KLEE’s Sonar-Search
Reviewing in the context of Greybox Fuzzing

Saahil Ognawala, Alexander Pretschner, Thomas Hutzelmann,
Eirini Psallida, Ricardo Nales

Technical University of Munich, Germany
Agenda

Sonar-search

Compositional analysis

Greybox Fuzzing
Motivation

• Symbolic execution and fuzzing - insufficient coverage => vulnerabilities not discovered

• Idea - Analyze individual functions (C programs) for vulnerabilities

• Do reachability analysis for vulnerabilities using symbolic execution

• But what about path explosion?
Targeted symbolic execution

- To tackle path-explosion
- Set targets (function entry points or vulnerable instructions in functions)
- Terminate states that do not reach targets-of-interest
Sonar-search

- Implementation of targeted-search in KLEE — *KLEE22*
- Target may be *function-call, function-return, LLVM bb* or *klee_assert*
- Nested searcher
- *nurs: covnew* when target is reached
Sonar-search

void mid() {
    // <- position of the current execution state
    return;
}

void foo() {
    mid();
    // <- target
}

void bar() {
    mid();
}

void entry(int a) {
    if (a > 0) // target reachable
        foo();
    else // target unreachable
        bar();
}
void mid() {
    // <- position of the current execution state
    return;
}

void foo() {
    mid();
    // <- target
}

void bar() {
    mid();
}

void entry(int a) {
    if (a > 0) // target reachable
        foo();
    else // target unreachable
        bar();
}
Sonar-search

```c
void mid() {
    // <- position of the current execution state
    return;
}

void foo() {
    mid();
    // <- target
}

void bar() {
    mid();
}

void entry(int a) {
    if (a > 0) // target reachable
        foo();
    else // target unreachable
        bar();
}
```
Sonar-search

```c
void mid() {
    // <- position of the current execution state
    return;
}

void foo() {
    mid();
    // <- target
}

void bar() {
    mid();
}

void entry(int a) {
    if (a > 0) // target reachable
        foo();
    else // target unreachable
        bar();
}
```
Compositional symbolic execution with *Macke*
Real-life programs too big

Partial call-graph of Grep 2.25
Symbolic execution of isolated functions

Partial call-graph of Grep 2.25
Symbolic execution of isolated functions

Partial call-graph of Grep 2.25

Vulnerability here
Summarization of vulnerability as PC (or PC solution)

Partial call-graph of Grep 2.25
Sonar-search to vulnerable functions
## Macke - Results

<table>
<thead>
<tr>
<th>Program</th>
<th>MACKE only</th>
<th>both</th>
<th>KLEE only</th>
</tr>
</thead>
<tbody>
<tr>
<td>bc 1.06</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bison 3.0.4</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bzip2 1.0.6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>coreutils 6.10</td>
<td>24</td>
<td>28</td>
<td>0</td>
</tr>
<tr>
<td>coreutils 8.25</td>
<td>34</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>diff 3.4</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>flex 2.6.0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>flex SIR</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>goahead 3.6.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>grep 2.25</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>grep SIR</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>jq 1.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>less 481</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>lz4 r131</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ngircd 23</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sed 4.2.2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>tar 1.29</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>zopfli 1.0.1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

in total 67 41 0
Increasing function coverage with *Munch*
Symbolic execution vs. Fuzzing

• Path explosion and constraint solver in symbolic execution -> Low path coverage in “deep” parts of program
• Weak input (mutation) strategy with fuzzing -> Low path coverage everywhere.
• Idea: Greybox fuzzing (symbolic execution + fuzzing)
Munch - Greybox Fuzzer for Function Coverage

**FS hybrid**
- Fuzzing for limited time
- List uncovered functions
- *Sonar-search* to uncovered functions

**SF hybrid**
- Symbolic execution (KLEE default strategy) for limited time
- Fuzzing with seed inputs generated by KLEE
Munch - Results

- 9 C-programs (⊂ Macke)

<table>
<thead>
<tr>
<th>Technique</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Munch FS</td>
<td>~ 2 hours</td>
</tr>
<tr>
<td>Munch SF</td>
<td>2 hours</td>
</tr>
<tr>
<td>Symbolic execution</td>
<td>5 hours</td>
</tr>
<tr>
<td>Fuzzing</td>
<td>5 hours</td>
</tr>
</tbody>
</table>
Munch - Results

Average function coverage (%)
Munch - Results

![Graph showing function coverage vs. call-graph depth]

- Symex
- Fuzzing
- Munch SF
- Munch FS

Function coverage (%) vs. Call-graph depth
Conclusion

• Sonar-search (KLEE22) for targeted symbolic execution
• Two use-cases of sonar-search
  • Compositional analysis of C programs
  • Greybox fuzzing to increase function coverage
• Compositional analysis performs better in terms of vulnerability discovery than symbolic execution
• Greybox fuzzing performs better in terms of function coverage, than symbolic execution or blackbox fuzzing.
References

  • https://github.com/tum-i22/macke

  • https://github.com/tum-i22/munch

• KLEE22 with Sonar-search - https://github.com/tum-i22/klee22 (branch “sonar”)

• Severity assessment tool - https://vmpretschner18.informatik.tu-muenchen.de/

• ognawala@in.tum.de