bugcrowd

Ultimate Guide to **fuzz testing**

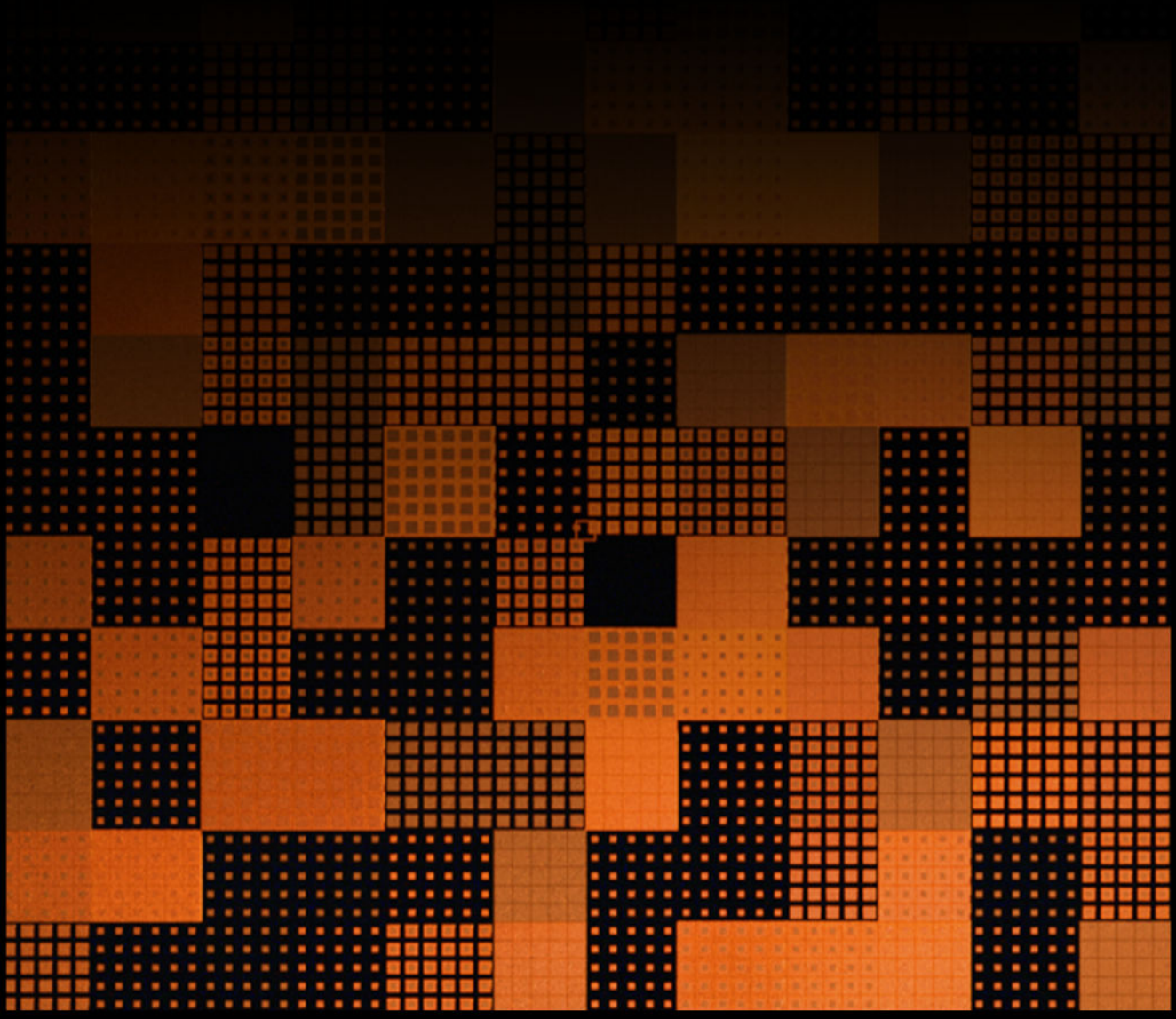


Table of Contents

Introduction

Software is everywhere, powering the planes we fly, the cars we drive, the medical devices keeping us alive, and the financial systems managing our money. As software has become the backbone of modern life, the consequences of vulnerabilities within it have never been heavier. **Yet many organizations still rely on security testing approaches that only find what they already know to look for.**

That's the gap fuzz testing was built to close. Fuzzing is a dynamic security testing technique that goes beyond identifying known vulnerabilities and common attack patterns. Instead of checking boxes against a list of CVEs, fuzz testing throws unexpected, malformed, and random inputs at software to uncover the unknown weaknesses that traditional tools miss—the kind that sophisticated attackers actively seek out and exploit.

Once the exclusive domain of elite security researchers, fuzzing has matured dramatically. Today, developers, security teams, and test engineers can integrate fuzzing into their workflows without deep specialized expertise. The tooling has caught up to the need.

This eBook is your guide to understanding fuzz testing from the ground up: what it is, how it works, where it fits alongside other security techniques like penetration testing and bug bounty programs, which industries need it most, and what regulations are beginning to require it.

Whether you're a developer looking to write your first fuzz harness, a security professional building out a testing program, or a business leader trying to understand your organization's risk exposure, this guide will help you establish the foundation for informed decisions.

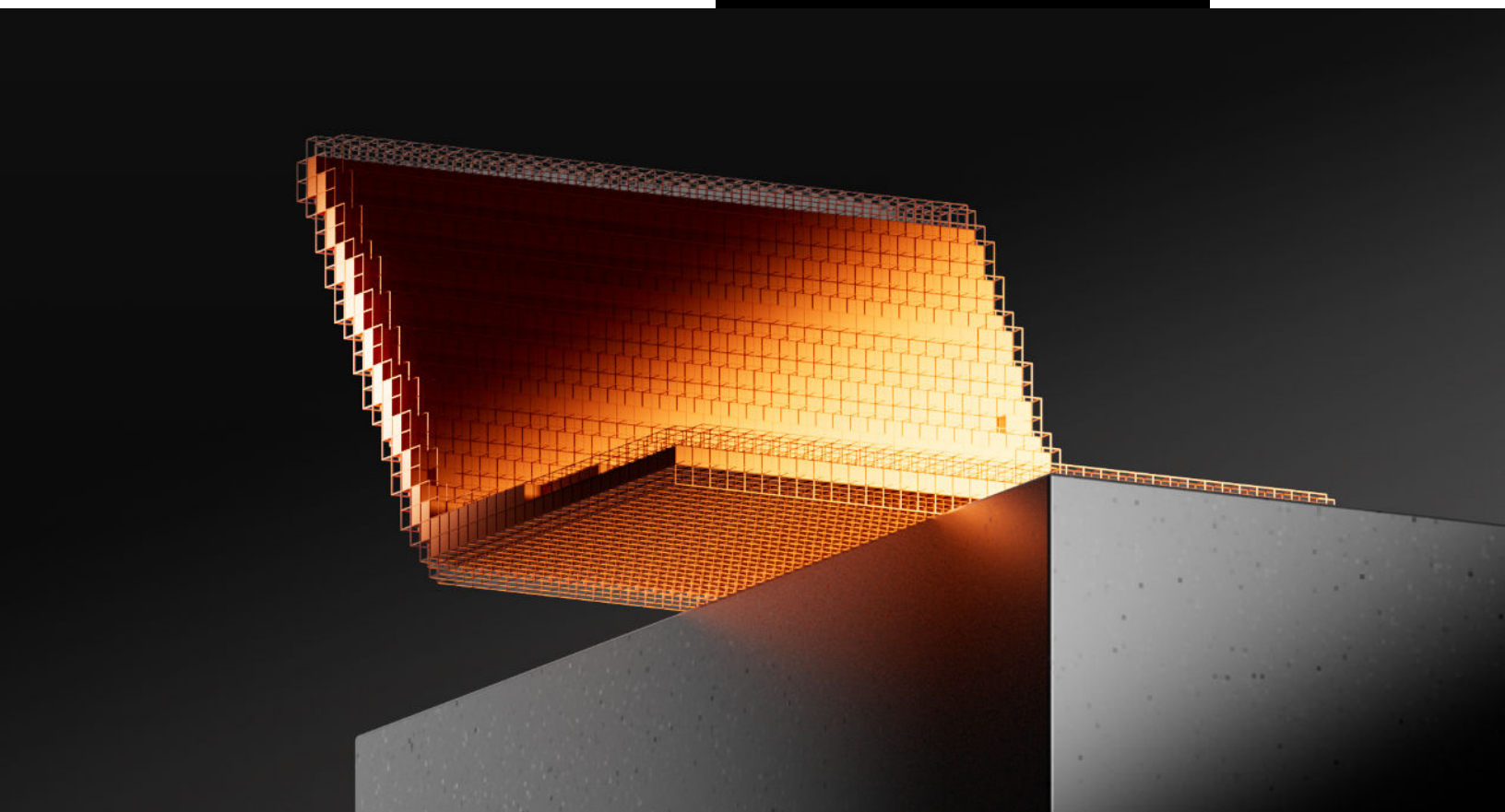
What is fuzz testing?

Fuzz testing is a **technique for discovering vulnerabilities** by injecting invalid, unexpected, or random data into a program to trigger bad behaviors (like crashes, infinite loops, or memory leaks), which can indicate underlying vulnerabilities.

Fuzzing tools can take many shapes and forms. Some randomly generate inputs, while others use templates (grammar-based fuzzing) or analyze their target application's behavior to generate inputs (behavioral or guided fuzzing). Guided fuzzing tools often use advanced techniques such as dynamic analysis and symbolic execution to systematically explore program paths and generate inputs that reveal deep vulnerabilities.

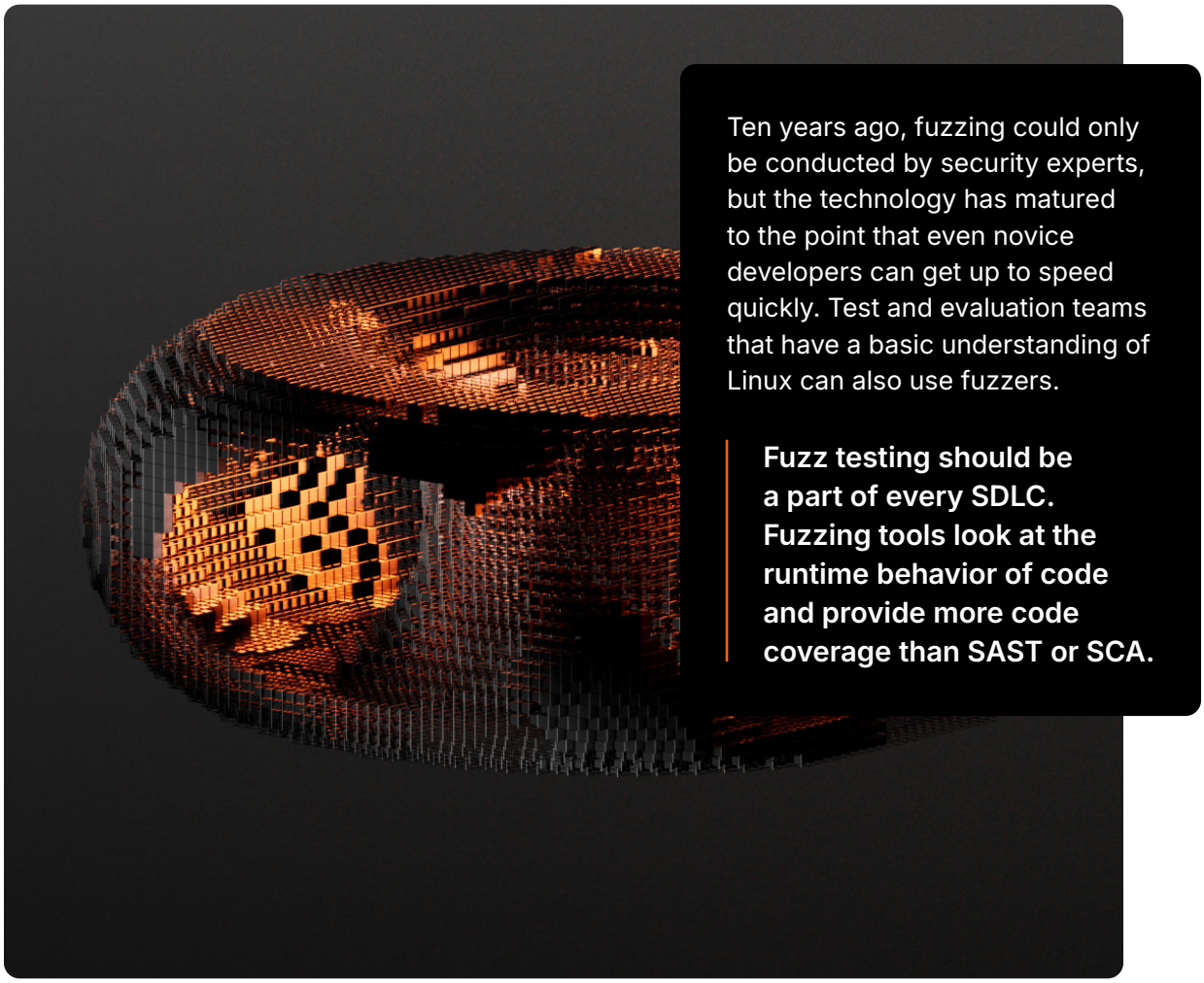
As a result, fuzz testing is effective for uncovering novel vulnerabilities often missed by traditional security tools.

E.g., static analyzers or scanners that address only known vulnerabilities.



What is a fuzz testing tool?

A fuzzing tool can be used to create a test case and send malformed or random inputs to fuzz targets. **These tools' objective is to trigger bad behaviors,** such as crashes, infinite loops, and/or memory leaks. These anomalous behaviors are often a sign of an underlying security vulnerability.



Ten years ago, fuzzing could only be conducted by security experts, but the technology has matured to the point that even novice developers can get up to speed quickly. Test and evaluation teams that have a basic understanding of Linux can also use fuzzers.

> **Fuzz testing should be a part of every SDLC. Fuzzing tools look at the runtime behavior of code and provide more code coverage than SAST or SCA.**

How does fuzzing work?

Fuzzers send malformed inputs to targets. Their objective is to trigger bad behaviors, which often signal an underlying vulnerability.

What is a seed corpus?

A seed corpus is a set of valid inputs that serve as a starting point for fuzzing a target.

What do you need to fuzz test?

To fuzz test, a fuzzer needs a way to interact with the application. Unit and integration tests both typically involve running the software under test with a specific input and asserting that a specific output was observed.

Fuzzing extends this form of testing by parameterizing the test within an array of bytes and then searching for strings of input bytes that trigger bugs. Fortunately, developers can write a fuzz test harness in much less time than required to write individual unit tests. Better yet, these harnesses typically only need to be written once for a given application.

What is fuzzing used to test?

The rise in fuzzing has resulted in security vulnerabilities getting found and fixed at an amazing rate. But it has raised some new questions: How do we find good fuzz targets quickly, and what is left to fuzz? These questions require tools and workflows that remain uncommon among software developers and security researchers alike, and one potential solution lies in automated coverage analysis.

What is automated coverage analysis?

Automated coverage analysis—also known as code coverage analysis or test coverage analysis—is a process used in software testing to measure how much of the code is exercised by test cases. It automatically checks which parts of the program's source code have been executed during testing, helping developers identify areas that need more testing and ensuring better software quality.

Types of fuzzing tools

Fuzzing has been around for nearly four decades. Each generation of **fuzzing software has evolved** and adapted to the needs of its time. Below are a few different types of fuzzing tools that have emerged over the years.

Random fuzzing tools

Random fuzzing involves sending random inputs to an application. There is no systematic method with a mutation fuzzer, so the inputs of this random testing might not even penetrate applications! This is sometimes compared with monkeys typing on a keyboard.

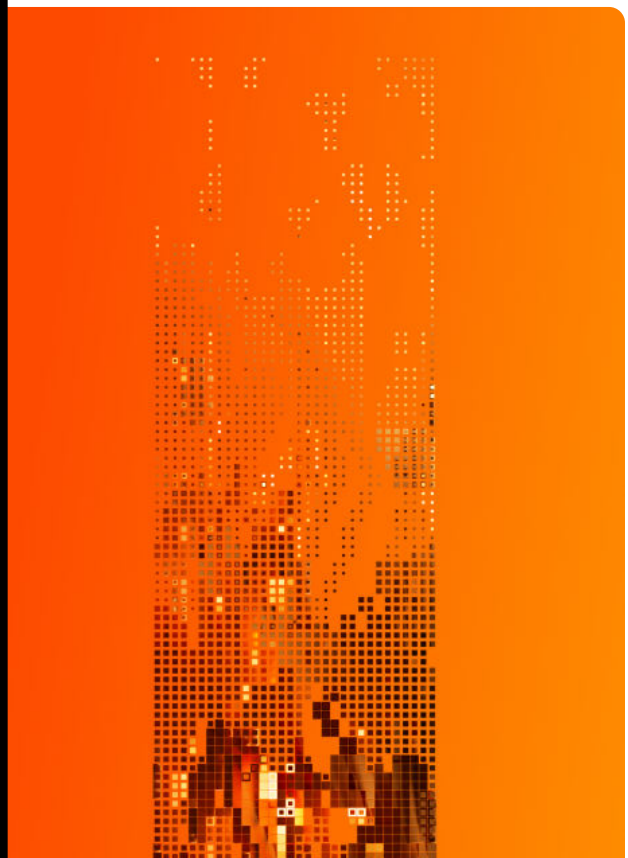
Behavioral or guided fuzzing tools

Guided fuzzers rely solely on their target application's behavior to inform how to generate inputs. For example, the fuzzing engine will generate an input, observe how the application behaves, learn from it, then generate the next input. These fuzzers are most effective because they custom generate tests specific to the applications.

Template or grammar-based fuzzing tools

Grammar-based fuzzers rely on a template that's manually generated. It informs the fuzzing engine how to generate each input. Typically, the individuals creating these templates have an understanding of how the protocol is built, so there is intelligence around how those inputs are generated.

> **However, these templates can inadvertently constrain the areas of applications to explore and take a one-size-fits-all approach.**



What industries should prioritize fuzz testing?

Fuzz testing is relevant across many different industries like aerospace, defense, automotive, healthcare, financial services, telecommunications, cloud and infrastructure, firmware, and software and SaaS.

Industries should prioritize fuzz testing when they have any of these characteristics:

- **Memory-unsafe code** (C, C++, and Assembly) in the codebase
- **Complex input parsing** from untrusted or external sources
- **Safety or life-critical** consequences from software failure
- **High regulatory scrutiny** with demonstrable testing requirements
- **Large, legacy codebases** with limited original test coverage
- **Network-exposed protocol handlers** accepting structured data
- **Long patch cycles** where finding bugs early is far cheaper than post-deployment

Regulations that require fuzz testing

Fuzz testing is legally mandated for certain industries. For example, ED-203A requires any entity involved in aviation to ensure aircraft systems are safe from cyber threats. **ED-203A** requires organizations to meet a new standard: refutation, which requires organizations to show the presence or absence of a vulnerability in a test scenario.

Another example is the Digital Operational Resilience Act **(DORA)** in the EU, which requires financial entities to conduct advanced threat-led penetration testing. This includes vulnerability testing of critical systems where fuzz testing is an expected component.

For medical devices, the **FDA Cybersecurity Guidance** explicitly lists fuzz testing as an expected security testing method for premarket submissions.

Why should you fuzz?

A review of software security investments reveals that **a majority of spending is in application testing solutions,** such as static analysis, software composition analysis, and scanners. However, these conventional testing approaches test known or common attack patterns, only addressing CVEs or CWEs.

What about the unknown vulnerabilities—the weaknesses malicious hackers often exploit?

Combine fuzzing with software security techniques for the best coverage

You may have invested quite a bit of time, money, and effort into your SAST solution. So, you may not be warm to the idea of breaking up with it. Great news! You don't have to.

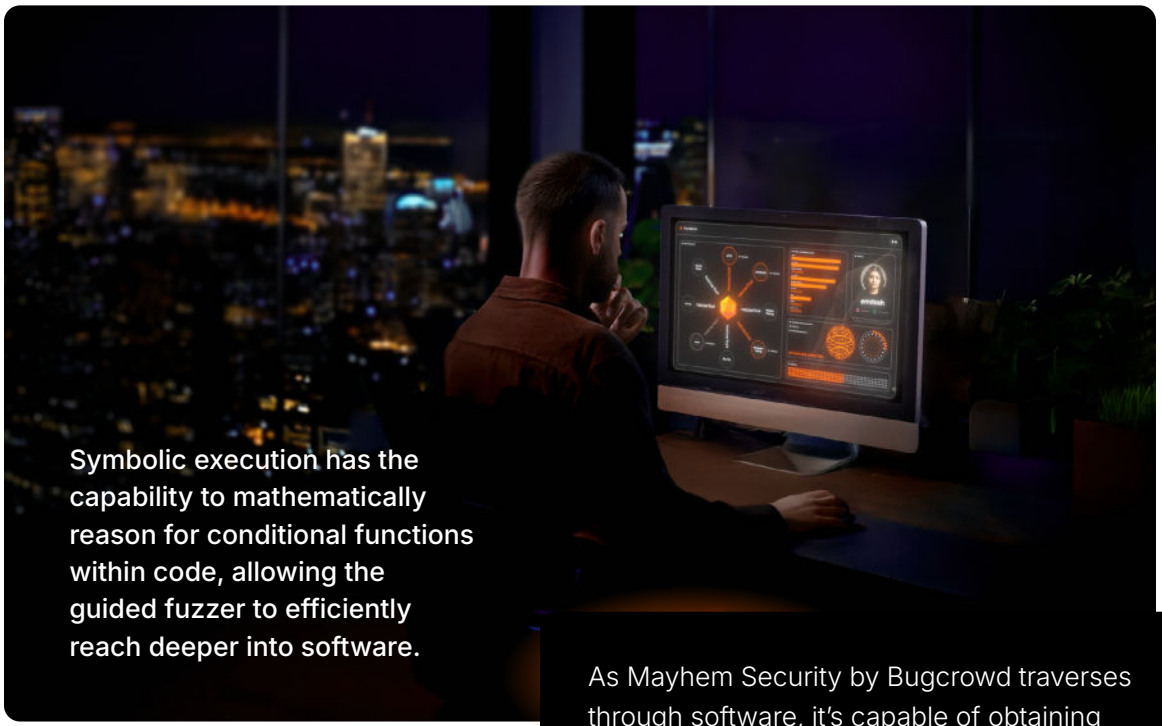
> **When it comes to product security, best practices call for layering. Much like defense in depth, consider it like testing in depth. Assess the gaps left behind by your SAST solution, then identify additional solutions for augmentation.**

Fuzz testing is a complement to Dynamic Application Security Testing (DAST) for negative testing. DAST solutions that use fuzzing are proven to maximize defect detection with the least amount of time and resources.

As a result, these tools not only buy organizations time and money but also free scarce technical resources from manual, mundane tasks and allow them to focus on strategic initiatives that require true expertise.

How Bugcrowd combines several security testing techniques

In 2025, **Bugcrowd acquired Mayhem Security,** a leading provider of offensive security built on the tried-and-true methods of coverage-guided fuzzing by combining it with the ingenuity of symbolic execution.



Symbolic execution has the capability to mathematically reason for conditional functions within code, allowing the guided fuzzer to efficiently reach deeper into software.

Its systematic and thorough approach enables Mayhem to uncover at least 25% more defects than using coverage-guided fuzzing alone.

As Mayhem Security by Bugcrowd traverses through software, it's capable of obtaining knowledge of its software under test (SUT) over time. It takes in feedback from its targets to influence the autonomous generation of future test cases, increasing the likelihood of uncovering deeper defects. This approach offers scalability advantages over manual penetration testing efforts and enables both security and development teams to spend less time on tedious vulnerability management efforts.

Symbolic execution vs. concolic execution in software security

"Symbolic execution" usually means "static symbolic execution" in which you analyze a non-executing program to consider how it might behave when it does execute for real. Symbolic execution is a program analysis technique that uses formal computer science methods to determine an input that triggers a node in the application to execute.

Once determined, the valid input is used to derive invalid inputs for negative testing.

Concolic execution is a form of dynamic symbolic execution and a type of analysis that runs on a trace of a program that ran (for real) on a specific input.

Mayhem Security by Bugcrowd technically only does concolic execution, not static symbolic execution, which is what someone might mean when they say "symbolic execution." When we say it, we mean in the more general sense of "static or dynamic symbolic execution," of which we do one of those things.

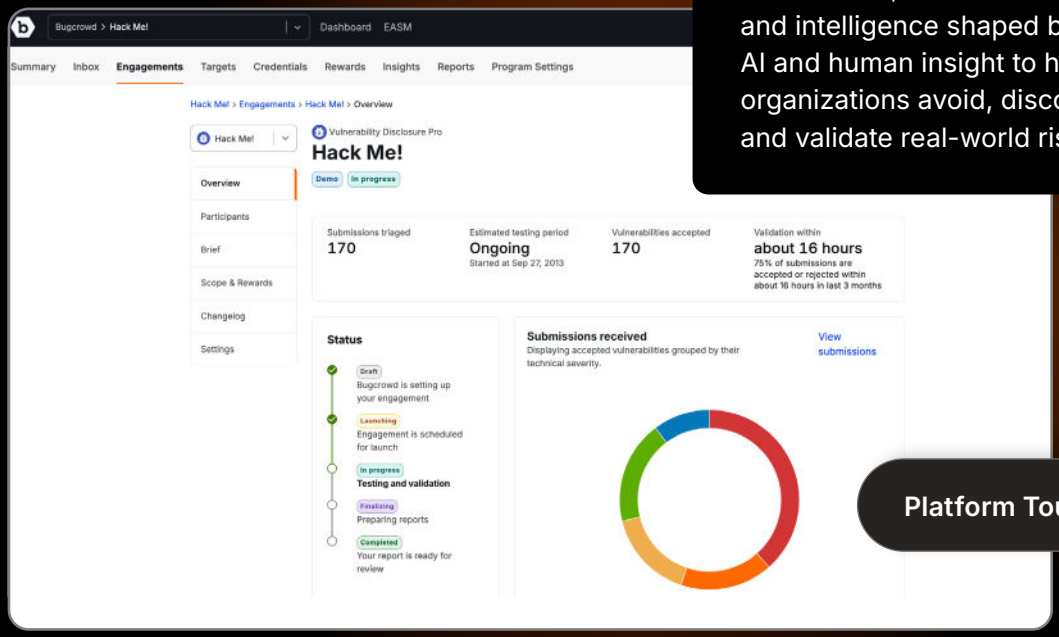
Complementing fuzz testing with pen testing and bug bounty

Fuzz testing is an excellent complement to a bug bounty program and traditional AppSec solutions. Fuzz testing feeds bug bounties, and bug bounties improve fuzzing strategy. Security teams benefit from coverage at scale and vulnerability class intelligence when combining these two tactics.

Pen testing and fuzz testing are also a strong pairing. While fuzz testing is automated with a wide breadth and shallow reasoning, pen testing is more goal-oriented and finds flaws deep within a system, revealing issues such as chained exploits and logic flaws.

See the Bugcrowd Platform in action

Take a **5-minute tour** to get an overview of how the Bugcrowd Platform unifies exposure discovery and assessment, offensive testing, and intelligence shaped by AI and human insight to help organizations avoid, discover, and validate real-world risk.



Platform Tour

Unleash human and AI ingenuity for preemptive security

Try Bugcrowd