A Framework for Automated Architecture-Independent Gadget Search

Thomas Dullien
Tim Kornau
Ralf-Philipp Weinmann

Overview

1. Goal
2. Motivation
3. History
4. Strategy
5. Algorithms
6. Current research state
7. Further work

Goal

The goal of this research is to be able to use return-oriented programming platform independently across multiple platforms.

Motivation I

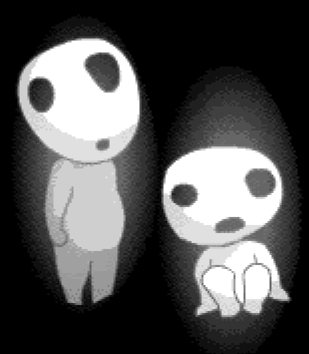
MIPS TECHNOLOGIES

Power™

IA32

ARM POWERED

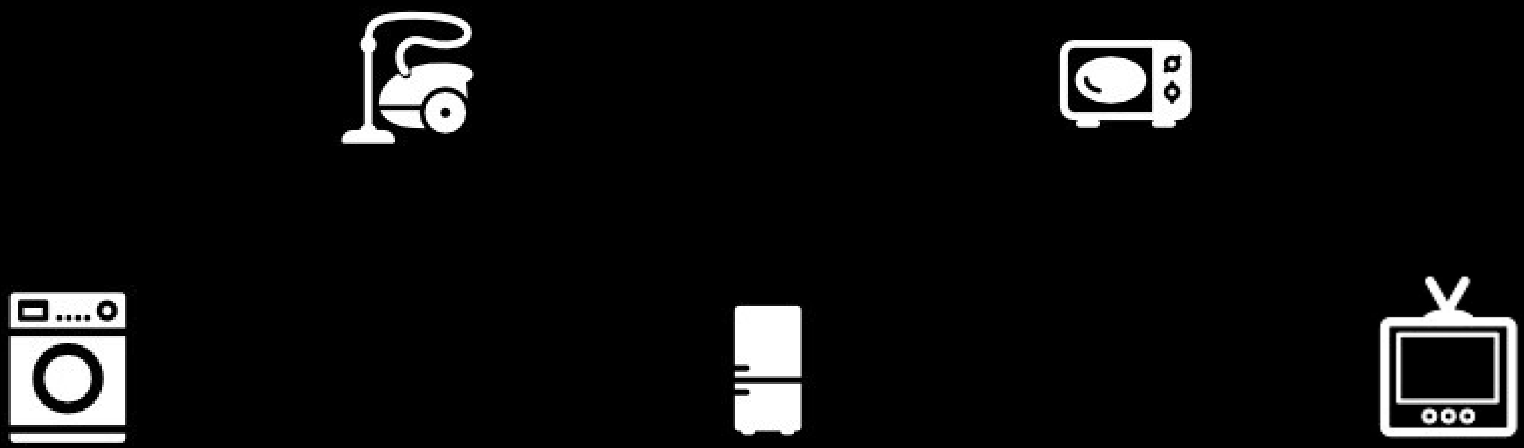
AMD64



Little spirits need access to a wide range of devices.
Because what is a device without a spirit ?

Motivation II

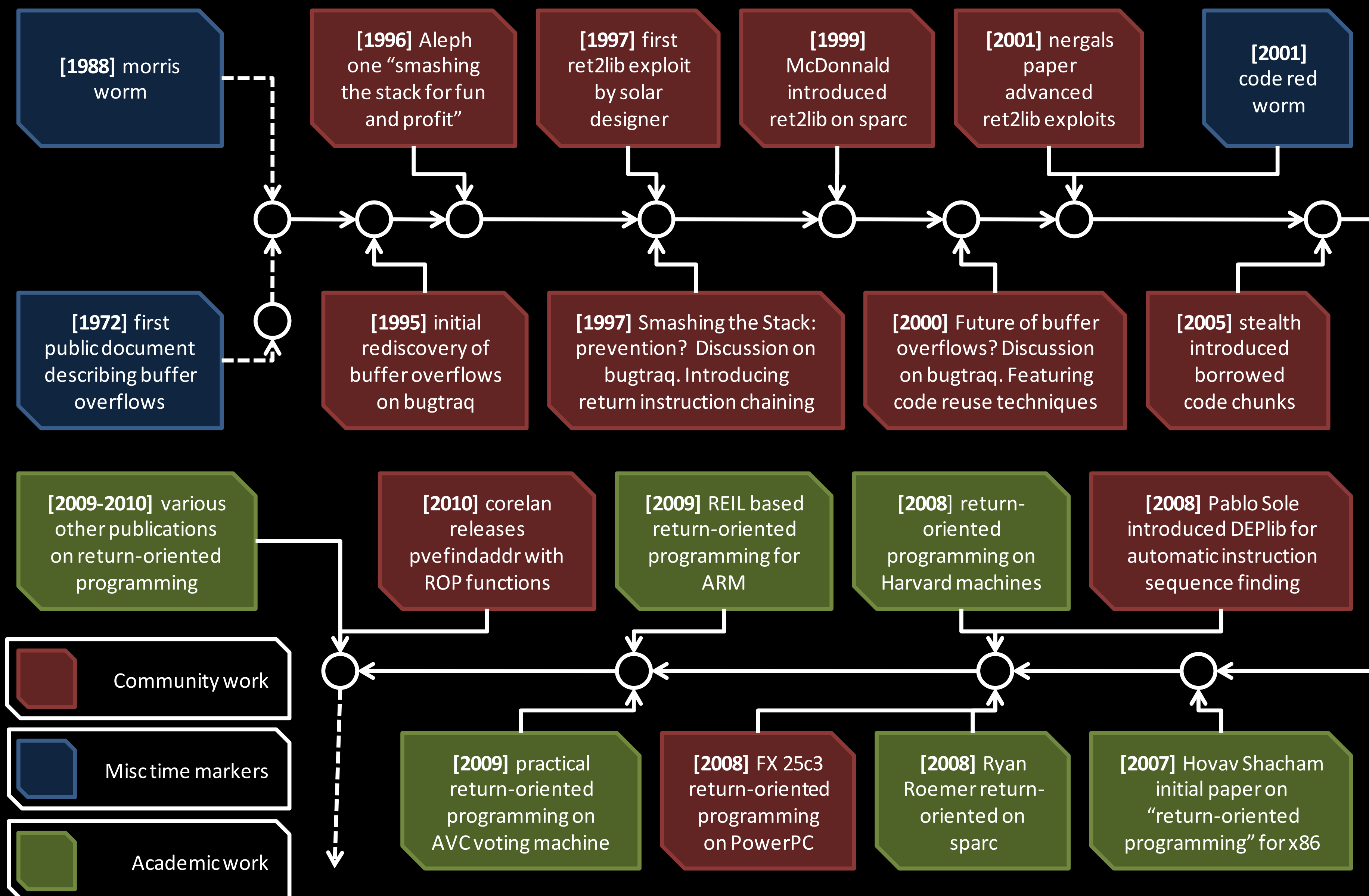
CPU Architecture diversity is increasing.



Motivation III

We want to execute code on machines despite the presence of non-executable memory, but we do not aim for ASLR!

History I


[1972] first public document describing buffer overflows
[1988] morris worm
[1995] initial rediscovery of buffer overflows on bugtraq
[1996] Aleph one “smashing the stack for fun and profit”
[1997] first ret2lib exploit by solar designer
[1997] Smashing the Stack: prevention? Discussion on bugtraq. Introducing return instruction chaining
[1999] McDonnald introduced ret2lib on sparc
[2000] Future of buffer overflows? Discussion on bugtraq. Featuring code reuse techniques
[2001] nergals paper advanced ret2lib exploits
[2001] code red worm
[2005] stealth introduced borrowed code chunks
[2007] Hovav Shacham initial paper on “return-oriented programming” for x86
[2008] Pablo Sole introduced DEPlib for automatic instruction sequence finding
[2008] return-oriented programming on Harvard machines
[2008] Ryan Roemer return-oriented on sparc
[2008] FX 25c3 return-oriented programming on PowerPC
[2009] REIL based return-oriented programming for ARM
[2009] practical return-oriented programming on AVC voting machine
[2010] corelan releases pvefindaddr with ROP functions
[2009-2010] various other publications on return-oriented programming
Community work
Misc time markers
Academic work

History II

```
/* e2.c                                                 *
/* specially crafted to feed your brain by gera */

/* Now, your misson is to make abo1 act like this other program:
 *
        char buf[100];

        while (1) {
                scanf("%100s",buf);
                system(buf);
        }

 * But, you cannot execute code in stack.
 */

int main(int argv,char **argc) {
        char buf[256];

        strcpy(buf,argc[1]);
}
```

Strategy I

- Use only already present code
- No single instruction / return like approach
- Use REIL to be platform independent
- Use “free-branch” instructions rather than ret only
- “Find all first, then filter useful ones” approach
- Keep an eye on side-effects and minimize them

Strategy II

Small RISC instruction set

- 17 instructions for arithmetic, control flow and misc functionality
- Instructions are always side-effect free

Interpreter

- Virtually unlimited memory and temporary registers
- Implemented as a register machine

No support for

- Exceptions, floating point instructions, 64Bit instructions yet

Algorithms

Stage I -> Collect data from the binary

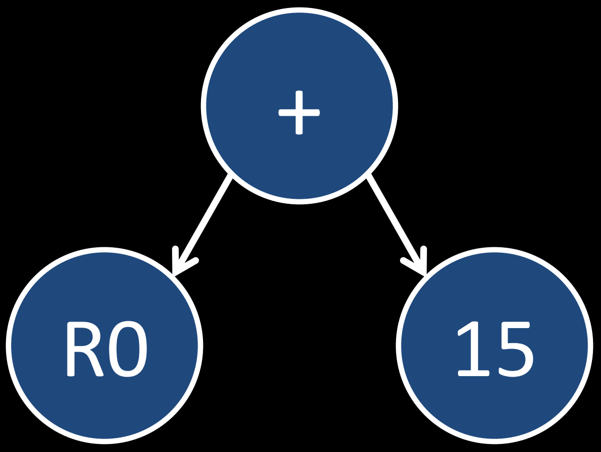
Stage II → Merge the collected data

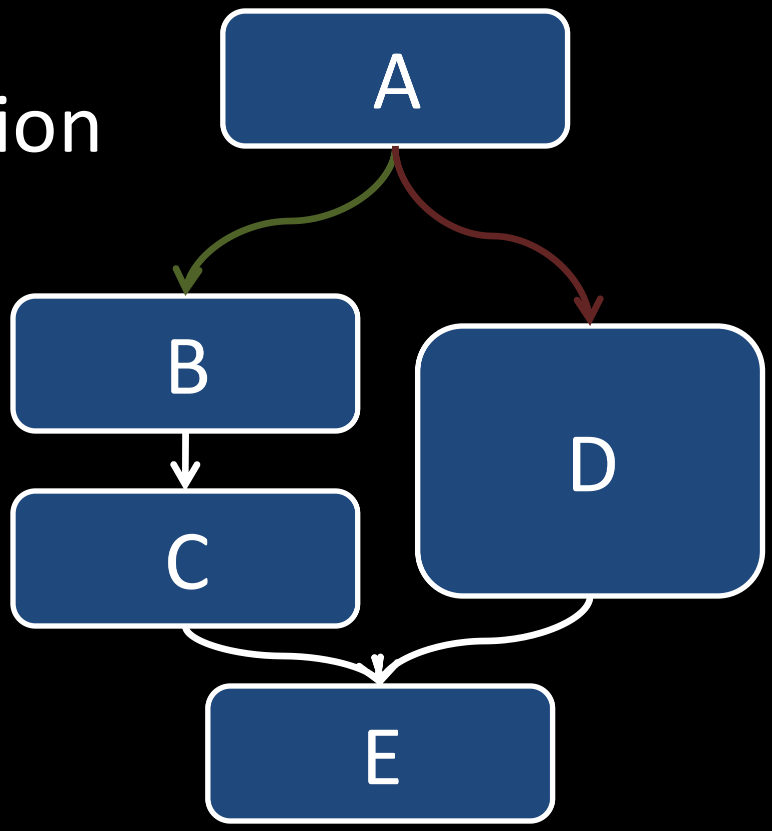
Stage III → Locate useful gadgets in merged data

Algorithms stage I (I)

Goal of the stage I algorithms:

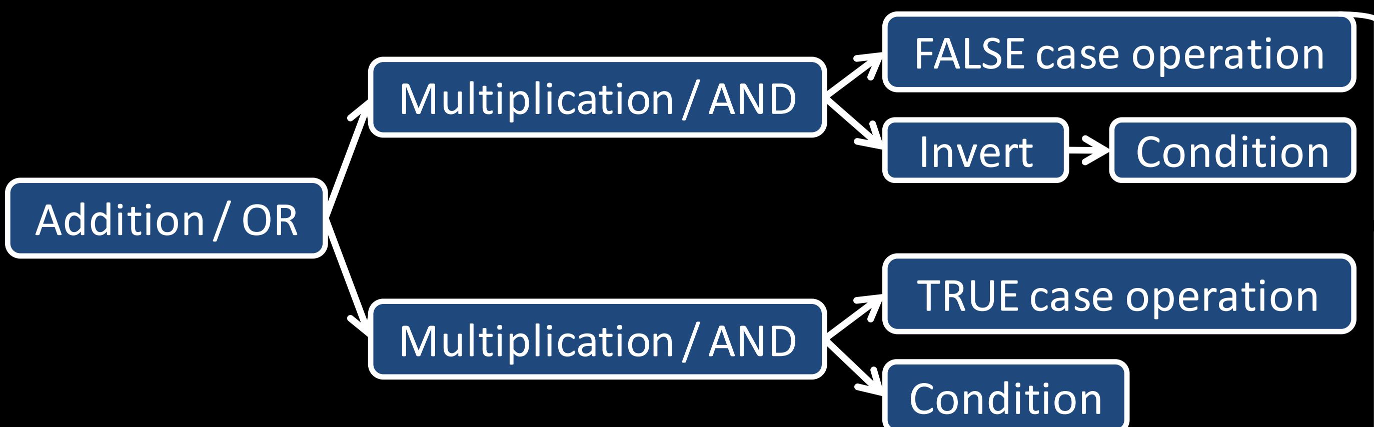
- Collect data from the binary
  1. Extract expression trees from native instructions
  2. Extract path information





Algorithms stage I (II)

1. Expression tree extraction details:
   - Handlers for each possible REIL instruction
     1. Most of the handlers are simple transformations
     2. Memory store and conditional execution need special treatment
2. Path extraction details:
   - Path is extracted in reverse control flow order



We want to have all possible outcomes for a conditional execution in a single expression tree

Algorithms stage II (I)

Goal of the stage II algorithms:

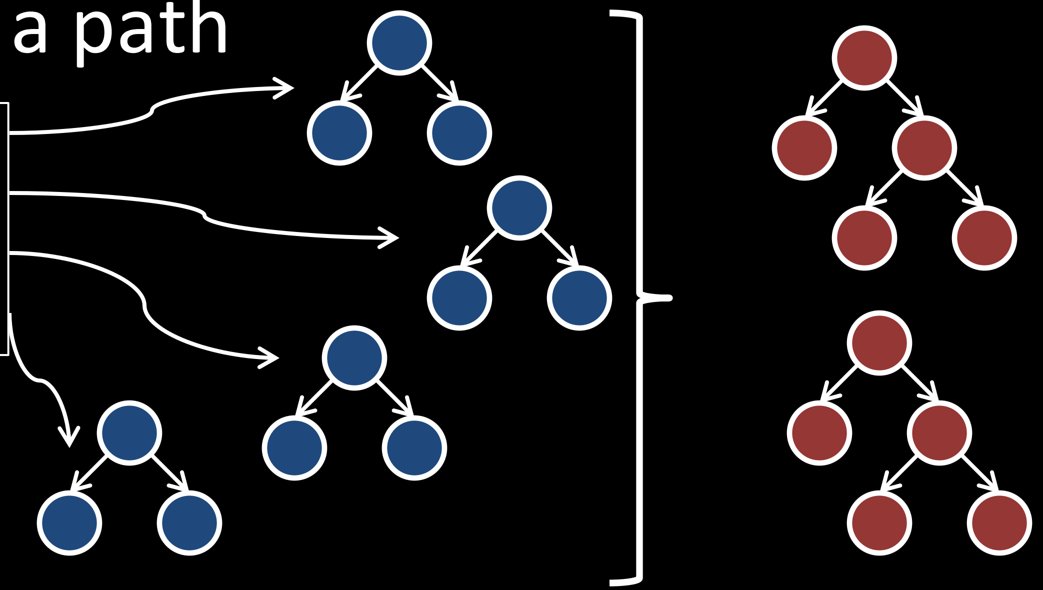
- Merge the collected data from stage I
  1. Combine the expression trees for single native instructions along a path
  2. Determine jump conditions on the path
  3. Simplify the result

Algorithms stage II (II)

Details of the stage II algorithms:

- Combine the expression trees for single native instructions along a path

1. 0x00000001 ADD R0, R1, R2
2. 0x00000002 STR R0, R4
3. 0x00000003 LDMFD SP! {R4,LR}
4. 0x00000004 BX LR



Algorithms stage II (III)

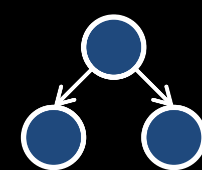
Details of the stage II algorithms:

- Determine jump conditions on the path:

1. 0x00000001 SOME INSTRUCTION
2. 0x00000002 BEQ 0xADDRESS
3. 0x00000003 SOME INSTRUCTION
4. 0x00000004 SOME INSTRUCTION

Z FLAG MUST BE FALSE

Generate condition tree



- Simplify the result:

R0 = ((((((R2+4)+4)+4)+4) OR 0) AND 0xFFFFFFFF)
R0 = R2+16

Algorithms stage III (I)

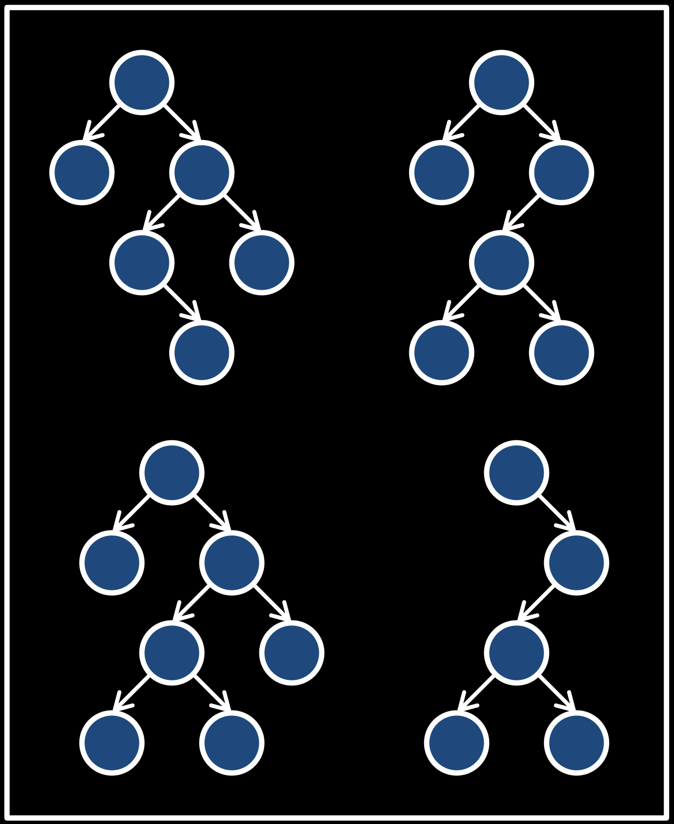
Goal of the stage III algorithms:

- Search for useful gadgets in the merged data
  - Use a tree match handler for each operation.
- Select the simplest gadget for each operation
  - Use a complexity value to determine the gadget which is least complex. (side-effects)

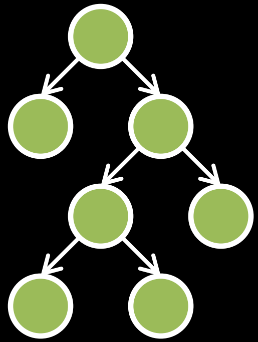
Algorithms stage III (II)

Details of the stage III algorithms:

- Search for useful gadgets in the merged data



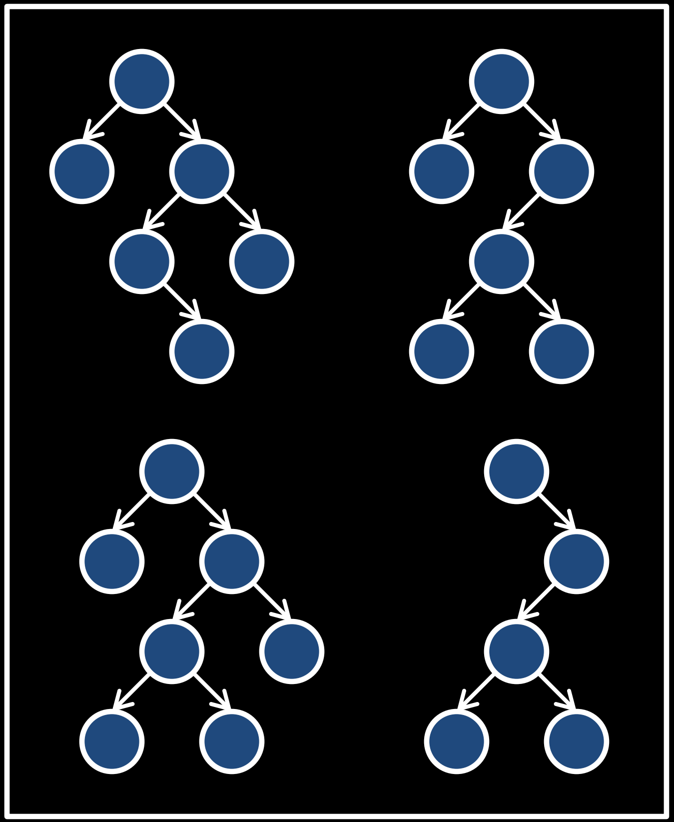
Trees of a gadget candidate are compared to the tree of a specific operation.
Can you spot the match ?

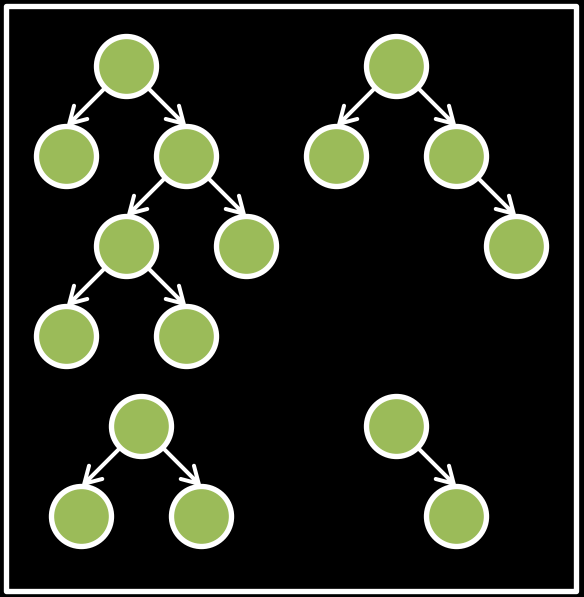


Algorithms stage III (III)

Details of the stage III algorithms:

- Select the simplest gadget for each operation





In most cases there is more than one instruction sequence that provides a specific operation. The overall complexity of all trees is used to determine which gadget is the simplest.

Results

- Algorithms for platform independent return-oriented programming are possible
- We are able to find all necessary gadgets for return-oriented programming using our tool
- Searching for gadgets is not only platform but also very compiler dependent
- Minimizing side-effects is possible if the right approach is chosen

Further work

- Abstract gadget description language
- Automatic gadget compiler for all platforms
- Bring more platforms to REIL
- Better understand the implications of different compilers

Thank you for your time !

Questions ?