HIPStR: Heterogeneous-ISA Program State Relocation

Ashish Venkat  Sriskanda Shamasunder  Hovav Shacham  Dean M. Tullsen

University of California, San Diego
On-chip Heterogeneity

Qualcomm Snapdragon 810

"Big" Cores

"Little" Cores

GPU

Cryptographic Acceleration

Image Signal Processing

Digital Signal Processing

Multimedia Processing

Offers varying degrees of micro-architectural complexity and specialization.
On-chip Heterogeneity

Exploit both architectural (ISA) and micro-architectural heterogeneity to realize
Heterogeneous-ISA Chip Multiprocessors
Heterogeneous-ISA Chip Multiprocessors

- Prior research suggests **21% performance gains** and **23% energy savings** over a single-ISA heterogeneous CMP targeted at general purpose computing.
- **Instantaneous migration at <0.7% performance overhead** allows different code regions to execute on the ISA of preference, and thereby maximize performance.

*Harnessing ISA Diversity: Design of a Heterogeneous-ISA Chip Multiprocessor, Ashish Venkat and Dean M. Tullsen, ISCA’14.*
This talk will showcase the immense security potential of this architecture, in particular, to thwart Return-Oriented Programming.
Buffer Overflow Exploits – Code Injection

Inject malicious code on stack/heap and subvert control flow

Malicious Code

xor %eax, %eax
mov $0x1, %al
xor %ebx, %ebx
int $0x80

Application Code

lea -0x78(%ebp),%eax
mov %eax,0x8(%esp)
mov -0x80(%ebp),%eax
mov %eax,0x4(%ebp)
movl $0x3,%esp
jmp *%eax

Bad Behavior
Buffer Overflow Exploits – Code Reuse

Malicious Code

- xor %eax, %eax
- mov $0x1, %al
- xor %ebx, %ebx
- int $0x80

Application Code

- lea -0x78(%ebp),%eax
- mov %eax,0x8(%esp)
- mov -0x80(%ebp),%eax
- mov %eax,0x4(%ebp)
- movl $0x3,(%esp)
- jmp *%eax

Bad Behavior

Return-Oriented Programming
HIPStR: Heterogeneous-ISA Program State Relocation

Synergistically combines two strong and independent defense techniques:

• Binary Translation driven Program State Relocation
• Non-deterministic Execution Migration across Heterogeneous-ISAs
Outline

• Motivation
• Return-Oriented Programming
  • HIPStR: Heterogeneous-ISA Program State Relocation
    • Program State Relocation
    • Heterogeneous-ISA Migration
• Evaluation
  • Brute Force attacks
  • JIT-ROP attacks
  • Tailored Anti-diversification attacks
• Key Points
Return-Oriented Programming

Read only Text Section

```
lea  -0x78(%ebp),%eax
mov  %eax,0x8(%esp)
call d92e0 <memcpy>
....
ret
....
mov  %edx,-0x94(%ebp)
movl  $0x3,%esp
mov  %eax,0x4(%esp)
ret
....
....
xor  %eax,%eax
ret
....
....
pop  %ebx
ret
```

Stack

<table>
<thead>
<tr>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x20d1b0</td>
</tr>
<tr>
<td>0x17049d</td>
</tr>
<tr>
<td>0x10ad</td>
</tr>
<tr>
<td>0x80436a</td>
</tr>
<tr>
<td>0xbfff8a40</td>
</tr>
<tr>
<td>0x74636570</td>
</tr>
</tbody>
</table>

Caller Frame ends here
Return-Oriented Programming

Read only Text Section

lea -0x78(%ebp),%eax
mov %eax,0x8(%esp)
call d92e0 <memcpy>
....
ret
....
mov %edx,-0x94(%ebp)
movl $0x3,%esp
mov %eax,0x4(%esp)
ret
....
....
xor %eax,%eax
ret
....
....
pop %ebx
ret

Exploit buffer overflow

Stack

0x870f65
0x87098d
0xbfff8076
0x870234
0x432a123
0x65708ad6

Caller Frame ends here
Return-Oriented Programming

Read only Text Section

le a -0x78(%ebp),%eax
mov %eax,0x8(%esp)
call d92e0 <memcpy>
....
ret
....
mov %edx,-0x94(%ebp)
movl $0x3,(%esp)
mov %eax,0x4(%esp)
ret
....
....
xor %eax,%eax
ret
....
....
pop %ebx
ret

Return To Gadget 1

Stack

0x870f65
0x87098d
0xbfff8076
0x870234
0x432a123
0x65708ad6

Dynamic Execution Stream

pop %ebx
Return-Oriented Programming

Read only Text Section

```
leal -0x78(%ebp),%eax
mov %eax,0x8(%esp)
call d92e0 <memcpy>
....
ret
....
mov %edx,-0x94(%ebp)
movl $0x3,(%esp)
mov %eax,0x4(%esp)
ret
....
....
xor %eax,%eax
ret
....
....
popl %ebx
ret
```

Return To Gadget 2

Stack

```
0x870f65
0x87098d
0xbfff8076
0x870234
0x432a123
0x6570ad6
```

Dynamic Execution Stream

```
popl %ebx
xor %eax, %eax
```

Stack
Return-Oriented Programming

Read only Text Section

```
lea  -0x78(%ebp),%eax
mov  %eax,0x8(%esp)
call d92e0 <memcpy>
....
ret
....
mov  %edx,-0x94(%ebp)
movl  $0x3,(%esp)
mov  %eax,0x4(%esp)
ret
....
....
xor  %eax,%eax
ret
....
pop  %ebx
ret
```

Return To Gadget 3

Stack

```
0x870f65
0x87098d
0xbfff8076
0x870234
0x432a123
0x65708ad6
```

Dynamic Execution Stream

```
pop  %ebx
xor  %eax, %eax
mov  %edx,-0x94(%ebp)
movl  $0x3,(%esp)
mov  %eax,0x4(%esp)
```
Escape from ROP

ROP thrives on 2 fundamental characteristics:

• Ability to hijack control flow
• Prior knowledge of gadget locations
Escape from ROP

ROP thrives on 2 fundamental characteristics:

• Ability to hijack control flow
  - Control Flow Integrity (CFI) Techniques take advantage of this.
  - Classic CFI: Constrain control flow to a pre-defined CFG (hard to accomplish without run-time knowledge).
  - Modern CFI: CCFIR, bin-CFI, Branch Regulation, Code Pointer Integrity.

• Prior knowledge of gadget locations

Several backdoors exist that can completely bypass modern CFI

- Missing the Point(er) (Oakland’15)
- Out of Control (Oakland’14)
- Control Flow Bending (USENIX Security’15)
Escape from ROP

ROP thrives on 2 fundamental characteristics:

• Ability to hijack control flow
• Prior knowledge of gadget locations
  - Code Randomization Techniques take advantage of this.
  - Gadget Location Randomization and Obfuscation proposed at module, function, basic block, and instruction levels.
  - Not “fool-proof”. They just reduce the probability of a successful mount.

How easy is it to mount an attack with state-of-the-art randomization?
- Hacking Blind – Brute Force attack possible in under 20 minutes.
- Information Leakage - Just-In-Time ROP possible in 23 seconds.

Need more randomness and more resilience to information leakage
Escape from ROP

ROP thrives on 2 fundamental characteristics:
• Ability to hijack control flow
• Prior knowledge of gadget locations
Escape from ROP

ROP thrives on 4 fundamental characteristics:
• Ability to hijack control flow
• Prior knowledge of gadget locations
• Requires program state (registers/memory) to perform computation
• Knowledge of the underlying ISA

More Randomness

Low Performance Overhead

More Resilience to Information Leakage

Massive Attack Surface Reduction

HIPStR: Heterogeneous-ISA Program State Relocation
Outline

• Motivation

• Return-Oriented Programming

• HIPStR: Heterogeneous-ISA Program State Relocation
  • Program State Relocation
  • Heterogeneous-ISA Migration

• Evaluation
  • Brute Force attacks
  • JIT-ROP attacks
  • Tailored Anti-diversification attacks

• Key Points
Escape from ROP

ROP thrives on 4 fundamental characteristics:

• Ability to hijack control flow
• Prior knowledge of gadget locations
• Requires program state (registers/memory) to perform computation
• Knowledge of the underlying ISA

More Randomness
Low Performance Overhead

More Resilience to Information Leakage
Massive Attack Surface Reduction

HIPStR: Heterogeneous-ISA Program State Relocation
Program State Relocation Architecture

Dynamic Binary Translator

Randomizer

Translation Engine

Code Cache

or %al, 0x80c (%esp)
add $c54, %esp
ret

Processor

Program Binary

Code Section

BB#1:
  mov 0x30(%esp), %esi
  ....
  ret

BB#2:
  or %dl, %bl
  ret

BB#3:
  add %eax, 0x48
  call *(%eax)
Program State Relocation Architecture

All execution happens within the code cache, but it is critical to not leak code cache addresses to the attacker.
Program State Relocation Architecture

Program Binary

Code Section

BB#1:
- mov 0x30(%esp), %esi
- ...
- ret

BB#2:
- or %dl, %bl
- ret

BB#3:
- add %eax, 0x48
- call *(%eax)

Dynamic Binary Translator

Randomizer

Translation Engine

Code Cache

or %al, 0x80c(%esp)
add $c54, %esp
ret

Code Cache Miss Handler

Processor

H/W Return Address Table

<table>
<thead>
<tr>
<th>Source Address</th>
<th>Target Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1001beef</td>
<td>0x08048abc</td>
</tr>
</tbody>
</table>
Program State Relocation Example

ROP gadget before PSR

or %dl,%bl
ret

ROP gadget after PSR

or %al,0x80c(%esp)
add $c54, %esp
ret

Function-level Relocation Map

Registers:
- ebx -> [esp+0x80c]
- edx -> eax
- esi -> [esp+0x1800]
- ebp -> PSR Temporary

Stack Objects:
- [esp+0x30] -> [esp + 0x14a8]
- %RET -> [esp + 0xc58]
Program State Relocation Example

### ROP gadget before PSR

<table>
<thead>
<tr>
<th>or</th>
<th>%dl,%bl</th>
</tr>
</thead>
<tbody>
<tr>
<td>ret</td>
<td></td>
</tr>
</tbody>
</table>

### ROP gadget after PSR

<table>
<thead>
<tr>
<th>or</th>
<th>%al,0x80c(%esp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>$c54, %esp</td>
</tr>
<tr>
<td>ret</td>
<td></td>
</tr>
</tbody>
</table>

### Function-level Relocation Map

**Registers:**
- ebx -> [esp+0x80c]
- edx -> eax
- esi -> [esp+0x1800]
- ebp -> PSR Temporary

**Stack Objects:**
- [esp+0x30] -> [esp + 0x14a8]
- %RET -> [esp + 0xc58]
How much randomness does PSR provide?

Each instruction operand can relocate to $2^{13} - 2^{16}$ random stack objects.

**X86 stack frame**
- registers spill
- locals and temporaries (high to low)
- argument n
  - ...
- argument 0

**X86-PSR stack frame**
- Randomization Space
- arguments
- register spill
- Randomization Space
- register spill
- arguments
- locals
- Randomization space
- aggregates
- Frame Entropy

2-16 pages of randomness per frame
Program State Relocation Performance

- Overall performance degradation vs native unsecure execution = 13%
- Speedup over competition = 16%
Program State Relocation

Entropy

- Entropy provided by PSR supersedes state-of-the-art defenses (64-bits)
- Entropy provided by PSR can be orthogonally applied on other defenses
Is PSR capable of detecting an attack?

Yes, code cache misses triggered by indirect control transfers could potentially mean a security breach.

• Compulsory miss: An indirect jump/return to a basic block that was never translated by PSR.

• Conflict miss: An indirect jump/return to a basic block that was previously evicted.

• ROP attack: An indirect jump/return that can hijack control-flow.
HIPStR: Heterogeneous-ISA Program State Relocation

Synergistically combines two strong and independent defense techniques:

- **Binary Translation driven Program State Relocation**
- **Non-deterministic Execution Migration across Heterogeneous-ISAs**
Escape from ROP

ROP thrives on 4 fundamental characteristics:

- Ability to hijack control flow
- Prior knowledge of gadget locations
- Requires program state (registers/memory) to perform computation
- Knowledge of the underlying ISA

More Randomness
Low Performance Overhead

More Resilience to Information Leakage
Massive Attack Surface Reduction

HIPStR: Heterogeneous-ISA Program State Relocation
Execution Migration in a Heterogeneous-ISA CMP

Migration Runtime

ISA-I relocation map

ISA-II relocation map

Stack transformation on core -II

Stack

Data Sections
ISA agnostic

Code Section (ISA – I)

Code Section (ISA – II)

PC

Heterogeneous-ISA Architecture

ISA - I

ISA - II

ISA - III

L1

L2

L1

L2

L1

L2

PC

PC

Native execution on core - I

Native execution on core - II

Migration requested !!

Migration to core II

Native execution on core - II

Native execution on core - II

transformable basic block
(known at compile-time)
When is the right time to migrate?

• Performance-induced Migrations:
  • Migrate execution when a program phase-change alters the ISA of preference.
  • Provides as much as 9% additional speedup, sacrificing only 0.3% for migration overhead.

• Security-induced Migrations:
  • Migrate execution (probabilistically) when an indirect control transfer misses the code cache.
  • Forces an attacker to chain gadgets from different ISAs, making exploit generation extremely difficult.
Migration Overhead
(in the absence of an attack)

With a code cache as small as 768KB, we perform no security-induced migrations in the absence of an attack.
HIPStR: Heterogeneous-ISA Program State Relocation

PSR renders brute-force attacks computationally infeasible

Heterogeneous-ISA migration shields PSR from JIT-ROP attacks

Together, they form a formidable defense
Outline

• Motivation
• Return-Oriented Programming
• HIPStR: Heterogeneous-ISA Program State Relocation
  • Program State Relocation
  • Heterogeneous-ISA Migration
• Evaluation
  • Brute Force attacks
  • JIT-ROP attacks
  • Tailored Anti-diversification attacks
• Key Points
**Brute Force Attacks**

**Goal:** Construct a simple 4-gadget shellcode exploit.
i.e., populate %eax, %ebx, %ecx, and %edx with attacker-provided values.

Need just 20 minutes to accomplish all of this.
Brute Force Attack Surface under PSR

Best Case Scenario: Brute Force with surviving gadgets would take 56 trillion years to break PSR
**Just-In-Time Code Reuse Attacks**

**Goal:** Construct a simple 4-gadget shellcode exploit.

i.e., populate %eax, %ebx, %ecx, and %edx with attacker-provided values.

Thousands of gadgets in just 23 seconds
JIT-ROP Attack Surface under HIPStR

Only 27 gadgets bypass migration – insufficient to construct a simple shellcode exploit.
Software Diversity vs ISA Diversity

Isomeron (NDSS 2015):
Why not migrate execution to a randomized version (isomer) of the same ISA at the flip of a coin?
Tailored Anti-Diversification Attacks

Goal: Stitch together gadgets across heterogeneous-ISAs (or isomers)

• **NOP gadgets**: Gadget performs useful operation in one ISA (isomer) and acts as a NOP in another.

• **Immutable gadgets**: Gadget performs the same operation on both ISAs (isomers) without clobbering any previously stored values.
HIPStR Attack Surface Reduction

Hundreds of gadgets survive Isomeron, but only 2 gadgets survive HIPStR
HIPStR Performance

HIPStR outperforms Isomerond by an average of 15.6%
Key Points

• Harnessing ISA Diversity is important – it not just beneficial in terms of performance and efficiency, but provides immense security benefits.

• HIPStR removes one of the last remaining constants available to the attacker – knowledge of the underlying ISA.

• HIPStR outperforms the only other JIT-ROP defense by 15.6%, while simultaneously providing greater protection against JIT-ROP, Blind-ROP, and many evasive variants.
Thank You!