allvm - Binary Decompilation

Sandeep Dasgupta
University of Illinois Urbana Champaign

March 25, 2016

Outline

Goal & Motivation

Possible Directions

Our Approach

Research Goal

- Research Goal
  - Obtain “richer” LLVM IR than native machine code.
- Motivation
  - Absence of source-code
  - What-you-see-is-not-what-you-execute
  - End-user security enforcement
  - Platform aware optimizations

Outline

Goal & Motivation

Possible Directions

Our Approach

Decompile `Machine Code` → `LLVM IR`

- Challenge: Quality
  - Reconstructing code and control flow - Much researched.
  - Variable recovery
  - Function & ABI rules recovery

Variable & Function parameter recovery

- Benefit
  - Important for many fundamental analysis like Dependence and Pointer analysis.
  - Make the recovered IR functional.
- State of the art
  - Grammatech
    - value set analysis (VSA) & structure aggregate identification.
  - Second Write
    - Heuristics for function parameter detection
    - Scalable VSA
  - TIE
    - Type Recovery

"Annotated" `Machine Code` $\longrightarrow$ `LLVM IR`

- Challenge: Annotations must be "minimal" & sufficient.

Ship `LLVM IR`

- Challenges: Adoption, risks to itellectual property

Outline

Goal & Motivation

Possible Directions

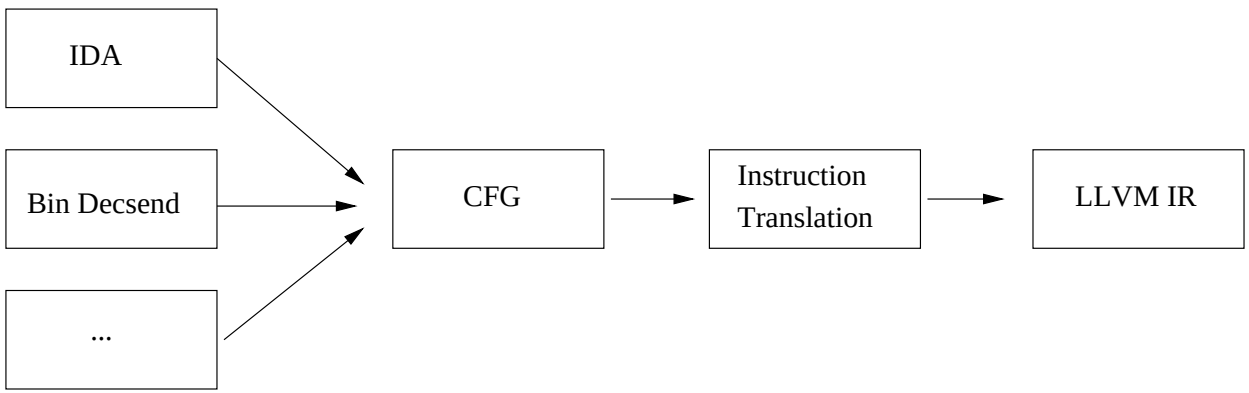
Our Approach

Our Approach

- Choose an existing decompilation framework.
- Experimentaion with various variable and type recovery strategies
- Use the knowledge to infer “minimal annotation”

Choosing Mcsema

- Functional `LLVM IR`
- Separation of modules: CFG recovery and CFG $\rightarrow$ `LLVM IR`
- Actively supported and open sourced



mcsema: Support & Limitations

- What Works
  - Integer Instructions
  - FPU and SSE registers
  - Callbacks, External Call, Jump tables
- What Needs to be done
  - FPU and SSE Instructions: Not fully supported
  - Exceptions
  - Better Optimizations

mcsema: demo