Compiler-Agnostic Function Detection in Binaries

Dennis Andriesse[†], Asia Slowinska, Herbert Bos[†]

[†]Vrije Universiteit Amsterdam

EuroS&P 2017

Introduction

Disassembly in Systems Security

Disassembly is the backbone of all binary-level systems security work (and more)

- Control-Flow Integrity
- Automatic Vulnerability/Bug Search
- Lifting binaries to LLVM/IR (e.g., for reoptimization)
- Malware Analysis
- Binary Hardening
- Binary Instrumentation
- ...

Introduction

Results from Previous Work

Function detection currently the main disassembly challenge

- Even function start detection yields many FPs/FNs (20%+)
- Complex cases: non-standard prologues, tailcalls, inlining, . . .
- Binary analysis commonly requires function information

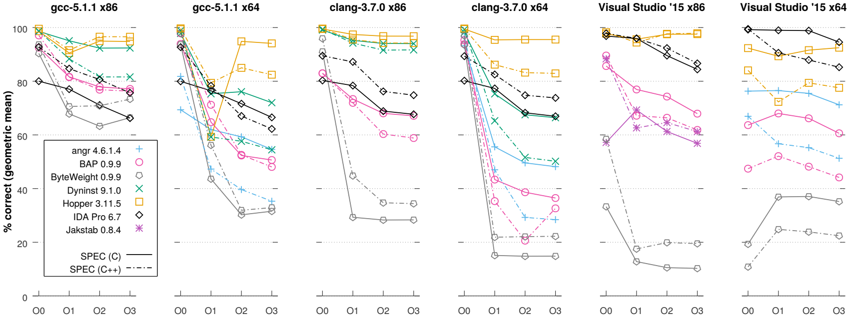


Figure: Correctly detected function start addresses

Introduction

Function Detection: False Negative

Listing: False negative indirectly called function for IDA Pro 6.7 (gcc compiled with gcc at O3 for x64 ELF)

```
6caf10 <ix86_fp_compare_mode>:
  6caf10:  mov  0x3f0dde(%rip),%eax
  6caf16:  and  $0x10,%eax
  6caf19:  cmp  $0x1,%eax
  6caf1c:  sbb  %eax,%eax
  6caf1e:  add  $0x3a,%eax
  6caf21:  retq
```

Function Detection: False Positive

Listing: False positive function (shaded) for Dyninst (`perlbench` compiled with `gcc` at `O3` for x64 ELF)

```
46b990 <Perl_pp_enterloop>:
            [...]
  46ba02:   ja      46bb50 <Perl_pp_enterloop+0x1c0>
  46ba08:   mov     %rsi,%rdi
  46ba0b:   shl     %cl,%rdi
  46ba0e:   mov     %rdi,%rcx
  46ba11:   and     $0x46,%ecx
  46ba14:   je      46bb50 <Perl_pp_enterloop+0x1c0>
            [...]
  46bb47:   pop     %r12
  46bb49:   retq
  46bb4a:   nopw    0x0(%rax,%rax,1)
  46bb50:   sub     $0x90,%rax
```

Current Approaches

Signature-Based Function Detection

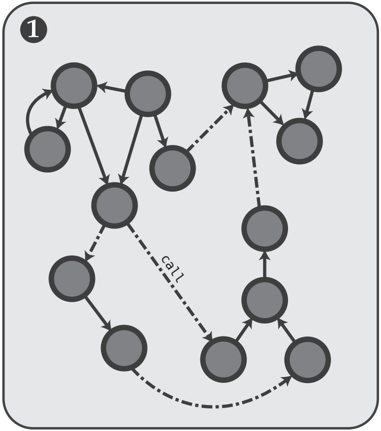
- Most current approaches scan for prologue/epilogue signatures
  - IDA Pro, Dyninst, ByteWeight (Bao et al. 2014), (Shin et al. 2015)
- Error-prone: sigs may be missing/optimized away
- Non-scalable: new sigs needed for every compiler version/platform
- Even machine learning approaches need continuous retraining

Overview of Our Approach

Compiler-Agnostic Function Detection

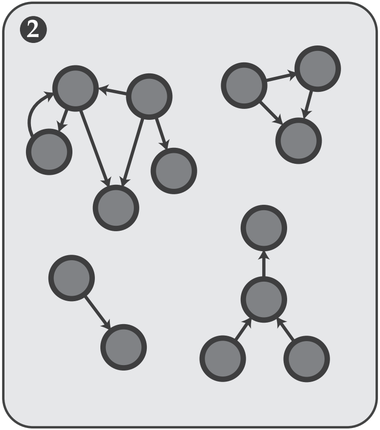
- We propose a signature-less approach based on structural analysis of the Control-Flow Graph (CFG)
- Basic premise: Weakly Connected Components Analysis
- Compiler-agnostic: no training/maintenance needed
- Able to detect all basic blocks of a function
- Inherent support for corner cases such as non-contiguous functions

Overview of Our Approach



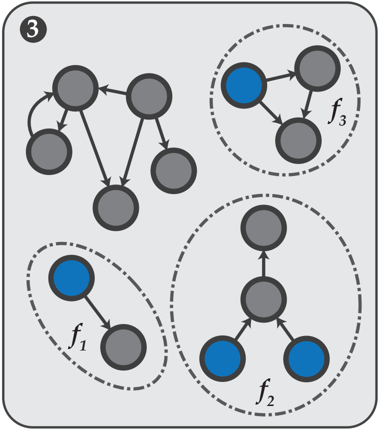
① Disassemble binary and generate interprocedural CFG (linear disassembly + switch/inline data detection)

Overview of Our Approach



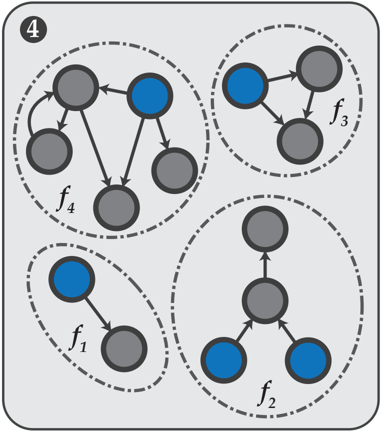
② Hide edges $e \in E_{call}$

Overview of Our Approach



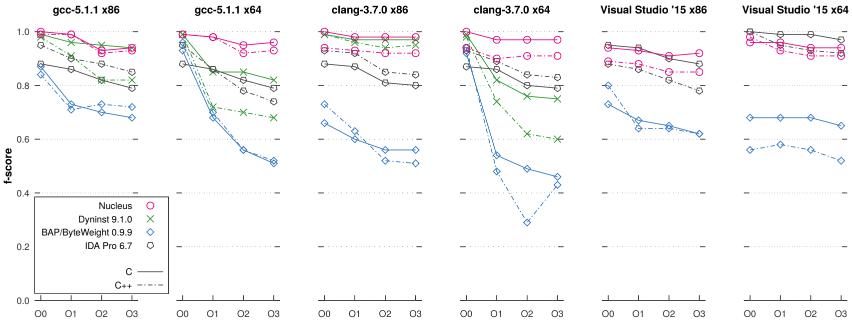
③ Locate directly called entry points and expand functions by following control flow (ignoring direction)

Overview of Our Approach



④ Find remaining functions using Connected Components Analysis, analyze control-flow to find entry points

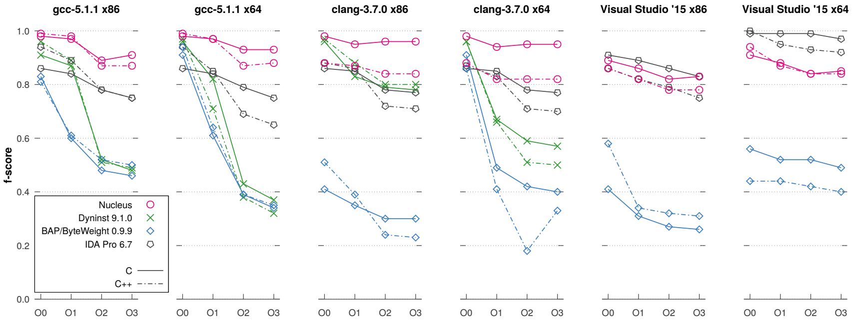
Evaluation



Function Start Detection

- Overall average F-score of 0.96 for SPEC CPU 2006 (similar for servers)
- Stable performance across compiler/platform/optimization level
- Main improvement over others: higher recall (fewer FNs)

Evaluation



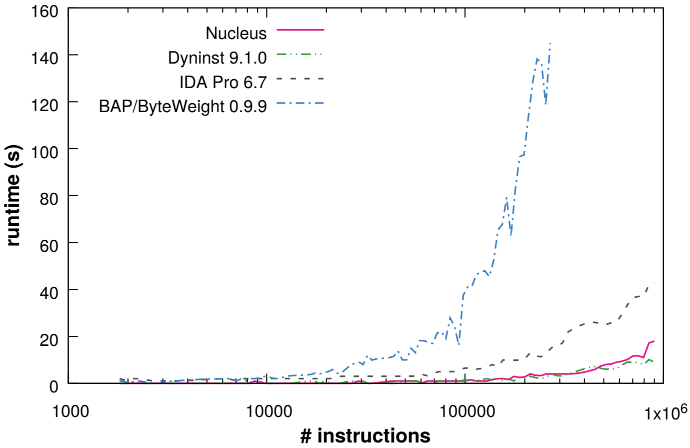
Function Boundary Detection

- Overall average F-score of 0.90 for SPEC CPU 2006
- Even better for C-only server tests (average F-score 0.97)
- Again, more stable than other approaches
- Best alternative: IDA Pro, average F-score of 0.84

More Results

- In-depth analysis of results (including FPs/FNs) in paper
- Most complex cases handled correctly (non-contiguous functions, multi-entry functions, ...)
- Main problematic case: tail calls

Evaluation



Runtime

- On par with fastest alternatives

Applicability to Malware Analysis

Resistance to Obfuscation

- Although this talk is in the Malware session, we do not explicitly target malware
- That said, our approach is agnostic of some basic obfuscation approaches
  - Instruction-level polymorphism
  - Mangling of function prologues/epilogues
  - Some control flow obfuscations (e.g., converting direct calls to indirect, branching functions, ...)
- But we make no promises for arbitrary obfuscations!

Issues with Evaluation of Machine Learning Approaches

Performance Discrepancies

- During our evaluation, noticed far lower performance for ByteWeight than previously reported (Bao et al. 2014)
- Mean F-score 0.32 points lower than expected
- Observation persists for `gcc` (v4.7–v5.1), `clang`, and Visual Studio
- Upon closer inspection, discovered issues with test suite used to evaluate *all* major machine learning-based function detection work (Bao et al. 2014 and Shin et al. 2015)

Issues with Evaluation of Machine Learning Approaches

Test Suite Issues

- Both Bao et al. and Shin et al. use ten-fold cross-validation to evaluate their work
- Partition test suite into training set (B_T, 90% of binaries) and evaluation set (B_E)
- Repeat ten times such that each binary is in B_E exactly once
- Crucial to ensure sufficient variation in test suite to prevent overfitting!

Issues with Evaluation of Machine Learning Approaches

Test Suite Issues

- Linux test suite used by Bao et al. and Shin et al. consists of `coreutils` (106 binaries), `binutils` (16 binaries), and `findutils` (7 binaries)
- Average `coreutils` binary shares 54% of its functions with *all other* `coreutils` binaries
- Average `coreutils` binary shares 94% of its functions with *at least one other* `coreutils` binary
- For the average `coreutils` binary in B_E, at least 86% of its functions are expected to occur in B_T
- **Large degree of overfitting in evaluation of machine learning approaches, re-evaluation needed**

Conclusion

- We introduced a novel compiler-agnostic function detector
- No maintenance/learning phase required
- More accurate results than existing approaches
- Inherent support for complex cases

- Available open source:
  `https://www.vusec.net/projects/function-detection/`
- Features export to IDA Pro $\rightarrow$ easy to use in real-world setting