Achieving Low-Overhead Defense Against ROP Hacker Attacks Using Safe Page Sets

Morgan Burcham [1][2]

Dr. David Koppelman [2][3]

[1] Department of Computer Science, University of North Alabama
[2] Center for Computation & Technology, Louisiana State University
[3] College of Engineering, Division of Electrical & Computer Engineering, Louisiana State University

Abstract

With return-oriented programming attacks, an attacker creates his/her own code by chaining together instructions present within the original (good guy's) program. The attacker creates a set of gadgets which are Turing complete (meaning they can run any program), gets the attack code into memory, and links the attack code together with return instructions. Luckily, defenses do exist for these attacks but at the cost of efficiency. The goal of the research is to find a more efficient defense.

Our proposed defense monitors the execution of instructions. A set of unlocked pages in memory are managed so that a Turing complete set of instructions is never found.

The proposed defense has a basic implementation. Optimization will be needed. More benchmarks to test with will also need to be found. Also, a more complete set of gadget classes will need to be found. Finally, the defense will need to be compared to other defenses to determine how effective it is.

Introduction

Code injection is a common method of attack. For example, with buffer overflow on the stack, an attacker can inject code into memory and overwrite the saved return address of the executing function. Upon function return, execution will be redirected to the address of the attacker's code.

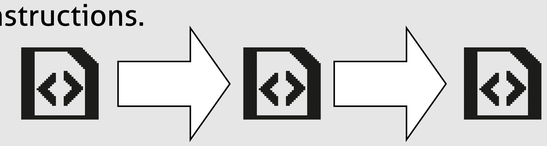
However, a defense called Writable XOR Executable prevents such attacks by preventing memory from being both writable and executable. However, Writable XOR Executable does NOT prevent return-oriented programming attacks.

The Attack

1. Reproduce snippets of the original code to create your own attack code. Find code that can do anything you want.

2. Get the code into memory (i.e. buffer overflow on stack). Overwrite the return address of the executing function with the address of the inserted code.

3. Link the code together with return instructions.



Methods

A set of instructions is considered Turing complete if it can run any program you want. The proposed defense will monitor the execution of instructions. The defense will manage a set of unlocked pages in memory so that a Turing complete set of instructions is never present.

1. If the code ever visits a new page which will create a Turing complete set:
   - -Execution will be slowed and the current page will be allowed.
   - -Other pages will be locked so that a Turing complete set is nonexistent.
   - -Memory protection will then be utilized to reacquire control of the program.
2. If no Turing complete set of pages exists:
   - -Execution will continue as normal.

Results

Percentage Page Limit (based on total number of scanned pages)

Benchmarks	25%	50%	75%	100%
Perl	70 pgs.	141 pgs.	212 pgs.	283 pgs.
Bzip2	22 pgs.	45 pgs.	67 pgs.	90 pgs.
gcc	211 pgs.	422 pgs.	634 pgs.	845 pgs.
TeX	38 pgs.	77 pgs.	116 pgs.	155 pgs.

These results show the average number of unlocked pages for each benchmark when setting the page limit to 25%, 50%, 75%, and 100% of the total scanned pages respectively. We restricted the gadgets that must be found for qualification as a Turing complete set. None of our data qualified as Turing complete.

Future Work

-Optimize the defense.
-Find a better set of gadgets to work with.
-Define what gadgets are needed for Turing completeness.
-Test more vigorously on other data sets.
-Compare efficiency to other known defenses.
-Need a fallback for when a set is Turing complete.

Acknowledgments

This material is based upon work supported by the National Science Foundation under award OCI-1263236 with additional support from the Center for Computation & Technology at Louisiana State University.

Center for Computation & Technology