IMPROVED KERNEL SECURITY THROUGH MEMORY LAYOUT RANDOMIZATION

IPCCC 2013

Dannie M. Stanley

Graduate Student

Special thanks to my advisors: Professors Dongyan Xu and Eugene Spafford

CERIAS

PURDUE UNIVERSITY

INTRODUCTION

PROBLEM

PURDUE
UNIVERSITY

INTRODUCTION
MOTIVATION

Organizations have incentives to:

- Standardize the software that they use
- Choose market-leading software

INTRODUCTION
HOMOGENEITY

Software homogeneity:

- Majority of internet hosts run one of three kernel families (nt, xnu, linux)
- Attackers craft a single exploit that has the potential to infect millions of hosts

INTRODUCTION
DIVERSIFICATION

Automatic software diversification:

- Attackers must customize attacks for each software variation
- Mass exploitation is much more difficult

INTRODUCTION

INVESTIGATE

- How can we add diversification to an operating system kernel?
- Will diversification prevent malicious kernel attacks (rootkits)?

APPROACH

KERNEL CODE DIVERSIFICATION

PURDUE
UNIVERSITY

APPROACH

COMPILE-TIME

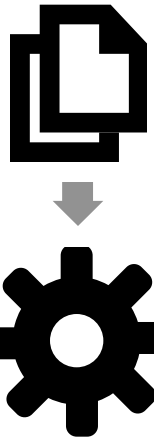
Our approach, compile-time diversification:

- Compiler input
  - *Unmodified* source code
  - Randomization configuration
- Compiler output
  - Unique program for each randomization configuration

APPROACH

COMPILE-TIME DIVERSIFICATION

Source Files



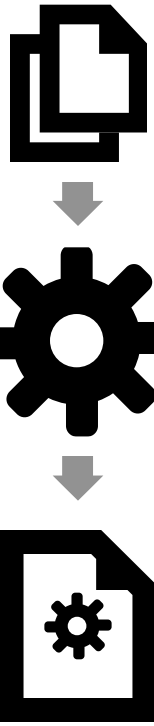
Compiler

Program

APPROACH

COMPILE-TIME DIVERSIFICATION

Source Files



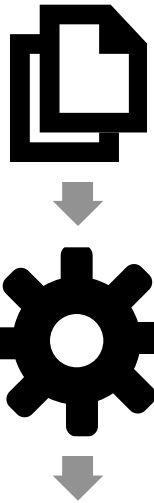
Compiler

Program

APPROACH

COMPILE-TIME DIVERSIFICATION

Source Files



Compiler

Program

APPROACH

RANDOMIZE

Randomize what?

- Record field order
- Subroutine argument order

APPROACH
RECORD FIELD ORDER RANDOMIZATION

PURDUE
UNIVERSITY

APPROACH

RECORD FIELD-ORDER RANDOMIZATION

Record Field Order Randomization (RFOR)

- Compiler input:
  - Source code
  - Randomization configuration
- Compiler output:
  - Randomized program

APPROACH
RANDOMIZATION

Randomization configuration:

- Randomization seed
  - Allows for compilation of compatible programs
- Add dummy fields to increase permutations

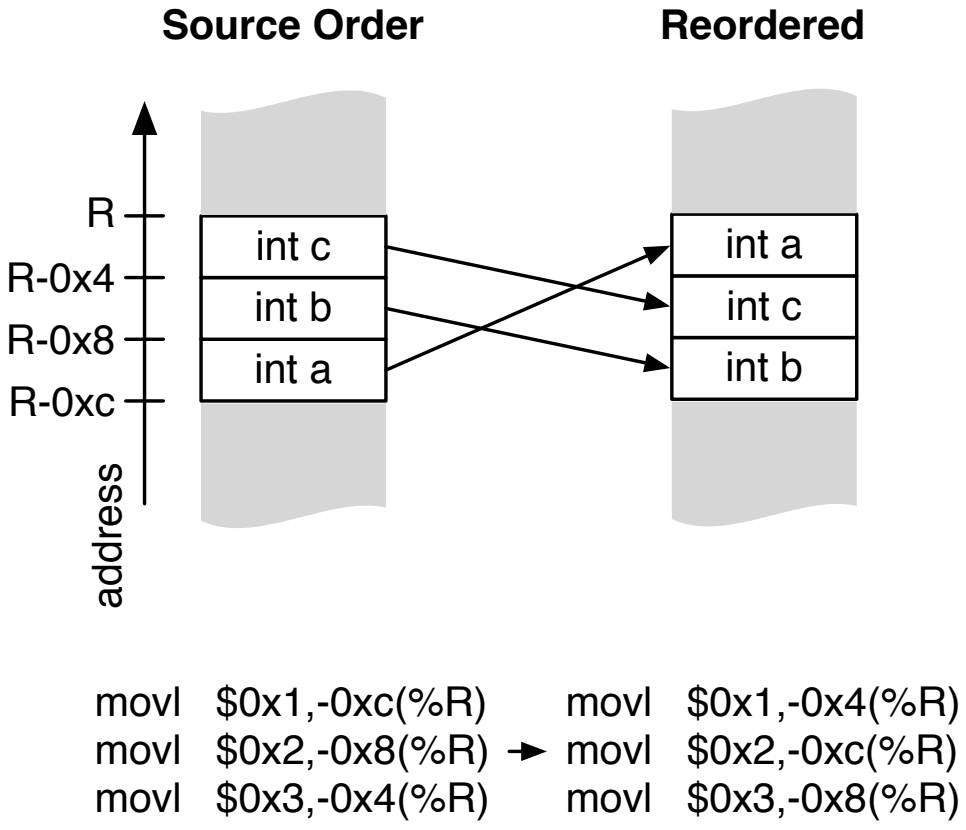
Security benefits:

- Attackers will compile code using unrandomized record definitions
- Attack code will not run as intended

APPROACH

RECORD FIELD-ORDER RANDOMIZATION

```
struct foo {
   int a;
   int b;
   int c;
} bar = {
   a=1,
   b=2,
   c=3
};
```

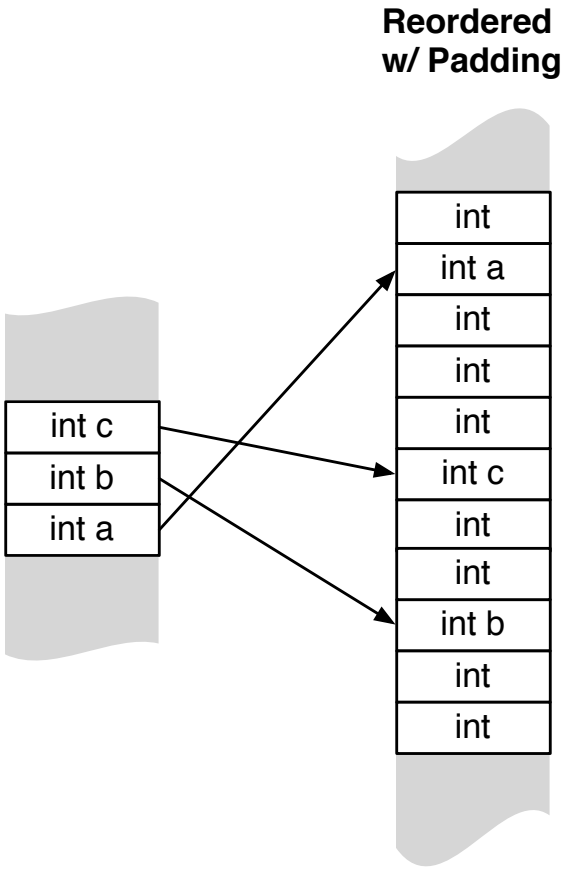



```
movl   $0x1,-0xc(%R)        movl   $0x1,-0x4(%R)
movl   $0x2,-0x8(%R)   →    movl   $0x2,-0xc(%R)
movl   $0x3,-0x4(%R)        movl   $0x3,-0x8(%R)
```

APPROACH

RECORD FIELD-ORDER RANDOMIZATION

```
struct foo {
   int a;
   int b;
   int c;
} bar = {
   a=1,
   b=2,
   c=3
};
```



APPROACH

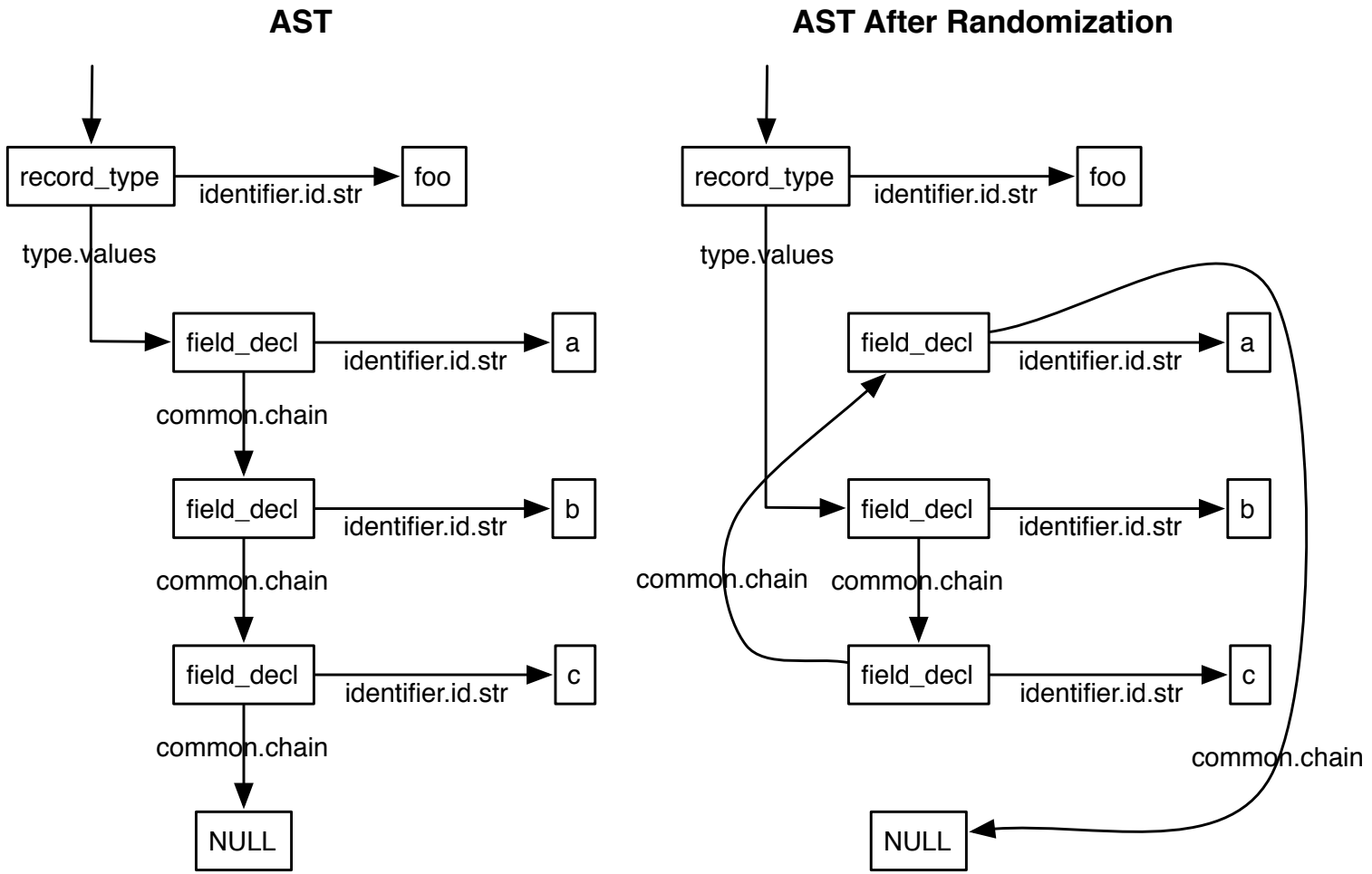
AST RANDOMIZATION

Compile-time randomization occurs:

- After parsing
- Before optimization passes
- Shuffling occurs in the abstract syntax tree (AST)

APPROACH

AST



APPROACH
RFOR SUITABILITY ANALYSIS

PURDUE
UNIVERSITY

APPROACH

PROBLEM WITH RFOR

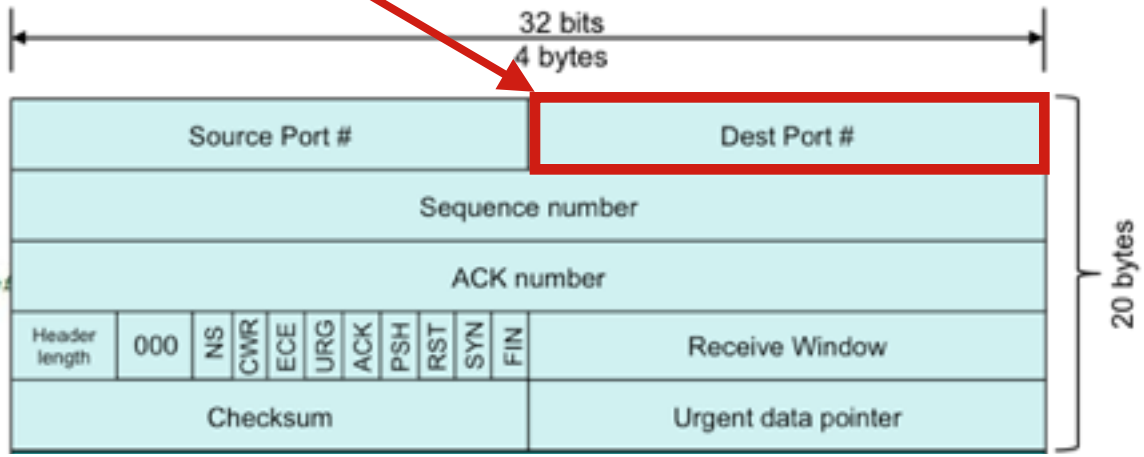
Problem: not all records are suitable for randomization

- Ex. struct that represents TCP headers

APPROACH

SUITABILITY

```
struct tcphdr {
        __be16  source;
        __be16  dest;
        __be32  seq;
        __be32  ack_seq;
#if defined(__LITTLE_ENDIAN_BITFIELD)
        __u16   res1:4,
                doff:4,
                fin:1,
                syn:1,
                rst:1,
                psh:1,
                ack:1,
                urg:1,
                ece:1,
                cwr:1;
#elif defined(__BIG_ENDIAN_BITFIELD)
        __u16   doff:4,
                res1:4,
                cwr:1,
                ece:1,
                urg:1,
                ack:1,
                psh:1,
                rst:1,
                syn:1,
                fin:1;
#else
#error  "Adjust your <asm/byteorder.h> def
#endif
        __be16  window;
        __sum16 check;
        __be16  urg_ptr;
};
```



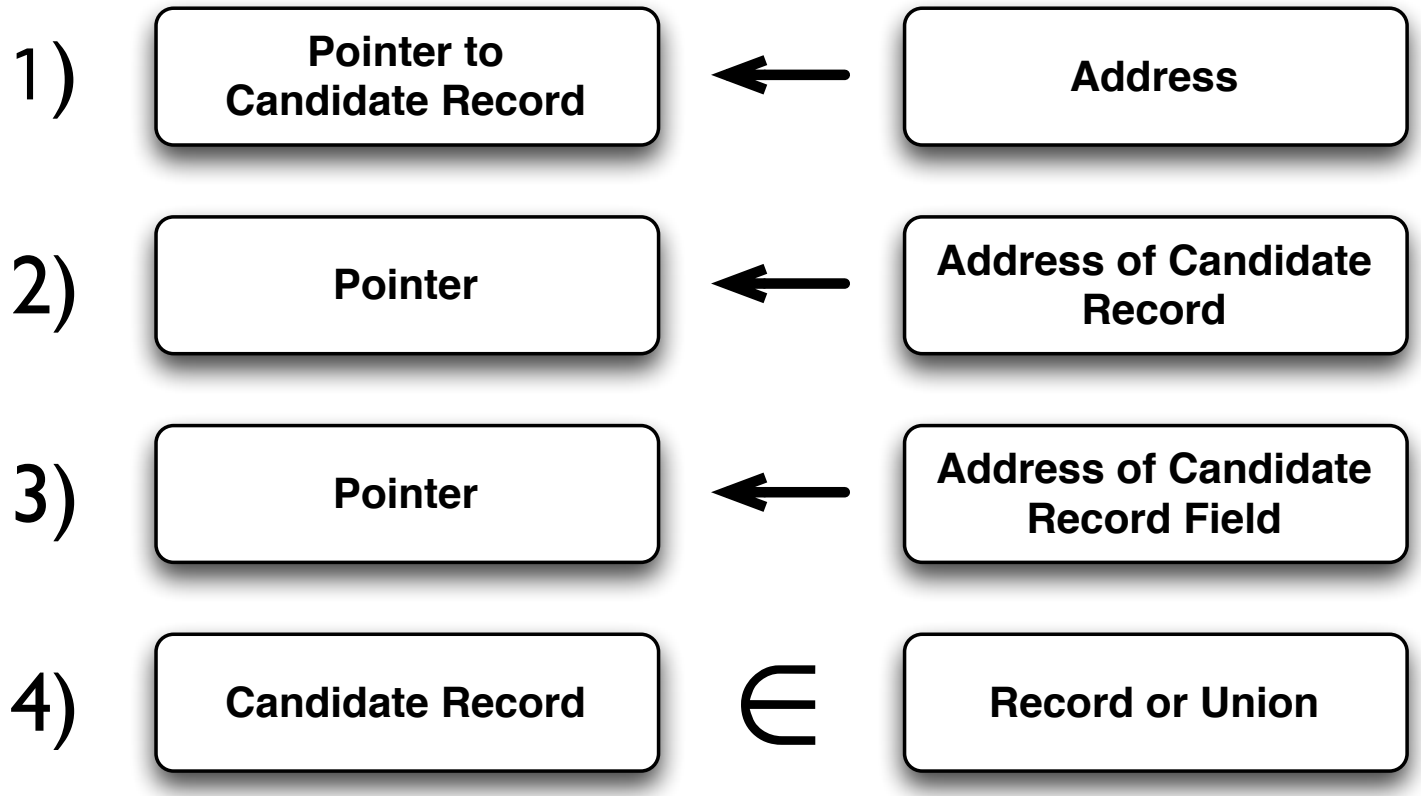
APPROACH
STATIC ANALYSIS

RFOR suitability static analysis technique

- Input:
  - Candidate struct
  - Source code
- Output:
  - Is the candidate fit for RFOR?

APPROACH

FOUR CONDITIONS



APPROACH

SUITABILITY REPORT

Report if potentially unsafe operations occur

- File name
- Line number

APPROACH

SUBROUTINE ARGUMENT ORDER RANDOMIZATION

PURDUE
UNIVERSITY

Subroutine Argument Order Randomization (SAOR)

- Compiler input:
  - Source code
  - Randomization configuration (padding, seed)
- Compiler output:
  - Randomized program

APPROACH
AST RANDOMIZATION

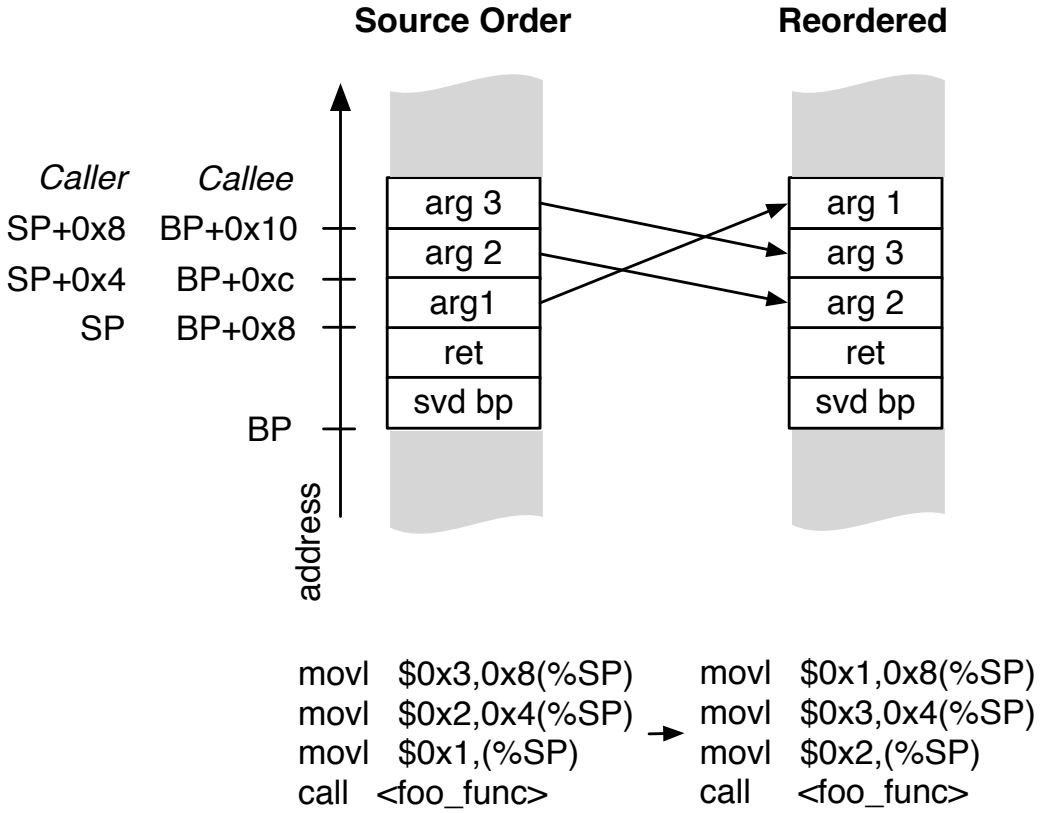
Similar to RFOR, SAOR compile-time randomization occurs:

- After parsing
- Before optimization passes
- Shuffling occurs in the abstract syntax tree (AST)

APPROACH

SAOR

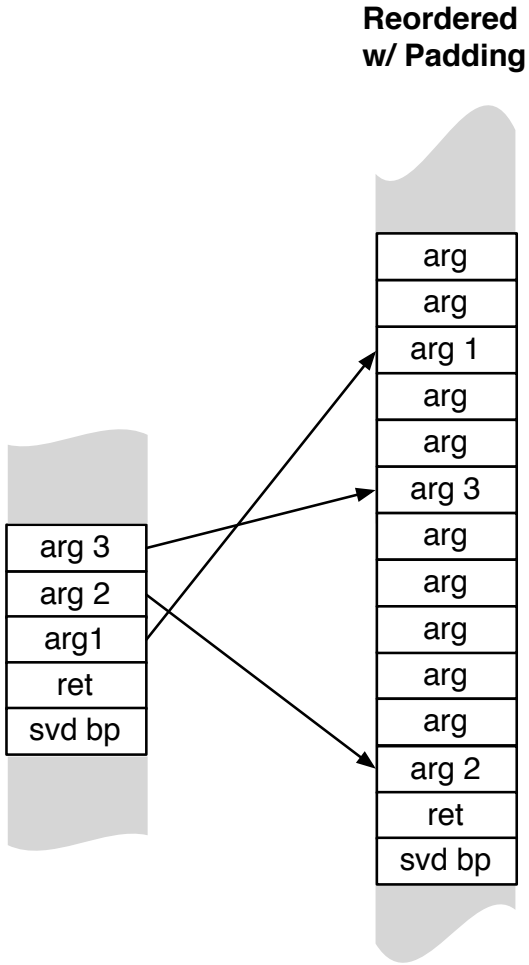
foo_func(1,2,3);



APPROACH

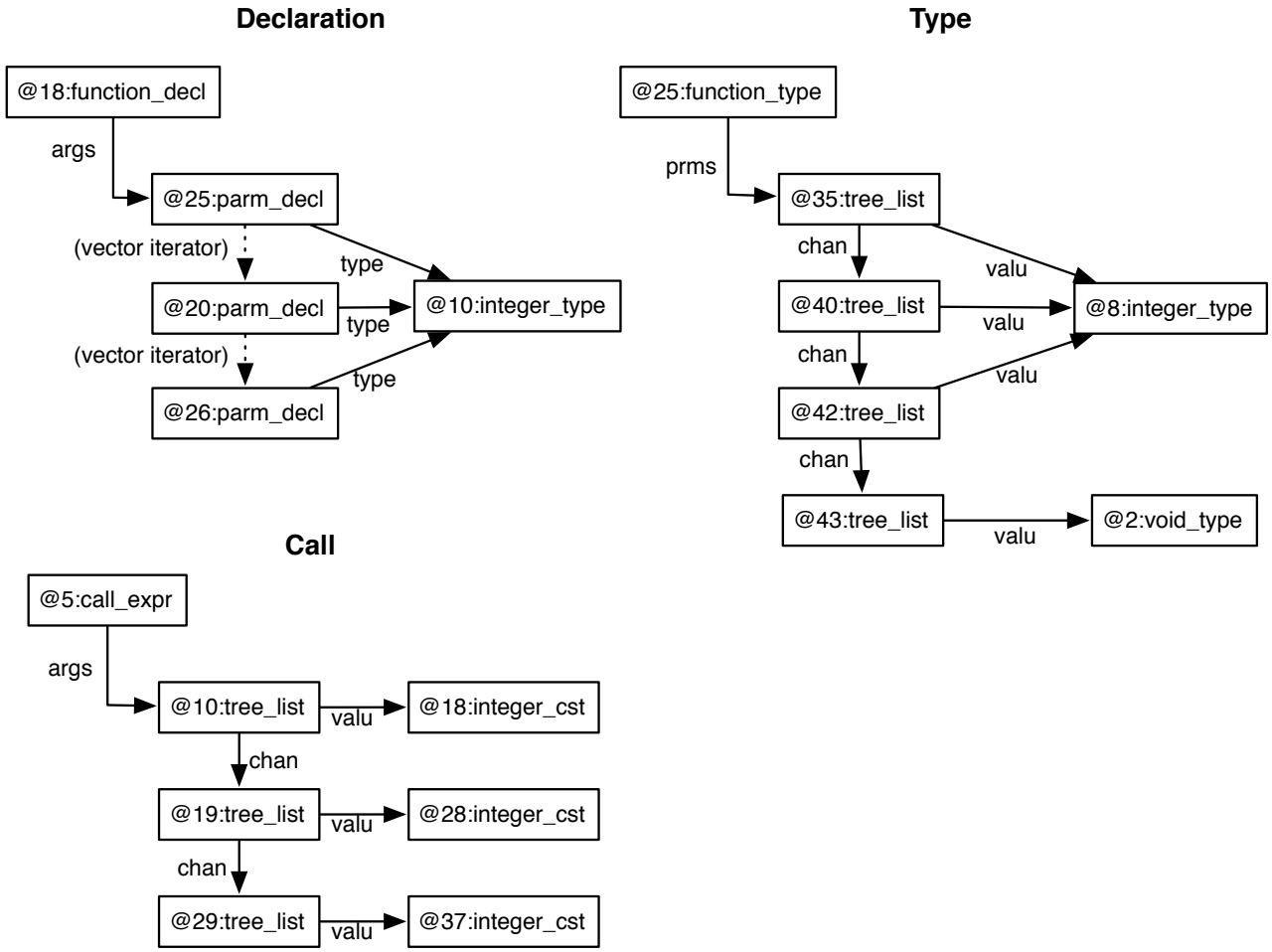
SAOR

foo_func(1,2,3);



APPROACH

SAOR IN AST



EVALUATION

DOES IT WORK

PURDUE
UNIVERSITY

Implementation:

- GCC Plugin Architecture
  - RFOR Plugin
  - RFOR Fitness Plugin
  - SAOR Plugin

EVALUATION
EXPERIMENTS

Evaluation

- Compiled multiple versions of Linux kernel with GCC randomization plugins
- Subjected randomized kernels to four rootkits
  - All tested rootkits were prevented by RFOR and/or SAOR by randomizing only a few objects

EVALUATION

ROOTKITS

Randomized Linux Kernel (RFOR & SAOR):

- task_struct
- module
- file
- proc_dir_entry
- inode_operations
- pid_task()

Neutralized rootkits:

- Adore-NG
- HP
- hidefile
- hideproc

EVALUATION
HEURISTICS

RFOR fitness plugin evaluation

- Conservative static analysis approach
  - False-positives
- Refined analysis using heuristics

EVALUATION
WHITELIST

Heuristic-based whitelisting:

- Generics
  - ex. cast to list_head pointer
- Memory allocation functions
  - ex. cast from malloc
- Address of field conversion (optional)
  - Passing field by reference (OK)
  - Using field address to calculate sibling address (NOT OK)

EVALUATION

FALSE-POSITIVES

RFOR suitability analysis:

- Linux kernel 2.6.38.8
- task_struct

Test	W/O Whitelists	W/ Whitelists
Assignment To task_struct *	52	1
Assignment From task_struct *	46	2
Address Taken of task_struct Field	214	0
	312	3

EVALUATION

PERFORMANCE

Options	Avg Time
w/o Plug-Ins	86.730s
w/ RFOR Plugin	87.9087s
w/ Fitness Plugin	87.2170s
w/ SAOR Plugin	87.8303s

EVALUATION
SUMMARY

Summary

- Introduced argument order layout randomization technique
- Refined record field-order randomization
  - Introduce static analysis technique for determining the suitability of a record for field-order randomization
- Demonstrate effectiveness of techniques against common rootkits

THANK YOU

ANY QUESTIONS?

PURDUE
UNIVERSITY