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
INTRODUCTION

PROBLEM
Organizations have incentives to:

- Standardize the software that they use
- Choose market-leading software
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
Automatic software diversification:

- Attackers must customize attacks for each software variation
- Mass exploitation is much more difficult
How can we add diversification to an operating system kernel?

Will diversification prevent malicious kernel attacks (rootkits)?
APPROACH
KERNEL CODE DIVERSIFICATION
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
COMPILATION

Source Files

Compiler

Program
APPROACH

COMPIL-E-TIME DIVERSIFICATION

Source Files

Compiler

Program

...
Randomize what?

- Record field order
- Subroutine argument order
APPROACH

RECORD FIELD ORDER RANDOMIZATION
**Record Field Order Randomization (RFOR)**

- **Compiler input:**
  - Source code
  - Randomization configuration

- **Compiler output:**
  - Randomized program
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
struct foo {
    int a;
    int b;
    int c;
} bar = {
    a=1,
    b=2,
    c=3
};

Source Order

Reordered

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

movl $0x1,-0x4(%R)
movl $0x2,-0xc(%R)
movl $0x3,-0x8(%R)
struct foo {
    int a;
    int b;
    int c;
} bar = {
    a=1,
    b=2,
    c=3
};

Reordered w/ Padding

```
int c
int b
int a
```
Compile-time randomization occurs:

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

AST After Randomization

record_type
  | identifier.id.str
  | foo
  | type.values
  | field_decl
  | identifier.id.str
  | a
  | common.chain
  | field_decl
  | identifier.id.str
  | b
  | common.chain
  | field_decl
  | identifier.id.str
  | c
  | common.chain
  | NULL

record_type
  | identifier.id.str
  | foo
  | type.values
  | field_decl
  | identifier.id.str
  | a
  | common.chain
  | field_decl
  | identifier.id.str
  | b
  | common.chain
  | field_decl
  | identifier.id.str
  | c
  | common.chain
  | NULL
APPROACH
FOR SUITABILITY ANALYSIS
Problem: not all records are suitable for randomization

- Ex. struct that represents TCP headers
### Approach Suitability

```c
struct tcp_hdr {
    __be16 source;
    __be16 dest;
    __be24 sport;
    __be24 dport;
    __be16 seq;
    __be12 ack_seq;
    #if defined(__LITTLE_ENDIAN_BITFIELD)
      __ul16 real:4,
      __ul16 doff:4,
      __be16 fin:1,
      __be16 syn:1,
      __be16 rst:1,
      __be16 psh:1,
      __be16 ack:1,
      __be16 urg:1,
      __be16 cwr:1,
      __be16 urg_ptr;
    #else
      __ul16 doff:4,
      __ul16 real:4,
      __be16 cwr:1,
      __be16 urg:1,
      __be16 urg_ptr;
    #endif
    __be16 window;
    __sum16 check;
    __be16 urg_ptr;
};
```

The diagram illustrates the layout of a TCP header, showing fields such as Source Port, Dest Port, Sequence number, ACK number, and Receive Window.
RFOR suitability static analysis technique

Input:
- Candidate struct
- Source code

Output:
- Is the candidate fit for RFOR?
FOUR CONDITIONS

1) Pointer to Candidate Record \rightarrow Address

2) Pointer \rightarrow Address of Candidate Record

3) Pointer \rightarrow Address of Candidate Record Field

4) Candidate Record \in \text{Record or Union}
Report if potentially unsafe operations occur

- File name
- Line number
APPROACH
SUBROUTINE ARGUMENT ORDER RANDOMIZATION
Subroutine Argument Order Randomization (SAOR)

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

- Compiler output:
  - Randomized program
Similar to RFOR, SAOR compile-time randomization occurs:

- After parsing
- Before optimization passes
- Shuffling occurs in the abstract syntax tree (AST)
foo_func(1,2,3);
foo_func(1,2,3);
APPROACH
SAOR IN AST

Declaration

Call

Type
EVALUATION
DOES IT WORK
Implementation:

- GCC Plugin Architecture
- RFOR Plugin
- RFOR Fitness Plugin
- SAOR Plugin
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
Randomized Linux Kernel (RFOR & SAOR):

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

Neutralized rootkits:

- Adore-NG
- HP
- hidefile
- hideproc
RFOR fitness plugin evaluation

- Conservative static analysis approach
- False-positives
- Refined analysis using heuristics
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)
**RFOR suitability analysis:**
- Linux kernel 2.6.38.8
- task struct

<table>
<thead>
<tr>
<th>Test</th>
<th>W/O Whitelists</th>
<th>W/ Whitelists</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignment To task struct *</td>
<td>52</td>
<td>1</td>
</tr>
<tr>
<td>Assignment From task struct *</td>
<td>46</td>
<td>2</td>
</tr>
<tr>
<td>Address Taken of task struct Field</td>
<td>214</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>312</strong></td>
<td><strong>3</strong></td>
</tr>
<tr>
<td>Options</td>
<td>Avg Time</td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------</td>
<td></td>
</tr>
<tr>
<td>w/o Plug-Ins</td>
<td>86.730s</td>
<td></td>
</tr>
<tr>
<td>w/ RFOR Plugin</td>
<td>87.9087s</td>
<td></td>
</tr>
<tr>
<td>w/ Fitness Plugin</td>
<td>87.2170s</td>
<td></td>
</tr>
<tr>
<td>w/ SAOR Plugin</td>
<td>87.8303s</td>
<td></td>
</tr>
</tbody>
</table>
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?