Program Reachability for Vulnerability and Malware Analysis

Problem

Highly skilled Department of Defense (DoD) malware and vulnerability analysts currently spend significant amounts of time manually coercing specific portions of executable code to run.

Solution

Automate the analysis of binary code, choosing program inputs that will trigger specific behavior to reduce the time that DoD cyber personnel spend performing complex software analysis.

Approach

Use model checking techniques to identify these inputs and generate a simplified executable free of complex and convoluted dependencies that can be analyzed by existing code analysis tools.

Intended Impact (FY18–20)

Improve the DoD's ability to measure and monitor the advancement of path-reachability research, especially as Ghidra decompilation quality improve improves.

Testing Method

A total of 91 test programs were compiled for three optimization levels and two architectures. Each test attempted to find a path from a starting location to a reachable goal and an unreachable goal. If both answers were correct, the test passed. The test timeout was 30 minutes.

2,184 test configurations found several successful approaches, but **none** that **consistently outperformed** the others, suggesting a needed hybrid **approach**.

Test Case Configuration		Pharos Function Summaries			Weakest Precondition			Property Directed Reachability			Ghidra + Seahorn		
Optimized	**Arch**	**Fail**	**Timeout**	**Pass**	**Fail**	**Timeout**	**Pass**	**Fail**	**Timeout**	**Pass**	**Fail**	**Timeout**	**Pass**
None	32-bit	55	2	34	16	2	73	3	29	59	21	7	63
None	64-bit	47	0	44	15	3	73	2	36	53	28	2	61
Medium	32-bit	40	0	51	9	3	79	1	13	77	12	7	72
Medium	64-bit	53	0	38	9	4	78	1	17	73	21	6	64
High	32-bit	50	0	41	6	2	83	1	12	78	18	7	66
High	64-bit	32	1	58	28	3	60	2	16	73	32	5	54
Total		**257**	**3**	**266**	**83**	**17**	**446**	**10**	**123**	**413**	**132**	**34**	**380**

Key

- Best result
- Second best result
- Third best result
- Worst result

Pharos Function Summaries

SEI's Pharos binary analysis framework computes symbolic function summaries by symbolically executing binary code. This technique converts these function summaries into light-weight constraints. The conversion uses a simple model of memory that is very efficient, but is known to be incorrect in the presence of interprocedural reasoning.

This approach is very fast, but its imprecision results in a large number of failures and a small number of passing tests.

Weakest Precondition

The weakest precondition algorithm finds the weakest constraints on the program input that are required for the program to terminate successfully. We force execution to the desired program locations by adding assertions. This technique uses an array encoding of memory, which is precise but expensive to reason about. It also cannot reason generally about loops.

This well-known approach is still the benchmark to beat. It performs well but has significant deficiencies when analyzing code with loops.

Property Directed Reachability

Property Directed Reachability (PDR) is a technique used in source code software model checking. It iteratively generates an inductive invariant to prove that the target code is unreachable, and it uses counter-examples to refine the invariant, so it can prove targets are unreachable even when there are loops. It uses the same array encoding of memory as the previous technique.

This approach is very accurate but has severe performance problems in the binary domain due the array memory model, which is not necessary at the source code level.

Ghidra + Seahorn

This technique uses the NSA's Ghidra decompiler to raise the executable code to a C-like language rather than trying to express the binary semantics directly. The Seahorn software model checker is then used to check reachability using PDR. Because it operates on a source code representation, the encoding is very different than the other PDR approach.

PDR can be fast when using a source code representation. Unfortunately, decompilation can fail in myriad ways, and this accounts for the majority of failures for this approach.

Cory F Cohen | Dr. Edward J Schwartz | info@sei.cmu.edu

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM20-0861