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.