Slide 1 : Introduction

"Good morning/afternoon everyone! I’m excited to share my research on developing a custom browser fuzzing tool that targets vulnerabilities in modern browser engines. Today, I'll walk you through how we designed the tool, overcame limitations in existing approaches, and integrated advanced mutation strategies to uncover critical security issues."

Slide3,4:

"Browsers are our gateways to the web, yet their complexity makes them a prime target for attackers. Although tools like Domato and AFL have paved the way, they don’t fully address the intricate nature of browser internals. This gap in effective testing motivates our approach: combining diverse mutation techniques with real-time crash detection."

Slide5: Aim and objective

"Our tool is designed to bridge the gap between theory and practice in browser fuzzing. By combining robust mutation techniques with automated testing and detailed crash analysis, we aim to empower researchers with actionable insights. This slide outlines the core steps that drive our system."

Slide 6: research

"Domato’s early work in DOM fuzzing set a strong foundation, but its limited mutation approach left gaps in coverage. On the other hand, AFL++ showed us the power of code-coverage feedback, which inspired our dual-phase mutation strategy. By merging these ideas, our approach benefits from the robustness of seed-based generation and the efficiency of coverage-guided mutation."

Slide 7: Methodology

"Our methodology is structured in clear phases. Phase 0 sets up our testing environment with ASan-enabled headless browsers and Selenium. In Phase 1, we design a modular architecture for efficient testing. Phase 2 generates a diverse test corpus using seed files and mutation engines like Radamsa and AFL++. Finally, in Phase 3, we implement an automated crash detection and logging system with ASan to capture detailed error data, which feeds back into refining our testing strategies."

Slide 8: achievement

"On the technical front, our project has made significant strides. We've generated over 500 seed files and created more than 1,200 mutated test cases through Radamsa, allowing us to explore various edge cases. Additionally, integrating AFL++ has empowered us with coverage-guided fuzzing for more in-depth code path exploration.

Beyond these achievements, we set up headless versions of Chrome and Firefox with ASan enabled. This setup enables automated testing without the overhead of a graphical user interface, making our process both faster and more resource-efficient. Complementing this, we developed a Selenium framework to automate test case execution. Selenium handles launching our headless browsers, executing the mutated inputs, and capturing real-time logs. This end-to-end automation is critical for efficiently detecting and logging vulnerabilities, streamlining the entire testing pipeline."