Software Assurance Tips

A product of the Software Assurance Tips Team[1]

Jon Hood

Monday 18th November, 2024

1 Fuzzy Coverage

Updated Saturday 23rd May, 2026

One of the questions we get asked frequently is, "How much fuzzing is enough?" This question is usually derived from a DoD mentality of checking the box on cybersecurity activities. With Security Technical Implementation Guides (STIGs), IAVMs, and RMF Control checklists, cybersecurity practitioners can become complacent switching from a compliance checkbox mindset to a true risk management mindset.

Fuzzing is one of the dynamic code analysis tasks permitted in Risk Management Framework (RMF) control SA-11(8). "Fuzz testing strategies are derived from the intended use of applications and the functional and design specifications for the applications."[3, p. 279] While fuzzing is *derived* from functional test cases (such as unit testing or formal qualification testing cases), fuzz testing enters into the security domain by executing varying inputs into functionality test cases. These generated, often mutating inputs are used to identify security-critical bugs and defects in the software. "More generally, fuzzing is used to demonstrate the presence of bugs rather than their absence."[4] Popular sanitizers for fuzzing include memory related issues (CWE-118 and its children such as buffer overflows CWE-120, out-of-bounds access CWE-125 and CWE-787, overflows CWE-190, and more), race conditions (CWE-362), undefined behavior (CWE-758), and control flow integrity (CWE-691). By providing a hook into an application with a fuzzing harness and given an expected input, a fuzzer can iterate over unexpected and untested inputs rapidly to find cybersecurity issues which may lead to compromise.

Notice how fuzzing is intended to prove a negative: a failed test case indicates a high likelihood that the code has an error. This introduces the first misconception of fuzzing. The reverse is not necessitated: **the lack of findings from fuzzing does not indicate the absence of defects.** Using fuzz testing as evidence for the lack of cybersecurity defects in software is not one of the main goals for implementing a fuzzing program. When setting up a Software Assurance program, results from fuzz testing should be given very low assurance when it is used as the input to prove that an application is free from defects.

The idea that fuzz testing is a checkbox to complete directly feeds into the second mistake of fuzzing: **declaring that any amount of fuzzing is enough**. This is most evident when someone asks, "Did you fuzz the application?" The correct question is, "How much of the application are you fuzzing?" Always be fuzzing. And if it has been a while since the test cases have found anything, consider updating the harnesses to execute tests differently.

Finally, most metrics for fuzz testing in the DoD declare a code coverage percentage that is required for the test harnesses. But **code coverage is not a meaningful metric for fuzz test completion.** Consider the example in Listing 1. If this code were to be harnessed and passed an input of `Jon`, then 100% of the code is executed but the buffer overflow condition is not detected. Stopping the fuzzing activity when a percentage of code branches are completed or when `lcov` or `gcov` return that a certain percentage of the code paths have been executed creates a false sense of security.

```c
#include <stdio.h>
#include <string.h>
int main() {
    char buffer[10];
    printf("Enter your name: ");
    fgets(buffer, 100, stdin);
    printf("Hello, %s!\n", buffer);
    return 0;
}
```

Listing 1: Simple Buffer Overflow

Instead, experts should review the harnesses to make sure that the critical areas of code are properly being fuzzed continuously. One way to complete this is to do a criticality analysis and functional traceability report into the code, such as the design traceability analysis[2, p. 114] and source code traceability analysis[2, p. 119]. Leveraging these functionality reviews to define whether

the critical parts of the code have been harnessed allows a DoD program office to say, “Yea, verily, the fuzzing being conducted by this effort satisfies our fuzzing requirements.”

Summary

Remember the misconceptions of fuzzing:

- The lack of findings from fuzzing does not indicate the absence of defects.
- No amount of fuzzing is “enough.”
- Code coverage is not a meaningful metric for fuzz test completion.

The right fuzzing program is the one still running against critically important code.

References

[1] Jon Hood, ed. SwATips. https://www.SwATips.com/.

[2] IEEE Standards Association. “IEEE Standard for System, Software, and Hardware Verification and Validation”. In: IEEE Std 1012-2016 (Revision of IEEE Std 1012-2012/ Incorporates IEEE Std 1012-2016/Cor1 (2017), pp. 1–260. DOI: 10.1109/IEEESTD.2017.8055462.

[3] National Institute of Standards and Technology. Security and Privacy Controls for Information Systems and Or Tech. rep. Special Publication (SP) 800-53 Revision 5. Washington, D.C.: U.S. Department of Commerce, 2020. DOI: 10.6028/NIST.SP.800-53r5. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf.

[4] Wikipedia contributors. Fuzzing — Wikipedia, The Free Encyclopedia. [Online; accessed 18-November-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Fuzzing&oldid=1249540069.