DEEPVSA: Facilitating Value-set Analysis with Deep Learning for Postmortem Program Analysis

Wenbo Guo, Dongliang Mu, Xinyu (X.Y.) Xing, Min Du, Dawn Song
Pennsylvania State University & UC Berkeley
Background

• Software inevitably contains bugs and vulnerabilities
• Remediation requires accurate diagnosis to the root cause of a software failure
• Failure diagnosis is a lengthy process because
  ➢ Failure occurs at post-deployed stage, at which the bad input is typically not available
  ➢ The information left over for diagnosis is very limited (e.g., a stack trace indicating only part of the functions that have been executed)
Information Available for Crash Diagnosis

• Most broadly used debugging information is still the core dump

• After a program crash, computer system generates a core dump file for a crashing process

• A core dump typically contains
  ➢ The memory snapshot of a crashing process
  ➢ The register values at the time of program crash
  ➢ The execution trace pertaining to the crashing process (i.e., the critical instructions that can be used to restore the entire instruction trace executed prior to crash)
Representative Techniques for Crash Diagnosis

• Backward taint analysis techniques are broadly used for diagnosing root cause of a software failure
  • E.g., REPT [OSDI 18]; POMP [USENIX SEC 17]; CREDAL [CCS 16]

• The basic idea of backward taint analysis is
  • Taking as input the snapshot of the crashing memory as well as the full instruction trace
  • Backpropagating a bad pointer along the execution trace
  • Pinpointing all the instructions pertaining to the crash
Key Challenge of Backward Taint Analysis

• Well-designed backward taint requires analyzing memory alias accurately
• Inaccurate alias analysis could lead to an over-tainted issue
• Over-tainted issue could jeopardize the diagnosis of software crash

Which instructions are the key contributor to the program crash?
Roadmap

- Existing technique and challenges of memory alias analysis
- Our Proposed approach – DEEPVSA
- Experiment Design and Results
- Conclusion
Existing approach for memory alias analysis

- Value-Set Analysis (VSA) typically partitions memory into 3 disjoint memory regions (stack, heap and global)
- VSA determines the regions tied to instructions by using:
  - Instruction semantics
  - Forward data flow analysis
Existing approach for memory alias analysis (cont.)

- Within each region, VSA further estimates the fine-grained region that each instruction attempts to access.
- For each instruction, VSA uses a 3-tuple to represent the fine-grained region it accesses.

```
1 sub esp, 0x14
2 call malloc
......
3 ret
4 mov [eax], test
5 mov [esp+0x8], eax
6 push eax
7 call child
8 push ebp
9 mov ebp, esp
......
```

<table>
<thead>
<tr>
<th>Line</th>
<th>A-loc</th>
<th>Value-set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>esp</td>
<td>(⊥, [-0x14, -0x14], ⊥)</td>
</tr>
<tr>
<td>4</td>
<td>[eax](⊥,[0,0], ⊥)</td>
<td>(test, ⊥, ⊥)</td>
</tr>
<tr>
<td>5</td>
<td>[esp+8](⊥,[-0xC,0xC], ⊥)</td>
<td>(⊥, ⊥, [0, 0])</td>
</tr>
<tr>
<td>6</td>
<td>esp</td>
<td>(⊥,[-0x18,-0x18], ⊥)</td>
</tr>
<tr>
<td></td>
<td>[esp](⊥,[-0x18,-0x18], ⊥)</td>
<td>(⊥, ⊥, [0, 0])</td>
</tr>
<tr>
<td>7</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Software Failure Diagnosis with Facilitation of VSA

1. Construct a confusion matrix

2. Use the matrix to compare the overlaps between memory regions

3. Deem the instructions with nonoverlapped region as non-alias relationships

4. Perform backward taint analysis under the guidance of identified alias relationship
Practical Issues Observed in the Usage of VSA

- Hardware has limited capability in storing all the instructions executed
- The execution trace in a core dump is usually incomplete
- An incomplete trace influences VSA upon its ability to identify memory alias

Which region does \([eax]\) refer to?

```
lea eax, [esp+4]
...
call malloc
...
ret
mov [eax], test
...
mov edx, [0x8050684]
```
Roadmap

• Existing technique and challenges of memory alias analysis
• **Our Proposed approach – DEEPVSA**
• Experiment Design and Results
• Conclusion
Our Proposed Approach -- DEEPVSA

• Inspired by the power of deep learning (DL) in other binary analysis applications, e.g.,
  • Identifying function boundary [USENIX SEC 15]
  • Recognize function type signatures [USENIX SEC 17]
• Use a deep neural network to predict the region that an instruction could access
• Utilize the region information to assist traditional VSA in region identification
Proposed Neural Network Architecture

A sequence of instructions

0x8b ... 0x8b 0x44 0x24 0x0c

push eax ...

mov eax, [esp+0xc]

call [eax]
Proposed Neural Network Architecture (cont.)

- **Instruction Embedding (IE)**
  - Use a word embedding to encode machine code
  - Use a bi-directional LSTM to generate instruction embeddings

- **Identification network**
  - Utilize a bi-directional LSTM to predict the region that an instruction accesses
Roadmap

• Existing technique and challenges of memory alias analysis
• Our Proposed approach – DEEPVSA
• Experiment Design and Results
• Conclusion

Presenter – Wenbo Guo
(http://www.personal.psu.edu/wzg13/)
Experiment Dataset

• Training set
  ➢ ~50 million lines of instructions and their memory accesses (generated from the test cases of 78 unique programs in a package of GNU software)

• Testing set
  ➢ ~1.6 million lines of instructions and their memory accesses (generated from the PoC program of 40 real-world vulnerabilities)
Experiment Design

• Baseline ML models
  ➢ Hidden Markov Model (HMM)
  ➢ Conditional Random Field (CRF)
  ➢ Bi-directional RNN / LSTM / GRU

• Evaluation
  • Model precision, recall and F-measure
  • DEEPVSA vs. VSA
  • DEEPVSA vs. a state-of-the-art tool (POMP)
Results – Learning Model Performance

• Experiment result summary
  ➢ Our approach significantly outperforms traditional ML methods (HMM, CRF)
  ➢ In comparison with other DL approaches, our approach demonstrates slight performance improvement (1% ~ 12%)

<table>
<thead>
<tr>
<th></th>
<th>Global</th>
<th>Heap</th>
<th>Stack</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precision</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HMM</td>
<td>67.99%</td>
<td>53.99%</td>
<td>74.47%</td>
<td>86.56%</td>
</tr>
<tr>
<td>CRF</td>
<td>15.93%</td>
<td>12.31%</td>
<td>62.10%</td>
<td>71.82%</td>
</tr>
<tr>
<td>Bi-RNN</td>
<td>98.63%</td>
<td>72.74%</td>
<td>95.30%</td>
<td>97.39%</td>
</tr>
<tr>
<td>Bi-GRU</td>
<td>90.71%</td>
<td>78.44%</td>
<td>95.11%</td>
<td>98.40%</td>
</tr>
<tr>
<td>Bi-LSTM</td>
<td>89.75%</td>
<td>78.47%</td>
<td>94.98%</td>
<td>97.92%</td>
</tr>
<tr>
<td>Our Model</td>
<td><strong>96.98%</strong></td>
<td><strong>94.62%</strong></td>
<td><strong>98.92%</strong></td>
<td><strong>99.32%</strong></td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HMM</td>
<td>77.52%</td>
<td>39.31%</td>
<td>81.40%</td>
<td>85.28%</td>
</tr>
<tr>
<td>CRF</td>
<td>10.23%</td>
<td>17.88%</td>
<td>51.95%</td>
<td>88.71%</td>
</tr>
<tr>
<td>Bi-RNN</td>
<td>84.17%</td>
<td>83.72%</td>
<td>95.96%</td>
<td>95.43%</td>
</tr>
<tr>
<td>Bi-GRU</td>
<td>88.04%</td>
<td>87.46%</td>
<td>97.59%</td>
<td>95.84%</td>
</tr>
<tr>
<td>Bi-LSTM</td>
<td>91.71%</td>
<td>86.53%</td>
<td>97.16%</td>
<td>95.37%</td>
</tr>
<tr>
<td>Our Model</td>
<td><strong>86.67%</strong></td>
<td><strong>95.99%</strong></td>
<td><strong>98.59%</strong></td>
<td><strong>99.45%</strong></td>
</tr>
<tr>
<td><strong>F1 Score</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HMM</td>
<td>72.44%</td>
<td>45.50%</td>
<td>77.78%</td>
<td>85.92%</td>
</tr>
<tr>
<td>CRF</td>
<td>12.46%</td>
<td>14.57%</td>
<td>56.57%</td>
<td>79.38%</td>
</tr>
<tr>
<td>Bi-RNN</td>
<td>90.83%</td>
<td>77.85%</td>
<td>95.63%</td>
<td>96.40%</td>
</tr>
<tr>
<td>Bi-GRU</td>
<td>89.35%</td>
<td>82.71%</td>
<td>96.33%</td>
<td>97.10%</td>
</tr>
<tr>
<td>Bi-LSTM</td>
<td>90.72%</td>
<td>82.30%</td>
<td>96.06%</td>
<td>96.63%</td>
</tr>
<tr>
<td>Our Model</td>
<td><strong>91.54%</strong></td>
<td><strong>95.30%</strong></td>
<td><strong>98.75%</strong></td>
<td><strong>99.39%</strong></td>
</tr>
</tbody>
</table>
Results – Root Cause Diagnosis (DEEPVSA vs. VSA)

• Experiment result summary
  ➢ Execution trace length stored in a core dump influences alias analysis
  ➢ In comparison with VSA, DEEPVSA is more effective in finding the root cause of software crashes

<table>
<thead>
<tr>
<th>Trace length prior to the site of the root cause instruction</th>
<th>Average amount of non-alias pairs (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>VSA</td>
</tr>
<tr>
<td>400</td>
<td>60%</td>
</tr>
<tr>
<td>800</td>
<td>65%</td>
</tr>
<tr>
<td>1600</td>
<td>70%</td>
</tr>
<tr>
<td>3200</td>
<td>75%</td>
</tr>
<tr>
<td>6400</td>
<td>80%</td>
</tr>
<tr>
<td>12800</td>
<td>85%</td>
</tr>
<tr>
<td>19600</td>
<td>90%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total # of cases backward taint pinpoints the root cause of the software crash</th>
<th>DEEPVSA</th>
<th>VSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>37/40</td>
<td></td>
<td>27/40</td>
</tr>
</tbody>
</table>
Results – Root Cause Diagnosis (DEEPVSA vs. POMP)

• About POMP
  • POMP is a postmortem program analysis framework [USENIX SEC 17]
  • POMP diagnoses program crashes using alias analysis and backward taint
  • The alias analysis of POMP relies upon a reverse execution method which is computationally intensive
Results – DEEPVSA vs. POMP (cont.)

- Experiment results summary
  - On average, DEEPVSA is 2~10x faster than POMP
Conclusion

• Conventional VSA suffers from inaccurate alias analysis when an execution trace is incomplete
• VSA combined with deep learning (DEEPVSA) could improve alias analysis in binary analysis and thus facilitate software failure diagnosis
• DEEPVSA could serve as an efficient, effective diagnosis tool to pinpoint the root cause of a software failure
Thank you very much!

Code and data can be found @ https://github.com/Henrygwb/deepvsa