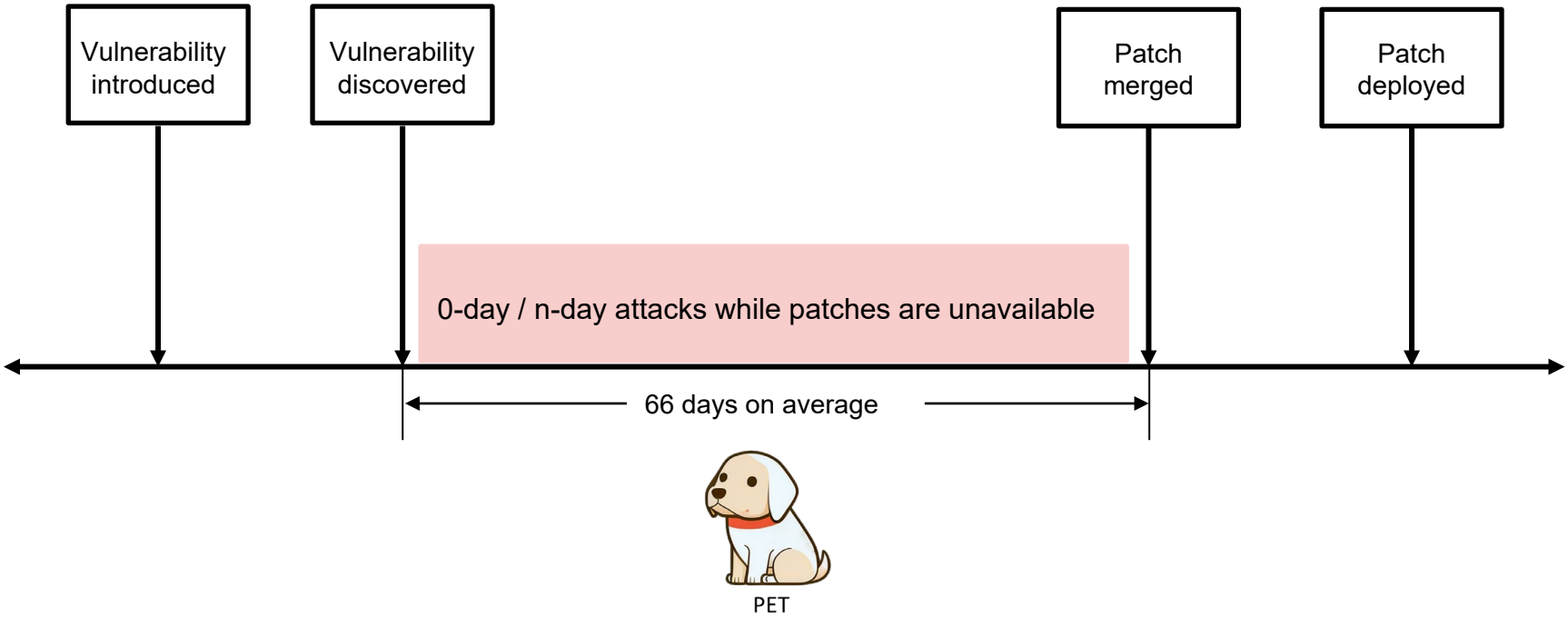
南京大学 NANJING UNIVERSITY

University of Colorado Boulder

PET: Prevent Discovered Errors from Being Triggered in the Linux Kernel

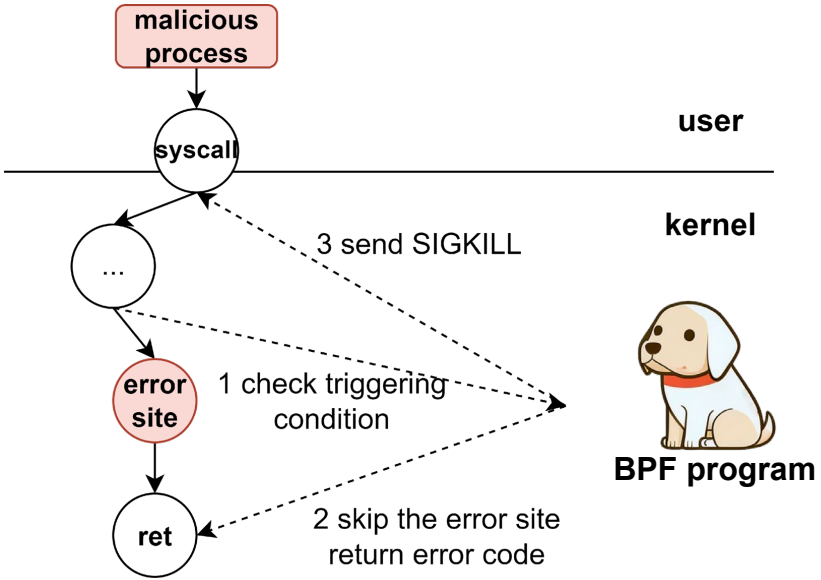
Zicheng Wang, Yueqi Chen, Qingkai Zeng

Motivation: Protect Kernel before patches are available



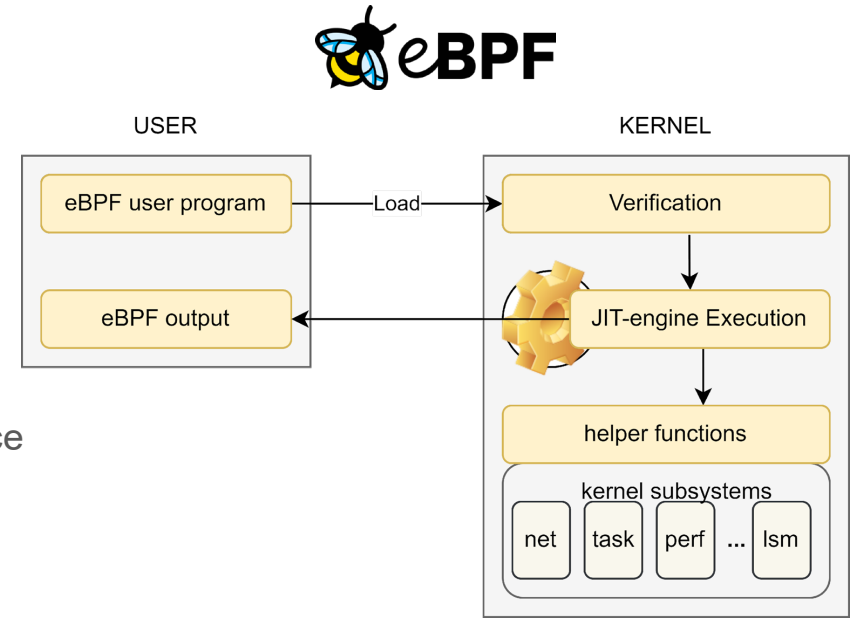
Key Idea: Prevent Vulnerabilities From Being Triggered

- Take **Sanitizer report** as input, and generate an **eBPF program**
- Check if error triggering condition is met right before the error site
- Skip the error site if condition is met



Background: eBPF in-kernel Virtual Machine

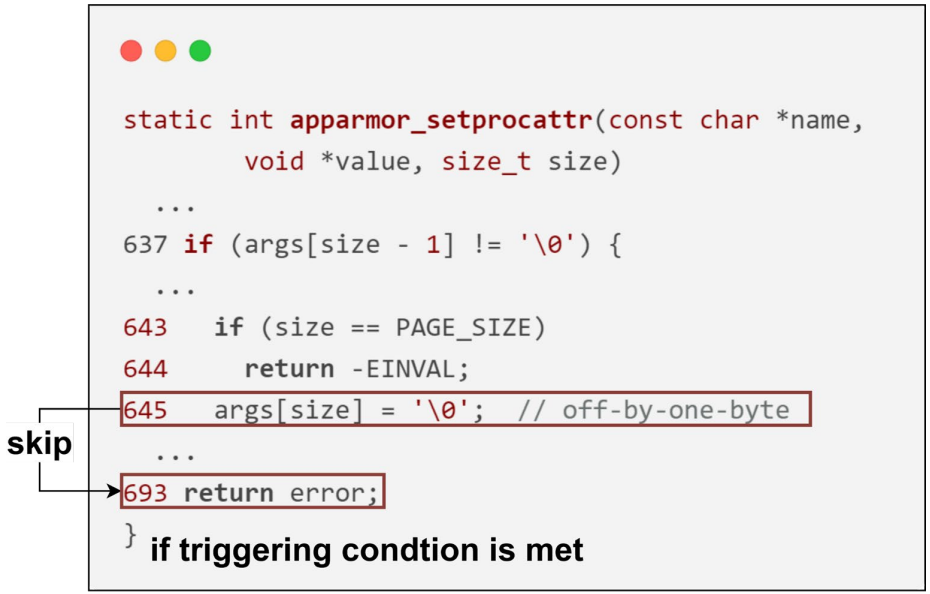
- in-kernel virtual machine that safely executes programs from user space
  - **Safety**: a verifier to ensure memory safety, termination, information flow security
  - **Efficiency**: a JIT-engine to execute BPF bytecode, achieving native machine performance
  - **Expressiveness**: a set of helper functions as interfaces between eBPF programs and other kernel subsystems



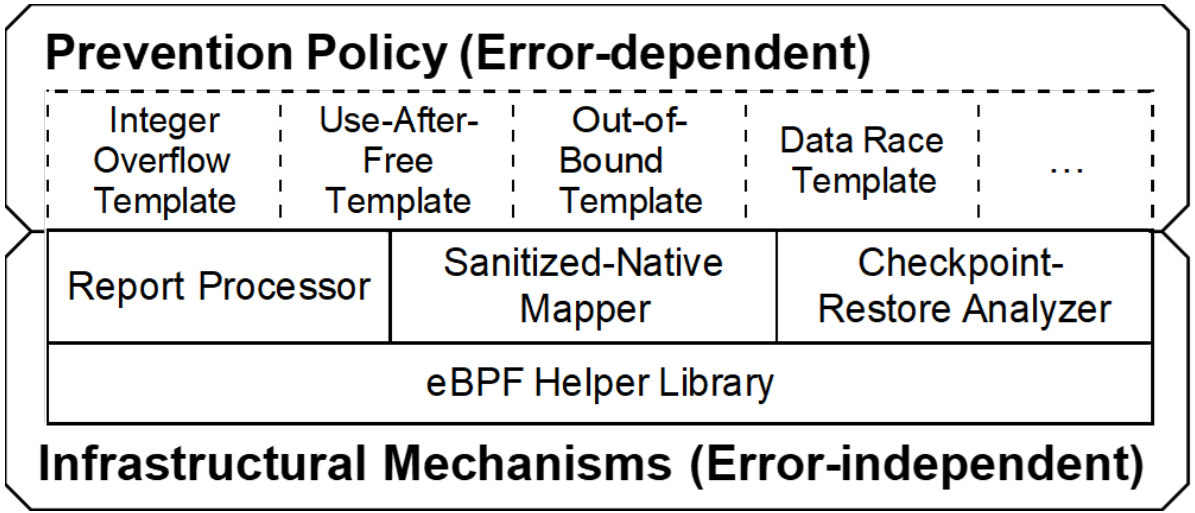
- **an eBPF program can be attached to arbitrary error site in kernel**

Example: CVE-2016-6187

- **Error site**: At line 645, length of `args[]` is `size` while `args[size]` is written
- **Triggering condition**: `args[size]` is out of the boundary of `args[]` at line 645
- **Prevention**: skip line 645 and jump to line 693 to return `-EINVAL`



Overview: PET Framework



Mechanisms: Report processor & Sanitized-Native Mapper

error site in the sanitized image

sanitized inst -> statement-> native inst

sanitized reg -> var -> native reg

source code

```
645     args[size] = $0x0;
```

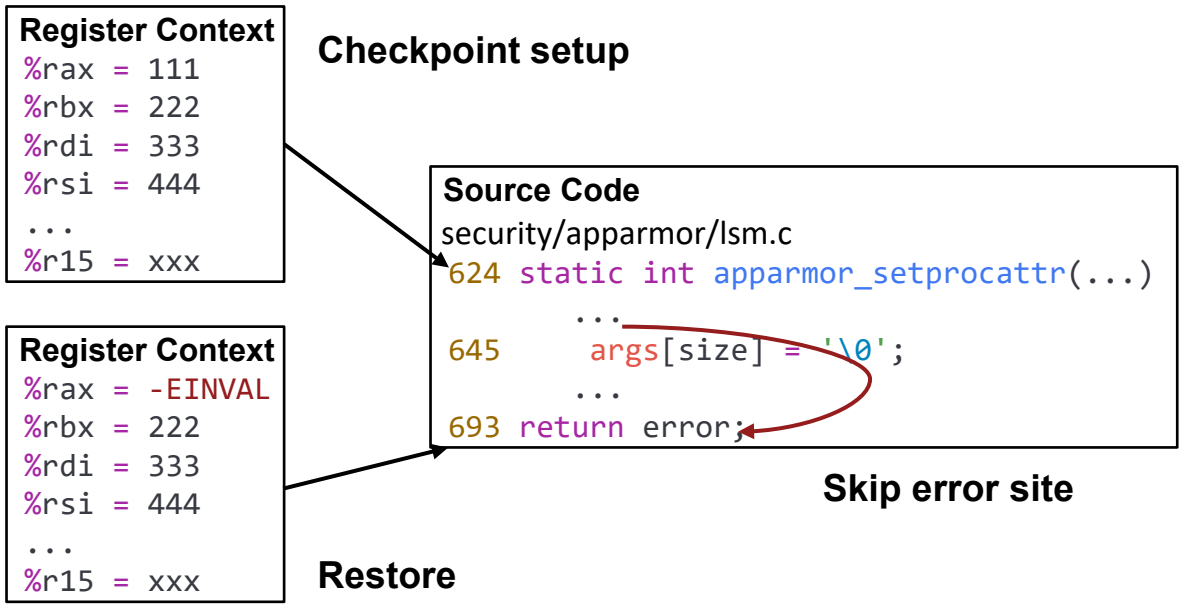
error site at the source code level

native binary

```
movb   $0x0, (%rsi, %rdx, 1)
```

error site in the native image

Mechanisms: Checkpoints & Restore



paired operations

```
int func(...)
{
    spin_lock(lock);
    a = kmalloc(size);

    ...

clean:
    kfree(a);
    spin_unlock(lock);
    return error;
}
```

Policies: Out-of-bound Policy & Template

- PET can be extend to any types of vulnerability as long as proper policies
- policies are designed based on the error conditions of each types of vulnerability
- Templates describe the policies, and new helper functions support templates

```
ptr[offset] = '\0';

ptr + offset ∈ [ bpf_get_start(ptr),
   bpf_get_start(ptr)+bpf_get_len(ptr) )
```

Out-of-bound Policy

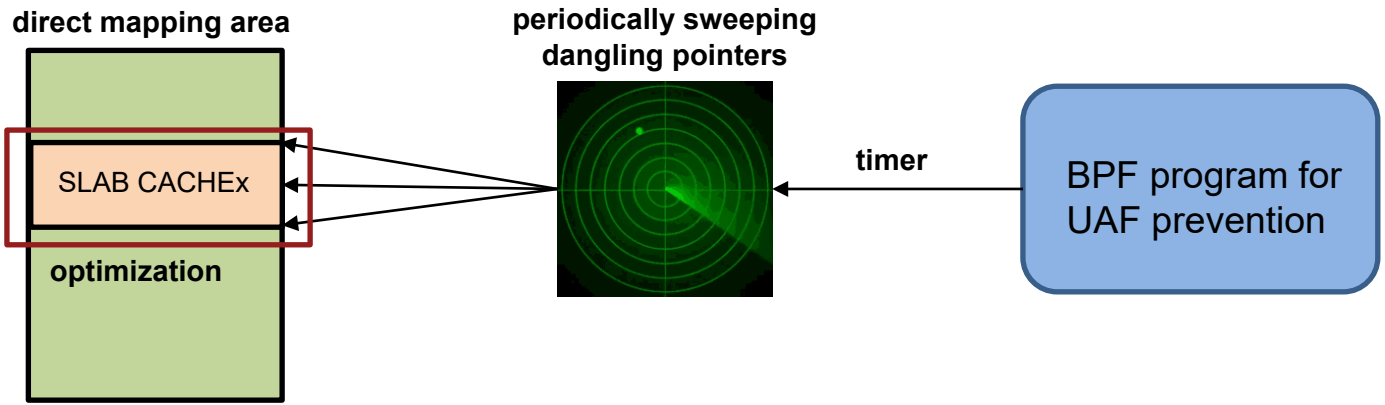
```
SEC("kprobe/func?+offset?") // error site
int BPF_KPROBE(...) {
    u64 addr = ?;
    u64 start = bpf_get_start(addr);
    u64 end = start + bpf_get_len(addr);
    if (addr<start||addr>=end)
        // error condition
        // send SIGKILL signal
        // skip the error instruction
        // direct to function exit
        return -1;
}
```

New helper functions

Out-of-bound Template

Policies: Complex Use-after-free Policy

- Quarantine & sweepine
- Quarantine the freed object until no dangling pointer exists
- Periodically sweep physical memory for dangling pointers
- Optimization: only sweep certain slab cache



Effectiveness

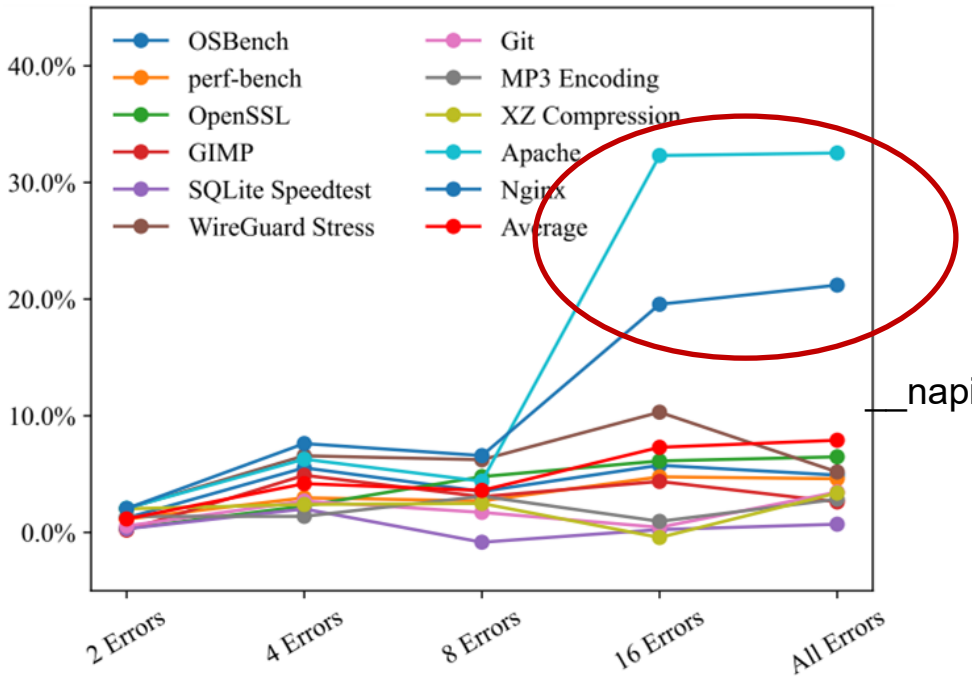
CVE/SYZ ID	Sites for eBPF Installation	Action & Triggering Condition	Effectiveness	Time Window (days)
		Integer Underflow/Overflow		
b5b251b	`dummy_hub_control+0x3f` (spinlock) `dummy_hub_control+0x225`	`lock_map[pid] = $rdi` `$eax<<$edx == $rax<<$edx & $edx<32 ? false : true`	●	79
		Out-of-bound Access on Stack		
2022-1015	`nft_do_chain+0x243`	`$rdi ∈ [$rsp+0x50, $rsp+0xa0)? false : true`	●	147
		Out-of-bound Access on Buddy System Heap		
2022-27666	`null_skcipher_crypt+0x4b`	`$rdi+$rdx ∈ [start($rdi), start($rdi)+len($rdi))? false : true`	●	17
		Out-of-bound Access on SLAB/SLUB Heap		
2022-34918	`nft_set_elem_init+0x3e`	`$rdi+$rcx ∈ [start($rdi), start($rdi)+len($rdi))? false : true`	●	38
797c55d	`watch_queue_set_filter+0x81` (alloc) `watch_queue_set_filter+0x78d`	`alloc_map[pid]=$rdi` `$r15+0x8 ∈ [start($r15), start($r15)+len($r15))? false : true`	●	344
		Use-After-Free		
2022-2586	`nft_obj_destroy+0x3f` (free) `nf_tables_fill_setelem.isra.0+0x140` (use)	`map ∪ $rdi; selective_sweep(kmalloc-256, 0x20)` `$rbx+$rax ∈ map ? true: false`	●	97
be93025d	`__route4_delete_filter+0x3c` (free) `__route4_delete_filter+0x3c` (use)	`map ∪ $rdi; selective_sweep(kmalloc-192, 0x28)` `$rdi ∈ map ? true : false`	●	73
		Uninitialized Access		
2039c557	`__sys_recvfrom` (create) `tcp_recvmsg+0xb8` (use)	`map[$rsp+8-200] = mem($rsp-0xc0, 0x60)` `map[$r13] == mem($r13, 0x60)? false : true`	●(default conservative) ◐(aggressive)	248

The sampled results for the effectiveness, ● indicates that BPF prevention program can be generated and prevent the error.

Performance Overhead

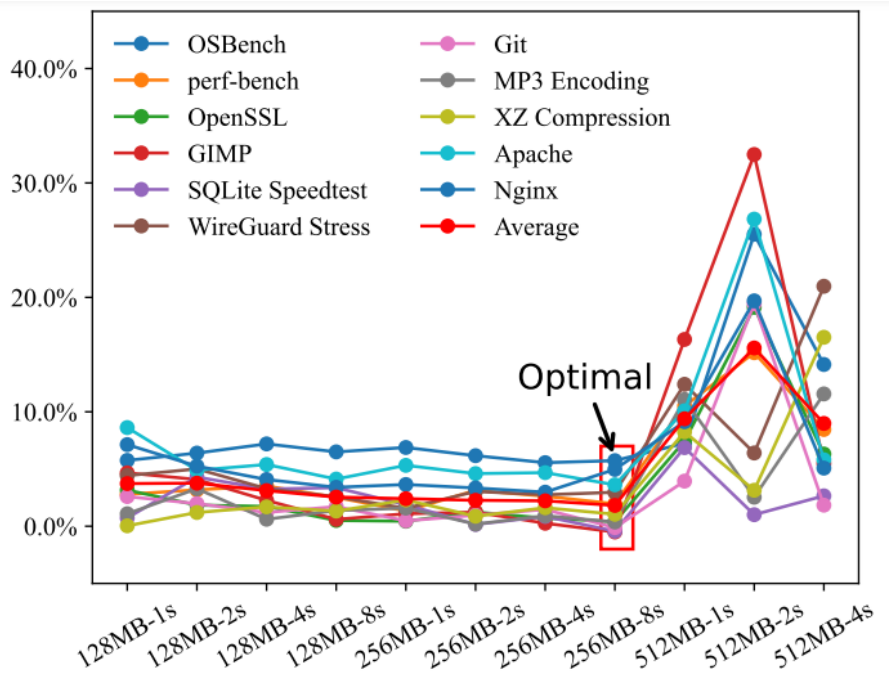
	Slab OOB	Page OOB	Stack OOB	Global OOB	UAF		Integer
	2021-34693	2022-27666	2c0912	2017-18344	be93025	2022-4154	b5b251b
OS Core primitives							
OSBench	0.01%	0.71%	0.38%	0.09%	2.12%	3.05%	0.75%
perf-bench	0.35%	0.03%	-0.18%	0.12%	3.42%	5.86%	0.61%
Calculation intensive							
OpenSSL	0.03%	-0.07%	0.19%	0.19%	1.24%	0.44%	1.90%
MP3 Encoding	0.19%	0.19%	0.95%	0.59%	0.71%	1.59%	0.79%
GIMP	-1.13%	1.34%	-1.12%	-2.96%	-0.17%	1.09%	-0.46%
I/O intensive							
SQLite Speedtest	-0.71%	-0.20%	-0.39%	-1.50%	-0.01%	1.88%	-1.86%
WireGuard Stress	0.14%	0.05%	-0.19%	-0.47%	1.06%	1.57%	-0.85%
Common Server Tasks							
Git	0.07%	0.24%	0.39%	0.16%	0.58%	0.47%	0.86%
Linux Kernel Compile	-0.12%	0.10%	0.03%	0.25%	2.15%	3.23%	1.94%
XZ Compression	0.66%	0.91%	0.03%	0.45%	1.62%	2.29%	-0.72%
Apache	0.38%	0.38%	-1.14%	-0.39%	4.11%	3.64%	-1.86%
Nginx	0.80%	-0.18%	0.23%	0.55%	6.00%	5.32%	1.14%
Average	0.06%	0.29%	-0.07%	-0.24%	1.83%	2.54%	0.19%

Scalability



__napi_poll, called very frequently

Optimal Use-After-Free Sweeper

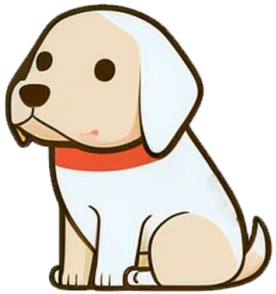


Conclusion

- PET protects kernel before patches are available
  - PET supports error-depent prevention policies for various types of vulnerabilities
  - PET provides error-indepent mechanisms to support prevention policies
  - A thorough evaluation of overhead and scalability

Thank You

- Source
  - https://github.com/purplewall1206/PET
- Contacts
  - wzc@smail.nju.edu.cn
  - yueqi.chen@colorado.edu
  - zqk@nju.edu.cn



PET