Optimizing Seed Selection for Fuzzing

Fuzzing = Bug Finding

Program Parameters

Fuzzer

Bugs
Optimizing **Seed** Selection for Fuzzing

BFF, FileFuzz, jsfunfuzz, Peach, Sage, ZZUF and many more ...
Seed = Well-Structured Input
Seed Selection Challenge

Given:
- Program
- Fuzzer
- Time limit $T$
Seed Selection Challenge

You can run the fuzzer with any seed for any arbitrary time period (total time $\leq T$)

Given:
- Program
- Fuzzer
- Time limit $T$

PDF File

Fuzzer

Bugs

Goal: find as many bugs as possible
Research Questions

Universe of PDF Files

PDF File

Fuzzer

Bugs
Research Questions

Universe of PDF Files

Set of PDF Files

Question #1
How to select seeds to use?

PDF File

Fuzzer

Bugs
Question #2
How to schedule seeds?
Can we obtain the maximum
# of bugs that can be found
for a given set of seeds?

#bugs found =
#unique crashes
identified by stackhash
Q1: Seed Selection

Universe of PDF Files

Set of PDF Files

PDF File

Fuzzer

Question #1
How to select seeds to use?

Bugs
Find a Set of Seeds
Maximizing Code Coverage

- Miller reports an 1% increase in code coverage increases the percentage of bugs found by 0.92%\(^1\)
- Peach uses code coverage to select seeds\(^2\)

---

\(^1\) Fuzz by Number, CanSecWest 2008
\(^2\) [http://peachfuzzer.com](http://peachfuzzer.com)
Minimal Set-Cover Problem (MSCP)
Minimal Set-Cover Problem (MSCP)

- $S_1 = \{2\}$
- $S_2 = \{1\}$
- $S_3 = \{3,4\}$
- $S_4 = \{1,2\}$
Minimal Set-Cover Problem (MSCP)

- $S_1$: $\{2\}$
- $S_2$: $\{1\}$
- $S_3$: $\{3,4\}$
- $S_4$: $\{1,2\}$
MSCP is NP-Hard, But

We use a greedy polynomial-time approximation algorithm

- **Unweighted MinSet**: MSCP
- **Time MinSet**: Weighted MSCP with exec. time
- **Size MinSet**: Weighted MSCP with seed file size
- **Peach Set**: derived from peach fuzzier

More details in the paper
Comparing Seed Selection Algorithms

Universe of PDF Files

Set of PDF Files

PDF File

Fuzzer

Randomly selected 100 seeds per file extension

4,912,142 seed files (≈ 6TB) (274 file extensions)

We do this for every applications (10 apps.)
# Seeds after Seed Selection
(From 100 Seeds)
Q2: Optimal Seed Scheduling

For a given set of seeds, what is the maximum # of bugs that can be found within a time limit?

We introduce a methodology of evaluating seed selection algorithms.
Compute Optimal Scheduling from Collected Ground Truth Data

Per-Seed Ground Truth Collection

Ground Truth = a sequence of

\[(\text{bug ID}, \text{seed ID}, \text{time stamp})\]

\[(B_1, S_1, T_1), (B_2, S_1, T_2), \ldots\]
Compute Optimal Scheduling from Collected Ground Truth Data

For all the seeds in the universe:

\[ (B_1, S_1, T_1), (B_2, S_1, T_2), \ldots \]
\[ (B_4, S_2, T_1), (B_2, S_3, T_2), \ldots \]
\[ \ldots \]
\[ (B_4, S_2, T_1), (B_2, S_3, T_2), \ldots \]

Finding an optimal scheduling is NP-hard

\[ \Rightarrow ILP \text{ (Integer Linear Programming)} \]
ILP Formulation Example

- Fuzzing 1 program with 2 seed files ($S_1$ and $S_2$)
- 1 minute fuzzing run with each seed
- 2 bugs found in total ($B_1$ and $B_2$)
Steps in ILP Formulation

1. Define the goal
   - Maximize the # of Bugs
2. Define ILP variables
3. Define constraints over the variables
Introducing Crash Indicator Variable $c_{i,j}$

If we select $S_1$ for 15 sec., then $c_{1,1} = 0$, $c_{1,2} = 0$

If we select $S_2$ for 40 sec., then $c_{2,1} = 1$, $c_{2,2} = 0$

$c_{i,j} = j^{th}$ crash in the $i^{th}$ seed
Introducing Time Variable $t_{i,j}$

$t_{1,1}$

$S_1 \rightarrow (B_1, S_1, 30) \rightarrow (B_1, S_1, 60)$

$t_{1,2}$

$c_{1,1}$

$c_{1,2}$

30 sec.

30 sec.

$c_{2,1}$

$c_{2,2}$

$S_2 \rightarrow (B_2, S_2, 15) \rightarrow (B_1, S_2, 55)$

$t_{2,1}$

$t_{2,2}$

15 sec.

40 sec.

$t_{i,j} = j^{th}$ time interval of the $i^{th}$ seed
Introducing Bug Indicator Variable $b_x$

If we select $S_2$ for 40 sec., $b_2 = 1$
Constraint 1: Order of Crashes

\[ \forall i,j \cdot c_{i,j+1} \leq c_{i,j} \]

Preserve the order of crashes
Constraint 2: Time Limit

\[ \sum_{i,j} c_{i,j} \cdot t_{i,j} \leq t_{\text{thres}} \]

Do not exceed the time limit
Constraint 3: Crash $\Rightarrow$ Bug

If a crash is found, then the corresponding bug is found.

\[ \forall_{i,j} c_{i,j} \leq b_x \text{ where } \mu(c_{i,j}) = x \]
Constraint 4: Bug $\Rightarrow$ Crash

If a bug is found, then one of the corresponding crashes is found.
Final ILP Formulation

maximize \[ \sum_{x} b_x \]  

subject to

- \[ \forall i,j . \quad c_{i,j+1} \leq c_{i,j} \]
- \[ \sum_{i,j} c_{i,j} \cdot t_{i,j} \leq t_{\text{thres}} \]
- \[ \forall i,j . \quad c_{i,j} \leq b_x \text{ where } \mu(c_{i,j}) = x \]
- \[ \forall x . \quad b_x \leq \sum_{i,j} c_{i,j} \text{ where } \mu(c_{i,j}) = x \]
# Seeds after Seed Selection
(From 100 Seeds)
# of Maximum Attainable Bugs using 20 Seeds over 10 Apps.

<table>
<thead>
<tr>
<th># of Maximum Attainable Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>minset</td>
</tr>
<tr>
<td>sminest</td>
</tr>
<tr>
<td>tminset</td>
</tr>
<tr>
<td>peach</td>
</tr>
</tbody>
</table>
Comparing Seed Selection Algorithms Against Random Set

- **Random Set**: pick \( k \) seeds at random
- **Unweighted MinSet**: MSCP
- **Time MinSet**: WMSCP with execution time
- **Size MinSet**: WMSCP with seed file size
- **Peach Set**: derived from peach fuzzing

Simulated random set **1000** times per program

Compare # of bugs found per \( k \)
Unweighted MinSet Performs Best

Unweighted MinSet is always better than random
More on the Paper

• Detailed seed selection algorithms

• Detailed ILP formulation

• More evaluation
Conclusion

• We formalized, implemented, and tested a number of seed selection algorithms for fuzzing
• We introduced a methodology for evaluating seed selection algorithms for fuzzing
Thank You

Sang Kil Cha

sangkilc@cmu.edu

Code & Data will be soon available:

http://security.ece.cmu.edu/coverset