Neural Machine Translation Inspired Binary Code Similarity Comparison beyond Function Pairs

Fei Zuo, Xiaopeng Li, Patrick Young, Lannan Luo*, Qiang Zeng*, Zhexin Zhang

NDSS 2019, Feb 27
Why Cross-Architecture Binary Code Similarity Comparison?
Cross-architecture binary code similarity comparison

- Plagiarism detection
- Malware family identification
- Vulnerability discovery
A challenging task due to different

• Instruction sets
• Registers
• Memory addressing
• Calling conventions
• Compilation optimizations
• ...

X86  ARM  MIPS
What is the current research status?
• Non-machine-learning approaches
  – Multi-MH [S&P’15]: fuzzing (on basic blocks)
  – Esh [PLDI’16]: SMT (on IR)
  – David et al. [PLDI’17]: re-compilation (of IR)
    ➢ Slow

• Machine-learning approaches
  – Genius [CCS’16]: traditional ML
  – Gemini [CCS’17]: deep learning (graph)
    ➢ Fast; accurate (at function level)
    ➢ But…
*Gemini* used some manually selected features to represent a basic block, e.g., # of instructions, # calls, etc.

- Is it good enough?
  - Basic-block comparison: AUC = 0.85

- Could we do better?

- What information is lost?
  - Instruction meaning
  - Instruction dependence
• **Neural Machine Translation**: deep learning for translation
• First proposed in 2014
• Already adopted by Google and Microsoft
A binary, after disassembly, is represented in some assembly language. Can NMT handle assembly languages as well?

More specifically, given that NMT can translate sentences, can it also compare code of different architectures?
Interesting idea, but tons of questions

- **Words** ↔ instructions, but an infinite vocabulary?
  - E.g., mov edx, 200

- **Sentences** ↔ basic blocks vs. functions?
  - A sentence: a sequence of words
  - A basic block: a sequence of instructions
  - A function: a graph

- **Corpus** of equivalent basic block pairs?
  - Unlike functions, which have names

- **Expensive hardware**?
  - We are not Google
  - Would be impractical if expensive facilities are required

- **Interesting application**?
  - Submitted to S&P in 05/2018; comment: no interesting application
MOVL %ESI, $.L.STR.31  |  MOVL ESI, <STR>
MOVL %EDX, $3       |  MOVL EDX, 0
MOVQ %RDI, %RAX    |  MOVQ RDI, RAX
CALLQ STRNCMP     |  CALLQ FOO
TESTL %EAX, %EAX   |  TESTL EAX, EAX
JE .LBB0_5         |  JE <TAG>

Instruction preprocessing:
(1) Constant value => 0
(2) Strings => <str>
(3) Function names => FOO
(4) Other labels => <TAG>
The size of vocabulary ($10^6$).

ARM vocabulary: 21K
X86 vocabulary: 28K
The *word2vec* network is then trained using the preprocessed instructions.

Then, the network is used to convert each instruction into an *instruction embedding*.
Corpus of equivalent BB pairs

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVSLQ RSI,EBP</td>
<td>LDRB R0,[R8+R4]</td>
</tr>
<tr>
<td>MOVZBL ECX,[R14,RBX]</td>
<td>STR R9,[SP]</td>
</tr>
<tr>
<td>MOVL EDX,&lt;STR&gt;</td>
<td>STR R0,[SP+0]</td>
</tr>
<tr>
<td>XORL EAX,EAX</td>
<td>ASR R3,R7,0</td>
</tr>
<tr>
<td>MOVQ RDI,R13</td>
<td>MOV R0,R6</td>
</tr>
<tr>
<td>CALLQ FOO</td>
<td>MOV R2,R7</td>
</tr>
<tr>
<td>TESTL EAX,EAX</td>
<td>BL FOO</td>
</tr>
<tr>
<td>JLE &lt;TAG&gt;</td>
<td>CMP R0,0</td>
</tr>
<tr>
<td></td>
<td>BLT &lt;TAG&gt;</td>
</tr>
</tbody>
</table>

At backends, BBs generated from the same IR BB obtain the same annotated ID
Architecture for cross-architecture BB similarity comparison

X86 => ARM, then compare two ARM BBs?
- No, NLP researchers use the *Siamese architecture* to compare the similarity of two sentences [AAAI’16]
Architecture for cross-architecture BB similarity comparison

\[ \text{Similarity Score} := \exp(-L_1) \]

\[ L_1 = \| h^{(1)}_T - h^{(2)}_S \|_1 \]

\[ LST M_1 \quad \text{X86} \]

\[ LST M_2 \quad \text{ARM} \]

Layer 1

Layer n

\[ B_1 \]

\[ B_2 \]
Interesting Application

• Prior cross-architecture binary analysis
  – answers whether $C_1$ is equivalent to $C_2$
  – cannot answer whether $C_1$ is contained in program $P$

• The code containment problem:
  – Vulnerable code is inlined as part of another function
  – An attacker reuses a crypto in multiple malware
  – One steals a piece of code and inserts it into program
  – ...

• Not explored yet in cross-architecture scenarios
• To determine whether $C$ is contained in $P$
  – The CFG of $C$ is decomposed into multiple paths
  – For each path $x$ of $C$, LCS (longest common subsequence) and breadth-first search are combined to search in the CFG of $P$, and calculate a score for path $x$
  – Based on all path scores, a final score is calculated

• It was proposed by Luo et al. [FSE’14]
  – Symbolic execution for BB comparison
  – Mono-architecture code analysis

• Applying our NMT-based BB comparison
  – The first solution to cross-architecture code containment
  – Much faster
Hardware

• Actually, a Dell laptop
  – 2.7 GHz Intel i7
  – 32 GB RAM
  – No GPUs
Datasets for training InnerEye-BB

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>Sim.</th>
<th>Dissim.</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>87,209</td>
<td>43,686</td>
<td>43,523</td>
<td></td>
</tr>
<tr>
<td>O2</td>
<td>112,019</td>
<td>56,082</td>
<td>55,937</td>
<td></td>
</tr>
<tr>
<td>O3</td>
<td>119,860</td>
<td>60,003</td>
<td>59,857</td>
<td></td>
</tr>
<tr>
<td>Cross-opts</td>
<td>84,555</td>
<td>42,481</td>
<td>42,074</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>403,643</td>
<td>202,252</td>
<td>201,391</td>
<td></td>
</tr>
</tbody>
</table>

- **Training : validation : testing = 0.8 : 0.1 : 0.1**

- **Deduplication**: any BB in training does not re-appear in validation or testing
Cross-optimization levels, different sizes of BBs
Large BBs at O3

Small BBs at O3
Good accuracy after 20 epochs
Each epoch takes 971 seconds
Training time: 5.5 hours

Testing time per BB pair: 0.76 ms
Case studies on code containment

• Whether the *URL checking loop* of *thttpd* is contained in other programs
  – *sthttpd* got a score 0.91, while others got < 0.04
  – Consistent with manual checking

• Whether *MD5* code of *OpenSSL* is included in other 12 programs
  – High scores (0.88~0.93) for *cryptlib*, *openssh*, *libgcrypt*, etc.
  – Low scores for others
t-SNE of instructions
• A good word embedding model
  \[ \cos(\text{“man”, “woman”}) \approx \cos(\text{“king”, “queen”}) \]

• Our instruction embedding model
  \[ \cos(\text{BEQ <TAG>, BNE <TAG>}) \approx \cos(\text{JE <TAG>, JNE <TAG>}) \]
  \[ \cos(\{\text{ADD SP,SP,0}, \{\text{SUB SP,SP,0}\}) \approx \cos(\{\text{ADDQ RSP,0}, \{\text{SUBQ RSP,0}\}} \]
Take-away messages

• NMT-inspired cross-architecture binary code similarity comparison works well (AUC = 0.98)
  – Can NLP inspire us (binary analysts) more?
• Does not need “big data” (400k samples)
• A laptop without GPU can do the job
• First solution to cross-arch code containment

• Uncertain: cross-compiler? (on-going work)
Thank you!
Q&A
Out-of-Vocabulary (OOV) rate

The proportion of unseen instructions in test corpus (%).

3.7% vs. 90%

With pre-processing
Without pre-processing

The proportion of used corpus (%).
How about BBs of different optimization levels of the same architecture?

- O3 B1 ← src code ← O0 B2
- Src code (file name, line, row) info. is recorded as part of the BB annotation
How about dissimilar BB pairs?

- ARM O3 BB1 => ARM O0 BB2
- X86 O2 BB3 => X86 O0 BB4 => ARM O0 BB5
- Use *n-gram* to compare BB2 and BB5
- If they are dissimilar, BB1 and BB3 are dissimilar