Guide Me to Exploit: Assisted ROP Exploit Generation for ActionScript Virtual Machine

Fadi Yilmaz, Meera Sridhar, and Wontae Choi

36th Annual Computer Security Applications Conference (ACSAC’20)
December 7-11, 2020, Online

Work supported in part by National Science Foundation Grant No. 1566321
Motivation of Automated Exploit Generation (AEG)

• Monitoring the execution of exploit scripts is crucial
  • Underlying weaknesses of target applications
  • Unorthodox methods to exploit vulnerabilities
AEG

- Determining the *exploitability* [Younis et al. SQJ’16]
- Explores all possible execution paths [Avgerinos et al. NDSS’11]
AEG Components

- **Fuzzer** [Miller et al. ACM’90, Jayaraman et al. NFM’09, Rawat et al. NDSS’17]
  - Explores only one execution path in one run

![Diagram of state transitions](image-url)
AEG Components

• Fuzzer
  • Explores only one execution path in one run
AEG Components

- Symbolic Execution [King et al. ACM’76]
  - Explores all execution paths symbolically in one run
AEG Components

• Symbolic Execution
  • Explores all execution paths symbolically in one run
AEG Components

- Symbolic Execution
  - Explores all execution paths symbolically in one run
AEG Components

- Symbolic Execution
  - Explores all execution paths symbolically in one run
AEG Components

- Symbolic Execution
  - Explores all execution paths symbolically in one run
AEG Components

**Fuzzer**
- Fast, easy to build
- Complex grammar rules for executables
- Infinitesimal chance

**Symbolic Execution**
- Explores all execution paths in one run
- The path-explosion problem

**Pros**
- Fast, easy to build
- Explores all execution paths in one run

**Cons**
- Complex grammar rules for executables
- Infinitesimal chance
GUIDEXP : A Prototype Semi-Automatic AEG Tool

- The first *guided* (semi-automatic) exploit generation tool for the AVM implementations

- Does not rely on a fuzzer or a symbolic execution tool
Intuition Behind Target Exploit Generation

- Structure of our target exploit
- Exploit pattern
Exploit Subgoals

- A search space
  - Set of instructions

- An invariant
  - The test
Preparation: Defining Exploit Subgoals, Inputs & Outputs

Phase 1
- Target Exploit Subgoals
  - $\tau_1, \tau_2, \ldots, \tau_n$
  - $\tau_i$ (Exploit Subgoal)

Phase 2
- Parser
- Search Space($\tau_i$)
- Exploit Subgoal Parser
- Invariant($\tau_i$)
- OUR AEG TOOL
- Decision(Candidate Slice, $\tau_i$)
- The Exploit

Phase 3
- Code Generator
- AST
- Candidate Slice
- Invariant Validator
- Next Exploit Subgoal
- Exploit Subgoal Manager
- Checkpoint($\tau_i$)
Preparation: Defining Exploit Subgoals, Inputs & Outputs
Phase 1: Exploit Subgoal Processing

Reads exploit subgoals and invariant

Phase 1

Target Exploit Subgoals

\( \tau_1, \tau_2, \ldots, \tau_n \)

\( \tau_i \) (Exploit Subgoal)

Search Space(\( \tau_i \))

Exploit Subgoal Parser

Invariant(\( \tau_i \))

OUR AEG TOOL

Phase 2

Trigger Slice

Parser

AST

Code Generator

Invariant Slice

Invariant Validator

Decision(Candidate Slice, \( \tau_i \))

Checkpoint(\( \tau_i \))

Phase 3

Next Exploit Subgoal

Exploit Subgoal Manager

Decision(Candidate Slice, \( \tau_i \))

The Exploit

Reads exploit subgoals and invariant
Phase 2: Generates Candidate Slices and Validating Invariant
Phase 2: Generates Candidate Slices and Validating Invariant

Phase 1

Target Exploit Subgoals

\( \tau_1 \), \( \tau_2 \), \ldots, \( \tau_n \)

Phase 2

\( \tau_i \) (Exploit Subgoal)

Parses the trigger slice

Next Exploit Subgoal

Parser

Trigger Slice

Search Space(\( \tau_i \))

Exploit Subgoal Parser

Exploit Subgoal

Parser

Code Generator

Invariant(\( \tau_i \))

Invariant Validator

Decision(Candidate Slice, \( \tau_i \))

Checkpoint(\( \tau_i \))

Exploit Subgoal Manager

The Exploit

OUR AEG TOOL
Phase 2: Generates Candidate Slices and Validating Invariant

Phase 1
- Target Exploit Subgoals
  - $\tau_1$, $\tau_2$, ..., $\tau_n$

Phase 2
- Trigger Slice
  - Parser
  - Code Generator
  - Candidate Slice
  - Invariant Validator

Phase 3
- Next Exploit Subgoal
  - Exploit Subgoal Manager
  - Decision(Candidate Slice, $\tau_i$)

Our AEG TOOL

Generates candidate slices
Phase 2: Generates Candidate Slices and Validating Invariant

- **Phase 1**: Target Exploit Subgoals
  - \(\tau_1\), \(\tau_2\), \(\ldots\), \(\tau_n\)

- **Phase 2**: Trigger Slice
  - Parser
  - Search Space(\(\tau_i\))
  - Exploit Subgoal Parser
  - Invariant(\(\tau_i\))
  - OUR AEG TOOL
  - Code Generator
  - Candidate Slice
  - Invariant Validator
  - Decision(Candidate Slice, \(\tau_i\))

- **Phase 3**: Next Exploit Subgoal
  - Exploit Subgoal Manager
  - Checkpoint(\(\tau_i\))

Executes and tests candidate slices
Phase 3: Evaluating Candidate Slices

Evaluates the execution of candidate slices
Optimization Techniques

- **Exploit Deconstruction (~$10^{45}$)**
  - Synthesizing smaller exploit subgoals in sequence
Optimization Techniques

- **Exploit Deconstruction ($\sim 10^{45}$)**
  - Synthesizing smaller exploit subgoals in sequence
Optimization Techniques

- **Exploit Deconstruction ($\sim 10^{45}$)**
  - Synthesizing smaller exploit subgoals in sequence
Optimization Techniques

- **Exploit Deconstruction ($\sim 10^{45}$)**
  - Synthesizing smaller exploit subgoals in sequence
Optimization Techniques

• Exploit Deconstruction ($\sim 10^{45}$)
  • Synthesizing smaller exploit subgoals in sequence

Vulnerability State

$n_1$

Exploit State

Level 1

Level 2

Level $k$

Level $k + 1$

Level $r$
Optimization Techniques

- **Exploit Deconstruction ($\sim 10^{45}$)
  - Synthesizing smaller exploit subgoals in sequence
Optimization Techniques

- **Exploit Deconstruction (~$10^{45}$)**
  - Synthesizing smaller exploit subgoals in sequence

![Diagram showing levels and vulnerability states](image)
Optimization Techniques

• Exploit Deconstruction ($\sim 10^{45}$)
  • Synthesizing smaller exploit subgoals in sequence

• Operand Stack Verification (98.78%)
  • Disqualifies candidate slices that perform illegal stack operations
Optimization Techniques

- Exploit Deconstruction ($\sim 10^{45}$)
  - Synthesizing smaller exploit subgoals in sequence

- Operand Stack Verification (98.78%)  
  - Disqualifies candidate slices that perform illegal stack operations

- Instruction Tiling ($\sim 10^{13.5}$)  
  - Glues instructions together to obtain more coarse-grained, meaningful instruction chains
Optimization Techniques

- **Exploit Deconstruction ($\sim 10^{45}$)**
  - Synthesizing smaller exploit subgoals in sequence

- **Operand Stack Verification (98.78%)**
  - Disqualifies candidate slices that perform illegal stack operations

- **Instruction Tiling ($\sim 10^{13.5}$)**
  - Glues instructions together to obtain more coarse-grained, meaningful instruction chains

- **Feedback from the AVM (58%)**
  - Detects error-raising instruction prefixes
Experimental Results -1

- Two sets of experiments
- CVE-2015-5119
- Open-source core implementation
- Closed-source standalone Flash Player Debugger

<table>
<thead>
<tr>
<th>Exploit Subgoal</th>
<th>Number of Generated Candidate Slices</th>
<th>Number of Executed Candidate Slices</th>
<th>Percentage of Executed Candidate Slices</th>
<th>Synthesizing Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corrupting a Buffer Space Implicitly</td>
<td>2,396,744</td>
<td>12,229</td>
<td>0.51</td>
<td>9.35</td>
</tr>
<tr>
<td>Spraying Helper Elements</td>
<td>19,173,952</td>
<td>73,997</td>
<td>0.38</td>
<td>55.90</td>
</tr>
<tr>
<td>Locating Sprayed Elements</td>
<td>37,448</td>
<td>357</td>
<td>0.95</td>
<td>1.72</td>
</tr>
<tr>
<td>Disclosing the Offset of the Located Elements</td>
<td>55,345,757</td>
<td>282,392</td>
<td>0.51</td>
<td>138.26</td>
</tr>
<tr>
<td>Corrupting the Disclosed Buffer</td>
<td>4,793,488</td>
<td>21,591</td>
<td>0.45</td>
<td>17.03</td>
</tr>
<tr>
<td>Locating ELF Object Files</td>
<td>19,173,952</td>
<td>81,545</td>
<td>0.42</td>
<td>57.12</td>
</tr>
<tr>
<td>Locating libc Libraries</td>
<td>55,345,757</td>
<td>278,385</td>
<td>0.50</td>
<td>138.05</td>
</tr>
<tr>
<td>Locating Executable Segment</td>
<td>76,695,808</td>
<td>379,587</td>
<td>0.49</td>
<td>199.78</td>
</tr>
<tr>
<td>Locating Gadgets and Building the ROP Chain</td>
<td>435,848,049</td>
<td>1,648,451</td>
<td>0.37</td>
<td>240.92</td>
</tr>
</tbody>
</table>

Total: 858.13 (14m 18.13s)

<table>
<thead>
<tr>
<th>Exploit Subgoal</th>
<th>Number of Generated Candidate Slices</th>
<th>Number of Executed Candidate Slices</th>
<th>Percentage of Executed Candidate Slices</th>
<th>Synthesizing Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corrupting a Buffer Space Implicitly</td>
<td>2,396,744</td>
<td>29,167</td>
<td>1.21</td>
<td>605.58</td>
</tr>
<tr>
<td>Spraying Helper Elements</td>
<td>19,173,952</td>
<td>210,225</td>
<td>1.09</td>
<td>3,895.64</td>
</tr>
<tr>
<td>Locating Sprayed Elements</td>
<td>37,448</td>
<td>769</td>
<td>2.05</td>
<td>12.76</td>
</tr>
<tr>
<td>Disclosing the Offset of the Located Elements</td>
<td>55,345,757</td>
<td>508,339</td>
<td>0.91</td>
<td>6,845.86</td>
</tr>
<tr>
<td>Corrupting the Disclosed Buffer</td>
<td>4,793,488</td>
<td>41,342</td>
<td>0.86</td>
<td>963.86</td>
</tr>
<tr>
<td>Locating ELF Object Files</td>
<td>19,173,952</td>
<td>201,852</td>
<td>1.05</td>
<td>3,364.89</td>
</tr>
<tr>
<td>Locating libc Libraries</td>
<td>55,345,757</td>
<td>459,336</td>
<td>0.82</td>
<td>6,276.25</td>
</tr>
<tr>
<td>Locating Executable Segment</td>
<td>76,695,808</td>
<td>706,031</td>
<td>0.92</td>
<td>9,546.07</td>
</tr>
<tr>
<td>Locating Gadgets and Building the ROP Chain</td>
<td>435,848,049</td>
<td>2,954,400</td>
<td>0.67</td>
<td>11,512.47</td>
</tr>
</tbody>
</table>

Total: 43,023.38 (11h 57m 03.38s)
Experimental Results -1

- The difference is due to starting/closing of the Flash Player
- It takes 85ms on average, equivalent to 89% of the time

<table>
<thead>
<tr>
<th>Exploit Subgoal</th>
<th>Number of Generated Candidate Slices</th>
<th>Number of Executed Candidate Slices</th>
<th>Percentage of Executed Candidate Slices</th>
<th>Synthesizing Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corrupting a Buffer Space Implicitly</td>
<td>2,396,744</td>
<td>12,229</td>
<td>0.51</td>
<td>9.35</td>
</tr>
<tr>
<td>Spraying Helper Elements</td>
<td>19,173,952</td>
<td>73,997</td>
<td>0.38</td>
<td>55.90</td>
</tr>
<tr>
<td>Locating Sprayed Elements</td>
<td>37,448</td>
<td>337</td>
<td>0.95</td>
<td>1.72</td>
</tr>
<tr>
<td>Disclosing the Offset of the Located Elements</td>
<td>55,345,757</td>
<td>282,392</td>
<td>0.51</td>
<td>138.26</td>
</tr>
<tr>
<td>Corrupting the Disclosed Buffer</td>
<td>4,793,488</td>
<td>21,591</td>
<td>0.45</td>
<td>17.03</td>
</tr>
<tr>
<td>Locating ELF Object Files</td>
<td>19,173,952</td>
<td>81,545</td>
<td>0.42</td>
<td>57.12</td>
</tr>
<tr>
<td>Locating libc Libraries</td>
<td>55,345,757</td>
<td>278,385</td>
<td>0.50</td>
<td>138.05</td>
</tr>
<tr>
<td>Locating Executable Segment</td>
<td>76,695,808</td>
<td>379,587</td>
<td>0.49</td>
<td>199.78</td>
</tr>
<tr>
<td>Locating Gadgets and Building the ROP Chain</td>
<td>435,848,049</td>
<td>1,648,451</td>
<td>0.37</td>
<td>240.92</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>858.13 (14m 18.13s)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Corrupting a Buffer Space Implicitly</td>
<td>2,396,744</td>
<td>29,167</td>
<td>1.21</td>
<td>605.58</td>
</tr>
<tr>
<td>Spraying Helper Elements</td>
<td>19,173,952</td>
<td>210,225</td>
<td>1.09</td>
<td>3,895.64</td>
</tr>
<tr>
<td>Locating Sprayed Elements</td>
<td>37,448</td>
<td>769</td>
<td>2.05</td>
<td>12.76</td>
</tr>
<tr>
<td>Disclosing the Offset of the Located Elements</td>
<td>55,345,757</td>
<td>508,339</td>
<td>0.91</td>
<td>6,845.86</td>
</tr>
<tr>
<td>Corrupting the Disclosed Buffer</td>
<td>4,793,488</td>
<td>41,342</td>
<td>0.86</td>
<td>963.86</td>
</tr>
<tr>
<td>Locating ELF Object Files</td>
<td>19,173,952</td>
<td>201,852</td>
<td>1.05</td>
<td>3,364.89</td>
</tr>
<tr>
<td>Locating libc Libraries</td>
<td>55,345,757</td>
<td>459,336</td>
<td>0.82</td>
<td>6,276.25</td>
</tr>
<tr>
<td>Locating Executable Segment</td>
<td>76,695,808</td>
<td>706,031</td>
<td>0.92</td>
<td>9,546.07</td>
</tr>
<tr>
<td>Locating Gadgets and Building the ROP Chain</td>
<td>435,848,049</td>
<td>2,954,400</td>
<td>0.67</td>
<td>11,512.47</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>43,023.38 (11h 57m 03.38s)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Experimental Results -II

- Generating exploit scripts for different vulnerabilities with the closed-source debugger

<table>
<thead>
<tr>
<th>Selected Vulnerabilities</th>
<th>Synthesizing Time</th>
<th>Flash Player Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2015-5119</td>
<td>11h 57m 03.38s</td>
<td>v11.2.202.262</td>
</tr>
<tr>
<td>CVE-2013-0634</td>
<td>12h 09m 14.50s</td>
<td>v11.2.202.262</td>
</tr>
<tr>
<td>CVE-2014-0502</td>
<td>12h 54m 15.19s</td>
<td>v11.2.202.262</td>
</tr>
<tr>
<td>CVE-2014-0515</td>
<td>12h 51m 26.67s</td>
<td>v11.2.202.262</td>
</tr>
<tr>
<td>CVE-2014-0556</td>
<td>12h 08m 35.29s</td>
<td>v11.2.202.262</td>
</tr>
<tr>
<td>CVE-2015-0311</td>
<td>11h 56m 19.10s</td>
<td>v11.2.202.262</td>
</tr>
<tr>
<td>CVE-2015-0313</td>
<td>12h 20m 47.98s</td>
<td>v11.2.202.442</td>
</tr>
<tr>
<td>CVE-2015-0359</td>
<td>11h 05m 05.61s</td>
<td>v11.2.202.262</td>
</tr>
<tr>
<td>CVE-2015-3090</td>
<td>12h 01m 33.16s</td>
<td>v11.2.202.262</td>
</tr>
<tr>
<td>CVE-2015-3105</td>
<td>13h 25m 46.80s</td>
<td>v11.2.202.262</td>
</tr>
<tr>
<td>CVE-2015-5122</td>
<td>12h 07m 02.59s</td>
<td>v11.2.202.262</td>
</tr>
</tbody>
</table>
Limitations and Challenges

- **Compatibility**
  - Different memory offset
  - Calling external libraries

- **Debuggers**
  - PoCs perform their malicious activities *implicitly*

- **Accuracy of the exploit subgoals**
  - Having unnecessary instructions significantly increases the time (combinatorial rate)
Thank You

Fadi Yilmaz  
UNC Charlotte  
fyilmaz@uncc.edu

Meera Sridhar  
UNC Charlotte  
msridhar@uncc.edu

Wontae Choi  
wtchoi.kr@gmail.com
Key References


