Fortgeschrittene Softwaretechnik
Wintersemester 2024/25
Name: Simon Szulik
Matr.Nr: 1474315

Datum: 10. November 2024
Reading Group Sessions
Inhaltsskizze zu Block: 02

Titel des Papiers

Continuous Fuzzing

Kurzzusammenfassung

- **Hintergrund und Motivation**
  - Fuzzing ist eine wichtige Methode zur Entdeckung von Sicherheitslücken, aber es ist oft ressourcen- und zeitaufwendig. In CI/CD-Pipelines, die schnelle und effiziente Tests erfordern, stellt sich die Frage, wie Fuzzing effektiv integriert werden kann. Die Motivation hinter dieser Untersuchung liegt darin, den Fuzzing-Aufwand zu verringern, ohne die Entdeckung von Sicherheitslücken zu gefährden. Da Studien zeigen, dass die meisten bugs durch erst kürzlich vorgenommene Codeänderungen eingeführt werden untersuchen die Autoren in welchen Fällen von Commits Fuzzing nötig ist. Desweiteren wird geprüft wie lange Fuzzing dauern sollte, und sowohl Ressourcen zu sparen als auch Sicherheitslücken zuverlässig zu finden.
- **RS 1**: Wie kann Fuzz-Testing an die Anforderungen von CI/CD-Pipelines angepasst werden, unter Berücksichtigung der begrenzten Rechenressourcen?
- **RS 2**: Wie lange sollte eine Fuzzing-Kampagne dauern, um mit den Zeitrahmen von CI/CD-Tests kompatibel zu sein und gleichzeitig effektiv Sicherheitslücken zu finden?
- Vorgehen
  - Die Autoren nutzen die Magma-Benchmark welche insgesammt 138 bekannte Fehler aus realen Softwareprojekten umfasst. Damit wurden Fuzzer in CI/CD-Umgebungen getestet und deren Effektivität bei verschiedenen Fuzzing-Dauern untersucht. Die Autoren erweiterten die Plattform, um Ensemble-Fuzzing zu integrieren, bei dem mehrere Fuzzer zusammen an denselben Zielen arbeiten. Das Ziel war es, Ressourcen zu sparen, indem nur die Fuzz-Ziele bearbeitet werden, die durch Codeänderungen betroffen sind. Dies wurde erreicht, indem man Prüfziffern für die Ziele berechnete und so unnötige Fuzzing-Läufe vermeiden konnte.
- **Ergebnisse**
  - Das Experiment zeigt, dass bearbeitete Commits bei den insgesamt neun 9 getesteten Bibliotheken zu 6-42% identische Fuzz-Ziele lieferten. Zwei Bibliotheken (libxml2, sqlite3) hatten aufgrund von Versionsunterschieden unterschiedliche Ziele. Bei den übrigen sieben Bibliotheken lag der Anteil identischer Fuzz-Ziele bei 20-64%, mit einem gewichteten Durchschnitt von 55%. Dies deutet darauf hin, dass gezielte Zielauswahl die Rechenressourcen deutlich spart
    Zudem war die Dauer des Fuzzings entscheidend: Kürzere Ausführungen (10 Minuten) waren oft genauso effektiv wie längere (mehrere Stunden), insbesondere bei Bibliotheken mit wenigen Bugs. Eine Dauer von 10-30 Minuten reicht für gute Ergebnisse meistens aus, wobei kürzere Fuzzings für Pipelines und längere für weniger zeitkritische Phasen empfohlen werden um eine breitere Masse an Bugs zu identifizieren.

Eigene (offene) Diskussionspunkte

- **Allgemeine Schwierigkeiten der Methode**
  - Identifiziereun von anfälligen Bibliotheken
    - Die Ergebnisse zeigen, dass kürzere Fuzzing-Sessions oft genauso effektiv wie längere sein können. In der Praxis stellt sich jedoch die Frage, wie man Bibliotheken identifiziert, die besonders anfällig für Bugs sind und intensiver getestet werden sollten. Ein allgemeines Maß könnte hier helfen, das Risiko von Bibliotheken zu bewerten, etwa basierend auf Code-Komplexität oder der Häufigkeit von Sicherheitslücken.
- **Kritik am Paper selbst**
  - Begrenzende Betrachtung
    - Das Paper konzentriert sich auf die Anwendung von Fuzzing in CI/CD-Pipelines, jedoch fehlt eine Analyse zur Anpassungsfähigkeit in unterschiedlichen Softwareumgebungen und Programmiersprachen.
  - Mangel an Langzeittests
    - Die im Paper beschriebenen Experimente sind auf kürzere Testphasen beschränkt, sodass die Langzeitwirksamkeit des kontinuierlichen Fuzzings in CI/CD-Pipelines unklar bleibt.