Project Description

Optimizing Symbolic Execution Through Taint Analysis and Path Prioritization

Short Description

Symbolic execution is a powerful technique for program analysis, but it suffers from scalability issues. Optimizing its path exploration is crucial for improving efficiency. This work aims to enhance symbolic execution by leveraging taint analysis to prioritize paths originating from memory allocations and user inputs, enabling more effective exploration of relevant program behaviors.

Long Description

German version Die symbolische Ausführung ist eine leistungsstarke Technik zur Analyse von Software, leidet jedoch unter Skalierbarkeitsproblemen. Die Effizienz dieser Technik hängt stark davon ab, welche Programmpfade priorisiert werden. Eine gezielte Auswahl relevanter Knoten kann die Exploration effizienter gestalten.

Dieses Projekt verfolgt das Ziel, die symbolische Ausführung durch den gezielten Einsatz von Taint-Analyse zu optimieren. Statt eine gleichmässige Exploration aller Programmpfade durchzuführen, sollen insbesondere Speicherallokationen und Benutzereingaben als sicherheitskritische Punkte priorisiert werden.

Mit Hilfe des Kontrollflussgraphen (CFG) oder beispielsweise durch das Scannen nach Systemaufrufen (System Calls), die externe Eingaben verarbeiten, werden relevante Knoten identifiziert. Diese dienen als Ausgangspunkte für die symbolische Ausführung. Anschliessend erfolgt die Integration mit einer symbolischen Ausführungs-Engine (z.B. angr), um die Exploration gezielt auf sicherheitsrelevante Pfade zu lenken und redundante oder wenig aussagekräftige Pfade zu vermeiden. Programmpfade, die keine Abhängigkeit zu Eingabedaten haben, werden ignoriert (z.B. durch Taint-Analyse).

Die Wirksamkeit der Optimierung wird durch eine vergleichende Analyse der symbolischen Ausführung mit und ohne Priorisierung evaluiert. Dabei werden Kriterien wie Laufzeit, erkannte Programmpfade und potenzielle Schwachstellen untersucht.

English version Symbolic execution is a powerful technique for analyzing software, but suffers from scalability problems. The efficiency of this technique strongly depends on which program paths are prioritized. A targeted selection of relevant nodes can make exploration more efficient.

This project aims to optimize symbolic execution through the targeted use of taint analysis. Instead of performing an even exploration of all program paths, memory allocations and user input in particular are to be prioritized as safety-critical points.

Relevant nodes are identified using the control flow graph (CFG) or, for example, by scanning for system calls that process external input. These serve as starting points for the symbolic execution. They are then integrated with a symbolic execution engine (e.g. angr) in order to direct the exploration specifically to security-relevant paths and avoid redundant or less meaningful paths. Program paths that have no dependency on input data are ignored (e.g. through taint analysis).

The effectiveness of the optimization is evaluated by a comparative analysis of the symbolic execution with and without prioritization. Criteria such as runtime, detected program paths and potential vulnerabilities are examined.