Binary Code Search
Problem Definition

• Given a piece of binary code (e.g., a binary function)
• Quickly return a set of candidates
  • Semantically equivalent or similar
  • May come from different architectures
  • May be generated by different compilers and options
Applications

• Plagiarism Detection
• Malware Classification
• Vulnerability Search
  • Emerging topic: vulnerability search in IoT
Internet of Things
Firmwares

Operating systems to IoT devices

Vulnerability
Open source libraries e.g., OpenSSL

When **new vulnerabilities** are discovered in OpenSSL, all firmware using it may be affected e.g., Heartbleed
Vulnerability Detection

Vulnerability

Similar?

Similar?

Similar?

Similar?

i.e., Heartbleed

Important!
Challenges for **Binary Code Search**

Cross-Platform

- x86
- ARM
- MIPS

Similar or not similar? It’s a problem!

Scalability
An Example

```
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
cmove     eax, edx
pop     ebx
ret

pop     ebx
ret
```

```
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

bne      $v0, $v1, locret_19830
nop

bne      $v0, $v1, locret_19830
nop

bne      $v0, $v1, locret_19830
nop

la       $v0, loc_20080

ja       $ra
nop
```

a) x86 assembly  

b) MIPS assembly
Existing Binary Code Search Techniques

• Syntax-based Approach
  • Mnemonic code sequence [S. M. Tabish et al. SIGKDD ’09; W. M. Khoo et al. MSR’13]
  • Control flow graph [H. Flake. et al. DIMVA’04; J. Pewny et al. Oakland’15; Eschweiler et al. NDSS’16]
  • Call graph [X. Hu et al. CCS’09]

• Semantics-based Approach
  • Tracelet [Y. David et al. PLDI’14]
  • Tree expression on basic blocks [J. Pewny et al. ACSAC’14]
  • Symbolic execution [D. Gao et al. ICS’08; J. Ming, et al ISC’12]
Search for known vulnerabilities

Key challenge: cross-platform code search

• String
  • Backdoors in devices
  • Lack of generality

• “Multi-MH & Multi-k-MH” [Pewny et al. Oakland’15]
  • Control-flow graph + I/O pairs
  • Lack of scalability

• “DiscovRe” [Eschweiler et al. NDSS’16]
  • Control-flow graph + Statistics features
  • Lack of scalability
  • Lightweight filtering is unreliable
Pair-wise graph matching is expensive!

-> More complex feature representation
  -> More accurate
  -> Less search efficiency

Graph matching is NP-hard problem!
The most efficient algorithm is $O(n^3)$ for two graph matching

“Multi-MH & Multi-k-MH” [Pewny et al. Oakland’15]
“DiscovRe” [Eschweiler et al. NDSS’16]

It is impossible to conduct pair-wise graph matching in large code repo!

Vulnerability Search Engine

CFG Ranking List
A similar problem

• Image search: tag a similar object in millions of images
We don’t compare images one by one
How can we learn high-level feature representations from CFGs?

Each dimension represents a high-level property of the original CFG!

c. Codebook
d. Feature vector
Codebook-based approach (Genius, CCS’16)

(i) Binary functions

(ii) Feature extraction

(iii) Vector quantization

(iv) High-level features

(a) Raw Feature Extraction

(b) Feature Learning

(c) High-level feature encoding

(d) LSH and search
Raw feature extraction

• Attributed Control Flow Graph

**Definition 1.** (Attributed Control Flow Graph) The attributed control flow graph, or ACFG in short, is a directed graph \( G = (V, E, \phi) \), where \( V \) is a set of basic blocks; \( E \subseteq V \times V \) is a set of edges representing the connections between these basic blocks, and \( \phi : V \to \Sigma \) is the labeling function which maps a basic block in \( V \) to a set of attributes in \( \Sigma \).

<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>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>
An example of ACFG

(a) Partial control flow graph of dtls1_process_heartbeat

(b) The corresponding ACFG
Feature learning

Learn a codebook from raw features. Each code word represents one property shared by raw features.

Codebook

code word

Codebook

code word
Feature learning

• Codebook
  • Each code word is the **centroid** of a cluster of ACFGs

• Clustering on raw features (ACFGs)
  • K-means, hierarchical-k-means, etc.

• Codebook size
  • Predetermined by # of clusters
  • Bigger Size -> Higher accuracy & Lower Encoding Performance
High-level feature encoding

- **VLAD encoding:**
  - Measure the distance between a given ACFG to each centroid
  - To normalize the feature vector, we use graph similarity instead
  - VLAD quantizer is shown below:

\[
q(g_i) = \sum_{g_i:NN(g_i)=c_j} [(1(1 = j)\kappa(g_i, c_1), \ldots, 1(n = j)\kappa(g_i, c_n)]^T
\]

The similarity score is calculated via graph edit distance
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, 0.4, 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
Evaluating Genius

• Dataset Preparation
  • 0.6 billion functions and hundreds of vulnerabilities

• Baseline Preparation
  • Compare with Multi-MH and Multi-k-MH, DiscoveRe, Centroid.

• Performance Evaluation
  • TPR and FPR
  • Search Efficiency
  • Preparation Time

• Case Studies

Genius: Graph Encoding for Bug Search
Evaluation: Datasets

• Baseline Dataset
  • BusyBox (v1.21 and v1.20), OpenSSL (v1.0.1f and v1.0.1a) and coreutils (v6.5 and v6.7)
  • x86, ARM, MIPS; all 32 bit
  • 568,134+ functions.

• Firmware Image Dataset
  • 33,045 firmware images
  • 26 different vendors

• Vulnerability Dataset
  • 154 vulnerable functions
Evaluation: Baseline Comparison

- **DiscovRe** [Eschweiler et al. NDSS’16]
  - Re-implemented its core part about graph matching and feature learning

- **Multi-MH and Multi-k-MH** [Pewny et al. Oakland’15]
  - Compared on the same dataset

- **Centroid** [Chen et al. USENIX Security’15]
  - Re-implemented its algorithm
  - A simple encoding that converts a CFG into a number
Evaluation: True Positive Rate

![Graph showing recall rates across different threshold K]

- Genius
- DiscovRe without filtering
- DiscovRe with filtering
- Centroid

a) Recall rates across different threshold K
Evaluation: Search Efficiency

Figure 2. The CDFs of search time on Dataset I.
Evaluation: Case Study I

• Search 2 vulnerabilities on 8126 firmware images
  • CVE-2015-1791: top 50 candidates, 14 firmware images potentially affected, 10 confirmed. Two vendors: D-Link and Belkin.

  • CVE-2014-3508: 24 firmware images potentially vulnerable, 13 confirmed. Vendors are CenturyLink, D-Link and Actiontec.
Evaluation: Case Study II

- Search two latest firmware images for all vulnerabilities
  - D-Link DIR-810 models
  - 154 Vulnerabilities
  - Search time: < 0.1s
  - Check top 100 candidates

Table 4: Case study results for Scenario II

<table>
<thead>
<tr>
<th>DIR-810L_REVBR_FIRMWARE_2.03B02</th>
<th>DIR-810L_REVBR_FIRMWARE_2.02.B01</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE</td>
<td>Patched</td>
</tr>
<tr>
<td>-----</td>
<td>---------</td>
</tr>
<tr>
<td>CVE-2016-0703</td>
<td>No</td>
</tr>
<tr>
<td>CVE-2015-1790</td>
<td>No</td>
</tr>
<tr>
<td>CVE-2015-1791</td>
<td>Yes</td>
</tr>
<tr>
<td>CVE-2015-0289</td>
<td>No</td>
</tr>
<tr>
<td>CVE-2014-8275</td>
<td>No</td>
</tr>
<tr>
<td>CVE-2015-0209</td>
<td>No</td>
</tr>
<tr>
<td>CVE-2015-3195</td>
<td>No</td>
</tr>
<tr>
<td>#</td>
<td>#</td>
</tr>
<tr>
<td>#</td>
<td>#</td>
</tr>
</tbody>
</table>
Limitations of Genius

• Encoding is still expensive
  • 1 graph comparison for each word in codebook

• Feature dimension has to be small
  • Confine the search accuracy

• Codebook generation is expensive
  • May take a week to retrain the codebook
Neural Network-based Graph Embedding for Cross-Platform Binary Code Similarity Detection

Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, Dawn Song
Two unbeatable advantages of neural network-based similarity detection

Previous approaches on expensive graph-matching based algorithms to detect similarity are very SLOW!

We will show that a neural network-based approach can be much more efficient!
Takeaways

Message 1. Our work is one of the first demonstrations to show that deep learning techniques can be applied to binary analysis

Message 2. We hope our work can foster more investigations on using deep learning approaches for binary analysis
Overall workflow

Previous approaches
• Manually designed graph-matching-based algorithms
• Slow
• Effectiveness is limited by graph-matching

Our approaches:
• Deep graph embedding network
• Design a neural network to extract the features automatically
• Combine Struct2vec and Siamese network

Firmware files → Raw Feature Extraction (disassembler) → Attributed CFG → Attributed CFG → Embedding Network → Embeddings → Cosine similarity

Vulnerability → Embeddings
Our approach: structure2vec

Take a closer look at the 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
Take a closer look at propagation
1. Application-independent pretraining
   - Compile given source code into different platforms using different compilers and different optimization-levels
   - A pair of binary functions compiled from the same source code is labeled with +1
   - Otherwise, -1

2. Application-dependent retraining
   - Human can label similar and dissimilar pairs of binary functions
   - This additional training data can be used in a retraining process
Training Data Details

• OpenSSL (version 1.0.1f and 1.0.1u)
  • Compiled using GCC v5.4
  • Emit code to x86, MIPS, ARM
  • Using optimization level O0-O3

<table>
<thead>
<tr>
<th></th>
<th>Training</th>
<th>Validation</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86</td>
<td>30,994</td>
<td>3,868</td>
<td>3,973</td>
</tr>
<tr>
<td>MIPS</td>
<td>41,477</td>
<td>5,181</td>
<td>5,209</td>
</tr>
<tr>
<td>ARM</td>
<td>30,892</td>
<td>3,805</td>
<td>3,966</td>
</tr>
<tr>
<td>Total</td>
<td>103,363</td>
<td>12,854</td>
<td>13,148</td>
</tr>
</tbody>
</table>
Visualizing the embeddings
Accuracy: ROC curve on test data
Serving time (per function processing time)

Previous work: a few secs to a few mins

Now: a few milliseconds

2500 × to 16000 × faster!
Training time

Previous work: > 1 week

Now: < 30 mins
### Identified Vulnerabilities in Large Scale Dataset

#### Among top 50:
- 42 out of 50 are confirmed vulnerabilities

#### Previous work:
- 10/50

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Vendor</th>
<th>Firmware</th>
<th>Binary File</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_new_session_ticket</td>
<td>D-Link</td>
<td>DAP-1582 Firmware 1.00</td>
<td>wp_supplicant.dcfs</td>
<td>0.9623745096</td>
</tr>
<tr>
<td>port_check_v6</td>
<td>D-Link</td>
<td>DES-1210-28_REV3_FIRMWARE 3.12.015</td>
<td>in.ftpd.dcfs</td>
<td>0.955408692</td>
</tr>
<tr>
<td>sub_42EE7C</td>
<td>TP-Link</td>
<td>TD-W8970B_V1_140024</td>
<td>racoon.dcfs</td>
<td>0.95474193</td>
</tr>
<tr>
<td>sub_42EE7C</td>
<td>TP-Link</td>
<td>TD-W8970B_V1_140024</td>
<td>racoon.dcfs</td>
<td>0.95474193</td>
</tr>
<tr>
<td>pna_parse_file</td>
<td>TP-Link</td>
<td>Archer_D0_V1_140104</td>
<td>racoon.dcfs</td>
<td>0.949034439</td>
</tr>
<tr>
<td>sub_422B8C</td>
<td>TP-Link</td>
<td>TD-W8970B_V1_140024</td>
<td>racoon.dcfs</td>
<td>0.949583828</td>
</tr>
<tr>
<td>sub_422B8C</td>
<td>TP-Link</td>
<td>TD-W8970B_V1_140024</td>
<td>racoon.dcfs</td>
<td>0.949583828</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>DD-wrt</td>
<td>dd-wrt.v3-23838 NFWD-2 X3.x mega-WNR3500v2 VC</td>
<td>openwrt.dcfs</td>
<td>0.94666827</td>
</tr>
<tr>
<td>wudebugserver</td>
<td>TP-Link</td>
<td>WD6490 V2_130115</td>
<td>httpd.dcfs</td>
<td>0.946312708</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Netgear</td>
<td>tomato-Cisco-M310v2-NVRAM32K-1.28-RT-N5S-MIPS32R2-110-PL-Mini</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459330444</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Tomato_by_Shibby</td>
<td>tomato-K26-1.28-RT-MIPS32R_109-Mini</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459330444</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Tomato_by_Shibby</td>
<td>tomato-K26UB-1.28-RT-N5S-MIPS32R2-110-VPN</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459320848</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Tomato_by_Shibby</td>
<td>tomato-E4200USB-NVRAM60K-1.28-RT-MIPS32R2-110-PL-RT</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459320848</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Tomato_by_Shibby</td>
<td>tomato-E4200USB-NVRAM60K-1.28-RT-MIPS32R2-110-PL-VPN</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459320848</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Tomato_by_Shibby</td>
<td>tomato-K26USB-1.28-RT-MIPS32R2-109-AO</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459320848</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Tomato_by_Shibby</td>
<td>tomato-Netgear-3500Lv2-K26USB-1.28-RT-N5S-109-AO</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459320848</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Tomato_by_Shibby</td>
<td>tomato-E4200USB-NVRAM60K-1.28-RT-MIPS32R2-109-AO</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459320848</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Tomato_by_Shibby</td>
<td>tomato-E4200USB-NVRAM60K-1.28-RT-N5S-MIPS32R2-110-Nocat-VPN</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459320848</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Tomato_by_Shibby</td>
<td>tomato-K26USB-1.28-RT-N5S-MIPS32R2-115-PL-LI600N</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459320848</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Tomato_by_Shibby</td>
<td>tomato-E4200USB-NVRAM60K-1.28-RT-MIPS32R2-108-PL-VPN</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459320848</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Tomato_by_Shibby</td>
<td>tomato-E4200USB-NVRAM60K-1.28-RT-MIPS32R2-108-PL-VPN</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459320848</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Tomato_by_Shibby</td>
<td>tomato-E4200USB-NVRAM60K-1.28-RT-MIPS32R2-108-PL-VPN</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459320848</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Tomato_by_Shibby</td>
<td>tomato-E4200USB-NVRAM60K-1.28-RT-MIPS32R2-108-PL-VPN</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459320848</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Tomato_by_Shibby</td>
<td>tomato-E4200USB-NVRAM60K-1.28-RT-MIPS32R2-108-PL-VPN</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459320848</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Tomato_by_Shibby</td>
<td>tomato-E4200USB-NVRAM60K-1.28-RT-MIPS32R2-108-PL-VPN</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459320848</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Tomato_by_Shibby</td>
<td>tomato-E4200USB-NVRAM60K-1.28-RT-MIPS32R2-108-PL-VPN</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459320848</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Tomato_by_Shibby</td>
<td>tomato-E4200USB-NVRAM60K-1.28-RT-MIPS32R2-108-PL-VPN</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459320848</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Tomato_by_Shibby</td>
<td>tomato-E4200USB-NVRAM60K-1.28-RT-MIPS32R2-108-PL-VPN</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459320848</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Tomato_by_Shibby</td>
<td>tomato-E4200USB-NVRAM60K-1.28-RT-MIPS32R2-108-PL-VPN</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459320848</td>
</tr>
<tr>
<td>get_new_session_ticket</td>
<td>Tomato_by_Shibby</td>
<td>tomato-E4200USB-NVRAM60K-1.28-RT-MIPS32R2-108-PL-VPN</td>
<td>libssl.so.1.0.0.dcfs</td>
<td>0.9459320848</td>
</tr>
</tbody>
</table>
Takeaways

Message 3. Deep learning approaches can be not only more effective, but also more efficient in learning embedding representations for binary programs.

Message 4. Program analysis can be a novel application domain of deep learning techniques toward a more secure world.