SyML: Guiding Symbolic Execution Toward Vulnerable States Through Pattern Learning

Nicola Ruaro, Lukas Dresel, Kyle Zeng, Tiffany Bao, Mario Polino, Andrea Continella, Stefano Zanero, Christopher Kruegel, Giovanni Vigna
Dynamic Symbolic Execution?
Dynamic Symbolic Execution
Dynamic?

Emulated Environment
(Replayability)

def foo(x, y):
    z = 2*y;
    if (z == x):
        if (x > y+10):
            ERROR;
Symbolic?

def foo(x, y):
    z = 2*y;
    if (z == x):
        if (x > y+10):
            ERROR;
    α_z == α_x
    α_z != α_x
    α_x > α_y + 10
    ERROR

Abstract Domain
(Semantic Insight)
Dynamic Symbolic Execution

Emulated Environment
(Replayability)

```
def foo(x, y):
    z = 2*y;
    if (z == x):
        if (x > y+10):
            ERROR;
```

Abstract Domain
(Semantic Insight)

- Program Verification
- Vulnerability Analysis
- Exploit Generation
- Test-case Generation
- De-obfuscation
- ...

RAiD 2021
Dynamic Symbolic Execution

Emulated Environment
(Replayability)

```
def foo (x, y):
    z = 2*y;
    if (z == x):
        if (x > y+10):
            ERROR;
```

Abstract Domain
(Semantic Insight)

- Program Verification
- **Vulnerability Analysis**
- Exploit Generation
- Test-case Generation
- De-obfuscation
- ...
Path Explosion Problem

N Conditional Nodes
Path Explosion Problem

N Conditional Nodes
$2^N$ Execution Paths

Limit exploration to a selected subset of execution paths
State-of-the-art

1. Symbolic-Assisted Fuzzing (Driller)
2. Under-Constrained Symbolic Execution
3. Merging Execution Paths (Veritesting)
4. Interleaved Symbolic Execution (Symbion)
5. Path Prioritization
1. Symbolic-Assisted Fuzzing (Driller)
2. Under-Constrained Symbolic Execution
3. Merging Execution Paths (Veritesting)
4. Interleaved Symbolic Execution (Symbion)

5. **Path Prioritization**
   A. Classic Tree Traversal
      • Depth First
      • Breadth First
      • Random
   B. Heuristic-Based
      • Loop Exhaustion
      • Coverage Optimization
      • ...
State-of-the-art

1. Symbolic-Assisted Fuzzing (Driller)
2. Under-Constrained Symbolic Execution
3. Merging Execution Paths (Veritest)
4. Interleaved Symbolic Execution (Symbion)

5. Path Prioritization
   A. Classic Tree Traversal
      • Depth First
      • Breadth First
      • Random
   B. Heuristic-Based
      • Loop Exhaustion
      • Coverage Optimization
      • ...

Shallow and Vulnerability-specific
Approach
Intuition

- More **coverage** != more **bugs**
- Replicate the expertise of a **human analyst**
- Similar bugs == **similar patterns**
  (API calls, complex functions, etc.)
- Find **interesting execution contexts**
Approach Overview

Stage 1: Concrete Tracing

Stage 2: (Dynamic) Symbolic Tracing

Stage 3: Training

Stage 4: Prioritization

= angr
Stage 1: Concrete Tracing

- Dataset (binaries and known vulnerabilities)
- Run binary inside the QEMU emulator
- Send crashing input
- Monitor the execution
- Collect execution traces
Stage 2: Symbolic Tracing

- **Static analysis** *(CFG, symbols, etc.)*
- Execute in angr
- **Synchronize** execution with recorded trace
- At every conditional node:
  - Create 2 new training points
  - **Extract features**

```
branch visits

- centrality
- function
- community
- ...
```

```
syscalls
- registers
- memory
- ...
```

![Diagram of the process flow](image)
Stage 3: Training

Clean Dataset:
- Numerical features
- Categorical features

Models: Log. Regression, SVM, Dec. Tree, etc.
Metrics: Accuracy, Coverage, F-1, etc.
Cross Validation: Leave-One-Out
Example
Stage 1: Concrete Tracing

- Crash inputs
- Binaries
- Concrete tracing

- DSE tracing
  - Static analysis
  - Feature extraction

- Training
  - Model

- Prioritization
  - Static analysis
  - Feature extraction
  - Crash monitoring

- Prioritization strategies
Stage 1: Concrete Tracing

```
Crash inputs
Binaries
```

```
Concrete tracing
```

```
DSE tracing
Static analysis
Feature extraction
```

```
Training
Model
```

```
Prioritization strategies
```

```
Static analysis
Feature Extraction
Crash monitoring
```

```
Prioritization
```

QEMU

EXE  EXE  EXE
Stage 1: Concrete Tracing

**INPUT:** A!@^F^J%$#@!~(

**TRACE:** 1, 2, 5, 6, 8, 5, 6, 8, 13 ..

**QEMU**

**TRACE:** 1, 2, 5, 6, 8, 5, 6, 8, 13 ..
Stage 2: Symbolic Tracing

1, 2, 5, 6, 8, 5, 6, 8, 13 ..
Stage 2: Symbolic Tracing

1, 2, 5, 6, 8, 5, 6, 8, 13 ..

+ STATIC INFO

Crash inputs
Binaries
Concrete tracing

DSE tracing
Static analysis
Feature extraction

Training
Prioritization strategies
Model

Prioritization
Static analysis
Feature Extraction
Crash monitoring
Stage 2: Symbolic Tracing

1, 2, 5, 6, 8, 5, 6, 8, 13 ..

+ STATIC INFO

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>111</td>
<td>12</td>
<td>0.0929</td>
<td>{'transmit', '_terminate'}</td>
</tr>
<tr>
<td>TRUE</td>
<td>117</td>
<td>13</td>
<td>0.0112</td>
<td>{'_terminate', 'receive'}</td>
</tr>
</tbody>
</table>
Stage 3: Training

<table>
<thead>
<tr>
<th></th>
<th>FALSE</th>
<th>111</th>
<th>12</th>
<th>0.0929</th>
<th>{'transmit', '_terminate'}</th>
<th>{'<strong>ne</strong>(SYM,CONCR)'}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TRUE</td>
<td>117</td>
<td>13</td>
<td>0.0112</td>
<td>{'_terminate', 'receive'}</td>
<td>{'CONCR'}</td>
</tr>
</tbody>
</table>
Stage 3: Training

<table>
<thead>
<tr>
<th>TRUE</th>
<th>117</th>
<th>13</th>
<th>0.0112</th>
<th>{'terminate', 'receive'}</th>
<th>{'CONCR'}</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>111</td>
<td>12</td>
<td>0.0929</td>
<td>{'transmit', '_terminate'}</td>
<td>{'<strong>ne</strong>(SYM,CONCR)'}</td>
</tr>
</tbody>
</table>

Numerical  Categorical

0 111 12 0.0929
1 117 13 0.0112

Crash inputs

Concrete tracing

DSE tracing

Static analysis

Feature extraction

Training

Model

Prioritization strategies

Prioritization

Static analysis

Feature Extraction

Crash monitoring
Stage 3: Training

<table>
<thead>
<tr>
<th>FALSE</th>
<th>111</th>
<th>12</th>
<th>0.0929</th>
<th>{'transmit', '__terminate'}</th>
<th>{'<strong>ne</strong>(SYM,CONCR)'}</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>117</td>
<td>13</td>
<td>0.0112</td>
<td>{'__terminate', 'receive'}</td>
<td>{'CONCR'}</td>
</tr>
</tbody>
</table>

Accuracy? Coverage? Time-to-Score?

XGBoost model

RAiD 2021
Stage 4: Prioritization

XGBoost model

Fast strategy

\[ \text{next} = \arg \max_{p \in \text{active}} \{ \text{score}(p) \} \]

Balanced strategy

\[ \Pr(\text{next} = p) = \text{score}(p) \]

Crash inputs

Binaries

Concrete tracing

DSE tracing

Static analysis

Feature extraction

Training

Model

Prioritization strategies

Prioritization

Static analysis

Feature Extraction

Crash monitoring

RAiD 2021
Stage 4: Prioritization

XGBoost model

Fast strategy
\[ \text{next} = \arg \max_{p \in \text{active}} \{ \text{score}(p) \} \]

Balanced strategy
\[ \Pr(\text{next} = p) = \text{score}(p) \]

Score

Crash inputs \rightarrow Concrete tracing

Binaries

DSE tracing

Static analysis \rightarrow Feature extraction

Training

Model

Prioritization strategies

Static analysis

Feature Extraction \rightarrow Crash monitoring

RAiD 2021
Stage 4: Prioritization

XGBoost model

Fast strategy
$$\text{next} = \arg \max_{p \in \text{active}} \{\text{score}(p)\}$$

Balanced strategy
$$\Pr(\text{next} = p) = \frac{\text{score}(p)}{\sum_{p \in \text{active}} \text{score}(p)}$$

Score

Choose

Crash inputs

Concrete tracing

Binaries

DSE tracing

Static analysis

Feature extraction

Training

Model

Prioritization strategies

Prioritization

Static analysis

Feature Extraction

Crash monitoring
Stage 4: Prioritization

XGBoost model

Fast strategy
\[ \text{next} = \arg \max_{p \in \text{active}} \{\text{score}(p)\} \]

Balanced strategy
\[ \Pr(\text{next} = p) = \text{score}(p) \]

Choose

Score

Crash inputs

Concrete tracing

Binaries

DSE tracing

Static analysis

Feature extraction

Training

Model

Prioritization strategies

Prioritization

Static analysis

Feature Extraction

Crash monitoring

RAiD 2021
Evaluation
Experimental Setup

• Reimplement the state-of-the-art in a **unified framework** (angr)
  • AEG Loop Exhaustion
  • KLEE Coverage Optimization
  • KLEE Random

• Binaries and crashing inputs
  • **CGC Dataset**
  • 3 real-world **Linux CVEs** (transfer learning)

• 1 Binary per CPU Core (3.6GHz)
• Run and monitor for 24 hours
• **Check and classify crashes**
Dataset

- **CGC dataset** (binaries and known vulnerabilities)
  - Statically compiled x86 binaries
  - Semantics equivalent to Linux binaries
  - Running on DECREE—a different OS with a smaller set of system calls

- **Linux CVEs**
  - CVE-2004-1261 (asp2php)
  - CVE-2004-1288 (o3read)
  - CVE-2004-1292 (ringtonetools)
## Model Choice

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
<th>Accuracy</th>
<th>Trace Coverage</th>
<th>Time-to-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogRegr</td>
<td>77%</td>
<td>66%</td>
<td>73%</td>
<td>0.01s</td>
</tr>
<tr>
<td>LinDiscr</td>
<td>76%</td>
<td>68%</td>
<td>75%</td>
<td>0.01s</td>
</tr>
<tr>
<td>KNN</td>
<td>79%</td>
<td>63%</td>
<td>70%</td>
<td>0.1s</td>
</tr>
<tr>
<td>SVM</td>
<td>82%</td>
<td>76%</td>
<td>72%</td>
<td>0.04s</td>
</tr>
<tr>
<td>MLP</td>
<td>81%</td>
<td>80%</td>
<td>68%</td>
<td>0.04s</td>
</tr>
<tr>
<td>DecisionTree</td>
<td>85%</td>
<td>80%</td>
<td>78%</td>
<td>0.02s</td>
</tr>
<tr>
<td>RandomForest</td>
<td>92%</td>
<td>90%</td>
<td>90%</td>
<td>0.32s</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>93%</td>
<td>91%</td>
<td>83%</td>
<td>0.02s</td>
</tr>
<tr>
<td>XGBoost</td>
<td>94%</td>
<td>93%</td>
<td>91%</td>
<td>0.2s</td>
</tr>
</tbody>
</table>
## Model Choice

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
<th>Accuracy</th>
<th>Trace Coverage</th>
<th>Time-to-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogRegr</td>
<td>77%</td>
<td>66%</td>
<td>73%</td>
<td>0.01s</td>
</tr>
<tr>
<td>LinDiscr</td>
<td>76%</td>
<td>68%</td>
<td>75%</td>
<td>0.01s</td>
</tr>
<tr>
<td>KNN</td>
<td>79%</td>
<td>63%</td>
<td>70%</td>
<td>0.1s</td>
</tr>
<tr>
<td>SVM</td>
<td>82%</td>
<td>76%</td>
<td>72%</td>
<td>0.04s</td>
</tr>
<tr>
<td>MLP</td>
<td>81%</td>
<td>80%</td>
<td>68%</td>
<td>0.04s</td>
</tr>
<tr>
<td>DecisionTree</td>
<td>85%</td>
<td>80%</td>
<td>78%</td>
<td>0.02s</td>
</tr>
<tr>
<td>RandomForest</td>
<td>92%</td>
<td>90%</td>
<td>90%</td>
<td>0.32s</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>93%</td>
<td>91%</td>
<td>83%</td>
<td>0.02s</td>
</tr>
<tr>
<td>XGBoost</td>
<td>94%</td>
<td>93%</td>
<td>91%</td>
<td>0.2s</td>
</tr>
</tbody>
</table>
## Model Choice

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
<th>Accuracy</th>
<th>Trace Coverage</th>
<th>Time-to-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogRegr</td>
<td>77%</td>
<td>66%</td>
<td>73%</td>
<td>0.01s</td>
</tr>
<tr>
<td>LinDiscr</td>
<td>76%</td>
<td>68%</td>
<td>75%</td>
<td>0.01s</td>
</tr>
<tr>
<td>KNN</td>
<td>79%</td>
<td>63%</td>
<td>70%</td>
<td>0.1s</td>
</tr>
<tr>
<td>SVM</td>
<td>82%</td>
<td>76%</td>
<td>72%</td>
<td>0.04s</td>
</tr>
<tr>
<td>MLP</td>
<td>81%</td>
<td>80%</td>
<td>68%</td>
<td>0.04s</td>
</tr>
<tr>
<td>DecisionTree</td>
<td>85%</td>
<td>80%</td>
<td>78%</td>
<td>0.02s</td>
</tr>
<tr>
<td>RandomForest</td>
<td>92%</td>
<td>90%</td>
<td>90%</td>
<td>0.32s</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>93%</td>
<td>91%</td>
<td>83%</td>
<td>0.02s</td>
</tr>
<tr>
<td>XGBoost</td>
<td>94%</td>
<td>93%</td>
<td>91%</td>
<td>0.2s</td>
</tr>
</tbody>
</table>

Performance constraints:

- **Simpler/Faster model**
# Model Choice

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
<th>Accuracy</th>
<th>Trace Coverage</th>
<th>Time-to-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogRegr</td>
<td>77%</td>
<td>66%</td>
<td>73%</td>
<td>0.01s</td>
</tr>
<tr>
<td>LinDiscr</td>
<td>76%</td>
<td>68%</td>
<td>75%</td>
<td>0.01s</td>
</tr>
<tr>
<td>KNN</td>
<td>79%</td>
<td>63%</td>
<td>70%</td>
<td>0.1s</td>
</tr>
<tr>
<td>SVM</td>
<td>82%</td>
<td>76%</td>
<td>72%</td>
<td>0.04s</td>
</tr>
<tr>
<td>MLP</td>
<td>81%</td>
<td>80%</td>
<td>68%</td>
<td>0.04s</td>
</tr>
<tr>
<td>DecisionTree</td>
<td>85%</td>
<td>80%</td>
<td>78%</td>
<td>0.02s</td>
</tr>
<tr>
<td>RandomForest</td>
<td>92%</td>
<td>90%</td>
<td>90%</td>
<td>0.32s</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>93%</td>
<td>91%</td>
<td>83%</td>
<td>0.02s</td>
</tr>
<tr>
<td>XGBoost</td>
<td>94%</td>
<td>93%</td>
<td>91%</td>
<td>0.2s</td>
</tr>
</tbody>
</table>

Performance constraints:
- **Simpler/Faster model**

RAiD 2021
Comparison Results

Comparison of crash rates over time for different tools:
- SyML
- AEG Loop Exhaustion
- KLEE Random
- KLEE Coverage Optimize

The graph shows the number of crashes over minutes, with SyML consistently having the highest crash rate.
Comparison Results

- More
- Different
Model Analysis

Features Importance

Prediction Scores Distribution
Model Analysis

Features Importance

Prediction Scores Distribution
Model Analysis

Features Importance

Prediction Scores Distribution
Transfer Learning

- DSE inaccuracies make it hard to re-trace Linux binaries
- CGC semantics are analogous to the Linux x86 semantics
  - This allows us to transfer some of the knowledge learned from the larger CGC dataset to the Linux dataset

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>RandomForest (Linux)</td>
<td>63%</td>
<td>70%</td>
</tr>
<tr>
<td>AdaBoost (Linux)</td>
<td>63%</td>
<td>63%</td>
</tr>
<tr>
<td>XGBoost (Linux)</td>
<td>51%</td>
<td>56%</td>
</tr>
<tr>
<td>XGBoost (CGC)</td>
<td>69%</td>
<td>54%</td>
</tr>
<tr>
<td>XGBoost (CGC+Linux)</td>
<td>77%</td>
<td>66%</td>
</tr>
</tbody>
</table>
Conclusion

• We propose a **novel path prioritization** approach, leveraging supervised learning algorithms to steer DSE and reach interesting paths

• We evaluate our approach on the CGC dataset, **outperforming prior work** with more (and different) vulnerabilities

• We effectively **transfer the models learned** on the CGC dataset to achieve a better prediction accuracy on 3 real-world CVEs affecting Linux

Future Work

• Train on a **large dataset of Linux binaries** using a different re-tracing framework

• Adapt and apply to guide **hybrid fuzzing**
Thank You!

SyML: Guiding Symbolic Execution Toward Vulnerable States Through Pattern Learning

Nicola Ruaro, Lukas Dresel, Kyle Zeng, Tiffany Bao, Mario Polino, Andrea Continella, Stefano Zanero, Christopher Kruegel, Giovanni Vigna

Nicola Ruaro - ruaronicola@ucsb.edu