Jenny: Securing Syscalls for PKU-based Memory Isolation Systems

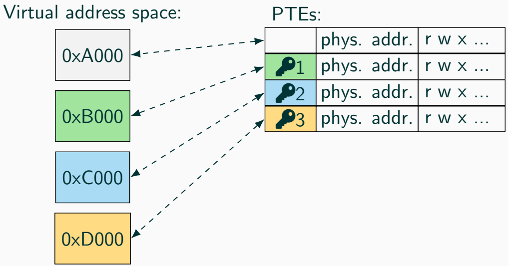
David Schrammel, **Samuel Weiser, Richard Sadek, Stefan Mangard**

IAIK – Graz University of Technology

Intra-Process Isolation

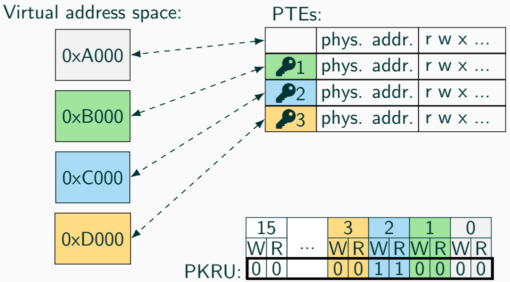
- Protection Keys for Userspace (MPK or PKU)
  - Fine-grained privilege separation
  - e.g., ERIM, Hodor, Donky
- Problems:
  - Insufficient syscall filtering
  - Slow syscall interception (ptrace) / or relying on custom hardware
- **Our Contribution**
  - Analyze Linux's syscall interface
  - Define 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

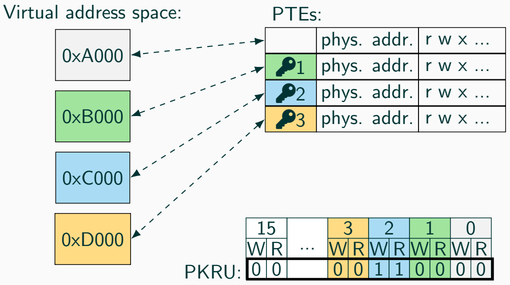


Background: Memory Protection Keys for Userspace

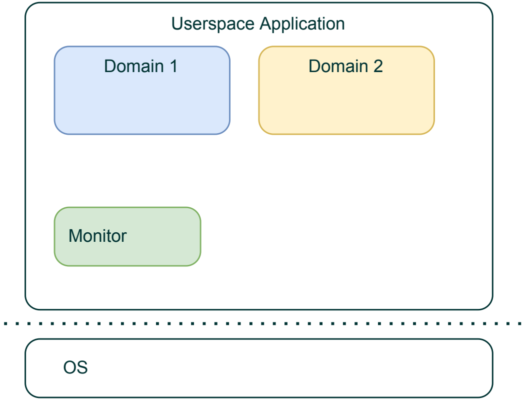
- Memory pages tagged with a "protection key"
  - Key stored in Page Table Entry
  - Intel MPK: 16 keys
- Permissions in PKRU register:
  - Allows to quickly change memory permissions (from userspace)

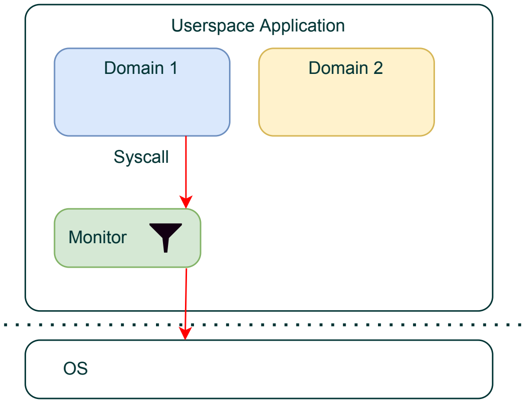


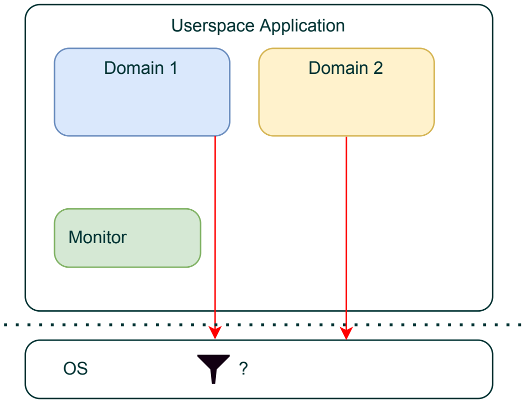
- Memory pages tagged with a "protection key"
  - Key stored in Page Table Entry
  - Intel MPK: 16 keys
- Permissions in PKRU register:
  - Allows to quickly change memory permissions (from userspace)
- PKU-based sandboxing:
  - Safeguard PKRU & WRPKRU
    - No unsafe writes to the register exist (WRPKRU)
    - Limit syscalls that bypass PKRU



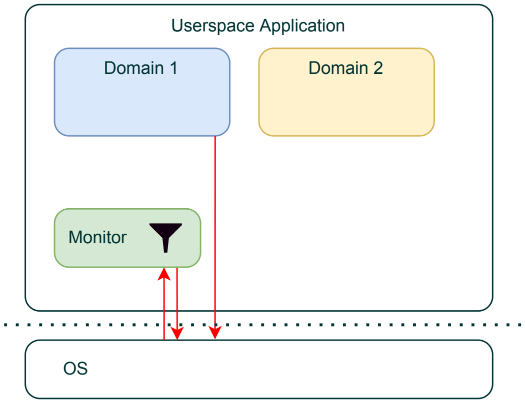
Setting: In-Process Domains and Syscalls



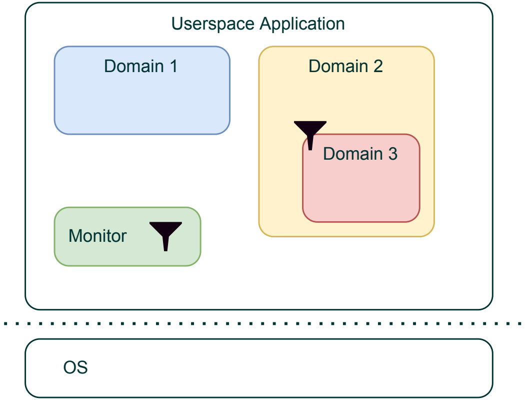

Userspace Application
Domain 1
Domain 2
Syscall
Monitor
OS


Userspace Application
Domain 1
Domain 2
Monitor
OS
?

Setting: In-Process Domains and Syscalls



Setting: In-Process Domains and Syscalls



Challenges

- **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

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:

Effective Filters

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, ...)

Effective Filters

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

Effective Filters

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`

Syscall 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	Linux seccomp-bpf	Linux seccomp-user	Linux seccomp-trap	Linux ptrace(-seccomp)	Linux sysc. user dispatch	RISC-V user interrupts
Delegate	Run in kernel (S), user (U)	S	S/U	S/U	U	U	U
	PKRU-aware delegation	n.a.	—	—	✔	—	✔
	Intercept pre-syscall	✔	✔	✔	✔	✔	✔
	Allow syscall	✔	—	—	✔	✔	✔
	Intercept post-syscall	—	n.a.	n.a.	✔	✔	✔
Emulate	Read syscall arguments	✔	✔	✔	✔	✔	✔
	Write syscall arguments	—	n.a.	n.a.	✔	✔	✔
	Read/Write any memory	—	✔	✔	✔	✔	✔
	Read/Write PKRU register	—	—	✔	✔	✔	✔
	Read/Write other registers	—	—	✔	✔	✔	✔
	Manipulate syscall return value	✔	✔	✔	✔	✔	✔
	Perform arbitrary syscalls	—	—	—	✔	✔	✔
Impersonate (threading) syscalls		—	—	✔	✔	✔	✔
Kernel context switches on deny		1	2	2	4+	2	0

Existing Syscall Filtering Mechanisms

	Filter can	Linux seccomp-bpf	Linux seccomp-user	Linux seccomp-trap	Linux ptrace(-seccomp)	Linux sysc. user dispatch	RISC-V user interrupts
	Run in kernel (S), user (U)	S	S/U	S/U	U	U	U
Delegate	PKRU-aware delegation	n.a.	—	—	✔	—	✔
Delegate	Intercept pre-syscall	✔	✔	✔	✔	✔	✔
Delegate	Allow syscall	✔	—	—	✔	✔	✔
Delegate	Intercept post-syscall	—	n.a.	n.a.	✔	✔	✔
Emulate	Read syscall arguments	✔	✔	✔	✔	✔	✔
Emulate	Write syscall arguments	—	n.a.	n.a.	✔	✔	✔
Emulate	Read/Write any memory	—	✔	✔	✔	✔	✔
Emulate	Read/Write PKRU register	—	—	✔	✔	✔	✔
Emulate	Read/Write other registers	—	—	✔	✔	✔	✔
Emulate	Manipulate syscall return value	✔	✔	✔	✔	✔	✔
Emulate	Perform arbitrary syscalls	—	—	—	✔	✔	✔
Impersonate (threading) syscalls		—	—	✔	✔	✔	✔
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

Syscall Filtering Mechanisms (for x86)

- Designed new "pku-user-delegate"
  - Kernel module
  - Selectively intercept if

    in non-monitor domain

    filter exists
  - Delegate the userspace monitor

Syscall Filtering Mechanisms (for x86)

- Designed new "pku-user-delegate"
  - Kernel module
  - Selectively intercept if

    in non-monitor domain

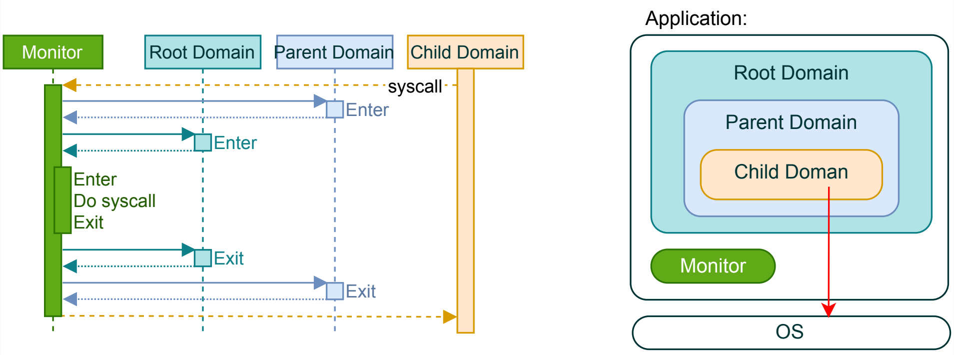
    filter exists
  - Delegate the userspace monitor

+ "libc-indirect"
  - Modify libc to dispatch syscalls to monitor

  → approximate HW extension for x86

Signals

- 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



Monitor
Root Domain
Parent Domain
Child Domain
syscall
Enter
Enter
Enter
Do syscall
Exit
Exit
Exit
Application:
Root Domain
Parent Domain
Child Doman
Monitor
OS

(Skipped) Implementation Details

- 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

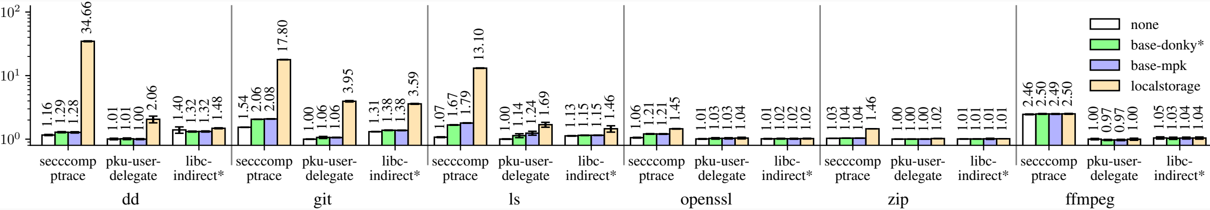


Figure 1: Relative application runtime

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