Fuzzers CLI + GUI Suite

Modern Security Testing with Integrated UI/UX

Hardish Singh

Project Overview: Unifying Fuzzing

What is Fuzzing?

Fuzzing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program. The goal is to discover software bugs, security vulnerabilities, or crashes by monitoring the program's behavior.

Why Combine CLI + GUI?

A unified interface offers the best of both worlds: the power and scripting capabilities of CLI for advanced users, and the ease-of-use and visual feedback of GUI for beginners. This integration streamlines workflows, enhances accessibility, and accelerates vulnerability discovery.

Benefits for Bug Bounty Hunters & Pentesters

- Accelerated vulnerability discovery
- Streamlined workflow for diverse fuzzers
- Improved collaboration and reporting
- Reduced learning curve for new tools

Architecture Diagram: Next.js ⇔ Node.js ⇔ Fuzzers

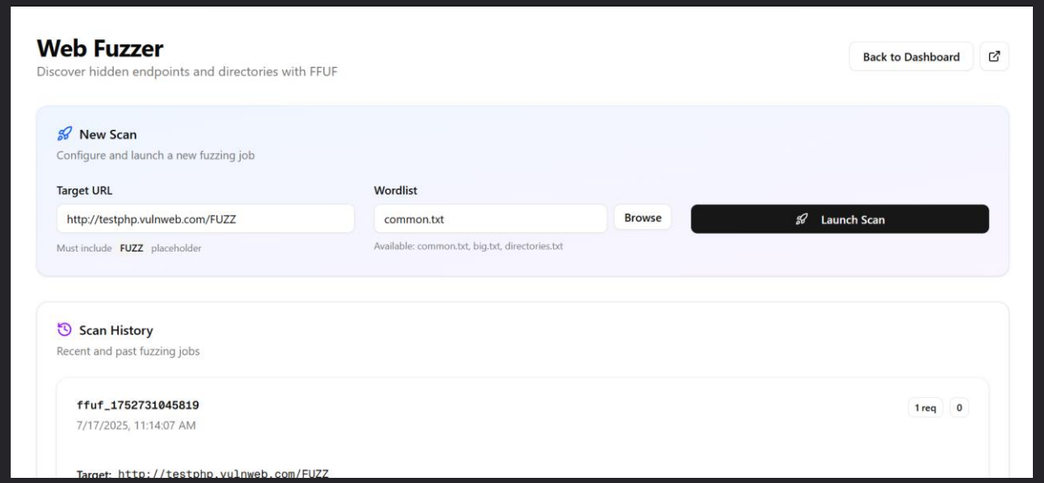
FFUF Integration: Fast Web Fuzzing

FFUF (Fuzz Faster U Fool) is a fast web fuzzer often used for directory and file enumeration, virtual host discovery, and GET/POST parameter fuzzing. Its speed and versatility make it a staple in web application security testing.

CLI Sample:

```
ffuf -u https://target/FUZZ -w wordlist.txt
```

The GUI integrates FFUF's core functionalities, offering intuitive controls for target input, wordlist selection, and HTTP response analysis. Users can easily visualize findings and filter results based on status codes or content length.



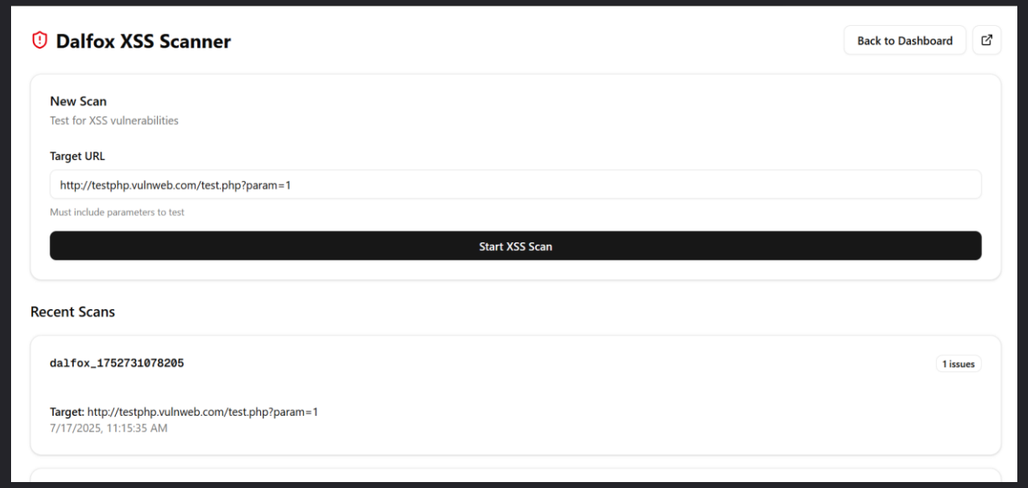
Dalfox Integration: XSS Scanning

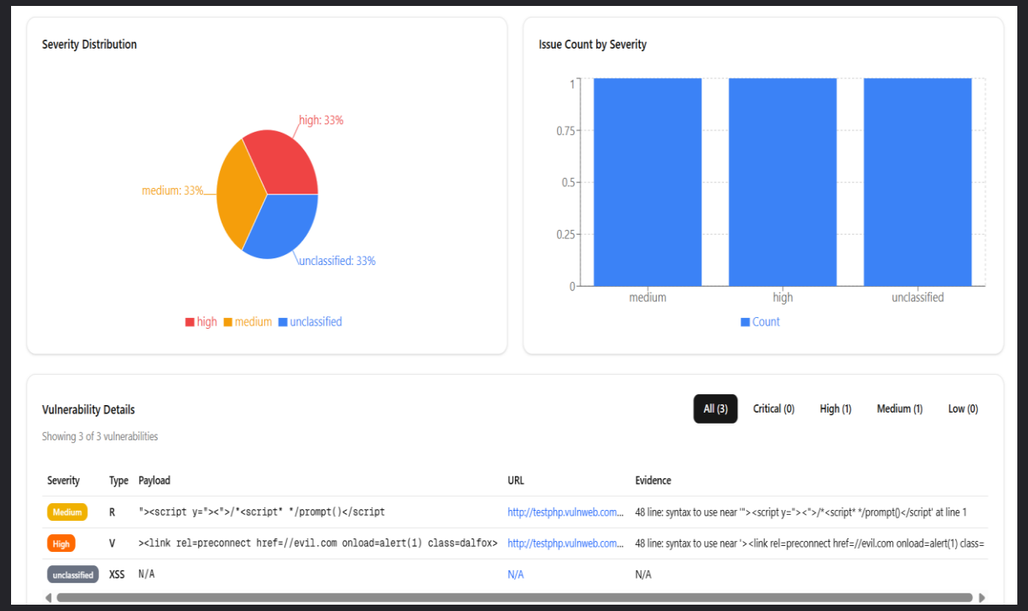
Dalfox is a powerful tool designed for detecting Cross-Site Scripting (XSS) vulnerabilities. It supports various payload injection points and provides detailed reports, making it an essential asset for web penetration testers.

CLI Sample:

```
dalfox url https://target
```

Our GUI provides a clear overview of Dalfox scan results, allowing for easy filtering and sorting of identified vulnerabilities. Interactive elements enable users to drill down into specific findings and analyze the impact.





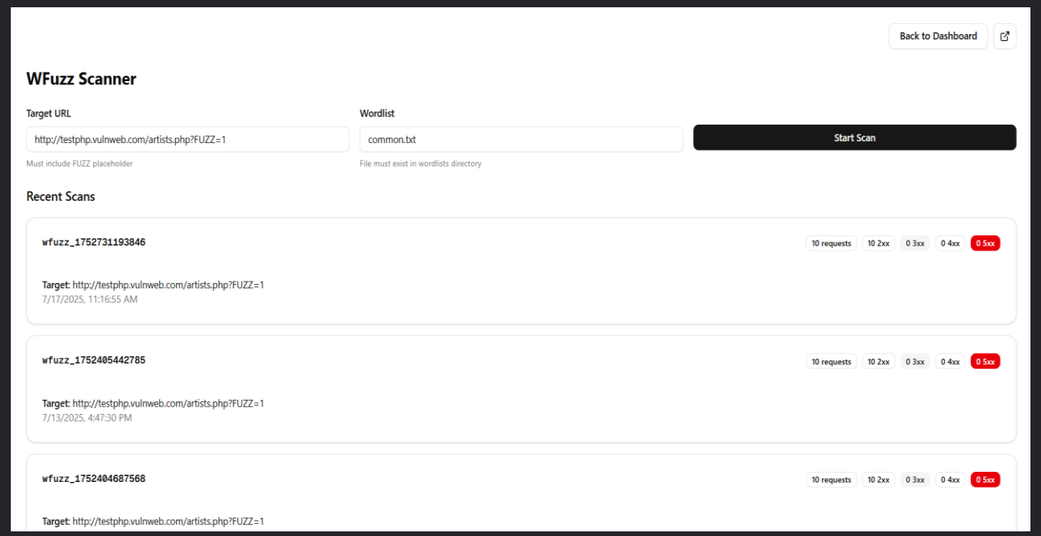
WFuzz Integration: Advanced Fuzzing

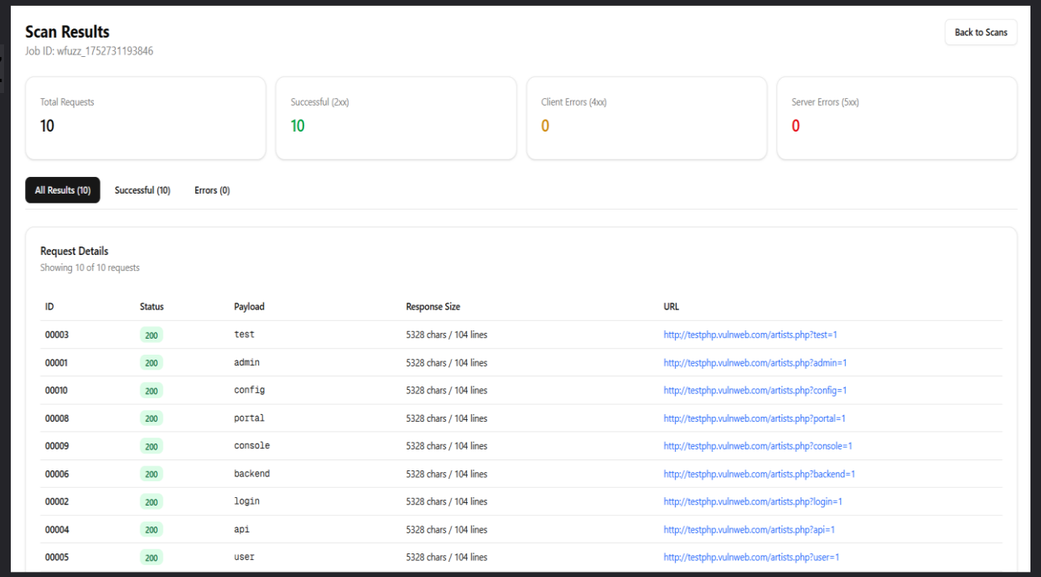
WFuzz is a highly flexible and powerful web application fuzzer. It offers extensive customization options, including payload encoding, different fuzzer types, and filters, enabling deep and precise vulnerability hunting.

CLI Sample:

```
wfuzz -c -z file,wordlist.txt -u https://target/FUZZ
```

The GUI for WFuzz simplifies complex configurations, providing a visual builder for fuzzing parameters and real-time display of response times. This helps in identifying slow responses or potential denial-of-service vulnerabilities efficiently.





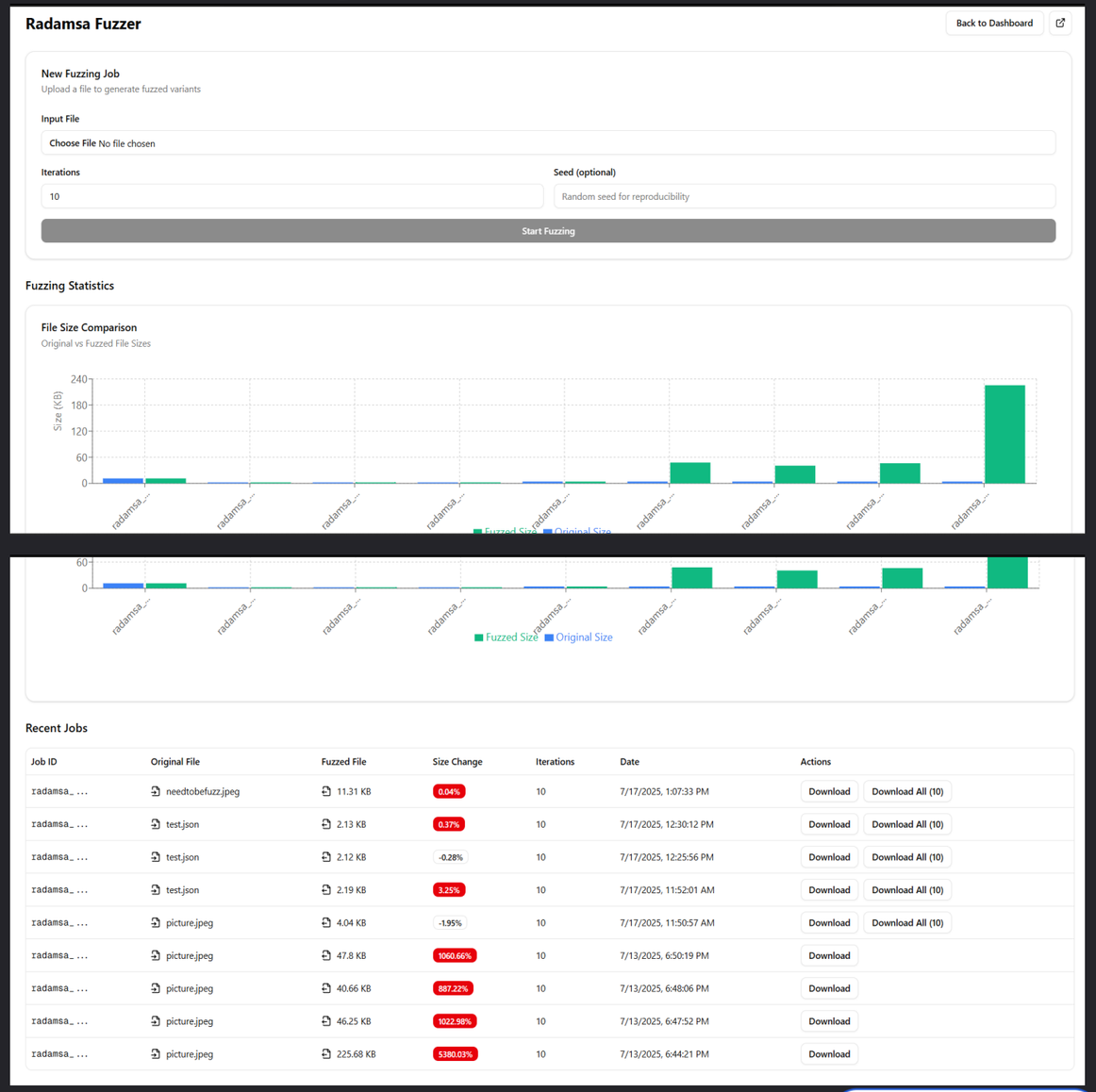
Radamsa Integration: Smart Fuzzing

Radamsa is a general-purpose fuzzer that generates test cases based on a given input. It's known for its ability to produce highly effective and "interesting" inputs, making it valuable for discovering unexpected edge cases and crashes in various applications.

CLI Sample:

```
echo "input" | radamsa
```

The Radamsa GUI provides an interactive interface for input mutation, allowing users to preview fuzzed samples in real-time. This visual feedback helps in understanding how inputs are transformed and refining the fuzzing strategy.



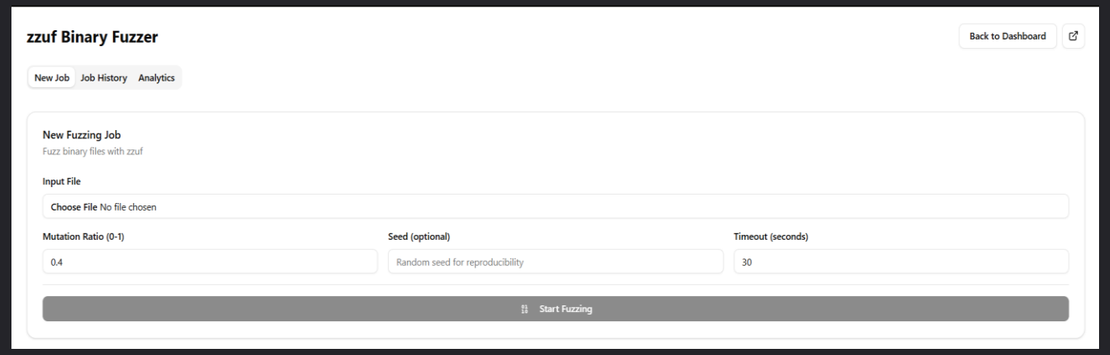
ZFuzz Integration: API Fuzzing

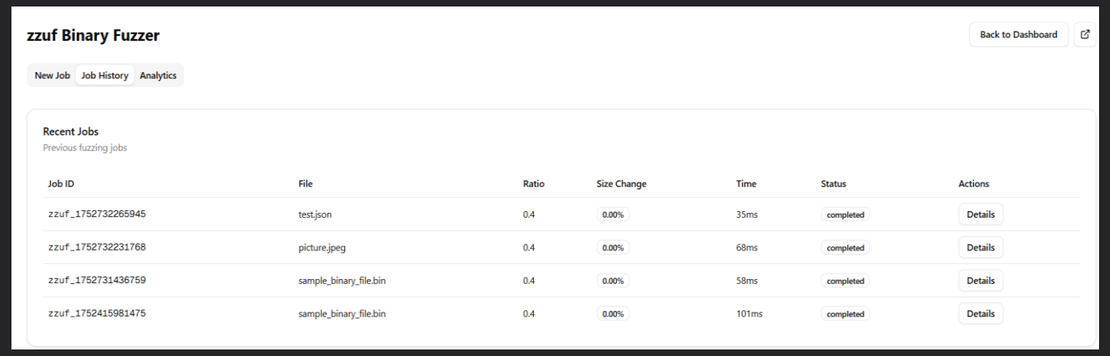
ZFuzz is tailored for efficient API fuzzing, focusing on web APIs and their parameters. It can quickly uncover vulnerabilities like injection flaws, improper authentication, or data exposure by systematically testing API endpoints.

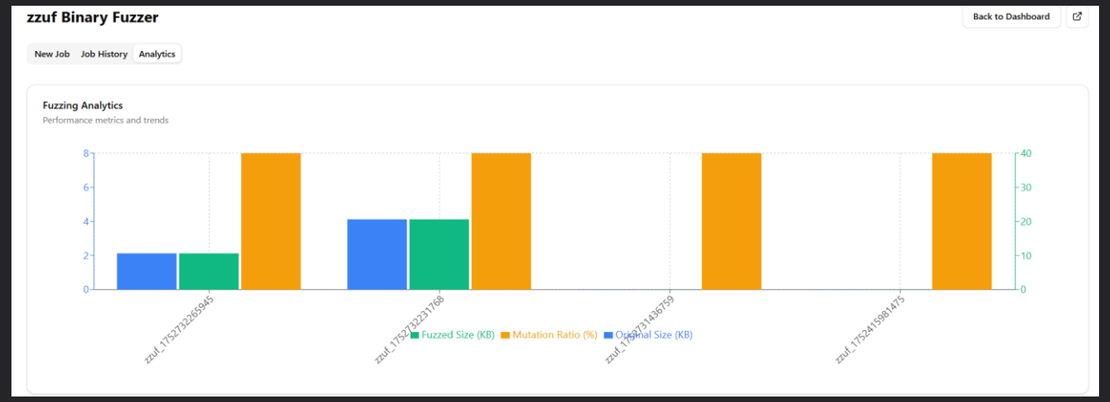
CLI Sample:

```
zfuzz -u https://target --wordlist wordlist.txt
```

The ZFuzz GUI features a live results stream, providing immediate feedback on API responses and identified issues. This real-time monitoring allows security professionals to react quickly to discovered vulnerabilities and adjust their testing strategy on the fly.







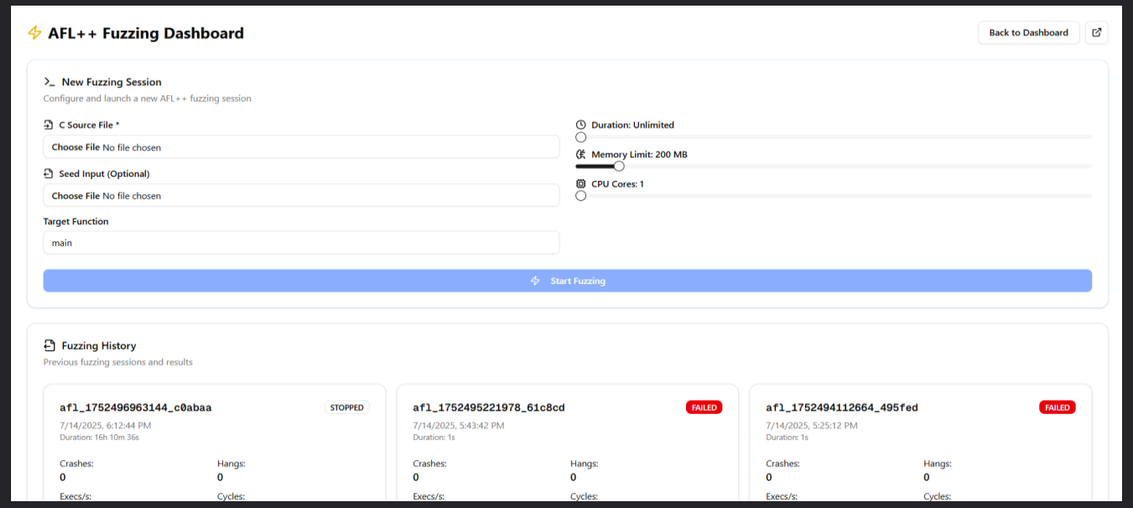
AFL++ Integration: Advanced Coverage Fuzzing

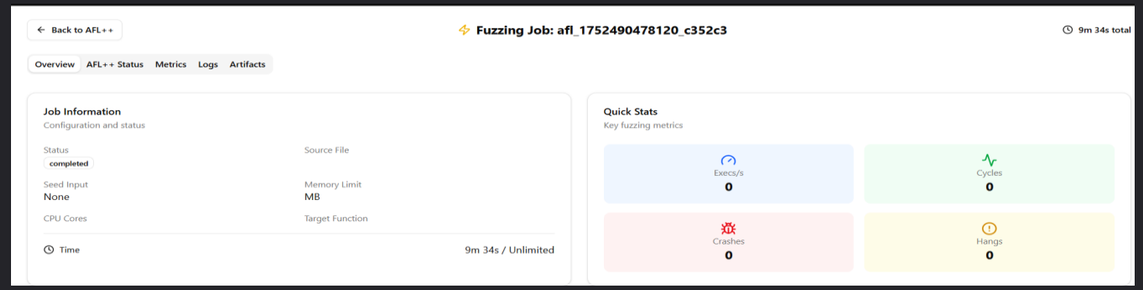
AFL++ (American Fuzzy Lop Plus Plus) is a highly optimized, coverage-guided fuzzer known for its effectiveness in finding deep-seated bugs and vulnerabilities in complex binaries. It uses advanced techniques to explore program execution paths.

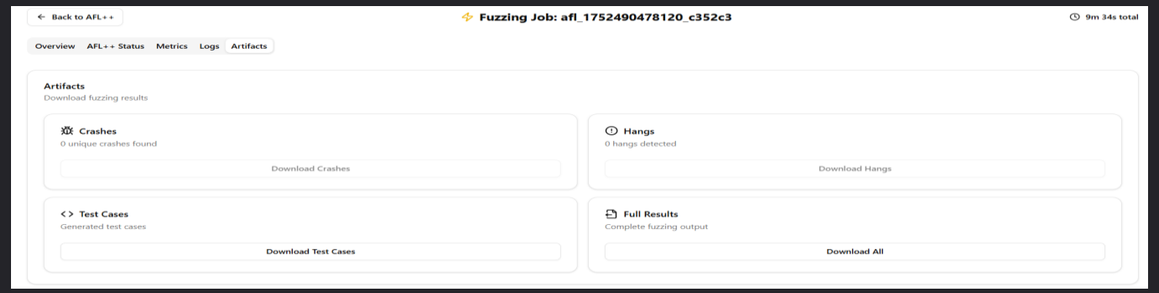
CLI Sample:

```
afl-fuzz -i inputs -o outputs -- ./target
```

Our GUI for AFL++ is not ready yet I am working on it . One's ready This visual representation helps users track fuzzing progress, identify bottlenecks, and understand the code coverage achieved, optimizing the fuzzing campaign .







Project Tech Stack

Our Fuzzers CLI + GUI Suite is built on a robust and modern technology stack designed for performance, scalability, and an intuitive user experience.

Next.js

Frontend framework for interactive and responsive UI/UX.

Node.js

Backend for CLI commands and fuzzer orchestration.

Firebase

Database for configuration persistence and CLI management.

This combination ensures a seamless blend of powerful backend operations with a modern, dynamic frontend, providing a superior fuzzing experience.

Conclusion & Future Plans

The Fuzzers CLI + GUI Suite provides a unified, efficient, and user-friendly platform for cybersecurity professionals. By combining the power of various fuzzers with a modern interface .

Summary of Features

- Integrated CLI + GUI for flexibility
- Modern tech stack (Next.js, Node.js, Firebase)
- Enhanced usability and workflow

Future Plans

- Real-time logging and advanced analytics
- Custom plugin architecture for extensibility

Follow me on GitHub if you feel that something more needed in this let’s contribute to the future of integrated security testing!

GitHub Repository:

github.com/Hardish-singh

Contact:

hardishrajpu35@gmail.com


THANK YOU!