YAGOfuzz

SCALABLE FUZZING FOR CONNECTED SYSTEMS

Created by :
Yagoba GmbH

Contents

Executive Summary

In today's hyper-connected digital infrastructure, software-driven technologies lie at the core of innovation in sectors such as automotive, medical devices, and IoT.
However, the growing complexity of software-driven functionalities brings significant cybersecurity risks.
Fuzz testing has emerged as a critical technique to expose hidden vulnerabilities in protocols and applications.
www.yagoba.com

YAGOfuzz is a powerful, customizable black-box fuzzing framework designed to address the limitations of conventional fuzzing tools.
With full access to YAGOfuzz's source code, no running license fees, and support for custom configurations, the tool offers both depth and flexibility in vulnerability detection.

With increasing regulatory requirements and the complexity of connected systems, fuzzing is now a security imperative.

Traditional fuzzing tools, while valuable, are often limited by costly licenses, seat restrictions, and rigid configurations that make them difficult to set up and adapt.

YAGOfuzz removes these barriers. With no license or seat limitations, full source code access, and total control over test data, it gives organizations the flexibility to adapt fuzzing to their needs and integrate it seamlessly into their security processes.

Fuzz Testing

Fuzz testing, also known as fuzzing, is a proven technique for discovering vulnerabilities by feeding a system, that we want to test, unexpected or malformed inputs.
Fuzzing requires a tool called the fuzzer, which generates random or unexpected inputs, which are then sent to the Target of Evaluation (TOE). The fuzzer then monitors the TOE for crashes and unexpected behaviour. If the TOE remains stable, new input is created, and the fuzzing process continues. If the crash occurs, the defective input is marked and stored in the crash database.

Fuzzers can be classified based on the access they have to the TOE internals and based on how they utilize this knowledge to improve fuzzing.

Blackbox Fuzzers

Blackbox fuzzers do not employ any techniques to infer knowledge about the TOE. To improve performance and reduce the overhead caused by only sending strictly random inputs, some fuzzers require testers to provide more information about the input format. This knowledge is then used to let the fuzzer know which parts of the input should be randomly generated and which parts must stay constant.

Whitebox Fuzzer

Whitebox fuzzers have access to the internals of the TOE and they try to take advantage of this knowledge to reach the deeper states of the system under test.

Greybox Fuzzer

Greybox fuzzers are considered the “middle ground” between whitebox and blackbox fuzzers, as they have some knowledge about the TOE, in contrast to blackbox fuzzers. Still, this knowledge is less extensive than the one that whitebox fuzzers have. Greybox fuzzers usually employ instrumentation as a method of collecting feedback that will be used to improve the inputs for the TOE.

Fuzzing Techniques

Mutation-Based Fuzzing	Mutates valid inputs to produce variants that test the system's behaviour.
Block-Based Fuzzing	Generates inputs based on a structural knowledge of the data (focusing on the ability to carry dynamic values).
Model-Based (Smart) Fuzzing	Utilizes comprehensive protocol or format knowledge, and sometimes even a state machine, to produce efficient, targeted inputs.
Evolutionary Fuzzing	Uses mutations and feedback to refine and improve input generation over time.

Integration with SDLC

Fuzz testing can be integrated at various stages of the software development lifecycle (SDLC), including development, integration testing, and release. It complements traditional static and dynamic analysis methods by simulating real-world input conditions that are difficult to anticipate.

Benefits of Fuzzing

Identifies zero-day vulnerabilities	Enhances code robustness	Ensures better input handling	Meets regulatory expectations
It uncovers previously unknown security flaws before attackers can exploit them.	By exposing systems to unpredictable inputs, it helps strengthen code against crashes and edge cases.	It simulates real-world misuse, helping developers validate and harden input validation logic.	It supports compliance with industry standards that require proactive security verification.

Limitations in Existing Fuzzing Tools

Despite its proven effectiveness, fuzz testing adoption has been hindered by several persistent shortcomings:

Configurability
Many fuzzers lack flexible configuration mechanisms, especially when it comes to defining what parts of the input should be fuzzed and which should remain static. Moreover, configuration files are often written in proprietary formats that are difficult to understand, reuse, or adapt across different protocols or systems.

Handling of Stateful Systems
Network protocols and complex applications often involve multi-step interactions where the system's response depends on its internal state or previous inputs. Most fuzzers are designed for stateless testing and fail to track the state transitions required to test such systems effectively.

High Operational Costs
Traditional tools impose their rigid and expensive licensing models. Many leading fuzzers charge per seat or per project, often requiring annual subscriptions. These costs escalate rapidly in large-scale environments such as automotive development, where multiple teams, ECUs, and test environments are in constant flux.

YAGOfuzz was created to address these limitations head-on, offering a modern, flexible, and cost-effective approach for fuzzing at scale.

YAGOfuzz

Not just a black-box fuzzer, but a scalable fuzzing framework designed for easy configuration, stateful system support, and smarter bug detection.

Easy, Flexible Configuration	YAGOfuzz introduces a unified configuration format that defines the structure of the input packets, specifies which fields are to be fuzzed, and allows for the inclusion of fixed values where necessary. The configuration format can be reused across different protocols or projects, saving time and reducing the overhead associated with setting up new fuzzing campaigns.
Support for Stateful Systems	Many fuzzers only work with single, stateless inputs. YAGOfuzz is different. It can model multi-step interactions using built-in finite state machine logic. That means you can test login sequences, authentication flows, or any protocol that requires a conversation, not just one-off requests.
Smarter Bug Detection	Not every bug causes a crash. YAGOfuzz goes beyond simple fault detection by watching system logs, outputs, and other behavioural signals that indicate something went wrong, helping you catching both visible and hidden issues.
Scalable Architecture	YAGOfuzz's architecture is modular and supports distributed operation, allowing multiple Workers to run on different machines, all reporting back to a centralized Monitor. This makes YAGOfuzz well-suited for large-scale fuzzing campaigns that target complex systems. The ability to start and stop Monitors without interrupting ongoing fuzzing processes further enhances its flexibility.

Modular Architecture

The YAGOfuzz framework is organized into three key components:

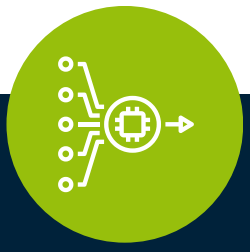
- Worker(s) are the engines that generate test data, communicate with the Target of Evaluation, handle the database, and monitor direct and indirect TOE outputs. They execute the actual fuzzing, supporting a parallel execution.
- Monitor oversees the fuzzing process by tracking worker activity and TOE status. It provides a central dashboard for managing and observing ongoing operations.
- Target of Evaluation (TOE) represents the item being tested, such as a device, system, or product.

This modular design makes YAGOfuzz highly adaptable, enabling it to scale across multiple environments while maintaining clear separation of responsibilities.

Monitor

Dashboard for the fuzzing process

- Status observation of Worker(s)
- Status check of TOE



Worker(s)

Main fuzzing engine(s)

- Data Creation
- TOE Communication
- Output/File Watching
- Start/Reset/Stop Mechanism

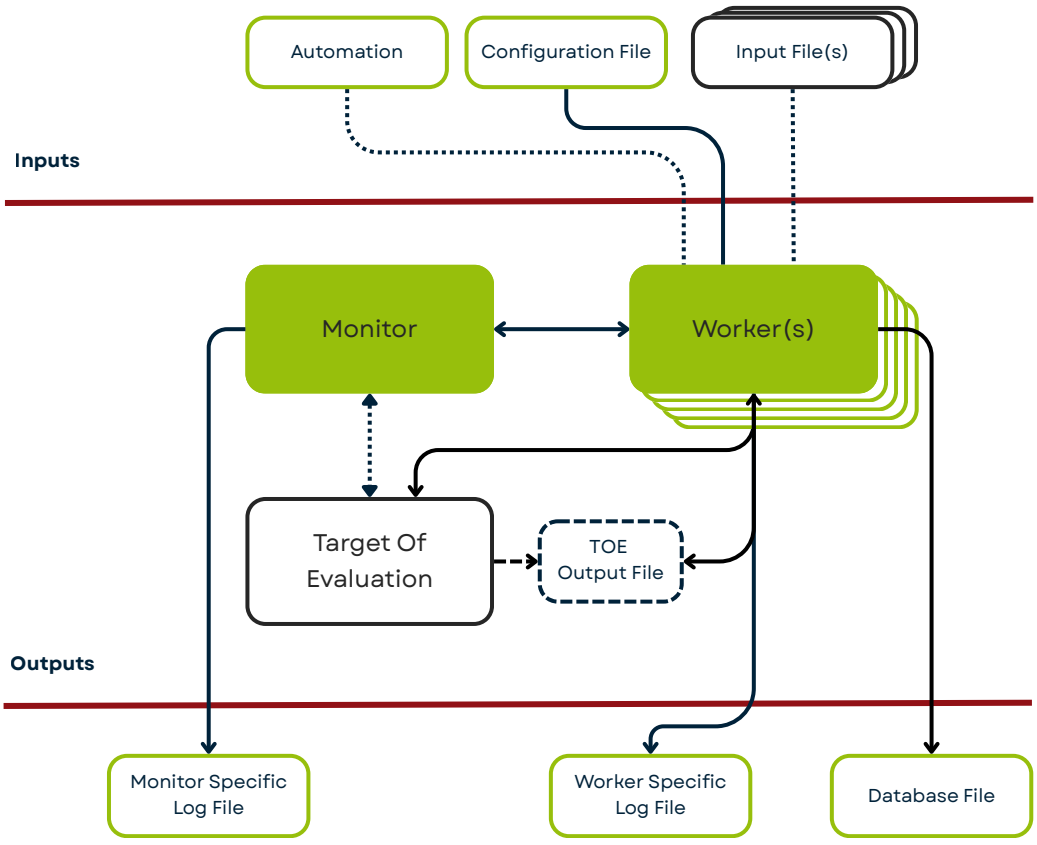
TOE

Item being tested.
Also called:

- Device under test (DUT)
- System under test (SUT)
- Product under test (PUT)

Architecture Overview

YAGOfuzz architecture consists of a modular Worker core connected to monitoring tools and the Target Of Evaluation (TOE).
The architecture is fully extensible and headless-ready, enabling seamless integration into CI pipelines and protocol-specific test benches.



Fuzzing Tools Compared

Unlike traditional tools, YAGOfuzz combines external observability, protocol awareness, and flexible state modeling without requiring source code access. This makes it more adaptable and cost-effective for real-world, stateful systems.

YAGOfuzz vs other Blackbox Fuzzers

Greater control and support for complex stateful systems

Most blackbox fuzzers operate as rigid binaries with limited customization. YAGOfuzz, on the other hand, is a flexible framework that allows users to craft and fine tune inputs via Python and perform true stateful fuzzing using finite state machine models, making it ideal for complex real-world protocols.

YAGOfuzz vs Whitebox Fuzzers

Accessibility and deployment flexibility

Whitebox fuzzers require TOE source code and deep instrumentation, which limits their use in proprietary or embedded environments. YAGOfuzz requires no source access. It integrates easily with CI/CD pipelines, making it practical and scalable for agile security teams.

YAGOfuzz vs Greybox Fuzzers

External control and observability

While greybox fuzzers rely on lightweight instrumentation and internal feedback, they often fail with stateful or session-based protocols. YAGOfuzz instead emphasizes external control and observability. It defines complete test flows and monitors system behavior from the outside to define full test flows and track outcomes.

	YAGOfuzz	Other Blackbox Fuzzers	Whitebox Fuzzers	Greybox Fuzzers
Stateful Protocol Support	Full FSM modeling	Very limited	Code-based	Limited heuristics
No Source Code Required	Interface-level only	Interface-level only	Required	Sometimes
Flexible Configuration	Reusable & intuitive	Often static	Hardcoded or missing	Minimal
Crash + Log Monitoring	Multi-layer (logs, output, behavior)	Basic crash only	Internal tracing	Coverage-based
Protocol Coverage	Broad: Internet, Automotive, IOT, Smartcards + easily extendible.	Protocol-specific	Local code only	File or CLI focused
CI/CD Integration	Built-in compatibility	Manual or wrapper-based	Challenging	Possible with effort
Distributed Scalability	Native multi-instance support	Single-node focus	Often serial execution	Limited parallelism
Ease of Use for Dev Teams	CLI, config, Python-based	Expert-only setup	High complexity	Moderate scripting
Observability Without Instrumentation	External system/log analysis	None	Internal only	Coverage metrics
Enterprise-Readiness	Designed for modern security teams	Too narrow	Too heavy	Limited by interface types

YAGOfuzz stands out in the crowded landscape of fuzz testing tools by offering a unique combination of flexibility, usability, and real-world effectiveness that positions it ahead of other blackbox fuzzers, and indeed, ahead of many whitebox and greybox fuzzers currently in use. It's designed from the ground up to make fuzzing more intuitive, reusable, and scalable, without compromising on depth or flexibility.

Case Study

Vehicles have evolved from simple machines with isolated control systems to complex, interconnected networks powered by tens of millions of lines of code. While this shift has greatly enhanced performance and connectivity, it has also introduced significant cybersecurity challenges. The need for robust security testing is no longer optional, but it is a regulatory and operational imperative.

Automotive Software

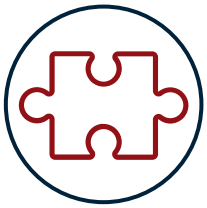
Modern vehicles are becoming more connected and reliant on complex software systems, enabling advanced functionalities such as autonomous driving and real-time data integration. This increasing complexity, however, expands the potential attack surface, making cybersecurity a critical concern. Fixing vulnerabilities after production can be both costly and harmful to a manufacturer's reputation.
Regulatory standards mandate that automotive components demonstrate resilience against cyber threats, including malformed inputs, protocol misuse, and logic manipulation. Identifying and resolving software issues early in the development process is essential to ensure both compliance and long-term product integrity.

Road Vehicle Cybersecurity and Safety Standards

ISO/SAE 21434

UNECE WP.29 R155

ISO 26262



Automotive SPICE

YAGOfuzz supports compliance with key automotive cybersecurity and safety standards, delivering stateful fuzzing, customizable protocol testing, and comprehensive monitoring for robust system protection.

Challenges

Complex Regulations

Industry standards and compliance checks, though essential for quality, introduce delays and higher development costs.

Emerging Threat Landscape

New security threats require continuous, automated discovery techniques.

Software Complexity

The growing size and interconnectivity of automotive codebases make manual testing impractical.

YAGOfuzz

A Strategic Imperative

YAGOfuzz delivers strategic, stateful fuzzing for automotive systems with customizable protocols, multi-target support, CI/CD integration, and advanced monitoring.

1

Customizable Protocol and Payload Modeling

2

Stateful Fuzzing with Automaton Integration

3

Hardware Flexibility and Multi-Target Support

4

Headless Execution and CI/CD Integration

5

Advanced Monitoring, Logging, and Repeatable Fuzzing Campaign

YAGOfuzz Results & Impact

The adoption of YAGOfuzz led to measurable improvements in security, development efficiency, and regulatory compliance.

Increased Vulnerability Detection	YAGOfuzz has uncovered a wide range of critical issues, including input validation flaws, logic bugs, protocol misimplementations, and crashable states in ECUs.
Accelerated Time-to-Fix	Thanks to its seamless integration with CI/CD pipelines and support for automated, repeatable test cases, teams have significantly shortened the window between bug discovery and resolution.
Enhanced Compliance Support	By generating structured, protocol-aware fuzz tests and comprehensive logs, YAGOfuzz helps satisfy key requirements of automotive standards.
Cost-Efficient Scalability	The absence of per-seat licensing and the open framework design allow organizations to conduct parallel testing across multiple test benches, ECUs, and environments without incurring additional operational costs.

YAGOfuzz goes beyond traditional fuzzing by combining cost efficiency, openness, and scalability with expert support. Its extendible design adapts to complex environments, while YAGOBA's customization services ensure every deployment meets the unique needs of the organization.
This combination of technology and expert support transforms YAGOfuzz from a security tool into a long-term strategic partner.
If your organization is ready to strengthen resilience, meet regulatory expectations, and reduce security risk with a cost-efficient and adaptable solution, the next step is clear: connect with YAGOfuzz experts and start fuzzing smarter today.

YAGOfuzz Experts

Christoph Herbst

Chief Administrative Officer

Vanessa Di Benedetto

Sales Manager

Srđan Ljepojević

Product Owner

Ready to move beyond traditional fuzzing?

Start fuzzing smarter with YAGOfuzz

If you are interested in knowing more about YAGOfuzz, you can connect with us via these contact details :

fuzzing@yagoba.com

www.yagoba.com

Beethovenstraße 20, Graz (AT)


YAGOfuzz
SCALABLE FUZZING FOR CONNECTED SYSTEMS