Evocatio: Conjuring Bug Capabilities from a Single PoC

Zhiyuan Jiang, Flavio Toffalini, Manuel Egele, Shuitao Gan, Lucio Romerio, Chao Zhang, Adrian Herrera, Chaojing Tang, Mathias Payer
Motivation

Fuzzing finds 1000s of crashes

- How severe are the crashes?
- Which bug should be fixed first?

So far, the user has to inspect each crash manually
Severity Assessment

- Scoring bug severity is subjective
- Highly dependent on threat model

We want:

- Determine bug severity across multiple dimensions
- Calculate severity based on user-defined threat model
- Fully automatic and objective
Before assessing the bug severity

- What can the bug do?

(Bug_type, Acc_type, Acc_len, Buf_name, Off, Loc)

- on the stack
- starts at 10th bytes into buffer
- buffer
- 5 bytes
- read
- out of bounds
Capabilities of Bug A

Capabilities of Bug B

Threat model
Evocatio: Automatically Assessing Bug Capabilities

The Architecture of CAPFUZZ

- Critical Bytes Guided Mutation
  - Critical Bytes Prioritization
  - C-Bytes/D-Bytes Mutation Rule

- Test (Core Fuzzing)
  - Capability Detection
  - Seed Retention
  - Seed Selection

- Critical Byte Inference Engine
  - Single-byte inference
  - Capability Detection
  - Byte-sequence inference

Initial input (PoC)

PoCs with new Capabilities
I) Capability Detection: CapSan

- Extract capability of a PoC automatically
- Sensitive to capability changing
- Configurable monitor items
- Convenient and light-weight
II) Capability Discovery: Critical Bytes Inference

Assess impact for each input byte

- $C_{\text{byte}}$: affecting control flow
- $D_{\text{byte}}$: affecting data flow

❖ Single-byte inference
❖ Byte-sequence inference
III) CapFuzz: Capability guided Fuzzing

Goal: find more capabilities of a bug
Input: single crashing seed
Output: seeds with different capabilities

- Prioritize Critical Bytes
- Mutation
- Seed Retention
- Seed Selection
Severity Assessment

Example threat model

- **Goal:** achieve remote code execution
- Bug type
- Max. length of OOB reads/writes
- Readable/writable address ranges
- Num. of OOB objects
- Max. OOB size objects
- Num. of different read/write offsets
Evaluation

- 38 bugs (34 CVEs + 4 issues)
- One PoC for each bug
- 8 real-world programs
- 6 bug types
## Evaluation: Capabilities discovered by Evocatio

<table>
<thead>
<tr>
<th>CVE</th>
<th>Bug Effect</th>
<th>Size</th>
<th>Origin</th>
<th>Origin Size</th>
<th>Origin Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read</td>
<td>Write</td>
<td>Stack</td>
<td>Heap</td>
<td>Read</td>
</tr>
<tr>
<td>CVE-2018-7871</td>
<td>HOF[W</td>
<td>UAF</td>
<td>N]</td>
<td>$[2^0 \ldots 2^3]4$</td>
<td>$[2^2 \ldots 2^6]6$</td>
</tr>
<tr>
<td>CVE-2019-16705</td>
<td>HOF[W</td>
<td>UAF]</td>
<td>$[2^0 \ldots 2^0 \ldots 2^{10}]3$</td>
<td>$[2^5 \ldots 2^5]1$</td>
<td>0</td>
</tr>
<tr>
<td>CVE-2021-3156</td>
<td>HOF[-]</td>
<td>-</td>
<td>$[2^0 \ldots 2^{10}]694$</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

- ~50% in the same risk level, quantitative estimate of severity
- Out of 16 patched CVEs, 7 patches were incomplete (and bypassable)
Key takeaways

Fuzzing detects bugs, assessing their severity is hard

- Programmers are overwhelmed by too many reports
- Bug severity assessment must be automatic and objective
- Completely fixing a bug is hard based on a single PoC

Our findings

- Bug capabilities give developers context
- Guided fuzzing detects underlying bug capabilities
- Evocatio detected 7 incomplete patches, generating new capabilities
- [https://github.com/HexHive/Evocatio](https://github.com/HexHive/Evocatio)