Poster: Automatic Identification and Protection of Memory-resident Sensitive Data to Defend Against Data-Oriented Attacks

Salman Ahmed
Department of Computer Science
Virginia Tech
Blacksburg, VA, USA
ahmedms@vt.edu

Hans Liljestrand, N. Asokan
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada
hans@liljestrand.dev, asokan@acm.org

Danfeng (Daphne) Yao
Department of Computer Science
Virginia Tech
Blacksburg, VA, USA
danfeng@vt.edu

***Abstract*—Software and hardware-based countermeasures for protecting memory-resident data to prevent data-oriented attacks suffer from high performance overhead due to a large number of memory data objects and their pointers. In this ongoing work, we propose a framework utilizing rule-based heuristics to identify sensitive memory data and pointers automatically from an application and protect those sensitive data and pointers utilizing existing countermeasures. Our evaluation suggests that an application contains as low as 3% sensitive data and needs to protect less than 30% of its total data and pointers, on average. Besides, our preliminary result shows that this prioritized protection reduces the performance overhead of existing countermeasures by 50%.**

With the advances toward practical code pointer protection countermeasures and practical Control-Flow Integrity (CFI), we anticipate a shift towards the manipulation of memory-resident sensitive data or pointers as the attack vectors. In recent research, we observe an uptick in Data-Oriented Attacks (DOAs), also known as non-control attacks [12]–[14], [16], [23], [24], [28] even though DOAs were introduced more than a decade ago [6]. DOAs conform to CFI and manipulates of memory-resident sensitive data or their pointers. Ideally, DOAs [6], [12], [14] can modify all kinds of memory data to change program behavior for leaking sensitive information [3] or performing privilege escalations [8]. But the corruption of data pointers [7] is often desirable. For example, the manipulation of data pointers can lead to the leak of critical information about an application's address space layout [10], [25], gadget stitching in Data-Oriented Programming-based attacks [13], stack-based exploitations [6], and heap-based exploitations [26].

Researchers have proposed both software and hardware-based countermeasures to stop attackers from manipulating memory-resident data or their pointers. However, software-based countermeasures such as Data-Flow-Integrity (DFI) [5], Data Space Randomization (DSR) [1], [4], [22], and memory tagging [17], [18] usually suffer from performance overhead (48-116% [17], [18]) due to inter-procedural DFI, encryption, and masking. On the other hand, hardware-based countermeasures (e.g., HDFI [27], Intel's Control-Flow Enforcement Technology, ARM Pointer Authentication (PA), and Intel's Memory Protection Extensions (MPX)) are efficient, but in general, limited to one or a few platforms. Furthermore, the overhead is non-negligible. For example, ARM Pointer Authentication and Intel's MPX cost on average around 19.5% [11], [15] and 50% [19] overhead, respectively, for protecting data pointers.

The main reason for this runtime overhead is the huge number of data objects and pointers in an application, on average $\sim$100x compared to code pointers in an application. One solution for reducing this overhead is to identify the *sensitive* data objects and prioritize them for protection, rather than protecting all data objects. There are two approaches to identifying sensitive data. One approach is manual, and the other one is best effort semi-automatic. Prior work [12], [20], [21] have suggested the *manual* earmarking of sensitive data. However, manual earmarking is time-consuming and error-prone. A few best-effort semi-automated approaches [14], [16] can determine the criticality or sensitiveness of data. But these works require traces of data accesses, including traces for both normal and violating execution. As a result, these works are not scalable due to the need for huge and relevant execution and access traces. Besides, exercising all the violating execution paths is challenging. Furthermore, these techniques may not be application-agnostic and cannot work with existing countermeasures. Thus, there is a need for a scalable and platform- or application-agnostic automated approach for identifying and prioritizing sensitive data or their pointers.

In this ongoing work, we automate the identification and prioritization of sensitive data objects through our Data and Pointer Prioritization (DPP) framework. DPP uses shared and reusable vulnerability patterns to identify and prioritize sensitive data objects. These shared and reusable vulnerability patterns enable DPP to prevent unknown and future DOAs. DPP is also platform- or application-agnostic and adaptable with existing countermeasures. DPP uses rule-based heuristics to identify sensitive data objects.

We address two key challenges. First, it is challenging to find a good representative set of rules with comprehensive

coverage since DOA are constantly evolving. To address the challenge regarding the coverage and representativeness of rules, we extract the rules by breaking down exploits into common and reusable smaller vulnerability patterns. These common patterns are applicable to many exploits and future or unknown attacks. Second, it is also challenging to evaluate the accuracy of our rules. Because to the best of our knowledge, there exists no ground truth dataset of sensitive data objects, which we can use to evaluate DPP. To evaluate the accuracy and effectiveness of our rule-based heuristics, we construct the ground truths of 33 sensitive data objects from 18 programs, including five real-world applications, 13 relevant challenges from DARPA CGC, and ten representative test cases from the SAR dataset.

Our preliminary evaluation using manually constructed ground truths of vulnerable data objects or pointers by identifying vulnerable data objects or pointers from vulnerable datasets [2], [9] including 5 real-world applications shows that less than 30% of the data objects and their pointers are sensitive. Thus, in our testing environment, protecting less than 30% of total memory-resident sensitive data or their pointers is sufficient to protect the tested applications from data-orient attacks. Besides, the rule-based identification of sensitive memory data and pointers can lead to almost 50% performance improvements in existing defenses in our tested environments.

REFERENCES

[1] Sandeep Bhatkar and R. Sekar. Data space randomization. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 1–22. Springer, 2008.

[2] Paul E. Black. A software assurance reference dataset: Thousands of programs with known bugs. *Journal of Research of the National Institute of Standards and Technology*, 123:123005, April 2018.

[3] Heartbleed Bug. http://heartbleed.com, 2020. Accessed April 03, 2020.

[4] Cristian Cadar, Periklis Akritidis, Manuel Costa, Jean-Phillipe Martin, and Miguel Castro. Data randomization. Technical report, Technical Report TR-2008-120, Microsoft Research, 2008, 2008.

[5] Miguel Castro, Manuel Costa, and Tim Harris. Securing software by enforcing data-flow integrity. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 147–160. USENIX Association, 2006.

[6] Shuo Chen, Jun Xu, Emre Can Sezer, Prachi Gauriar, and Ravishankar K Iyer. Non-control-data attacks are realistic threats. In *USENIX Security Symposium*, volume 5, 2005.

[7] Crispin Cowan, Steve Beattie, John Johansen, and Perry Wagle. PointGuardTM: Protecting Pointers From Buffer Overflow Vulnerabilities. In *Proceedings of the 12th conference on USENIX Security Symposium*, volume 12, pages 91–104, 2003.

[8] Daniel Moghimi. Subverting without EIP. https://moghimi.org/blog/subverting-without-eip.html, 2014. Last accessed 6 January 2021.

[9] DARPA. Cyber grand challenge samples. https://github.com/CyberGrandChallenge/samples, 2022. Last accessed March 14, 2022.

[10] Isaac Evans, Sam Fingeret, Julian Gonzalez, Ulziibayar Otgonbaatar, Tiffany Tang, Howard Shrobe, Stelios Sidiroglou-Douskos, Martin Rinard, and Hamed Okhravi. Missing the point (er): On the effectiveness of code pointer integrity. In *2015 IEEE Symposium on Security and Privacy*, pages 781–796. IEEE, 2015.

[11] Reza Mirzazade Farkhani, Mansour Ahmadi, and Long Lu. {PTAuth}: Temporal memory safety via robust points-to authentication. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1037–1054, 2021.

[12] Hong Hu, Zheng Leong Chua, Sendroiu Adrian, Prateek Saxena, and Zhenkai Liang. Automatic generation of data-oriented exploits. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 177–192, 2015.

[13] Hong Hu, Shweta Shinde, Sendroiu Adrian, Zheng Leong Chua, Prateek Saxena, and Zhenkai Liang. Data-oriented programming: On the expressiveness of non-control data attacks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 969–986. IEEE, 2016.

[14] Yaoqi Jia, Zheng Leong Chua, Hong Hu, Shuo Chen, Prateek Saxena, and Zhenkai Liang. "the web/local" boundary is fuzzy: A security study of chrome's process-based sandboxing. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 791–804, 2016.

[15] Hans Liljestrand, Thomas Nyman, Kui Wang, Carlos Chinea Perez, Jan-Erik Ekberg, and N Asokan. {PAC} it up: Towards pointer integrity using {ARM} pointer authentication. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 177–194, 2019.

[16] Micah Morton, Jan Werner, Panagiotis Kintis, Kevin Snow, Manos Antonakakis, Michalis Polychronakis, and Fabian Monrose. Security risks in asynchronous web servers: When performance optimizations amplify the impact of data-oriented attacks. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 167–182. IEEE, 2018.

[17] Santosh Nagarakatte, Jianzhou Zhao, Milo MK Martin, and Steve Zdancewic. Softbound: Highly compatible and complete spatial memory safety for c. In *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 245–258, 2009.

[18] Santosh Nagarakatte, Jianzhou Zhao, Milo MK Martin, and Steve Zdancewic. Cets: compiler enforced temporal safety for c. In *Proceedings of the 2010 international symposium on Memory management*, pages 31–40, 2010.

[19] Oleksii Oleksenko, Dmitrii Kuvaiskii, Pramod Bhatotia, Pascal Felber, and Christof Fetzer. Intel mpx explained: A cross-layer analysis of the intel mpx system stack. *SIGMETRICS Perform. Eval. Rev.*, 46(1):111–112, June 2018.

[20] Tapti Palit, Jarin Firose Moon, Fabian Monrose, and Michalis Polychronakis. DynPTA: Combining static and dynamic analysis for practical selective data protection. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1919–1937, May 2021.

[21] Tapti Palit, Fabian Monrose, and Michalis Polychronakis. Mitigating data leakage by protecting memory-resident sensitive data. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 598–611, 2019.

[22] Prabhu Rajasekaran, Stephen Crane, David Gens, Yeoul Na, Stijn Volckaert, and Michael Franz. Codarr: Continuous data space randomization against data-only attacks. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 494–505, 2020.

[23] Roman Rogowski, Micah Morton, Forrest Li, Fabian Monrose, Kevin Z. Snow, and Michalis Polychronakis. Revisiting browser security in the modern era: New data-only attacks and defenses. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 366–381. IEEE, 2017.

[24] Cole Schlesinger, Karthik Pattabiraman, Nikhil Swamy, David Walker, and Benjamin Zorn. Modular protections against non-control data attacks. *Journal of Computer Security*, 22(5):699–742, 2014.

[25] Jeff Seibert, Hamed Okhravi, and Eric Söderström. Information leaks without memory disclosures: Remote side channel attacks on diversified code. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 54–65, 2014.

[26] Shellphish. Educational Heap Exploitation: how2heap . https://github.com/shellphish/how2heap, 2019. Last accessed 6 January 2021.

[27] Chengyu Song, Hyungon Moon, Monjur Alam, Insu Yun, Byoungyoung Lee, Taesoo Kim, Wenke Lee, and Yunheung Paek. Hdfi: Hardware-assisted data-flow isolation. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 1–17. IEEE, 2016.

[28] Laszlo Szekeres, Mathias Payer, Tao Wei, and Dawn Song. Sok: Eternal war in memory. In *2013 IEEE Symposium on Security and Privacy*, pages 48–62. IEEE, 2013.