Week 15A: Compiler Fuzzing & The Future of Fuzzing

Stefan Nagy
University of Utah
## Reminders

### Part 4: New Frontiers in Fuzzing

<table>
<thead>
<tr>
<th>Tuesday Meeting</th>
<th>Thursday Meeting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nov. 15</td>
<td>Nov. 17</td>
</tr>
<tr>
<td><strong>Fuzzing Science</strong> (<a href="#">slides</a>)</td>
<td><strong>Fun Targets: Hardware</strong></td>
</tr>
<tr>
<td>• Reading: <a href="#">FIXREVERTER: A Realistic Bug Injection Methodology for Benchmarking Fuzz Testing</a></td>
<td>• Reading: <a href="#">Fuzzing Hardware Like Software</a></td>
</tr>
<tr>
<td>Nov. 22</td>
<td>Nov. 24</td>
</tr>
<tr>
<td>No Class (Thanksgiving break)</td>
<td>No Class (Thanksgiving break)</td>
</tr>
<tr>
<td>Nov. 29</td>
<td>Dec. 1</td>
</tr>
<tr>
<td><strong>Compiler Fuzzing &amp; The Future of Fuzzing</strong></td>
<td><strong>Guest lecture by Dr. Christophe Hauser</strong></td>
</tr>
<tr>
<td>• Reading: <a href="#">Finding and Understanding Bugs in C Compilers</a></td>
<td></td>
</tr>
<tr>
<td>Dec. 6</td>
<td>Dec. 8</td>
</tr>
<tr>
<td><strong>Team Project presentations</strong></td>
<td><strong>Team Project presentations</strong></td>
</tr>
</tbody>
</table>
Reminders

Part 4: New Frontiers in Fuzzing

<table>
<thead>
<tr>
<th>Tuesday Meeting</th>
<th>Thursday Meeting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nov. 15</td>
<td>Nov. 17</td>
</tr>
<tr>
<td><strong>Fuzzing Science</strong> <em>(slides)</em></td>
<td><strong>Fun Targets: Hardware</strong></td>
</tr>
<tr>
<td>• Reading: <em>FIXREVERTER: A Realistic Bug Injection Methodology for Benchmarking Fuzz Testing</em></td>
<td>• Reading: <em>Fuzzing Hardware Like Software</em></td>
</tr>
<tr>
<td>Nov. 22</td>
<td>Nov. 24</td>
</tr>
<tr>
<td>No Class (Thanksgiving break)</td>
<td>No Class (Thanksgiving break)</td>
</tr>
<tr>
<td>Nov. 29</td>
<td></td>
</tr>
<tr>
<td><strong>Compiler Fuzzing &amp; The Future of Fuzzing</strong></td>
<td></td>
</tr>
<tr>
<td>• Reading: <em>Finding and Understanding Bugs in C Compilers</em></td>
<td></td>
</tr>
<tr>
<td>Dec. 1</td>
<td></td>
</tr>
<tr>
<td><strong>Guest lecture by Dr. Christophe Hauser</strong></td>
<td></td>
</tr>
<tr>
<td>Dec. 6</td>
<td></td>
</tr>
<tr>
<td><strong>Team Project presentations</strong></td>
<td><strong>Team Project presentations</strong></td>
</tr>
</tbody>
</table>
Thursday’s Guest Lecture

- Dr. Christophe Hauser
  - Research Scientist & Research Lead
    - University of Southern California
    - bass.isi.edu/members/chris/
  - Expert on binary analysis and software security
    - Core developer of Angr
  - Core member of the famed CTF team Shellphish
    - One of the top academic hacking teams
    - Frequent qualifiers of DEFCON CTF
      - The SuperBowl of hacking
Friday: CTF Q&A

- Interested in competitive hacking?
  - Come ask Christophe your questions!
    - Time: 9–11AM
    - Date: Friday, December 2nd
    - Room: 3147 MEB
  - Fill out the UCC interest form (so we have your email)
    - softsec.cs.utah.edu/cyber
Questions?
# Reminders

## Part 4: New Frontiers in Fuzzing

<table>
<thead>
<tr>
<th>Tuesday Meeting</th>
<th>Thursday Meeting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nov. 15</td>
<td>Nov. 17</td>
</tr>
<tr>
<td><strong>Fuzzing Science</strong> <em>(slides)</em></td>
<td><strong>Fun Targets: Hardware</strong></td>
</tr>
<tr>
<td>• Reading: <a href="#">FIXREVERTER: A Realistic Bug Injection Methodology for Benchmarking Fuzz Testing</a></td>
<td>• Reading: <a href="#">Fuzzing Hardware Like Software</a></td>
</tr>
<tr>
<td>Nov. 22</td>
<td>Nov. 24</td>
</tr>
<tr>
<td>No Class (Thanksgiving break)</td>
<td>No Class (Thanksgiving break)</td>
</tr>
<tr>
<td>Nov. 29</td>
<td>Dec. 1</td>
</tr>
<tr>
<td><strong>Compiler Fuzzing &amp; The Future of Fuzzing</strong></td>
<td><strong>Guest lecture by Dr. Christophe Hauser</strong></td>
</tr>
<tr>
<td>• Reading: <a href="#">Finding and Understanding Bugs in C Compilers</a></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dec. 6</th>
<th>Dec. 8</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Team Project presentations</strong></td>
<td><strong>Team Project presentations</strong></td>
</tr>
</tbody>
</table>
Last week of class: Final Project Presentations

- **Project deliverables:**
  - 1–3 page report detailing your work and developed tools
  - 15–20 minute class presentation (see schedule below)
  - Both must explain what you did, why, and your results
  - Submit materials on Canvas by 11:59PM Thursday, December 8th
    - Choose one team member to submit report and talk slides
    - Merge talk slides and report into a single submission PDF
Last week of class: Final Project Presentations

- **Project deliverables:**
  - 1–3 page report detailing your work and developed tools
  - 15–20 minute class presentation *(see schedule below)*
  - Both must explain what you did, why, and your results
  - Submit materials on Canvas by **11:59PM Thursday, December 8th**
    - Choose one team member to submit report and talk slides
    - Merge talk slides and report into a single submission PDF

- **Presentation Schedule:**
  - **Tuesday 12/06:** Brian, Colin, Luciano, Yuvaraj
  - **Thursday 12/08:** Email fuzzing team, TA Portal team
    - Course wrap-up and breakfast party
Questions?
Fuzzing Compilers
What is a compiler?

- **Compiled languages**: C, C++, Rust, Go, Haxe, ...
What is a compiler?

- **Compiled languages:** C, C++, Rust, Go, Haxe, ...

```c
#include<stdio.h>

int main()
{
    printf("Hello, World!\n");
    return 0;
}
```

```
hello_world.c
```

```
0110011001100100010011000111
11000000011111111110000001
11110001101010100011000111
00110010001011000111101100
00000111111111111000001111
10001101010011000110100111
10001000101100111111100000
00111111111110000011111110
01101010100011001101001100
01000100110001111011000000
11111111000000111110011111
```

```
hello_world.o
```
What is a compiler?

- **Compiled languages**: C, C++, Rust, Go, Haxe, ...

```
#include<stdio.h>
int main()
{
    printf("Hello, World!\n");
    return 0;
}
```

![Diagram](image)
What is a compiler?

- **Compiled languages:** C, C++, Rust, Go, Haxe, ...

```c
#include<stdio.h>
int main()
{
    printf("Hello, World!\n");
    return 0;
}
```

Is the compiled program correct?
Compilers need love (fuzzing) too!

- Compilers are complex beasts with lots of moving parts
Compilers need love (fuzzing) too!

- Compilers are complex beasts with lots of moving parts
Compilers need love (fuzzing) too!

- Compilers are complex beasts with lots of moving parts
Compilers need love (fuzzing) too!

- Compilers are complex beasts with lots of moving parts

Security flaws caused by compiler optimizations
Challenges of compiler fuzzing

- **Main problem:** need complex, well-formed inputs
  - Sensitive to naive insertions and deletions
    - AFL-style random fuzzing won’t do well
  - Need specialized input generation
  - Must cover broad range of code semantics

- **Other problems:**
  - How to determine code correctness?
    - Differential testing, formal verification
  - How to utilize code coverage?
    - Does not tell the whole story
The Future of Fuzzing
Expanding fuzzing’s reach: new targets

- Need **faster and sound instrumentation** for fuzzing binary-only targets
Expanding fuzzing’s reach: new targets

- Need faster and sound instrumentation for fuzzing binary-only targets
Expanding fuzzing’s reach: new targets

- Need **faster and sound instrumentation** for fuzzing binary-only targets
- Need **automatic harnessing** that doesn’t require API usage examples
Expanding fuzzing’s reach: new bug types

- Feedback & oracles for **hard-to-fuzz bugs** (e.g., concurrency, side channels)
Expanding fuzzing’s reach: new bug types

- Feedback & oracles for hard-to-fuzz bugs (e.g., concurrency, side channels)
- Techniques for vetting the completeness of vulnerability fixes

<table>
<thead>
<tr>
<th>Product</th>
<th>Vulnerability exploited in-the-wild</th>
<th>Variant of...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mozilla Firefox</td>
<td>CVE-2020-6820</td>
<td>Mozilla Bug 1507180</td>
</tr>
<tr>
<td>Google Chrome</td>
<td>CVE-2020-6572</td>
<td>CVE-2019-5870 CVE-2019-13695</td>
</tr>
<tr>
<td>Microsoft Windows</td>
<td>CVE-2020-0986</td>
<td>CVE-2019-0880*</td>
</tr>
<tr>
<td>Google Chrome/Freetype</td>
<td>CVE-2020-15999</td>
<td>CVE-2014-9665</td>
</tr>
<tr>
<td>Apple Safari</td>
<td>CVE-2020-27930</td>
<td>CVE-2015-0093</td>
</tr>
</tbody>
</table>

* vulnerability was also exploited in-the-wild in previous years
Making fuzzing user-friendly

- Find why code and bugs missed by fuzzers and harnesses
Making fuzzing user-friendly

- Find why **code and bugs missed** by fuzzers and harnesses
- Techniques to know **when to stop fuzzing** (or switch things up)
Better automatic error analysis

- CVE scoring is incredibly subjective; requires lots of domain expertise
Better automatic error analysis

- CVE scoring is incredibly subjective; requires lots of domain expertise
- Techniques to **automatically classify critical bugs** vs. non-critical ones
Other interesting directions

What do you think is next?