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.

From: Yan I <yan@trailofbits.com>
Subject: Automated Exploitability Reasoning (AER) Project Update
Date: December 1, 2015 at 12:52:49 PM EST
To: "Law, Joe CIV USARMY RDECOM (US)" <joe.law.civ@mail.mil>
Cc: Andrew Ruef <andrew@trailofbits.com>

Joe,

Since last week’s meeting fell on Thanksgiving, wanted to provide a brief update on the project via email.

We have been focusing on building a proof of concept for a system capable of tracking tainted input to a crashing condition. To that effect, we've been looking at a few directions: the implementation of '!exploitable' extension to WinDbg, and dynamic analysis platforms like Angr and Triton.

We are currently working on two open problems: A way to implement a subset of '!exploitable' rules that we extracted from its source in Triton (or Angr), and determine these tools' capability for taint analysis.

We are currently focusing on Linux binaries due to the amount of documentation and support for that platform, but the tools should be portable to Windows.

In terms of staffing, Andrew Ruef and myself (Yan Ivnitskiy) are the investigators on this project.

If you have any questions, please don't hesitate to get in touch.

Thanks!
Yan

Sienna Locomotive Status Report
Trail of Bits, Inc
November 2015, December 2015, January 2016
Prepared for: Joe Law

Since last report update, we first focused on selecting a platform to continue our work on. We then selected a combination of Triton and a custom-built application tracing auxiliary tool. Our goals for an analysis platform were two-fold: to implement or use a taint-tracking system to build further analysis on, and to implement existing state-of-the-art rules.

Taint tracking is a powerful tool in determining the magnitude of influence a program input has in causing a crash and what potential security repercussions that crashing condition can have. This approach can determine the severity of a crash with more precision and a less false positive rate than current state of the art.

Tool Selection: We initially focused on a collection of static, dynamic, and instrumentation tools to base the remainder of Phase 1 work on. We spent time with the Angr[1] binary analysis platform. Angr provides advanced static and concolic analysis implementations and is commonly used to solve Capture The Flag (CTF) challenges. We decided to not continue with Angr as a platform as it is not as robust for use in real-life examples. Angr only supports emulated target execution and is optimized for small binaries such as firmware images or contrived challenges.

We then considered building a custom framework using a dynamic tool like Qemu or Pin. Since crash triage is a priority, we are focused on dynamic analysis and tracing tools that can scale to large binaries. Using Pin will provide the introspection, but lack the existing bindings to supporting code such as taint tracking or a theorem prover. Instrumenting Qemu is a very heavyweight that will make it more challenging to eventually apply to real-world examples. Qemu also provides an even less defined interface to instrument the execution.

Triton[2] is a tool developed by Quarkslab that is built on top of Intel's Pin. It adds some necessary components such as basic memory taint tracing, support for Python bindings, and the integration with an SMT solver. We are choosing to create our prototype using Triton with the express goal of implementing two proof of concept utilities: a reimplementation of an "!exploitable" (WinDbg module) rule as well as show basic taint tracking functioning with a sample crashing binary.

We chose to build our future research and our basic prototype using the Triton framework.

[1] http://angr.io
[2] http://triton.quarkslab.com/

Current Approach: Pin, and all tools built upon it, rely on introspecting a target process from within it. Although Pin can preempt most crashes, Triton as a framework ensures that all memory accesses are only done on mapped memory. This creates issues when dealing with binaries that are known to crash with memory dereferencing issues. To that effect, we are working on a lightweight binary tracer to record the execution of a target binary in a “light-touch” manner and store the execution trace to disk. We are then using this trace to execute the target once more with full instrumentation applied.

This gives us two benefits over plain instrumentation: we can run analysis algorithms on a not-yet crashed binaries to determine input influence in a valid program and we can use the recorded trace for post-hoc analysis.

Future: We are working to build a prototype showing the promise of such an approach on sample binaries. A basic tracer is being implemented in C++ and Python, and Triton modules are being implemented in Python.