FETA
_Fuzzer Evaluation Through Automation_

Michael Rodler
Sebastian Surminski

Prof. Dr.-Ing. Lucas Davi
Juniorprofessur für Informatik
https://www.syssec.wiwi.uni-due.de/
Complexity of Software Systems

Large attack surface for exploits
From Bug to Vulnerability

• C/C++ are plagued by memory safety violations (Bugs)
  • Missing bounds-checking (spatial)
  • Object lifetime violations (temporal)

• Memory Error – Memory Corruption – Arbitrary Code Execution
  • Often memory errors can be abused to corrupt memory
  • Code-pointers, data-pointers etc.
  • Hijack program execution – Return-Oriented Programming (ROP)
Example: Buffer Overflow

```c
char* input = get_input();
char buffer[128];
int len = *((int*)input);
memcpy(buffer, input + sizeof(len), len);
```

Stack diagram:

```
[0] [1] ... [128] char* input ... return address
```

- **Fixed Buffer Length!**
- **Attacker controlled length!**

Buffer Overflow

Hijack Program Execution
Vulnerability Discovery

White Hat Hacker → Input → Program → Crash

How to find the Inputs that trigger Bugs?
Fuzzing

Brute-Force Vulnerability Discovery
Basic Fuzzer Workflow

- Input
- Mutate Input
- Run Program
- Crash?
- Report Crash
- Exploitable?

Flowchart:
- Input
  - 89 50 4e 47
  - ff 0a 1a 0a
  - 00 00 00 0d
  - 49 48 44 52
  - 00 00 02 9a
  - 00 00 42 cd
  - 08 00 00 00
  - 00 3d 73 8f

- Mutate Input
- Run Program
- Crash?
  - No
  - Yes
- Report Crash
- Exploitable?
Fuzzing – The Good, The Bad, ...

+ Simple
+ Scalable / Parallel
+ Good Results in Practice

- Simple == dumb?
- Resource-hungry
- Manual optimizations
The Ugly

Program

```c
int magic = input();
if (magic == 0xdeadbeef) {
    vulnerability();
}
```

• How long does it take?
• It is not guaranteed to succeed
• Probabilistic process

Inputs

```
ff 42 43 ff
ff 42 4b ff
ff 42 ff 4b
ff 42 bf 4b
...  
```

Mutation

- Bit-flip
- Swap Bytes
- Bit-flip
- ???

... The Ugly
Research in Optimizing Fuzzing

Coverage-based Greybox Fuzzing

Directed Greybox Fuzzing

Angora: Efficient Fuzzing by Principled Search

Driller: Augmenting Fuzzing Through Selective Symbolic Execution

VUzzer: Application-aware Evolutionary Fuzzing

T-Fuzz: fuzzing by program transformation

Hui Peng
Purdue University
peng134@purdue.edu

Yan Shoshitaishvili
Arizona State University
yan@asu.edu

Mathias Payer
Purdue University
mathias.payer@nebelwelt.net

Vusec / vuzzer
No description, website, or topics provided.
Optimized Fuzzing

Program

```c
uint32_t magic = input();
if (magic == 0xdeadbeef) {
    vulnerability();
}
```

Program Transformation

```c
uint8_t magic[4] = input();
if (magic[0] == 0xde) {
    if (magic[1] == 0xad) {
        if (magic[2] == 0xbe) {
            if (magic[3] == 0xef) {
                vulnerability();
            }
        }
    }
}
```

Inputs

- `ff 42 43 ff`
- `de 42 4b ff`
- `de 42 4b bf`
- `de ad 4b bf`
- `de ad be ef`

Mutation

- Try all possible bytes at 0
- Try all possible bytes at 1
- Fix byte 0
- Fix byte 1
- Bit-flip

New Program Path discovered!

New Path

Reaching this is feasible now. In fact it is now even fast.
Research Question:

Evaluation and Comparison of State-of-the-Art Fuzzing Approaches
Goals and Features

1. Automated Setup of Fuzzing Instances
2. Sharing of Generated Test-Cases
3. Comparison and Evaluation of Fuzzers
4. Combine all Fuzzers against one Target
5. Report Bugs and Write PoC Exploits
PG Outline

- Lectures
- Seminar
- Implementation
- Report
Lectures – The Basics

• Program Exploitation: „From bugs to vulnerabilities to exploits“
  • Types of vulnerabilities (e.g. buffer overflow, use-after-free)
  • Practical exploit development

• Memory Error Detection
  • Hardening and Debugging Tools (AddressSanitizer, valgrind, etc.)
  • Root-cause analysis

• Basics of Program Analysis and Transformation
  • E.g. symbolic/concolic execution
  • Useful for understanding some optimized Fuzzers
Seminar Topics – The Fuzzers

• Summarize relevant literature
  • Presentation
  • Related work in final report

• Possible Topics:
  • Improved Fuzzing Search Strategies
    [Chen and Chen, IEEE S&P 2018], [Böhme et al., ACM SIGSAC 2016]
  • Improving Fuzzing with Static Program Analysis
    [Rawar et al., NDSS 2017], [Peng et al., IEEE S&P 2018]
  • Hybrid Fuzzing and Symbolic Execution
    [Stephens et al., NDSS 2016], [Yun et al., USENIX Security 2018]
  • Fuzzing Embedded Systems
    [Chen et al., NDSS 2018], [Muench et al., NDSS 2018]
Implementation

• Project Management
  • Issues and Milestones
  • Everything in git

• Implementation
  • Choose automation framework
  • Choose/Create Data-Sets for Evaluation
  • Setup and automate Experiments
  • Analyse results & write PoC exploits
Documentation

• **Goal:** *Publishable* project report!
  • Continuous writing and feedback loop

• Related work
  • i.e. the Fuzzers explained
  • Comparison of approaches

• Description of Experiments
  • Data-sets
  • Statistics

• PoC Exploits
### American Fuzzy Top 0.47b (readpng)

<table>
<thead>
<tr>
<th>Process Timing</th>
<th>Overall Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Time: 0 days, 0 hrs, 0 min. 43 sec</td>
<td>Cycles done: 131</td>
</tr>
<tr>
<td>Last hitters: 0 days, 0 hrs, 0 min. 26 sec</td>
<td>Uniq path: 0</td>
</tr>
<tr>
<td>Last unique hang: 0 days, 0 hrs, 1 min. 53 sec</td>
<td>Uniq hangs: 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cycle Progress</th>
<th>Map Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>New process: 0</td>
<td>Map density: 1317 (7.4%)</td>
</tr>
<tr>
<td>Last process: 0</td>
<td>Total cycles: 1317</td>
</tr>
<tr>
<td>Last unique process: 0</td>
<td>Unique cycles: 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stage Progress</th>
<th>Favored Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>New executions: 0</td>
<td>Found new edges: 138</td>
</tr>
<tr>
<td>Last executions: 0</td>
<td>Total crashes: 138</td>
</tr>
<tr>
<td>Last unique executions: 0</td>
<td>Total hangs: 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mutating Strategy</th>
<th>Fuzzing Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit flips: 0/100k</td>
<td>Levels:</td>
</tr>
<tr>
<td>Byte flips: 0/100k</td>
<td>Pending: 0</td>
</tr>
<tr>
<td>Arithmetics: 31/1344</td>
<td>Pended fav: 0.14</td>
</tr>
<tr>
<td>Known ints: 12/11384</td>
<td>Reported: 0</td>
</tr>
<tr>
<td>Known floats: 0</td>
<td>Variable: 0</td>
</tr>
<tr>
<td>Hwacc: 3/2345</td>
<td>Latent: 0</td>
</tr>
<tr>
<td>Trim: 20/573</td>
<td></td>
</tr>
</tbody>
</table>

---

**Questions?**
References


• H. Peng, Y. Shoshtarishvili, and M. Payer, “T-Fuzz: fuzzing by program transformation.”

• B. Shastry et al., “Static Program Analysis as a Fuzzing Aid,” in Research in Attacks, Intrusions, and Defenses, 2017, pp. 26–47.


