RESEARCH ARTICLE OPEN ACCESS

Smart Fuzzer

Mr. Deepak Moud, Mr. B. L Pal
MTech Scholar Mewar University, Chittorgarh
Associate Professor, Mewar Universty, Chittorgarh

Abstract
Fuzzing is basically bug finding technique.
This is done by providing an application with semi-valid input. The input should in most cases be good enough so applications will assume it as valid input, but at the same time be broken enough so that parsing done on this input will fail. Such failing can lead to unexpected results such as crashes, information leaks, delays, etc. It also requires understanding possible bugs that can be found in code.
Smart fuzzer creates fuzzed files to be used for fuzz testing. It finds bug automatically. We will develop a web application user interface which has the smart fuzzer developed as the back end.
Keywords: smart fuzzer, asp.net with c#, platform: Visual Studio 2010

I. INTRODUCTION

You can never test quality or security into an application. If the application is written in an insecure manner with poor coding practices or has a large attack surface, no amount of testing will make it secure.
Although known to only a few developers, non-security experts can conduct effective security testing, most notably by fuzz testing. Fuzz testing includes penetration testing, run-time verification, re-reviewing threat models and re-evaluating the attack surface.
The goals of testing are:
Verify that the application block is able to meet all requirements in accordance with the functional specifications document.
Make sure that the application block has consistent and expected output for all usage scenarios for both valid and invalid inputs.
For example, make sure the error messages are meaningful and help the user in diagnosing the actual problem.
White box testing assumes that the tester can take a look at the code for the application block and create test cases that look for any potential failure scenarios. During white box testing, you analyze the code of the application block and prepare test cases for testing the functionality to ensure that the class is behaving in accordance with the specifications and testing for robustness. In software testing, fuzzing is the practice of forcing applications to consume corrupted data and observing the results. Poorly coded applications will "fall over" when they get fuzzed data, typically trying to process the data without checking to see if it's correct and complete. Well-coded applications will not crash, nor will they become unstable because they reject improper data. The practice of fuzz testing can quickly uncover dramatic coding errors and is relatively simple to do, which explains its increasing popularity in the test community and its use in the Microsoft® SDL. Unfortunately, the same things that make it popular with testers, make it popular with hackers whose goal is to crash applications and look for sensitive data. This popularity with hackers now makes the practice in the software development lab a requirement, not just a cool new way to test—applications have to be prepared for fuzzing.

II. METHODOLOGY

Fuzz testing is accomplished by attacking all of an application's data interfaces, typically the file system, network, libraries, registry and GUI. In its simplest form, fuzzing is accomplished using randomized data—a jumbled raw network stream for example. This type of attack will cause poorly coded applications to crash, but are non-deterministic in nature and just tell the software team that there is a problem and points to its general locus. More sophisticated fuzzing includes parametric data designed to attack specific parts of an application—a corrupted word processing file header for example. This type of testing provides deterministic results that allow software teams to zero in on specific areas of concern and attack in a very focused and reproducible manner. Both are good, both provide valuable data on the fragility and security of an application.

Specialized tooling is not necessary to effectively fuzz test an application. It is, a simple thing to make a spreadsheet tool try to consume an image file and see what happens. It is also simple to break out a hex editor and change values in a file

header. However, it is a tedious process and automation allows a greater range and depth of fuzz testing. A dramatic example of where random fuzz testing needs automation is in the Microsoft® SDL where applications are required to consume 100,000 fuzzed files. If an application fails to open just one, be it the first or the 99,999th, they have to be repaired and retested using the entire file collection. Creating 100,000 files for testing is a big job and automated tooling can help dramatically. A decent automated tool will generate a collection of randomly fuzzed files with the proper extensions for the application to consume; a better tool will use a sample collection of files as a template and do very specific fuzzing to specific parts of files; and a great tool will virtualize the file stream and obviate the need for physical files in the first place. In any case, the benefit of automation in a large test file collection is as clear as the benefits of fuzz testing itself.

Fuzzer

Fuzzer is a testing tool which is meant for checking the robustness of the the application. And the fuzzer does it by providing the semi-valid inputs to the application. Semi-valid inputs are kind of random inputs which are fed to the application under test and then we see how the application responds to it. If the application respond abnormally or terminate the execution or anything which we cannot call as normal response then the bug in the application is exposed and the application cannot be called as secure and robust.

III. WORKING

Dumb Fuzzer

Dumb Fuzzer changes the data at random. Dumb fuzzing is a shotgun approach: you take a valid file and randomly corrupt it.

Smart Fuzzer

Smart fuzzer knows the data structure of the file format and is used to change specific values within the file.

Ways to fuzz a file

- You can smart fuzz or dumb fuzz a file in many ways, including these:
- Making the file smaller than normal
- Filling the entire file with random data
- Filling portions of the file with random data
- Searching for null-terminated strings (in ASCII and Unicode) and setting the trailing null to non-null
- Setting numeric data types to negative values
- Exchanging adjacent bytes
- Setting numeric data types to zero
- Toggling, setting, or clearing high bits (0x80, 0x8000, and so on)
- Doing an exclusive OR (XOR) operation on all bits in a byte, one bit at a time
- Looking back at the PNG format, you could be very specific and smart fuzz a file by using the following techniques:
- Set the chunk length to a bogus value.
- Create random chunk names. (They are case-sensitive, and the case has specific meaning.)
- Build a file with no IHDR chunk.
- Build a file with more than one IHDR chunk.
- Set the width, height or color depth to invalid values (0, negative values, and so on).
- Set invalid compression, filter or interlace modes.
- Set an invalid color type.

IV. SCOPE OF WORK AND BOUNDARY CONDITIONS

Smart fuzzer can be used for quality assurance of applications developed in industry or by individuals. It is used during the testing phase of applications. The fuzzed files generated by fuzzer are injected into the applications under test to analyze its response.

Smart fuzzer would be a useful tool for the team involved in the testing of an application. It automatically generates efficient test cases thereby eliminating the overhead of manual testing.

The Smart Fuzzer for PNG File Format works only for PNG files and generates test cases that can be used to test only the applications used to view images.

The fuzzing options provided in the fuzzer include the following chunks:

IHDR Chunk
sRGB Chunk
gAMA Chunk
pHYs Chunk
IDAT Chunk
tEXt Chunk
sBIT Chunk
bKGD Chunk

The above mentioned chunks are selected on the basis of their high frequency of occurrence in the PNG images.

These chunks therefore, produce effective results when PNG files are fuzzed.

V. CONCLUSION

Smart fuzzer for PNG file format is a fuzzing tool which is used to produce fuzzed files.

Smart fuzzer would play an important role in software industry. Smart fuzzer would be used during the testing phase of the software development. The automatic generation of test cases would save both time and effort thus making it a good choice for the test case generation.

Smart fuzzer is a scalable tool which can be further extended for other file formats as well. It would add to its functionality and make it a versatile application. It would help in development of high quality applications and ensure that the applications are robust enough to handle exceptions.

REFERENCES

[1] Michael Sutton, Adam Greene, Pedram Amini (2007).Fuzzing: Brute Force Vulnerability Discovery. Addison-Wesley. ISBN 0-321-44611-9.
[2] John Neystadt (2008-02). "Automated Penetration Testing with White-Box Fuzzing". Microsoft. Retrieved 2009-05-14.
[3] Barton Miller (2008). "Preface". In Ari Takanen, Jared DeMott and Charlie Miller, Fuzzing for Software Security Testing and Quality Assurance, ISBN 978-1-59693-214-2
[4] "Fuzz Testing of Application Reliability". University of Wisconsin-Madison. Retrieved 2009-05-14.
[5] "Kaksonen, Rauli. (2001) A Functional Method for Assessing Protocol Implementation Security (Licentiate thesis). Espoo. Technical Research Centre of Finland, VTT Publications 447. 128 p. + app. 15 p. ISBN 951-38-5873-1 (soft back ed.) ISBN 951-38-5874-X (on-line ed.)." (PDF). Retrieved 2010-05-28.
[6] Stefan Wappler, Joachim Wegener: Evolutionary unit testing of object-oriented software using strongly-typed genetic programming. GECCO 2006: 1925-1932
[7] Goldberg, David E. Genetic Algorithms in Search, Optimization and Machine Learning Addison-Wesley Pub. Co. 1989. ISBN: 0201157675
[8] http://www.appliedsec.com/resources.html
[9] http://www.goldenftpserver.com/
[10] Patrice Godefroid, Adam Kiezun, Michael Y. Levin."Grammar-based Whitebox uzzing". Microsoft Research.
[11] "Software Fault Injection: Inoculating Programs Against Errors by Jeffrey M. Voas and Gary McGraw". John Wiley & Sons. January 28, 1998.
[12] A. Takanen, J. De Mott, C. Miller, Fuzzing for Software Security Testing and Quality Assurance, 2008, ISBN 978-1-59693-214-2
[13] Wegener, Sthamer, & Baresel. "Application Fields for Evolutionary Testing", Euro STAR, 2001.
[14] H. Pohl, Cost-Effective Identification of Zero-Day Vulnerabilities with the Aid of Threat Modeling and Fuzzing
[15] J. DeMott, "Benchmarking Grey-box Robustness Testing Tools with an Analy