Fuzz Testing (Fuzzing) Tutorial (What is, Types, Tools, Example)

Fuzz Testing

Fuzz Testing, often known as fuzzing, is a software testing approach that involves injecting incorrect or random data (FUZZ) into a software system in order to find coding errors and security flaws. Fuzz testing involves introducing data using automated or semi-automatic approaches and evaluating the system for different exceptions such as system crashes or built-in code failure, among other things.

Barton Miller, a professor at the University of Wisconsin, invented fuzz testing in 1989. Fuzz testing, often known as fuzzing, is a type of software testing that falls within the security testing umbrella.

What is the purpose of fuzz testing?

- Fuzzy testing usually uncovers the most significant security flaw or vulnerability.
- When combined with Black Box Testing, Beta Testing, and other debugging techniques, fuzz testing produces more effective results.
- Fuzz testing is a technique for determining a software's vulnerability. It is one of the most cost-effective testing methods.
- One of the black box testing techniques is fuzz testing. One of the most frequent methods hackers employs to identify system vulnerabilities is fuzzing.

Steps of Fuzz Testing

The fundamental testing phases are included in the fuzzy testing process.

- Determine the system to be targeted.
- Determine the inputs
- Produce Fuzzed Data
- Run the test with ambiguous data.
- Keep an eye on the system's performance.
- Keep a defect log.

Example of Fuzzers

Fuzzers that modify existing data samples to produce new test data are known as mutation-based fuzzers. This is the most basic and easy technique; it starts with acceptable protocol samples and continues to mutilate every byte or file.

Generation-Based Fuzzers create new data based on the model's input. It starts from the beginning, producing input depending on the requirements.

The most successful fuzzer is PROTOCOL-BASED-FUZZER, which has extensive knowledge of the protocol format being tested. The comprehension is determined by the specification. It entails entering an array of the specification into the tool, then going through the specification and adding irregularities to the data contents, sequence, and so on using a model-based test generating approach. Syntax testing, grammar testing, robustness testing, and other terms are used to describe this process. Fuzzer may create test cases from scratch or from inputs that are valid or invalid.

Protocol-based fuzzing has two major drawbacks

- Testing will not be possible until the specification is complete.
- A lot of important protocols are extensions of already published protocols. Test coverage for new protocols will be restricted if fuzz testing is based on public specifications.
- Sending random input to the software, either as protocol packets or as an event, is the simplest type of fuzzing approach. This approach of using random input to identify vulnerabilities in a variety of apps and services is quite effective. Other methods are also accessible, and they are extremely simple to use. We just need to modify the existing inputs to apply these approaches. We can alter the input simply by swapping the bits.

Types of bugs detected by Fuzz Testing

Assertion failures and memory leaks are two types of vulnerabilities that Fuzz Testing may identify. This approach is commonly employed in big systems where flaws compromise memory safety, which is a serious flaw.

- **Invalid input** − Fuzzers are used in fuzz testing to produce incorrect input that is used to test error-handling algorithms, which is crucial for software that does not have control over its input. Simple fuzzing is a technique for automating negative testing.
- **Correctness bugs** − Some sorts of "correctness" issues can also be detected via fuzzing. For example, a corrupted database, inadequate search results, and so on.

Tools for Fuzz Testing

Burp Suite, Peach Fuzzer, and other online security tools may be utilized extensively in fuzz testing.

- **Peach Fuzzer** − Peach Fuzzer outperforms scanners in terms of coverage and security. Peach Fuzzer can discover both known and unknown threads, unlike other testing tools that can only identify known threads.
- **Spike Proxy** − It's a professional-grade tool that scans web applications for application-level flaws. SPIKE Proxy includes the fundamentals, such as SQL Injection and cross-site scripting, but it's built on a Python architecture that's entirely open. SPIKE Proxy is a Linux and Windows application.
- **Webscarab** − Because Webscarab is developed in Java, it can run on a variety of systems. The Webscarab framework is used to analyze the application, which communicates using HTTP and HTTPS protocols. Webscarab, for example, acts as an intercepting proxy, allowing the operator to monitor and alter browser requests before they are sent to the server. Also, before the browser receives the answer created by the server, it is possible to review and edit it. In this approach, if web scarab discovers a flaw, it will be included in the list of reported problems.
- **OWASP WSFuzzer** − WSFuzzer is a Python-based GPL-licensed application. Currently, software that has been GPL'd is aimed at Web Services. The major target of the current version of OWASPWSFuzzer is HTTP-based SOAP services.

Drawbacks of Fuzz Testing

Fuzz testing has a number of drawbacks −

- Only fuzz testing can provide you a comprehensive view of a security threat or flaws.
- Fuzz testing is less successful when dealing with security risks that don't cause software crashes, such as viruses, worms, Trojans, and other similar threats.
- Fuzz testing can only discover minor flaws or dangers.
- It will take a substantial amount of time to function well.
- Setting a boundary value condition with random inputs is difficult, but most tests are now able to handle this challenge by utilizing deterministic algorithms based on user inputs.

Difficulties in setting up Fuzzers

One of the factors limiting developers from using fuzz testing more widely is the time it takes to set it up.

On the one hand, these technologies are becoming more user-friendly all the time, according to DeMott. “On the other hand, setting up, monitoring, managing, and triaging the problems that come out still requires significant experience, security, and coding.”

The hardest aspect for clients, according to Knudsen, whose firm produces a generational fuzzer, is setting out the data model for the fuzzer to utilize as a template to construct test cases. Customers who utilize common network protocols and file formats may easily use existing templates rather than spending time creating their own with the commercial product, which contains a huge database of test suites.

“Assume you're creating an Internet of Things device, such as a smart thermostat,” Knudsen explained. “It'll have to connect with a server somewhere,” says the author. So you can reuse a lot of current solutions because there are already defined network protocols that people can use.”

However, most applications will still have bespoke code that has to be tested with new data models on top of these old protocols.

Developers may utilize Defensics' software development kit (SDK) to construct bespoke data models, although creating them, as with other generational fuzzing technologies, takes time.

In the past, people who used fuzzing had to be specialized in the field.

It's also critical to incorporate fuzz testing into the DevOps pipeline, as well as to ensure those test failures are properly documented. Developers will be unable to fix the code if tests fail without logging the input data that caused the problem.

ClusterFuzz is a Google tool for managing fuzzing infrastructure and tracking fuzzer performance.DeMott stated, "I initially came up with the name many years ago, but my adviser insisted on altering it."

GitLab has purchased Peach Tech and Fuzzit, two-generational fuzzing technologies that will be integrated into the company's continuous integration and continuous delivery (CI/CD) workflow.

DeSanto stated, "We thought there was a void," alluding to GitLab's previous security testing tool offerings, which included SAST and DAST. Today, our security team uses GitLab Secure as its primary testing tool to check our application's security and dependability.

The objective of GitLab is to make it easier for developers to include fuzz testing into their development processes.

Developers that want to use one of the fuzzers can do so by adding a line to the YAML script that manages the CI/CD process.

According to DeSanto, “people who utilize fuzzing previously had to be experts in fuzzing to use it.” “The benefit of utilizing fuzzing with GitLab is that we know what you're doing. You use us for source code management and continuous integration. Rather than telling [the fuzzer], "Here is the structure of my API," we can scan your source code and generate that specification for you automatically. There's no need for you to try to comprehend the instrument in order to use it.”

Fuzz testing may become a more common and effective component of the testing environment if additional fuzzing technologies are more easily incorporated into developers' processes.

Summary

Fuzz testing is a type of software engineering that identifies the presence of flaws in an application. Fuzzing does not ensure that all flaws in a program will be detected. However, employing the Fuzz approach guarantees that the application is both resilient and safe since it helps to reveal the majority of frequent flaws.