SyzVegas: Beating Kernel Fuzzing Odds With Reinforcement Learning

Daimeng Wang, Zheng Zhang, Hang Zhang, Zhiyun Qian, Srikanth V. Krishnamurthy, Nael Abu-Ghazaleh

University of California, Riverside
Introduction

- **Fuzzing**
  - Simple in principle
  - Difficult to optimize

- **Kernel Fuzzing: Syzkaller**
  - Evergrowing syscall templates
  - Numerous decisions based on
    - Strong intuitions and domain expertise
    - Empirical testing and tuning
Observations

- Generation is powerful early-on
- Mutation can be more effective
Observations

- **Generation is powerful early-on**
  - Generation-only outperforms vanilla until this point

- **Mutation can be more effective**

```
Coverage (1000 edges)

Time elapsed (hr)

- Core Generate Only
- Full Generate Only
- Core Default
- Full Default

CDF

- Eff Mutation
- Last Eff Mutation
- Total Mutations
```
Observations

- Generation is powerful early-on
  - Generation-only outperforms vanilla until this point

- Mutation can be more effective
  - Seed programs got 100 mutations, some of them effective
  - Seed programs got 100 mutations, none of them effective
  - Seed programs got no mutation
Intuition

- SyzVegas: Treat task scheduling and seed selection as Adversarial Multi-armed-bandit (MAB) problems
  - MAB: No state required. Lightweight.
  - Adversarial: Reward of each choice changes over time.

- Challenge:
  - How to assess the reward?
    - Goal: Max coverage. Min time.
  - How to adapt
    - Goal: Fast reaction
Reward Assessment

- Single reward metric that captures coverage and time spent
- Idea: Currency conversion
  - Conversion rate: total time / total coverage
  - $g = c \times (T/C) - t$
Reward Assessment: Triage, Mutate

\[ T, C \]

\[ \approx \frac{T}{C} \]
Reward Assessment: Triage, Mutate

Entry Fee

$T, ¥C$

$¥ \rightarrow \approx \frac{T}{C}$

$¥ \rightarrow $ $\frac{T}{C}$

$\frac{T}{C}$

$¥0$

$¥c_{min}$

$\delta t = t_{p'} - t_p$

$¥c_{p}$

$¥c_{1}$

$¥c_{m}$
Reward Assessment: Triage, Mutate

Entry Fee

$ T, ¥ C

$ T, ¥ C \rightarrow ¥ \approx T/C

$ t_{ver}^p$

$ t_{min}^p$

$ t_1^p$

$ t_m^p$

$ t^p$

Verify

Minimize

Mutate 1

Mutate m

¥ 0

¥ ($\sum c_i \cdot \frac{t_{ver}^p}{t_{ver}^p + \sum t_i^p}$)

¥ $c_{min}^p + m \cdot \Delta t$

¥ $t^{p'} - t^p$

¥ $c_1^p$

¥ $c_m^p$

Distribute Mutation reward to Triage

Discount $\Delta t = t^{p'} - t^p$

¥ ($\sum c_i \cdot \frac{\sum t_i^p}{t_{ver}^p + \sum t_i^p}$)
Task Selection

- Exp3-IX + Exp3.1
- Record total normalized reward for each task.
- After executing a task, adds the normalized reward to its total.
- Probability of selecting each task is exponential to its total reward.
- Reset the total reward to zero periodically.

\[
\begin{align*}
\hat{g}_{\text{gen}} &= \frac{1 - e^{-g_{\text{gen}}/\sigma_g}}{1 + e^{-g_{\text{gen}}/\sigma_g}} \\
\hat{x}_{\text{gen}} &= \frac{x_{\text{gen}}}{p_{\text{gen}} + \gamma} \\
\hat{g}_{\text{mut}} &= \frac{1 - e^{-g_{\text{mut}}/\sigma_g}}{1 + e^{-g_{\text{mut}}/\sigma_g}} \\
\hat{x}_{\text{mut}} &= \frac{x_{\text{mut}}}{p_{\text{mut}} + \gamma} \\
\hat{g}_{\text{tri}} &= \frac{1 - e^{-g_{\text{tri}}/\sigma_g}}{1 + e^{-g_{\text{tri}}/\sigma_g}} \\
\hat{x}_{\text{tri}} &= \frac{x_{\text{tri}}}{p_{\text{tri}} + \gamma} or \frac{x_{\text{tri}}}{p_{\text{mut}} + \gamma}
\end{align*}
\]
Seed Selection

- Different “conversion” rate
- No need to split reward with triage
- Ever-increasing number of arms
- No need to reset
  - Diminishing reward

\[ \hat{\chi}_1^{(ss)} = \frac{x_1^{(ss)}}{p_{r1} + \gamma} \]

\[ \hat{\chi}_n^{(ss)} = \frac{x_n^{(ss)}}{p_{rn} + \gamma} \]

\[ \$ T_{mut}, \ ¥ C_{mut} \]

\[ ¥ -> $ \approx T_{mut}/C_{mut} \]
Evaluation: Linux Kernel 5.6.13

- Median Coverage
  - TS=Task Selection
  - SS=Seed Selection

- Breakdown

![Graph showing coverage over time and breakdown by methods: TS+SS, TS-Only, SS-Only, Default.]

Coverage (1000 edges) vs Time elapsed (hr)
Evaluation: Linux Kernel 5.6.13

- **Median Coverage**
  - TS=Task Selection
  - SS=Seed Selection

- **Breakdown**
  - Mutation is more effective with seed selection
  - Generation plays a bigger role with task selection
Evaluation: Task Choice

- Syzkaller
- SyzVegas
Evaluation: Task Choice

› Syzkaller

› SyzVegas

SyzVegas performs ~10 times more mutations than vanilla
Evaluation: MAB Behavior

- Gen vs Mut

- Triage
Evaluation: MAB Behavior

Gen vs Mut

Mutation has 200-100 times probability than generation. This is greater than vanilla syzkaller.

Triage

Triage can be delayed for 4-10 hours.
Evaluation: Seed Programs

Seed Number

Seed Power
Evaluation: Seed Programs

Seed Number

Seed Power

Seed created by SyzVegas yields more coverage when mutated.
Evaluation

- 7 days
- With an initial seed corpus

For more experiments, check our paper 😊
<table>
<thead>
<tr>
<th>Crash Type</th>
<th>Functions</th>
<th>Discovered</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SyzVegas</td>
<td>Syzkaller</td>
</tr>
<tr>
<td>Protection Fault</td>
<td>kmem_cache_alloc</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>wait_consider_task</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RCU Stall</td>
<td>ext4_file_write_iter</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>io_uring_release</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>io_uring_setup</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>tty_write</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slab OOB</td>
<td>do_update_region</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>vcs_scr_readw</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>vgacon_scrolldelta</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>vgacon_scroll</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use-after-free</td>
<td>ata_scsi_mode_select_xlat</td>
<td>12</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>clear_buffer_attributes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>complement_pos</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>con_scroll, do_update_region</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>screen_glyph</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>screen_glyph_unicode</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>vc_do_resize</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>vcs_scr_readw</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>vc_uniscr_check</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>vgacon_invert_region</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>vgacon_scroll</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>do_con_write</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>dev_watchdog</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>generic_make_request_checks</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>xfrm_policy_insert_list</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

All crashes are fixed at the time of making this slide.
Discussion & Future Work

- Adversarial MAB over other reinforcement learning
  - No definition of state required
  - Adapt to changing reward
  - Performance overhead 2.1%

- Combining with white-box methods
  - Static analysis, symbolic execution, etc.
  - Some Adversarial MAB algorithms (e.g. EXP4) can take external inputs

- Adjusting other parameters
  - Program size. Mutation operator choice, etc.
  - https://github.com/google/syzkaller/issues/1950
Conclusion

- Identify opportunities of optimization
- Introduce dynamic fuzzing to Syzkaller
- Improve coverage growth

- Git Repo:
  - https://github.com/seclab-ucr/SyzVegas

- Upstream effort:
  - https://github.com/google/syzkaller/pull/1895
Thanks!

Q & A Contact:
• Daimeng Wang: dwang030@ucr.edu
• Zhiyun Qian: zhiyunq@ucr.edu