Tutorial: Investigating Advanced Exploits for System Security Assurance

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

Long Cheng
School of Computing
Clemson University
Clemson, SC, USA
lcheng2@clemson.edu

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

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

Abstract—**Investigation of existing advanced exploits is crucial for system security assurance. One way to achieve system security assurance is through evaluating defenses using qualitative security metrics and accurate measurement methodologies. Analyzing existing exploit techniques can provide crucial insights about qualitative security metrics and measurement methodologies.**

In this tutorial, we investigate existing advanced exploit techniques by dividing the exploits into their constituent components. Our analyses focus on the impact of different defense techniques on the individual exploit components. These impact analyses provide insights for finding security metrics/methodologies as well as improving existing defenses. In this tutorial, we aim to focus on Return-Oriented Programming (ROP), Just-In-Time Return-Oriented Programming (JITROP), and Data-Oriented Attacks (DOAs). We aim to cover defenses such as fine-grained Address Space Layout Randomization (ASLR) and pointer protection techniques. More specifically, we aim to quantify the impact of fine-grained ASLR on different components of advanced ROP attacks. Besides, we will demonstrate a data-oriented exploit–an attack technique that circumvents currently deployed defenses–and explore defense techniques for defending against DOAs.

Through this tutorial, we aim to improve people's understanding and awareness of fundamental operating system security. The hands-on portion of the proposed tutorial will empower participants and researchers by providing knowledge on low-level security, application-level defenses, and security metrics/methodologies.

Index Terms—**ASLR, ROP, JITROP, DOP, Pointer Protection**

I. Control-Oriented Exploits and Defenses

ROP attack technique [1] first demonstrated by Hovav Shacham utilizes short instruction sequences called gadgets. The advantage of gadgets is that they are part of code binary and can perform various operations such as memory, assignment, arithmetic, logic, etc. These operations are called Turing-complete computations in literature [2]. One key defense strategy to defend against ROP attacks is to limit the access of gadgets, i.e., fewer operations can be performed. The PaX team introduced ASLR [3] to randomize the layout of a binary to limit the gadget lookup. However, attackers can reconstruct the binary layout in the presence of information leaks. Thus, fine-grained ASLR defenses (e.g., Zipr [4], Multi-compiler [5], Selfrando [6], CCR [7], and Shuffler [8]) randomize a binary layout with finer granularity than ASLR aiming to make the randomization resilient to information leaks and improve entropy. Entropy measures how well a defense can randomize a binary layout.

Snow et al. demonstrated such limitations of fine-grained randomizations under the JITROP [9] threat model using a technique called recursive code harvest. The recursive code harvest technique has the ability to discover new code pages (i.e., more gadgets) by leveraging control-flow transfer instructions, such as call and jmp from a single memory leak. It is important to note that stack buffer overflow, heap overflow, etc. serve as the initial gateways to enter other stages of such a sophisticated exploit.

This recursive code harvesting technique enables the opportunities to quantify the impact of fine-grained ASLR on different components of advanced ROP exploits. We aim to quantify two such impacts (i) gadget-lookup (i.e., collecting gadgets) and (ii) gadget-lookup times (i.e., collecting gadgets within a time-bound) using the Turing-complete [2] and MOV Turing-complete [10] gadget sets. Each of these gadget sets contains a set of gadgets for Turing-complete operations [11].

II. Data-Oriented Exploits and Defenses

Many researchers have shifted their focus from control-oriented attacks to data-oriented attacks (DOAs) in recent years [12]–[14] due to the unreliability of code-reuse attacks when CFI is enforced. Though DOAs [15] have been demonstrated years ago, such attacks have gained momentum in recent years. In DOAs, data pointer manipulation has become an appealing attack technique. Data pointer overwrites allow attackers to corrupt data pointers to point to arbitrary and unintended locations [16]. For example, data pointer manipulation can leak critical information about an application's address space layout [17]. Data-Oriented Programming (DOP) [12] requires the address of some non-control data pointers to accomplish DOP-based attacks. Chen et al. [15] demonstrated a DOA by corrupting a data pointer in the ghttpd HTTP server through a stack buffer overflow to bypass security checks of input strings. In summary, data pointer manipulation has become an attractive technique for DOAs. As a result, we observe pointer protection mechanisms such as ARM pointer authentication [18]. In this part of the tutorial, we aim to demonstrate a data-oriented exploit using pointer manipulation. We will also demonstrate how ARM pointer authentication can defend the exploit.

III. Format

Our tutorial includes a hands-on part on the control- and non-control exploits and their defenses. Specially, we aim to cover ROP, JITROP, and data-oriented exploits with fine-grained ASLR and ARM pointer authentication defenses. The tutorial is 90 minutes long. All the tools, exploits, and instructions will be packaged in a docker image with necessary libraries installed and will be released publicly on GitHub [1]. Attendees need to install the latest version of docker. We will structure the tutorial using the following four parts:

1) **Background on exploits and defenses**. The tutorial will start with short background information regarding the advancement of control- and non-control exploits, with an emphasis on advanced exploits such as ROP, JITROP, and DOP. We will end this short session by discussing two defense strategies, namely fine-grained ASLR and ARM pointer authentication.
2) **Hands-on on control-oriented exploits and defenses**. We will start this session with a basic ROP exploit. Attendees will be able to run the exploit in the docker container. Next, we will provide normal and hardened versions of a binary. Three fine-grained ASLR tools (i.e., function-level, basic block-level, and instruction-level) harden the binary and make three hardened versions of the binary. We will provide scripts for the attendees to run and observe looked-up gadgets and lookup times in numeric values. The numeric values will help attendees to compare the results regarding the impact of different levels of fine-grained ASLR's granularities.
3) **Exercise on code harvesting**. In this short session, we will encourage the audience to practice JITROP's code harvesting technique from different starting points. We will also provide a simple benchmark to compare the three ASLR techniques.
4) **Hands-on demonstration on non-control exploits and defenses**. In this session, we will demonstrate a data-oriented exploit and how the ARM pointer authentication [18] technique can prevent data-oriented exploits.

IV. Target Audience and Outcomes

We expect the audience to have a basic knowledge of running scripts. We will provide the scripts and commands. The skills of GDB commands would be a plus for a smooth hands-on experience. We recommend audience install the latest version of docker. This tutorial will achieve the following goals.

1) The participants will get hands-on experience on advanced control- and data-oriented exploits and various defense strategies.
2) They will learn how to explore advanced exploits by assessing the impact of various defenses.
3) They will also be encouraged in applying application-level defenses and strengthening low-level security understanding.

[1]https://github.com/salmanyam/tutorial-secdev-2021

V. Acknowledgement

This work has been partly supported by the National Science Foundation under Grant No. CNS-1929701.

References

[1] H. Shacham, "The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)," in *Proceedings of the 14th ACM conference on Computer and Communications Security*. ACM, 2007, pp. 552–561.

[2] R. Roemer, E. Buchanan, H. Shacham, and S. Savage, "Return-oriented programming: Systems, languages, and applications," *ACM Transactions on Information and System Security (TISSEC)*, vol. 15, no. 1, p. 2, 2012.

[3] P. Team, "Pax address space layout randomization (aslr)," 2003.

[4] W. H. Hawkins, J. D. Hiser, M. Co, A. Nguyen-Tuong, and J. W. Davidson, "Zipr: Efficient static binary rewriting for security," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2017, pp. 559–566.

[5] A. Homescu, S. Neisius, P. Larsen, S. Brunthaler, and M. Franz, "Profile-guided automated software diversity," in *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE Computer Society, 2013, pp. 1–11.

[6] M. Conti, S. Crane, T. Frassetto, A. Homescu, G. Koppen, P. Larsen, C. Liebchen, M. Perry, and A.-R. Sadeghi, "Selfrando: Securing the tor browser against de-anonymization exploits," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 4, pp. 454–469, 2016.

[7] H. Koo, Y. Chen, L. Lu, V. P. Kemerlis, and M. Polychronakis, "Compiler-assisted code randomization," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 461–477.

[8] D. Williams-King, G. Gobieski, K. Williams-King, J. P. Blake, X. Yuan, P. Colp, M. Zheng, V. P. Kemerlis, J. Yang, and W. Aiello, "Shuffler: Fast and deployable continuous code re-randomization," in *OSDI*, 2016, pp. 367–382.

[9] K. Z. Snow, F. Monrose, L. Davi, A. Dmitrienko, C. Liebchen, and A.-R. Sadeghi, "Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 574–588.

[10] S. Dolan, "mov is turing-complete," *Cl. Cam. Ac. Uk*, pp. 1–4, 2013.

[11] S. Ahmed, Y. Xiao, K. Z. Snow, G. Tan, F. Monrose, and D. Yao, "Methodologies for quantifying (re-) randomization security and timing under jit-rop," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1803–1820.

[12] 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 *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 969–986.

[13] K. K. Ispoglou, B. AlBassam, T. Jaeger, and M. Payer, "Block oriented programming: Automating data-only attacks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1868–1882.

[14] E. J. Schwartz, C. F. Cohen, J. S. Gennari, and S. M. Schwartz, "A generic technique for automatically finding defense-aware code reuse attacks," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1789–1801.

[15] S. Chen, J. Xu, E. C. Sezer, P. Gauriar, and R. K. Iyer, "Non-control-data attacks are realistic threats," in *USENIX Security Symposium*, vol. 5, 2005.

[16] C. Cowan, S. Beattie, J. Johansen, and P. Wagle, "Pointguardtm: Protecting pointers from buffer overflow vulnerabilities," in *Proceedings of the 12th conference on USENIX Security Symposium*, vol. 12, 2003, pp. 91–104.

[17] J. Seibert, H. Okhravi, and E. 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*, 2014, pp. 54–65.

[18] Qualcomm Technologies Inc., "Pointer Authentication on ARMv8.3," https://www.qualcomm.com/media/documents/files/whitepaper-pointer-authentication-on-armv8-3.pdf, 2017.