A LOOK BACK ON A FUNCTION IDENTIFICATION PROBLEM

Hyungjoon Koo, Soyeon Park, and Taesoo Kim

SUNG KYUN KWAN UNIVERSITY(SKKU)

Georgia Tech

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 (e.g., user-defined function)
- 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, malware anlaysis, call graph reconstruction, etc.
- Almost every binary analysis tool includes a feature of function recognition

Common Challenges

Code optimization often blurs a clear function signature
- e.g., removing function prologue and epilogue, 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 (tail call)

Code from a manually written assembly

Existing Approaches

Rule-based approach

- Disassemble code
  - Linear disassembly: Linearly disassemble all code (e.g., `objdump`)
  - Recursive traversal: Starting from an entry point and following a direct control flow transfer
- Seek function signature matching (e.g., function prologue)
- Downsides
  - Cannot identify functions in case of missing signatures (patterns)
  - Disassembly failure: code/data intermixed or indirectly reachable (or unreachable) functions cannot be recognized

Machine learning oriented approach

- Conditional random field (CRF)
- Weighted prefix tree
- Recurrent neural network (RNN)

Summary of Prior Works

Work	Year	Dataset	F1	Arch	Binaries	Compared To
Nucleus	2017	SPEC2006, nginx, lighttpd, opensshd, vsfpd, exim	0.97	x86/x64	476	Dyninst, ByteWeight, IDA
Qiao et al.	2017	GNU Utils, SPEC2006, Glibc	0.985	x86/x64	2,488	ByteWeight, Shin:RNN
Jima	2019	GNU Utils, SPEC2017, Chrome	0.997	x86/x64	2,860	ByteWeight, Shin:RNN, IDA Free, Ghidra, Nucleus
ByteWeight	2014	GNU Utils	0.929	x86/x64	2,200	Dyninst, BAP, IDA
Shin:RNN	2015	GNU Utils	0.983	x86/x64	2,200	ByteWeight
FID	2017	GNU coreutils	0.930	x86/x64	4,240	IDA, ByteWeight

A Look Back on Function Identification

(Q1) Is the previous dataset appropriate for evaluation?

(Q2) Have prior evaluations been properly interpreted?

(Q3) Is the current metric (i.e., precision, recall, F1) fair enough?

(Q4) Are recent advances with an ML-centered approaches superior to rule-based ones?

(Q5) Is there a tool’s own characteristic or behavior?

Our Focus

Is NOT about

- Raising a question on the reproducibility or correctness of prior work
- Ranking the existing approaches (i.e., which one is the best and the worst?)

Is about

- Filling the void of what has been overlooked or misinterpreted
- Revisiting the previous datasets, metrics, and evaluations
- Bringing up the question of “Has the function identification problem been fully addressed?”

Q1. Appropriateness of Dataset

GNU utilities (129)

- `ByteWeight` released 16 **`binutils`**, 104 **`coreutils`**, and 9 **`findutils`**
- Most subsequent works utilize them for their evaluations
- **`coreutils`** has a static library (`libcoreutils.a`) in common → many redundant functions

Normalization for ML approaches

- Pre-processing step to reduce the number of instructions to feed a model
- 17.6K / 146K (12.1%) remain unique (ByteWeight)
- 91.4% in a test set has been discovered in a training set

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

Q2. Re-interpretation of Prior Evaluations

Noticeable 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

Q3. Rethinking of Current Metrics

(Case 1) Non-continuous functions

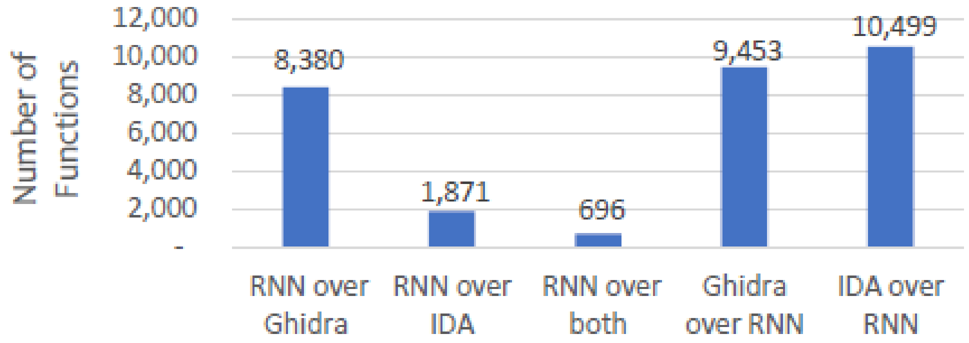
```
MagickExport ImageInfo *AcquireImageInfo(void) {
  ImageInfo *image_info;
  image_info=(ImageInfo *) AcquireMagickMemory(sizeof(*
     image_info));
  if (image_info == (ImageInfo *) NULL)
    ThrowFatalException(ResourceLimitFatalError,"
     MemoryAllocationFailed");
  GetImageInfo(image_info);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

Q4. Effectiveness of ML Techniques

Comparison of the number of true functions

- RNN VS Rule-based approaches



- Non-returning function detection

(ending with `call`, `jump`, or `__exit)`

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%

Q5. Faithfulness of Tools

[Case study – IDA Pro] Under Reporting

- Disassembly with recursive traversal
- Intentionally does not seek unused functions (e.g., from an object file at link time)

[Case study – Ghidra] Over Reporting

- `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; e.g, 13,380 such 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
```

- Recall Q3 → This may distort current metrics!

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 rule-based ones
- The current metrics may not be reasonable in corner cases
- The capability of function identification relies on a tool's strategic choice

Conclusion

- A function detection problem has *yet* been fully resolved, necessitating better dataset and metrics

Q&A