STUDY PAPER ON

FUZZ TESTING

---TSA Division, TEC
# Table of Contents

1. Introduction ................................................................................................................. 3

2. Effectiveness ............................................................................................................... 3

3. Fuzzing Process ........................................................................................................... 4

4. Fuzz Testing Approaches ............................................................................................ 5

5. Types of Fuzzers .......................................................................................................... 6

6. Open Source Fuzzing tools .......................................................................................... 7

7. Commercial Fuzzing tools ........................................................................................... 10

8. Future Trends in Fuzzing ............................................................................................. 12

9. Standardisation Activities ........................................................................................... 12

10. Telecom Network vulnerabilities discovered by Fuzz testing .................................. 12

11. Conclusion and Recommendation ............................................................................. 13

GLOSSARY ......................................................................................................................... 14

REFERENCES .................................................................................................................... 14
Fuzz Testing

1. Introduction

Fuzz testing or fuzzing is a software testing technique used to discover security vulnerabilities in network protocols, applications, file formats etc. Although the idea behind this technique is conceptually very simple, it is a well-known and widely established methodology employed in COTS software vulnerability discovery process. Fuzz testing was originally developed by Barton Miller at the University of Wisconsin in 1989. Since then the technique has evolved a lot and it is used internally by many companies to find bugs in their software. All the big companies such as Microsoft, Cisco, IBM, etc. have been using this technique to improve software quality.

The real world is full of unexpected conditions and badly formed inputs. Softwares used in the telecom protocol stacks implementation and other associated implementations must be able to deal with the poorly formed inputs, unexpected actions and misuse of the software. Fuzzing is the process of sending intentionally malformed inputs to a piece of software to see whether it crashes or show any other unexpected behavior like CPU overload, memory leaks etc. While using fuzzers to improve security, the end objective is not just finding vulnerabilities, but fixing them as well. Finding and fixing vulnerabilities in the products before their deployment in the network certainly saves money, protects customers and improves reputation.

2. Effectiveness

"Fuzzers are a good way to find bugs, but not so good about making assurances that no bugs exist."

---- Michael Eddington, author of the widely used open source fuzzer Peach

Fuzz testing works best for vulnerabilities that can cause a program to crash, such as buffer overflow, cross-site scripting, denial of service attacks, format bugs and SQL injection. Fuzzing is less effective for dealing with security threats that do not cause program crashes, such as spyware, viruses, worms, Trojans and keyloggers. Fuzz testing can undoubtedly reveal defects that are overlooked when software is written and debugged. Nevertheless, it usually finds only the most serious faults. This testing alone cannot provide a complete picture of the overall security of the target. Fuzzers are most effective when used in conjunction with other proven security testing methods.
3. Fuzzing Process

The basic phases involved in the fuzzing process are as follows:

3.1 Identification of target
Before selecting the fuzzing tool it is required to identify the target, whether it is a protocol stack, application, file format or a web browser.

3.2 Identification of inputs
Most of the exploitations are caused due to the fact that target applications accept and process the user input without sanitizing and validating the input data. Hence, identification of inputs is a key to success of the fuzzing process.

3.3 Generation of fuzzed data
After identification of inputs, fuzzed data must be generated. This data may be some predetermined values, mutation of existing samples or some dynamically generated data depending upon the target and inputs identified.

3.4 Execution of fuzzed data
This step goes hand in hand with the previous one. It involves sending a fuzzed data packet towards the target after launching a target process.

3.5 Monitoring for exceptions
Exception or fault monitoring process is one of the most vital steps of fuzzing. Transmitting thousands of fuzzed data packets towards a target and ultimately causing that target to crash will be a futile exercise, if we are not able to pinpoint the packet responsible for crash.

3.6 Reporting
Finally, the findings of the fuzzing process are compiled in the form of a report, which later on helps in mitigation efforts.

**A Typical Example:**

![Diagram](image)

### 4. Fuzz Testing Approaches

One of the most important aspects of fuzzing is the generation of fuzzed inputs. These inputs or test cases are normally generated by using one of the following two approaches.

#### 4.1 Generation based fuzzing

The starting point for this approach is a specification or RFC, which describes the file format or network protocol. Test cases are generated based on the input data format specified in these documents. Each generated test case should differ from the valid input to such an extent that a problem is caused in the target application, but should not be too invalid to be discarded by the target application. Generation based fuzzing requires a significant amount
of upfront work to study the test specification and generate the test case from scratch. But, the extra knowledge gained by understanding the format results in higher quality test cases.

4.2 Mutation based fuzzing

In this approach valid data samples e.g. files, traffic captures etc. is collected and then modified. Examples of mutations include bit flipping, replacing small string with longer strings etc. The advantage of this approach is little or no knowledge of target is required. All that is needed is some good data samples. Since, the protocol stacks and applications contain large sections of code that may not execute with mutated inputs, mutation based fuzzers do not often perform an effective fuzzing.

**Which approach is more suitable?**

It depends on requirements, availability of time and target. For example,

i. Some clear text protocols such as FTP, SMTP, etc. lend themselves well to mutation fuzzing

ii. Some protocols such as SSH, IKE, etc will require generation fuzzing because of the complexity of the protocol.

Hence, it is always advisable to use both the approaches, if possible

5. Types of Fuzzers

Fuzzers are classified into the following categories based on the targets.

5.1 Protocol Fuzzer

Flaws in the implementations of network protocols are some of the most serious security problems. Such flaws could allow a malicious user to attack vulnerable systems remotely over the Internet. A protocol fuzzer is used to discover implementation flaws in a protocol stack by sending the unusual inputs towards the target in hopes of producing unexpected behavior.

5.2 File format Fuzzer

A large number of applications deal with file input and output and are susceptible to vulnerabilities that occur while parsing maliciously crafted files. File format fuzzers discover these vulnerabilities. It dynamically creates different malformed files and sends them towards the target application and monitors the target for an unexpected behaviour.

5.3 Web application Fuzzer

Web application fuzzing can discover vulnerabilities in the web application itself or any of the underlying components like database server with whom it might integrate with. For web
application fuzzing the delay caused in transport of inputs from fuzzer to the target is a challenge. Hence, if possible rather than running the target application on a remote server it should be hosted locally so that the packets don’t have to traverse a network. Another option is to run the web application on a VM, while the fuzzer can be run locally. These fuzzers are used to discover vulnerabilities such as SQL injection, cross site scripting, command injection vulnerability etc.

5.4 Web browser Fuzzer

Vulnerabilities in web browsers have become a major area of concern as they are making their users victims of several attacks such as phishing attack, identity theft, creation of large botnets etc. Time and again, vulnerabilities have been discovered in almost all popular web browsers like internet explorer, Mozilla firefox, Google chrome etc. Web browser fuzzers are capable to discover these kinds of vulnerabilities, which occur while parsing of malformed HTML tags, javascript, image files like JPG, GIF and PNG, and ActiveX controls etc.

6. Open Source Fuzzing tools

Open source Fuzzing tools typically fall into one of three categories: Fuzzing frameworks, Special purpose fuzzers and General purpose fuzzers.

Fuzzing frameworks are good if one is looking to write or develop a new fuzzer or need to fuzz a custom or proprietary protocol. They basically provide quick, flexible, reusable and homogeneous development environment for fuzzer developers.

Special purpose fuzzers are fuzzers that were written for a specific protocol or application. These fuzzers have a very limited scope and were usually developed to find few loopholes in a particular protocol or application and after that were left unmaintained.

General purpose fuzzers are generic fuzzers designed for minimizing setup time during fuzzing sessions and are especially useful for fast testing of proprietary and undocumented protocols.

6.1 Open Source Fuzzing frameworks

Some popular open source fuzzing frameworks are listed below.

i. Antiparser --- http://antiparser.sourceforge.net/

Written in Python, antiparser is an API designed to assist in creation of random data for construction of fuzzers. This framework can be used to develop simple fuzzers that will run
across multiple platforms, but inadequate for handling complex tasks. This framework has not been updated since August 2005.

ii. Autodafe --- http://autodafe.sourceforge.net/

Autodafe supports fuzzing of both network protocols and file formats. It is designed to run on unix platforms and lacks Microsoft windows support. It takes an xml based packet dump as an input to generate fuzzing test cases and has discovered mainly buffer over flow vulnerabilities. This tool has not been updated since August 2006.

iii. Peach Fuzzer --- http://peachfuzz.sourceforge.net/

Peach is a cross-platform fuzzing framework originally written in Python and released in 2004. It uses a combination of mutation and generation based fuzzing. Peach Fuzzer supports fuzz testing of any file format, network protocol, application protocol, Android device, or embedded hardware. It can be extended to enable fuzzing of proprietary systems and interfaces. Its commercial version is also available. It is under active development and the latest update was released in November 2014. Although, Peach is highly advanced in theory it is not so well documented.


Dfuz is a remote protocol fuzzer, which is capable of sending random data combined with the positive inputs. It also sends rule files to the target along with the inputs, which is parsed by the target to know how the input data is to be used. Dfuz can even be used by the nonprogrammers to fuzz protocols. But, there is a very limited scope for code reuse, as any duplication or modification of the code is not permitted without permission of the author. This framework has been used to uncover a variety of vulnerabilities affecting vendors such as Microsoft, Ipswitch and RealNetworks.

v. SULLEY --- http://fuzzing.org/sulley/

Sulley is a python fuzzing framework, which exceeds the capabilities of almost all previously published technologies in the public domain. Mostly, fuzzers are focused on data generation. Sulley not only has impressive data generation, but monitors the health of the target as well as is capable of reverting the target to a good state. It detects, tracks and categorizes detected faults. Sulley can pin point the unique sequence of test cases, which triggers faults. Sulley will be taken as a separate study paper during next year and may be used as an open source fuzzing framework in the upcoming security test lab.
6.2 Open Source Special-Purpose Fuzzing tools

These tools are generally written for a specific protocol or application. Some of the commonly available open source special-purpose fuzzing tools are:

i. SPIKE Proxy --- [www.immunitysec.com/resources-freesoftware.shtml](http://www.immunitysec.com/resources-freesoftware.shtml) for web applications.

ii. Mangle --- [www.lcamtuf.coredump.cx/](http://www.lcamtuf.coredump.cx/) for HTML file fuzzing


6.3 Open Source General-Purpose Fuzzing tools

These tools are good beginning to fuzzing with minimal effort and to get ideas on how fuzzing should be done and how it works. Some of the most common open source general purpose fuzzing tools are mentioned below.

i. SPIKE --- [www.immunitysec.com/resources-freesoftware.shtml](http://www.immunitysec.com/resources-freesoftware.shtml)

It is the most preferred tool to analyze a new protocol for buffer overflows or similar weaknesses. It is available for Linux platform only and requires a strong knowledge of C to use. One of the first block based open source fuzzing tool and is known to have found vulnerabilities in Microsoft Windows’ RPC framework and other products. This tool has not been updated since August 2005.


It is designed to be a good general purpose tool for developers and vendors who want to test how well their products can withstand malicious inputs. It has been used to find previously unknown security vulnerabilities in handling of many kinds of file formats with very different structure, like bmp, png, gif, jpeg, svg, xml, ogg, avi, html, gz, bzip2, tiff, pdf and zip. This tool is available for platforms such as GNU/Linux, OpenBSD (and probably other BSDs). Mac OS X and Windows (experimental).

Microsoft SDL MiniFuzz File Fuzzer is designed to help detect code flaws that may expose security vulnerabilities in file-handling code. This tool creates multiple random variations of file content and feeds it to the application to exercise the code in an attempt to expose unexpected and potentially insecure application behaviors. This tool is available for windows platform only.

iv. Flayer --- https://code.google.com/p/flayer/

Google's security team has released this fuzz testing tool, which was used internally to find multiple vulnerabilities in Internet-critical software products. The fuzzer, called Flayer, is an analysis and flow alteration tool that has been used to find errors in real software. In the past years, results from Flayer has led to the discovery of security holes in several open-source products, including OpenSSH, OpenSSL, LibTIFF and libPNG.

7. Commercial Fuzzing tools

Entrance of commercial players in a technology space strongly indicates that a certain level of maturity has already been achieved. This trend has already been witnessed with fuzzing technology. A large number of software developers such as Microsoft, Google etc. have adopted fuzzing as a means of identifying security vulnerabilities earlier in the SDLC. This has resulted into the emergence of companies to fill the need of robust, user friendly fuzzers. Some of the commercial solutions are listed below.

7.1 BeSTORM

BeStorm by Beyond Security is a Fuzzing framework, a network protocol fuzzer, a file fuzzer and DLL / ActiveX fuzzer. It runs on Windows, UNIX and Linux platforms. It comes with a predefined set of protocol modules each containing a generation based fuzzer as per the full description of protocol in the relevant RFCs. It also comes with a module building tool for developing mutation based fuzzer for proprietary protocols. It has an in-built monitor that can attach itself to the target and monitor it for exceptions, memory leaks, buffer overflows etc. This allows to pinpoint the problems which cause exceptional behavior in the targets like crash, memory leaks etc.

7.2 Defensics

Defensics from CODENOMICON is the most widely known software Fuzzing solutions. It is a commercial version of PROTOS test suite developed by University of Oulu, Finland. It is not a single product, but separate testing tools available for a wide range of network
protocols. Testing suites are also available for file fuzzing. The negative test cases are sent towards the target in an attempt to see if the target survives the attacks and continue to provide the useful service. Target monitoring module is not there, but it utilizes known good test cases to poll the health of the target during testing. The testing modules cannot be extended for creating test cases for proprietary protocols.

7.3 IXIA Breaking Point Systems

It is a custom hardware solution capable of breaking into the target with known attacks, yet unknown attacks and variation of attacks that try to bypass security systems. It can be best used to perform a benchmark between security devices (firewalls, IDSs etc.). This appliance does the testing by creating complicated traffic scenarios or by capturing the network traffic and recreating it, altering it or amplifying it. It can create fuzzed Ethernet, IP, TCP or UDP packets using a library of prebuilt test cases that are updated periodically.

7.4 GLEG ProtoVer Professional

This fuzzer has been developed by a Russian company GLEG. The protocols supported by the fuzzer include LDAP, IMAP, SSL and NFS. The test suite included various interfaces including command line, GUI and web based interface. ProtoVer test suite is written in Python and provided with full source code. The source code allows users to write their own tests using ProtoVer's testing engine or change the existing tests to satisfy their needs.

7.5 SPIRENT Studio Security (earlier Mu Security)

It is a hardware based solution that is capable of detecting known and unknown vulnerabilities in the target systems by methodically attacking it. It supports fuzzing throughout every message exchange and transition that a protocol supports in a real operating environment. It can function as a protocol client or as both the client and server (or equivalent) in order to test pass-through devices such as UTM devices, Ethernet switches, Routers, proxies, firewalls, and entire network architectures. If the target device crashes during a test, this fuzzer can automatically detect the failure and take user defined steps (which may include custom scripts or power cycling the DUT) to re-initialize the device so that the testing can continue. It can also monitor what is going on inside the device under test (DUT), and correlate cause and effect between the specific negative test case(s) and the isolated fault(s). It is capable of finding and isolating hard faults (e.g., crashes) as well as soft faults such as a CPU utilization spike.
or memory leak, which can have disastrous consequences on mission critical network infrastructure devices or revenue generating services.

8. **Challenges and Future Trends in Fuzzing**

Although there are many open source and commercial tools available for fuzz testing of targets such as network protocols, file formats, web browsers, web applications etc., many problems are expected to arise during the testing. For example, some mutation-based fuzzers may run indefinitely, endlessly modifying data and supplying it to the target. Assuming the target never crashes, how do we know when to turn off this type of fuzzer.

Another problem, which is associated with generation-based fuzzer with finite set of fuzzed test case is even if the target does not crash after running all the test cases, what to do next? Should we conclude that the target is secure? Is there a measurement technique available for depth of fuzzing? Is there a way to improve the fuzzing test cases based on the results of previous test runs. The answer to all these questions can be found in a new evolving technology known as white box fuzzing.

For white box fuzzing source code is required. It is basically code coverage along with fuzzing, which is used to measure the effectiveness of fuzzing. It can be used to identify the portions of the code, which was not covered during the fuzzing. Additional test cases can be generated to increase the effectiveness of fuzzing. Microsoft is already using an internally developed white box fuzzer “SAGE” for security testing of their software products.

9. **Standardisation Activities**

3GPP has designed a security evaluation approach under the name SECAM (Security assurance methodology), which has been documented in technical report TR33.805 release 12. Clause 5.2.4.4 of this document talks about basic vulnerability testing (BVT), which consist of requirements for running automated Free and Open Source Software (FOSS) and Commercial off-the-shelf (COTS) security testing tools against the external interfaces of a Network Product. This activity covers at least three aspects: Port Scanning, Vulnerability Scanning by the use of Vulnerability scanners and robustness/fuzz testing.

10. **Telecom Network vulnerabilities discovered by Fuzz testing**
In recent past, many software vulnerabilities in the telecom network entities have been discovered by fuzz testing. Some of them are mentioned below.

i. In 2013, MME and HS-GW of a mobile network operator in Japan crashed, which affected more than 16M subscribers. Fuzz testing of MME and HS-GW software revealed bugs in the software, which were found vulnerable to DOS attacks.

ii. In 2013 again another Asia Pacific MNO had a trouble with its HSS disclosing subscribers key material as a result of NAS injection attack from UE over the Air (OTA). Fuzz testing of mobile (LTE) packet core network revealed lack of sanitization on NAS stack on MME exposing HSS to OTA attacks

iii. In 2010, HLR of an European MNO crashed due to malformed SS7 traffic resulting into 12 hour downtime of the complete network. On fuzz testing of the HLR SS7 stack software, DOS bugs were found.

iv. Most recently in April 2014, Heartbleed bug (CVE-2014-0160) in the OpenSSL’s implementation of the TLS/DTLS (transport layer security protocols) was discovered again with the help of fuzz testing.

11. Conclusion and Recommendation

Fuzz testing is simple and offers a high benefit-to-cost ratio. Fuzz testing can often reveal defects that are overlooked when software is written and debugged. However, fuzz testing alone cannot provide a complete picture of the overall security of a target. It has to be used in conjunction with other security testing techniques such as static analysis, binary analysis etc.

There are many open source and commercial tools available for fuzzing, each having its own strengths and weaknesses. No single approach is full-proof, which means by using multiple approaches for fuzzing more vulnerabilities can be discovered. Hence, a security test lab under DoT must use fuzz testing as one of the security testing approaches to discover vulnerabilities in the telecom network entities. It is also recommended that fuzz testing should be done using more than one commercial as well as open source tools, as each tool has its own strengths and weaknesses.
GLOSSARY

COTS  Commercial off-the-shelf
CVE  Common Vulnerabilities and Exposures
DOS  Denial Of Service
FTP  File Transfer Protocol
GUI  Graphical User Interface
IKE  Internet Key Exchange
MNO  Mobile Network Operator
OTA  Over The Air
SMTP  Simple Mail Transfer Protocol
SQL  Structured Query Language
SSH  Secure Shell
SSL  Secure Socket Layer
TLS  Transport Layer Security
UTM  Unified Threat Management
VM  Virtual Machine

REFERENCES

i. Book on “Fuzzing - Brute force Vulnerability Discovery”

ii. E-book on “Open Source Fuzzing tools” from SYNGRESS

iii. White paper on “Fuzzing : the Past, the Present and the Future” by Ari Takanen, Codenomicon Ltd., Finland

iv. White paper on “ Automated Testing With Commercial Fuzzing Tools” by Prof. Dr. Hartmut Pohl and Daniel Baier, Department of Computer Sciences, Bonn-Rhein-Sieg University of Applied Sciences


vii. Various video tutorials available on internet

viii. Inputs collected from vendors while preparing estimate and technical requirements of security test lab at TEC