Second)

✪ BPF-{ISR, NX, CFI} Performance (cont'd)

- Impact on packet filtering (`setsockopt`)

Raw Socket Filter	Vanilla Kernel	Hardened Kernel
bpf1	0.625	0.611
bpf2	0.580	0.568
bpf3	0.528	0.517
bpf4	0.520	0.509
bpf5	0.439	0.425
bpf6	0.330	0.326

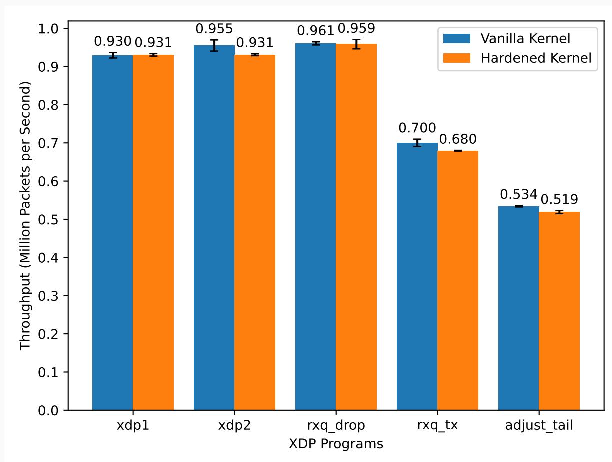
Throughput (Million Packets per Second)

↓ 1.2–3.2%

BROWN

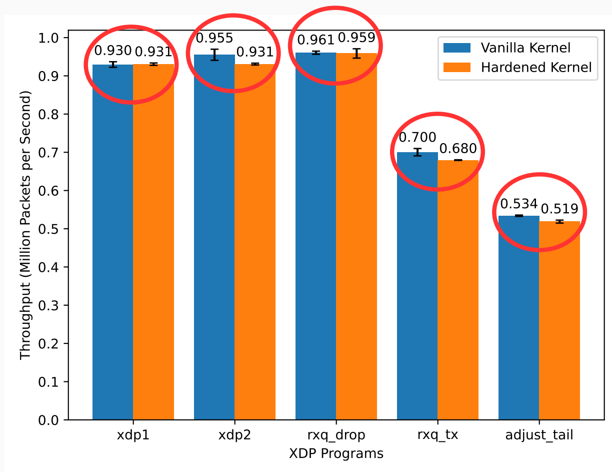
✪ BPF-{ISR, NX, CFI} Performance (cont'd)

- Impact on XDP (eXpress Data Path)[12]



[12] *The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel.* CoNEXT 2018

- Impact on XDP (eXpress Data Path)[12]



↓ 0–2.9%

[12] *The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel.* CoNEXT 2018

Conclusion

- Two attacks (ab)using BPF (interpreter-focused)
  - BPF-Reuse (EPF v1), BPF-ROP (EPF v2)
  - Applied on real-world vulnerabilities
- Three defenses against EPF
  - BPF-ISR, BPF-NX, BPF-CFI
  - Negligible/Moderate performance overhead

✪ **https://gitlab.com/brown-ssl/epf**

Backup Slides

BROWN

Arbitrary Immediates (BPF-Reuse)

- Load the value 11 (1011 in binary) onto register R0
  - Register-only BPF instructions

```
R0 = R0 ^ R0    /* R0 =    {0} */
R1 = R1 / R1    /* R1 =    {1} */
R0 += R1        /* R0 =    {1} */
R0 += R0        /* R0 =   {10} */
R0 += R0        /* R0 =  {100} */
R0 += R1        /* R0 =  {101} */
R0 += R0        /* R0 = {1010} */
R0 += R1        /* R0 = {1011} */
```

Testbed

- CPU ➜ 16-core 3.7GHz Intel Xeon W-2145
- RAM ➜ 64GB
- Kernel ➜ Linux v5.10
- Userland ➜ Ubuntu 18.04 LTS

BROWN

EPF Defenses (cont'd)

```
1  u64 bpf_interpreter(struct bpf_prog *prog)
2  {
3      ...
4      enter_bpf_mode();
5      check_bpf_cfi(prog);
6      initialize_context();
7      mask = prog->mask;
8      ...
9      insn = prog->insns;
10 select_insn:
11     tmp_insn = *insn;
12     check_bpf_nx(insn);
13     check_bpf_mode();
14     tmp_insn = unmask(tmp_insn, mask);
15     execute_bpf_insn(tmp_insn);
16     if (finished) { goto done; }
17     else { insn++; goto select_insn; }
18 done:
19     leave_bpf_mode();
20     return result;
21 }
```