SIENNA LOCOMOTIVE Progress Report 1

Andrew Ruef
Trail of Bits

October 14, 2015

1 Overview

Our goal is to create a system that, given a crashing input for a program, provides a comprehensible metric for the exploitability of the crash. We have reviewed existing research, met with the Army to kick the program off, and identified staff within Trail of Bits who will carry out the work.

2 Research

Our initial research has focused along two paths: what existing work has been done in an industrial setting, and what can a notion of **influence** do when judging the exploitability of crashes?

2.1 Industrial Research

We have identified existing literature from the field of crash analysis and severity ranking [3] [2]. This work is useful because it tells us where things currently are. Work from Matt Miller at Microsoft provides a useful road map for classifying different types of program faults when crashes have been found. Microsoft has significant investment into fuzzing and fuzzing architectures and they review crashes from multiple sources to assess severity when deciding how to allocate development resources. Understanding their process will take the system closer to best-in-class.

2.2 Influence

Information flow theory, an extension of taint tracking, can be used for fuzzing and crash triage. In information flow theory, data in the system can be given an integrity label based on data sensitivity. High integrity data is sensitive data in the system, while low integrity data is data introduced from outside the system. An analysis on traces can then search for violations of noninterference - when low integrity data interferes with high integrity data in a manner that can be observed by an attacker. This interference and subsequent observation

could result in the compromise of the data by the attacker. Information flow theory is directly applicable to finding and mitigating side-channel attacks in cryptosystems[4].
We are hopeful that this notion of influence can produce more actionable results from a fuzzing campaign. In an information assurance setting it is important to provide a notion of scale and severity for a given list of crashes and having a precise metric for the number of bits in a crashing state are under an attackers control could communicate urgency to development teams in a way that crashing inputs alone could not.

3 Execution

3.1 Existing tools

For an unrelated project we have adopted the open source symbolic execution system PySymEmu[1] to operate on mini-dump files produced by Windows debugging tools. We're hopeful that this infrastructure is useful when reasoning about the exploitability of a given crash.

3.2 Goals

Our goal by the end of the period of performance is to have a prototype that can take a crashing input or interaction sequence/script, and a dump of the crashing state as identified by the fuzzer, and produce a fine grained severity ranking for that crashing input/interaction. This ranking will hopefully incorporate the notion of influence discussed above but perhaps our research will discover another metric that is better.

3.3 Meetings

Andrew Ruef, Yan Ivnitskiy and Nicholas DePetrillo met with Army sponsors on date to discuss the program. From the Army side the meeting was joined by Joe Law, Roy Radhika, William Daddario, and Rex Johnson. After this meeting Andrew Ruef met with William Daddario and Rex Johnson to explore the existing system. This conversation was very useful in giving Trail of Bits a view of the needs of the Information Assurance group and their existing capabilities.

4 Programatics

We have identified the Trail of Bits staff that will perform the work on this project and introduced them to staff at the Army sponsor organization. Yan Ivnitskiy will begin work on the program within either the next reporting period or the following reporting period.

5 Future

Soon we hope to show a proof of concept from our PySymEmu tool where the tool can take a concrete input and, at a crashing condition, demonstrate the degree of freedom, represented as a constraint, in the pointer dereferenced at the crashing location. This work can begin when Yan Ivnitskiy becomes available.

References

[1] Pysymemu. `https://github.com/feliam/pysymemu`.

[2] Kratkiewicz, K., and Lippmann, R. A taxonomy of buffer overflows for evaluating static and dynamic software testing tools. In *Proceedings of Workshop on Software Security Assurance Tools, Techniques, and Metrics* (2006), vol. 500, p. 44.

[3] Miller, M. Modeling the exploitation and mitigation of memory safety vulnerabilities. `http://2012.ruxconbreakpoint.com/assets/Uploads/bpx/Modeling\%20the\%20exploitation\%20and\%20mitigation\%20of\%20memory\%20safety\%20vulnerabilities.pptx`, 2012.

[4] Molnar, D., Piotrowski, M., Schultz, D., and Wagner, D. The program counter security model: Automatic detection and removal of control-flow side channel attacks. In *Information Security and Cryptology-ICISC 2005*. Springer, 2006, pp. 156–168.