Fuzzing: An introduction to Sulley Framework

May 6th, 2013
Brian MARIANI

WHAT IS FUZZ TESTING?

- According to Wikipedia:

  ✓ Fuzzing is a **software testing technique**, often **automated or semi-automated**.

  ✓ It involves **providing improper, unexpected or random data to the inputs of a computer program**.

  ✓ While the fuzzing process is running, the targeted **program is monitored for exceptions,** such as crashes, in order **to find potential memory corruption scenarios**.

  ✓ Fuzzing is commonly used **to test for security issues,** so as to evaluate a wide variety of software utilities on various platforms.

DIFFERENT FORMS OF FUZZING

- Even if **everybody does not agrees** with the terms, there are basically **two main forms** of fuzzing techniques:
  - ✓ **mutation-based fuzzing**
  - ✓ **generation-based fuzzing**

MUTATION-BASED FUZZING

- When **mutation-based fuzzing** is applied as a fuzzing form, known good data is collected, such as **files or network traffic**.

- Later, this data **will be slightly modified**. These modifications could be **random** or using **heuristic methods**.

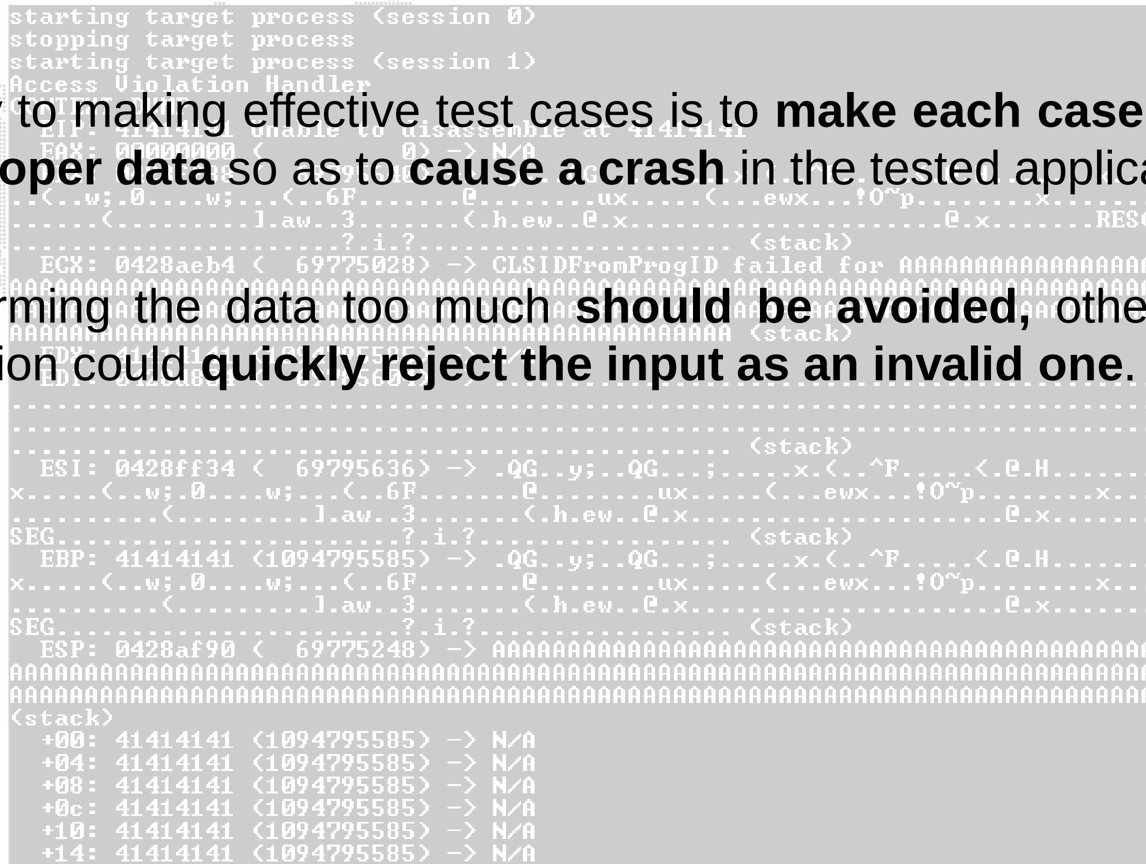
- Some examples of heuristic mutations include **replacing small strings with longer strings** or changing **length values** to large or small values.

GENERATION-BASED FUZZING

- **Generation-based fuzzing** starts from a specification or RFC which describes the internals of a specific format or network protocol.

- The key to making effective test cases is to **make each case different from proper data** so as to **cause a crash** in the tested application.

- Transforming the data too much **should be avoided,** otherwise the application could **quickly reject the input as an invalid one**.



DISCOVERED VULNERABILITIES

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- **Any kind of security vulnerabilities** can be found using fuzzing techniques. Security researchers often rely on fuzzing to find security issues.

- According to the excellent book "**Fuzzing for software security testing and quality assurance**" some statistics show that:

  - ✓ **Over 80%** of communications software implementations today **are vulnerable to implementation-level security flaws**.

  - ✓ **25 out of 30 Bluetooth implementations crashed** when they were tested with Bluetooth fuzzing tools.

WHAT IS A FUZZER?

- A fuzzer is therefore a software **that deliberately sends out malformed data** to the input of a program.
- One of the first who wrote a fuzzer was **Barton Miller** from the **University of Wisconsin**.
- He realized that if **arbitrary inputs were given** to core Unix command line utilities, such as **ls, grep or ps**, these tools will react in **an unexpected way**.
- This surprised him, and **he started to write one of the first automated tools** specifically designed to crash a program.
- In add, he provided **public access to his tool** source code, the test procedures and raw result data.

TYPE OF FUZZERS

- **Static and random template-based**: It **only tests simple request-response protocols**, or **file formats**. There is **no dynamic functionality** involved.

- **Block-based fuzzers**: They implement **an elementary structure** for a **simple request-response protocol** and could contain some **basic dynamical functionalities**.

- **Dynamic generation or evolution based fuzzers**: These fuzzers **do not automatically understand the fuzzed protocol** or file format, but they will absorb it based on a feedback loop from the target system.

- **Model-based or simulation-based fuzzers**: They implement the tested interface either through a model or a simulation.

CLIENT AND SERVER-SIDE FUZZERS

- Some fuzzers are designed for **client side testing** and others for **server side testing**.
- For example a client-side test for **HTTP protocol** will target **browser software**.
- Likewise, a **server-side** fuzzing tests the robustness of a **web server.**
- Some of the existent fuzzers support **both server and client** testing, or even middleboxes that simply proxify, forward and analyze protocol traffic.

Well-known fuzzers

- Our goal is not to mention all the existent fuzzers in the security arena, but **the more relevant of them are**:

  - ✓ **GPF**
  - ✓ **Taof**
  - ✓ **ProxyFuzz**
  - ✓ **Mu-4000**
  - ✓ **Codenomicon**
  - ✓ **beStorm**
  - ✓ **Peach**
  - ✓ **Sulley**
  - ✓ **SPIKE**
  - ✓ **COMRaider**
  - ✓ **AXman**

THE SULLEY FUZZING FRAMEWORK

- Sulley was authored by two renowned security researchers, **Pedram AMINI** and **Aaron Portnoy.**
- It is a fuzzer development and fuzz testing framework consisting of **multiple extensible components**.
- The real goal of this excellent framework is to simplify not only **data representation** but to **simplify data transmission and target monitoring** as well.
- Sulley not only has **impressive data generation** but includes **many other important aspects** that new generation fuzzers should provide.

THE POWER OF SULLEY

- Sulley **monitors the network** and systematically **maintains records**.
- It **instruments and monitors** the **health of the target**, capable of reverting to a known good state using multiple methods.
- It is **capable to detect, track and categorize the** uncovered faults into the fuzzed application.
- Sulley can also **fuzz in parallel mode**, which significantly **increase the fuzzing speed.**
- It can automatically **determine what unique sequence of test cases has triggered the faults**.

DATA REPRESENTATION

- To represent a dialog or protocol between two computers Sulley used the **block-based** approach which combines **simplicity and flexibility**.
- Sulley uses the **block-based method** to produce **individual requests**.
- The requests will **later be tied together** to form what Sulley calls a **Session**.

```
s_initialize("your request")
```

- When the basic structure is done, one can **start to add primitives**, **blocks and nested blocks** to the request.
- We do not intend to describe **all the supported data representation in Sulley**. The following slides **gives you a preview of what Sulley is capable to do**. For more information please consult reference **[4]**.

STATIC AND RANDOM PRIMITIVES

- The simplest primitive is the **s_static(),** which adds a **static unmutating value** of an arbitrary length to the request.

- It exists several aliases in Sulley, for example: **s_dunno(), s_raw() and s_unknown()** are all aliases of the **s_static primitive.**

```
s_static("sulley\x00was\x01here\x02")
s_raw("sulley\x00was\x01here\x02")
s_dunno("sulley\x00was\x01here\x02")
s_unknown("sulley\x00was\x01here\x02")
```

INTEGERS

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- **ASCII protocols** and **binary data** contains many sized integers values. An example can be the **Etag** field in **HTTP** protocol.

- **Sulley takes good care** to represent this type of information **implementing different types of primitives** such as:

1 byte: s_byte(), s_char()

2 bytes: s_word(), s_short()

4 bytes: s_dword(), s_long(), s_int()

8 bytes: s_qword(), s_double()

STRINGS AND DELIMITERS

- **Hostnames, passwords and usernames** are some of the strings that **can be found everywhere**.
- The Sulley framework provides the **s_string()** primitive for **representing the data string**.
- The primitive takes a **single** and **mandatory** argument.
- Lets say you would like to fuzz the following string **<meta name="robots">,** here is how Sulley will understand your whishes:

```
s_delim("<")
s_string("meta")
s_delim(" ")
s_string("name")
s_delim("=")
s_delim("\"")
s_string("robots")
s_delim("\"")
s_delim(">")
```

BLOCKS

- Once the **primitives are well defined** the next step is to nest them properly **within blocks**.
- Blocks are defined and opened with **s_block_start()** and closed with **s_block_end().**
- Each block **must be given a name**, specified as the first argument to **s_block_start().**
- Because we will later analyze a real fuzzing case, we will not give more details **about blocks** in this slide.

- When **the requests are defined** one must **attach them** in a **session**.

- Sulley is **efficient to fuzz very deep within a protocol**. This is done by linking the requests together. The next example is **a sequence of requests which are tied together**:

```
from sulley import*
s_initialize("helo")
s_static("helo")
s_initialize("ehlo")
s_static("ehlo")
s_initialize("mail from")
s_static("mail from")
s_initialize("rcpt to")
s_static("rcpt to")
s_initialize("data")
s_static("data")

sess = sessions.session()
sess.connect(s_get("helo"))
sess.connect(s_get("ehlo"))
sess.connect(s_get("helo"), s_get("mail from"))
sess.connect(s_get("ehlo"), s_get("mail from"))
sess.connect(s_get("mail from"), s_get("rcpt to"))
sess.connect(s_get("rcpt to"), s_get("data"))
fh = open("session_test.udg", "w+")
fh.write(sess.render_graph_udraw())
fh.close()
```

A REAL CASE FUZZING EXAMPLE (1)

- **Let's stop with theory** and analyse a real case study about a vulnerability found in **October 15th** by **High-Tech Bridge Security Research Lab**.
- The flaw was found in a media webserver with the name of **TVMOBiLi.**
- After fuzzing for a while we can find the possibility **to crash the entire server** just by sending **malicious HTTP** crafted requests to it.
- In the following slides **we will explain how the setup of Sulley can be done**, so as to better understand the framework, and we will also **show the first crash that Sulley caught.**
- Studying or reversing the vulnerable code in detail **is out of the scope of this document**. More information about this vulnerability can be found here.

A REAL CASE FUZZING EXAMPLE (2)

- Our scenario relies in a **VMware Workstation** environment with two **Windows XP SP3 machines** up to date.

- The **attacker machine** has the IP address **192.168.175.130** and the **victim machine IP** is **192.168.175.129.**

- When fuzzing with Sulley or other fuzzing framework, it is very important that the **Attacker** and **Victim** machine are in an **isolated environment.**

- **Sulley will send network packets at a respectable speed**, so if your environment is well isolated this will **increase efficiency** and you will not disturb other hosts.

Attacker Machine

A REAL CASE FUZZING EXAMPLE (4)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- Let's first check the **python script that takes care of the HTTP fuzz protocol**.

```
from sulley import *
s_initialize("HTTP")

s_group("verbs", values=["GET", "HEAD"])
if s_block_start("body", group="verbs"):
    s_delim(" ")
    s_delim("/")
    s_string("AAAAA")
    s_delim(" ")
    s_string("HTTP")
    s_delim("/")
    s_string("1")
    s_delim(".")
    s_string("1")
    s_static("\r\n\r\n")
s_block_end("body")
```

- First of all we create our **Sulley request**. Then we define a **s_group primitive** that will contain all the **HTTP methods** that we would like to fuzz.

- Later between two **s_block primitives** we define our **string and delimiters** in order to perfectly respect the **HTTP protocol definition. Finally** we named this file **httpcallAX.py**

A REAL CASE FUZZING EXAMPLE (5)

- Now is time to **define our main session** file and its **agents**.

```
from sulley import *
from requests import httpcallAX

sess   = sessions.session(session_filename="audits/http.session")
target = sessions.target("192.168.175.129", 30888)
target.netmon = pedrpc.client("192.168.175.129", 26001)
target.procmon = pedrpc.client("192.168.175.129", 26002)
#target.procmon_options = { "proc_name" : "tvMobiliService.exe" }

target.procmon_options = \
{
    "proc_name"          : "tvMobiliService.exe",
    "stop_commands"      : ['net stop tvMobiliService'],
    "start_commands"     : ['net start tvMobiliService'],
}

sess.add_target(target)
sess.connect(sess.root, s_get("HTTP"))
sess.fuzz()
```

- The **session file** imports our **httpcallAX module** previously created. Then the Sulley **session name is defined**.

- Later **the target information is specified** within the **IP address** and the **TCP port** to connect to.

A REAL CASE FUZZING EXAMPLE (6)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- The Sulley **network monitor** and **process monitor agents** are defined too. We will give more information on them later.

```python
from sulley import *
from requests import httpcallAX

sess   = sessions.session(session_filename="audits/http.session")
target = sessions.target("192.168.175.129", 30888)
target.netmon = pedrpc.client("192.168.175.129", 26001)
target.procmon = pedrpc.client("192.168.175.129", 26002)
#target.procmon_options = { "proc_name" : "tvMobiliService.exe" }

target.procmon_options = \
{
    "proc_name"         : "tvMobiliService.exe",
    "stop_commands"     : ['net stop tvMobiliService'],
    "start_commands"    : ['net start tvMobiliService'],
}

sess.add_target(target)
sess.connect(sess.root, s_get("HTTP"))
sess.fuzz()
```

- The name of the **target binary** is provided into the **procmon_options** block.

- It’s very important to provide to Sulley the right command **in order to stop and start the target application.**

- With these commands Sulley will be able to **properly restart the application** if a crash is produced. We will name this file **kickfuzz.py.**

A REAL CASE FUZZING EXAMPLE (7)

Victim Machine

A REAL CASE FUZZING EXAMPLE (8)

- The Sulley **process monitor agent** is responsible for perceiving errors which may occur during fuzzing process.
- This agent is hard coded to bind to **TCP port 26002** and accepts connections from the Sulley session over the **PedRPC custom binary protocol**.
- After processing each individual test case, **Sulley contacts the process agent** in order to determine **if a fault was detected.**
- If a fault is detected, information concerning **the nature of the crash is transmitted to the Sulley session** in order to display it onto the embedded Sulley Web server.
- All **the crashes are logged for posterior analysis,** which is very useful to a security researcher.

A REAL CASE FUZZING EXAMPLE (9)

- Here is the command line that appropriately **starts the process agent**.

```
python c:\sulley\process_monitor.py -c c:\sulley\audits\tvMobiliService.crash -p "tvMobiliService.exe"
```

- The **filename to serialize the crash bin class** is defined in the **audits directory**.
- The **process name** to search for and attach to is defined using **the –p option**.
- We could also use the **–L** option in order to **increase the fuzzing process verbosity**.

A REAL CASE FUZZING EXAMPLE (10)

- The Sulley **network monitor agent** is responsible for **monitoring network communications** and logging them to **PCAP files**.
- This agent binds to **TCP port 26001** and accepts connections from the Sulley session over the **PedRPC custom binary protocol.**
- Once the test case has been successfully transmitted, **Sulley contacts this agent** requesting it to flush recorded traffic to a PCAP file on disk.
- The **PCAP files are named by test case number.** This agent **does not have to be launched** on the same system as the target software.
- Let’s see how we start the network agent from the command line.

- Here is the command line that properly **starts the network agent**.

```
python c:\sulley\network_monitor.py -d 0 -f "src or dst port 30888" -P c:\sulley\audits\ -l 5
```

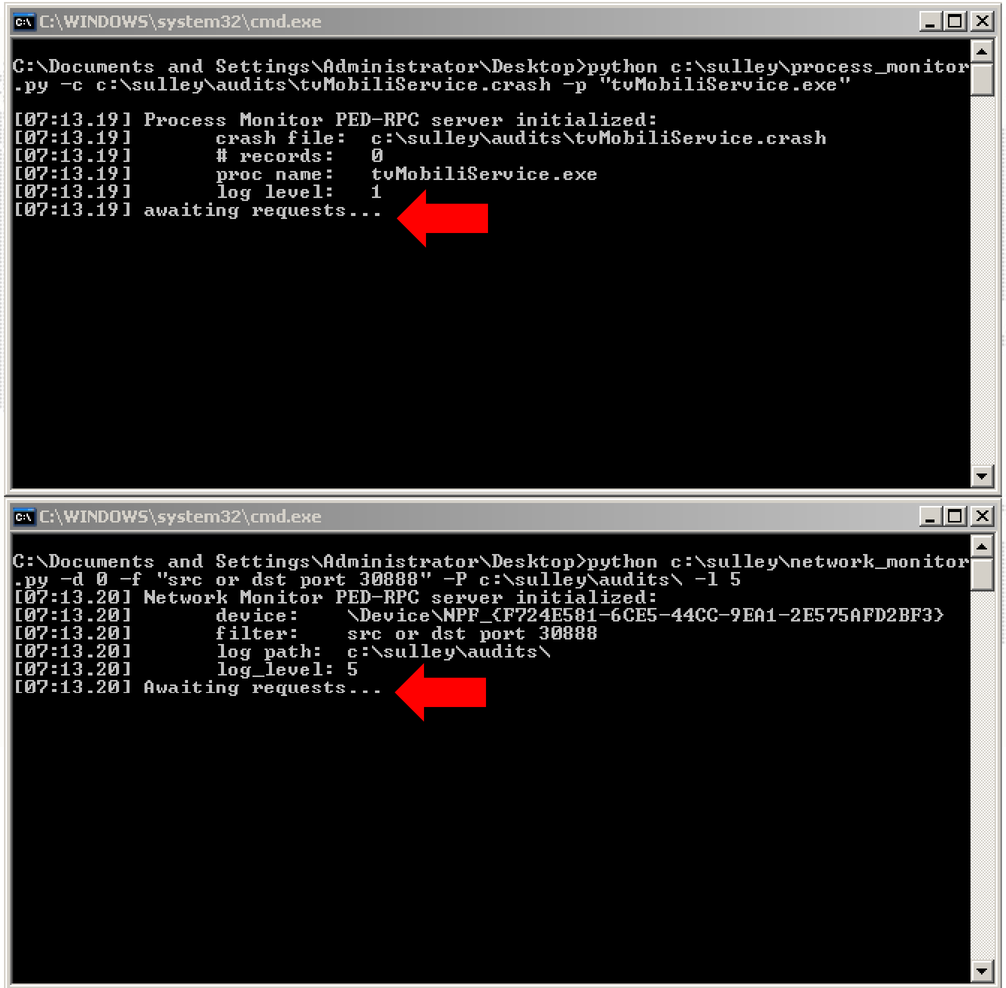
- First of all **we define the Ethernet device** to be used in order **to sniff the network traffic**. In this particular case the target device **is 0**.

- The **PCAP filter** is setup to target the **TCP port 30888,** which is the **default TCP** port where **our vulnerable application** listens to.

- Finally, we **specify the path to store our test files** and we fix the verbosity to the level five **in order to have the most complete log messages**.

A REAL CASE FUZZING EXAMPLE (12)

- Here are the agents **when they are started** on the **victim machine**:



A REAL CASE FUZZING EXAMPLE (13)

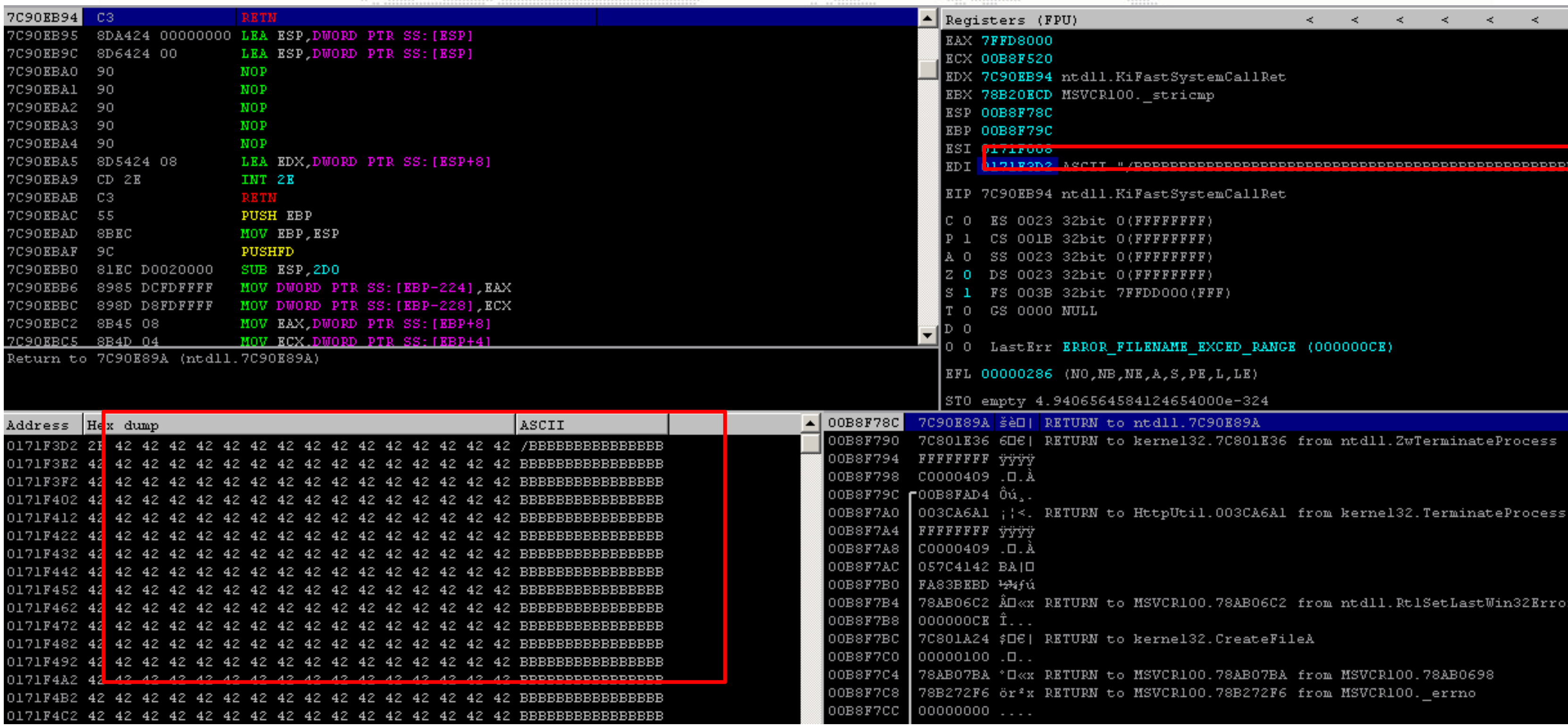
- Sulley has also a Web service who listens on **TCP port 26000,** which permits to observe produced crashes.

- In this example we are just going to attach **immunity debugger** to the vulnerable process during the first crash.

- After lunching the Sulley fuzzer on the **attacker machine, the magic of Sulley can be observed.** :]

A REAL CASE FUZZING EXAMPLE (14)

- After almost **seven minutes,** Sulley **wins over its opponent** and finds the first fault.



CONCLUSIONS

- Sulley is a powerful fuzzer consisting of **multiple extensible components**.

- It's **very easy to use.** Finding security issues with this framework can be very easy, even in complex applications.

- Sulley is an **Open Source software** and can be categorized as **one of the greatest fuzzers** nowadays.

- In future articles we will discuss **how more complex vulnerabilities can also be discovered** using the power of Sulley framework.

REFERENCES

- [1] **Fuzzing** – The software security testing and quality assurance (**Ari Takanen – Jared D. Demott – Charles Miller from Artech House**)
- [2] **Fuzzing** – Brute Force Vulnerability Testing (**Michael Sutton – Adam Grenne – Pedram Amini H. D. Moore – Addison-Wesley**)
- [3] **Analysis of Mutation and Generation-Based Fuzzing** – Charlie Miller http://ise.virtual.vps-host.net/files/papers/analysisfuzzing.pdf
- [4] **The Sulley Fuzzing Framework** www.**fuzzing**.org/wp-content/**Sulley**Manual.pdf
- [5] http://pentest.cryptocity.net/files/fuzzing/sulley/introducing_sulley.pdf

Thank you for reading

Your questions are always welcome!

brian.mariani@htbridge.com