Jenny: Securing Syscalls for PKU-based Memory Isolation Systems

David Schrammel, Samuel Weiser, Richard Sadek, Stefan Mangard

IAIK – Graz University of Technology
Intra-Process Isolation

- Protection Keys for Userspace (MPK or PKU)
  - Fine-grained privilege separation
  - e.g., ERIM, Hodor, Donky
- Problems:
  - Insufficient syscall filtering
  - Slow syscall interception (ptrace) / or relying on custom hardware
- Our Contribution
  - Analyze Linux’s syscall interface
  - Define filter rules for PKU sandboxes
  - Analyze interception mechanisms
  - Based on Donky, we implement:
    - Same-thread syscall filtering
    - Nested syscall filtering
    - Safe signal handling
Memory pages tagged with a “protection key”
- Key stored in Page Table Entry
- Intel MPK: 16 keys
Memory pages tagged with a “protection key”
- Key stored in Page Table Entry
- Intel MPK: 16 keys

Permissions in PKRU register:
- Allows to quickly change memory permissions (from userspace)

Virtual address space:

<table>
<thead>
<tr>
<th>Address</th>
<th>PTEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA000</td>
<td>phys. addr. r w x ...</td>
</tr>
<tr>
<td>0xB000</td>
<td>phys. addr. r w x ...</td>
</tr>
<tr>
<td>0xC000</td>
<td>phys. addr. r w x ...</td>
</tr>
<tr>
<td>0xD000</td>
<td>phys. addr. r w x ...</td>
</tr>
</tbody>
</table>

PKRU: 0 0 0 0 1 1 0 0 0 0
Memory pages tagged with a “protection key”
- Key stored in Page Table Entry
- Intel MPK: 16 keys

Permissions in PKRU register:
- Allows to quickly change memory permissions (from userspace)

PKU-based sandboxing:
- Safeguard PKRU & WRPKRU
  - No unsafe writes to the register exist (WRPKRU)
  - Limit system calls that bypass PKRU
Setting: In-Process Domains and Syscalls

Diagram:

- Userspace Application
  - Domain 1
  - Domain 2
  - Monitor

- OS
Setting: In-Process Domains and Syscalls

Userspace Application

Domain 1

Domain 2

Monitor

Syscall

OS
Setting: In-Process Domains and Syscalls

Userspace Application

Domain 1

Domain 2

Monitor

OS

?
Setting: In-Process Domains and Syscalls

Userspace Application

Domain 1  Domain 2

Monitor

OS
Setting: In-Process Domains and Syscalls

Userspace Application

Domain 1

Domain 2

Domain 3

Monitor

OS
• **Efficient** mechanism
• **Effective** Filters
• Kernel *unaware* of userspace domains.
• Nesting
• Signals
Existing Syscall-based attacks (e.g., Connor et al.)

- Indirect memory accesses:
  - procfs /proc/self/mem
  - process_vm_write, ptrace

- Mutably-backed read-only memory

- Seccomp filters

- Signal Handlers
  - modify PKRU via sigreturn
  - unprotected stack
New Syscall-based attacks

- `madvise` → can clear pages irrespective of their protection keys
- `brk`, `sbrk` → can clear pages and remove protection keys
- `userfaultfd` → can write arbitrary values to protected pages
- `personality(READ IMPLIES EXEC)`
- `core dumps`
- `fork`, `clone`, `exec`, `arch` prctl, `set thread` area, ...
New Syscall-based attacks

- madvise
  - can clear pages irrespective of their protection keys
New Syscall-based attacks

- madvise
  → can clear pages irrespective of their protection keys

- brk, sbrk
  → can clear pages and remove protection keys
New Syscall-based attacks

- madvise
  → can clear pages irrespective of their protection keys
- brk,sbrk
  → can clear pages and remove protection keys
- userfaultfd
  → can write arbitrary values to protected pages
New Syscall-based attacks

- `madvise`
  - can clear pages irrespective of their protection keys

- `brk, sbrk`
  - can clear pages and remove protection keys

- `userfaultfd`
  - can write arbitrary values to protected pages

- `personality(READ_IMPLIES_EXEC)`
New Syscall-based attacks

- **madvise**
  - can clear pages irrespective of their protection keys
- **brk,sbrk**
  - can clear pages and remove protection keys
- **userfaultfd**
  - can write arbitrary values to protected pages
- **personality(READ.IMPLIES_EXEC)**
- **core dumps**
New Syscall-based attacks

- **madvise**
  - can clear pages irrespective of their protection keys
- **brk,sbrk**
  - can clear pages and remove protection keys
- **userfaultfd**
  - can write arbitrary values to protected pages
- **personality(READ(IMPLIES_EXEC)**
- **core dumps**
- **fork, clone, exec, arch_prctl, set_thread_area, . . .**
We define three filter sets:

- “base-donky” (for Donky’s HW extension)
- Memory (Data and Code) of Monitor & Domains
- Signals
- Application resources (bpf, seccomp, prctl, ...)

- “base-mpk” (for x86-64 CPUs)
- Prevent exploitation from unsafe instructions (e.g., WRPKRU)
  → Binary Scanning
  → Executable Memory is immutable

- “localstorage”
  - Per-domain filesystem and file-descriptors
We define three filter sets:

- “base-donky” (for Donky’s HW extension)
  - Memory (Data and Code) of Monitor & Domains
  - Signals
  - Application resources (bpf, seccomp, prctl, ...)

- “base-mpk” (for x86-64 CPUs)
  - Prevent exploitation from unsafe instructions (e.g., WRPKRU)

- "localstorage"
  - Per-domain filesystem and file-descriptors
We define three filter sets:

- **“base-donky”** (for Donky’s HW extension)
  - Memory (Data and Code) of Monitor & Domains
  - Signals
  - Application resources (bpf, seccomp, prctl, ...)

- **“base-mpk”** (for x86-64 CPUs)
  - Prevent exploitation from unsafe instructions (e.g., WRPKRU)
    → Binary Scanning
    → Executable Memory is immutable
We define three filter sets:

- "base-donky" (for Donky’s HW extension)
  - Memory (Data and Code) of Monitor & Domains
  - Signals
  - Application resources (bpf, seccomp, prctl, ...)

- "base-mpk" (for x86-64 CPUs)
  - Prevent exploitation from unsafe instructions (e.g., WRPKRU)
    → Binary Scanning
    → Executable Memory is immutable

- "localstorage"
  - Per-domain filesystem and file-descriptors
1. in memory (e.g., by protecting it with protection keys)
2. its memory mapping (i.e., rwx permissions and protection keys)
3. on disk, if mapping is file-backed
   → disallow mapping executable memory backed by writable files
• any aliases
  • shared memory, symlinks, ...
  • procfs (/proc/[pid]/mem) and core-dumps
   → Instead of filtering paths, we disable the procfs and core dumps via prctl
Sysscall Filtering Mechanism Requirements

- Delegation
  - to the userspace monitor
  - Intercept pre- & post-syscall
- Domain-aware
  - trusted domain (i.e., monitor) should not be intercepted (again)
- Emulation/Expressiveness
  - Read/Write syscall arguments (e.g., manipulate paths)
  - Perform arbitrary syscalls
- Impersonation
  - Perform syscalls on behalf of filtered domain
- Nesting
  - Hierarchical syscall filtering
  - Each domain can filter their child domains
- Performance
## Existing Syscall Filtering Mechanisms

### Filter can

<table>
<thead>
<tr>
<th>Filter can</th>
<th>Linux seccomp-bpf</th>
<th>Linux seccomp-user</th>
<th>Linux seccomp-trap</th>
<th>Linux prctl(seccomp)</th>
<th>Linux syscall user dispatch</th>
<th>RISC-Y user interrupts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run in kernel (S), user (U)</td>
<td>S</td>
<td>S/U</td>
<td>S/U</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>PKRU-aware delegation</td>
<td>n.a.</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Intercept pre-syscall</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Allow syscall</td>
<td>✔️</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Intercept post-syscall</td>
<td>—</td>
<td>n.a.</td>
<td>n.a.</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

### Emulate

<table>
<thead>
<tr>
<th>Emulate</th>
<th>Linux seccomp-bpf</th>
<th>Linux seccomp-user</th>
<th>Linux seccomp-trap</th>
<th>Linux prctl(seccomp)</th>
<th>Linux syscall user dispatch</th>
<th>RISC-Y user interrupts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read syscall arguments</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Write syscall arguments</td>
<td>—</td>
<td>n.a.</td>
<td>n.a.</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Read/Write any memory</td>
<td>—</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Read/Write PKRU register</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Read/Write other registers</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Manipulate syscall return value</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Perform arbitrary syscalls</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

### Impersonate (threading) syscalls

<table>
<thead>
<tr>
<th>Impersonate (threading) syscalls</th>
<th>Linux seccomp-bpf</th>
<th>Linux seccomp-user</th>
<th>Linux seccomp-trap</th>
<th>Linux prctl(seccomp)</th>
<th>Linux syscall user dispatch</th>
<th>RISC-Y user interrupts</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

### Kernel context switches on deny

| Kernel context switches on deny | 1 | 2 | 2 | 4+ | 2 | 0 |
None fit our needs

- **Unaware** of PKU domains
- Relying on **insecure** signal handlers
- Filters are too **limited**
- **Slow**
- unavailable on x86-64
• Designed new “pku-user-delegate”
  • Kernel module
  • Selectively intercept if
    in non-monitor domain
    filter exists
  • Delegate the userspace monitor
• Designed new “pku-user-delegate”
  • Kernel module
  • Selectively intercept if
    in non-monitor domain
    filter exists
  • Delegate the userspace monitor
  + “libc-indirect”
    • Modify libc to dispatch syscalls to monitor
      → approximate HW extension for x86
• Signal handlers could not run on a PKU-protected stack
  • Propose change to support protected stacks
  • Kernel patch (33 LoC)
• Provide Signal API
  • Domains can register signal handlers
  • Filter/Virtualize signals for child-domains
  • Signals land in monitor and get dispatched
Nesting
Same-Thread Filtering
  • incl. ptrace
Secure filter design
  • Protected arguments
  • TOCTOU, PKRU checks, Locking, argument copying, ...
Binary Scanning
  • wrpkru, xrstor
  • whenever executable mappings change
Multi-Domain Call-Gates
Figure 1: Relative application runtime
Conclusion

- Investigate syscall filtering for PKU-based sandboxes
- Identify new syscall-based attacks
- Derive efficient filter rules
- Study of syscall interposition techniques w.r.t PKU suitability
- New interposition mechanism: fast and secure
- Domain specific
  - syscall filtering (in a nested way)
  - secure signal handling
Jenny: Securing Syscalls for PKU-based Memory Isolation Systems

David Schrammel, Samuel Weiser, Richard Sadek, Stefan Mangard

IAIK – Graz University of Technology