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

Fadi Yilmaz, Meera Sridhar, and Wontae Choi

Learning from Authoritative Security Experiment Results (LASER) 2020
Part II
December 8, 2020, Online

presented by
Fadi Yilmaz

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
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

![Diagram showing levels and states]
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

\[ n_1, n_2, ..., n_k \]

Vulnerability State

Level 1

Level 2

Level \( k \)

Level \( k+1 \)

Level \( r \)

Exploit State
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

**Pros**

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

**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
Experimental Results - I

- 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>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>
<tr>
<td><strong>Total</strong>:</td>
<td><strong>858.13</strong></td>
<td><strong>1,685,161</strong></td>
<td><strong>1.18</strong></td>
<td><strong>605.58</strong></td>
</tr>
</tbody>
</table>

<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>
<tr>
<td><strong>Total</strong>:</td>
<td><strong>43,023.38</strong></td>
<td><strong>11h 57m 03.38s</strong></td>
<td><strong>1.15</strong></td>
<td><strong>11,512.47</strong></td>
</tr>
</tbody>
</table>
Experiment

- 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>
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>
Experimental Setup

• Focus
  • General numbers
  • Applicability of the tool

• Goal
  • To demonstrate that our tool is actually useful
Experimental Setup – Cont.

- Two set of experiments
  - Executing candidate slices with open-source AVM
  - Executing candidate slices with closed-source AVM
Experimental Setup – Cont.

• Used artifacts
  • The golden example
    • The only vulnerability in the open-source AVM
    • To explain difference between targeting open-source and closed-source AVM
  • Eleven vulnerabilities collected for a closed-source AVM
    • Includes the golden example
Execution Flow Recap

Code Generator

- Synthesizes
- Search Space

Invariant Validator

- Merges
- Invariant

Open-source AVM

Result

Ins: [208, 197, 65, 55, 45, 103]
Para: [0, 1, 2, 3, 4]
Execution Flow Recap – Cont.

- Candidate slices are written into the file system
- AVM Core is invoked and reads a candidate slice from the file system
- The result is written into a file system and read from the file system
Execution Flow Recap – Cont.

- Candidate slices are written into the file system
- AVM Core is invoked and reads a candidate slice from the file system
- The result is written into a file system and read from the file system
Initial Development

- Measure the initial performance
  - Memory
  - Running time
  - Interaction Cost

- To get something fast and lean enough to be used
Initial Development

• Used Artifact
  • The golden example
  • A single vulnerability from an open-source AVM
    • CVE-2015-5119
      • Details are in our paper
    • Real example
    • Not too complicated
    • Still not too simple
Development Cycle

- Implement a prototype
- Run it on the benchmark
- Evaluate the numbers
- Identify bottlenecks
- Optimize
- Go back to step (1)
Development Cycle - I

- Everything was written into HDD
  - Huge bottleneck
- This is not a part of our algorithm!
- Easy to solve!
  - Ask for an SSD!
    - Not good enough
  - Use VM
Development Cycle -II

- The number of execution paths to explore is too big!

Solution

- Adding search space limitation
Development Cycle - III

- The search may last infinite
- Tested various search target prioritization techniques (DFS or BFS or Random)
- Final decision: BFS
  - Level is limited
Development Cycle - IV

- The search still takes too much time
  - Number of candidate slice is more than billions
- Optimize
  - Lots of type errors happened
    - Feedback optimization
  - Stack simulation
  - Tiling
Initial Development – The Bottom Line

- Golden Example
  - Good Part
    - Iterate extremely fast
    - Identify all the small details of algorithms and artifacts
  - Danger
    - Development can be biased
- Mitigation
  - Use more than one golden example
Actual Evaluation

• What we did
  • Applied our technique to all these examples
  • Showed that everything passes

• What we observed
  • Not biased with \textit{golden example}
  • Performance of our tool with closed-source VM is not as good as it is with open-source VM
Actual Evaluation – The Bottom Line

• We were lucky that we started with a vulnerability in an open-source VM
  • With a closed-source VM, our initial development process could be infeasible

• Generalizing different configurations can be challenging
Manual Intervention

- Evaluating manual intervention is not in our focus

- Our focus is to move from “unable” to “able”

- This is the future work!
Artifacts Borrowed from the Community

- Synthesizes a ROP exploit for given AVM vulnerabilities

- AVM vulnerabilities
  - Exploit databases
    - exploit-db.com
    - Google’s Project Zero*
  - Tech Reports
- We synthesized different exploits

*https://bugs.chromium.org/
Artifacts Borrowed from the Community

• Synthesizes a **ROP exploit** for given **AVM vulnerabilities**

• ROP chain
  • ROPgadget*
  • Locates and build the ROP chain
  • Execute ‘int 0x80’

• We copied the idea

*Jonathan Salwan - https://github.com/JonathanSalwan/ROPgadget*
Intermediate Results

• Development Cycle
  • Many iterations
  • Many results

• Gradually getting faster tool
  • Start with months
  • Down to 15 minutes
Intermediate Results - Optimizations

- Multi-threading
  - Three threads
  - From months to weeks
- Stack Simulation
  - Almost hundred times faster
  - From weeks to hours
Intermediate Results - Optimizations

• Run-time Errors
  • From hours to minutes
  • There are thousands of different run-time error messages*
  • Not all of them is raised
  • Not all of them is useful

What can be learned from your methodology and your experience using your methodology?
Any Failed Attempts

• Not really
  • Aimed to implement more powerful system
  • More optimization techniques
Did you attempt to replicate or reproduce results of earlier research as part of your work?
Future Works

• Need to measure
  • How much human interaction is required
  • How much human expertise is required
    • Can a newbie use the tool?
  • How much effort does our tool save for a seasoned developer
Future Works

• User-study
• Two dimensions of expertise
  • Exploits
  • ActionScript language
• Three level of expertise
  • Newbie
  • Intermediate
  • Seasoned
Thank You

Fadi Yilmaz  
UNC Charlotte  
fyilmaz@uncc.edu

Meera Sridhar  
UNC Charlotte  
msridhar@uncc.edu

Wontae Choi  
wtchoi.kr@gmail.com
Key References


