



Using fuzzing to detect security vulnerabilities

INFIGO-TD-01-04-2006

25-04-2006

Leon Juranić
Leon.Juranic@infigo.hr



©Infigo IS. All rights reserved.

This document contains information, which is protected by copyright. No part of this document may be photocopied, reproduced, or translated to another language without the prior written consent of Infigo IS.

Infigo IS d.o.o.
Horvatovac 20
10000 Zagreb

tel. +385 1 4662 700
fax. +385 1 4662 701
info@infigo.hr
www.infigo.hr



Table of Contents

1. INTRODUCTION	4
2. FUZZING TECHNIQUES	5
2.1. SESSION DATA FUZZING	5
2.2. SPECIALIZED FUZZERS	5
2.3. GENERIC FUZZERS	5
3. SERVER APPLICATION FUZZING	6
4. APPLICATION MONITORING DURING A TEST	8
5. FTP SERVER FUZZING	9
6. REAL-WORLD APPLICATION FUZZING	12
6.1. GOLDENFTPD	12
6.2. WAR FTP DAEMON - WARFTPD	15
6.3. ARGOSOFT FTP SERVER	16
7. CONCLUSION	18

1. INTRODUCTION

Most computer system intrusions are a result of security vulnerabilities in applications. Detection and identification of security vulnerabilities is an interesting process not only for security experts and system administrators, but also for intruders attempting to penetrate computer systems.

Once detected, exploits for new security vulnerabilities can be created and intruders can penetrate a high number of systems on the Internet. This is a significant threat to all information system users.

There are several methods that are used to find new security vulnerabilities:

- Source code analysis,
- Binary file analysis (static and dynamic (runtime) analysis),
- Runtime analysis of API functions,
- Fuzzing methods (fault injection) and
- Hybrid methods (various combinations of above methods).

Although all of the methods listed above can lead to detection of new security vulnerabilities, some of them are better because of their speed and simpler detection. In last couple of years, special attention was given to the technique called fuzzing. This method allows relatively fast detection of critical security vulnerabilities in various applications. Critical security vulnerabilities are usually variations of buffer overflow attacks, which allow an unauthorized user to overwrite critical parts of a vulnerable process memory. The result of exploiting this vulnerability is usually execution of a shellcode, a specially written code that was injected into a process by the unauthorized user in order to get access to the target system.

The fuzzing method is based on the fault injection technique that, by sending various input data to the target application, tries to detect a security vulnerability. When using this technique it is important to change the input data so a security vulnerability can be detected while, at the same time, it passes sanity checks by the target application. Fuzzing methods are usually used on server and client applications, file parsers, SUID applications etc. Generally, any application that is parsing data entered by a user can be submitted to a fuzzing method in order to detect security vulnerabilities.

Fuzzing techniques allow detection of almost all types of security vulnerabilities:

- Buffer overflows,
- Integer overflows,
- Format string vulnerabilities,
- Race condition vulnerabilities,
- SQL injection,
- Cross Site Scripting (XSS),
- Remote command execution,
- Filesystem attacks (reverse traversal, etc.),
- Information leaking vulnerabilities.

A high number of security vulnerabilities that are today published on various security mailing lists are detected by using a fuzzing method. There is a high number of various fuzzer tools that allow a user to check application security; this adds to the popularity of fuzzing methods.

Besides all the advantages that the fuzzing techniques offer, it is important to note it is not a universal method for security vulnerability detection. In order to detect a certain security vulnerability, the target application has a set of specific conditions for which the fuzzing tool might not be used. In cases like this, some other methods have to be applied, depending on the type of a security vulnerability that is being analyzed. When network applications are being discussed, it is important to note that the fuzzing technique is very useful for general testing of the application stability as tests like these can be destructive.

2. FUZZING TECHNIQUES

Although the term fuzzing is related to detection of security vulnerabilities by sending various, potentially malicious, input data to an application, there are different fuzzing techniques and methods, depending on how the input data is generated. Below are described some fuzzing types.

2.1. SESSION DATA FUZZING

This method is based on the analysis of legitimate sessions of the application. Certain data parts of the session are incremented and sent to the tested application. This technique is based on the fact that the fuzzer tool changes data that already exists, so it is possible that the application will enter an undetermined state which results in a security vulnerability. Below is a simple example of this fuzzing technique used in the SMTP protocol. The data that is sent to the server is being changed by using parameters that were already seen in the protocol

```
mail from: test@localhost
mailmailmailmailmailmailmailmail from: test@localhost
mail fromfromfromfromfromfromfromfrom: test@localhost
mail from::::::::::::::::::::::::::::: test@localhost
mail from: testtesttesttesttesttesttesttesttesttest@localhost
mail from: test@@@@@@@@@@@@@@@@@@@@@localhost
mail from: test@localhostlocalhostlocalhostlocalhost
```

Implementation of this fuzzing method is relatively simple, and it can result in detection of very interesting security vulnerabilities.

2.2. SPECIALIZED FUZZERS

The second, very popular, fuzzing method is building of a specialized fuzzer application that is adjusted to test implementation of a certain protocol. An advantage of this type of fuzzer application is good knowledge and adaptation to the analyzed protocol. This allows detailed and efficient tests to be performed on the target application. Specialized fuzzer tools are especially useful when testing protocols or network applications that open sessions to exchange control data between the client and server (ie. SMTP, FTP, IMAP, SSH, etc.).

In this case the fuzzer is usually built only for a specific protocol and elements of that protocol that are to be tested. A disadvantage of this type of fuzzer application is that the programmer himself defines the protocol and it's elements that will be fuzzed, so the final quality and effectiveness of the fuzzer will depend on the competency of the programmer.

2.3. GENERIC FUZZERS

The term generic fuzzers describes tools that allow a user to describe a protocol and it's elements that are to be fuzzed. A big advantage of tools like this is that they can relatively fast test security of various protocols' implementations. Generic fuzzers are not, however, as efficient as specialized fuzzers and they can overlook certain security vulnerabilities. All the tests that a fuzzer will carry out have to be defined by the user, and this can be a limitation when the user wants to test complex protocols. A very popular generic fuzzer tool is SPIKE, which can be downloaded at the following URL address: <http://www.immunitysec.com>.

One of the disadvantages of generic fuzzer tools is that less experienced users usually find them difficult to use.

3. SERVER APPLICATION FUZZING

Intruders find network applications to be the most interesting fuzzing targets. The reason for this is that they are used on the Internet and that they are typically accessible to a large number of users. These server applications are usually network services for popular protocols, such as HTTP, FTP, SMTP, POP3 and SSH. There are also other, less popular, network services that can be interesting for fuzzing.

Considering that these protocols are very popular and that they are being used every day by a large number of users, new security vulnerabilities allow intruders to compromise a great number of vulnerable servers.

The number of security vulnerabilities that can exist in an implementation of a protocol (or in a server application) grows exponentially as the complexity (or number of features) of the protocol are bigger. Features and design of a fuzzer application are very important factors that influence how efficient it will be. The goal is to build a fuzzer that will test application security as good and detailed as possible, in a reasonable period of time.

When discussing fuzzing of a service application, it is important to cover all components of the protocol that is being tested, as missing implementation of a seemingly non important part of the protocol can result in missing a critical security vulnerabilities.


In order to shorten time needed to generate all possible input data combinations that can result in detection of a security vulnerability, fuzzer tools usually have a list of predefined input data and its length. Test input data consists of character strings that will be multiplied until they reach a defined length, after which the input data is sent to the target application. The length of the input data is very important for the whole fuzzing process. A character string that is too short will miss the security vulnerability being tested, while a character string that is too long may be discarded by the application so the security vulnerability again will not be detected. For this reason, it is best to define character string lengths by the usual size of buffers in applications.

The buffer sizes that are most commonly used in applications are typically power of 2. It is recommended that, when defining the test input data, lengths that are a bit larger than buffers are used (ie. 20, 40, 70, 130, 260 etc.). It is also important that the input data is correlated with the protocol being tested; this is important for special characters and values that are specific for that protocol and its implementation. For example, if a SMTP server is being fuzzed, the data that is sent to the target application has to be correlated to the construction of e-mail addresses. The data has to consist of characters like '@', ';', '.', '<', '>' and similar. If a HTTP server is being fuzzed, the data has to consist of characters like '%xx', './', '?', '=' and so on.

The better the test input data is adjusted to the protocol that is being tested, the higher is the chance that a security vulnerability will be detected. The fuzzer also has to test the application with as much input data as possible, including the characters which are not in the visible (alphanumeric, punctuation etc.) ASCII set. Besides the predefined data and buffer lengths that will be used to send the data, the fuzzer can also generate pseudo-random data that will expand the test set defined by the programmer.

The usage of fuzzer methods is demonstrated further in this document. *Infigo FTPStress Fuzzer* tool (http://www.infigo.hr/en/in_focus/tools) was used on various FTP server applications. This tool proved to be very successful in detecting security vulnerabilities in a large number of commercial and free FTP service applications. The tool's design is very simple and is based on the concepts previously described in this document.

The testing process of security vulnerabilities in server applications consists of preauth and postauth vulnerabilities. The preauth vulnerabilities, as their name implies, do not require that a user is previously authenticated, so they are much more dangerous than the postauth vulnerabilities. The postauth vulnerabilities require an intruder to have a valid username and password. Preauth security vulnerabilities are typically less present than postauth vulnerabilities.



A good fuzzer tool should be able to test a complete protocol, including both the preauth and postauth phases.

A fuzzer tool can also detect security vulnerabilities that have already been published and patched in the application that is being tested. This allows a user to test the efficiency of the fuzzer tool.

4. APPLICATION MONITORING DURING A TEST

It is very important to monitor the application behaviour during a fuzzing test. When an application is being fuzzed in order to detect a buffer overflow, the most important part of this process is to monitor it. Debuggers are typically used to monitor the target application during the test.

A very popular and free debugger for Windows environments is OllyDbg (<http://home.t-online.de/home/Ollydbg>), while GDB is the most common choice on Unix platforms (<http://www.gnu.org/software/gdb>). GDB comes with most of the Unix based operating systems.

Besides debugging the application, there are other parameters that should be monitored during the fuzzing process. These parameters are the memory usage, network activities, file system activities, registry file accesses and similar. There are very useful and free tools which allow monitoring of these parameters. The Sysinternals Web page (<http://www.sysinternals.com>) has most of the tools that are needed.

The tool called *File Monitor* is used to monitor filesystem activities; *Process Explorer* is used to get detailed information about running processes and *Registry Monitor* can be used to monitor access to the registry file.

5. FTP SERVER FUZZING

FTP is a relatively simple protocol, but due to the high number of commands and various parameters many input interfaces exist that can result in a security vulnerability. Security vulnerabilities that are found in FTP servers are usually buffer overflows and other vulnerabilities related to memory operations, such as format string vulnerabilities, filesystem access vulnerabilities that allow an attacker to browse through the file system (outside the *ftproot* directory) and others.

When fuzzing a FTP server, it is important to test all the FTP commands with as many as possible different input data groups. It is also important to test various states of the FTP server when a file is being transferred, before and after a user has been authenticated.

One of the tools that allows security testing of FTP server applications is *Infigo FTPStress Fuzzer*. The tool is available for download from the Infigo IS Web site (http://www.infigo.hr/en/in_focus/tools). It allows a user to define FTP commands that need to be tested and the length and type of data that will be sent to the target application. The tool is free and works on Windows operating systems.

The following figure (Figure 1) shows the main window of the *Infigo FTPStress Fuzzer* tool.

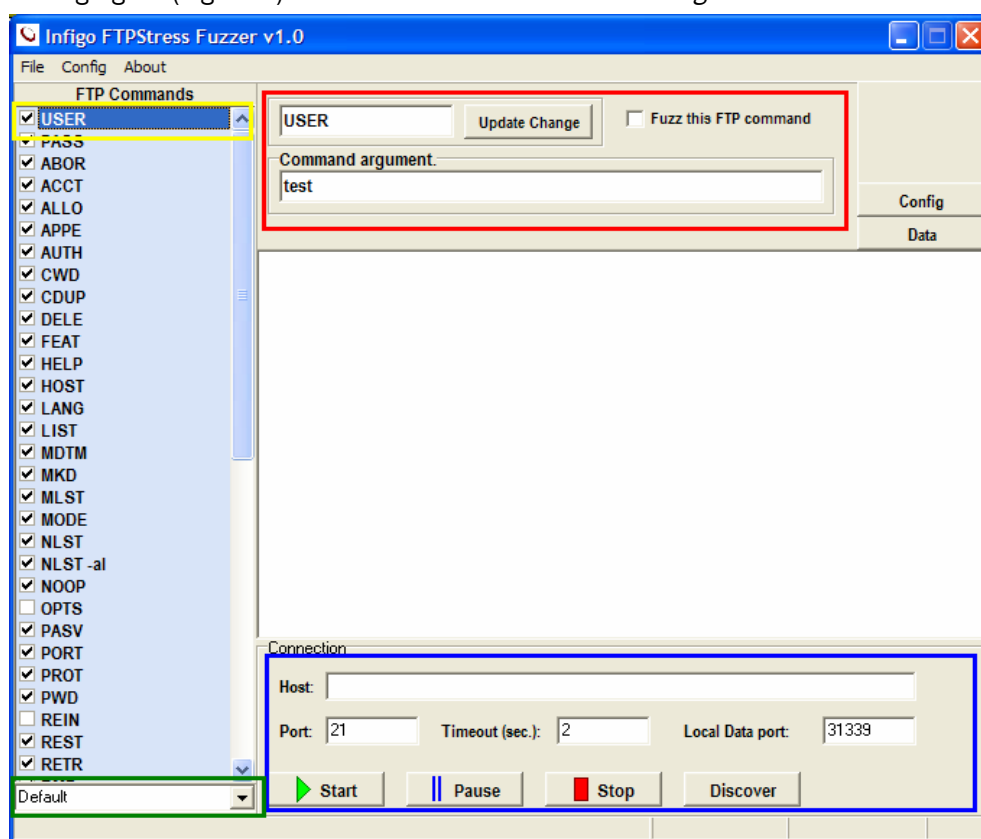


Figure 1: Infigo FTPStress Fuzzer

On the left side of the main window is a list of basic FTP commands and extensions. Inside the yellow frame is the FTP command **USER** that is used to send the username to the application. If the checkbox is ticked, during the fuzzing process this command will be sent to the target FTP server.

Inside the red frame are additional parameters for the marked command:

- command name (**USER**),
- command argument(s) (**test**),

- the field **Fuzz this FTP command**, which defines whether a fuzzing method will be applied to this command. If this checkbox is ticked, the defined argument is ignored and will be replaced with fuzzing values as configured. If this checkbox is not ticked, the application will sent the argument defined in the field **Command argument** to the target server.

If a user wants to test preauth security vulnerabilities, **USER** and **PASS** command have to have the **Fuzz this FTP command** checkbox ticked. Otherwise, if the postauth phase needs to be tested, these commands have to have a valid username and password assigned, and the **Fuzz this FTP command** checkbox must not be ticked. In the green frame it is possible to select certain sets of commands which are grouped in predefined categories.

In the blue frame a user has to define the target server's IP address, the target port, the timeout value and a local data port that will be used to send the input data to the server. The timeout value is very important for the fuzzing process because a wrong value can result in a security vulnerability being missed or the whole fuzzing process can be considerably slowed down. When a FTP server on the local host is tested, the optimal value for the timeout configuration variable is between 4 and 8, depending on the target FTP server.

By pressing the button **Config** the configuration screen will be opened. This screen allows detailed configuration of the fuzzer, including configuration of the length and data type that will be sent to the target server and the type of fuzzing method that will be used.

All predefined configuration options in the tool can be changed; this allows a flexible fuzzing process. *Infigo FTPStress Fuzzer v1.0* supports two main fuzzing methods:

- **Fuzz only one command at time** and
- **Fuzz all selected commands in same FTP session**

If the first fuzzing type is selected (**Fuzz only one command at time**), the fuzzer will test every command independantly with every selected data of every length. Figure below shows the result of sending the data to the target application when this fuzzing method is selected:

```
[ CMD: [ABOR] FUZZ: [AAAAAAAAAAAAAAAAAAAAA] SIZE: 30 ]
RECV: 226 abort successful

[ CMD: [ABOR] FUZZ: [AAAAAAAAAAAAAAAAAAAAA] SIZE: 70 ]
RECV: 226 abort successful

[ CMD: [ABOR] FUZZ: [AAAAAAAAAAAAAAAAAAAAA] SIZE: 150 ]
RECV: 226 abort successful

[ CMD: [ABOR] FUZZ: [AAAAAAAAAAAAAAAAAAAAA] SIZE: 330 ]
RECV: 226 abort successful

[ CMD: [ABOR] FUZZ: [AAAAAAAAAAAAAAAAAAAAA] SIZE: 520 ]
RECV: 226 abort successful

[ CMD: [ABOR] FUZZ: [AAAAAAAAAAAAAAAAAAAAA] SIZE: 700 ]
RECV: 226 abort successful
```

The fuzzer first sends the **ABOR** command with every defined string length and with every defined string data.

If the second fuzzing method is selected (**Fuzz all selected commands in same FTP session**), the result will be as shown below:

```
[ CMD: [ABOR] FUZZ: [AAAAAAAAAAAAAAAAAAAAA] SIZE: 30 ]
RECV: 226 abort successful

[ CMD: [ACCT] FUZZ: [AAAAAAAAAAAAAAAAAAAAA] SIZE: 30 ]
RECV: 500 Account disabled, contact administrator

[ CMD: [ALLO] FUZZ: [AAAAAAAAAAAAAAAAAAAAA] SIZE: 30 ]
```

RECV: 200 allocation ok

[CMD: [APPE] FUZZ: [AAAAAAAAAAAAAAAAAAAAA] SIZE: 30]
RECV: (null)

In this example the tool will send every selected command with the same test data and same length of the data. After all the commands have been tested, the fuzzer will use the second test length.

Different fuzzing methods can detect different security vulnerabilities. During the fuzzing process it is important to analyze how the target server is behaving so the fuzzer can be properly adjusted.

6. REAL-WORLD APPLICATION FUZZING

FTP server fuzzing is demonstrated in couple of real-world examples which show 0-day security vulnerabilities (these security vulnerabilities were not known before the server was tested). The following applications are used in these examples: *GoldenFTPd*, *WarFTPD* i *ArgoSoft FTP Server*.

6.1. GOLDENFTPD

Golden FTP Server can be downloaded from <http://www.goldenftpserver.com/>. After the application has been installed, a user with the username 'test' and password 'hack' has to be created for the fuzzing process. Now the FTP server can be started together with the *Infigo FTPStress Fuzzer* tool.

The USER command argument is set to 'test' and the PASS command argument is set to 'hack'. In the field 'Host', the IP address of the system on which Golden FTP Server is running has to be entered and the target TCP port is configured in the 'Port' field. The timeout value for this example is 8 and the **Fuzz all selected commands in the same FTP session** fuzzing method is selected. The following figure (Figure 2) shows start of the fuzzer configured as described above.

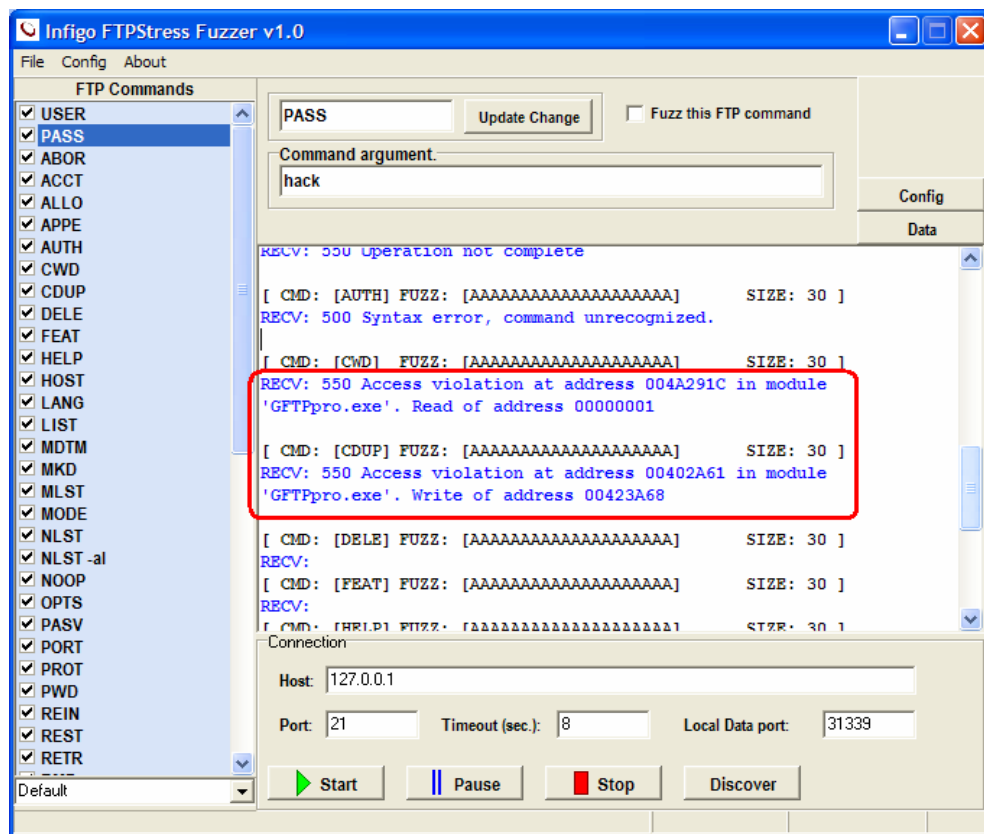


Figure 2: Proces fuzzinga

Fuzzing results are visible in first couple of tests when GoldenFTPd generated the **Access violation** exception. This means that the server process is trying to read a part of the memory which is not available to it. The second error is even more interesting as the server process tried to write to the memory location which was not available or the process didn't have the authorization to write data to that memory location. By using an exception handler to catch exceptions, GoldenFTPd server makes security vulnerability detection easier. Once an exception has been caught, GoldenFTPd will list detailed information on the exception type and why it happened. This also makes it easier to exploit buffer overflow vulnerabilities as the target application will reveal sensitive information about the process' memory environment. If there

are serious security vulnerabilities in GoldenFTPd, an intruder has enough information to write an exploit, even without having a locally installed GoldenFTPd version.

In order to analyze this security vulnerability in more details, OllyDbg debugger has to be attached to the GFTPpro.exe process. As there are a lot of Access violation exceptions and as the server application has an exception handler defined, OllyDbg should be configured so that all Access violation exceptions are delivered to the application. In this case only exceptions that can't be processed will abort the application. This can be configured in the **Options->Debugging Options->Exceptions**; checkboxes **Memory access violation** and **Single step break** have to be ticked. Also, in order to shorten the fuzzing process, only the basic group of commands (**File Commands** in this case) will be selected to be fuzzed.

After about one minute of fuzzing, *Infigo FTPStress Fuzzer* will report that it can not connect to the target server anymore.

```
[ Connecting to 127.0.0.1:21... ]
[ Connected, starting fuzz process... ]
220 Golden FTP Server PRO ready v2.50

[ USER: [test] ]
331 User name okay, need password.

[ PASS: [hack] ]
230 User logged in, proceed.

> Sending PORT Command - PORT 127,0,0,1,122,107

RECV: 200 PORT Command successful.

[ CMD: [NLST -al] FUZZ: [./A./A./A./A./A./A./] SIZE: 330 ]
[ Connecting to 127.0.0.1:21... ]
[ Connected, starting fuzz process... ]
[ USER: [test] ]
[ PASS: [hack] ]
[ CMD: [RMD] FUZZ: [./A./A./A./A./A./A./] SIZE: 330 ]
[ Connecting to 127.0.0.1:21... ]
[ Connected, starting fuzz process... ]
[ USER: [test] ]
[ PASS: [hack] ]
[ CMD: [RNFR] FUZZ: [./A./A./A./A./A./A./] SIZE: 330 ]
[ Connecting to 127.0.0.1:21... ]
[ Connected, starting fuzz process... ]
[ USER: [test] ]
[ PASS: [hack] ]
[ CMD: [RNT0] FUZZ: [./A./A./A./A./A./A./] SIZE: 330 ]
[ Connecting to 127.0.0.1:21... ]
[ Connected, starting fuzz process... ]
[ USER: [test] ]
[ PASS: [hack] ]
[ CMD: [SIZE] FUZZ: [./A./A./A./A./A./A./] SIZE: 330 ]
[ Connecting to 127.0.0.1:21... ]
[ Connected, starting fuzz process... ]
[ USER: [test] ]
[ PASS: [hack] ]
[ CMD: [XCUP] FUZZ: [./A./A./A./A./A./A./] SIZE: 330 ]
[ Connecting to 127.0.0.1:21... ]
[ ERROR: Cannot connect to target!!! ]
[ SERVER IS MAYBE DEAD BECAUSE OF FUZZING!!! ]
```

As shown above, the target server stops answering after the marked line was executed (NLST command with the parameter of './A' repeated 330 times). Analyzing the process with the

OldyDbg debugger shows that the EIP register contains **0x0000412F**, which is ASCII value of the string **./A** - the test input data that was generated by the fuzzer. Stack analysis further shows that the SEH (*Structured Exception Handler*) was overwritten with **0x0000412F**. The register status is shown below (Figure 3).

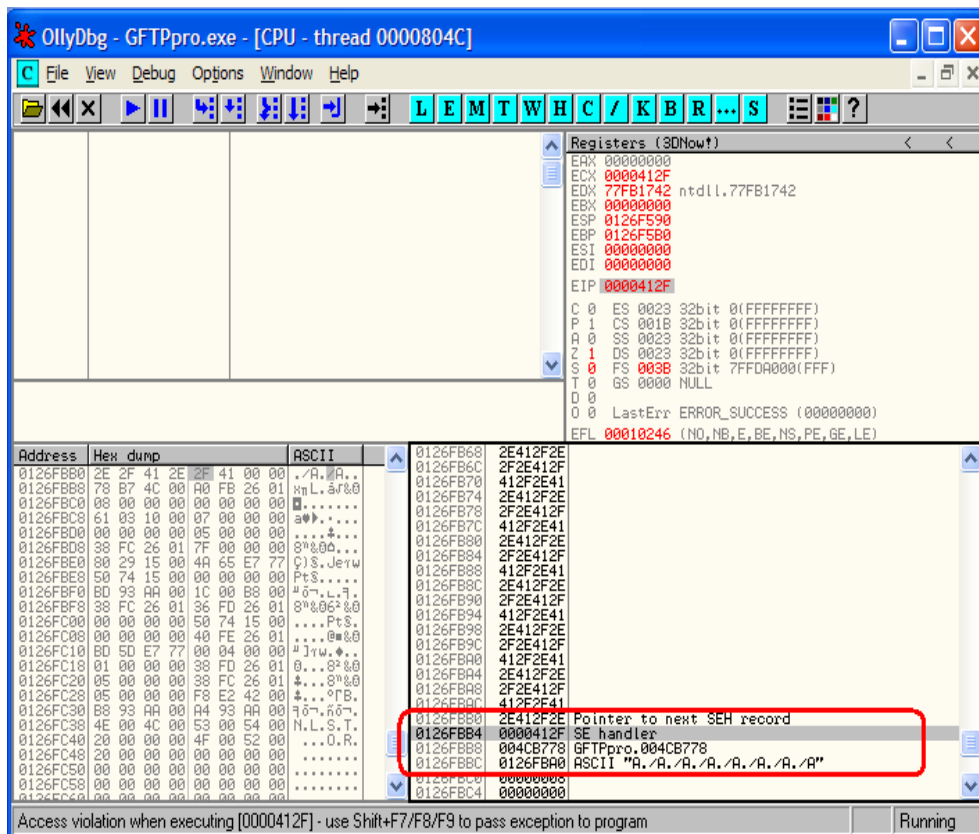


Figure 3: SEH overflow

Considering that the final goal is to completely control the EIP register, the configuration of *Infigo FTPStress Fuzzer* has to be changed and the test length of **330** (which caused the program to crash) has to be changed to **332**. This way the SEH handler will be completely overwritten and it is possible to control the EIP register. Full control of the EIP register is shown below (Figure 4).

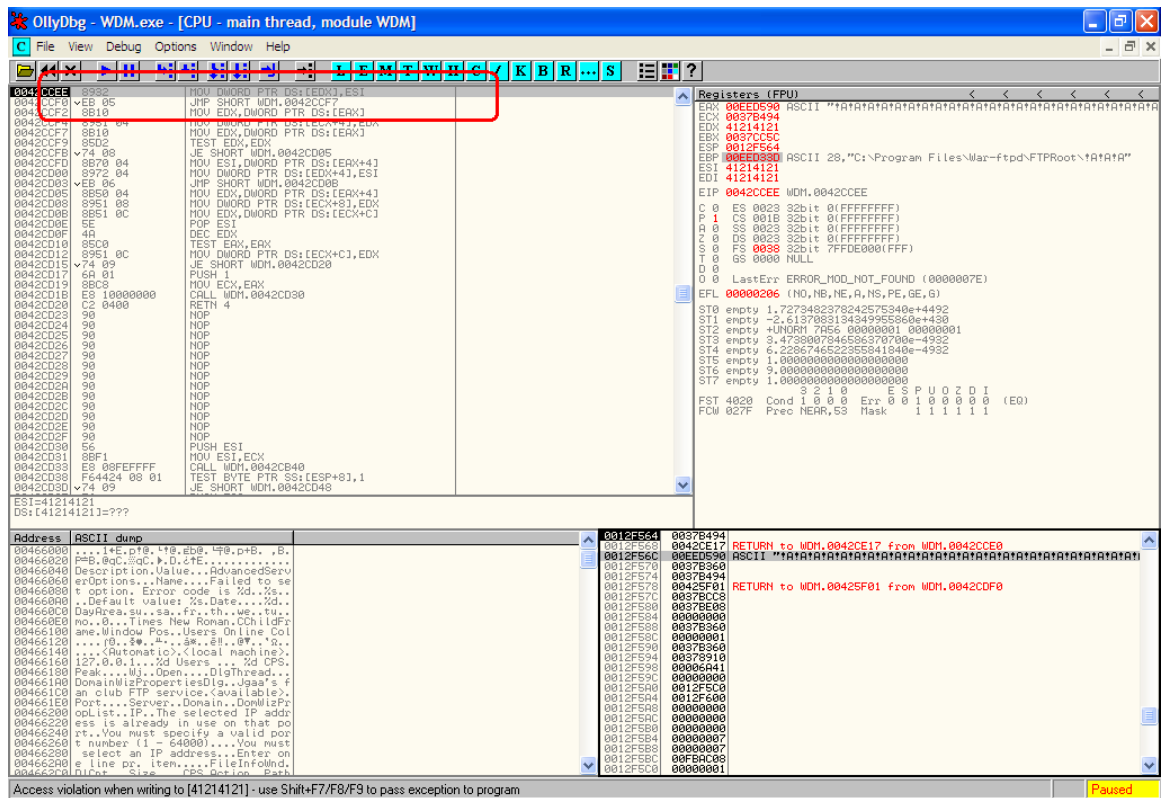



Figure 5: WDM.exe buffer overflow

6.3. ARGOSOFT FTP SERVER

ArgoSoft FTP Server can be downloaded from <http://www.argosoft.com>. This FTP server application is difficult to fuzz because there are many security vulnerabilities that cause the program to crash often. This makes an analysis difficult. As ArgoSoft FTP Server also has problems when the OllyDbg debugger attaches to it's process, a different product has to be used to monitor it. The product that will be used to monitor it is FaultMon by eEye (<http://www.eeye.com>).

After couple of minutes of fuzzing, the process will be aborted and FaultMon will show status of the processor when the process crashed. Analysis of the data that *Infigo FTPStress Fuzzer* used to test the FTP server shows that the execution was aborted after the **RNTTO** command with the test input data of '&A', size of 3000 octets.

```
C:\test\win\Chapter_14\bin>faultmon.exe -p 26844
...
pid=7E94 tid=83F8 EXCEPTION (unhandled)
-----
Exception C0000005 (ACCESS_VIOLATION writing [00000000])
-----
EAX=00000000: ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??
EBX=00000000: ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??
ECX=00260041: 01 90 01 00 01 00 01 00-01 A0 01 00 01 00 01 B0
EDX=77FB1742: 8B 4C 24 04 F7 41 04 06-00 00 00 B8 01 00 00 00
ESP=01111204: 2E 17 FB 77 E8 12 11 01-B0 F4 20 01 04 13 11 01
EBP=01111224: D0 12 11 01 00 17 FB 77-E8 12 11 01 B0 F4 20 01
ESI=00000000: ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??
EDI=00000000: ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??
EIP=00260047: 01 00 01 A0 01 00 01 00-01 B0 01 00 01 00 01 00
--> ADD [EAX],EAX
```



As this example shows, the **EIP** register will have the value of **0x00260047** when the server will crash. This is the UNICODE value for the string '&A' + 6, which is the input data generated by the fuzzer.

7. CONCLUSION

Fuzzing techniques allow detection of critical security vulnerabilities in short time periods for various applications, as was shown in this document. It is important to say that fuzzing techniques, although being effective, are not the final solution for detection of all security vulnerabilities that exist in an application. Some security vulnerabilities, due to their nature, are impossible to detect with a fuzzing technique.

The main benefits of using a fuzzing method are simplicity, efficiency and automation which lead to a considerable speed up of the whole process of security vulnerabilities detection. These methods are useful for testing a large number of applications and products; this is confirmed by a growing number of free and commercial fuzzer tools that have been released lately.