Binary analysis: Code Search

Yue Duan
Outline

- Code Search basics
- Research papers:
  - Scalable Graph-based Bug Search for Firmware Images
  - Neural Network-based Graph Embedding for Cross-Platform Binary Code Similarity Detection
  - DeepBinDiff: Learning Program-Wide Code Representations for Binary Differencing
Problem definition

- given two pieces of binary code (e.g., binary functions)
  - maybe in different architectures
  - maybe by different compilation configs
    - compilers
    - compiler versions
    - optimization levels
    - other options
  - check if they are semantically equivalent or similar
Security applications

- plagiarism detection
Security applications

- vulnerability analysis
  - i.e., vulnerability search in IoT
Vulnerability analysis

Vulnerability

i.e., Heartbleed

Firmware Image Database

Similar?

Similar?

Similar?

Similar?

Important!
Challenges

Cross-Platform

- ARM
- MIPS
- x86

Scalability
Example

a) x86 assembly

push ebx
mov eax, [esp+4+arg_0]
mov edx, [eax+58h]
mov ebx, [edx+344h]
mov edx, [eax]
mov eax, [ebx+24h]
mov ecx, edx
sar ecx, 8
cmp ecx, 3
jz short loc_80A9550

cmp edx, 302h
jle short loc_80A954D

cmp eax, 0C030h
mov edx, 20080h
cmovz eax, edx
pop ebx
ret

b) MIPS assembly

lw $v0, 0x58($a0)
lw $v1, 0($a0)
lw $v0, 0x344($v0)
sra $a1, $v1, 8
li $a0, 3
bne $a1, $a0, locret_19830
lw $v0, 0x24($v0)

slti $v1, 0x303
bnez $v1, locret_19830
li $v1, 0xC030

bne $v0, $v1, locret_19830
nop

la $v0, loc_20080

jr $ra
nop
Existing techniques - static approaches

- BinDiff [https://www.zynamics.com/bindiff.html](https://www.zynamics.com/bindiff.html)
  - de-facto commercial tool
  - binary => Control-flow graph
  - graph isomorphism detection
  - heuristics for runtime performance
Existing techniques - dynamic approaches

- Blanket execution \textit{[USENIX Sec'14]}
  - dynamically execute two given binaries
  - collect runtime information during execution
  - checking the semantic level equivalence based on the information
Existing techniques

- static analysis
  - low accuracy
  - syntax rather than semantics
- dynamic analysis
  - code coverage issue
  - poor scalability

new trend - Learning-based approaches!
Scalable Graph-based Bug Search for Firmware Images

Qian Feng, Rundong Zhou, Chengcheng Xu, Yao Cheng, Brian Testa, and Heng Yin

ACM CCS 2016
Motivation

● Key challenge: cross-platform code search
  ○ string pattern or constant matching [Costin et al. USENIX Sec’14]
    ■ lack of generality
  ○ I/O pairs [Pewny et al. IEEE S&P’15]
    ■ lack of scalability
  ○ DiscovRe [Eschweiler et al. NDSS’16]
    ■ still not scalable enough
    ■ unreliable

● Biggest problem
  ○ graph matching is EXPENSIVE!

● How to achieve high performance matching?
  ○ graph representation learning
Graph Representation Learning

- Key idea:
  - learn from image processing
  - use a high-dimensional vector, a.k.a. embeddings, to represent a graph
Genius Overview
Attributed Control Flow Graph

- Attributed Control Flow Graph
  - a control-flow graph with features

<table>
<thead>
<tr>
<th>Type</th>
<th>Feature Name</th>
<th>Weight ($\alpha$)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Statistical Features</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>String Constants</td>
<td>10.82</td>
</tr>
<tr>
<td></td>
<td>Numeric Constants</td>
<td>14.47</td>
</tr>
<tr>
<td></td>
<td>No. of Transfer Instructions</td>
<td>6.54</td>
</tr>
<tr>
<td></td>
<td>No. of Calls</td>
<td>66.22</td>
</tr>
<tr>
<td></td>
<td>No. of Instructions</td>
<td>41.37</td>
</tr>
<tr>
<td></td>
<td>No. of Arithmetic Instructions</td>
<td>55.65</td>
</tr>
<tr>
<td><strong>Structural Features</strong></td>
<td>No. of offspring</td>
<td>198.67</td>
</tr>
<tr>
<td></td>
<td>Betweeness</td>
<td>30.66</td>
</tr>
</tbody>
</table>

Table 1: Basic-block level features.
Attributed Control Flow Graph

- Example

(a) Partial control flow graph of dltls1_process_heartbeat

(b) The corresponding ACFG
Index and Search

a. ACFG

b. Codebook

[0.1, 0, 0, 0, 0.9, 0.7, 0.1]

c. Encoded feature vector (VLAD encoding)

Encoded Feature Vector

<table>
<thead>
<tr>
<th>ID</th>
<th>Feature vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[0.3, 0, 0, 0, 0.9, 0.7, 0.1]</td>
</tr>
<tr>
<td>1</td>
<td>[0.2, 0, 0, 0.4, 0.9, 0, 0.1]</td>
</tr>
<tr>
<td>2</td>
<td>[0.7, 0.01, 0.8, 0, 0.5, 0.2]</td>
</tr>
<tr>
<td>3</td>
<td>[0.1, 0, 0, 0, 0.9, 0.7, 0.1]</td>
</tr>
</tbody>
</table>

Vulnerability Search Engine

Locality Sensitive Hashing

<table>
<thead>
<tr>
<th>ID</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1.0</td>
</tr>
<tr>
<td>10</td>
<td>0.99</td>
</tr>
<tr>
<td>5</td>
<td>0.98</td>
</tr>
</tbody>
</table>

d. Ranking list of search results
Evaluation: True positive rate
Evaluation: Efficiency
Neural Network-based Graph Embedding for Cross-Platform Binary Code Similarity Detection

Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, Dawn Song

ACM CCS 2017
Motivation

- Genius is great
  - more accurate: ACFG
  - faster: graph embeddings

- However
  - codebook generation can bring inaccuracy
  - graph representation learning is naive

- Key idea:
  - Can we use deep learning to learn graph embeddings?
Approach: Gemini
Embedding Network

1. Initially, each vertex has an embedding vector computed from each code block

2. In each iteration, the embedding on each vertex is propagated to its neighbors

3. After the last iteration, the embeddings on all vertexes are aggregated together

4. An affine transformation is applied in the end to compute the embedding for the graph
Training: Siamese Network

- Training data: a large number of functions
  - similar and dissimilar functions
- Train the network so that
  - similar functions will generate high similarity score
  - dissimilar functions will generate low similarity score
Evaluation: Efficiency

- Per function processing time
  - Genius: a few seconds to a few mins
  - Now: a few milliseconds

- Training time
  - Genius: > 1 week
  - Now: < 30 mins
Evaluation: Effectiveness

- Among top 50 vulnerabilities
  - 42/50 can be identified
  - Genius: 10/50
DeepBinDiff: Learning Program-Wide Code Representations for Binary Diffing

Yue Duan, Xuezixiang Li, Jinghan Wang, and Heng Yin

NDSS 2020
Motivation

- **Existing techniques**
  - No efficient binary diffing at basic block level
    - Genius and Gemini: function comparison
  - No program-wide contextual information
    - why useful?
  - Heavily rely on labelled training data
    - balanced training data can be hard
System Design

Complete unsupervised learning approach

semantic info learning
calculate similarity
efficient matching
A TEACHING GRAMMAR OF THE ENGLISH LANGUAGE

Patricia L. McElwain

The performance of any complex English sentences requires a facility for simple but different sentence patterns and the use of different grammatical structures. The sentence patterns and their structures provide the foundation for understanding the core elements of English language. This understanding is based on the notion of the sentence as a unit of communication. The sentence patterns and their structures help one to identify the subject and predicate of the sentence. The sentence patterns and their structures also provide a framework for understanding the meaning of the sentence. The sentence patterns and their structures are important in the study of English language and are essential for the development of effective communication skills.

In any grammatical area the teacher of English or any native speaker needs a great deal of information which he cannot readily find in ordinary descriptive grammars. The type of grammar does not often exist, for instance, explicit instructions on the use of certain forms, are explicitly identified when used in context in fiction. This indicates a need for why might be a natural and proper question, but in some cases, the use of certain forms in ordinary descriptive grammars is anything but explicit and readily available. Certain information is only in the teacher. This paper outlines the teacher's grammatical needs with reference to sentence usage in English, and is to doing, is proposed as an outline or a version of such a teaching grammar.
Program-wide Contextual Info Learning

if str == 'hello' do
  Basic Block A
  Basic Block A'
end

if str == 'hello' do
  Basic Block B
  Basic Block B'
end
Program-wide Contextual Info Learning

**feature vector**

- 0.053, 0.16, 0.032 …
- 0.12, 0.44, -0.009 …
- 0.411, -0.2206, 0.4 …
- 0.55, 0.656, 0.33 …

**merged graph**

Graph Representation Learning

**basic block embeddings**

- 0.055, 0.004, -0.07 …
- 0.07, -0.314, 0.305 …
- 0.335, -0.93, 0.1189 …
- -1.8e-06, 0.092, 0.06 …

semantics & contextual info

calculate basic block similarity
Outperform state-of-the-art techniques by 30% for cross-version and cross-optimization-level diffing
Thank you!

Question?