Week 11B: Directed Fuzzing II

Stefan Nagy
University of Utah
How are semester projects going?

Problems with your tools? Successfully up and running?
Announcements

- **Coming in Spring 2023...**
  - **The Utah Cybersecurity Club**
    - [https://softsec.cs.utah.edu/cyber](https://softsec.cs.utah.edu/cyber)
    - Fill out our recruitment survey!
      - Bonus points if you solve the hidden CTF challenge ;)

SCHOOL OF COMPUTING
UNIVERSITY OF UTAH
Questions?
Directed Fuzzing Recap
Recap: Targeted Testing

- **Regression testing**
  - Did my PR break the software?

- **Patch testing**
  - Have I actually fixed this vulnerability?

- **Crash reproduction**
  - Is this random person’s bug report valid?
Recap: “Directed” Fuzzing

- Guided fuzzing steered to **specific locations**
  - E.g., Patch-changed code lines
  - E.g., An ASAN-reported crash line

- **Key differences versus guided fuzzing:**
  - **Instrumentation:**
    - Track **distance** relative to targeted site(s)
    - Compute this for **every** generated test case
  - **Seed selection:**
    - Pick inputs that get you **closer** to target(s)
    - Progress stalls? Pick a new input and restart

```
1   if (input < 100)         2
2       f(0);                1
3
4   if (input > 100)         3
5       if (input > 200)     2
6       f(input)             1
7
8   void f(int x) {
9       if (x == 999)       1
10 // target              0
11   }
```

Source: KATCH: High-Coverage Testing of Software Patches
Bug-tailored Directed Fuzzing
Fuzzing multiple code locations

- Sometimes must fuzz **multiple targets**
  - E.g., patch-changed source lines
  - E.g., reproducing specific bugs

- General-purpose directed fuzzing
  - Distances relative to these sites
  - **No ranking or sequential order**
    - Tries to reach all sites at once

```c
#include<stdio.h>

-int main(void)
   
   printf("Hello, world!\n");
   
   return 0;
```

-1,5 +1,6
“Spatial” Memory Safety

- **Spatial** = relating to **occupying space**

- **Spatial memory safety** violations
  - Buffer overflows
  - Heap overflows
  - Underflows
  - Invalid reads/writes
  - Uninitialized data
  - ...

- Directed fuzzing on **limited target set**
“Temporal” Memory Safety

- Temporal = relates to time

- Temporal memory safety violations
  - Dangling pointers
    - Heap use-after-free (UAF)
    - Double free (DF)

- Requires a sequence of events
  - Thus, must fuzz multiple targets in order
Today’s focus: Use-After-Frees (UAFs)

- **Over one third** of Chromium vulnerabilities

![Pie chart showing the distribution of Chromium vulnerabilities.](https://www.chromium.org/Home/chromium-security/memory-safety/)

Source: https://www.chromium.org/Home/chromium-security/memory-safety/
A (crash) course on UAFs

- **The Heap** = *dynamically*-allocated memory
  - Allocated via `malloc()`, and freed via `free()`
  - Chunks may get allocated, freed, split, coalesced
  - Regions accessed via **pointers**

- **Management is programmer’s job**
  - Pointers must point to **live objects**
  - Must point to objects of the **right type**
  - Only pointers to **functions** can be executed
  - ...

![Diagram](image-url)
A (crash) course on UAFs

Are use-after-frees exploitable?
- Overwrite a free’d chunk
  - Leak information
  - Redirect execution
  - Type confusion
  - Other evil things

- Short answer: very much so!
Fuzzing for UAFs

What call sequence is required for a UAF?

- An object allocation (e.g., malloc())
- A free() of that same object
- A use (dereference) of that same object
  - E.g., calling a function pointer
Directed Fuzzing for UAFs

- What call sequence is required for a UAF?
  - An object allocation (e.g., malloc())
  - A free() of that same object
  - A use (dereference) of that same object
    - E.g., calling a function pointer

- Pick inputs that match this call sequence
  - Mine their locations statically
  - Pick inputs that hit them in order

Making directed fuzzing sequence-aware

- **AFL-Go**: biases exploration toward single target func E
  - No sequential ordering

- For UAFs, must bias toward hitting correct sequence
Making directed fuzzing sequence-aware

- **Solution:** weight the edges between allocs, uses, frees
  - Small weights = more priority
  - Bias the fuzzer to move from one state to the other
Making directed fuzzing sequence-aware

- **Solution:** weight the edges between allocs, uses, frees
  - Small weights = more priority
  - Bias the fuzzer to move from one state to the other

- **What about double frees?**
  - Just hit a second `free()`
Results

- **UAFuzz**: binary-level fuzzer for use-after-frees

Results

- **UAFuzz**: binary-level fuzzer for use-after-frees

<table>
<thead>
<tr>
<th>Program</th>
<th>Code Size</th>
<th>Version (Commit)</th>
<th>Bug ID</th>
<th>Vulnerability Type</th>
<th>Crash</th>
<th>Vulnerable Function</th>
<th>Status</th>
<th>CVE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7.1</td>
<td>(987169b)</td>
<td>#1269</td>
<td>User after free</td>
<td>×</td>
<td>gf_m2ts_process_pmt</td>
<td>Fixed</td>
<td>CVE-2019-20628</td>
<td></td>
</tr>
<tr>
<td>0.8.0</td>
<td>(56eaea8)</td>
<td>#1440-1</td>
<td>User after free</td>
<td>×</td>
<td>gf_isom_box_del</td>
<td>Fixed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.8.0</td>
<td>(56eaea8)</td>
<td>#1440-2</td>
<td>User after free</td>
<td>×</td>
<td>gf_isom_box_del</td>
<td>Fixed</td>
<td>CVE-2020-11558</td>
<td></td>
</tr>
<tr>
<td>0.8.0</td>
<td>(56eaea8)</td>
<td>#1440-3</td>
<td>User after free</td>
<td>×</td>
<td>gf_isom_box_del</td>
<td>Fixed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(5b37b21)</td>
<td>#1427</td>
<td>User after free</td>
<td>✓</td>
<td>fz_drop_band_writer</td>
<td>Fixed</td>
<td>CVE-2020-16600</td>
<td></td>
</tr>
</tbody>
</table>

- **MuPDF**: 539K

<table>
<thead>
<tr>
<th>Program</th>
<th>Code Size</th>
<th>Version (Commit)</th>
<th>Bug ID</th>
<th>Vulnerability Type</th>
<th>Crash</th>
<th>Vulnerable Function</th>
<th>Status</th>
<th>CVE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.31.3</td>
<td>(a3c7756)</td>
<td>#134324</td>
<td>User after free</td>
<td>✓</td>
<td>S_reg</td>
<td>Confirmed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.31.3</td>
<td>(a3c7756)</td>
<td>#134326</td>
<td>Use after free</td>
<td>✓</td>
<td>Perl_regnext</td>
<td>Fixed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.31.3</td>
<td>(a3c7756)</td>
<td>#134329</td>
<td>User after free</td>
<td>✓</td>
<td>Perl_regnext</td>
<td>Fixed</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **readelf**: 1.0 M

<table>
<thead>
<tr>
<th>Program</th>
<th>Code Size</th>
<th>Version (Commit)</th>
<th>Bug ID</th>
<th>Vulnerability Type</th>
<th>Crash</th>
<th>Vulnerable Function</th>
<th>Status</th>
<th>CVE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.34</td>
<td>(f717994)</td>
<td>#25821</td>
<td>Double free</td>
<td>✓</td>
<td>process_symbol_table</td>
<td>Fixed</td>
<td>CVE-2020-16590</td>
<td></td>
</tr>
</tbody>
</table>

- **nm-new**: 6.7 M

<table>
<thead>
<tr>
<th>Program</th>
<th>Code Size</th>
<th>Version (Commit)</th>
<th>Bug ID</th>
<th>Vulnerability Type</th>
<th>Crash</th>
<th>Vulnerable Function</th>
<th>Status</th>
<th>CVE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.34</td>
<td>(c98a454)</td>
<td>#25823</td>
<td>Use after free</td>
<td>✓</td>
<td>bfd_hash_lookup</td>
<td>Fixed</td>
<td>CVE-2020-16592</td>
<td></td>
</tr>
</tbody>
</table>

Discovered **many new dangling pointer** vulnerabilities

Trade-offs

- **The more program introspection, the better**
  - Open-source is always easier than closed-source
    - Likely won’t scale to many closed-source targets
    - E.g., Microsoft Word
  - **Static analysis becomes very costly**
    - Target identification
    - Distance computation

- **Can this be extended to other bug types?**
  - Yes... if it can be expressed as a temporal ordering
    - E.g., heap overflows (allocation + access)
    - Others? *(open research problem)*
Questions?