POSTER: Extracting Anti-specifications from Vulnerabilities for Program Hardening

Md Salman Ahmed, Danfeng (Daphne) Yao
Computer Science, Virginia Tech
{ahmedms, danfeng@vt.edu}@vt.edu

Haipeng Cai
Electrical Engineering and Computer Science
Washington State University
haipeng.cai@wsu.edu

***Abstract*—Understanding the stages or patterns of an offensive exploit is very crucial for fixing security bugs in programs and developing defenses. However, this is a challenging task and often requires manual efforts. Thus, scalability is a major concern for this task. In this work, we present a technique to extract exploit patterns from vulnerable programs for hardening future programs by utilizing the extracted exploit patterns.**

The cat and mouse games between cyber attackers and defenders have been prevalent for decades. In recent years, attackers [3]–[5] proved their capabilities to bypass advanced defenses including but not limited to Address Space Layout Randomization (ASLR), Non-Executable (NX a.k.a. W⊕X), and Control Flow Integrity (CFI). Often, the software developers or security defenders understand the offensive exploit techniques demonstrated in their software and develop patches to harden those exploits. The understanding of the exploit techniques is not a straightforward task and requires manual efforts. However, the manual efforts are not scalable. One way to solve the problem is to extract the exploit techniques or patterns automatically and use these patterns to screen and harden future programs. We call the exploit patterns anti-specifications (or anti-specs in short) [6]. In simple words, specifications are what a program should do and anti-specs are what a program should not do. However, an anti-specification has an impact down the road in a program. For example, attackers can leverage an anti-specification to hijack the control flow of a program or leak information. An example of an anti-specification is unsanitized user input.

In this work, we develop a technique for extracting anti-specs from real-world programs and build an anti-specification database. This database is very useful. One of the uses of this database is to screen a new input program against the anti-specs from the database and harden the program to be immune to the attack vectors (if any) by automatically patching the program. However, the anti-specification extraction process poses several challenges including accuracy, prioritization, and scalability. Accuracy is required for preventing false alerts, prioritization for identifying risky anti-specs, and scalability for extracting anti-specs from millions of programs.

We design our approach by considering the challenges described above. Our approach works in four stages. In stage ①, we manually analyze 130+ vulnerabilities from 100+ vulnerable programs provided in The Defense Advanced Research Projects Agency (DARPA)'s Cyber Grand Challenge (CGC) [2]. In stage ②, we manually extract anti-specs. We generate new anti-specification knowledge using the existing anti-specs in stage ③. The purpose of stage ④ is to screen and harden new input programs utilizing the extracted anti-specs. However, stage ④ is out of the scope of this work.

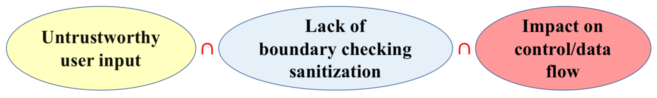
Untrustworthy user input
∩
Lack of boundary checking sanitization
∩
Impact on control/data flow

Fig. 1. Three-tuple form of an anti-specification

We generalize the anti-specification extraction process in stage ②. The generalized extraction process works in four steps. First, we identify the vulnerable function (provided by CGC). Second, we trace back and locate the entry point of the untrustworthy input. Third, we confirm that there is no boundary checking/sanitization issue between the entry point and the vulnerable function. Fourth, we identify the location of downstream impact. The downstream impact is two types: 1) impact on the control flow (e.g., controlling *eip*) and 2) impact on the data flow (similar to Heartbleed).

We extract an anti-specification for a vulnerability in a three-tuple form as illustrated in Figure 1. Each anti-specification covers untrustworthy user input (if any), lack of boundary-checking or sanitization of user input, and the impact of the lack of boundary-checking or sanitization on the control/data flow. The three components in the three-tuple form must be present in a vulnerability to be considered as an anti-specification.

We are currently conducting a static analysis using angr [1] for automatically extracting anti-specs. Through binary analysis with angr, we start by identifying callsites of the user input sources, check guards of the data retrieved at the callsites through dominance analysis (i.e., computing backward if there are any control dependencies of any reaching definitions at the callsite), and propagate the impact of unguarded user input access through forward data flow analysis (i.e., computing forward any reachable uses of the definitions at the callsite).

Acknowledgment

This work is supported by DARPA/ONR Grant N66001-17-C-4052.

References

[1] angr. http://angr.io/. Last accessed August 8, 2018.

[2] The defense advanced research projects agency (darpa). https://github.com/CyberGrandChallenge/samples. Last accessed August 8, 2018.

[3] A. Bittau, A. Belay, A. Mashtizadeh, D. Mazières, and D. Boneh. Hacking blind. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 227–242. IEEE, 2014.

[4] 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*, pages 161–176, 2015.

[5] 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 *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 574–588. IEEE, 2013.

[6] J. Vanegue. The automated exploitation grand challenge. https://openwall.info/wiki/_media/people/jvanegue/files/aegc_vanegue.pdf. Last accessed August 8, 2018.