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

Zicheng Wang, Yueqi Chen, Qingkai Zeng
Motivation: Protect Kernel before patches are available

- Vulnerability introduced
- Vulnerability discovered
- Patch merged
- Patch deployed

0-day / n-day attacks while patches are unavailable

66 days on average
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

- An 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

Prevention Policy (Error-dependent)

- Integer Overflow Template
- Use-After-Free Template
- Out-of-Bound Template
- Data Race Template
- ...

- Report Processor
- Sanitized-Native Mapper
- Checkpoint-Restore Analyzer

- eBPF Helper Library

Infrastructural Mechanisms (Error-independent)
Mechanisms: Report processor & Sanitized-Native Mapper

Sanitizer report

1. KASAN: slab-out-of-bounds in apparmor_setprocattr+0x116/0x590
2. Write of size 1 at addr ffff888007449c80
3. Call Trace:
   - apparmor_setprocattr+0x116/0x590
   - proc_pid_attr_write+0x15f/0x1e0

error site in the
sanitized image

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

sanitized inst -> statement-> native inst

sanitized reg -> var -> native reg
Mechanisms: Checkpoints & Restore

Register Context
%rax = 111
%rbx = 222
%rdi = 333
%rsi = 444
...
%r15 = xxx

Register Context
%rax = -EINVAL
%rbx = 222
%rdi = 333
%rsi = 444
...
%r15 = xxx

Source Code
security/apparmor/lsm.c

624 static int apparmor_setprocattr(...)
...  
645 args[size] = '\0';
...  
693 return error;

Checkpoint setup

%rax = 111
%rbx = 222
%rdi = 333
%rsi = 444
...
%r15 = xxx

Skip error site

int func(...)
{
spin_lock(lock);
a = kmalloc(size);
...  
clean:
kfree(a);
spin_unlock(lock);
return error;
}

Restore

paired operations
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

```c
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;
}
```

**Out-of-bound Template**

```
ptr[offset] = '\0';
ptr + offset ∈ [ bpf_get_start(ptr), bpf_get_start(ptr)+bpf_get_len(ptr) )
```

**Out-of-bound Policy**
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

<table>
<thead>
<tr>
<th>CVE/SYZ ID</th>
<th>Sites for eBPF Installation</th>
<th>Action &amp; Triggering Condition</th>
<th>Effectiveness</th>
<th>Time Window (days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>b5b251b</td>
<td>dummy_hub_control+0x3f (spinlock)</td>
<td>lock_map[pid] = $rdi $eax&lt;&lt;$edx == $rax&lt;&lt;$edx &amp; $edx32 ? false : true</td>
<td>●</td>
<td>79</td>
</tr>
<tr>
<td></td>
<td>dummy_hub_control+0x225</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2022-1015</td>
<td>nft_do_chain+0x243</td>
<td>$rdi ∈ [$rsp+0x50, $rsp+0xa0]? false : true</td>
<td>●</td>
<td>147</td>
</tr>
<tr>
<td>2022-27666</td>
<td>null_skcipher_crypt+0x4b</td>
<td>$rdi+$rdx ∈ [start($rdi), start($rdi)+len($rdi)]? false : true</td>
<td>●</td>
<td>17</td>
</tr>
<tr>
<td>2022-34918</td>
<td>nft_set_elem_init+0x3e</td>
<td>$rdi+$rcx ∈ [start($rdi), start($rdi)+len($rdi)]? false : true</td>
<td>●</td>
<td>38</td>
</tr>
<tr>
<td>797c55d</td>
<td>watch_queue_set_filter+0x81 (alloc)</td>
<td>alloc_map[pid]=$rdi</td>
<td>●</td>
<td>344</td>
</tr>
<tr>
<td></td>
<td>watch_queue_set_filter+0x78d</td>
<td>$r15+0x8 ∈ [start($r15), start($r15)+len($r15)]? false : true</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>2022-2586</td>
<td>nft_obj_destroy+0x3f (free)</td>
<td>map $rdi; selective_sweep(kmalloc-256, 0x20)</td>
<td>●</td>
<td>97</td>
</tr>
<tr>
<td></td>
<td>nf_tables_fill_setelem.isra.0+0x140 (use)</td>
<td>$rbx+$rax ∈ map ? true: false</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>be93025d</td>
<td>__route4_delete_filter+0x3c (free)</td>
<td>map $rdi; selective_sweep(kmalloc-192, 0x28)</td>
<td>●</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>__route4_delete_filter+0x3c (use)</td>
<td>$rdi ∈ map ? true : false</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>2039c557</td>
<td>__sys_recvfrom (create)</td>
<td>map[$rsp+8-200] = mem($rsp-0xc0, 0x60)</td>
<td>●(default conservative) ●(aggressive)</td>
<td>248</td>
</tr>
<tr>
<td></td>
<td>tcp_recvmsg+0xb8 (use)</td>
<td>map[$r13] == mem($r13, 0x60)? false : true</td>
<td>●</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>Slab OOB</th>
<th>Page OOB</th>
<th>Stack OOB</th>
<th>Global OOB</th>
<th>UAF</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2021-34693</td>
<td>2022-27666</td>
<td>2c0912</td>
<td>2017-18344</td>
<td>be93025</td>
<td>2022-4154</td>
</tr>
<tr>
<td><strong>OS Core primitives</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OSBench</td>
<td>0.01%</td>
<td>0.71%</td>
<td>0.38%</td>
<td>0.09%</td>
<td>2.12%</td>
<td>3.05%</td>
</tr>
<tr>
<td>perf-bench</td>
<td>0.35%</td>
<td>0.03%</td>
<td>-0.18%</td>
<td>0.12%</td>
<td>3.42%</td>
<td>5.86%</td>
</tr>
<tr>
<td><strong>Calculation intensive</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenSSL</td>
<td>0.03%</td>
<td>-0.07%</td>
<td>0.19%</td>
<td>0.19%</td>
<td>1.24%</td>
<td>0.44%</td>
</tr>
<tr>
<td>MP3 Encoding</td>
<td>0.19%</td>
<td>0.19%</td>
<td>0.95%</td>
<td>0.59%</td>
<td>0.71%</td>
<td>1.59%</td>
</tr>
<tr>
<td>GIMP</td>
<td>-1.13%</td>
<td>1.34%</td>
<td>-1.12%</td>
<td>-2.96%</td>
<td>-0.17%</td>
<td>1.09%</td>
</tr>
<tr>
<td><strong>I/O intensive</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQLite Speedtest</td>
<td>-0.71%</td>
<td>-0.20%</td>
<td>-0.39%</td>
<td>-1.50%</td>
<td>-0.01%</td>
<td>1.88%</td>
</tr>
<tr>
<td>WireGuard Stress</td>
<td>0.14%</td>
<td>0.05%</td>
<td>-0.19%</td>
<td>-0.47%</td>
<td>1.06%</td>
<td>1.57%</td>
</tr>
<tr>
<td><strong>Common Server Tasks</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Git</td>
<td>0.07%</td>
<td>0.24%</td>
<td>0.39%</td>
<td>0.16%</td>
<td>0.58%</td>
<td>0.47%</td>
</tr>
<tr>
<td>Linux Kernel Compile</td>
<td>-0.12%</td>
<td>0.10%</td>
<td>0.03%</td>
<td>0.25%</td>
<td>2.15%</td>
<td>3.23%</td>
</tr>
<tr>
<td>XZ Compression</td>
<td>0.66%</td>
<td>0.91%</td>
<td>0.03%</td>
<td>0.45%</td>
<td>1.62%</td>
<td>2.29%</td>
</tr>
<tr>
<td>Apache</td>
<td>0.38%</td>
<td>0.38%</td>
<td>-1.14%</td>
<td>-0.39%</td>
<td>4.11%</td>
<td>3.64%</td>
</tr>
<tr>
<td>Nginx</td>
<td>0.90%</td>
<td>-0.18%</td>
<td>0.23%</td>
<td>0.55%</td>
<td>6.00%</td>
<td>5.32%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>0.06%</td>
<td>0.29%</td>
<td>-0.07%</td>
<td>-0.24%</td>
<td>1.83%</td>
<td>2.54%</td>
</tr>
</tbody>
</table>
Scalability

napi_poll, called very frequently
Optimal Use-After-Free Sweeper
Conclusion

- PET protects kernel before patches are available
  - PET supports error-dependent prevention policies for various types of vulnerabilities
  - PET provides error-independent 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