Vulnerability-Specific Execution Filtering with Log-Based Architecture Lifeguards

15-740 Project Milestone

Peter Chapman peter@cmu.edu
Deby Katz dskatz@cs.cmu.edu
Stefan Muller smuller@cs.cmu.edu

November 20, 2012

All project documents are available at `http://www.cs.cmu.edu/afs/cs/user/pmchapma/www/15740.html` .

Accomplishments

Literature Review

We have read dozens of papers relating to dynamic taint tracking, input and vulnerablity resistant systems, and log-based architectures. While we have notes and small-scale summaries on each of these papers, we have not yet compiled a complete literature review.

LBA Compilation and Setup

Early into the project we were able to successfully run (what we thought was) the LBA virtualized in `Simics`. Due to our lack of familiarity of the system and nonexistent documentation, booting `Simics` is a very different accomplishment from running a simulation of the log-based architecture with actual lifeguards. The true system setup lasted multiple weeks. Any documentation from when this version of the code was written in 2007 has been lost and current members of the LBA team only well understood newer code on their systems. Through much trial and error and assistance we were able to get the LBA simulator running with functional lifeguards, with the last problem being a corrupted version of the codebase fixed with a fresh checkout. We now have our own documentation for this process ($\sim$ 800 words).

VSEF Generation

The original goal was to use prior work in generating vulnerability-specific execution filters as a black-box input to our LBA lifeguard. Unfortunately, the author and implementor of the project withheld his implementation and has been unresponsive to email-based inquiry. Instead we decided to put together a simple filter to use as a starting point. We have two simple and well understood exploits against Linux tools `iwconfig` and `Ghostscript`. We used the Next-Generation Binary

Analysis Platform[1] (BAP) to track user input through our exploits to produce a log of tainted instructions. That log was then used as the basis for our development filters.

The prototype filters have a few limitations: 1) the filters were generated by hand but that process will be automated soon; 2) the addresses in the filter are absolute and we have not been able to confirm that they correspond to the addresses we will find in the simulator; 3) the filter only contains instructions corresponding the data flow in the exploit, but control flow information is necessary to increase accuracy.

VSEF Lifeguard

We have begun the process of modifying the LBA code to run callbacks on only instructions contained in a filter. Our goal is to modify only the main LBA code and not individual lifeguards, so new lifeguards can be used unmodified with our filtering mechanism. On initialization of logging, the lifeguard code loads a filter, specified as a list of instruction addresses, from a file and builds a set of checked instructions. Our intention is that when a record is pulled from the log during processing, it is only procesed if the instruction is in the set. However, based on our initial tests, it seems that, while the set is being initialized properly and the code to check for membership works, this check is misplaced within the structure of the LBA code and the callbacks are still running on all instructions. Once we fix this, there are two issues we might attend to as well. First, the path of the filter must currently be hardcoded. Second, we currently use the set implementation from the C++ Standard Library, which generally uses a binary tree implementation. If performance is not as high as we would like, we could switch to an implementation such as hashing the address and indexing into a bit vector, which would be asymptotically faster in the common case where the address is not in the set.

Milestone

In line with our originally stated goals we are on our way to modifying LBA to handle execution filters. The difficulty in setting up the system has pushed us behind schedule in preparing the evaluation and benchmarking. Finally, we are a bit behind on preparing the literature review.

Revised Schedule

The primary change in schedule is that we will not have time to complete the tertiary goal. We still believe that it is feasible to finish the prototype execution filter with a limited evaluation.

[1] http://bap.ece.cmu.edu/