MSE | MASTER OF SCIENCE IN ENGINEERING

Forschungsprojektarbeit des Profils Information and Cyber Security

Qemu, LibAFL und Concolic Execution

Ausgangslage

Fuzzing hat sich als effektive Methode zur Entdeckung von Sicherheitslücken in Software etabliert. Jedoch stossen klassische Fuzzing-Ansätze an Grenzen, wenn es um komplexe Programmlogik mit vielen Bedingungsprüfungen geht. Die LibAFL-Bibliothek bietet einen modernen, modularen Ansatz für die Entwicklung von Fuzzern und unterstützt verschiedene Backends, darunter QEMU für die architekturunabhängige Instrumentierung von Binärdateien ohne Quellcode. An verschiedenen Hochschulen und in der Industrie werden Fuzzing-Techniken kontinuierlich weiterentwickelt, um ihre Effektivität zu steigern. Besonders vielversprechend ist die Kombination von Fuzzing mit symbolischer Ausführung, bekannt als Concolic Execution. Diese Technik kann helfen, komplexe Programmzweige zu erreichen, die für reines Fuzzing schwer zugänglich sind. Trotz der Fortschritte in diesem Bereich fehlt es an praktikablen Implementierungen, die Concolic Execution für Binärdateien in moderne Fuzzing-Frameworks wie LibAFL integrieren.

Zielsetzung / Methodik / Vorgehen

Das Ziel dieses Projekts ist die Erweiterung des QEMU-Fuzzers von LibAFL mit Concolic-Execution-Fähigkeiten, um die Testabdeckung und Effektivität beim Auffinden von Softwarefehlern zu erhöhen. Zunächst soll eine gründliche Analyse der LibAFL-Architektur und des QEMU-Instrumentierungssystems durchgeführt werden, um Integrationspunkte für symbolische Ausführung zu identifizieren. Darauf aufbauend wird ein hybrides Analysesystem entwickelt, das während der Ausführung Pfadbedingungen sammelt und mittels eines SMT-Solvers neue Eingaben generiert, die alternative Programmpfade erreichen können. Die Implementation erfordert tiefgreifende Kenntnisse in den Bereichen Programmanalyse, LLVM/QEMU-Instrumentierung und Constraint-Solving. Ein besonderer Schwerpunkt liegt auf der Optimierung der Performance, da die symbolische Ausführung rechenintensiv ist und die Geschwindigkeit des Fuzzings nicht übermäßig beeinträchtigen darf. Die Lösung soll modular gestaltet sein, sodass verschiedene Constraint-Solver (z.B. Z3, Boolector) eingebunden werden können. Zur Evaluation wird eine Testumgebung mit verschiedenen Zielprogrammen eingerichtet, die bekannte Schwachstellen enthalten, die für traditionelles Fuzzing schwer zu finden sind. Die Effektivität wird anhand von Metriken wie Pfadabdeckung, Zeit bis zur Entdeckung bekannter Bugs und Ressourcenverbrauch gemessen.
Verlangte Skills: Fortgeschrittene Programmierkenntnisse (Rust, C/C++), Verständnis von Programmanalysetechniken

Teilaufgaben

Das Projekt kann in 3 Phasen durchgeführt werden: Fortlaufende Projekte 7 (Analyse der LibAFL-Architektur, Konzept zur Integration von Concolic Execution), 8 (Implementation der symbolischen Ausführung und Constraint-Sammlung, Anbindung eines SMT-Solvers) und 9 (Optimierung der Performance, umfassende Evaluation und Vergleich mit bestehenden Lösungen, Dokumentation)

Studienart: [ ] Vollzeitstudium
[ ] Teilzeitstudium 50%

Projektorganisation: Einzelarbeit (im Umfeld eines Projektteams)

Projektfinanzierung: -

Arbeitsort: Windisch

Advisor: Prof. Dr. Christopher Scherb