Machine Learning for Program Analysis (MLPA)

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

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

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**

Group	Files	Funcs	Set	Group	Files	Funcs	Set
Group 1	57	19,996	train	Group 6	49	12,236	train
Group 2	55	9,475	train	Group 7	48	12,197	train
Group 3	51	18,442	train	Group 8	46	12,324	train
Group 4	57	13,779	train	Group 9	46	20,680	test
Group 5	55	13,481	train	Group 10	52	13,519	train

Re-interpretation of Prior Evaluations

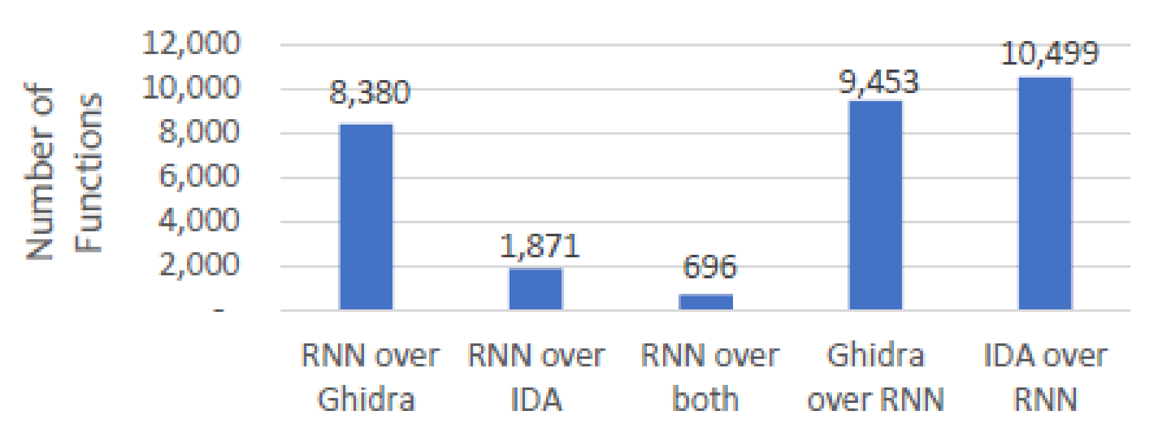
Remarkable reports
- ByteWeight: F1 of 98.8 for ELF x64
- Shin's RNN: F1 of 98.3
- LEMNA (Shin's RNN re-implementation): 99.99% accuracy

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

Tool	# of Missing	Total	Rate
IDA Pro	0	9,409	0.00%
Ghidra	54	9,409	0.57%
Nucleus	1,186	9,409	12.60%
Byteweight	4,615	9,409	49.05%
Byteweight*	2,024	5,125	39.49%
Shin:RNN	24	250	9.60%

Rethinking of Current Metrics (1/2)

Precision, Recall and F1 values

$$P = \frac{|TP|}{|TP| + |FP|},\ R = \frac{|TP|}{|TP| + |FN|},\ F1 = \frac{2 * P * R}{P + R}$$

[Case 1] Non-continuous functions

```
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);
}
```

```
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
0x4C6BDB   mov     rax, image_info
0x4C6BDE   pop     image_info
0x4C6BDF   retn
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

Tool	Ground Truth	Precision	Recall	F1
Nucleus	796,069	86.91	94.21	90.42
IDA Pro	796,069	99.55	87.88	93.35
Ghidra	796,069	93.55	98.5	96.03
ByteWeight	514,082	63.15	60.26	61.67
ByteWeight (retrained)	463,323	85.44	71.78	78.02
Shin:RNN (LEMNA impl)	80,532	94.50	86.09	90.10

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