YOUR ULTIMATE GUIDE

TO FUZZING

Mayhem

TABLE OF CONTENTS

2 What is Fuzzing?

3 What Is A Fuzz Testing Tool?

4 How Does Fuzzing Work?

5 What Is Fuzzing Used To Test?

6 Types Of Fuzzing Tools

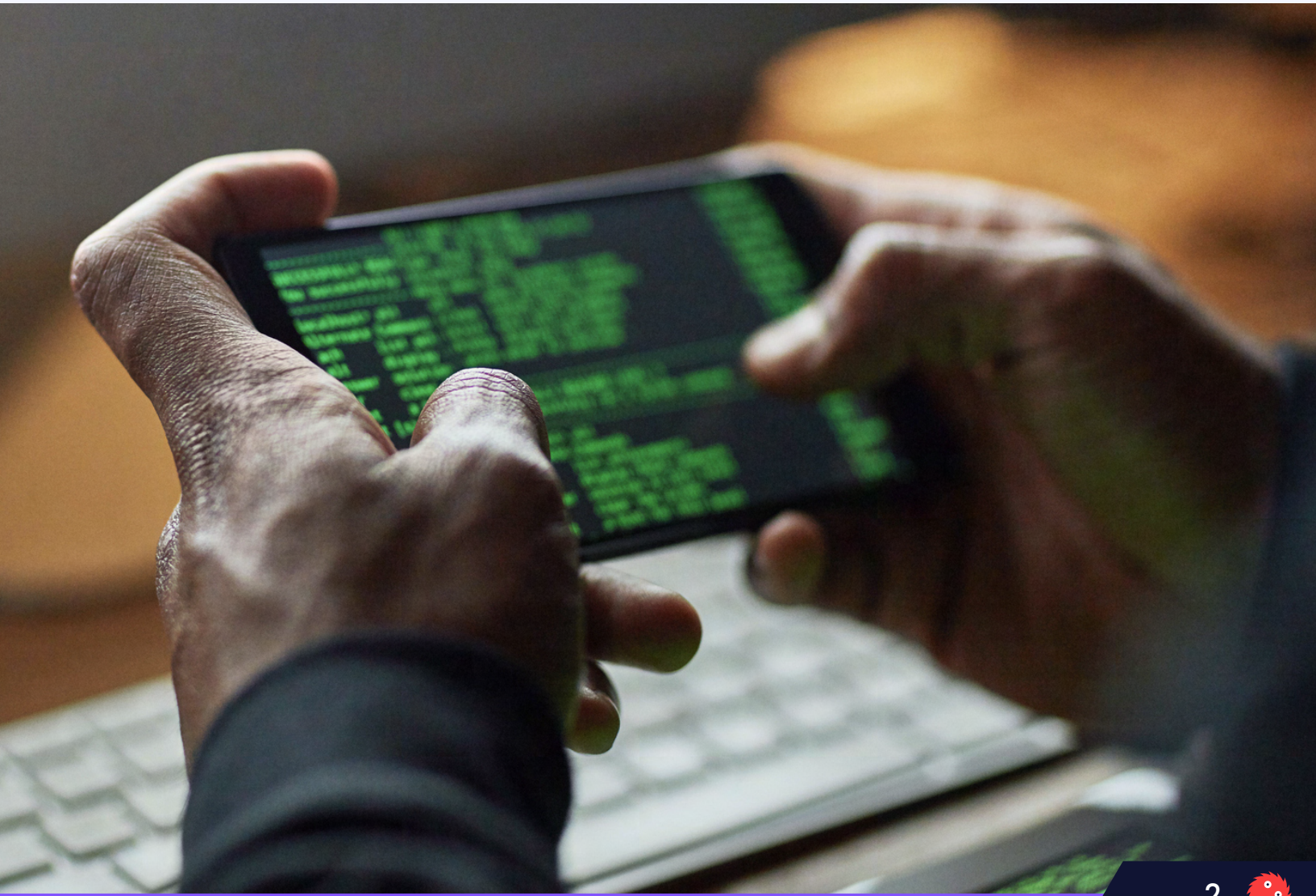
7-8 Why Should You Fuzz?

9-10 How Mayhem Combines Techniques

10 Interested In Trying Mayhem?

WHAT IS FUZZING?

Fuzzing, or fuzz testing, is a DAST (Dynamic Application Security Testing) technique for negative testing. **Fuzzing aims to detect known, unknown, and zero-day vulnerabilities.**



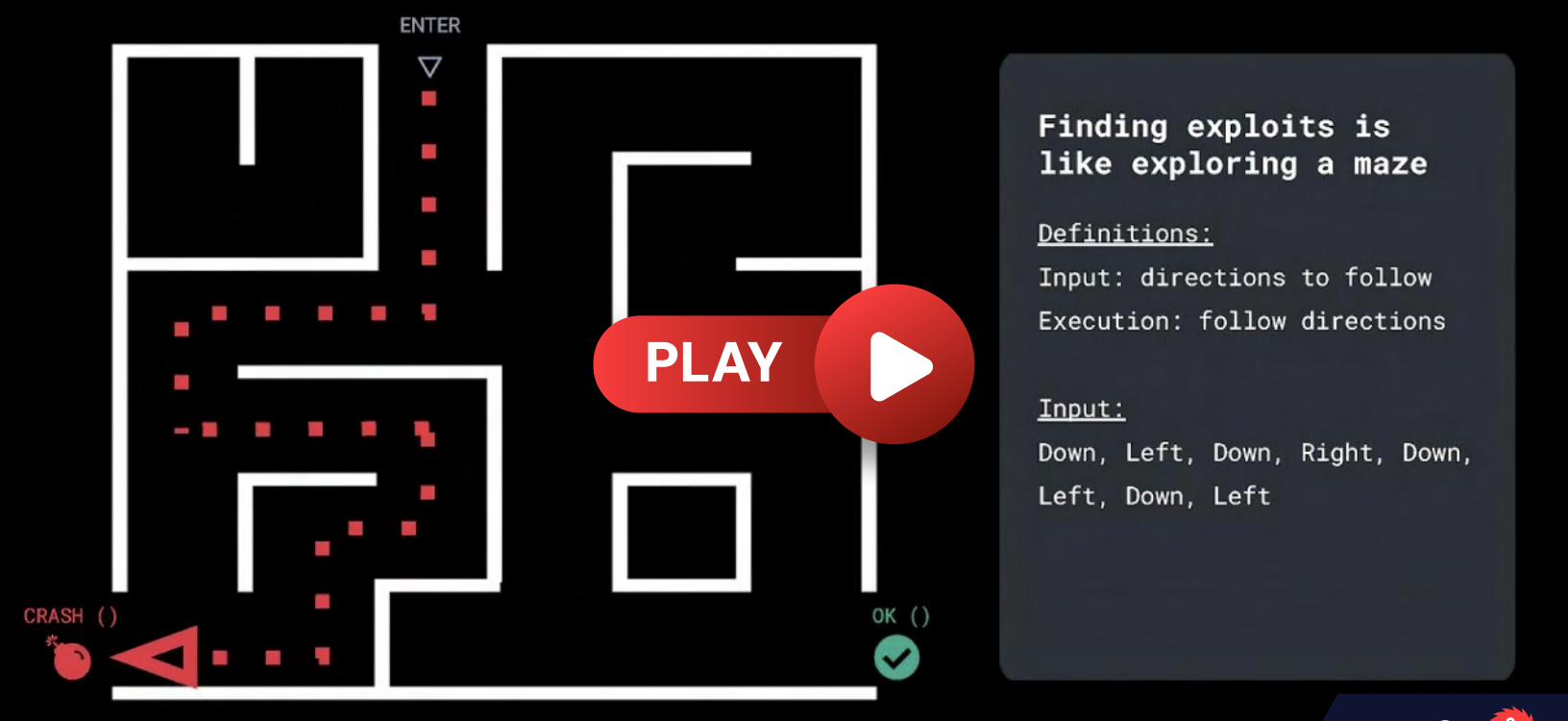
WHAT IS A FUZZ TESTING TOOL?

A fuzzing tool can be used to create a test case and send malformed or random inputs to fuzz targets. Their 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 the code and provide more code coverage than SAST or SCA.

Fuzzing 101 - Testing Tools: https://www.youtube.com/watch?v=twaIAioyaEc&t=490s



HOW DOES FUZZING WORK?

Fuzzers send malformed inputs to targets. Their 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 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?

In order to fuzz test, a fuzzer needs a way to interact with the application. Unit tests 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.

Why Fuzzing Works: mayhem.security/CITHf2

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 is in automated coverage analysis.

What is Automated Coverage Analysis?

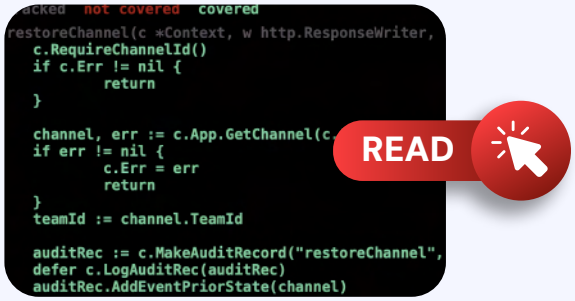
Automated Coverage Analysis—also known as Code Coverage Analysis or Test Coverage Analysis—is a process used in software testing to measure how much of the code is exercised by the test cases. It automatically checks which parts of the program's source code have been executed during testing, helping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

Read More:

Why Fuzz Test: 20 Mozilla Vulnerabilities Found With Fuzz Testing

mayhem.security/ywvujuyx

How to Increase Test Coverage (And Confidence!) With Mayhem in 4 Easy Steps



mayhem.security/3xcNlDl

TYPES OF FUZZING TOOLS

Fuzz testing is a technique that has been around for nearly four decades. With each generation of fuzzing software, we're seeing evolution at play, adapting 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 is 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 the applications! This is sometimes compared with monkeys typing on a keyboard.

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 the applications to explore and take a one-size-fits-all approach.

Behavioral or Guided Fuzzing Tools

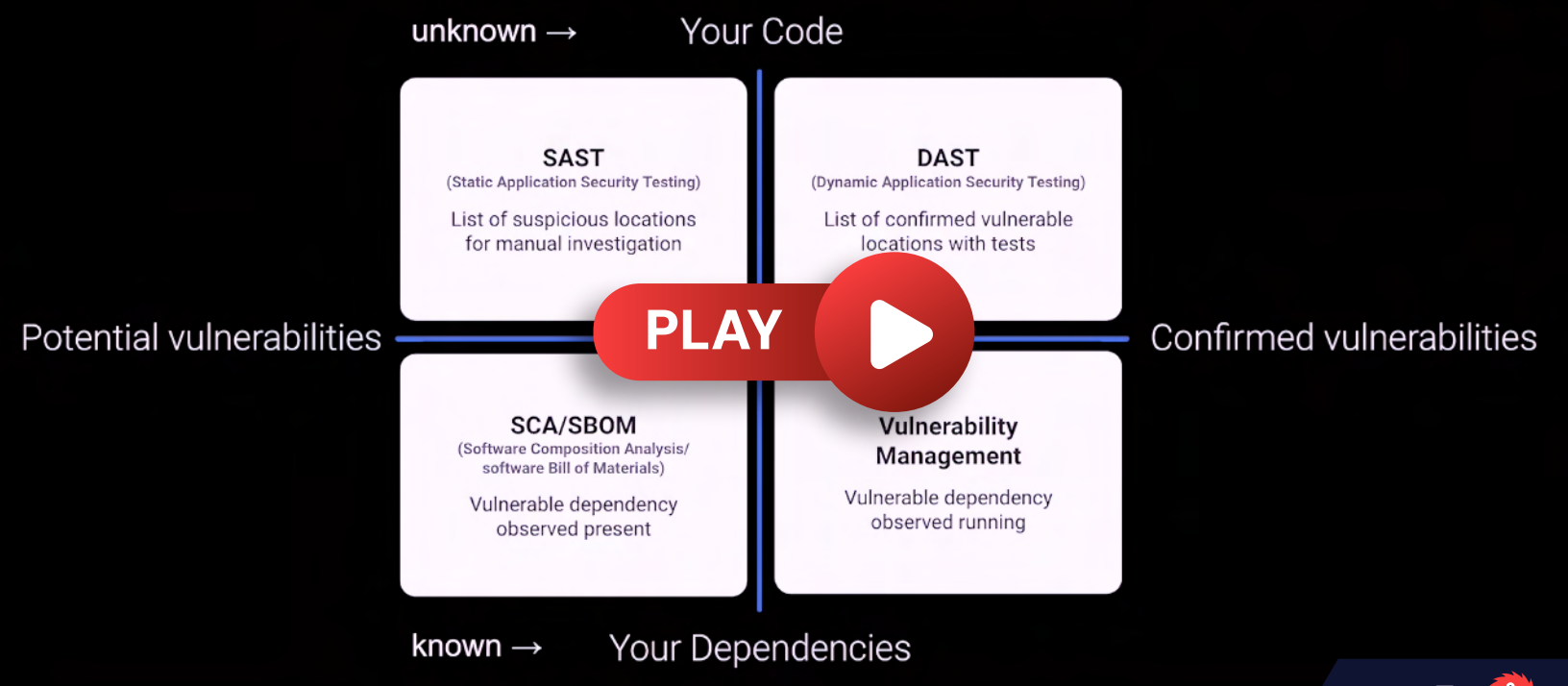
Guided fuzzers rely solely on their target application's behavior to inform how to generate inputs. For example, the fuzzing engine will generate an input, observe how the application behaves, learn from it, then generate the next input. These fuzzers are most effective because they custom generate tests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

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. These conventional testing approaches, however, test known or common attack patterns, only addressing CVEs or CWEs. But what about the unknown vulnerabilities—the weaknesses malicious hackers often exploit?

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.

The Four Corners of Application Security: https://www.youtube.com/watch?v=yztHvvCkE2M



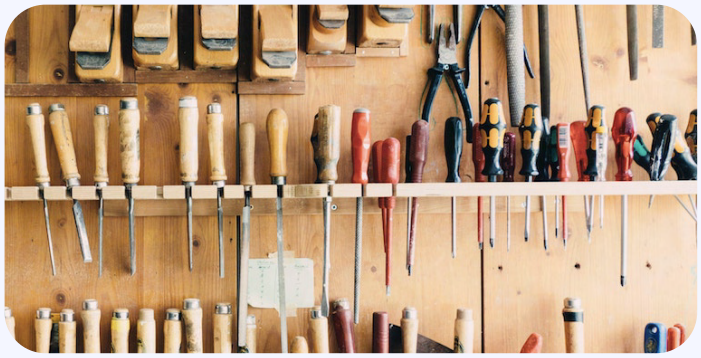
WHY SHOULD YOU FUZZ?

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.

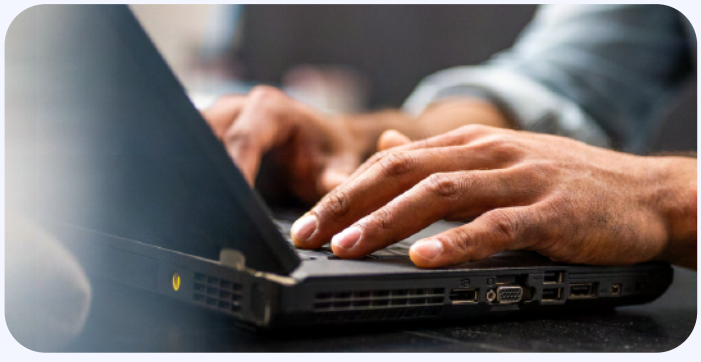
Read More:



SCA, SBOM, Vulnerability Management, SAST, or DAST Tools: Which Is Best for Your Team?

READ

mayhem.security/4eEW5n1



The Buyer's Guide to Application Security Testing

READ

mayhem.security/4bn4ljf

HOW MAYHEM COMBINES SEVERAL SECURITY TESTING TECHNIQUES

Mayhem builds on the tried-and-true methods of coverage guided fuzzing by combining it with the ingenuity of symbolic execution, patented technology from a decade of research at Carnegie Mellon University. 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 traverses through software, it's capable of obtaining knowledge of its software-under-test (SUT) over time. Mayhem 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.

What is Symbolic 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.

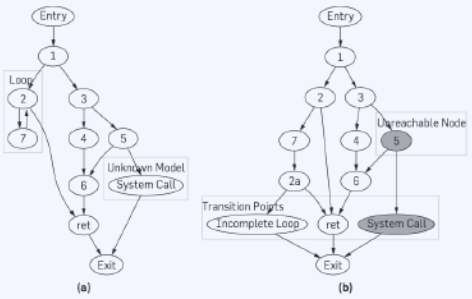
HOW MAYHEM COMBINES SEVERAL SECURITY TESTING TECHNIQUES

What is Concolic Execution in Software Security?

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 technically only does concolic execution, not static symbolic execution. We don't do 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.

Read the Academic Paper:



Enhancing Symbolic Execution with Veritesting:

READ

https://dl.acm.org/doi/pdf/10.1145/2927924

Interested in trying a security testing solution that uses easy fuzz testing?

We can't definitively say that one fuzzing technique or fuzzing framework is better than the other, because it really does depend on what you're trying to address. Regardless of whether you want to go with Mayhem, we can help you navigate those waters to find the right type of security testing solution for you.

SCHEDULE A DEMO

mayhem.security/demo