ForAllSecure

# What is Advanced Fuzzing?

# Table of Contents


| | |
|---|---|
| Executive Summary | 3 |
| Application Security 101 | 4 |
| What is a Vulnerability? | 4 |
| Static and Dynamic Analysis | 5 |
| Positive and Negative Testing | 6 |
| What is Fuzzing? | 6 |
| Not All Fuzzers Are Created Equal | 6 |
| ForAllSecure Harnesses the Power of Fuzzing | 8 |
| What is Advanced Fuzzing? | 9 |
| Why Advanced Fuzzing? | 11 |
| Conclusion | 12 |


## ForAllSecure Awards

Winner of

DARPA CGC CYBER GRAND CHALLENGE

Named to

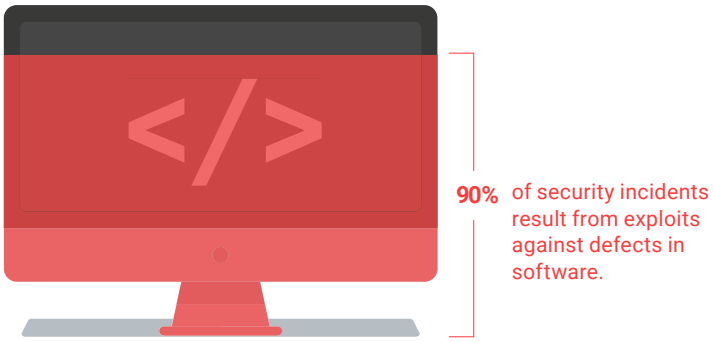
MIT Technology Review 50 SMARTEST COMPANIES

Exhibited at

Smithsonian

# Executive Summary

The fundamentals of software security is simple in theory -- deploy vulnerability-free code. Although most software security investments are in application testing, the Department of Homeland Security (DHS) claims 90% of security incidents result from exploits against defects.[1] Gartner maintains, “vulnerabilities, and the exploitation of them, are still the root cause of most information security breaches today”.[2]




Traditional testing approaches, however, only test a fraction of target software. Despite its fractional coverage, it requires considerable manual labor from scarce security experts. Due to this reason, organizations struggle to validate its value. The universal problem of limited security expertise is slated to grow worse. (ISC)[2], an international nonprofit association best known for the Certified Information Systems Security Professions (CISSP) certification, predicts that by 2022 there will be 1.8 million open jobs in software security. Security testing that relies on human expertise is unsustainable and unscalable.[3]

ForAllSecure Mayhem is an advanced security testing solution that automatically tests applications for exploitable defects. Mayhem builds on top of advanced fuzz testing techniques by combining the tried-and-true methods of guided fuzzing with the ingenuity of symbolic execution, patented technology from over a decade of research at Carnegie Mellon University. With Mayhem, organizations can uncover confirmed risks earlier in the software development lifecycle (SDLC). Shift left testing, a software security approach whereby testing is performed earlier in the SDLC, is proven to keep releases predictable and lowers development costs. With Mayhem, ForAllSecure enables organizations to test intelligently, remediate quickly, and release confidently.

# Application Security 101

Deploying vulnerability-free code is easier in principle than in execution. Software is complex, containing layers of code - ranging from proprietary code to third-party code to free open source components. The majority of software is built from third-party components. Irrespective of code type, all software must be thoroughly tested for vulnerabilities to minimize organizational risk. Afterall, organizations are responsible for all of the code they ship, regardless of whether you wrote them or sourced them.

# What is a Vulnerability?

Defects are born when developers make mistakes in their code. A defect that can be exploited for malicious activities is a vulnerability. Vulnerabilities are risks to an organization. There are three types of vulnerabilities:

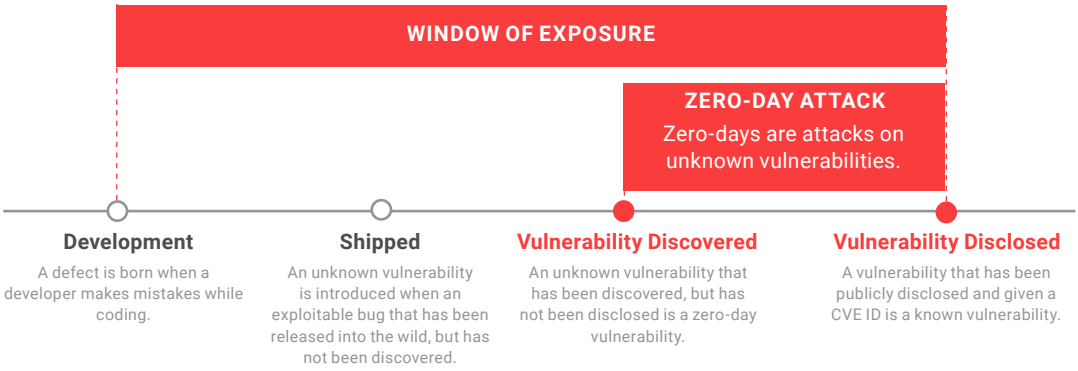
- **Known vulnerability.** Known vulnerabilities have been disclosed to the software vendor and security community. These vulnerabilities usually have an identifier (Common Vulnerabilities and Exposure number) and, in some cases, a patch.

- **Unknown vulnerability.** Unknown vulnerabilities have not been discovered by anyone. They are dormant, waiting to be found and scoped for severity. These vulnerabilities do not have an identifier or a patch. Because these vulnerabilities are unknown, it would require dedicated time and creativity from security researchers to detect and prevent them.

- **Zero-day vulnerability.** Zero-day vulnerabilities have not been disclosed to the vendor and/or community. However, zero-day vulnerabilities have already been discovered and select individuals or organizations are aware of them. These vulnerabilities do not have an identifier or a patch. Similar to unknown vulnerabilities, zero-days are dangerous because they are difficult to detect and prevent. They also enable nefarious actors to function unnoticed for extended periods of time.

> The application attack surface is growing by 111 billion new lines of software code every year, with newly reported zero-day exploits rising from one-per-week in 2015 to one-per day by 2021.

## The Life of a Vulnerability




# Static and Dynamic Testing

There are three categories of techniques for detecting vulnerabilities:

- **Static Analysis Security Testing (SAST).** Static analysis is a form of white-box testing that uncovers bugs, vulnerabilities, and weaknesses by analyzing source code. The analysis is run while the software is in a non-running state. Static analysis tools are able to provide prescriptive remediation advice, down to the line of code. However, they can have high false-positive rates that result in labor- and time-intensive vulnerability triaging and validating efforts. Because they have zero insight into how the application behaves in a running state, not all findings are exploitable.

- **Dynamic Analysis Security Testing (DAST).** Dynamic analysis is a form of black-box testing that uncovers vulnerabilities, configuration errors, and design weaknesses by analyzing software without access to source code. Dynamic analysis tools have low false-positive rates because they test software while it is in a running state. However, their remediation advice can leave much to be desired. Because they have zero insight into the source code, their remediation advice lack resolution, requiring security expertise for interpretation.

- **Grey-box testing.** Some technologies employ a combination of static and dynamic analysis. This technique is also known as grey-box testing. Grey-box solutions conduct dynamic analysis, with visibility into the source code. This type of technology is newer than SAST and DAST. Thus, it may not be as widely-recognized.

Secure software development best practices call for not only the use of both static and dynamic analysis, but also the use of various static and dynamic analysis tools.

Most static and dynamic analysis tools search for known vulnerabilities or test known attack patterns. For a security testing tool to discover unknown or zero-day vulnerabilities and test unknown attack patterns, it must conduct negative testing.

## Positive and Negative Testing

At its core, application testing aims to verify functionality. Verification and validation processes ensure that applications operate as expected. Verification and validation testing is conducted in two different ways:

- **Positive Testing.** Positive testing, or functional testing, is a testing process where an application is sent a valid set of inputs. The purpose of positive testing is to ensure the application behaves as expected. While this type of testing is typically conducted by QA teams, modern DevOps shops may collaborate closely with security or development teams.

- **Negative Testing.** Negative testing, or non-functional testing, is a testing process where an application is sent an invalid set of inputs. The purpose of negative testing is to ensure the application remains stable in unexpected use cases. While this type of testing is typically conducted by security teams, modern DevOps shops may collaborate closely with QA or development teams.

Positive testing is easier to conduct. There is a finite number of features and flows introduced per release. Therefore, there is a finite combination of valid inputs to test. Thus, there is a clear definition of what “done” means.

On the other hand, there is an infinite combination of invalid inputs. It is impossible and improbable to test all the possible ways to misuse and application before it is deployed. Organizations must determine, often arbitrarily, what “good enough” means.

Despite its challenges, negative testing is vital to application security. The world is chaotic, unpredictable, and sometimes malicious. Applications must interoperate with people and technologies, regardless of their unexpected behaviors. Application programs must be resilient to be considered safe, secure, or of quality.

> There is an infinite combination of invalid inputs. It is impossible and improbable to test all the possible way to misuse an application before it is deployed. Organizations must determine, often arbitrarily, what “good enough” means.

# What is Fuzzing?

Fuzz testing, or fuzzing, is a dynamic application security testing technique for negative testing. Fuzzing aims to detect known, unknown, and zero-day vulnerabilities. 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.

# Not All Fuzzers Are Created Equal

A fuzzing engine's method for generating test cases defines which category a fuzzer falls under. The four fuzzing categories are as follows:

- **Random**. Random fuzzing is the act of sending random inputs to an application. There is no systematic method to the generation of these test cases, and they do not resemble a valid input. As a result, most inputs do not penetrate the application, leading to low code coverage. For a random fuzzer to reach the same level of coverage as its more effective counterparts, it requires significantly more time and effort from security experts.

- **Template**. Template fuzzers utilize manually supplied custom inputs and modify them to include anomalies. They are more effective than random fuzzers, because they resemble valid inputs. The probability for these test cases to penetrate an application is higher than a random fuzzer's test cases. However, there are several drawbacks. Template fuzzers randomly include anomalies without an understanding of common error-detection techniques. As a result, their test cases are often blocked. Second, templates have limitations. Not only is the test case quality limited to the valid input template provided, but they also do not fuzz beyond the scope of the template. A template fuzzer can be effective for fuzzing a single function, given that it was provided an expertly crafted template. However, scaling template fuzzing requires expertise.

- **Generational**. Unlike template fuzzers, generational fuzzers understand the inner workings of its input type. Teams of engineers craft hundreds of thousands of test cases, which are then delivered by the fuzzer. These tests are written to resemble a valid input, while evading common error-detection techniques. Some advanced generational fuzzers may possess some level of intelligence and will prioritize their bank of test cases based on the target's feedback, enhancing the likelihood of penetrating applications and triggering anomalous behavior. A generational fuzzer's effectiveness is dependent on the quality of a vendor's test cases. While vendor-

maintained generational fuzzers require significantly less expertise than template fuzzers for testing, they still require expertise for operation and results interpretation. Most commercial fuzzers available today are generational fuzzers.

- **Evolutionary**. Evolutionary fuzzers are intelligent, containing the capability to monitor and leverage the target's behavior to autonomously generate new, custom test cases on-the-fly. These fuzzers have scoring capabilities that measure the effectiveness of the test cases it sends. High-scoring test cases influence the new set of test cases generated and sent. Unlike the aforementioned fuzzer types, evolutionary fuzzers only require a way to monitor its target and sent inputs to them. They are not supplied starter test cases. In recent years, modern fuzzers have grown in sophistication and wield robust automation capabilities, taking on evolutionary-like characteristics. This new pedigree of fuzzers is known as guided fuzzing. Guided fuzzers relieve significant strains introduced from manual test case creation. However, if users desire deep fuzzing analysis, guided fuzzers require expert assistance.

Although the effectiveness of fuzz testing has been well-documented, fuzzing, as a whole, has been criticized for its shallow analysis and inability to penetrate through the peripheral layers of an application. As outlined above, fuzzing requires varying degrees of expert assistance to be truly effective.

Despite criticism, fuzz testing is a proven method and a recommended practice for organizations who build their own software and rely on supplier software for business productivity. Commercial organizations are recognizing its impact. Trends show fuzzing adoption rates are on the rise, particularly with security-fluent software developers, due to its benefits.

> Commercial organizations are recognizing the impact of fuzzing. Trends show fuzzing adoption rates are on the rise, particularly with security-fluent software developers, due to its benefits.

## ForAllSecure Harnesses the Power of Fuzzing

ForAllSecure believes powerful, dynamic negative testing, such as fuzzing, should be accessible to organizations of all fuzzing proficiency. ForAllSecure sought to uncover a way to further the science behind fuzzing to address its technical shortcomings and improve its usability. This research pioneered a new technique we call advanced fuzzing.

# What is Advanced Fuzzing?

Advanced fuzzing unifies the tried-and-true method of guided fuzzing with the ingenuity of symbolic execution, a patented dynamic analysis technology stemming from a decade of research at Carnegie Mellon University.

Guided fuzzers rely on sample inputs, or a corpus, for initial guidance to explore a program. Thereafter, it monitors and leverages its target's behavioral feedback to generate new, customized test cases on-the-fly. These newly generated test cases aim to incrementally test new sections of code, checking the security of each new region it successfully penetrates. The speed of guided fuzzers is undeniable. However, they struggle to break through complex conditional clauses within a program, limiting their testing depth. Without guidance, guided fuzzer randomly bounce around a program, struggling to reach deep into the code.

Symbolic execution offer guided fuzzers unprecedented benefits in these prevalent scenarios. Symbolic execution, although slower in completing a single testing iteration, solves complex branch conditions and generates a corpus, allowing the fuzzer to methodically resume its testing and analyze deeper into the program. The synergistic relationship between guided fuzzing and symbolic execution is referred as Advanced Fuzzing (see Figure 1). With advanced fuzzing technologies, like ForAllSecure Mayhem, organizations are able to employ proven dynamic negative testing without the personnel and expertise its predecessors required. In summary, the results are greater coverage and findings in less time, resources, and cost.

## Advanced Fuzzing

10+ years research in binary analysis

4 patents in automatic exploit generation

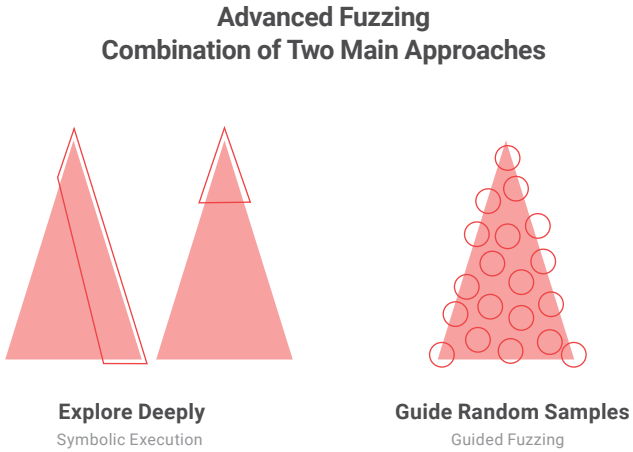
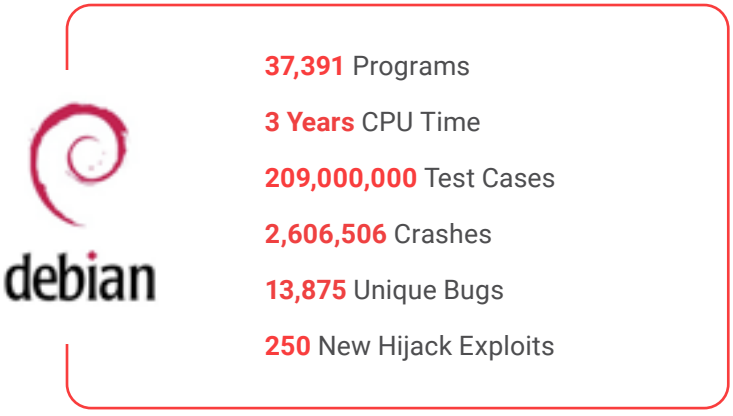




Figure 1: Advanced Fuzzing diagram

In a 2014 demonstration, the Mayhem prototype analyzed over 38,000 programs from scratch, performed over 209 million tests of those programs, of which 2 million resulted in successfully hacks of the program. Those 2.6 million successes were the result of 13,875 previously undiscovered vulnerabilities. Mayhem is an assisted-intelligence solution - meaning the highest resource-related cost for operation is CPU time, not expert time. On Amazon, it cost on average $0.28 for Mayhem to discover and prove each new vulnerability (See figure 2).

## Checking Debian Linux for Vulnerabilities




*[ARCB, ICSE 2014, ACM Distinguised Paper], [ACRSWB,CACM 2014]

Figure 2: In a 2014 demonstration, ForAllSecure checked Debian Linux for vulnerabilities.

# Why Advanced Fuzzing?

Efficiency improvements in dynamic negative testing, led by ForAllSecure's research in advanced fuzzing, has unlocked new advancements and benefits in software security, particularly for software development.

ForAllSecure pilot users integrated Mayhem as a complementary security technology within their software security's defense-in-depth initiatives, running alongside tools such as SAST, software composition analysis, and more. Users -- within defense, aerospace, automotive, and high-tech industries -- cited that Mayhem offers the following value adds to their initiatives:

- **Fuzz testing that uncovers deep defects.** Mayhem acquires intelligence of its applications over time. As Mayhem's knowledge grows, its test cases grow increasingly precise, deepening its analysis and yielding higher code coverage rates. Unlike most application security testing solutions, Mayhem tests uncommon or unknown attack patterns to detect unknown and zero-day vulnerabilities, ensuring the window of exposure never opens. With Mayhem, users consistently and continuously receive quality findings.

- **Actionable results with zero-false positive rates.** Sifting out false-positives is a laborious, cross-organizational effort that slows down productivity and cripples morale. All reports by Mayhem are verified defects. Mayhem's accurate findings keep developers, security, and QA focused, so development continues to move forward.

- **Autonomous testing at machine speed and scale.** Dynamic negative testing, though proven, requires experts to develop test cases and review results, adding unforeseen overhead costs. Dependency on manual intervention limits scale, speed, and effectiveness. Mayhem utilizes target feedback to custom generate test cases on-the-fly -- meaning no manual test case generation. Mayhem shares all of its test cases to make regression testing effortless, fast, and scalable.

- **DevOps flexibility shift-lefts dynamic negative testing.** Mayhem defies inherent limitations of dynamic negative testing by bringing it earlier in the software development lifecycle, helping organizations control remediation costs and prevent time-to-market delays. Mayhem conducts regression and component testing, and fits directly into the DevOps workflow.

- **Manage inherited risk from an unchecked supply chain.** Mayhem does not require source code for analysis, allowing users to verify and validate third-party code from their software supply chain. Mitigate risk inherited by today's intricate and unchecked software supply chain with Mayhem.

Mayhem is a versatile security testing solution fit for a range of industries and business initiatives. Pilot users recount that Mayhem has scaled their efforts to stabilize and secure the software they develop, and has offered transparency into code from their software supply chain.

# Conclusion

The application attack surface is growing by 111 billion new lines of software code every year, with newly reported zero-day exploits rising from one-per-week in 2015 to one-per-day by 2021. As development speeds and deployment frequencies intensify, security must scale in tandum.[4]

Dynamic negative testing techniques, specifically fuzzing, is a proven method for security testing. Despite its recent rapid advancements, fuzzing has been criticized for its shortcomings in usability and reliance on technical expertise, limiting its benefits.

ForAllSecure Mayhem, an advanced fuzzing technology, addresses these challenges to make this proven technique accessible to and usable by all organization. Uniting the tried-and-true methods of guided fuzzing and the ingenuity of symbolic execution, a patented dynamic analysis technology from over a decade of research at Carnegie Mellon University, Mayhem makes advanced security techniques, previously exclusive to the security research community, more accessible and scalable. Mayhem is a powerful solution that offers depth of analysis with a fraction of time, resources, and cost. With Mayhem, organizations, across various industries, are enabled to improve the stability and security of the software they develop and use.

For a demo of the Mayhem solution, sign up at www.forallsecure.com/demo/.

## Securing the world's most critical software.

ForAllSecure was founded on the mission to make the world's software secure. Utilizing patented technology from a decade of research at Carnegie Mellon University, ForAllSecure delivers an advanced fuzz testing solution. Fortune 1000 companies in aerospace, automotive, and high-tech partner with ForAllSecure for scalable, autonomous security testing that keeps pace with increasing development speeds and deployment frequencies. DARPA deemed ForAllSecure the winner in the Cyber Grand Challenge, and MIT Technology Review named ForAllSecure in the 50 Smartest Companies list. Efficiently and effectively secure critical software with ForAllSecure.

For more information, visit www.forallsecure.com
To learn more, contact info@forallsecure.com

1. “Is Poor Software Development the Biggest Cyber Threat?”. CSO from IDG. 2 September 2015. https://www.csoonline.com/article/2978858/application-security/is-poor-software-development-the-biggest-cyber-threat.html.

2. “Focus on the Biggest Security Threts, Not the Most Publicized”. Gartner, 2 November 2, 2017. https://www.gartner.com/smarterwithgartner/focus-on-the-biggest-security-threats-not-the-most-publicized/.

3. “Global Cybersecurity Workforce Shortage to Reach 1.8 Million as Threat Loom Larger and Stakes Rise Higher”. (ISC)². 7 June, 2017. https://www.isc2.org/News-and-Events/Press-Room/Posts/2017/06/07/2017-06-07-Workforce-Shortage.

4. “The Zero Days Report”. Cybersecurity Ventures. 3 January 2017. https://cybersecurityventures.com/zero-day-vulnerabilities-attacks-exploits-report-2017/