# Facilitating Postmortem Program Analysis with Deep Learning

Wenbo Guo[1], Xinyu Xing[1], Jimmy Su[1]

1. JD Security Research Center

京东安全

# Grim Reality

- Despite intensive in-house software testing, programs inevitably contain defects.
  - Accidentally terminate or crash at post-deployment stage.
  - Exploited as security loopholes.
- At JD, we have several large scale data centers.
  - Contains plenty of software.
  - Report a great many crashes everyday.

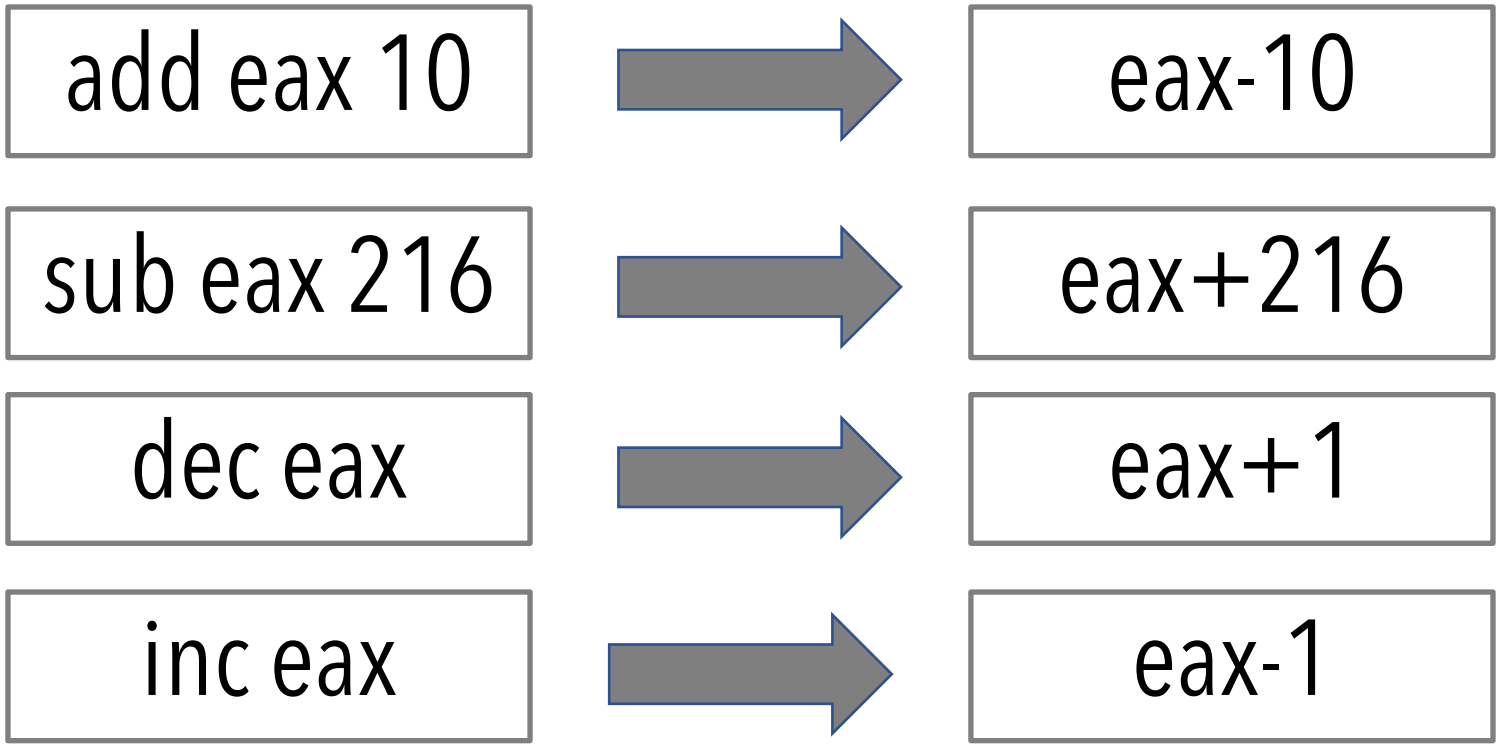
# Grim Reality

- Software analysts need to debug the program and patch the vulnerability.
- Software debugging is expensive and needs intensive manual efforts.
  - Debugging software cost has risen to $312 billion per year globally.
  - Developers spend 50% of their programming time finding and fixing bugs.

# Automatic debugging: Postmortem Program Analysis

- Track down the root cause of a software crash at the binary level with out source code.
- Analysis a crash dump and identify the execution path leading to the crash.
  - Reversely execute an instruction trace (starting from the crash site).

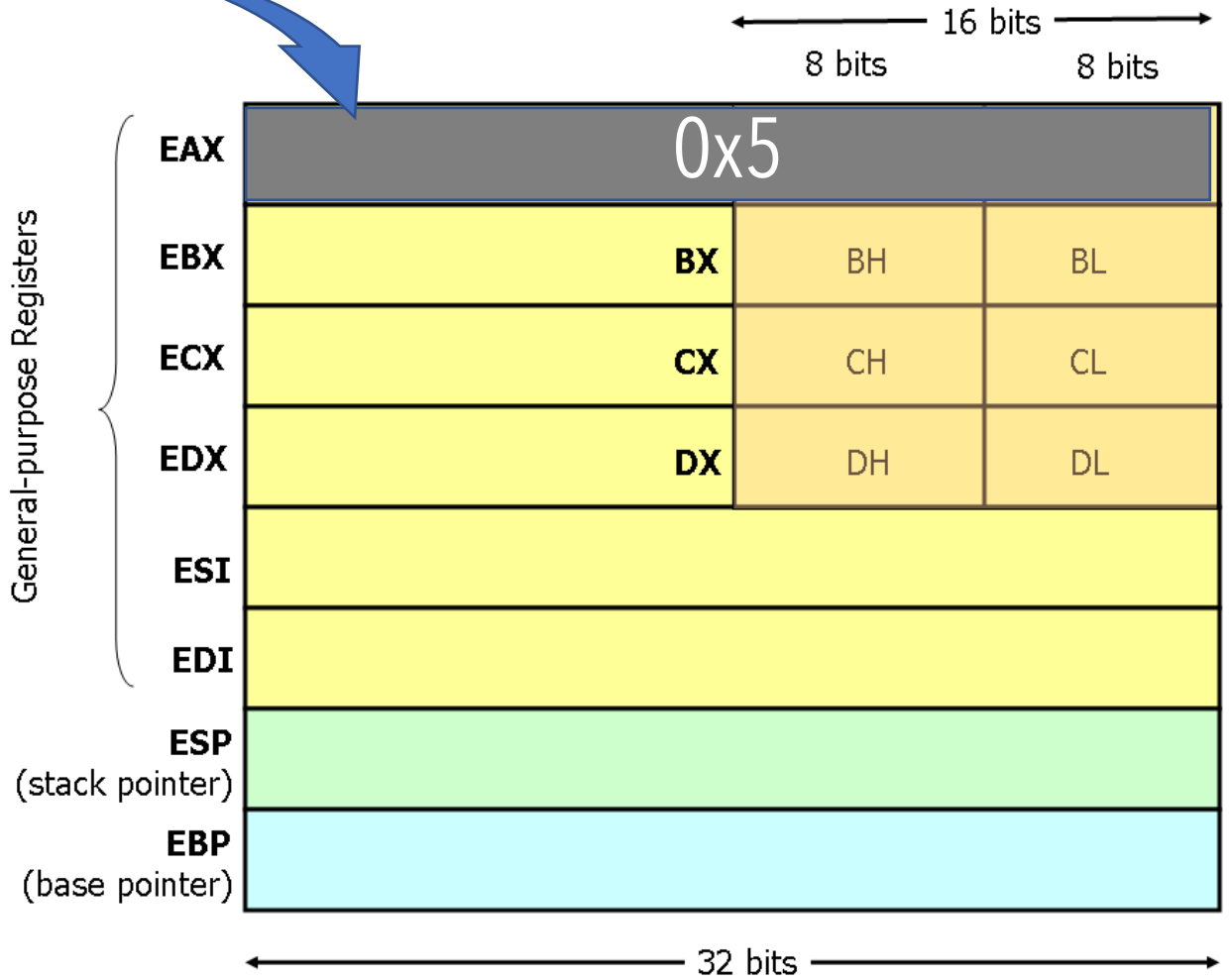
# Reverse Execution – Invertible Instructions



# Reverse Execution – Non-invertible Instructions

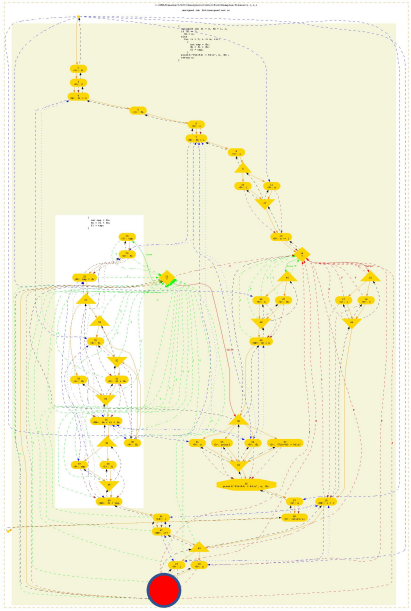
mov eax 0x5

xor ecx ecx



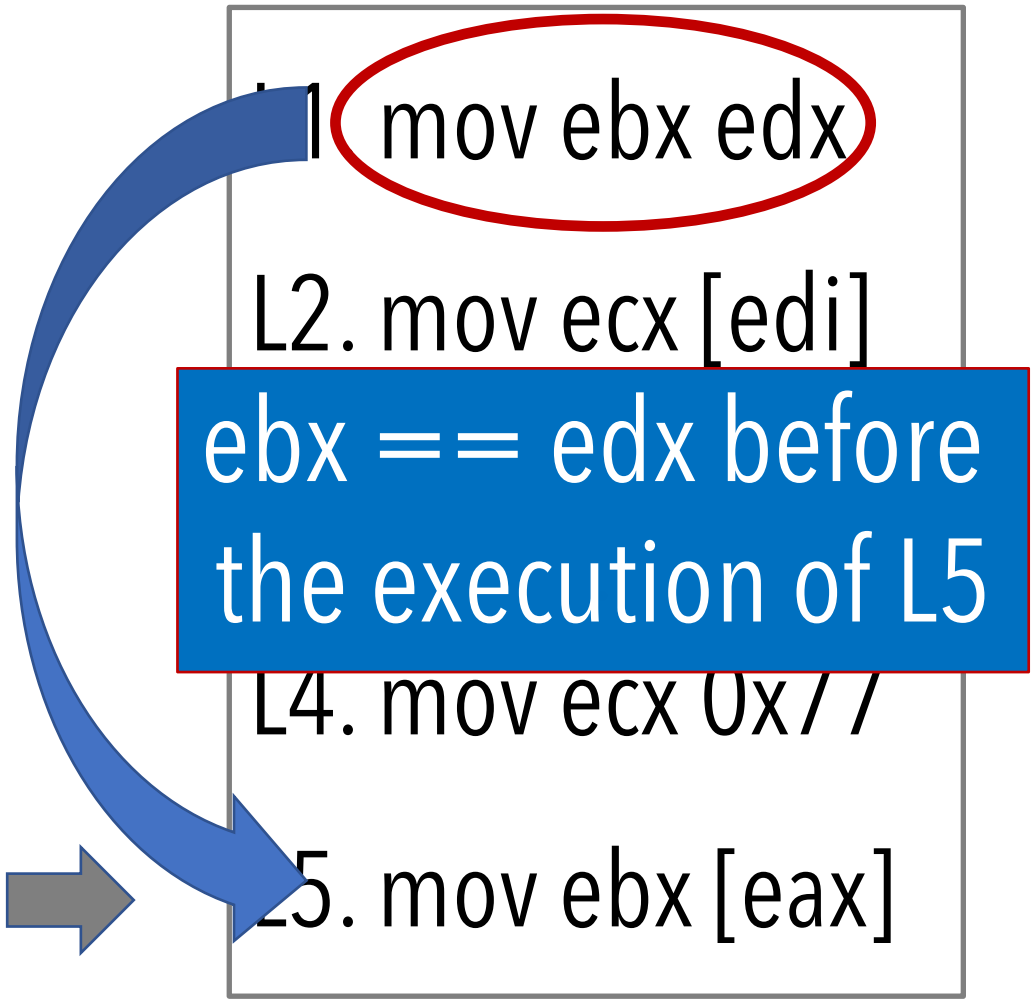
# Postmortem Program Analysis

- Perform data flow analysis against the identified path.
  - Reconstruct the data flow  that a program followed prior to its crash.
  - Examine how a bad value was passed to the crash site.
- Key problem in reconstructing data flow: alias detection.
  - Alias: two pointers point to the same memory location.



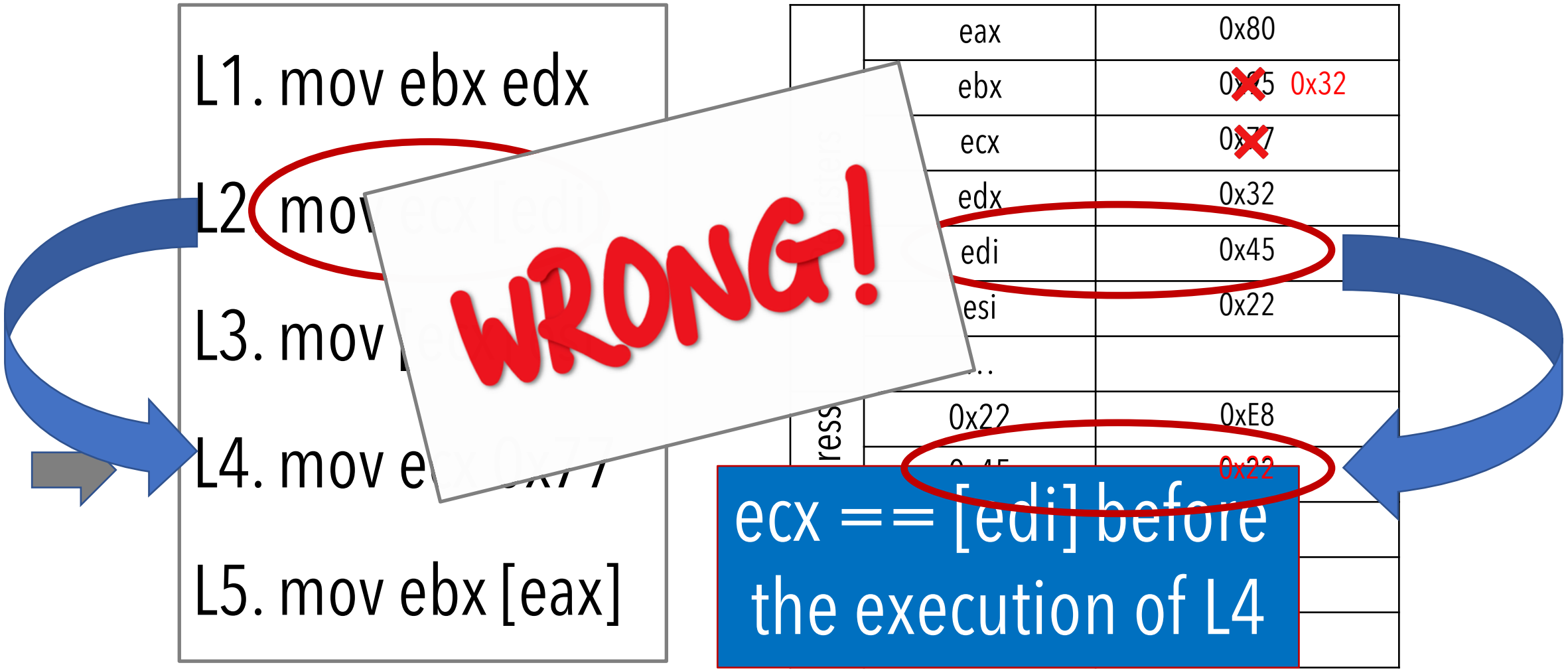
Data flow

# Reverse Execution with Backward Data Flow Analysis



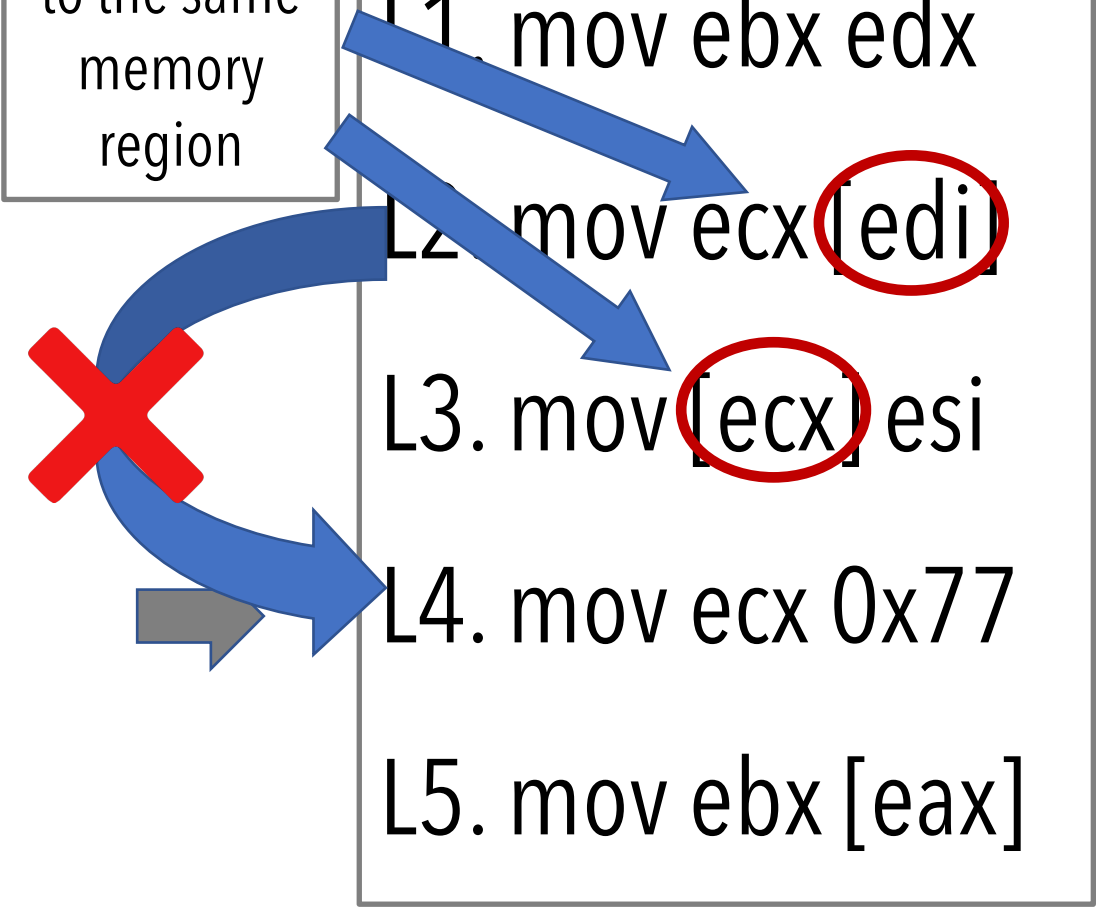

| | | |
|---|---|---|
| Registers | eax | 0x80 |
| | ebx | 0x45 |
| | ecx | 0x77 |
| | edx | 0x32 |
| | edi | 0x45 |
| | esi | 0x22 |
| | … | |
| Memory Address | 0x22 | 0xE8 |
| | 0x45 | 0x22 |
| | 0x77 | 0xB7 |
| | 0x80 | 0x95 |
| | … | |

# Challenge – Memory Alias Issue



# Challenge – Memory Alias Issue

Pointing to the same memory region




| | | |
|---|---|---|
| Registers | eax | 0x80 |
| | ebx | 0x45 0x32 |
| | ecx | 0x77 |
| | edx | 0x32 |
| | edi | 0x45 |
| | esi | 0x22 |
| | … | |
| ress | 0x22 | 0xE8 |

edi == ecx right before the execution of L4

# Alias Analysis

- Current postmortem program analysis system.
  - Adopt hypothesis testing to detect alias.
  - Extremely slow: spend weeks to find the root cause for a program.
  - Even slower than human analyst.
- Goal: accurate and efficient alias analysis.

京东安全

# Roadmap

- Value-set Analysis.
- Challenges of VSA in postmortem program analysis.
- Design overview.
- Technical details.
- Evaluation in real world crashes.
- Summary.

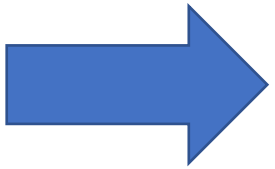
# Value-set Analysis

- State-of-the-art binary level alias analysis technique.
- Being integrated into a variety of binary analysis frameworks (Angr, BAP).
- High level idea:
  - Given a control flow, VSA first assigns each instruction into different memory region (Heap, Stack, global).
  - Tracks down *a-locs*:  register, memory call on stack, heap or global.
  - Compute a value set for each a-loc: (global, stack, heap).
  - Identify memory alias according to the value sets.

# Value-set Analysis

- Demonstration of Value-set Analysis.

```
1          sub     esp,44
2          lea     eax,[esp+4]
3          lea     ebx,[esp+24]
4          mov     [esp+0],eax
5          mov     ecx,0          Truncated Trace
-------------------------------------------------
L1         mov     edx,[0xC4]
7          mov     [eax],edx
8          mov     edx,[0xC8]
9          mov     [ebx],edx
10         add     eax,4
11         add     ebx,4
12         inc     ecx
13         cmp     ecx,1
14         jl      L1
15         mov     [esp+4],ecx
```



| | Complete Trace | |
|---|---|---|
| | *A-loc* | Value-set |
| 1 | esp | (⊥, [-44, -44], ⊥) |
| 2 | eax | (⊥, [-40, -40], ⊥) |
| 3 | ebx | (⊥, [-20, -20], ⊥) |
| 4 | [esp] (⊥, [-44, -44], ⊥) | (⊥, [-40, -40], ⊥) |
| 5 | ecx | ([0, 0], ⊥, ⊥) |
| L1 | [0xC4] ([0xC4, 0xC4], ⊥, ⊥) | ([0, 0], ⊥, ⊥) |
| | edx | ([0, 0], ⊥, ⊥) |
| 7 | [eax] (⊥, [-40, -40], ⊥) | ([0, 0], ⊥, ⊥) |
| 8 | [0xC8] ([0xC8, 0xC8], ⊥, ⊥) | ([0, 0], ⊥, ⊥) |
| | edx | ([0, 0], ⊥, ⊥) |
| 9 | [ebx] (⊥, [-20, -20], ⊥) | ([0, 0], ⊥, ⊥) |
| 10 | eax | (⊥, [-36, -36], ⊥) |
| 11 | ebx | (⊥, [-16, -16], ⊥) |
| 12 | ecx | ([1, 1], ⊥, ⊥) |
| 13 | - | - |
| 14 | - | - |
| 15 | [esp+4] (⊥, [-40, -40], ⊥)) | ([1, 1], ⊥, ⊥) |

# Why not Value-set Analysis in Postmortem Analysis

- In the content of postmortem program analysis:
  - The full control flow or execution trace is not available.
    - Core dump can only record limit length of execution trace.
  - VSA will perform over-approximation in value-set construction.

```
L1          mov     edx,[0xC4]
7           mov     [eax],edx
8           mov     edx,[0xC8]
9           mov     [ebx],edx
10          add     eax,4
11          add     ebx,4
12          inc     ecx
13          cmp     ecx,1
14          jl      L1
15          mov     [esp+4],ecx
```

| Incomplete Trace | |
|---|---|
| *A-loc* | Value-set |
| NA | NA |
| NA | NA |
| NA | NA |
| NA | NA |
| NA | NA |
| [0xC4] ([0xC4, 0xC4], ⊥, ⊥) | (⊤, ⊤, ⊤) |
| edx | (⊤, ⊤, ⊤) |
| [eax] (⊤, ⊤, ⊤) | (⊤, ⊤, ⊤) |
| [0xC8] ([0xC8, 0xC8], ⊥, ⊥) | (⊤, ⊤, ⊤) |
| edx | (⊤, ⊤, ⊤) |
| [ebx] (⊤, ⊤, ⊤) | (⊤, ⊤, ⊤) |
| eax | (⊤, ⊤, ⊤) |
| ebx | (⊤, ⊤, ⊤) |
| ecx | (⊤, ⊤, ⊤) |
| - | - |
| - | - |
| [esp+4] (⊥, [4, 4], ⊥) | (⊤, ⊤, ⊤) |

# Why not Value-set Analysis in Postmortem Analysis

- Alias Analysis Results.
  - Complete trace: **100%** correctly identify the alias pairs.
  - Incomplete trace:  mark **60%** of the memory pairs as may-alias.
    - Over-approximation.

Incomplete trace

| | [esp] | [0xC4] | [0xC8] | [eax] | [ebx] | [esp+4] |
|---|---|---|---|---|---|---|
| [esp] | - | 0 | 0 | 0 | 0 | 0 |
| [0xC4] | NA | - | 0 | 0 | 0 | 0 |
| [0xC8] | NA | 0 | - | 0 | 0 | 0 |
| [eax] | NA | ? | ? | - | 0 | 1 |
| [ebx] | NA | ? | ? | ? | - | 0 |
| [esp+4] | NA | ? | ? | ? | ? | - |

# Why Deep Learning

- Previous applications demonstrated that DL can be used to learn patterns from input and assign each input an label.
- For alias analysis:
  - Input data (instructions): a sequence of machine code.
  - Labels: memory regions each instruction is attached.
  - E.g. `push 0x68732f2f`: data [`68 2f 2f 73 68`].
    Label [stack stack stack stack stack].

# Why Deep Learning

- Learn from the previous sequence that reflects the process of determining memory region access.
- Memory region that an instruction accesses can be determined:
  - Semantics of that instruction.
    - `push eax`: indicate a stack access.
  - Context indicated by previous instruction.

```
0:   8d 1c 24   lea    ebx,[esp]
3:   89 0b      mov    DWORD PTR [ebx],ecx
```

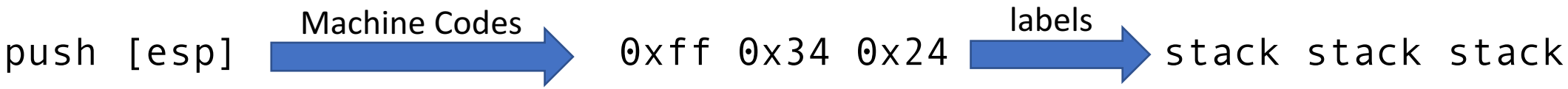
→ [ebx] indicates a stack access.

# Challenges for DL

- Capture the sequential dependence within input sequence.
  - Each hex in the binary code sequence are highly correlated with each other.
- The bi-directional dependence.
  - Forward analysis procedure.
  - Backward analysis procedure.

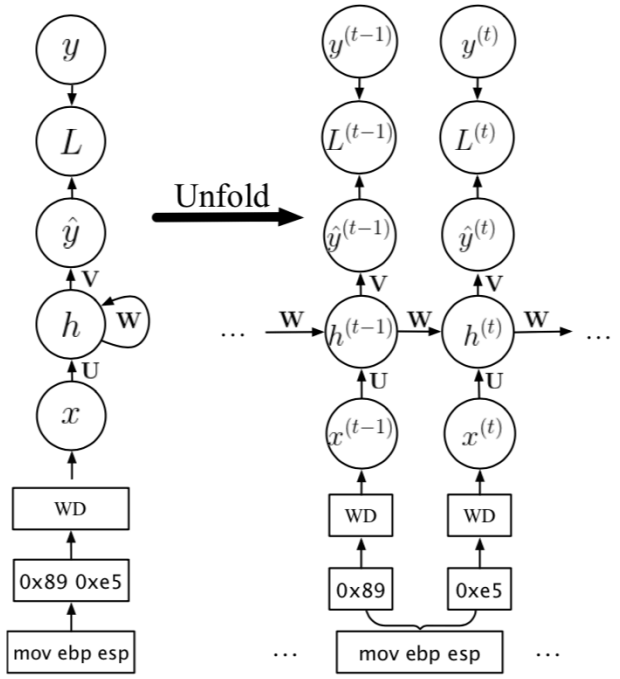
# Challenges for DL

- Catch the dependence between not only the dependency between and within instructions (input sequence) but also dependencies between adjacent labels.
  - Hexes in the same instruction have the same label.



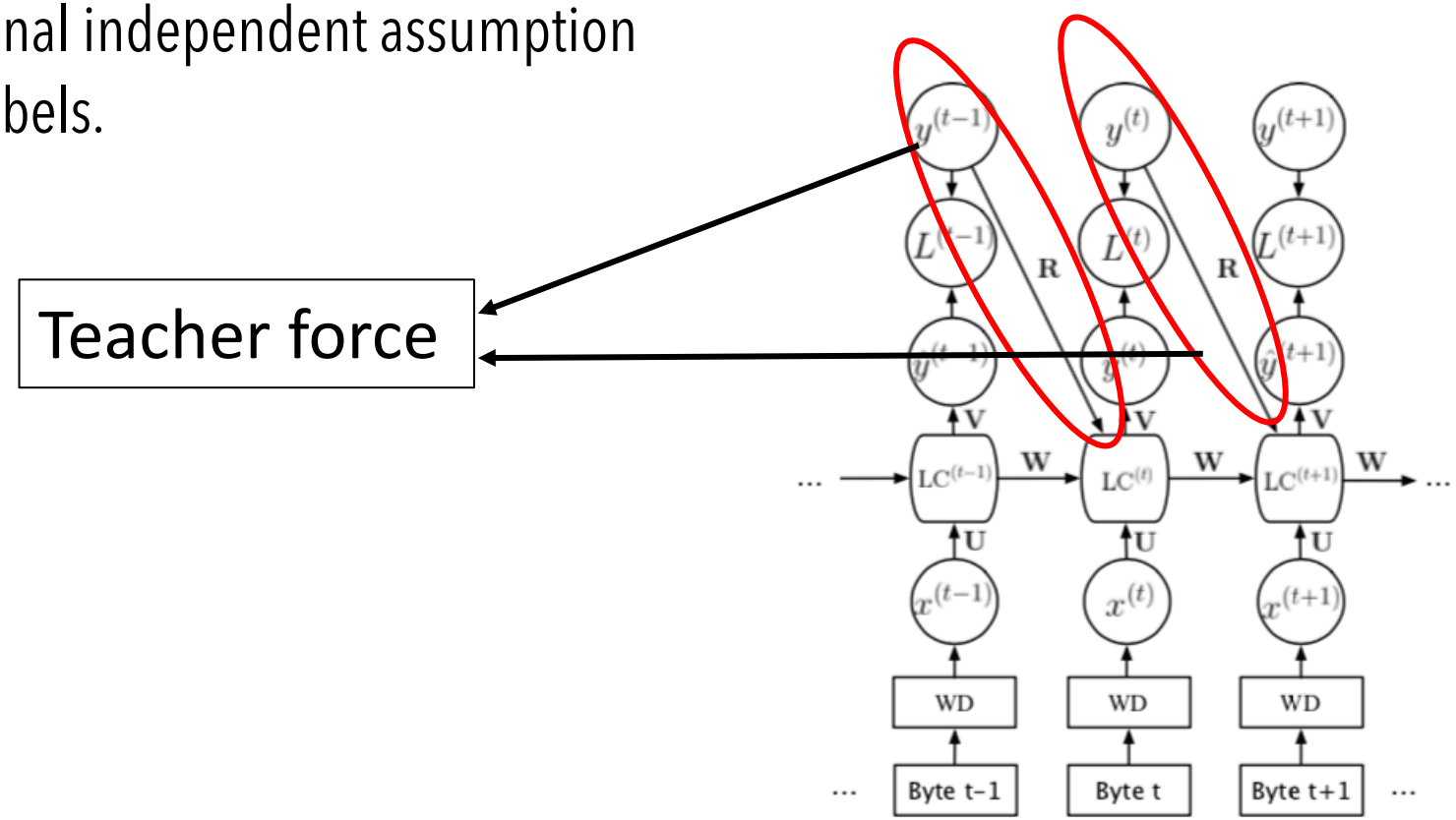
# Recurrent neural network – dependence within Hexes

- Recurrent neural network.
  - Take sequential data as input.
  - self-connected hidden units.
  - Model the  dynamic temporal behavior for a time sequence.
  - Types of hidden units:
    - SimpleRNN.
    - GRU.
    - LSTM (Capture long term dependence).



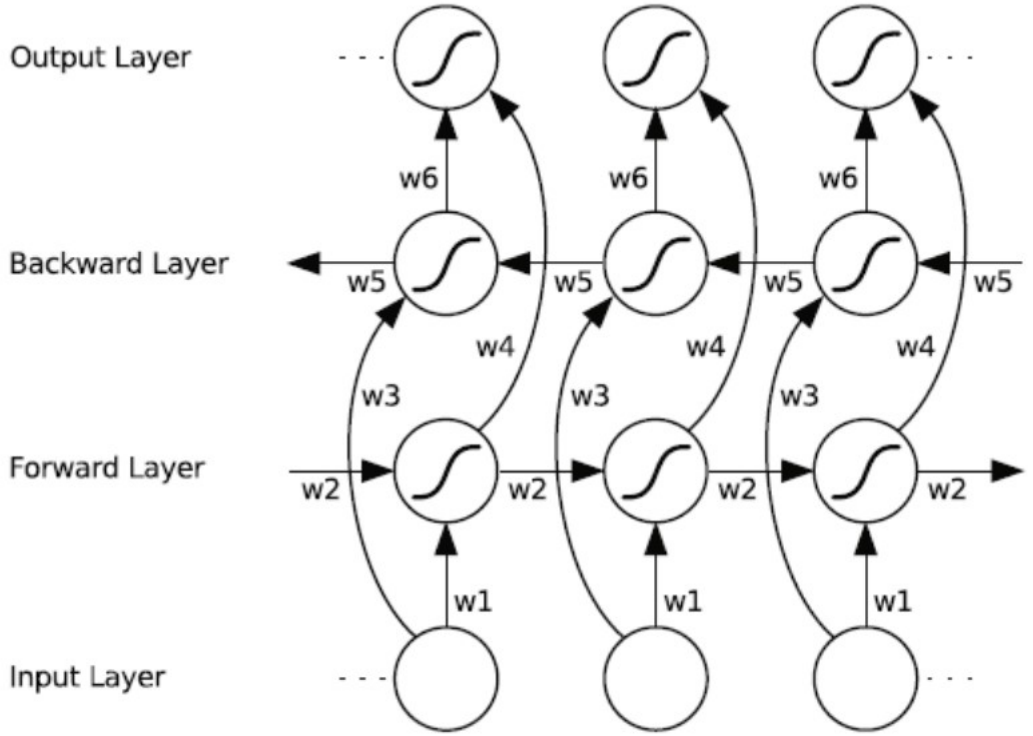
# Teacher force – dependence between labels

- Integrate the label of late time step into the current input.
  - Remove the conditional independent assumption between adjacent labels.



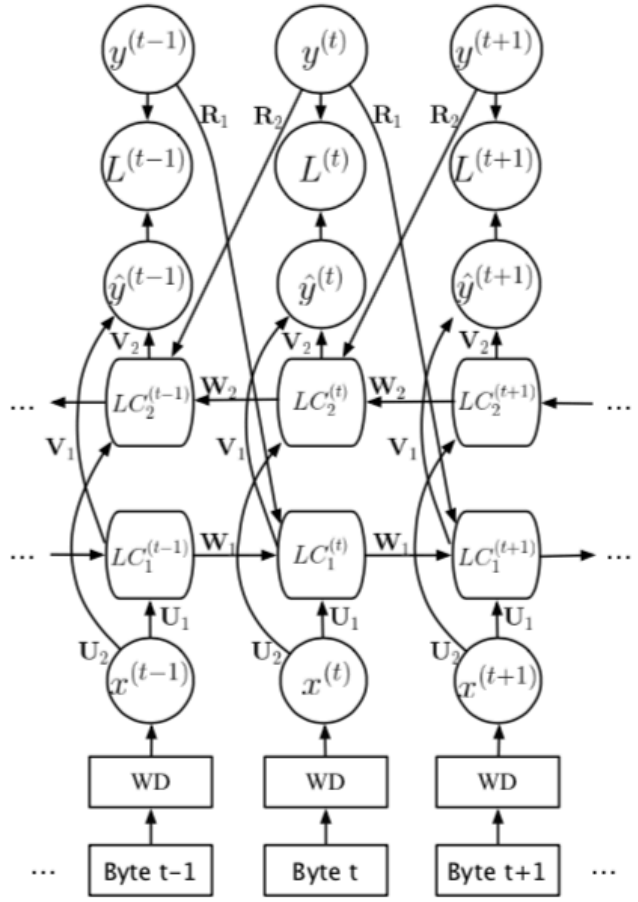
# Bi-directional RNN – backward dependence

- Bi-directional RNN combines a RNN that moves forward, beginning from the start of the byte sequence, with another RNN that moves backward.



# DEEPVSA: Deep learning Facilitated VSA

- Novel network structure: Bi-directional conditional LSTM.
  - Integrate LSTM, teacher force and Bi-directional connection together.
- Extension of conventional VSA.
  - Take the region predictions of DL model to determine non-alias relationships that the conventional VSA originally fails to identify.



# Evaluation

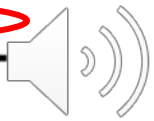
- Experimental setup.
  - Dataset:
    - Randomly select 54 vulnerability reports from the Exploit Database Archive.
    - Run the PoC programs tied to corresponding vulnerabilities, trigger software failures and collect the 54 execution traces pertaining to the crashes.
  - Answer the following questions:
    - Does the bi-directional conditional LSTM exhibit better performance than other RNN architecture?
    - Can the memory regions identified improve the memory alias analysis detection performance?

# Question one

- Comparison between different RNN structures.
  - Label the bytes tied to the execution traces with the memory regions.
  - Randomly divided the traces into 5 disjoint groups and train 5 distinct models for each network architectures.
    - Each model: take one group of traces as testing data and utilize the remaining to train our neural network models.

- Our model established highest precision and recall.

| | | Global | Heap | Stack | Other |
|---|---|---|---|---|---|
| Precision | Bi-RNN | 99.55% | 99.33% | 99.96% | 99.75% |
| | Bi-GRU | 99.55% | 99.49% | 99.98% | 99.80% |
| | Bi-LSTM | 99.55% | 99.30% | 99.97% | 99.76% |
| | Our model | **99.99%** | **99.79%** | **99.99%** | **99.88%** |
| Recall | Bi-RNN | 99.50% | 99.47% | 99.94% | 99.81% |
| | Bi-GRU | 99.54% | 99.49% | 99.94% | 99.78% |
| | Bi-LSTM | 99.51% | 99.55% | 99.94% | 99.81% |
| | Our model | **99.88%** | **99.76%** | **99.97%** | **99.90%** |



# Question two

- Memory alias pairs detection.
  - Apply the DL model to predict the memory regions tied to each trace.
  - Pass the predictions to DEEPVSA and compute the non-alias pairs.
- Use conventional VSA as baseline for comparison.

| Program | CVE/EDB-ID | LoC | TR? | VSA | | DEEPVSA | | Statistics | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Non-alias | Error rate | Non-alias | Error rate | Global | Heap | Stack | Other |
| DXFScope-0.2 | CVE-2004-1271 | 7697 | ✗ | 47.33% | 0% | 86.06% | 0% | 5 | 0 | 647 | 338 |
| autotrace-0.31.1 | CVE-2017-9180 | 12620 | ✗ | 54.90% | 0% | 93.30% | 0% | 75 | 517 | 12804 | 4801 |
| bento4-1.5.0-617 | CVE-2017-14638 | 43610 | ✗ | 58.73% | 0% | 96.14% | 0% | 254 | 5335 | 28428 | 4986 |
| psutils-p17 | EDB-890 | 1736 | ✗ | 29.96% | 0% | 81.73% | 0% | 47 | 88 | 21492 | 16680 |
| gif2png-2.5.2 | CVE-2009-5018 | 1331 | ✗ | 25.34% | 0% | 78.18% | 0% | 17 | 770 | 24040 | 16445 |
| bento4-1.5.0-617 | CVE-2017-14640 | 43610 | ✗ | 56.53% | 0% | 95.18% | 0% | 351 | 7784 | 37953 | 7380 |
| openjpeg-2.1.1 | CVE-2016-7445 | 169538 | ✗ | 36.38% | 0% | 84.93% | 0% | 159 | 195 | 28214 | 19736 |
| libpng-1.2.5 | CVE-2004-0597 | 33681 | ✗ | 37.37% | 0% | 85.88% | 0% | 30 | 1109 | 30908 | 21366 |
| unrtf-0.19.3 | CVE-2004-1297 | 5039 | ✗ | 35.84% | 0% | 87.52% | 0% | 994 | 5905 | 32803 | 23018 |
| LibSMI-0.4.8 | CVE-2010-2891 | 80461 | ✗ | 51.10% | 0% | 91.84% | 0% | 8 | 2638 | 50213 | 21606 |
| libzip-1.2.0 | CVE-2017-12858 | 37083 | ✗ | 17.91% | 0% | 80.71% | 0% | 17 | 15966 | 35285 | 21126 |
| TestDisk-6.14 | EDB-36881 | 64345 | ✓ | 34.62% | 0% | 83.47% | 0% | 262 | 29644 | 44752 | 52369 |
| unalz-0.52 | CVE-2005-3862 | 8546 | ✓ | 4.00% | 0% | 55.46% | 0% | 69 | 6229 | 52353 | 28511 |
| JPegToAvi-1.5 | CVE-2004-1279 | 580 | ✓ | 0.01% | 0% | 49.96% | 0% | 28 | 336 | 52363 | 46879 |
| o3read-0.0.3 | CVE-2004-1288 | 932 | ✓ | 28.00% | 0% | 79.00% | 0% | 3031 | 0 | 61488 | 63950 |
| corehttp-0.5.3.1 | CVE-2009-3586 | 935 | ✓ | 63.44% | 0% | 95.61% | 0% | 564 | 9905 | 92576 | 27357 |
| corehttp-0.5.3alpha | CVE-2007-4060 | 935 | ✓ | 63.90% | 0% | 95.67% | 0% | 564 | 10977 | 96820 | 28616 |
| mp3info-0.8.5a | CVE-2006-2465 | 3212 | ✓ | 0.03% | 0% | 50.74% | 0% | 138 | 4564 | 86747 | 58607 |
| bento4-1.5.0-617 | CVE-2017-14641 | 43610 | ✓ | 37.26% | 0% | 84.31% | 0% | 507 | 36751 | 85352 | 19009 |
| unrar-3.9.3 | EDB-17611 | 17575 | ✓ | 69.72% | 0% | 90.46% | 0% | 5793 | 0 | 90038 | 30720 |
| HTML2HDML-1.0.3 | CVE-2004-1275 | 7894 | ✓ | 28.29% | 0% | 80.19% | 0% | 192 | 28120 | 35438 | 4200 |
| SQLite-3.8.6 | CVE-2015-5895 | 98039 | ✓ | 0.13% | 0% | 43.72% | 0% | 196 | 9571 | 67889 | 22893 |
| htmldoc-1.8.27 | CVE-2009-3050 | 59237 | ✓ | 0.15% | 0% | 45.29% | 0% | 382 | 5256 | 71160 | 24923 |
| sudo-1.8.0 | CVE-2012-0809 | 38761 | ✓ | 0.11% | 0% | 50.29% | 0% | 81 | 8892 | 64057 | 27490 |
| GnuPG-1.9.14 | CVE-2006-3746 | 99053 | ✓ | 32.16% | 0% | 83.00% | 0% | 1 | 45073 | 44988 | 2784 |
| gas-2.12 | CVE-2005-4807 | 595504 | ✓ | 3.41% | 0% | 65.09% | 0% | 2369 | 33388 | 31986 | 10309 |
| mcrypt-2.5.8 | CVE-2012-4409 | 37439 | ✓ | 0.03% | 0% | 37.82% | 0% | 49 | 1955 | 49921 | 16862 |
| nasm-0.98.38 | CVE-2004-1287 | 33553 | ✓ | 48.68% | 0% | 92.96% | 0% | 5441 | 27841 | 56075 | 9899 |
| prozilla-1.3.6 | CVE-2004-1120 | 9000 | ✓ | 0.01% | 0% | 54.00% | 0% | 149 | 25455 | 50780 | 11355 |
| stftp-1.1.0 | EDB-9264 | 1559 | ✓ | 43.70% | 0% | 90.14% | 0% | 1647 | 32650 | 52504 | 7850 |
| gdb-7.5.1 | EDB-23523 | 1665735 | ✓ | 2.72% | 0% | 60.61% | 0% | 1268 | 46335 | 49770 | 10058 |
| 0verkill-0.16 | CVE-2006-2971 | 16361 | ✓ | 5.56% | 0% | 11.00% | 0% | 0 | 0 | 4652 | 74418 |
| gdb-6.6 | EDB-30142 | 1377746 | ✓ | 15.87% | 0% | 62.21% | 0% | 3654 | 13147 | 69093 | 22559 |
| make-3.81 | EDB-34164 | 24168 | ✓ | 0.01% | 0% | 19.93% | 0% | 57 | 25700 | 2886 | 867 |
| coreutils-8.4 | CVE-2013-0223 | 138135 | ✓ | 81.50% | 0% | 96.00% | 0% | 1524 | 4578 | 110193 | 8377 |
| ClamAV-0.93.3 | CVE-2008-5314 | 69430 | ✓ | 99.99% | 0% | 100.00% | 0% | 0 | 0 | 118264 | 16179 |
| ClamAV-0.88.2 | CVE-2006-5295 | 39052 | ✓ | 38.53% | 0% | 90.00% | 0% | 0 | 32758 | 59916 | 27333 |
| ClamAV-0.88.2 | CVE-2006-4018 | 39052 | ✓ | 30.60% | 0% | 78.03% | 0% | 0 | 43830 | 0 | 64745 |
| putty-0.66 | CVE-2016-2563 | 90165 | ✓ | 32.52% | 0% | 71.52% | 0% | 2704 | 17304 | 63094 | 10987 |
| coreutils-8.4 | CVE-2013-0222 | 138135 | ✓ | 0.39% | 0% | 19.03% | 0% | 30 | 74 | 5273 | 73534 |
| autotrace-0.31.1 | CVE-2017-9182 | 12620 | ✓ | 45.24% | 0% | 62.00% | 0% | 435 | 3045 | 112323 | 10878 |

# Question two

- Memory alias pairs detection.
  - DEEPVSA improves the detection rate from 24.84% to 66.43%.
  - DEEPVSA dose not introduce errors.

| Program | CVE/EDB-ID | LoC | TR? | VSA | | DEEPVSA | | Statistics | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Non-alias | Error rate | Non-alias | Error rate | Global | Heap | Stack | Other |
| GnuPG-1.9.14 | CVE-2006-3082 | 99053 | ✓ | 32.20% | 0% | 83.52% | 0% | 0 | 45069 | 45072 | 2818 |
| podofo-0.94 | CVE-2017-5854 | 60147 | ✓ | 0.49% | 0% | 27.86% | 0% | 1104 | 15285 | 105954 | 6200 |
| **Average** | - | - | - | 24.84% | 0% | 66.43% | 0% | 728 | 14575 | 49670 | 32795 |

# Summary

- DEEPVSA implements a novel RNN architecture customized for VSA.
  - Bi-directional conditional LSTM.
- DEEPVSA outperforms the off-the-shelf recurrent network architecture in terms of memory region identification.
- DEEPVSA significantly improves the VSA with respect to its capability in analyzing memory alias.
- DEEPVSA will enhance the **accuracy** and **efficiency** of the postmortem program analysis .

Thank you very much!