Revisiting Function Identification with Machine Learning

January 2021

Hyungjoon Koo, Soyeon Park, and Taesoo Kim
Function Identification Problem

Problem definition
- Discover a set of function boundaries in a binary
- No symbol or debugging information readily available

A binary function is
- Defined by a developer from source code
- Generated by a compiler (e.g., stack canary check)
- Inserted by a linker (e.g., CRT function)

Why important?
- Serve as a basis for reversing executable binaries
- Many applications: binary transformation, binary similarity analysis, call graph reconstruction
- Almost every binary analysis tool includes a feature of function recognition
Common Challenges

Code optimization often blurs a clear function signature
e.g., function inlining

Compiler-generated code or compiler-specific heuristics

Mixed code and data
e.g., jump table

Non-returning functions
e.g., ending with a call

Code from manually written assembly
Existing Approaches

Linear disassembly
- Linearly disassemble all code (e.g., objdump)
- Apply function signature matching (e.g., function prologue)
- Downside: no pattern, code/data intermixed

Recursive traversal
- Begin from an entry point
- Follow a direct control flow transfer
- Downside: indirectly reachable (or unreachable) functions cannot be recognized

ML-oriented approach
- Conditional random field (CRF)
- Weighted prefix tree
- Recurrent neural network (RNN)
## Summary of Prior Works

<table>
<thead>
<tr>
<th>Tool</th>
<th>Year</th>
<th>Dataset</th>
<th>Artifacts</th>
<th>Arch</th>
<th># Bins</th>
<th>Compared To</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nucleus</td>
<td>2017</td>
<td>SPEC2006, nginx, lighttpd, opensshd, vsfspd, exim</td>
<td>Y</td>
<td>x86/x64</td>
<td>476</td>
<td>Dyninst, ByteWeight, IDA</td>
</tr>
<tr>
<td>Qiao et al.</td>
<td>2017</td>
<td>GNU Utils, SPEC2006, Glibc</td>
<td>N</td>
<td>x86/x64</td>
<td>2,488</td>
<td>ByteWeight, Shin:RNN</td>
</tr>
<tr>
<td>Jima</td>
<td>2019</td>
<td>GNU Utils, SPEC2017, Chrome</td>
<td>Y</td>
<td>x86/x64</td>
<td>2,860</td>
<td>ByteWeight, Shin:RNN, IDA Free, Ghidra, Nucleus</td>
</tr>
<tr>
<td>ByteWeight</td>
<td>2014</td>
<td>GNU Utils</td>
<td>Y</td>
<td>x86/x64</td>
<td>2,200</td>
<td>Dyninst, BAP, IDA</td>
</tr>
<tr>
<td>Shin:RNN</td>
<td>2015</td>
<td>GNU Utils</td>
<td>N</td>
<td>x86/x64</td>
<td>2,200</td>
<td>ByteWeight</td>
</tr>
<tr>
<td>FID</td>
<td>2017</td>
<td>GNU coreutils</td>
<td>N</td>
<td>x86/x64</td>
<td>4,240</td>
<td>IDA, ByteWeight</td>
</tr>
</tbody>
</table>
Our Focus

Is NOT about

◦ Verifying the correctness of prior evaluations
◦ Ranking the existing approaches (i.e., which one is the best?)

Is about

◦ Filling the void of what has been overlooked or misinterpreted
◦ Revisiting the previous datasets, metrics, and evaluations

→ Has the function identification problem been fully addressed?
Research Questions

Is the previous dataset appropriate?
Has a function detection problem been fully resolved?
Are ML-oriented approaches superior to deterministic ones?
Is the current metric (i.e., precision, recall, F1) fair enough?
Appropriateness of Dataset

GNU utilities (129)
- ByteWeight released 16 binutils, 104 coreutils, and 9 findutils
- coreutils has a static library (libcoreutils.a) in common -> redundant functions
- Most subsequent works use them for their evaluations

Normalization
- ML approaches take “normalization” as a pre-processing step
- 17.6K / 146K (12.1%) remain unique
- 91.4% in a test set has been discovered in a training set -> overfitting

<table>
<thead>
<tr>
<th>Group</th>
<th>Files</th>
<th>Funs</th>
<th>Set</th>
<th>Group</th>
<th>Files</th>
<th>Funs</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1</td>
<td>57</td>
<td>10,996</td>
<td>train</td>
<td>Group 6</td>
<td>49</td>
<td>12,236</td>
<td>train</td>
</tr>
<tr>
<td>Group 2</td>
<td>55</td>
<td>9,475</td>
<td>train</td>
<td>Group 7</td>
<td>48</td>
<td>12,197</td>
<td>train</td>
</tr>
<tr>
<td>Group 3</td>
<td>51</td>
<td>18,442</td>
<td>train</td>
<td>Group 8</td>
<td>46</td>
<td>12,324</td>
<td>train</td>
</tr>
<tr>
<td>Group 4</td>
<td>57</td>
<td>13,779</td>
<td>train</td>
<td>Group 9</td>
<td>46</td>
<td>20,680</td>
<td>test</td>
</tr>
<tr>
<td>Group 5</td>
<td>55</td>
<td>13,481</td>
<td>train</td>
<td>Group 10</td>
<td>52</td>
<td>13,519</td>
<td>train</td>
</tr>
</tbody>
</table>
Re-interpretation of Prior Evaluations

Remarkable reports
- ByteWeight: F1 of 98.8 for ELF x64
- Shin’s RNN: F1 of 98.3
- LEMNA (Shin’s RNN re-implementation): 99.99% accuracy

Are we there yet?
- Re-experimentation with a different dataset (e.g., SPEC2017, other utilities of our choice)
- Retraining the ByteWeight model with our dataset: F1 of 78.0
- LEMNA’s accuracy comes from the number of decisions per byte (i.e., large # of true negatives)
- The LEMNA results with our dataset: precision of 94.5, recall of 86.1
Effectiveness of ML Techniques

Comparison of the number of true functions

- RNN VS Deterministic approaches

- Non-returning function (i.e., ending with call, jump, or __exit) detection

<table>
<thead>
<tr>
<th>Tool</th>
<th># of Missing</th>
<th>Total</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDA Pro</td>
<td>0</td>
<td>9,409</td>
<td>0.00%</td>
</tr>
<tr>
<td>Ghidra</td>
<td>54</td>
<td>9,409</td>
<td>0.57%</td>
</tr>
<tr>
<td>Nucleus</td>
<td>1,186</td>
<td>9,409</td>
<td>12.60%</td>
</tr>
<tr>
<td>Byteweight</td>
<td>4,615</td>
<td>9,409</td>
<td>49.05%</td>
</tr>
<tr>
<td>Byteweight*</td>
<td>2,024</td>
<td>5,125</td>
<td>39.49%</td>
</tr>
<tr>
<td>Shin:RNN</td>
<td>24</td>
<td>250</td>
<td>9.60%</td>
</tr>
</tbody>
</table>
Rethinking of Current Metrics (1/2)

Precision, Recall and F1 values

\[
P = \frac{|TP|}{|TP| + |FP|}, \quad R = \frac{|TP|}{|TP| + |FN|}, \quad F_1 = \frac{2 \times P \times R}{P + R}
\]

[Case 1] Non-continuous functions

```c
MagickExport ImageInfo *AcquireImageInfo(void) {
    ImageInfo *image_info;
    image_info=(ImageInfo *) AcquireMagickMemory(sizeof(*image_info));
    if (image_info == (ImageInfo *) NULL)
        ThrowFatalException(ResourceLimitFatalError, "MemoryAllocationFailed");
    getImageInfo(image_info);
    return(image_info);
}
```

```assembly
0x4C6BC0 push    rbx
0x4C6BC1 mov     edi, 4198h ; size
0x4C6BC6 call    AcquireMagickMemory
0x4C6BCB test    image_info, image_info
0x4C6BCE jz      loc_4C6BE0
0x4C6BD0 mov     rbx, image_info
0x4C6BD3 mov     rdi, image_info ; image_info
0x4C6BD6 call    GetImageInfo
0x4C6BDE mov     rax, image_info
0x4C6BDF ret
0x4C6BE0 call    AcquireImageInfo.part.2

; ImageInfo *__cdecl AcquireImageInfo.part.2()
0x402554 push    rbx
0x402555 sub     Rsp, 40h
0x402559 mov     rdi, rsp ; exception

... 0x4025C4 call    DestroyExceptionInfo
0x4025C9 call    MagickCoreTerminus
0x4025CE mov     edi, 1 ; status
0x4025D3 call    __exit
```
Rethinking of Current Metrics (2/2)

[Case 2] Ground truth from debugging information
- objdump or nm read function symbols merely from a symbol table
- Ghidra discovers more functions with a frame description entry (FDE) by parsing debugging sections
- Example (13,380 cases from cpugcc_r- amd64- clang- O1)

```
; _int64 __fastcall atol_317(const char *__nptr)
0x9c0a20  xor  esi, esi
0x9c0a22  mov  edx, 0Ah
0x9c0a27  jmp  __strtol
```

- Also, we need to consider cases when referring FDE may point to an incorrect function location!
Our Dataset and Tools

Dataset
- SPEC2017: 16 different binaries (120)
- 4 Utilities including nginx, vsftpd, and openssl (32)
- x64 ELFs that compiled with gcc/clang using O0-3 optimization levels

Tools
- Deterministic tools
  - IDA
  - Ghidra
  - Nucleus
- ML-embedded tools
  - LEMNA implementation of Shin:RNN
  - ByteWeight signature from the latest version of BAP
  - ByteWeight (for retraining): originally released version
## Evaluation

<table>
<thead>
<tr>
<th>Tool</th>
<th>Ground Truth</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nucleus</td>
<td>796,069</td>
<td>86.91</td>
<td>94.21</td>
<td>90.42</td>
</tr>
<tr>
<td>IDA Pro</td>
<td>796,069</td>
<td>99.55</td>
<td>87.88</td>
<td>93.35</td>
</tr>
<tr>
<td>Ghidra</td>
<td>796,069</td>
<td>93.55</td>
<td>98.5</td>
<td>96.03</td>
</tr>
<tr>
<td>ByteWeight</td>
<td>514,082</td>
<td>63.15</td>
<td>60.26</td>
<td>61.67</td>
</tr>
<tr>
<td>ByteWeight (retrained)</td>
<td>463,323</td>
<td>85.44</td>
<td>71.78</td>
<td>78.02</td>
</tr>
<tr>
<td>Shin:RNN (LEMNA impl)</td>
<td>80,532</td>
<td>94.50</td>
<td>86.09</td>
<td>90.10</td>
</tr>
</tbody>
</table>
Insights and Conclusion

Insights
- State-of-the-art function detection tools work well for binaries without optimizations
- Not a single tool dominates all the others
- Difficult to claim an ML-centric approach surpasses deterministic ones
- The current metrics may not be reasonable in some cases

Conclusion
- A function detection problem has yet been fully resolved
- Better metrics and dataset for fair comparison are needed
Q&A

Thank you