



ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

allvm - Binary Decompile

Sandeep Dasgupta
University of Illinois Urbana Champaign
March 29, 2016



Goal & Motivation

Possible Approaches

Our Approach

Ongoing Work

Backup



Research Goal

- Research Goal
 - Obtain “richer” LLVM IR than native machine code.
 - Enable advanced compiler techniques (e.g. pointer analysis, information flow tracking, automatic vectorization)



Why “richer” LLVM IR

- Source code analysis not possible
 - IP-protected software
 - Malicious executables
 - Legacy executables
- Source code analysis not sufficient
 - What-you-see-is-not-what-you-execute
- Platform aware optimizations



Goal & Motivation

Possible Approaches

Our Approach

Ongoing Work

Backup



The 3 Possible Approaches

- Decompile Machine Code \rightarrow LLVM IR
- “Annotated” Machine Code \rightarrow LLVM IR
- Ship LLVM IR



Decompile Machine Code \rightarrow LLVM IR

Benefits	Challenges
<ul style="list-style-type: none">• Easy to adopt• No compiler support needed	<ul style="list-style-type: none">• Reconstructing code and control flow• Variable recovery• Function & ABI rules recovery

- Tools Available: QEMU, BAP, Dagger, Mcsema, Fracture



“Annotated” Machine Code \rightarrow LLVM IR

Benefits	Challenges
<ul style="list-style-type: none">• Effective reconstruction• Minimal compiler support needed	<ul style="list-style-type: none">• Annotations must be<ul style="list-style-type: none">• Minimal• Compiler & IR independent• Adoption

- Tools Available: None



Ship LLVM IR

Benefits	Challenges
<ul style="list-style-type: none">• <i>No loss</i> of information	<ul style="list-style-type: none">• Adoption in Non LLVM based compilers• Code size bloat

- Tools Available: Portable Native Client, Renderscript, iOS, watchOS and tvOS apps.



Goal & Motivation

Possible Approaches

Our Approach

Ongoing Work

Backup



Our Approach

Long term goal

Minimal compiler-independent annotations to reconstruct high-quality IR

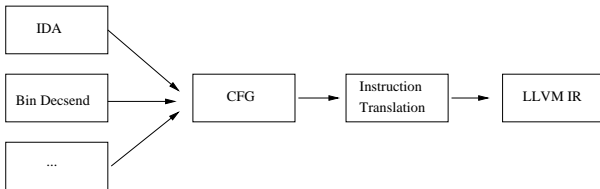
Short term goals

- ❶ Experiment with Machine Code \rightarrow LLVM IR, to **understand** the challenges better
 - To select an existing decompilation framework.
 - Experiment with different variable and type recovery strategies
- ❷ Design suitable annotations for what cannot be inferred without them



Choosing mcsema

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





mcsema: Demo

TM



Goal & Motivation

Possible Approaches

Our Approach

Ongoing Work

Backup



Variable & Function Parameter Recovery

- Benefit
 - Enables many fundamental analysis (Dependence, Pointer analysis)
 - Functional IR
- State of the art
 - Divine
 - State of the art variable recovery
 - Second Write
 - Heuristics for function parameter detection
 - Scalable variable and type recovery
 - TIE
 - Type recovery



Support & Limitations

- What Works
 - Integer Instructions
 - FPU and SSE registers
 - Callbacks, External Call, Jump tables
- In Progress
 - FPU and SSE Instructions: Not fully supported
 - Exceptions
 - Better Optimizations



Goal & Motivation

Possible Approaches

Our Approach

Ongoing Work

Backup



What-you-see-is-not-what-you-execute

The following compiler (Microsoft C++ .NET) induced vulnerability was discovered during the Windows security push in 2002

```
memset(password, '\0', len);  
free(password);
```

—————>

```
free(password);
```