Control-Flow Hijacking: Are We Making Progress?

Mathias Payer
Purdue University
mathias.payer@nebelwelt.net

ABSTRACT

Memory corruption errors in C/C++ programs remain the most common source of security vulnerabilities in today's systems. Over the last 10+ years the security community developed several defenses [4]. *Data Execution Prevention* (DEP) protects against *code injection* – eradicating this attack vector. Yet, control-flow hijacking and code reuse remain challenging despite wide deployment of *Address Space Layout Randomization* (ASLR) and *stack canaries*. These defenses are probabilistic and rely on information hiding.

The deployed defenses complicate attacks, yet control-flow hijack attacks (redirecting execution to a location that would not be reached in a benign execution) are still prevalent. Attacks reuse existing gadgets (short sequences of code), often leveraging information disclosures to learn the location of the desired gadgets. Strong defense mechanisms have not yet been widely deployed due to (i) the time it takes to roll out a security mechanism, (ii) incompatibility with specific features, and (iii) performance overhead. In the meantime, only a set of low-overhead but incomplete mitigations has been deployed in practice.

Control-Flow Integrity (CFI) [1, 2] and Code-Pointer Integrity (CPI) [3] are two promising upcoming defense mechanisms, protecting against control-flow hijacking. CFI guarantees that the runtime control flow follows the statically determined control-flow graph. An attacker may reuse any of the valid transitions at any control-flow transfer. We compare a broad range of CFI mechanisms using a unified nomenclature based on (i) a qualitative discussion of the conceptual security guarantees, (ii) a quantitative security evaluation, and (iii) an empirical evaluation of their performance in the same test environment. For each mechanism, we evaluate (i) protected types of control-flow transfers, (ii) the precision of the protection for forward and backward edges. For open-source compiler-based implementations, we additionally evaluate (iii) the generated equivalence classes and target sets, and (iv) the runtime performance. CPI on the other hand is a dynamic property that enforces selective memory safety through bounds checks for code pointers by separating code pointers from regular data.

KEYWORDS

Memory Safety; Control-Flow Hijacking; Control-Flow Integrity; Return-Oriented Programming

ASIA CCS '17 April 02-06, 2017, Abu Dhabi, United Arab Emirates
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-4944-4/17/04.
DOI: http://dx.doi.org/10.1145/3052973.3056127

BIOGRAPHY

Mathias Payer is a security researcher and an assistant professor in computer science at Purdue university, leading the HexHive group. His research focuses on protecting applications in the presence of vulnerabilities, with a focus on memory corruption. He is interested in system security, binary exploitation, software-based fault isolation, binary translation/recompilation, and (application) virtualization.

Before joining Purdue in 2014 he spent two years as PostDoc in Dawn Song's BitBlaze group at UC Berkeley. He graduated from ETH Zurich with a Dr. sc. ETH in 2012, focusing on low-level binary translation and security. He analyzed different exploit techniques and wondered how we can enforce integrity for a subset of data (e.g., code pointers). All prototype implementations are open-source. In 2014, he founded the b01lers Purdue CTF team.

This material is based upon work supported, in part, by the National Science Foundation under Grant No. CNS-1513783, CNS-1464155, and CNS-1657711.

REFERENCES

[1] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti. Control-flow integrity. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS '05, 2005.

[2] N. Burow, S. A. Carr, J. Nash, P. Larsen, M. Franz, S. Brunthaler, and M. Payer. Control-Flow Integrity: Precision, Security, and Performance. *ACM Computing Surveys*, 50(1), 2018, preprint: https://arxiv.org/abs/1602.04056.

[3] V. Kuzentsov, M. Payer, L. Szekeres, G. Candea, D. Song, and R. Sekar. Code Pointer Integrity. In *OSDI: Symp. on Operating Systems Design and Implementation*, 2014.

[4] L. Szekeres, M. Payer, L. Wei, D. Song, and R. Sekar. Eternal war in memory. *IEEE Security and Privacy Magazine*, 2014.