The Mithril Coat: Uncovering the Hidden Potential of Selective Memory Protection

***Abstract*—**

I. Introduction

As often in history, prophecies eventually become reality; over the past three decades, data-oriented attacks [1] evolved from theoretical assumptions [2] to serious threats [3]–[8]. Therefore, the everlasting race between attackers and defenders continues. In the past, we have witnessed effective security mechanisms that have urged attackers to investigate new directions and exploit insufficiently explored corners of the system. Similarly, recent advances in *Control-Flow Integrity* (CFI) [9]–[13], *Code-Pointer Integrity* (CPI) [14], [15], and code diversification [16]–[18] significantly raised the bar for code-reuse attacks. In fact, CFI mechanisms were adopted by Microsoft [19], Google [20], and LLVM [21] forcing attackers to explore the uncharted world of data-oriented intrusions.

Generally, code-reuse attacks chain short code-sequences, *gadgets*, to hijack the application's control-flow. It is sufficient to overwrite one control-flow structure, such as a function pointer or a return address on the stack, with the start of the crafted gadget chain, to cause a target application to perform arbitrary computation. In contrast, data-oriented attacks completely avoid changes to the control-flow. Instead, this class of attacks aims at modifying *non-control data* to cause the application to obey the attacker's intentions [6]–[8]. Typically, an attacker leverages memory corruption vulnerabilities that enable arbitrary *read* or *write primitives* to take control over the application's data. By stitching a chain of *data-oriented gadgets* that operate on the modified data allows the attacker either to disclose sensitive information or to escalate privileges without violating the application's control-flow. In this way, data-oriented attacks remain under the radar despite the presence of code-reuse mitigation techniques and can lead to disastrous consequences [4]. We anticipate further growth in this direction in the near future and emphasize the need for practical primitives that eliminate such threats beforehand.

Researchers suggested different strategies to counter data-oriented attacks. For instance, *Data-Flow Integrity* (DFI) [22] mechanisms dynamically track the binary's data-flow. On the other hand, by introducing memory-safety to the C/C++ programming language it becomes possible to completely eliminate memory corruption vulnerabilities [23], [24]. While both directions have the potential to thwart data-oriented attacks, they lack practicality due to high performance overhead or suffer from compatibility issues with legacy code. Instead of enforcing the integrity of the data-flow, researchers started to explore isolation techniques that govern access to sensitive code and data regions [25]–[27]. Yet, they are either limited to user space, focus on protecting only one specific data structure, or rely on policies managed by the hypervisor.

Softbound, CETS, Cling

In this paper, we explore the potential of modern virtualization extensions of the Intel architecture to establish *selective memory protection primitives* that have the capability of thwarting data-oriented attacks. Instead of equipping the hypervisor with semantic knowledge required to enforce memory isolation, we take advantage of the Extended Page Table *(EPT) pointer* (EPTP) switching capability on Intel to manage different views on the guest's physical memory from inside *Virtual Machine*s (VMs), without any hypervisor interaction. For this, we extend Xen `altp2m` [28], [29] and the Linux memory management system to establish primitives that can be applied to selected, sensitive data structures in user and kernel space. In other words, we encapsulate sensitive data in disjoint protection domains that are not subject to limited access permissions of the x86 memory management unit; [1] a strong attacker with arbitrary *read* and *write* primitives to memory, cannot access the fortified data without first having to enter the corresponding protection domain. Further, we equip pointers to sensitive data in protection domains with authentication codes, whose integrity is bound to a specific context. This way, we protect pointers from illegal modifications and hence obstruct data-oriented attacks targeting the fortified data.

We apply our primitives to two sensitive kernel data structures that are vital for the system security, yet often disregarded by defense mechanisms: *page tables* and *process credentials*. Besides, we demonstrate their ease of applicability by guarding sensitive data in common, security-critical user space libraries and applications. For all cases, we evaluate the performance and security of our primitives. We believe that our work introduces a powerful means that brings us closer towards winning the fight against data-oriented attacks.

In summary, we make the following main contributions:

- We use Intel's EPTP Switching and Xen `altp2m` to control different guest physical memory views to encapsulate sensitive data in *disjoint protection domains*.
- We extend the Linux kernel to introduce in-guest *memory protection primitives* to fortify arbitrary data structures against data-oriented attacks in user and kernel space.
- We apply our primitives to guard *page tables* and *process credentials* on Linux, as well as chosen sensitive user space components with minimal performance overhead.

[1] In this paper, we refer to both x86 and x86-64 as the x86 architecture.

REFERENCES

[1] L. Cheng, H. Liljestrand, T. Nyman, Y. T. Lee, D. Yao, T. Jaeger, and N. Asokan, "Exploitation Techniques and Defenses for Data-Oriented Attacks," *arXiv preprint arXiv:1902.08359v2*, 2019.

[2] W. D. Young and J. McHugh, "Coding for a Believable Specification to Implementation Mapping," in *IEEE Symposium on Security and Privacy (S&P)*, 1987.

[3] S. Chen, J. Xu, E. C. Sezer, P. Gauriar, and R. K. Iyer, "Non-Control-Data Attacks Are Realistic Threats," in *USENIX Security Symposium*, 2005.

[4] Synopsys, "The Heartbleed Bug," http://heartbleed.com/, 4 2014.

[5] N. Carlini, A. Barresi, M. Payer, D. Wagner, and T. R. Gross, "Control-Flow Bending: On the Effectiveness of Control-Flow Integrity," in *USENIX Security Symposium*, 2015.

[6] H. Hu, S. Shinde, S. Adrian, Z. L. Chua, P. Saxena, and Z. Liang, "Data-oriented programming: On the expressiveness of non-control data attacks," in *IEEE Symposium on Security and Privacy (S&P)*, 2016.

[7] B. Sun, C. Xu, and S. Chong, "The Power of Data-Oriented Attacks: Bypassing Memory Mitigation Using Data-Only Exploitation Technique, Part I," *Black Hat, Asia*, 2017.

[8] K. Ispoglou, B. AlBassam, T. Jaeger, and M. Payer, "Block Oriented Programming: Automating Data-Only Attacks," *arXiv preprint arXiv:1805.04767*, 2018.

[9] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti, "Control-Flow Integrity," in *ACM Conference on Computer and Communications Security (CCS)*, 2005.

[10] M. Backes, T. Holz, B. Kollenda, P. Koppe, S. Nürnberger, and J. Pewny, "You Can Run but You Can'T Read: Preventing Disclosure Exploits in Executable Code," in *ACM Conference on Computer and Communications Security (CCS)*, 2014.

[11] Y. Cheng, Z. Zhou, Y. Miao, X. Ding, and R. H. Deng, "ROPecker: A generic and practical approach for defending against ROP attack," in *ISOC Network and Distributed System Security Symposium (NDSS)*, 2014.

[12] A. J. Mashtizadeh, A. Bittau, D. Boneh, and D. Mazières, "CCFI: Cryptographically Enforced Control Flow Integrity," in *ACM Conference on Computer and Communications Security (CCS)*, 2015.

[13] J. Werner, G. Baltas, R. Dallara, N. Otterness, K. Z. Snow, F. Monrose, and M. Polychronakis, "No-Execute-After-Read: Preventing Code Disclosure in Commodity Software," in *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2016.

[14] V. Kuznetsov, L. Szekeres, M. Payer, G. Candea, R. Sekar, and D. Song, "Code-Pointer Integrity," in *USENIX Symposium on Operating System Design and Implementation (OSDI)*, 2014.

[15] P. Zieris and J. Horsch, "A Leak-Resilient Dual Stack Scheme for Backward-Edge Control-Flow Integrity," in *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2018.

[16] P. Team, "Address Space Layout Randomization," http://pax.grsecurity.net/docs/aslr.txt, 2003.

[17] S. Crane, C. Liebchen, A. Homescu, L. Davi, P. Larsen, A.-R. Sadeghi, S. Brunthaler, and M. Franz, "Readactor: Practical Code Randomization Resilient to Memory Disclosure," in *IEEE Symposium on Security and Privacy (S&P)*, 2015.

[18] D. Bigelow, T. Hobson, R. Rudd, W. Streilein, and H. Okhravi, "Timely Rerandomization for Mitigating Memory Disclosures," in *ACM Conference on Computer and Communications Security (CCS)*, 2015.

[19] Microsoft, "Control Flow Guard," https://docs.microsoft.com/en-us/windows/desktop/SecBP/control-flow-guard, 10 2018.

[20] Google, "The Chromium Projects," https://www.chromium.org/developers/testing/control-flow-integrity, 10 2018.

[21] LLVM, "Control Flow Integrity," http://clang.llvm.org/docs/ControlFlowIntegrity.html, 10 2018.

[22] M. Castro, M. Costa, and T. Harris, "Securing Software by Enforcing Data-Flow Integrity," in *USENIX Symposium on Operating System Design and Implementation (OSDI)*, 2006.

[23] G. C. Necula, S. McPeak, and W. Weimer, "CCured: Type-Safe Retrofitting of Legacy Code," in *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 2002.

[24] T. Jim, J. G. Morrisett, D. Grossman, M. W. Hicks, J. Cheney, and Y. Wang, "Cyclone: A Safe Dialect of C," in *USENIX Annual Technical Conference*, 2002.

[25] Y. Liu, T. Zhou, K. Chen, H. Chen, and Y. Xia, "Thwarting Memory Disclosure with Efficient Hypervisor-Enforced Intra-Domain Isolation," in *ACM Conference on Computer and Communications Security (CCS)*, 2015.

[26] K. Koning, X. Chen, H. Bos, C. Giuffrida, and E. Athanasopoulos, "No Need to Hide: Protecting Safe Regions on Commodity Hardware," in *ACM European Conference on Computer Systems (EuroSys)*, 2017.

[27] Q. Chen, A. M. Azab, G. Ganesh, and P. Ning, "PrivWatcher: Non-bypassable Monitoring and Protection of Process Credentials from Memory Corruption Attacks," in *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2017.

[28] T. K. Lengyel, S. Maresca, B. D. Payne, G. D. Webster, S. Vogl, and A. Kiayias, "Scalability, Fidelity and Stealth in the DRAKVUF Dynamic Malware Analysis System," in *Annual Computer Security Applications Conference (ACSAC)*, 2014.

[29] S. Proskurin, T. Lengyel, M. Momeu, C. Eckert, and A. Zarras, "Hiding in the Shadows: Empowering ARM for Stealthy Virtual Machine Introspection," in *Annual Computer Security Applications Conference (ACSAC)*, 2018.