A Mathematical Modeling of Exploitations and Mitigation Techniques Using Set Theory

Rodrigo Branco
Intel Corporation
Hillsboro, Oregon, USA
rodrigo.branco@intel.com

Kekai Hu
Intel Corporation
Hillsboro, Oregon, USA
kekai.hu@intel.com

Henrique Kawakami
Intel Corporation
Hillsboro, Oregon, USA
henrique.kawakami@intel.com

Ke Sun
Intel Corporation
Hillsboro, Oregon, USA
ke.sun@intel.com

***Abstract*—One of the most challenging problems in computer security is formalization of vulnerabilities, exploits, mitigations and their relationship. In spite of various existing researches and theories, a mathematical model that can be used to quantitatively represent and analyze exploit complexity and mitigation effectiveness is still in absence.**

In this work, we introduce a novel way of modeling exploits and mitigation techniques with mathematical concepts from set theory and big O notation. The proposed model establishes formulaic relationships between exploit primitives and exploit objectives, and enables the quantitative evaluation of vulnerabilities and security features in a system. We demonstrate the application of this model with two real world mitigation techniques. It serves as the first step toward a comprehensive mathematical understanding and modeling of exploitations and mitigations, which will largely benefit and facilitate the practice of system security assessment.

Keywords. Computer Security, Language-theoretic Security, Exploit Modeling, Mitigation Effectiveness, Set Theory

I. Introduction

Just as the history of spears and shields, exploits and mitigations have been evolving competitively and interactively since the very beginning of computer security. The forms, types and approaches of exploits have expanded extensively from the simple classic stack overflow to more diverse and advanced ones like return-oriented programing (ROP) and call/jump-oriented programing (COP/JOP). While on the side of defense, mitigation techniques have also been developed and introduced with increasing quantity and sophistication: from the early-age stack canary and data execution prevention (DEP) to the more recent ones such as control flow guard (CFG) [1] by Microsoft, reuse attack protector (RAP) [2] by Grsecurity, and the control-flow enforcement technology (CET) [3] by Intel.

With the blowing up of the diversity and complexity of exploits and mitigations, it becomes more and more challenging to accurately describe the essence of a specific exploit and comprehensively evaluate the effectiveness of a certain mitigation with respect to the exploit. It remains unfinished work in the security field to establish a generic modeling for both exploits and mitigations to distill their essential aspects in a standardized manner so that the assessment can be made more straightforward, accurate, and consistent across different cases. Although there are well documented and categorized records such as common weakness enumeration (CWE) [4], such classifications are generally more descriptive and lacking of the necessary abstraction to extract the core essences from the superficial properties of an exploit.

A widely known and accepted concept on exploits in the security community is *weird machine* [5]. It clearly describes how a exploit happens and successfully predicts the bypassing of many existing exploitation countermeasures such as the control flow integrity (CFI) technologies [2], [6], [7]. While the definition of a *weird machine* pioneered the formalization of exploitations, more works are left blank on mathematical modeling and quantitatively analyzing exploitabilities of security vulnerabilities and the effectiveness of mitigation technologies.

In this work, we propose a mathematical modeling on formulating exploitations. The main contributions include:

1) Formal definitions of exploitation and mitigation. The notation and logic of set theory is applied to construct the representation of exploits and mitigations, with their primitives and attributes defined as set members and grouped as sub sets.
2) Apply the big O notation to mathematically describe the complexity of exploitations and the effectiveness of mitigation technologies.
3) Lay down the foundation of a general and practical modeling approach to map exploits and mitigations to an abstracted representation that accurately captures their essential properties, which can standardize and facilitate exploit-related narratives in security researches and practices.

II. Related Work

Prior to this work, there have been multiple related efforts in security research trying to model and theorize exploits and system states or behavior. The work by Sergey Bratus et al of Langsec [5], [8] introduces the concept of weird machine and considers exploits as constructive proofs of the presence of a weird machine. It uses this abstracted computational model to describe the famous examples in the history of exploits.

The work by Thomas Dullien [9], [10] considers a computer program as a finite state machine and points out the essence of exploitation is setting up, instantiating, and programming the weird machine. It discusses the importance of program

implementation and provides a theoretical understanding to distinguish unexploitable programs from exploitable ones.

In another work by Julien Vanegue on heap-related exploit modeling [11], more detailed formal definitions are introduced to describe heap primitives and behaviors.

Despite the great value in these researches per se, they have yet to provide a general and actionable approach to map exploits and mitigations to a standard abstract representation, which will be covered by the scope of this work and discussed in detail in the following sections.

III. Exploit Modeling

In this section, we illustrate our set theory based mathematical modeling of security vulnerability and exploit.

A. Definitions and Terminologies

When a system has a security vulnerability, which is a flaw that can potentially undermine the system security [12], an attacker may be able to take advantage of it to maliciously manipulate the system. The process of an attacker manipulating a vulnerability is called exploit (verb). The word exploit as a noun is also used to refer a piece of software or a chunk of data or a sequence of commands that an attacker creates to make use of a security vulnerability. [13]

Not all of the security vulnerabilities can be used by an attacker to successfully launch an attack(i.e. they may not be exploitable). As one of the most challenging problems in exploit research, evaluating whether a vulnerability is exploitable depends on multiple conditions including but not limited to: the goal of an attacker, the ability provided by the vulnerability, the system status, and the mitigation implemented in the system, etc. In order to formalize the abstract representation of system properties, exploits and mitigations, a series of standard terms are defined here.

Definition III.1. An exploit primitive (EP) is an attack ability that an attacker can potentially achieve from a security vulnerability.

Each exploit primitive is composed of two elements: `type` and `property`. As suggested by their names, the `type` primitive identifies the type of an EP which can be read, write or execute, while the `property` primitive further describes the attack abilities associated with an EP type, such as location, timing, repeatability, etc. We use five major primitive properties in this model (as defined in [2]): arbitrary addresses (AA), arbitrary content (AC), arbitrary operation (AO), arbitrary number of times (AN) and at arbitrary time (AT).

All the security vulnerabilities and exploit technologies can be abstracted to exploit primitive representations. For example, a classical stack-based buffer overflow in a system with no protection can be represented with two exploit primitives: one write primitive with multiple bits on the stack and one execute primitive with arbitrary location on the stack.

Definition III.2. An exploit objective (EO) is the final goal that an attacker wants to achieve in a vulnerable system.

In a vulnerable system, an exploit objective can vary significantly based on different goals of different attacks. It can be either as simple as memory leakage of a few memory bits or as complex as remote code execution.

With an exploit objective defined in a vulnerable system, the next step is to distinguish the exploitable vulnerabilities from the non-exploitable ones. Exploit condition is defined for this purpose.

Definition III.3. An exploit condition (EC) is the minimal required combination of exploit primitives in a vulnerable system to make this system exploitable to an exploit objective.

Exploit condition has three fundamental attributes:

- An exploit condition is always associated with an exploit objective, i.e., an EC should not be considered as a fixed condition for all the systems in all scenarios, instead, it should be carefully specified with respect to an EO. For example, with an EO of information leakage, a read primitive with multiple bits at arbitrary location is definitely considered as exploitable while the same read primitive in the same system without other exploit primitives should be considered non-exploitable for an EO of remote code execution.
- An exploit condition is the minimal requirement of an exploit. Thus, if any primitive in an EC is removed, this EC is not exploitable any more.
- One exploit objective can have many different exploit conditions. If and only if any of these ECs is met, the EO is exploitable. Take information leak as an example, both an EC with an arbitrary read primitive and an EC with a write and an execute primitives in the memory location that has read access to the target information would give the attacker the capability to steal the secret. Any one of these two ECs makes the EO information leak exploitable.

Definition III.4. An exploit complexity or exploit difficulty (ED) applies big O notation to quantitatively describe the upper bound on the growth rate of the time and cost for the attackers to exploit an exploit condition.

In a vulnerable system with no protection and no mitigation implementation, any EP is considered as O(1) complexity. By adding mitigation to the system, different levels of EDs are introduced to corresponding EPs. We will further elaborate this in Section IV.

B. Set Representation of Exploits

As defined in Definition III.1, an EP has two elements: `type` and `property`. We use a set T to denote all the EP types and a set P to denote all the EP properties as follow:

$$T = \{All\ EP\ types\} = \{Read, Write, Execute\} \quad (1)$$

$$P = \{All\ EP\ properties\} = \{AA, AC, AO, AN, AT\} \quad (2)$$

Thus, an EP can be represented as a combination of a type $t \in T$ and a property $p \in P$.

$$ep = \{t,\ p\} \tag{3}$$

With EP defined, an EC is a set of EPs.

$$\begin{aligned} ec &= \{ep_1, ep_2, \ldots, ep_n\} \\ &= \{\{t_1, p_1\}, \{t_2, p_2\}, \ldots, \{t_n, p_n\}\} \end{aligned} \tag{4}$$

Depending on the number of security vulnerabilities and their capabilities in a system, the exploitability E of a defined EO in this system can be represented as a set of all the possible ECs.

$$E = \{ec_1, ec_2, \ldots, ec_n\} \tag{5}$$

With all the exploit primitives that an attacker have in the system, for a certain exploit objective, if any of the exploit conditions is met, i.e., $E \neq \emptyset$, this EO is exploitable. To protect an EO to be exploited, a mitigation need to block all the possible ECs in the system. Otherwise, the mitigation doesn't fully protect the system and can be bypassed. If and only if $E = \emptyset$, an EO is not exploitable.

IV. Mitigation Modeling

In this section, we introduce the set theory and big O notation based modeling to abstract mitigations. This way, the mathematical mitigation modeling is standardized with the exploits, and quantitatively analysis of mitigation effectiveness is feasible.

A. Probabilistic and Deterministic Mitigations

In order to eliminate as much as possible the exploitability of unknown security vulnerabilities in real world systems, innumerous exploit mitigation techniques are designed and implemented in a variety of computing environments. After adopting one or more mitigations in a system, the system designer intends to protect one or more EOs to be exploited by removing the attacker's abilities in the system, i.e., exploit primitives. If one or more EPs are removed from an EC by a mitigation, this EC will not meet. For a specific EO if all of its ECs have at least one EP removed from them, i.e., for all the $ec_i \in E, i = 1, 2, \ldots, n$, there is an $ep_j = 0, where\ ep_j \in ec_i$, this EO will be non-exploitable with $E = \emptyset$.

We propose a new taxonomy to classify these exploit mitigation techniques into two types: probabilistic mitigations and deterministic mitigations. The classification is based on their probabilities of preventing the exploitability of their target security vulnerabilities, as explained below.

Definition IV.1. A deterministic mitigation (DM) technique eliminates the exploitability of its target exploit primitives completely or with a very high probability.

In other words, a deterministic mitigation technique not only rules out its target EPs from an EO but it also cannot be cracked, or the chance of breaking it is negligible. The NX bit, which is a technology used in CPUs to mark certain memory areas not executable, is an example of a deterministic mitigation. It completely blocks the EPs of execute type in the marked memory ranges and malicious software cant break it since it is enforced at the CPU level.

Definition IV.2. A probabilistic mitigation (PM) is a mitigation that significantly increases the ED and reduces the successful rate of its target EPs, although it does not completely eliminate the EPs.

In our proposed mathematical model, a probabilistic mitigation increases $O(n)$ exploit complexity to its target complexity. A typical example of a probabilistic mitigation is stack canary. Although it can still be bypassed in many specific cases and does not fully remove the exploitability of stack based buffer overflows, it significantly reduces the possibility of exploiting a stack based buffer overflow.

Overall, a deterministic mitigation adds $O(\infty)$ exploit complexity to a certain EP thus completely blocks it while a probabilistic mitigation add $O(n)$ exploit complexity to its target EP and make it more difficult to be exploited.

B. Set Representation of Mitigation

From the security perspective, a computer system includes a set of valid mitigation technologies, each of which is either a deterministic mitigation or a probabilistic mitigation. An empty set of mitigations means the system has no protection. When a mitigation is applied to a system, it removes or reduces some of the attacker's abilities (EPs), but other EPs might not be affected at all. To generalize these ED effects, we consider that a mitigation always adds exploit difficulties to all the EPs in our model. For the target EPs of the mitigation, it adds $O(n)$ or $O(\infty)$ depending on whether the mitigation is a PM or DM to this EP. On the other hand, for the ineffective ones, it doesn't add any complexity, and $O(1)$ is used to represent that there is no complexity change. We use the new notation $ep' = ep(ed)$ to represent an exploit primitive with its exploit difficulty. In a system that has no protection, no ED is added to any of the EPs, $ep' = ep(1)$.

This way, a mitigation can be represented in a set of its EPs with the added EDs.

$$\begin{aligned} m = \{&ep'_1(ed_1), ep'_2(ed_2), \ldots, ep'_n(ed_n)\} \\ &where\ ed_i \in \{O(1), O(n), O(\infty)\} \end{aligned} \tag{6}$$

Now, when a mitigation is applied to a system, the EDs of the EPs in the system change, thus the ED of a certain EC will also change accordingly:

$$\begin{aligned} ec' &= ec + m \\ &= \{ep'_1(ed_1), ep'_2(ed_2), \ldots, ep'_n(ed_n)\} \\ &+ \{ep'_1(ed'_1), ep'_2(ed'_2), \ldots, ep'_n(ed'_n)\} \\ &= \{ep'_1(ed''_1), ep'_2(ed''_2), \ldots, ep'_n(ed''_n) \end{aligned} \tag{7}$$

In this operation, $ed_1, ed_2, \ldots, ed_n$ represent the EDs before the mitigation, $ed'_1, ed'_2, \ldots, ed'_n$ represent the EDs that are added by the mitigation, and $ed''_1, ed''_2, \ldots, ed''_n$ represent the EDs after the mitigation.

After applying a mitigation in a system, the new exploit condition ec has three possible cases:

1) The new exploit condition ec is not a valid exploit condition any more because at least one of its EPs are removed from the EC, i.e., for any of the $ep'_i \in ec'$ where $i = 1, 2, \ldots, n$, $ep'_i(ed''_i) = ep'_i(\infty)$.
2) The new exploit condition ec is still exploitable but with higher ED, i.e., for all of the $ep'_i \in ec'$ where $i = 1, 2, \ldots, n$, $ep'_i(ed''_i) \neq ep'_i(\infty)$ but some of the $ep'_i(ed''_i) > ep'_i(ed_i)$.
3) The new exploit condition ec is still exploitable with the same ED as before the mitigation, i.e., for all of the $ep'_i \in ec'$ where $i = 1, 2, \ldots, n$, $ep'_i(ed''_i) = ep'_i(ed_i)$.

Among these three cases, in contrast to the case 1 and 3 which are the straightforward yes or no cases, case 2 is the more complicated one depending on the number of EPs that have $O(n)$ ED. Take an EC that has three EPs as an example: assume that in a system that has no mitigation, all three EPs have $O(1)$ ED:

$$ec = \{ep'_1(1), ep'_2(1), ep'_3(1)\} \tag{8}$$

After implementing the mitigation, EDs for both ep_1 and ep_3 increase to $O(n)$:

$$ec = \{ep'_1(n), ep'_2(1), ep'_3(n)\} \tag{9}$$

For an attacker to achieve his EO, he needs to bypass the protection on both ep_1 and ep_3 with $O(n)$ more complexity each at the same time. In an other word, his exploit complexity increases to $O(n^2)$ eventually, noted as $ec'(n^2)$. With this example, we can see that the ED of ec in case 2 is $O(n^i)$ where i is the number of EPs that has $ep'_i(ed_i) = ep'_i(n)$.

Now, let's look at the exploitations and mitigations at the system level to see how this model can be applied to system security evaluation. Consider a vulnerable system where an attacker can have five different EPs that eventually meet three ECs for a certain EO. Assuming that there is no mitigation implemented in the initial state of the system, the exploitability of this EO in the system can be represented as:

$$E = \{ec_1(1), ec_2(1), ec_3(1)\} \tag{10}$$

Each EC is a combination of EPs, in this example, let's say ec_1 includes ep_1, ep_2, ec_2 includes ep_2, ep_3, ep_4, and ec_3 includes ep_1, ep_4, ep_5 as the following equation shows:

$$\begin{aligned} ec_1(1) &= \{ep'_1(1), ep'_2(1)\} \\ ec_2(1) &= \{ep'_2(1), ep'_3(1), ep'_4(1)\} \\ ec_3(1) &= \{ep'_1(1), ep'_4(1), ep'_5(1)\} \end{aligned} \tag{11}$$

When a mitigation that can probabilistically mitigate ep_2, ep_4 and deterministically mitigate ep_5 is implemented in the system, it introduces extra EDs to the EPs:

$$ec = \{ep'_1(1), ep'_2(n), ep'_3(1), ep'_4(n), ep'_5(\infty)\} \tag{12}$$

Thus, the EDs of the ECs change:

$$\begin{aligned} ec'_1(n) &= \{ep'_1(1), ep'_2(n)\} \\ ec'_2(n^2) &= \{ep'_2(n), ep'_3(1), ep'_4(n)\} \\ ec'_3(\infty) &= \{ep'_1(1), ep'_4(n), ep'_5(\infty)\} \end{aligned} \tag{13}$$

The overall exploitability E also changes:

$$E = \{ec'_1(n), ec'_2(n^2), ec'_3(\infty)\} \tag{14}$$

In the new system, we can see that although the mitigation completely mitigates ec_3 and increases the ED of ec_2 to $O(n^2)$, the exploit complexity of a certain EO is always depending on the EC that has minimum ED, in this case ec_1 with ED $O(n)$. Thus, the introduced mitigation is only a probabilistic mitigation to this EO with $O(n)$ efficiency.

V. From Theory to Application

With the set theory modeling and big O notation defined in previous sections, real-world mitigation techniques can be abstracted into mathematical forms and evaluated quantitatively for its impact on system security robustness, with regard to specific EOs, ECs and EPs. In this section, two classic mitigation technologies are used as examples, Control Flow Guard (CFG) by Microsoft and Control-flow Enforcement Technology (CET) by Intel, to demonstrate the application of the proposed model in evaluating effectiveness of mitigation techniques.

Control Flow Guard (CFG) is a mitigation technology to prevent control flow being redirected to unintended locations. It checks and validates if the target address of an indirect branch is a valid entry point before the branch can take place in the execution flow. In a system with CFG enabled, the number of legal target locations of an indirect branch is much smaller than an unprotected system. Thus, it is a probabilistic mitigation of call-oriented programming (COP) and jump-oriented programming (JOP) in the sense that, despite of not completely preventing such exploits, it largely reduces the availability of gadgets that can be used by COP and JOP.

In the form of set representation, for the EO of arbitrary code execution, COP and JOP need two basic EPs to meet an EC: overwrite the branch target address and execute at the target address, $ep_1 = Write, AC$ and $ep_2 = Exec, AA$.

$$ec(1) = \{ep_1(1), ep_2(1)\} \tag{15}$$

With CFG implemented, the arbitrary address in ep_2 is greatly reduced since only a small number of addresses are legal. Although CFG does not completely block ep_2, it makes it much harder and changes its ED from $O(1)$ to $O(n)$. Therefore, the ED of this EC also changes from $O(1)$ to $O(n)$, and CFG introduces $O(n)$ extra exploit complexity comparing to unmitigated case:

$$ec(n) = \{ep_1(1), ep_2(n)\} \tag{16}$$

Besides CFG, another control flow integrity mitigation is Control-flow Enforcement Technology (CET) which contains

two parts: indirect branch tracking (IBT) and shadow stack. The IBT part bears large similarity to CFG in terms of its purpose. It is a hardware-supported feature that inserts special labels to mark indirect branch targets as legal entry points and validates the label every time there is a indirect branch. Therefore IBT of CET is also a probabilistic mitigation of COP/JOP and has the same effectiveness $O(n)$ as CFG. Both of them greatly reduce the possible gadgets rather than completely blocking the exploits.

Considering a system with both CET and CFG enabled, both add $O(n)$ ED to ep_2, even if an attacker spends $O(n)$ effort to bypass one of these two technique, he still needs to deal with the other one. Thus, the ED of the system becomes $O(n^2)$:

$$ec(n^2) = \{ep_1(1), ep_2(n^2)\} \tag{17}$$

The second part of CET is the shadow stack (SS), which pushes and pops the return address into a hardware-controlled stack independently from the active stack used by the process. It mitigates return-oriented programming (ROP) by checking the return address on the process stack with the shadow stack every time when a return instruction needs to be executed. Within its security premises, SS can be considered as a deterministic mitigation for ROP (there could be exception cases such as implementation flaws that compromise partially or fully the mitigation, but in this work the mitigations are only considered by its designed figure of merits). In the proposed model, ROP must have an EP of overwrite the return address, $ep_1 = Write, AC$. With the mitigation of SS, the ED of this EP changes from $O(1)$ to $O(\infty)$, thus ROP attacks will be deterministically blocked:

$$ec(\infty) = \{ep_1(\infty), ep_2(1), \ldots, ep_n(1)\} \tag{18}$$

VI. Conclusion

In this paper, a set theory and big O notation based mathematical modeling of exploitations and mitigation techniques are introduced and assessed. The proposed model is designed to assist the evaluation of vulnerabilities and security features in a product, so that system designers can choose the optimal set of mitigations for a given set of exploits. On the other hand, the model can be also used by security analysts to find the optimal set of exploit primitives to achieve a given exploit goal. The current version of the model defines the theoretical framework and lay down the foundation for further elaboration and enrichment to make it more practical to be adopted in formal security analysis.

In summary, the use of this model can provide the following benefits: (1) Assist the evaluation of security features in a product, so that system designers can choose the optimal set of mitigations for a given set of exploits. (2) Find the optimal set of exploit primitives that are necessary to achieve a given exploit goal. (3) Provide a straightforward way of quantifying the impacts of a security-related bugs.

References

[1] Microsoft, "Control flow guard," 2015. [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/mt637065(v=vs.85).aspx

[2] P. Team, "Rap: Rip rop," 2015, hackers to Hackers Conference. [Online]. Available: https://pax.grsecurity.net/docs/PaXTeam-H2HC15-RAP-RIP-ROP.pdf

[3] Intel, "Control flow enforcement technology," 2016. [Online]. Available: https://software.intel.com/sites/default/files/managed/4d/2a/control-flow-enforcement-technology-preview.pdf

[4] MITRE, "Common weakness enumeration." [Online]. Available: https://cwe.mitre.org

[5] S. Bratus, M. Locasto, M. Patterson, L. Sassaman, and A. Shubina, "Exploit Programming: from Buffer Overflows to Weird Machines and Theory of Computation," *USENIX ;login:*, Dec. 2011. [Online]. Available: http://langsec.org/papers/Bratus.pdf

[6] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti, "Control-flow integrity," in *Proceedings of the 12th ACM Conference on Computer and Communications Security*, ser. CCS '05. New York, NY, USA: ACM, 2005, pp. 340–353. [Online]. Available: http://doi.acm.org/10.1145/1102120.1102165

[7] K. Hu, H. Chandrikakutty, R. Tessier, and T. Wolf, "Scalable hardware monitors to protect network processors from data plane attacks," in *2013 IEEE Conference on Communications and Network Security (CNS)*, Oct 2013, pp. 314–322.

[8] J. Bangert, S. Bratus, R. Shapiro, and S. W. Smith, "The page-fault weird machine: Lessons in instruction-less computation," in *Presented as part of the 7th USENIX Workshop on Offensive Technologies*. Washington, D.C.: USENIX, 2013. [Online]. Available: https://www.usenix.org/conference/woot13/workshop-program/presentation/Bangert

[9] T. F. Dullien, "Weird machines, exploitability, and provable unexploitability," *IEEE Transactions on Emerging Topics in Computing*, vol. PP, no. 99, pp. 1–1, 2017.

[10] ——, "Exploitation and state machines," 2011, infiltrate Offensive Security Conference. [Online]. Available: http://www.slideshare.net/scovetta/fundamentals-of-exploitationrevisited

[11] J. Vanegue, "The weird machines in proof-carrying code," in *2014 IEEE Security and Privacy Workshops*, May 2014, pp. 209–213.

[12] I. Arce, "On the quality of exploit code: An evaluation of publicly available exploit code," 2005, rSA Security Conference.

[13] C. Sarraute, "Automated attack planning," *CoRR*, vol. abs/1307.7808, 2013. [Online]. Available: http://arxiv.org/abs/1307.7808