

# Fault Injection-based Fuzzing

Nils Bars

## Summary

Digital data exchange is ubiquitous in our world, ranging from instant messaging to remote control of critical infrastructure. All such communication relies on format and protocol specifications, which, for historical and performance reasons, are often implemented in memory-unsafe languages. As a result, bugs in these implementations can have severe consequences, potentially allowing attackers to compromise systems.

Rigorous testing is essential to reduce this risk, and automated techniques are particularly valuable given the scale and diversity of modern software. In practice, fuzzing has emerged as one of the most successful techniques. However, current fuzzers face fundamental limitations because they lack knowledge of the underlying data formats and protocol semantics. Fuzzers struggle to generate inputs that involve complex data transformations, such as compression or encryption, or stateful network exchanges depending on ephemeral values, such as nonces or session identifiers. The problem is even more challenging for systems without a natural input interface, such as software-based fault isolation (SFI) systems. Unlike typical applications that receive input through files or network channels, SFI is a confinement technique that isolates untrusted code within the same process. Consequently, the security boundary lies within the same process and cannot be easily exercised by traditional fuzzers.

This thesis introduces a fundamentally new approach, *fault injection-based fuzzing*, to address these challenges. The core insight is that the problems above arise because traditional fuzzers are built so that domain-specific knowledge about the target format or protocol must be explicitly encoded into the fuzzer. Since fully reimplementing this knowledge is rarely feasible, especially when inputs must conform to complex specifications, we propose reusing existing software that already knows how to produce valid data. By injecting controlled faults into the process of generating output that becomes input to the tested target, we preserve its ability to produce well-formed, signed, or encrypted data while introducing mutations. Throughout this thesis, we demonstrate that this technique is broadly applicable across file-processing programs, stateful networked systems, and intra-process isolation mechanisms, such as the *heap sandbox* exploitation mitigation of the JavaScript engine V8.

## Zusammenfassung

Der digitale Datenaustausch ist in unserer Welt allgegenwärtig, von Instant Messaging bis hin zur Fernsteuerung kritischer Infrastrukturen. Jegliche Kommunikation beruht auf Format- und Protokollspezifikationen, die aus historischen und Performance-Gründen häufig in Sprachen implementiert sind, die nicht memory-safe sind. Infolgedessen können Fehler in diesen Implementierungen schwerwiegende Folgen haben und es Angreifern ermöglichen, Systeme zu kompromittieren.

Eine rigorose Testung ist entscheidend, um dieses Risiko zu verringern. Angesichts des Umfangs und der Vielfalt moderner Software sind automatisierte Techniken besonders nützlich. In der Praxis hat sich dabei das Fuzzing als eine der erfolgreichsten Methoden durchgesetzt. Aktuelle Fuzzer stoßen jedoch an ihre Grenzen, da ihnen das Wissen über die zugrunde liegenden Datenformate und Protokollsemantiken fehlt. Fuzzer haben Schwierigkeiten, Eingaben zu erzeugen, die komplexe Daten-Transformationen wie Kompression oder Verschlüsselung beinhalten, oder zustandsbehaftete Netzwerkaustausche, die von kurzlebigen Werten wie Nonces oder Sitzungs-Identifikatoren abhängen. Das Problem ist für Systeme ohne natürliche Input-Schnittstelle noch herausfordernder, etwa für Software-Based Fault Isolation (SFI) Systeme. Im Gegensatz zu typischen Anwendungen, die Eingaben über Dateien oder Netzwerke erhalten, ist SFI eine Isolierungstechnik, die nicht vertrauenswürdigen Code innerhalb desselben Prozesses isoliert. Folglich liegt die Sicherheitsgrenze innerhalb desselben Prozesses und kann von traditionellen Fuzzern nicht leicht getestet werden.

Diese Arbeit stellt einen prinzipiell neuen Ansatz vor: fault-injection-based Fuzzing, um diese Herausforderungen zu adressieren. Die zentrale Erkenntnis ist, dass die oben genannten Probleme darauf zurückzuführen sind, dass traditionelle Fuzzer so aufgebaut sind, dass domänenspezifisches Wissen über das Zielformat oder -protokoll explizit in den Fuzzer kodiert werden muss. Da eine vollständige Neuimplementierung dieses Wissens selten praktikabel ist, insbesondere wenn die Eingaben komplexen Spezifikationen entsprechen müssen, schlagen wir vor, vorhandene Software wiederzuverwenden, die bereits weiß, wie gültige Daten erzeugt werden. Durch das gezielte Einbringen kontrollierter Fehler in den Prozess der Ausgabeerzeugung, die anschließend als Input für das getestete Ziel dienen, nutzen wir die Fähigkeit dieser Software, wohlgeformte, signierte oder verschlüsselte Daten zu produzieren, und führen zugleich Mutationen ein. Im Verlauf dieser Arbeit zeigen wir, dass sich diese Technik bei Programmen zur Dateiverarbeitung, zustandsbehafteten Netzwerksystemen und intraprozessualen Isolationsmechanismen wie der Heap-Sandbox-Schutzmaßnahme der JavaScript-Engine V8 anwenden lässt.