Static Exploration of Taint-Style Vulnerabilities Found by Fuzzing

Bhargava Shastry, Federico Maggi, Fabian Yamaguchi, Konrad Rieck, and Jean-Pierre Seifert
How It Started

- Spun afl-fuzz on Open vSwitch
  - Found 8 vulnerabilities
  - Responsibly disclosed and now patched
  - 1 RCE
    - Crashing input tweetable

fffffffff00000000000008847
Bottleneck

- OvS has over 100 functional test cases
  - Only 3-4 fuzzable
  - Test coverage $\leq 3\%$

Duh, extensively write fuzzable test cases!
Problem

- Not faulting **OvS**, problem deep-rooted
- Writing fuzzable tests **challenging**
  - Applicability **limited**
  - Does not **scale**
  - Requires domain **expertise**

Fuzzing may not exercise every single LoC
Pitch

Fuzzer-directed static analysis
Proposal

Leverage hard data to ask the compiler specific questions

Fuzzer crash ⇒ Stack trace ⇒ Vulnerability Template ⇒ Recurrences
Implementation

- Fault localization & Ranking ⇒ custom python script
- Template matching engine ⇒ Clang libASTMatcher

https://github.com/test-pipeline
## Results: Effectiveness

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Num. matches</th>
<th>Num. issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2016-10377</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>CVE-2017-9264 (TCP)</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td><strong>CVE-2017-9264 (UDP)</strong></td>
<td><strong>2</strong></td>
<td><strong>1</strong></td>
</tr>
<tr>
<td>CVE-2017-9264 (IPv6)</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>CVE-2017-9214</td>
<td>41</td>
<td>0</td>
</tr>
<tr>
<td>CVE-2017-9263</td>
<td>34</td>
<td>0</td>
</tr>
<tr>
<td>CVE-2017-9265</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Ranking Matches

- Reports provides insufficient context
- We rank matches based on fuzzer coverage
- Matches containing uncovered code interesting

Only 36 out of 96 matches ranked high
Insight

Developers want contextual information

“I would like to hear about other similar problem(s) you find in the code. Whether they are exploitable or not, it is better for the code to be careful.”

– Ben Pfaff, OvS lead developer
Results: Run time

![Bar chart showing run time comparisons for different vulnerabilities.](chart.png)
Insight

● Structural (AST) analysis is relatively fast
● Semantic analysis is relatively slow
● Tension between analysis precision and speed
● Run time suitable for continuous integration
Summary

● Going beyond fuzzing is **necessary**
● **Static analysis** well-suited, results **promising**
● Evaluated on OvS, drew attention to **1 real issue** and several corner cases
● **Fast** enough for continuous integration
Future Work

● Reducing false positives
  ○ Formulating more precise vulnerability templates
● Easing manual review further
  ○ Use Angr for path reachability queries
  ○ Greetz to Dominic Maier
Acknowledgements

Thank OvS Security/Dev team for timely fixes
Questions?
Related Work

When vulnerable code pattern known

- **Code mining**
  - Rely on security patches ⇒ *Reactive*

When vulnerable code pattern unknown

- **Machine learning**
  - As good as training set ⇒ *Insufficient guarantees*