STOCHFUZZ: Sound and Cost-effective Fuzzing of Stripped Binaries by Incremental and Stochastic Rewriting

Zhuo Zhang, Wei You, Guanhong Tao, Youra Afar, Xuwei Liu, Xiangyu Zhang

Grey-box Fuzzing

Static Binary Instrumentation with Error Awareness

original space

original space

shadow space

0 : mov rbx, [14] // rbx=14 0 : jmp 90 0 : [AFL Tracing]
8 : hit 12 : hit 100 : mov rbx, [14]
12 : jmp rbx
14 : .word 0
22 : syscall

(a) original code

0 : jmp 90 8 : hit 12 : hit 14 : .word 0 22 : hit

(b) ideal case

FNs are detected by intentional crashes
Crashes are detected by intentional crashes which are not triggered by inserted instructions.

FPs are detected by unintentional crashes
Crashes which are not triggered by inserted instructions.

(c) false negative (regarding code as data)

(c) false positive (regarding data as code)

Architecture of STOCHFUZZ

Stochastic Rewriter

Program Dispenser

Execution Engine (AFL)

Incremental & Stochastic Rewriter

random

rewritten

binary

analysis request

binaries

execution request

Probability Analyzer

Crash Analyzer

analyzer triggers incremental rewriting when it determines a certain data coverage.

In the following, we explain how we define the problem formulation.

binary

analysis result

crash

Fault injection

Grey-box fuzzing found:

- More than 21,000 bugs in the Chromium projects
- More than 16,000 bugs in other open source projects

What if the source code is not available (e.g., closed-sourced programs)?

Existing Solutions for Binary-only Fuzzing

Dynamic Binary Translation (OVM-emulator): translates a subject binary during its execution. It is sound but expensive (high overhead >600%).

Hardware-Assisted Tracing (ptfuzzer): makes use of advanced hardware support such as Intel PT to collect runtime traces that can be post-processed (relatively high overhead and only coverage-based feedback).

Static Binary Instrumentation (patchcheck and ddiasm): leverage advanced binary analysis to directly instrument binaries (cost-effective but usually unsound).

Observation

Fuzzing is a highly repetitive process that provides a large number of opportunities for pre- and error.

We can try different data and code separations, which lead to different instrumented executables, in different fuzzing runs.

Progress of Incremental and Stochastic Rewriting

Over time, an increasing number of samples can be collected, allowing us to achieve precise separation and correct rewriting.

Observe the rewriting process converges quickly.

Evaluation

(b) Evaluation on Google FTS with intentional data/inlining (by mixing test and random data)

Mean and standard deviation of time-to-discovery (in minutes) for bugs in Google FTS

<table>
<thead>
<tr>
<th>Tool</th>
<th>Program Size</th>
<th>Days</th>
<th>Programs</th>
<th>Mean</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRead</td>
<td>250.0M</td>
<td>20</td>
<td>600</td>
<td>2.23</td>
<td>0.68</td>
</tr>
<tr>
<td>RetroWrite</td>
<td>175.4M</td>
<td>20</td>
<td>600</td>
<td>2.26</td>
<td>0.68</td>
</tr>
<tr>
<td>ddisasm</td>
<td>25.1M</td>
<td>20</td>
<td>600</td>
<td>2.52</td>
<td>0.68</td>
</tr>
<tr>
<td>ddasm</td>
<td>124.7M</td>
<td>20</td>
<td>600</td>
<td>2.56</td>
<td>0.68</td>
</tr>
<tr>
<td>STOCHFUZZ</td>
<td>124.2M</td>
<td>20</td>
<td>600</td>
<td>2.59</td>
<td>0.68</td>
</tr>
</tbody>
</table>

(c) Zero-day vulnerabilities in close-sourced software

<table>
<thead>
<tr>
<th>Program Size</th>
<th>Mean</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>134M</td>
<td>58</td>
<td>10</td>
</tr>
<tr>
<td>99K</td>
<td>60</td>
<td>12</td>
</tr>
<tr>
<td>CVE-2020-1914</td>
<td>89</td>
<td>17</td>
</tr>
</tbody>
</table>