IskiOS: Lightweight Defense Against Kernel-Level Code-Reuse Attacks

Spyridoula Gravani, Mohammad Hedayati, John Criswell, and Michael L. Scott
Department of Computer Science, University of Rochester
{sgravani,hedayati,criswell,scott}@cs.rochester.edu

ABSTRACT

Commodity operating systems such as Windows, Linux, and MacOS form the trusted computing base of today's computing systems, and execute with elevated privileges to protect applications from malicious behavior. However, since they are written in C and C++, they have memory safety errors and are vulnerable to kernel-level code-reuse attacks. In this talk, we present IskiOS: a system for operating systems on the x86 processor that mitigates code reuse attacks by providing three key pieces of functionality: a race-free integrity-protected shadow stack, execute-only code memory, and leakage-resilient diversification with code-pointer hiding.

KEYWORDS

security, operating systems, compiler transformations, code reuse

1 INTRODUCTION

Control-flow hijacking attacks [14, 18] exploit memory corruption vulnerabilities to take over execution and control the behavior of a program. When that program is the operating system kernel, everything running on the machine is at risk. Even in the absence of conventional control-flow attacks which inject a malicious payload [13], code-reuse attacks (CRAs) may repurpose existing code in memory, bypassing widely deployed data-execution prevention mechanisms. Advanced code-reuse attacks [15] also exploit memory disclosure vulnerabilities [16] to circumvent address space layout randomization [17]. Existing defenses against kernel-level CRAs [2, 3, 8, 12] rely on static analysis to tag legitimate code paths and use run-time checks to force execution to follow these code paths. Unfortunately, because static analysis is inevitably imprecise, such defenses are easily bypassed by advanced code-reuse attacks [1, 4].

To defend against kernel-level CRAs, we propose a comprehensive solution that (1) diversifies code layout through fine-grained address space randomization, (2) protects against direct disclosure of the layout by making executable memory unreadable, and (3) prevents corruption of return addresses during execution. This talk presents *IskiOS* [5], a system that leverages the PKU [7] feature available on Intel's recent x86 processors to implement execute-only memory and protected shadow stacks with low run-time overhead. Execute-only memory protects against direct disclosure of the code layout by making executable memory unreadable; protected shadow stacks prevent corruption of return addresses during execution. In addition, IskiOS diversifies kernel code at compile time to mitigate indirect memory disclosure attacks that leak code pointers from readable memory (e.g., heap and stack) to infer the protected code layout.

GLSD '19, September 6, 2019, Buffalo, NY
.

2 ISKIOS: DESIGN AND IMPLEMENTATION

The key to our design is a novel use of Intel's memory protection keys, which it calls Protection Keys for Userspace (PKU) [7]. PKU allows the program to indicate, dynamically, that read and/or write access should be disabled for certain sets of pages. As the name implies, PKU applies only to memory whose page table entries (PTEs) are marked as user space. Over the past year, however, widespread adoption of kernel page-table isolation (KPTI) [6] as mitigation for Meltdown attacks [9] has essentially obviated use of the user/supervisor bit in PTEs. Since we use an entirely separate page table when running in the kernel, there is no reason kernel pages cannot be marked as "user" memory, making PKU usable in kernel space. Intel's x86 processors use one PTE bit to distinguish between read/write and read-only pages and another to indicate executability [7]. This convention does not support an execute-only (unreadable) mode. By leveraging PKU and KPTI, however, IskiOS obtains the effect of such a mode by disabling read access for all code pages in the kernel. Unlike previous work, IskiOS places no restrictions on virtual address space layout: the OS kernel can scatter code pages throughout the address space, allowing randomization techniques to use as much entropy as desired.

Separately, IskiOS uses PKU to ensure the integrity of a race-free shadow stack that is used to protect all function returns and that is writable only during short sequences of straightline code in function prologues. Because the `wrpkru` instruction uses are hidden through diversification and execute-only memory, an attacker who cannot inject code into the kernel is unable to override either the execute-only code segment or the protected shadow stack.

3 RESULTS

We used the LMBench micro-benchmark suite [10] to quantify IskiOS's overhead on basic kernel operations and the Phoronix Test Suite [11] to measure the performance impact on real-world applications. We demonstrate that our PKU-based implementation of execute-only memory incurs a geometric mean of 10% overhead across the LMBench micro-benchmarks, almost all of which is due to the KPTI implementation. The addition of code-pointer hiding and protected shadow stacks bumps the overhead up to 103% (geomean) compared to an unmodified Linux 4.19 kernel. On the Phoronix system benchmarks, our full-IskiOS defenses lead to an overhead of 3.5% (geomean).

REFERENCES

[1] Nicolas Carlini, Antonio Barresi, Mathias Payer, David Wagner, and Thomas R. Gross. 2015. Control-flow Bending: On the Effectiveness of Control-flow Integrity. In *Proceedings of the 24th USENIX Security Symposium (SEC)*. Washington, D.C., 161–176. http://dl.acm.org/citation.cfm?id=2831143.2831154

[2] John Criswell, Nathan Dautenhahn, and Vikram Adve. 2014. KCoFI: Complete Control-Flow Integrity for Commodity Operating System Kernels. In *Proceedings*

of the 35th IEEE Symposium on Security and Privacy (S&P). San Jose, CA, 292–307. https://doi.org/10.1109/SP.2014.26

[3] X. Ge, N. Talele, M. Payer, and T. Jaeger. 2016. Fine-Grained Control-Flow Integrity for Kernel Software. In *Proceedings of the 1st IEEE European Symposium on Security and Privacy (EuroS&P)*. Saarbrücken, Germany, 179–194. https://doi.org/10.1109/EuroSP.2016.24

[4] Enes Göktas, Elias Athanasopoulos, Herbert Bos, and Georgios Portokalidis. 2014. Out of Control: Overcoming Control-Flow Integrity. In *Proceedings of the 35th IEEE Symposium on Security and Privacy (S&P)*. San Jose, CA, 575–589. https://doi.org/10.1109/SP.2014.43

[5] Spyridoula Gravani, Mohammad Hedayati, John Criswell, and Michael L. Scott. 2019. IskiOS: Lightweight Defense Against Kernel-Level Code-Reuse Attacks. *arXiv e-prints* (2019). arXiv:cs.CR/1903.04654

[6] Dave Hansen. 2017. KPTI Patch. https://lkml.org/lkml/2017/12/18/1523. [Online; accessed 11-March-2019].

[7] Intel. 2018. Intel 64 and IA-32 Architectures Software Developer's Manual. 325462-067US.

[8] J. Li, X. Tong, F. Zhang, and J. Ma. 2018. Fine-CFI: Fine-Grained Control-Flow Integrity for Operating System Kernels. *IEEE Transactions on Information Forensics and Security (TIFS)* 13, 6 (June 2018), 1535–1550. https://doi.org/10.1109/TIFS.2018.2797932

[9] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading Kernel Memory from User Space. In *Proceedings of the 27th USENIX Security Symposium (SEC)*. Baltimore, MD, 973–990. https://www.usenix.org/conference/usenixsecurity18/presentation/lipp

[10] Larry McVoy and Carl Staelin. 1996. lmbench: Portable Tools for Performance Analysis. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. San Diego, CA, 23–23. http://dl.acm.org/citation.cfm?id=1268299.1268322

[11] Phoronix Media. [n.d.]. Phoronix Test Suite. https://www.phoronix-test-suite.com. [Online; accessed 11-March-2019].

[12] João Moreira, Sandro Rigo, Michalis Polychronakis, and Vasileios P. Kemerlis. 2017. DROP THE ROP: Fine Grained Control-Flow Integrity for The Linux Kernel. In *Black Hat Asia*.

[13] Aleph One. 1996. Smashing the Stack for Fun and Profit. *Phrack* 7 (8 November 1996). Issue 49. http://www.phrack.org/issues/49/14.html [Online; accessed 11-March-2019].

[14] Ryan Roemer, Erik Buchanan, Hovav Shacham, and Stefan Savage. 2012. Return-Oriented Programming: Systems, Languages, and Applications. *ACM Transactions on Information Systems Security (TISSEC)* 15, 1, Article 2 (March 2012), 34 pages.

[15] Kevin Z. Snow, Fabian Monrose, Lucas Davi, Alexandra Dmitrienko, Christopher Liebchen, and Ahmad-Reza Sadeghi. 2013. Just-In-Time Code Reuse: On the Effectiveness of Fine-Grained Address Space Layout Randomization. In *Proceedings of the 34th IEEE Symposium on Security and Privacy (S&P)*. San Francisco, CA, 574–588. https://doi.org/10.1109/SP.2013.45

[16] Raoul Strackx, Yves Younan, Pieter Philippaerts, Frank Piessens, Sven Lachmund, and Thomas Walter. 2009. Breaking the Memory Secrecy Assumption. In *Proceedings of the 2nd European Workshop on System Security (EUROSEC)*. Nuremburg, Germany, 1–8. https://doi.org/10.1145/1519144.1519145

[17] The PaX Team. [n.d.]. ASLR. http://pax.grsecurity.net/docs/aslr.txt. [Online; accessed 11-March-2019].

[18] Minh Tran, Mark Etheridge, Tyler Bletsch, Xuxian Jiang, Vincent Freeh, and Peng Ning. 2011. On the Expressiveness of Return-into-libc Attacks. In *Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection (RAID)*. Menlo Park, CA, 121–141. https://doi.org/10.1007/978-3-642-23644-0_7