IN5290 Ethical Hacking

Lecture 10: Vulnerability finding with fuzzing, exploits

Universitetet i Oslo
Laszlo Erdödi

Lecture Overview

- What is fuzzing, why it is useful
- What king of fuzzing techniques exist
- File format fuzzing with the Peach Fuzzer
- More about exploits

Why to find software vulnerabilities?

Some software vulnerabilities can be exploited by the attackers (not all of them, depending on the circumstances, usually only the minority of bugs are exploitable). If a software contains a bug, attackers can carry out

- Denial of Service attacks
- Remote Code Execution!

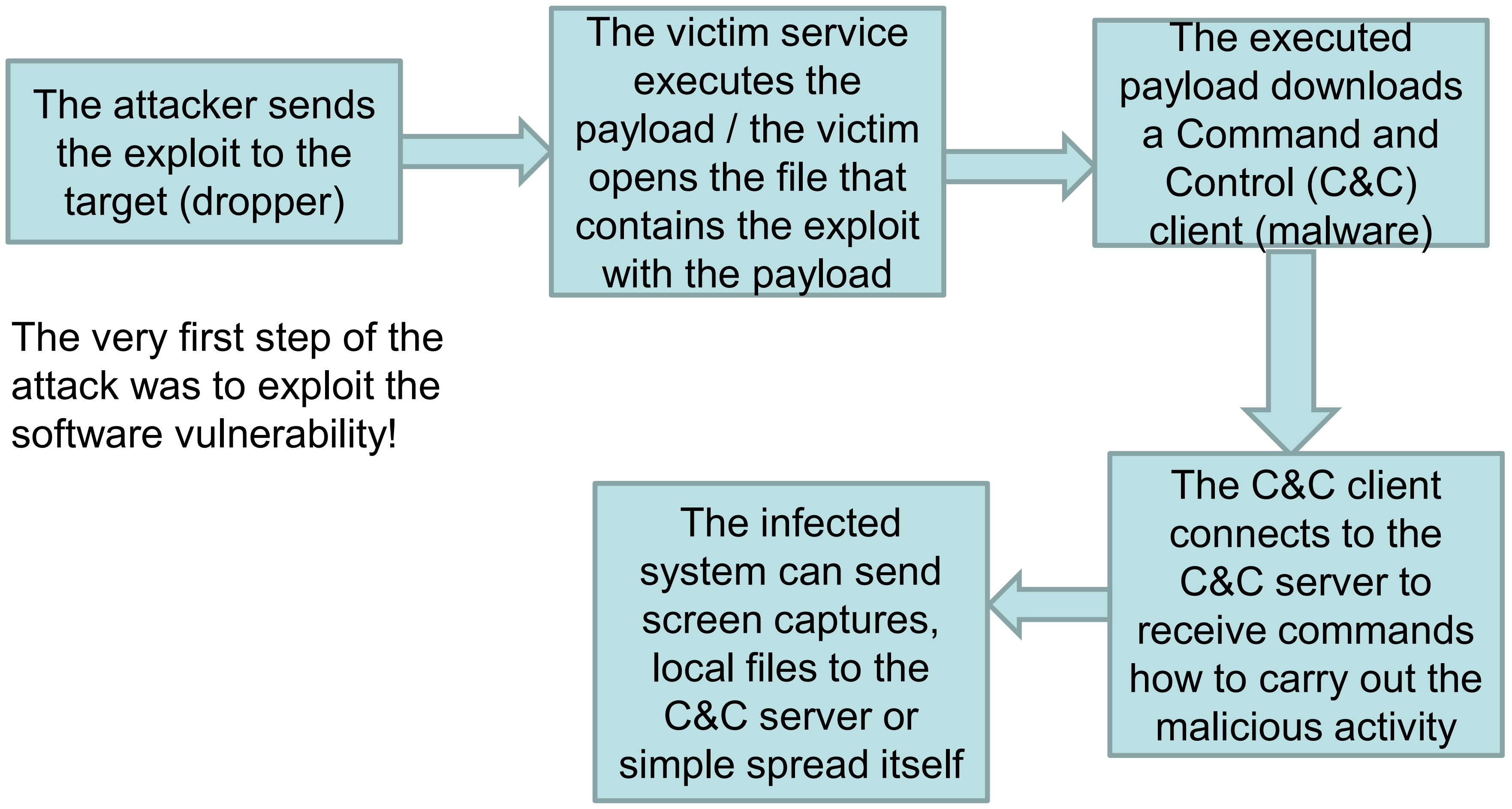
That is a huge risk. From the point of view of ethical hacking (depending on the contract) a real attack should be simulated: let’s see what is doable.

Exploit sources

- Exploit database (https://www.exploit-db.com/)
- Metasploit framework
- Github (e.g. https://github.com/0vercl0k/CVE-2019-9810)
- Darkweb
  (not for ethical hackers)

- Developing your own exploit?
  - It's a time-consuming task, the success rate can be low, but the effect can be enormous
  - A Proof of Concept should be created and warn the software producer (maybe some CERT as well), don't publish it before it is corrected

Example of a software vulnerability exploitation process


The attacker sends the exploit to the target (dropper)
The victim service executes the payload / the victim opens the file that contains the exploit with the payload
The executed payload downloads a Command and Control (C&C) client (malware)
The C&C client connects to the C&C server to receive commands how to carry out the malicious activity
The infected system can send screen captures, local files to the C&C server or simple spread itself

The very first step of the attack was to exploit the software vulnerability!

What are the steps of exploit development

- Finding the vulnerability (e.g. with fuzzing), the application crashes
- Find the reason of the crash (reverse engineering the code)
- Decide whether the control flow can be redirected or not
- Decide how and where to place the payload (e.g. on the stack, in the heap with spraying)
- Bypass all the mitigations (DEP, ASLR, sandboxing, etc.)
- Create a working version of the exploit (proof of concept)

Stack overflow:

Reason of crash: too long input, Control flow redirection: yes by overwriting the return address of the stack frame, payload place: on the stack after the *ret* together with the vulnerability exploitation, DEP bypass: ROP, ASLR bypass: memory leak, non PIE module

How to find software vulnerabilities?

- Accidently: e.g. my pdf reader is keep crashing for the same input. (Note, one crash is not crash! ☺ If it’s not possible to repeat then anything could have happened)
- AV tools can report suspicious activity such as a port is opened, a new suspicious registry entry is created. Analyzing it in a sandboxed environment can reveal unknown vulnerabilities. (Note that in this case the vulnerability was known by someone in advance who created the malware)
- Source code analysis (looking for patterns that can reflect vulnerabilities)
- Binary code static analysis: reverse engineering or advanced specific solutions (code property graphs)
- Binary code dynamic analysis (e.g. angr framework)
- Fuzzing

Fuzzing

Fuzzing or **fuzz testing** is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program. The program is then monitored for any sign of error (exceptions such as crashes, failing built-in code assertions, or memory leaks).

How the program accepts the input?

- File format fuzzing: invalid files are created and opened by the application (e.g. invalid pdf file is opened by a pdf reader)
- Protocol fuzzing or network based fuzzing: the input is provided through network protocols (e.g. http request is sent with a wrong format)

How to create invalid input?

- Mutation based input generation
  Using existing input to create slightly different versions (see demo later)
- Format description based input generation
  The format is described, the input is created using this (see demo later)
- Response based input generation
  The input is based on the received response (interactive generation)

Mutation based fuzzing

- The input is created based on existing valid input
- Mutations of input are made without the knowledge of the structure of the input (e.g. random)
- Requires little setup time
- The success is based on the mutation algorithm
- Mutation can mess up the file format and prevent it to be processed (e.g. file checksums)

Mutation based fuzzing – Peach Fuzzer

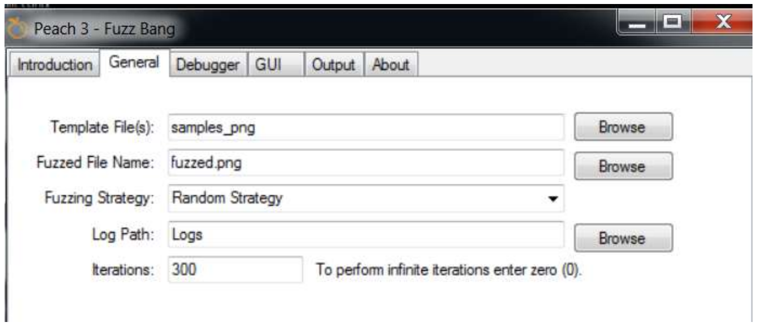
The random strategy will run forever. This strategy will select up to *MaxFieldsToMutate* elements to mutate at a time. For each selected element one of it's corresponding mutators is selected at random. Peach derives the randomness of these selections from randomly generated seed number.

- *MaxFieldsToMutate* — Maximum fields to mutate at once. default="6"
- *SwitchCount* — Number of iterations to perform before switching Data sets. default="200“

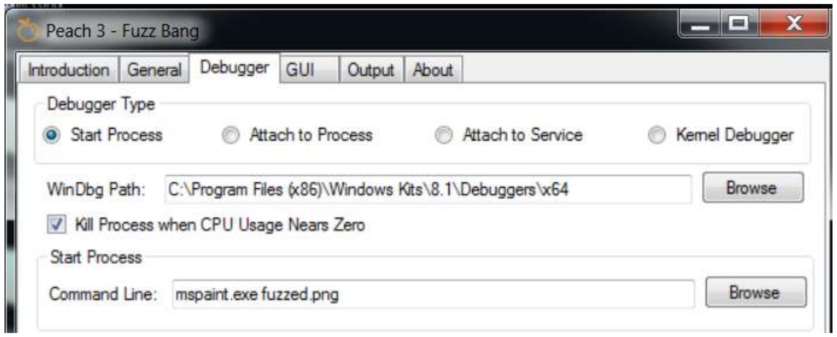
What is needed:

- PeachFuzzBang.exe
- Windbg debugger
- Valid files as input

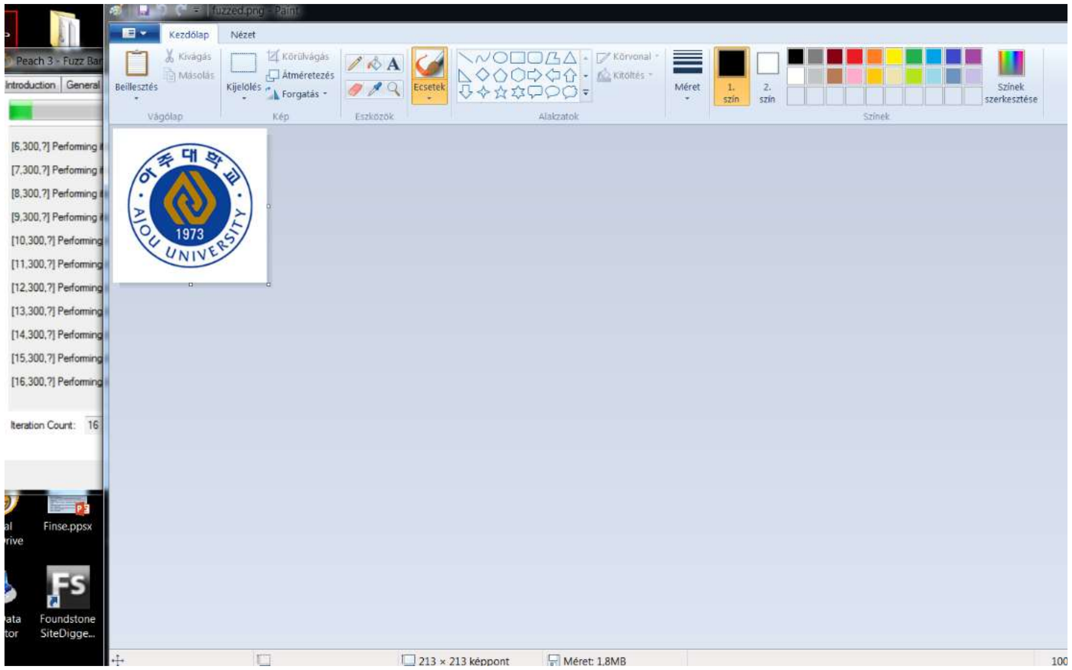
Mutation based fuzzing - Demo



Template File(s): samples_png
Fuzzed File Name: fuzzed.png
Fuzzing Strategy: Random Strategy
Log Path: Logs
Iterations: 300 To perform infinite iterations enter zero (0).

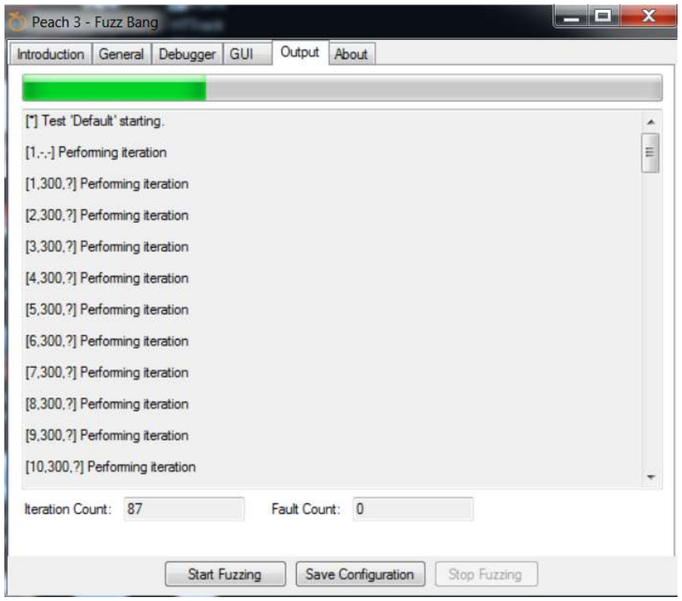


Debugger Type
● Start Process ○ Attach to Process ○ Attach to Service ○ Kernel Debugger
WinDbg Path: C:\Program Files (x86)\Windows Kits\8.1\Debuggers\x64
☑ Kill Process when CPU Usage Nears Zero
Start Process
Command Line: mspaint.exe fuzzed.png



```
[6,300,?] Performing
[7,300,?] Performing
[8,300,?] Performing
[9,300,?] Performing
[10,300,?] Performing
[11,300,?] Performing
[12,300,?] Performing
[13,300,?] Performing
[14,300,?] Performing
[15,300,?] Performing
[16,300,?] Performing
```

Iteration Count: 16



```
[*] Test 'Default' starting.
[1,-,-] Performing iteration
[1,300,?] Performing iteration
[2,300,?] Performing iteration
[3,300,?] Performing iteration
[4,300,?] Performing iteration
[5,300,?] Performing iteration
[6,300,?] Performing iteration
[7,300,?] Performing iteration
[8,300,?] Performing iteration
[9,300,?] Performing iteration
[10,300,?] Performing iteration
```

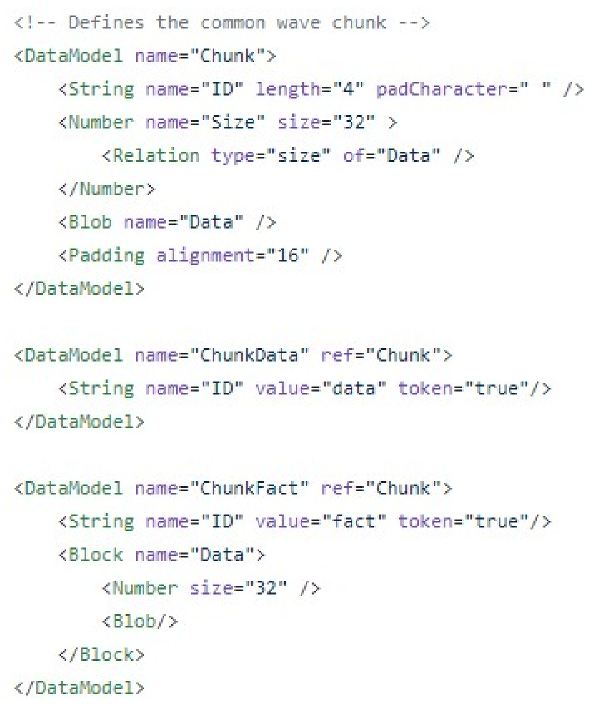
Iteration Count: 87 Fault Count: 0

Format description (generation) based fuzzing

- The file format of protocol is described (what kind of variables are stored in the file in which place, relations, etc)
- Very time consuming to describe the input format (e.g. the pdf reference 1-7 (file description from 2006) is 1310 pages
- All combinations can be created theoretically

Generation based fuzzing with Peach Fuzzer

- Peach fuzzer defines Peach pit files for the different file formats



```
<!-- Defines the common wave chunk -->
<DataModel name="Chunk">
    <String name="ID" length="4" padCharacter=" " />
    <Number name="Size" size="32" >
        <Relation type="size" of="Data" />
    </Number>
    <Blob name="Data" />
    <Padding alignment="16" />
</DataModel>

<DataModel name="ChunkData" ref="Chunk">
    <String name="ID" value="data" token="true"/>
</DataModel>

<DataModel name="ChunkFact" ref="Chunk">
    <String name="ID" value="fact" token="true"/>
    <Block name="Data">
        <Number size="32" />
        <Blob/>
    </Block>
</DataModel>
```

Public Peach pit files:

http://community.peachfuzzer.com/v2/PublicPits.html

End of lecture