PT-Rand
Practical Mitigation of Data-only Attacks against Page Tables

David Gens
Christopher Liebchen
Ahmad-Reza Sadeghi
Cyber Security Center
Technische Universität Darmstadt

Lucas Davi
University of Duisburg-Essen
Impact of Kernel Attacks

- User
- Lower Privileges
  - Browser
  - Document Viewer
  - Word Processor

Exploit Vulnerability

Control-Flow Integrity [Abadi et al., CCS 2005]

Return-oriented Programming [Shacham, CCS 2007]
CFI for Linux Kernel: Return Address Protection (RAP)

Grsecurity ends code reuse attacks with RAP

RAP Demonstrates World-First Fully CFI-Hardened OS Kernel
Type-based, high-performance, high-security, forward/backward-edge CFI
February 6, 2017

https://grsecurity.net/rap_announce_ret.php
Is Control-Flow Integrity enough?

• **Protects** against control-flow hijacking*

• **Vulnerable** to non-control data attack

*Terms and Conditions May Apply
Virtual Memory:
Page Tables

CPU

Page Table Root

Page Table Hierarchy

Physical Address | Permissions
-----------------|-------------
0x1000           | 0 1 0 0

Translation and Permission Enforcement

MMU = Memory Management Unit
Data-Only against Page Tables of a CFI-hardened Kernel
Data-Only Attacks Against Page Tables

Consider the following code snippet and diagram:

```
0xffff880016d2c000L 1087 kworker/1:8
0xffff88001a9e3540L 1132 kworker/0:2
0xffff88001ab4000L 1133 kworker/1:1
0xffff880016d2500L 1140 pythonroot
[+] mm: 0xffff880019c000L
getting pte for 0xfffffffff810a400L
p14 0xffff888015a59ff8L -> 0x1d28067L
p13 0xffff8880001d28ff0L -> 0x1d2a063L
2mb page
pte 0xffff8880001d2a040L -> 0x100001e1L
[+] mark sysns page as writable
[+] writing shellcode
mov rbx, prepare_kernel_cred
call rbx
mov rbx, commit_creds
call rbx
mov rax, 0x1337
ret
[+] getting r00t...
# id
uid=0(root) gid=0(root) groups=0(root)
```

The diagram illustrates the flow of execution:

1. **CFI-Hardened Kernel**
   - **Code**: RWX
   - **Data**: RW
   - **Shellcode** is placed in RWX, with RWX entries pointing to the Shellcode segment.
   - **Page Tables**: RW

2. **User Mode**
   - System call is triggered to execute the injected shellcode.
   - Overwrite existing function entries (e.g., system call) with shellcode.
   - Manipulate the page table to allow access to shellcode.

The diagram and code snippet together demonstrate how data-only attacks can exploit page tables to execute arbitrary code in user mode.
Page-Table Protection: Shortcomings of Related Work

• Proposed schemes to ensure page-table integrity
  • HyperSafe [Wang and Jiang, IEEE S&P 2010]
  • SPROBES [Ge et al., IEEE MoST 2014]
  • KCoFI [Criswell et al., IEEE S&P 2014]
  • SKEE [Azab et al., NDSS 2015]

• However, they suffer from the following problems
  • Require hardware trust anchors
  • Require a trusted hypervisor
  • Inefficient integrity check
Our Approach:
Page-Table Randomization
Assumptions and Threat Model

- Modern CPUs prevent ret2usr attacks (SMAP/SMEP)
- Cannot inject new code into the kernel (W^X)
- Code-reuse defense in place (CFI)

- Control over a user application
- Read/Write from/to known addresses
PT-Rand: High-level Idea

- Address space for 64 bit systems is huge
- Move to random location in unused memory page tables
- Protect all pointers
PT-Rand: Challenges & Details

• References to page tables
  → All references are replaced by physical addresses
  → Page table management patched process physical addresses

• Protection of the randomization secret
  → Store in debug register and make it leakage resilient

• Preserve Physmap functionality for regular accesses
  → Our approach only removes page table data from Physmap
Evaluation
Security

• Guessing Attacks
  • $p = 3.726 \times 10^{-9}$ (Desktop, 4000 Page-Table Entries)
  • $p = 3.762 \times 10^{-9}$ (Server w/ 9 parallel VMs, 33000 PTE)

• Memory-disclosure Attacks
  • Through pointers: All pointers are converted to physical address
  • Spilled registers
    • DR3 are not spilled during interrupts
    • Software interrupts are disabled during page walks
Implementation

• Linux Kernel v4.6 hardened with RAP
  • 45 source files
  • 1382 insertions
  • 15 deletions

• Intel Core i7-4790 CPU
• 8 GB RAM
• Debian 8.2
Performance

• SPEC CPU 2006: avg. 0.22% (max 1.7%)

• Phoronix: 0.08% (max. 1.8%)

• LMBench fork+exec: +0.1 ms

• Chromium
  • Start time (+ < 1ms)
  • Run time avg. ~0.294% (JetStream/Octan/Kraken)
Conclusion

• Page-table attacks pose a **serious threat** to kernel security
• First **practical** randomization-based defense for page tables
  • Mitigates data-only attacks
  • No dependencies on higher privileged execution modes
  • Complements kernel CFI
• Proof-of-concept implementation
  • Negligible overhead
  • No impact on the stability of the overall system