PKU Pitfalls:
Attacks on PKU-based Memory Isolation Systems

Joseph Connor
Tyler McDaniel
Jared M. Smith
Max Schuchard
Overview

• Intraprocess isolation: separating components within a process
• Improves security
• Allows finer-grained privilege separation
• But:
  • Not widely adopted
  • Has suffered from performance and complexity issues
Overview

• Recent research shows **improved performance** using new hardware feature:
  • Protection Keys for Userspace (PKU)
• Our contribution: identify challenges and gaps in current approach
  • Researchers, OS devs have different goals and views on hardware, process security
  • Commonly-used assumptions may not hold in real-world systems
Background - PKU

- Assigns a 4-bit “protection key” (0-15) to each page-table entry
- Adds a new unprivileged 32-bit register – PKRU
  - Modified with new instruction: `wrpkr u`
  - Pairs of bits control access to the 16 protection keys
- MMU hardware checks PKRU on each memory access

<table>
<thead>
<tr>
<th>Protection Key</th>
<th>PKEY 15</th>
<th>PKEY 14</th>
<th>PKEY 1</th>
<th>PKEY 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access/Write Disable</td>
<td>WD</td>
<td>AD</td>
<td>WD</td>
<td>AD</td>
</tr>
<tr>
<td>Bit</td>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
</tr>
</tbody>
</table>

Bits $2i$ and $2i+1$ control read/write access to pages with protection key $i$
Background - PKU

- PKRU register is **unprivileged**
- On its own, does not stop an attacker who has already hijacked control flow in a process
  - Code may contain \texttt{wrpkru} gadgets
  - Attacker may \texttt{mmap()} new code
- Proposed solutions to this problem:
  - ERIM (Vahldieck-Oberwagner et al.)
  - Hodor (Hedayati et al.)
Background – ERIM/Hodor

- Divide process into separate “components” using PKU to control memory access
- Ensure there are no wrpkru gadgets available for an attacker to exploit:
  - ERIM: using static binary rewriting
  - Hodor: using x86 hardware watchpoints
- Efficient context switching via safe “call gates”
  - wrpkru followed by jmp to trusted code
  - wrpkru followed by code to ensure PKRU is unprivileged
Background – ERIM/Hodor

- Monitor newly executable pages created by `mmap()` or `mprotect()`
  - Using `seccomp + ptrace`
  - Using kernel modifications
- Scan new code for unsafe `wrpkru` gadgets
- Enforce “W^X”: No memory is both writable and executable
  - Otherwise, attacker may create new `wrpkru` gadgets without interception
Our Approach

• Examined kernel documentation, code, and developer communications in context of PKU-based sandboxes
• Tested attacks on prototype in a realistic Linux system
• Assumptions:
  • Attacker can execute control-flow hijacking attack (e.g. ROP chain) in untrusted context
  • Attacker must access memory of trusted component without using a legitimate call gate
• Developed 12 proof-of-concepts
Challenges for Intraprocess Isolation

- Fundamental departure from traditional OS security boundaries
- Researchers/OS devs have different perspectives on security models for processes and PKU
  - Kernel can act as “confused deputy”
- Lack of a method for systemic validation
Kernel As Confused Deputy

- Mailing list discussions show that kernel developers envisioned PKU use cases for reliability, not security
- Linux kernel intentionally does not absolutely enforce:
  - PKRU access checks
  - Page table entry read/write permissions
- `get_user_pages_remote()` circumvents these checks
  - `ptrace()`
  - `process_vm_readv/writev()`
  - `/proc/<pid>/mem`
Difficulty of $W^X$

- Research often assumes $W^X$ – memory is not simultaneously writable and executable
  - PKU-based sandboxes rely on $W^X$ to ensure `wrpkru` gadgets cannot be written without being intercepted
  - Enforced via userspace syscall interception (`ptrace+seccomp`) or kernel modifications
- In practice, non-trivial to fully enforce on Linux
Difficulty of W^X

- Page permissions apply to *mappings*, not the physical memory
  - Pages may also be backed by other resources (e.g. files)
- Writes to memory-mapped files are reflected even in non-writable mappings
- Shared memory can be mapped more than once, with different permissions
- Some kernel interfaces ignore PTE permissions: ptrace(), /proc/<pid>mem
Conclusions

• Seemingly simple assumptions ($W^X$) may not apply as expected in realistic settings

• Retrofitting different security models is especially challenging:
  • One developer’s design choice is another developer’s vulnerability

• Systematic approach to validating security boundaries is needed
Thank You

**Joseph Connor** – rconnor6@utk.edu
Tyler McDaniel – bmcdn16@utk.edu
Jared M. Smith – jms@utk.edu
Max Schuchard – mschucha@utk.edu