Dumb Fuzzing in Practice

Martin Royal
Peter Pokorny
Cameron University

IT 4444 - Capstone
Dr. Diaz-Gomez
April 27, 2012
Dumb Fuzzing in Practice

Peter Pokorny, Martin Royal
Computing & Technology Department, Cameron University, Lawton, OK, USA

Abstract - A form of software and protocol testing commonly known as fuzzing has contributed to discovering software flaws and exploits and since its inception many new practices and ideas have risen from research and hacking communities alike. This project will be investigating the concept and techniques of “dumb” fuzzing. We will be investigating the concept and theory behind fuzzing, utilizing the dumb fuzzing technique against the File Transfer Protocol to analyze its effectiveness, and the benefits and drawbacks to the technique.

Keywords: Computer security, dumb fuzzing, fuzzing, software testing

1. Introduction

In today’s world of quality control and software security testing, major companies employ testing on their software. However this process can be slow and limited in results. The art of fuzzing is a solution to this issue. Fuzzing is a software testing technique that is used to discover errors and security threats within software by sending a massive amount of random data to the application in an attempt to discover a crash or vulnerability.

When looking at what fuzzing tools are available, one must decide between two types of methods to use: dumb fuzzing and smart fuzzing. Each of these methods has its place in penetration testing, but must be used accordingly depending on factors such as the type of software to be tested or the amount of time available for testing [4], [6].

In this paper we will review dumb fuzzing. In order to analyze the technique we plan on using it against the FTP protocol by fuzzing various FTP server software. After the testing, we will look at the benefits and drawbacks of using dumb fuzzing in testing for software vulnerabilities.

2. Definitions

The following definitions are used in this paper:

- **Dumb Fuzzing** – A fuzzing method that systematically pushes data to the program without waiting for proper responses. This method is closely tied with denial of service attacks.
• **FTP** – File Transfer Protocol. A protocol that allows files to be sent from host to host over a network.

• **Fuzzer** – A program or utility that is used to test specified parameters within another program.

• **Penetration** – Any incident that involves a breach of security to a computer system, network, or software.

• **Raw commands** – Basic commands that allow interaction within a protocol.

• **Smart Fuzzing** – A fuzzing method that involves the fuzzer pushing data to the program in a logical way, usually by waiting for responses and possibly altering the stack.

3. **Fuzzing Rationale**

In quality control and assurance, the benefits of fuzzing can help produce more stable builds of the software for the client, as well as secure the application against malicious intent. Developers may employ different concepts and methods of fuzzing to simulate different conditions the application may become subject to, both malicious and unintentional [3], [12]. The overall goal of the testing is to find unintended actions within the application and find the possible impact that it would have upon the application as a whole before end-users cause the actions. Those interested in exploiting an application may employ similar methods of fuzzing to cause a program to cause an unintended action, but will monitor where the action takes place and the conditions that cause it in order to further develop a successful exploit.

Fuzzing has been used by closed source, open source, and researchers as a means of improving the quality and performance of software [1], [8], [9]. Microsoft used fuzzing that enlisted the help of thousands of machines in a grid computing topology, commonly known as the “@Home Network.” In their tests, Microsoft was able to find about 1,800 bugs within the Microsoft Office 2010 code [8]. With these vulnerabilities found, Microsoft was able to patch both current and past vulnerabilities based on code that was exploited. This shows that fuzzing is now an acceptable and useful method of software testing utilized in an enterprise setting.

4. **FTP History**

As our test for fuzzing, we chose the FTP protocol as our target. Fuzzing the FTP protocol uses a dumb fuzzing technique called session data fuzzing. The fuzzer takes known commands and alters parameters to send bad or excessive data to find possible exploits [5], [11]. Sending a large amount of random data can possibly result in buffer overflows and program crashes, which can be analyzed to find precise memory locations to create attacks against the FTP server.

To get an idea of how the possibilities of FTP exploitation, we gathered data from exploit-db.net, an archival site which stores publically known exploits for numerous software products across many platforms [10]. We searched for the term “FTP” and manually filtered out server
exploits, rather than client exploits. We then examined the proof of concept codes that were supplied with the exploit details in order to confirm which commands were exploited to gain an understanding of the most common attack vectors. We also chose only exploits that allowed for buffer overflows to identify the worst case scenarios for a server operator (or best case scenario for a hacker).

![Most Vulnerable FTP Commands](image)

**Figure 4.1**

In Figure 4.1, we see the most common commands that are exploited. In this set, the most commonly exploited command was the LIST command. Normally, LIST has only a few flags that are called, but in these cases, the input is not always sanitized before passing it to the server, allowing for arbitrary input. This happened in ten instances, including FTP software such as AbsoluteFTP, Notepad++NppFTP, FreeFloat FTP, Microsoft IIS FTP, Easy FTP, FileCopa FTP, SlimFTPd, Odin Secure, Victory FTP, and FileZilla.

The USER command was the next most exploited command, with 10 separate instances. The vulnerable servers would not sanitize or check the USER input upon successful connection, and would allow arbitrary input. This has happened with versions of FreeFloat FTP, KnFTP, SolarFTP, ActFax Server FTP, GoldenFTP, 3Com 3CDaemon, War-FTPd, FreeFTPd, and KarjaSoft Sami.

The next most exploited command was the PASS command, after valid USER input. None of the servers listed above had the PASS command overflow vulnerability. KnFTP, GoldenFTP, Easy File Sharing FTP, Oracle 9i XDB, Vermillion FTP, and Titan FTP were the vulnerable servers in this group.
The category “other” was used for crashes that dealt with exploits outside of FTP commands, such as sending of excessive data without initiating a connection, sending excessive data without a command once logged in, accessing backdoors in software, and more.

5. Test setup

For our testing we set up two virtual machines through Oracle VirtualBox, and used the Microsoft XP operating system with Service Pack 2. Each virtual machine was bridged through the network interface, so each got its own unique IP address. The network being was locally hosted through a Cisco E1200 router. Each of the operating systems received no updates and any additional software other than Firefox and the FTP server applications. The Windows Firewall was set to allow port traffic for the FTP programs. All other software was left under default conditions and settings.

Our fuzzer choice was the Infigo FTP Stress Fuzzer. Infigo is a FTP fuzzing tool for Windows that allows defining of raw commands to test, choice of pre-generated or custom strings to test with the commands, time-out options, and more [7], [11]. Infigo was installed on both machines. One Windows machine would host an FTP server as the other would fuzz the FTP host with Infigo. We would first test the most common raw commands (similar to the commands in Figure 4.1) and then test nearly all raw commands possible.

To perform adequate testing we used multiple different FTP server software. We chose FTP servers that were freeware or open source. The full list of chosen software is as follows: CoreFTP, WarFTP, GoFTP, Apache FTP Server, ArgoSoft FTP Server, Titan FTP, WingFTP, SmartFTP, GoldenFTP, and FileZilla. All installers were up-to-date, and no updating or patching were done to the programs. We created an admin account user named “test”, with the password “test” to allow the fuzzer to login and send the required commands.

6. Results

Each test took from 2-3 hours to complete, and some commands had to be done in groups of tests due to an issue with Infigo as a fuzzer. Nearly all of the FTP software was able to handle the fuzzing from Infigo. All fuzzing combinations of commands and various strings would not alter the servers at all. However one tested server was affected; WarFTP. We were able to get a successful crash on the service by sending the CWD command with the string “%s” appended 15-35 times after the end. At this the WarFTP service crashed on the target machine and would no longer take input from the fuzzer. Previous versions of WarFTP were vulnerable to Denial of Service attacks from other commands, and it appears that the newest version still has these bugs.
7. Difficulties

Being that we used “dumb” fuzzing, the main difficulty we encountered was actually from the very method we used to test the servers originally. Our fuzzing application was sending commands so rapidly and was not waiting for the server to reply back in an ample amount of time, which would cause commands to be sent before they were prompted. The most common was the sending of the PASS command before the server had accepted or replied back from receiving the USER command. Usually this would be logged in the server applications as a "503 Bad Sequence of Commands" error. Once this happened all commands for that fuzzing sequence would not run. However this would usually occur farther into the fuzzing sequence, so we could test a limited amount of commands and restart the test with a different set of commands afterwards.

To prevent hackers or bots from attacking a sever most applications also utilized a flood detection feature which would look to see how fast commands were being sent and would drop the connection as soon as a predefined limit was reached. Most server software had this option set by default, so we had to disable it to properly test fuzzing on the servers. Another option could have been to send commands at a slower rate in order to get around the flood detection, but this would have greatly slowed our rate of testing.

8. Future work

While going through the research we discovered ideas to improve our results but were unable to implement them due to time constraints. One idea would be to test various operating systems, including Windows 7 builds, Windows Server operating systems, and Unix/Linux distributions along with the FTP software that is available for the respective platforms. Another possibility is to test out full and/or commercial FTP server software. These versions of FTP software are at a different development stage and have various updates over time giving the possibility for a wider fuzzing surface. Fuzzing other protocols, as well as file formats would expand the possibilities of the utilization of dumb fuzzing. We also would have wished to get a working smart fuzzer to allow a better performance comparison with the dumb fuzzer.

9. Conclusion

Although not the best method available, dumb fuzzing does have its benefits. It can be very quick to set up and utilize if a fuzzer for the target protocol or software already exists. A dumb fuzzer only scratches the surface of possible vulnerabilities and errors whereas a smart fuzzer can dive deeper and find more obscure issues. A dumb fuzzer is best treated as a first step towards testing software [2]. However not all programs are properly equipped to handle the strain that fuzzers create, and sometimes the most dangerous exploits can be opened from a routine software update rather than bad coding. With this being the case, dumb fuzzing should still have a place in security testing for years to come.
Works Cited


