EcoFuzz: Adaptive Energy-Saving Greybox Fuzzing as a Variant of the Adversarial Multi-Armed Bandit

Tai Yue        Pengfei Wang        Yong Tang        Enze Wang
Bo Yu        Kai Lu        Xu Zhou

Email: yuetai17@nudt.edu.cn
EcoFuzz: https://github.com/MoonLight-SteinsGate/EcoFuzz

National University of Defense Technology
• Effective approach for identifying vulnerabilities
• American Fuzzy Lop (AFL)

The bugs found by AFL
Coverage-based Greybox Fuzzing

- Effective approach for identifying vulnerabilities
- American Fuzzy Lop (AFL)
  - Mutation operator (MOPT, FairFuzz)
  - Initial seeds (Skyfire)

The bugs found by AFL
• Effective approach for identifying vulnerabilities

• American Fuzzy Lop (AFL)
  • Mutation operator (MOPT, FairFuzz)
  • Initial seeds (Skyfire)
Motivation

• Shortcomings in schedule algorithm
  - Assign too much energy on seeds exercising high-frequency paths
  - Simple select strategy
• Few works focus on this
  - AFLFast
• Limitation of current model
Motivation

• Shortcomings in schedule algorithm
  
  • Assign too much energy on seeds exercising high-frequency paths
  
  • Simple select strategy

• Few works focus on this
  
  • AFLFast

• Limitation of current model
Motivation

- Shortcomings in schedule algorithm
  - Assign too much energy on seeds exercising high-frequency paths
  - Simple select strategy
- Few works focus on this
  - AFLFast
- Limitation of current model
Motivation

- Proposing a new model
- Improving schedule algorithm
  - Search strategy: selecting which seed
  - Power schedule: assigning how many energy
Motivation

- Proposing a new model
- Improving schedule algorithm
  - Search strategy: selecting which seed
  - Power schedule: assigning how many energy
Contributions

- One model: a variant of the Adversarial Multi-Armed Bandit (VAMAB)
- One tool: an adaptive energy-saving fuzzer named EcoFuzz
- Comprehensive evaluation: a serial of experiments from different metrics
Contributions

- One model: a variant of the Adversarial Multi-Armed Bandit (VAMAB)
- One tool: an adaptive energy-saving fuzzer named EcoFuzz
- Comprehensive evaluation: a serial of experiments from different metrics
Classical Multi-Armed Bandit

- Constant number of arms
- Reward
- Reward probability
  - constant and unknown
- Target
  - maximizing the rewards in finite trials
<table>
<thead>
<tr>
<th>Classical MAB</th>
<th>CGF</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Arms</td>
<td>• Seeds</td>
</tr>
<tr>
<td>• Reward</td>
<td>• Finding a new path</td>
</tr>
<tr>
<td>• Maximize the rewards</td>
<td>• Maximize path coverage</td>
</tr>
</tbody>
</table>

Classical Multi-Armed Bandit
### Classical Multi-Armed Bandit

<table>
<thead>
<tr>
<th>Classical MAB</th>
<th>CGF</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Arms</td>
<td>• Seeds</td>
</tr>
<tr>
<td>• Reward</td>
<td>• Finding a new path</td>
</tr>
<tr>
<td>• Maximize the rewards</td>
<td>• Maximize path coverage</td>
</tr>
<tr>
<td>• The number of arms is constant</td>
<td>• The number of seeds is variable</td>
</tr>
<tr>
<td>• The reward probability is constant</td>
<td>• The probability of finding new paths is decreasing</td>
</tr>
</tbody>
</table>
A Variant of the Adversarial Multi-Armed Bandit (VAMAB)

VAMAB

- Arms (seeds)
- Reward (Finding a new path)
- Maximize the rewards (path coverage) in finite trails
- The number of arms is variable (increasing), with a upper bound of $n_p$
- The reward probability, which is the probability to find new paths, is variable (decreasing)
A Variant of the Adversarial Multi-Armed Bandit (VAMAB)

VAMAB

- Arms (seeds)
  - Seed-1
  - Seed-2
  - Seed-3
  - ... 
  - Seed-n

- Reward (Finding a new path)
  - $P_{R_{1,n}}$
  - $P_{R_{2,n}}$
  - $P_{R_{3,n}}$
  - $P_{R_{n,n}}$

- Maximize the rewards (path coverage) in finite trails

- The number of arms is variable (increasing), with an upper bound of $n_p$

- The *reward probability, which is the probability to find new paths*, is variable (decreasing)
A Variant of the Adversarial Multi-Armed Bandit (VAMAB)

VAMAB

- Arms (seeds)
- Reward (Finding a new path)
- Maximize the rewards (path coverage) in finite trails
- The number of arms is variable (increasing), with a upper bound of \( n_p \)
- The reward probability, which is the probability to find new paths, is variable (decreasing)

\[
P_{R_i,n} = \sum_{j=n+1}^{n_p} p_{ij} = 1 - \sum_{j=1}^{n} p_{ij}
\]
A Variant of the Adversarial Multi-Armed Bandit (VAMAB)

VAMAB

- Arms (seeds)
- Reward (Finding a new path)
- Maximize the rewards (path coverage) in finite trails
- The number of arms is variable (increasing), with an upper bound of \( n_p \)
- The reward probability, which is the probability to find new paths, is variable (decreasing)
A Variant of the Adversarial Multi-Armed Bandit (VAMAB)

VAMAB

- Arms (seeds)
- Reward (Finding a new path)
- Maximize the rewards (path coverage) in finite trails
- The number of arms is variable (increasing), with a upper bound of $n_p$
- The reward probability, which is the probability to find new paths, is variable (decreasing)

\[
P_{R_{i,n}} = \sum_{j=n+1}^{n_p} p_{ij} = 1 - \sum_{j=1}^{n} p_{ij}
\]

**Transition Probability**

The probability of mutating the seed $i$ to generate a test case executing the path $j$. 
A Variant of the Adversarial Multi-Armed Bandit (VAMAB)

VAMAB

- Arms (seeds)
- Reward (Finding a new path)
- Maximize the rewards (path coverage) in finite trails
- The number of arms is variable (increasing), with a upper bound of $n_p$
- The reward probability, which is the probability to find new paths, is variable (decreasing)

\[
PR_{i,n} = \sum_{j=n+1}^{n_p} p_{ij} = 1 - \sum_{j=1}^{n} p_{ij} \quad \Rightarrow \quad PR_{i,n} > PR_{i,n+1} > \ldots > PR_{i,n_p} = 0
\]

Transition Probability

- The probability of mutating the seed $i$ to generate a test case executing the path $j$. 
A Variant of the Adversarial Multi-Armed Bandit (VAMAB)

VAMAB

- Arms (seeds)
- Reward (Finding a new path)
- Maximize the rewards (path coverage) in finite trails
- The number of arms is variable (increasing), with an upper bound of $n_p$
- The reward probability, which is the probability to find new paths, is variable (decreasing)

\[
P_{R_i,n} = \sum_{j=n+1}^{n_p} p_{ij} = 1 - \sum_{j=1}^{n} p_{ij} \quad P_{R_i,n} > P_{R_{i,n+1}} > \ldots > P_{R_{i,n_p}} = 0
\]

Transition Probability

The probability of mutating the seed $i$ to generate a test case executing the path $j$.

Probability attenuation
A Variant of the Adversarial Multi-Armed Bandit (VAMAB)

Exploration

- Estimate their reward probabilities

Exploitation

- Select the seed with a high reward probability
A Variant of the Adversarial Multi-Armed Bandit (VAMAB)

**Exploration**

- Estimate their **reward probabilities**

**Exploitation**

- Select the seed with a high reward probability
A Variant of the Adversarial Multi-Armed Bandit (VAMAB)

**Exploration**
- Estimate their reward probabilities

**Exploitation**
- Select the seed with a high reward probability

![Diagram of exploration and exploitation strategies]
A Variant of the Adversarial Multi-Armed Bandit (VAMAB)

**Exploration**
- Estimate their reward probabilities
- Focusing on exploring new seeds:
  - Assigning fewer energy on the old seeds with high reward probabilities

**Exploitation**
- Select the seed with a high reward probability
- Focusing on exploiting old seeds:
  - Missing some new seeds with higher reward probabilities
A Variant of the Adversarial Multi-Armed Bandit (VAMAB)

**Exploration**
- Estimate their reward probabilities
- Focusing on exploring new seeds:
  - Assigning fewer energy on the old seeds with high reward probabilities

**Exploitation**
- Select the seed with a high reward probability
- Focusing on exploiting old seeds:
  - Missing some new seeds with higher reward probabilities
Three States in CGF

- **Initial state**: all seeds are unfuzzed
- **Exploration state**: part of seeds in the seed queue are fuzzed
- **Exploitation State**: all seeds in the seed queue have been fuzzed
How to Maximize Coverage

Search Strategy

- Estimating the **reward probability**
- Selecting the seeds with high **reward probabilities**

Power Schedule

- Avoiding assigning too much energy to some seeds
How to Maximize Coverage

Search Strategy

- Estimating the reward probability
- Selecting the seeds with high reward probabilities

Power Schedule

- Avoiding assigning too much energy on some seeds
Contributions

• One model: a variant of the Adversarial Multi-Armed Bandit (VAMAB)

• One tool: an adaptive energy-saving fuzzer named EcoFuzz

• Comprehensive evaluation: a serial of experiments from different metrics
EcoFuzz

- **Main Framework**

  - Initial Seed
  - Seed Queue
  - Exploration
  - Initial
  - Exploitation
  - Seed
  - Choose next seed in order
  - Choose next seed by SPEM
  - Assign energy by AAPS
  - Mutation
  - Random Stage
  - Test cases
  - Find new path

**Based on AFL**

- Search strategy: Self-transition-based Probability Estimation Method (SPEM)
- Power schedule: Adaptive Average-cost-based Power Schedule (AAPS)
- Three states of VAMAB
Main Framework

- Based on AFL
- Search strategy: **Self-transition-based Probability Estimation Method (SPEM)**
- Power schedule: Adaptive Average-cost-based Power Schedule (AAPS)
- Three states of VAMAB

Initial Seed

Seed Queue

Exploration

Initial

Exploitation

Seed

Choose next seed in order

Choose next seed by SPEM

Assign energy by AAPS

Mutation

Random Stage

Test cases

Find new path

State Determine
EcoFuzz

- **Main Framework**

- Based on AFL
- Search strategy: Self-transition-based Probability Estimation Method (SPEM)
- Power schedule: Adaptive Average-cost-based Power Schedule (AAPS)
- Three states of VAMAB

- Initial Seed
- Seed Queue
- Exploration
- Initial
- Exploitation
- Mutation
- Random Stage
- Test cases
- Find new path
- State Determine
- Choose next seed in order
- Choose next seed by SPEM
- Assign energy by AAPS
Main Framework

- **Seed Queue**
  - Initial Seed
  - Exploration
  - Initial
  - Exploitation
  - Choose next seed in order
  - Choose next seed by SPEM
  - Assign energy by AAPS

- **State Determine**
  - Find new path
  - Test cases
  - Mutation
  - Random Stage

- **EcoFuzz**

- Based on AFL
- Search strategy: Self-transition-based Probability Estimation Method (SPEM)
- Power schedule: Adaptive Average-cost-based Power Schedule (AAPS)
- Three states of VAMAB
EcoFuzz

- **Main Framework**

  - Based on AFL
  - Search strategy: Self-transition-based Probability Estimation Method (SPEM)
  - Power schedule: Adaptive Average-cost-based Power Schedule (AAPS)
  - Three states of VAMAB

  [Diagram of the EcoFuzz main framework]

  - Initial Seed
  - Seed Queue
  - Exploration
  - Initial
  - Exploitation
  - Seed
  - Mutation
  - Random Stage
  - Test cases
  - Find new path

  **States:**
  - State Determine
  - Choose next seed in order
  - Choose next seed by SPEM
  - Assign energy by AAPS
Main Framework

- Based on AFL
- Search strategy: Self-transition-based Probability Estimation Method (SPEM)
- Power schedule: Adaptive Average-cost-based Power Schedule (AAPS)
- Three states of VAMAB

Initial Seed

Seed Queue

State Determine

Exploration

Choose next seed in order

Initial

Exploitation

Choose next seed by SPEM

Seed

Mutation

Random Stage

Assign energy by AAPS

Test cases

Find new path

Test cases

Random Stage

Seed Queue

Initial Seed
EcoFuzz

- Main Framework

- Based on AFL
- Search strategy: Self-transition-based Probability Estimation Method (SPEM)
- Power schedule: Adaptive Average-cost-based Power Schedule (AAPS)
- Three states of VAMAB
Main Framework

- Based on AFL
- Search strategy: Self-transition-based Probability Estimation Method (SPEM)
- Power schedule: Adaptive Average-cost-based Power Schedule (AAPS)
- Three states of VAMAB
### EcoFuzz

**Main Framework**

- **Initial Seed**
  - Seed Queue
  - State Determine
- **Exploration**
- **Exploitation**
  - Choose next seed in order by SPEM
- **Mutation**
  - Random Stage
  - Test cases
  - Find new path

- **Seed**
  - Choose next seed by SPEM
  - Assign energy by AAPS

- **Based on AFL**
- **Search strategy: Self-transition-based Probability Estimation Method (SPEM)**
- **Power schedule: Adaptive Average-cost-based Power Schedule (AAPS)**
- **Three states of VAMAB**
EcoFuzz

- **Main Framework**
  - Initial Seed
  - Seed Queue
  - Exploration
  - State Determine
  - Initial
  - Exploitation
  - Choose next seed in order
  - Seed
  - Choose next seed by SPEM
  - Assign energy by AAPS
  - Find new path
  - Test cases
  - Mutation
  - Random Stage
  - Determination

- **Based on AFL**
- **Search strategy**: Self-transition-based Probability Estimation Method (SPEM)
- **Power schedule**: Adaptive Average-cost-based Power Schedule (AAPS)
- **Three states of VAMAB**
Self-transition-based Probability Estimation Method (SPEM)
Self-transition-based Probability Estimation Method (SPEM)

- Search strategy
• Search strategy
  • estimate the **reward probabilities**
  • select the next seed in **exploitation state**
Self-transition-based Probability Estimation Method (SPEM)

- Search strategy
  - estimate the reward probabilities
  - select the next seed in exploitation state
- Estimating probability by frequency

\[ P_{R_{i,n}} = 1 - \sum_{j=1}^{n} p_{ij} = 1 - p_{ii} - \sum_{j=1, j\neq i}^{n} p_{ij} \]
Self-transition-based Probability Estimation Method (SPEM)

- Search strategy
  
  - estimate the **reward probabilities**
  
  - select the next seed in **exploitation state**

- Estimating probability by frequency

\[ P_{R_{i,n}} = 1 - \sum_{j=1}^{n} p_{ij} = 1 - p_{ii} - \sum_{j=1, j\neq i}^{n} p_{ij} \]
Self-transition-based Probability Estimation Method (SPEM)

- Search strategy
  - estimate the **reward probabilities**
  - select the next seed in **exploitation state**
- Estimating probability by frequency
  - \( p_{ii} \approx f_{ii} \)

\[
P_{R_{i,n}} = 1 - \sum_{j=1}^{n} p_{ij} = 1 - p_{ii} - \sum_{j=1, j \neq i}^{n} p_{ij}
\]

\[
P_{R_{i,n}} = 1 - \sum_{j=1}^{n} p_{ij} \approx 1 - f_{ii} - \sum_{j=1, j \neq i}^{n} p_{ij}
\]
Self-transition-based Probability Estimation Method (SPEM)

- Search strategy
  - estimate the **reward probabilities**
  - select the next seed in **exploitation state**
- Estimating probability by frequency
  - \( p_{ii} \approx f_{ii} \)

\[
P_{R_{i,n}} = 1 - \sum_{j=1}^{n} p_{ij} = 1 - p_{ii} - \sum_{j=1, j \neq i}^{n} p_{ij}
\]

\[
P_{R_{i,n}} = 1 - \sum_{j=1}^{n} p_{ij} \approx 1 - f_{ii} - \sum_{j=1, j \neq i}^{n} p_{ij}
\]
Self-transition-based Probability Estimation Method (SPEM)

- Search strategy
  - estimate the **reward probabilities**
  - select the next seed in **exploitation state**
- Estimating probability by frequency
  - \( p_{ii} \approx f_{ii} \)
  - \( f_{ii} + \sum_{j=1, j \neq i}^{n} p_{ij} \approx \frac{f_{ii}}{\sqrt{i}} \)

\[
\begin{align*}
P_{R_{i,n}} &= 1 - \sum_{j=1}^{n} p_{ij} = 1 - p_{ii} - \sum_{j=1, j \neq i}^{n} p_{ij} \\
P_{R_{i,n}} &\approx 1 - f_{ii} - \sum_{j=1, j \neq i}^{n} p_{ij} \\
P_{R_{i,n}} &= 1 - \sum_{j=1}^{n} p_{ij} \approx 1 - \frac{f_{ii}}{\sqrt{i}}
\end{align*}
\]
Adaptive Average-cost-based Power Schedule (AAPS)
Adaptive Average-cost-based Power Schedule (AAPS)

- Average-cost
Adaptive Average-cost-based Power Schedule (AAPS)

- **Average-cost**

  \[
  C = \frac{\text{total\_testcases}}{\text{found\_paths}}
  \]
Adaptive Average-cost-based Power Schedule (AAPS)

• **Average-cost**

\[ C = \frac{\text{total\_testcases}}{\text{found\_paths}} \]

• Utilize **average-cost** as the **basic line**
Adaptive Average-cost-based Power Schedule (AAPS)

- **Average-cost**
  
  \[ C = \frac{\text{total\_testcases}}{\text{found\_paths}} \]

- Utilize **average-cost** as the **basic line**

### Algorithm 2 The AAPS algorithm

```plaintext
Require: s, state, rate, average_cost

Energy = 0

if state == Exploration then
  k = CalculateCoefficient(s.exec_num, average_cost)
  Energy = average_cost × k × rate
else if state == Exploitation then
  if s.last_found > 0 then
    Energy = Min(s.last_energy, M) × rate
  else
    Energy = Min(s.last_energy × 2, M) × rate
  end if
else
  Energy = 1024 × rate
end if

Ensure: Energy
```
### Adaptive Average-cost-based Power Schedule (AAPS)

- **Average-cost**
  
  \[ C = \frac{\text{total\_testcases}}{\text{found\_paths}} \]

- Utilize **average-cost** as the **basic line**

- Allocating energy no more than average-cost in the **exploration stage**

---

**Algorithm 2 The AAPS algorithm**

```plaintext
Require: s, state, rate, average_cost

Energy = 0
if state == Exploration then
    k = CalculateCoefficient(s.exec_num, average_cost)
    Energy = average_cost \times k \times rate
else if state == Exploitation then
    if s.last_found > 0 then
        Energy = Min(s.last_energy, M) \times rate
    else
        Energy = Min(s.last_energy \times 2, M) \times rate
    end if
else
    Energy = 1024 \times rate
end if

Ensure: Energy
```
Adaptive Average-cost-based Power Schedule (AAPS)

- **Average-cost**
  
  \[ C = \frac{\text{total\_testcases}}{\text{found\_paths}} \]

- Utilize **average-cost** as the **basic line**

- Allocating energy no more than average-cost in the **exploration stage**

- Less energy allocated on seeds exercising **high-frequency paths**

---

**Algorithm 2 The AAPS algorithm**

```
Require: s, state, rate, average_cost
Energy = 0
if state == Exploration then
    k = CalculateCoefficient(s.exec_num, average_cost)
    Energy = average_cost × k × rate
else if state == Exploitation then
    if s.last_found > 0 then
        Energy = Min(s.last_energy, M) × rate
    else
        Energy = Min(s.last_energy × 2, M) × rate
    end if
else
    Energy = 1024 × rate
end if
Ensure: Energy
```
Adaptive Average-cost-based Power Schedule (AAPS)

- **Average-cost**
  
  \[ C = \frac{\text{total\_testcases}}{\text{found\_paths}} \]

- Utilize **average-cost** as the **basic line**
  
  - Allocating energy no more than average-cost in the **exploration stage**
  
  - Less energy allocated on seeds exercising **high-frequency paths**
  
  - A **context-adaptive** energy allocation mechanism

---

**Algorithm 2** The AAPS algorithm

Require: \( s, \text{state}, \text{rate}, \text{average\_cost} \)

\[
\begin{align*}
\text{Energy} &= 0 \\
\text{if} \ & \text{state} == \text{Exploration} \text{ then} \\
\quad k &= \text{CalculateCoefficient}(s.\text{exec\_num}, \text{average\_cost}) \\
\quad \text{Energy} &= \text{average\_cost} \times k \times \text{rate} \\
\text{else if} \ & \text{state} == \text{Exploitation} \text{ then} \\
\quad \text{if} \ & s.\text{last\_found} > 0 \text{ then} \\
\quad\quad \text{Energy} &= \text{Min}(s.\text{last\_energy}, M) \times \text{rate} \\
\quad\quad \text{else} \\
\quad\quad\quad \text{Energy} &= \text{Min}(s.\text{last\_energy} \times 2, M) \times \text{rate} \\
\text{end if} \\
\text{else} \\
\quad \text{Energy} &= 1024 \times \text{rate} \\
\text{end if}
\end{align*}
\]

Ensure: \( \text{Energy} \)
Contributions

• One model: a variant of the Adversarial Multi-Armed Bandit (VAMAB)

• One tool: an adaptive energy-saving fuzzer named EcoFuzz

• Comprehensive evaluation: a serial of experiments from different metrics
• 14 real-world programs

• Compared with 7 state-of-the-art tools
  • AFL, AFLFast, FidgetyAFL, AFLFast.new, MOPT, FairFuzz

• Configuration:
  • 24 hours with 5 times

• Evaluation metric:
  • The number of discovered paths
  • The number of generated test cases
  • Average-cost

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Version</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>nm -C @ @</td>
<td>Binutils-v2.32</td>
<td>elf</td>
</tr>
<tr>
<td>objdump -d @ @</td>
<td>Binutils-v2.32</td>
<td>elf</td>
</tr>
<tr>
<td>readelf -a @ @</td>
<td>Binutils-v2.32</td>
<td>elf</td>
</tr>
<tr>
<td>size @ @</td>
<td>Binutils-v2.32</td>
<td>elf</td>
</tr>
<tr>
<td>c++filt @ @</td>
<td>Binutils-v2.32</td>
<td>elf</td>
</tr>
<tr>
<td>jpeg @ @</td>
<td>libjpeg-turbo-1.5.3</td>
<td>jpeg</td>
</tr>
<tr>
<td>xmlint @ @</td>
<td>libxml2-2.9.9</td>
<td>xml</td>
</tr>
<tr>
<td>gif2png @ @</td>
<td>gif2png-2.5.13</td>
<td>gif</td>
</tr>
<tr>
<td>readpng @ @</td>
<td>libpng-1.6.37</td>
<td>png</td>
</tr>
<tr>
<td>tcpdump -nr @ @</td>
<td>tcpdump-4.9.2</td>
<td>pcap</td>
</tr>
<tr>
<td>infotocap @ @</td>
<td>ncurses-6.1</td>
<td>text</td>
</tr>
<tr>
<td>jhead @ @</td>
<td>jhead-3.03</td>
<td>jpeg</td>
</tr>
<tr>
<td>magick convert @ @ /dev/null</td>
<td>ImageMagick-7.0.8-65</td>
<td>png</td>
</tr>
<tr>
<td>bsd tar -xf @ @ /dev/null</td>
<td>libarchive-3.4.0</td>
<td>tar</td>
</tr>
</tbody>
</table>
• 14 real-world programs

• Compared with 7 state-of-the-art tools
  - AFL, AFLFast, FidgetyAFL, AFLFast.new, MOPT, FairFuzz

• Configuration:
  - 24 hours with 5 times

• Evaluation metric:
  - The number of discovered paths
  - The number of generated test cases
  - Average-cost
Evaluation

- 14 real-world programs
- Compared with 7 state-of-the-art tools
  - AFL, AFLFast, FidgetyAFL, AFLFast.new, MOPT, FairFuzz
- Configuration:
  - 24 hours with 5 times
- Evaluation metric:
  - The number of discovered paths
  - The number of generated test cases
  - Average-cost
14 real-world programs

Compared with 7 state-of-the-art tools
- AFL, AFLFast, FidgetyAFL, AFLFast.new, MOPT, FairFuzz

Configuration:
- 24 hours with 5 times

Evaluation metric:
- The number of discovered paths
- The number of generated test cases
- Average-cost
### Evaluation

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Number of total paths / Number of executions finding these paths</th>
<th>Average-cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FidgetyAFL / AFLFast.new / FairFuzz / EcoFuzz</td>
<td></td>
</tr>
<tr>
<td>nm</td>
<td>4,975 / 80.34M / 8,127 / 60.95M / 3,890 / 51.42M / 8,266 / 42.88M / 16,152 / 7,500 / 13,222 / 5,188</td>
<td></td>
</tr>
<tr>
<td>objdump</td>
<td>7,186 / 65.03M / 7,241 / 62.45M / 5,287 / 43.34M / 7,474 / 42.78M / 9,051 / 8,626 / 8,200 / 5,724</td>
<td></td>
</tr>
<tr>
<td>readelf</td>
<td>13,063 / 51.73M / 14,048 / 60.90M / 8,813 / 47.47M / 12,649 / 53.90M / <strong>3,960</strong> / 4,335 / 5,387 / 4,261</td>
<td></td>
</tr>
<tr>
<td>size</td>
<td>3,352 / 87.12M / 3,601 / 85.31M / 2,782 / 48.90M / 3,939 / 76.45M / 25,998 / 23,698 / <strong>17,581</strong> / 19,412</td>
<td></td>
</tr>
<tr>
<td>cxxfilt</td>
<td>7,715 / 72.37M / 8,192 / 64.90M / 5,054 / 67.59M / 7,119 / 26.19M / 9,381 / 7,923 / 13,377 / <strong>3,679</strong></td>
<td></td>
</tr>
<tr>
<td>djeg</td>
<td>3,587 / 57.77M / 3,706 / 50.29M / 1,902 / 10.45M / 2,996 / 36.78M / 16,109 / 13,572 / <strong>5,498</strong> / 12,280</td>
<td></td>
</tr>
<tr>
<td>xmlint</td>
<td>6,269 / 55.69M / 7,214 / 52.12M / 5,322 / 43.21M / 6,803 / 33.11M / 8,884 / 7,225 / 8,120 / <strong>4,868</strong></td>
<td></td>
</tr>
<tr>
<td>gif2png</td>
<td>4,004 / 107.46M / 4,226 / 112.38M / 2,952 / 25.88M / 4,292 / 59.53M / 26,844 / 26,600 / <strong>8,769</strong> / 13,873</td>
<td></td>
</tr>
<tr>
<td>readpng</td>
<td>1,884 / 61.36M / 1,952 / 44.39M / 1,753 / 35.48M / 2,023 / 22.66M / 32,585 / 22,755 / 20,253 / <strong>11,205</strong></td>
<td></td>
</tr>
<tr>
<td>tcpdump</td>
<td>10,432 / 93.37M / 12,993 / 126.74M / 11,489 / 137.89M / 13,059 / 74.27M / 8,951 / 9,755 / 12,003 / <strong>5,688</strong></td>
<td></td>
</tr>
<tr>
<td>infotocap</td>
<td>6,125 / 36.23M / 6,389 / 33.47M / 3,921 / 25.23M / 5,840 / 12.36M / 5,917 / 5,239 / 6,436 / <strong>2,117</strong></td>
<td></td>
</tr>
<tr>
<td>jhead</td>
<td>538 / 120.60M / 539 / 32.16M / 506 / 49.69M / 594 / 64.86M / 224,575 / <strong>59,775</strong> / 98,402 / 278,008</td>
<td></td>
</tr>
<tr>
<td>magic</td>
<td>4,903 / 6.70M / 5,375 / 9.63M / 3,419 / 6.56M / 5,483 / 5.97M / 1,367 / 1,793 / 1,919 / <strong>1,089</strong></td>
<td></td>
</tr>
<tr>
<td>bsdar</td>
<td>6,685 / 54.84M / 7,143 / 51.15M / 3,981 / 39.55M / 7,209 / 45.17M / 8,204 / 7,162 / 9,936 / <strong>6,266</strong></td>
<td></td>
</tr>
</tbody>
</table>

*The number of executions finding these paths denotes the number of test cases are generated when the fuzzers have reached these paths, of which the unit is M(10^6). Bold fonts represent the best performance.

- Outperform other AFL-type techniques
  - EcoFuzz finds **214%** of the paths discovered by AFL and generates only **68%** test cases of AFL, while reducing **65%** average-cost of AFL
Evaluation

- Evaluate the efficiency of SPEM and AAPS

- Configuration:
  - choosing each best performance of EcoFuzz, FidgetyAFL, FairFuzz, and AFLFast.new on fuzzing nm
  - recording the energy allocated in random strategies of each turns, denoted as $E_i$, which $i$ is the order of turn $(1 \leq i \leq N)$
  - recording the consumed energy for discovering the newest path of each turns, denoted as $e_i$, $0 \leq e_i \leq E_i$
  - recording the frequency of allocation with finding new paths for the seeds chosen repeatedly in the exploitation stage
• Evaluate the efficiency of SPEM and AAPS

• Configuration:

  - choosing each best performance of EcoFuzz, FidgetyAFL, FairFuzz, and AFLFast.new on fuzzing

  - recording the energy allocated in random strategies of each turns, denoted as $E_i$, which $i$ is the order of turn $(1 \leq i \leq N)$

  - recording the consumed energy for discovering the newest path of each turns, denoted as $e_i$, $0 \leq e_i \leq E_i$

  - recording the frequency of allocation with finding new paths for the seeds chosen repeatedly in the exploitation stage
• Evaluate the efficiency of SPEM and AAPS

• Configuration:

  - choosing each best performance of EcoFuzz, FidgetyAFL, FairFuzz, and AFLFast.new on fuzzing nm

  - recording the energy allocated in random strategies of each turns, denoted as $E_i$, which $i$ is the order of turn ($1 \leq i \leq N$)

  - recording the consumed energy for discovering the newest path of each turns, denoted as $e_i$, $0 \leq e_i \leq E_i$

  - recording the frequency of allocation with finding new paths for the seeds chosen repeatedly in the exploitation stage
• Evaluate the efficiency of SPEM and AAPS

• Configuration:
  - choosing each best performance of EcoFuzz, FidgetyAFL, FairFuzz, and AFLFast.new on fuzzing
  - recording the energy allocated in random strategies of each turns, denoted as $E_i$, which $i$ is the order of turn ($1 \leq i \leq N$)
  - recording the consumed energy for discovering the newest path of each turns, denoted as $e_i$, $0 \leq e_i \leq E_i$
  - recording the frequency of allocation with finding new paths for the seeds chosen repeatedly in the exploitation stage
Evaluation

- Evaluate the efficiency of SPEM and AAPS

- Configuration:
  - choosing each best performance of **EcoFuzz**, **FidgetyAFL**, **FairFuzz**, and **AFLFast.new** on fuzzing
  - recording the energy allocated in random strategies of each turns, denoted as $E_i$, which $i$ is the order of turn $(1 \leq i \leq N)$
  - recording the consumed energy for discovering the newest path of each turns, denoted as $e_i$, $0 \leq e_i \leq E_i$
  - recording the frequency of allocation with finding new paths for the seeds chosen repeatedly in the exploitation state
Evaluation metric:

- **The utilization ratio of energy**
  \[ r_i = \frac{e_i}{E_i} \]

- **The average utilization ratio**
  \[ \bar{r} = \frac{\sum_{i=1}^{i=N} r_i}{N} \]

- **The frequency of effective allocation**
  \[ p = \frac{|\{i | e_i > 0, 1 \leq i \leq N\}|}{N} \]
Evaluation
Scatter map of $r_i$ with $i$
Scatter map of $r_i$ with $i$

- **FidgetyAFL** and **AFLFast.new**

  $r_i < 0.5$
Scatter map of $r_i$ with $i$

- **FidgetyAFL** and **AFLFast.new**
  - $r_i < 0.5$

- **EcoFuzz**
  - $r_i \rightarrow 1.0$
Table of $\bar{r}$ and $p$

- EcoFuzz demonstrates the best performance
  - The least average-cost
  - The highest average utilization
  - The highest frequency of effective allocation
  - The highest ratio of effective allocation to the repeated chosen times in exploitation state

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Average utilization ratio</th>
<th>Effective allocation</th>
<th>Average-cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>EcoFuzz</td>
<td>0.121</td>
<td>0.290</td>
<td>4.314</td>
</tr>
<tr>
<td>FidgetyAFL</td>
<td>0.005</td>
<td>0.013</td>
<td>9.078</td>
</tr>
<tr>
<td>AFLFast.new</td>
<td>0.010</td>
<td>0.031</td>
<td>7.046</td>
</tr>
<tr>
<td>FairFuzz</td>
<td>0.107</td>
<td>0.204</td>
<td>4.930</td>
</tr>
</tbody>
</table>

Table 4: The evaluation of search strategy

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Allocation with New Finding</th>
<th>Repeated Chosen</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>EcoFuzz</td>
<td>705</td>
<td>10,174</td>
<td>0.069</td>
</tr>
<tr>
<td>FidgetyAFL</td>
<td>364</td>
<td>11,703</td>
<td>0.031</td>
</tr>
<tr>
<td>AFLFast.new</td>
<td>54</td>
<td>2,066</td>
<td>0.026</td>
</tr>
<tr>
<td>FairFuzz</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>
• Detecting vulnerabilities

• **12 vulnerabilities**

• 2 CVEs

### Table 8: The discovered vulnerabilities

<table>
<thead>
<tr>
<th>Softwares</th>
<th>File/Function</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binutils-v2.32</td>
<td>cp-demangle.c/d_expression_1</td>
<td>CVE-2019-9070</td>
</tr>
<tr>
<td>Binutils-v2.32</td>
<td>hash.c/bfd_hash_hash</td>
<td>Submitted</td>
</tr>
<tr>
<td>Binutils-v2.32</td>
<td>bfd.c/_bfd_doprm</td>
<td>CVE-2019-12972</td>
</tr>
<tr>
<td>Binutils-v2.31</td>
<td>xmalloc.c/xmalloc</td>
<td>Patched</td>
</tr>
<tr>
<td>Binutils-v2.31</td>
<td>cplusplus-dem.c/string_append</td>
<td>Patched</td>
</tr>
<tr>
<td>Binutils-v2.31</td>
<td>cplusplus-dem.c/string_append_template_idx</td>
<td>Patched</td>
</tr>
<tr>
<td>Binutils-v2.31</td>
<td>cplusplus-dem.c/demangle_class_name</td>
<td>Patched</td>
</tr>
<tr>
<td>gif2png-2.5.13</td>
<td>gif2png.c/writefile</td>
<td>Submitted</td>
</tr>
<tr>
<td>gif2png-2.5.13</td>
<td>memory.c/xmalloc</td>
<td>Submitted</td>
</tr>
<tr>
<td>libpng-1.6.37</td>
<td>pngmem.c/png_malloc_warn</td>
<td>CVE-2019-17371</td>
</tr>
<tr>
<td>tcpdump-4.9.2</td>
<td>tcpdump.c/copy_argv</td>
<td>Submitted</td>
</tr>
<tr>
<td>jhead-3.03</td>
<td>jpegguess.c/process_DQT</td>
<td>Submitted</td>
</tr>
<tr>
<td>SNMP deamon</td>
<td>snmp/Context::createReply</td>
<td>Patched</td>
</tr>
</tbody>
</table>
**Evaluation**

- Detecting vulnerabilities
- 12 vulnerabilities
- 2 CVEs

<table>
<thead>
<tr>
<th>Softwares</th>
<th>File/Function</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binutils-v2.32</td>
<td>cp-demangle.c/d_expression_1</td>
<td>CVE-2019-9070</td>
</tr>
<tr>
<td>Binutils-v2.32</td>
<td>hash.c/bfd_hash_hash</td>
<td>Submitted</td>
</tr>
<tr>
<td>Binutils-v2.32</td>
<td>bfd.c/_bfd_doprm</td>
<td>CVE-2019-12972</td>
</tr>
<tr>
<td>Binutils-v2.31</td>
<td>xmalloc.c/xmalloc</td>
<td>Patched</td>
</tr>
<tr>
<td>Binutils-v2.31</td>
<td>cplus-dem.c/string_append</td>
<td>Patched</td>
</tr>
<tr>
<td>Binutils-v2.31</td>
<td>cplus-dem.c/string_append_template_idx</td>
<td>Patched</td>
</tr>
<tr>
<td>Binutils-v2.31</td>
<td>cplus-dem.c/demangle_class_name</td>
<td>Patched</td>
</tr>
<tr>
<td>gif2png-2.5.13</td>
<td>gif2png.c/writefile</td>
<td>Submitted</td>
</tr>
<tr>
<td>gif2png-2.5.13</td>
<td>memory.c/xalloc</td>
<td>Submitted</td>
</tr>
<tr>
<td>libpng-1.6.37</td>
<td>pngmem.c/png_malloc_warn</td>
<td>CVE-2019-17371</td>
</tr>
<tr>
<td>tcpdump-4.9.2</td>
<td>tcpdump.c/copy_argv</td>
<td>Submitted</td>
</tr>
<tr>
<td>jhead-3.03</td>
<td>jgqguess.c/process_DQT</td>
<td>Submitted</td>
</tr>
<tr>
<td>SNMP deamon</td>
<td>snmp/Context::createReply</td>
<td>Patched</td>
</tr>
</tbody>
</table>
Conclusion

- VAMAB
  - Model the schedule process of CGF
  - Explain the exploration and exploitation in CGF
  - Point out how to improve greybox fuzzing
Conclusion

• VAMAB
  - Model the schedule process of CGF
  - Explain the exploration and exploitation in CGF
  - Point out how to improve greybox fuzzing
Conclusion

• VAMAB
  - Model the schedule process of CGF
  - Explain the exploration and exploitation in CGF
  - Point out how to improve greybox fuzzing
Conclusion

• VAMAB
  • Model the schedule process of CGF
  • Explain the exploration and exploitation in CGF
  • Point out how to improve greybox fuzzing
Conclusion

- **VAMAB**
  - Model the schedule process of CGF
  - Explain the exploration and exploitation in CGF
  - Point out how to improve greybox fuzzing
    - Improving the reward probability of each seed:
      - FairFuzz (avoids mutating the crucial parts of seeds)
    - Establishing the mechanism of evaluating the seeds:
      - AFLGo (utilizes distance to evaluate each seed)
    - Optimizing the power schedule:
      - AFLFast (increases the assigned energy monotonically)
Conclusion

• **VAMAB**
  - Model the schedule process of CGF
  - Explain the exploration and exploitation in CGF
  - **Point out how to improve greybox fuzzing**
    - Improving the reward probability of each seed:
      - FairFuzz(avoid mutating the crucial parts of seeds)
    - Establishing the mechanism of evaluating the seeds:
      - AFLGo(utilizes distance to evaluate each seed)
    - Optimizing the power schedule:
      - AFLFast(increases the assigned energy monotonically)
Conclusion

• **VAMAB**
  - Model the schedule process of CGF
  - Explain the exploration and exploitation in CGF
  - Point out how to improve greybox fuzzing
    - Improving the reward probability of each seed:
      - FairFuzz (avoids mutating the crucial parts of seeds)
    - Establishing the mechanism of evaluating the seeds:
      - AFLGo (utilizes distance to evaluate each seed)
    - Optimizing the power schedule:
      - AFLFast (increases the assigned energy monotonically)
Conclusion

- **VAMAB**
  - Model the schedule process of CGF
  - Explain the exploration and exploitation in CGF
  - Point out how to improve greybox fuzzing
- **EcoFuzz**
  - Search strategy (Self-transition-based Probability Estimation Method)
  - Power schedule (Adaptive Average-cost-based Power Schedule)
- Comprehensive evaluation:
  - Attain 214% of the path coverage of AFL with reducing 32% test cases
  - Identifying 12 vulnerabilities
Conclusion

• VAMAB
  - Model the schedule process of CGF
  - Explain the exploration and exploitation in CGF
  - Point out how to improve greybox fuzzing

• EcoFuzz
  - Search strategy (Self-transition-based Probability Estimation Method)
  - Power schedule (Adaptive Average-cost-based Power Schedule)

• Comprehensive evaluation:
  - Attain 214% of the path coverage of AFL with reducing 32% test cases
  - Identifying 12 vulnerabilities
Conclusion

• **VAMAB**
  - Model the schedule process of CGF
  - Explain the exploration and exploitation in CGF
  - Point out how to improve greybox fuzzing

• **EcoFuzz**
  - Search strategy (Self-transition-based Probability Estimation Method)
  - Power schedule (Adaptive Average-cost-based Power Schedule)

• Comprehensive evaluation:
  - Attain 214% of the path coverage of AFL with reducing 32% test cases
  - Identifying 12 vulnerabilities
Conclusion

- **VAMAB:**
  - Model the schedule process of CGF
  - Explain the exploration and exploitation in CGF
  - Point out how to improve greybox fuzzing

- **EcoFuzz:**
  - Search strategy (Self-transition-based Probability Estimation Method)
  - Power schedule (Adaptive Average-cost-based Power Schedule)

- **Comprehensive evaluation:**
  - Attain 214% of the path coverage of AFL with reducing 32% test cases
  - Identifying 12 vulnerabilities
Conclusion

• VAMAB:
  • Model the schedule process of CGF
  • Explain the exploration and exploitation in CGF
  • Point out how to improve greybox fuzzing

• EcoFuzz:
  • Search strategy (Self-transition-based Probability Estimation Method)
  • Power schedule (Adaptive Average-cost-based Power Schedule)

• Comprehensive evaluation:
  • Attain 214% of the path coverage of AFL with reducing 32% test case
  • Identifying 12 vulnerabilities
Conclusion

• VAMAB:
  - Model the schedule process of CGF
  - Explain the exploration and exploitation in CGF
  - Point out how to improve greybox fuzzing

• EcoFuzz:
  - Search strategy (Self-transition-based Probability Estimation Method)
  - Power schedule (Adaptive Average-cost-based Power Schedule)

• Comprehensive evaluation:
  - Attain 214% of the path coverage of AFL with reducing 32% test case
  - Identifying 12 vulnerabilities
Thank you!

If you have some questions about our work, welcome to contact us!

Email: yuetai17@nudt.edu.cn
EcoFuzz: https://github.com/MoonLight-SteinsGate/EcoFuzz

National University of Defense Technology