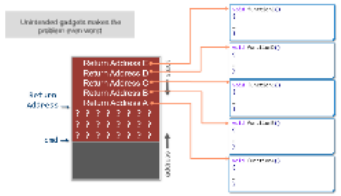
Protecting The Code

Control-flow Enforcement Technology

Oren Benita Ben Simhon Monday, April 16, 2018

ROP Attack
Manipulating The Stack



Overview & Agenda

ROP Attack
Using Unintended Gadgets

ROP Attack
Is It That Critical?

95%

Similar Attack Techniques

Defenses Against ROP/JOP/COP Attacks
Control-flow Enforcement Technology

- Control-flow Integrity (CFI) checks perform the following:
  - Indirect branches target only valid target addresses
  - Return instructions should only transfer control to the call site
- Intel® Control-flow Enforcement Technology (CET) is a CPU instruction set extension to implement CFI

Shadow Stack
- Prevents ROP attacks
- Saves control flow to a shadow stack

Indirect Branch Tracking
- Prevents JOP/COP
- Allows branching only to valid targets

Keeping Shadow Stack In Sync

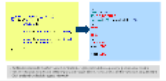
Protection #1: Shadow Stack
Protecting From ROP and Return Address Corruption On Stack

- Shadow stack is a separate stack used exclusively for control transfer operations and is separate from data stack
- Shadow stack supporting processors has a new register – Shadow Stack Pointer (SSP)
- Writes to the shadow stack are restricted to control transfer instructions and special protected instructions

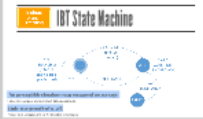
- Call -> Pushes return address on both stacks
  - No parameters passing on shadow stack
  - Far calls push code Segment (CS), Linear Instruction Pointer (LIP) and SSP
- Ret -> pops return address from both stacks
  - Control Flow Protection (#CP) exception in case return addresses don't match

Shadow Stack introduces mean Instruction Per Cycle loss of less than 2%

Fine-grained Indirect Branch Tracking
ARL TRACK Prefix and Legacy Compatibility

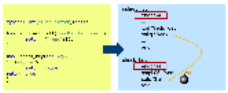


IBT State Machine



Protection #2: Indirect Branch Tracking
Protecting From JOP and COP Attacks

- Indirect Branch Tracking (IBT) detects and prevents attempts to redirect control flow to unintended targets
- IBT introduces new instructions
  - ENDBRANCH32 for 32 bit programs
  - ENDBRANCH64 for 64 bit programs
- ENDBRANCH instructions are NOP instructions on Intel® processors that do not support CET
- If a target instruction of indirect jump / call has no ENDBRANCH instruction a #CP exception is fired



- Compiler inserts ENDBRANCH instruction to:
  - Instructions that their address was taken
  - Global functions
- A new noct_check attribute was added to:
  - Disable ENDBRANCH instruction in the beginning of a function
  - Add no_track prefix to indirect jump/call to disable control flow check

CET Security Analysis

CET Status and Future Work

Summary

Control-flow Enforcement Technology (CET)
- Introduces new HW based Control Flow Integrity (CFI) mechanism

Shadow Stack and Indirect Branch Tracking (IBT)
- Shadow Stack protects against ROP attacks
- Indirect Branch Tracking protects against JOP/COP attacks

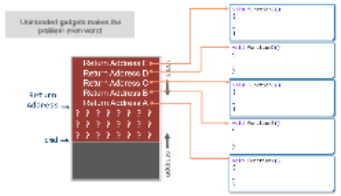
Low Overhead
- CET introduces competitive protection metric rates while maintaining very low performance overhead

Protecting The Code

Control-flow Enforcement Technology

Oren Benita Ben Simhon Monday, April 16, 2018

ROP Attack
Manipulating The Stack



Overview & Agenda

ROP Attack
Using Unintended Gadgets

ROP Attack
Is It That Critical?

95%

Similar Attack Techniques

Defenses Against ROP/JOP/COP Attacks
Control-flow Enforcement Technology

- Control-flow Integrity (CFI) checks perform the following:
  - Indirect branches target only valid target addresses
  - Return instructions should only transfer control to the call site
- Intel® Control-flow Enforcement Technology (CET) is a CPU instruction set extension to implement CFI

Shadow Stack
- Prevents ROP attacks
- Saves control flow to a shadow stack

Indirect Branch Tracking
- Prevents JOP/COP
- Allows branching only to valid targets

Keeping Shadow Stack In Sync

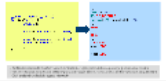
Protection #1: Shadow Stack
Protecting From ROP and Return Address Corruption On Stack

- Shadow stack is a separate stack used exclusively for control transfer operations and is separate from data stack
- Shadow stack supporting processors has a new register – Shadow Stack Pointer (SSP)
- Writes to the shadow stack are restricted to control transfer instructions and special protected instructions

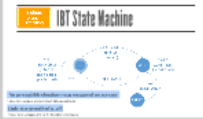
- Call -> Pushes return address on both stacks
  - No parameters passing on shadow stack
  - Far calls push code Segment (CS), Linear Instruction Pointer (LIP) and SSP
- Ret -> pops return address from both stacks
  - Control Flow Protection (#CP) exception in case return addresses don't match

Shadow Stack introduces mean Instruction Per Cycle loss of less than 2%

Fine-grained Indirect Branch Tracking
ARL TRACK Prefix and Legacy Compatibility

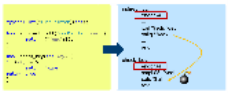


IBT State Machine



Protection #2: Indirect Branch Tracking
Protecting From JOP and COP Attacks

- Indirect Branch Tracking (IBT) detects and prevents attempts to redirect control flow to unintended targets
- IBT introduces new instructions
  - ENDBRANCH32 for 32 bit programs
  - ENDBRANCH64 for 64 bit programs
- ENDBRANCH instructions are NOP instructions on Intel® processors that do not support CET
- If a target instruction of indirect jump / call has no ENDBRANCH instruction a #CP exception is fired



- Compiler inserts ENDBRANCH instruction to:
  - Instructions that their address was taken
  - Global functions
- A new noct_check attribute was added to:
  - Disable ENDBRANCH instruction in the beginning of a function
  - Add no_track prefix to indirect jump/call to disable control flow check

CET Security Analysis

CET Status and Future Work

Summary

Control-flow Enforcement Technology (CET)
- Introduces new HW based Control Flow Integrity (CFI) mechanism

Shadow Stack and Indirect Branch Tracking (IBT)
- Shadow Stack protects against ROP attacks
- Indirect Branch Tracking protects against JOP/COP attacks

Low Overhead
- CET introduces competitive protection metric rates while maintaining very low performance overhead

Protecting The Code

Control-flow Enforcement Technology

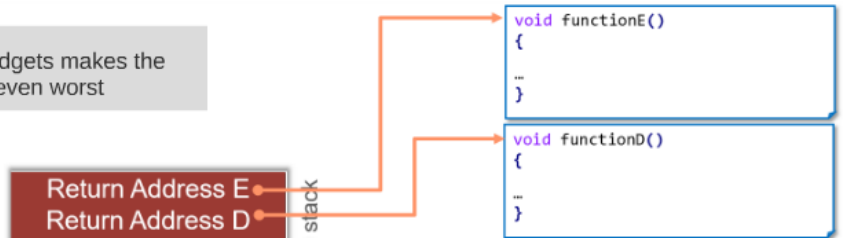
Oren Benita Ben Simhon Monday, April 16, 2018

intel

Motivation & Overview

ROP Attack
Manipulating The Stack

Unintended gadgets makes the problem even worst



Prezi

Overview & Agenda

Motivation & Overview → Shadow Stack → Indirect Branch Tracking → Analysis & Summary

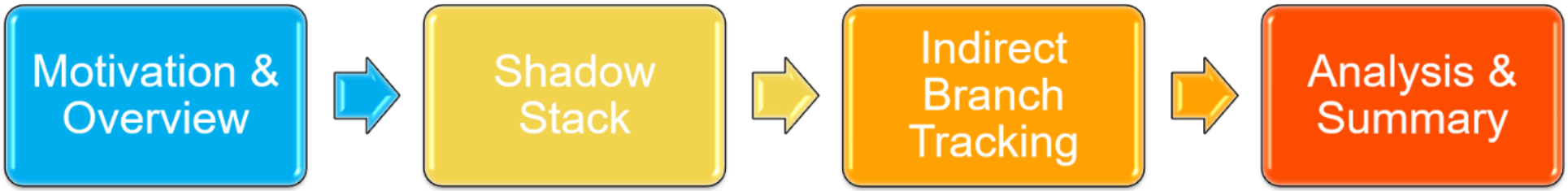
Motivation & Overview

ROP Attack
Using Unintended Gadgets

Disclaimers

- Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Learn more at Intel.com, or from the OEM or retailer.
- You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.
- No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.
- Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.
- No computer system can be absolutely secure.
- Intel and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.
-

Overview & Agenda



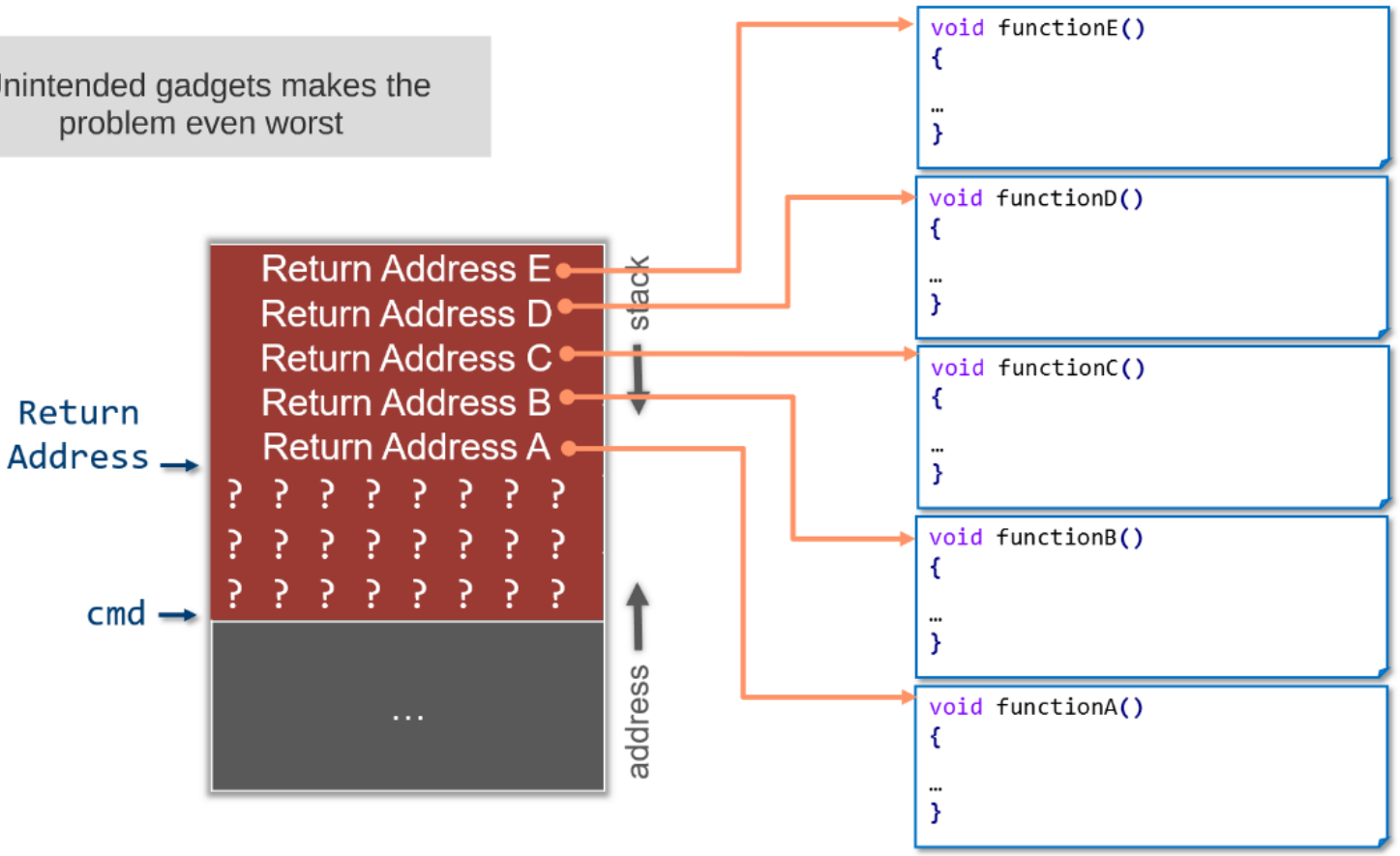
Motivation &

ROP Attack

Motivation & Overview

ROP Attack
Manipulating The Stack

Unintended gadgets makes the problem even worst

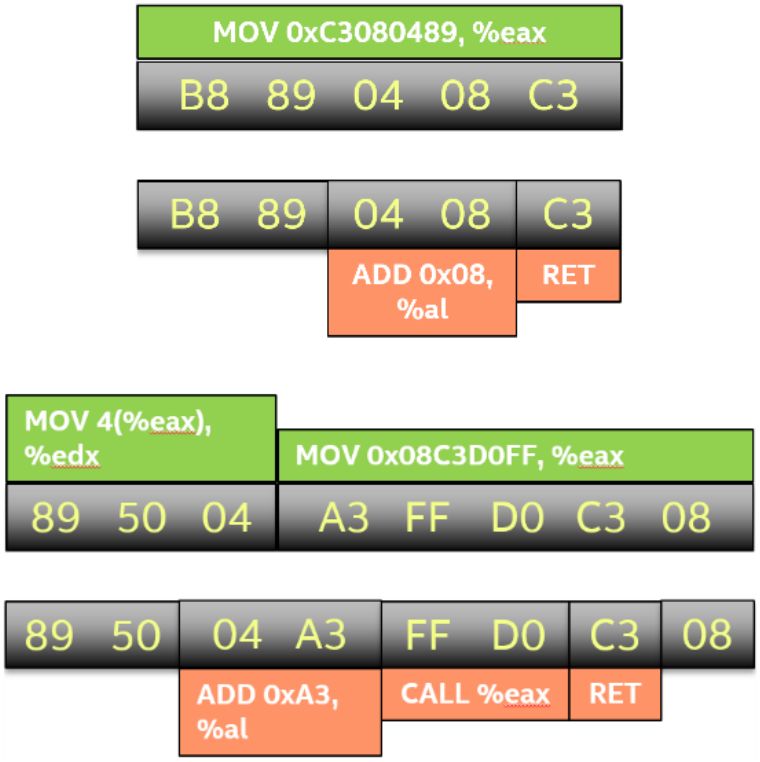


Motivation &
ROP Attack
Motivation &
Defenses Against ROP / JOP /

Motivation & Overview

ROP Attack

Using Unintended Gadgets



Overview

- Intel Arch allows instruction decoding to start from any byte
- Intel Arch has variable length instruction
- Attackers scan the code for meaningful snippets (gadgets)
- Attacker can execute chained gadgets

Motivation & Overview

ROP Attack

Is It That Critical?

* Tim Rains, David Weston, Matt Miller. Exploitation Trends: From Potential Risk to Actual Risk. In RSA conference 2015.

* Tim Rains, David Weston, Matt Miller. Exploitation Trends: From Potential Risk to Actual Risk. In RSA conference 2015.

Motivation & Overview

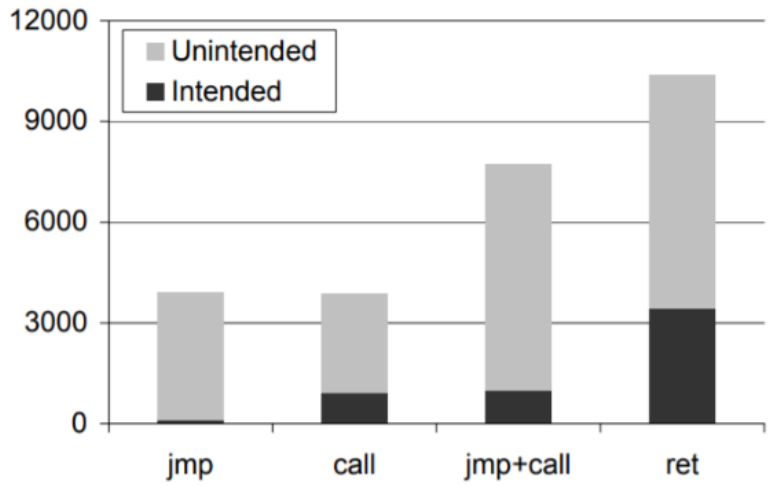
Similar Attack Techniques

JOP - Jump Oriented Programming

- Each gadget block ends with JMP instruction

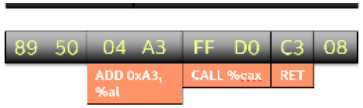
COP - Call Oriented Programming

- Each gadget block ends with Call instruction



* Tyler Bletsch, Xuxian Jiang, Vince Freeh. Jump-Oriented Programming: A New Class of Code-Reuse Attack. April 22, 2010

```
void functionA()
{
…
}
```



- Attacker can execute chained gadgets

Motivation & Overview

Defenses Against ROP/JOP/COP Attacks

Control-flow Enforcement Technology

- Control-flow Integrity (CFI) checks perform the following:
  - Indirect branches target only valid target addresses
  - Return instructions should only transfer control to the call site
- Intel® Control-flow Enforcement Technology (CET) is a CPU instruction set extension to implement CFI

Shadow Stack

- Prevents ROP attacks
- Saves control flow to a shadow stack

Indirect Branch Tracking

- Prevents JOP/COP
- Allows branching only to valid targets

Prezi Stack

Protection #1: Shadow Stack

targets

Shadow Stack

Protection #1: Shadow Stack

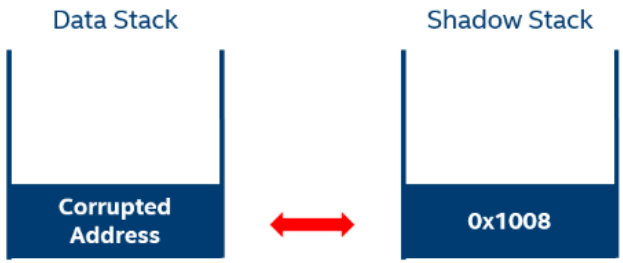
Protecting From ROP and Return Address Corruption On Stack

- Shadow stack is separate stack used exclusively for control transfer operations and is separate from data stack
- Shadow stack supporting processors use a new register – Shadow Stack Pointer (SSP)
- Writes to the shadow stack are restricted to control transfer instructions and special protected instructions

- Call -> Pushes return address on both stacks
  - No parameters passing on shadow stack
  - Far calls push Code Segment (CS), Linear Instruction Pointer (LIP) and SSP
- Ret -> pops return address from both stacks
  - Control Flow Protection (#CP) exception in case return addresses don't match

Shadow Stack Introduces mean Instruction-Per-Cycle loss of less than 2%

* Calculated using ICC compiler using a suite of microprocessor benchmarks




```
0x1000:
        call 0x2000
0x1008:
        ret
0x2000:
        <Stack Corruption>
        ret
```

Indirect Branch Tracking

Protection #2: Indirect Branch Tracking

Shadow Stack

Keeping Shadow Stack In Sync

Setjmp / Longjmp

```
int foo(int i) {
        if (!setjmp(buf)) {
                printf("After setjmp");
                bar(i);
        }
        return i + i;
}
```

```
int bar(int i) {
        printf("In longjmp");
        longjmp(buf, 1);
        return j;
}
```

- The compiler needs to save the SSP in the jump buffer
- The compiler increments SSP by skipped number of frames
- New instructions were introduced RDSSP and INCSSP

Exception Handling

```
int C() {
        try
        {
                B();
        } catch (int e) {
                cout << e << '\n';
        }

        return 0;
}
```

```
int B() {
        return A()
}

int A() {
        throw 20;
}
```

- C++ runtime library is updated to use indirect jump instead of return
- It also needs to increment the SSP to pop skipped call frames

Context Switching

Different shadow stacks for each privilege level

Each shadow stack is setup by Operating System

- The OS save/restore SSP for thread switching
- New ISA was added SAVEPREVSSP and RSTORSSP

Shadow Stack

Protecti

- Shadow sta
  transfer op
- Shadow st
  Shadow St
- Writes to t
  instructions

- Call -> Pus
  - No para
  - Far call
    Pointer
- Ret -> pop
  - Control
    address

Setjmp / Longjmp

```
int foo(int i) {
        if (!setjmp(buf)) {
                printf("After setjmp");
                bar(i);
        }
        return i + i;
}
```

```
int bar(int i) {
        printf("In longjmp");
        longjmp(buf, 1);
        return j;
}
```

- The compiler needs to save the SSP in the jump buffer
- The compiler increments SSP by skipped number of frames
- New instructions were introduced RDSSP and INCSSP

Exception Handling

```
int C() {
        try
        {
                B();
        } catch (int e) {
                cout << e << '\n';
        }

        return 0;
}
```

```
int B() {
        return A()
}

int A() {
…
        throw 20;
…
}
```

- C++ runtime library is updated to use indirect jump instead of return
- It also needs to increment the SSP to pop skipped call frames

Context Switching

Different shadow stacks for each privilege level

Each shadow stack is setup by Operating System

- The OS save/restore SSP for thread switching
- New ISA was added SAVEPREVSSP and RSTORSSP

- Control Flow Protection (#CP) exception in case return addresses don't match

Indirect Branch Tracking

Protection #2: Indirect Branch Tracking

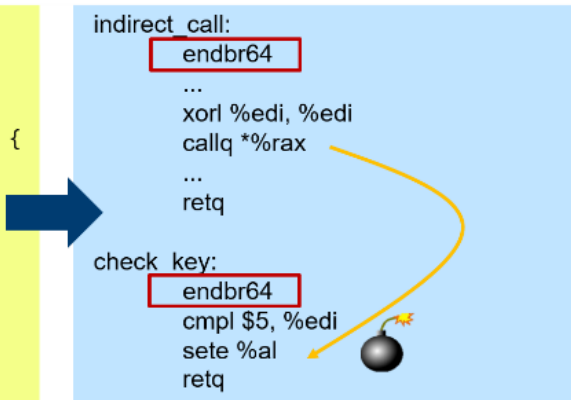
Protecting From JOP and COP Attacks

- Indirect Branch Tracking (IBT) detects and prevents attempts to redirect control flow to unintended targets
- IBT introduces new instructions:
  - ENDBRANCH32 for 32 bit programs
  - ENDBRANCH64 for 64 bit programs
- ENDBRANCH instructions are NOP instructions on Intel 64 processors that do not support CET
- If a target instruction of indirect jump / call has no ENDBRANCH instruction a #CP exception is fired

```
typedef int(*FuncPointer)(int);

bool indirect_call(FuncPointer func) {
        return (*func)(0);
}

bool check_key(int key) {
if (key != 5)
        return false;
return true;
}
```



- Compiler instruments ENDBRANCH instruction to:
  - Instructions/functions that their address was taken
  - Global functions
- A new nocf_check attribute was added to:
  - Disable ENDBRANCH instruction in the beginning of a function
  - Add no_track prefix to indirect jump/call to disable control flow check

Prezi Analysis & Summary

Summary

Indirect Branch Tracking

Fine-grained Indirect Branch Tracking

NO_TRACK Prefix and Legacy Compatibility

```
__attribute((nocf_check))
int foo(int a) {
        switch (a)
        {
        case 0: return a - 2; break;
        …
        case 8: return a >> 2; break;
        default: return a;
        }
}
```

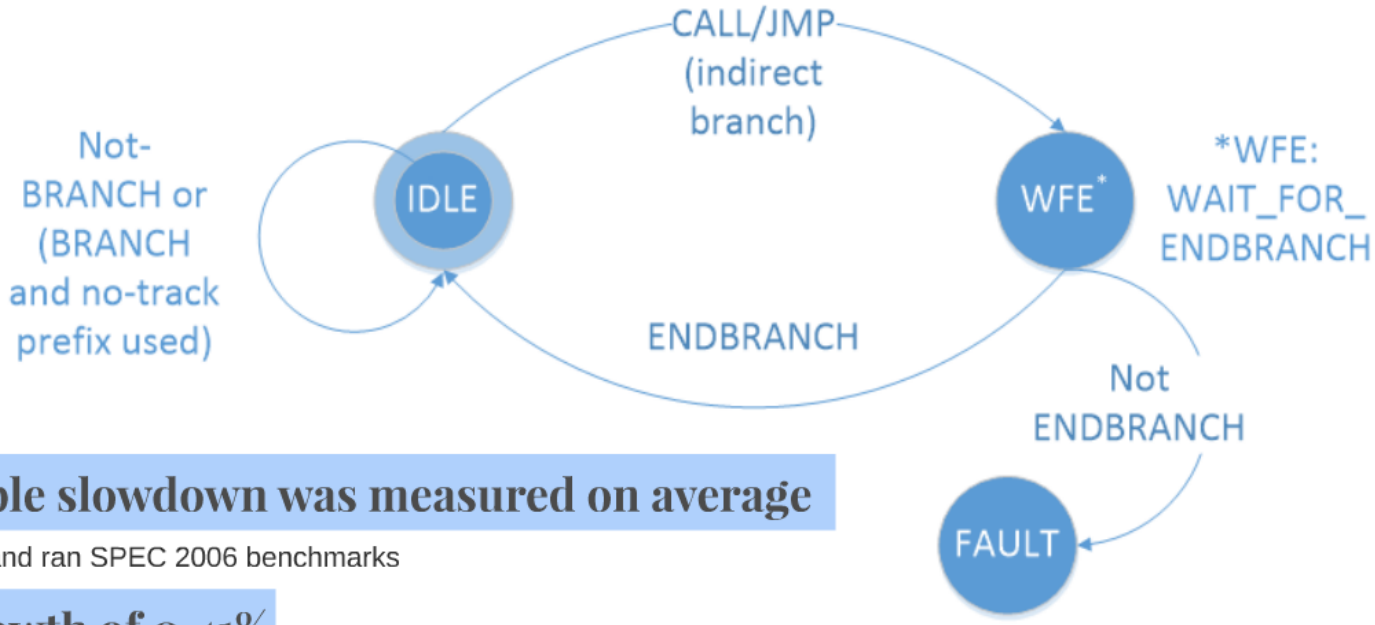
```
…
CMP $8, %RAX
JG .BB_DEFAULT
no_track JMP JumpTable(%RAX)
…
.BB0:
…
.BB8:
…
.JumpTable:
.quad .BB0
…
.quad .BB8
```

- Software may restrict certain sensitive functions in program address space (e.g. exec, execv, etc.)
- OS and dynamic loader can setup legacy code page bitmap to support code that was not compiled with CET enabled or disable legacy interwork

- Software may restrict certain sensitive functions in program address space (e.g. exec, execv, etc.)
- OS and dynamic loader can setup legacy code page bitmap to support code that was not compiled with CET enabled or disable legacy interwork

Indirect Branch Tracking

IBT State Machine



No perceptible slowdown was measured on average

* Used ICC compiler and ran SPEC 2006 benchmarks

Code size growth of 0.41%

* Used GCC compiler and ran SPEC 2006 benchmarks

Analysis & Summary

CET Security Analysis

Shadow Stack

- Enforces a function to return to its call site
  - Prevents pivoting the shadow stack because its page and the SSP are non-writeable
    - Prevents reusing old-frames in the shadow stack because you can only release frames
      - Keeps stack ABI intact – no changes to data stack layout

- Doesn't require other check tools
  - Can be applied to full stack (kernel to applications) and support dynamic linking
    - Not limited to C/C++ programming languages
      - Very low performance overhead

Indirect Branch Tracking

- Enforces indirect calling / jumping to valid addresses (no unintended gadgets)
  - AIR: SPEC 2006 benchmarks are prone to attacks by 0.02%
    - Very difficult to chain indirect branches to intended gadgets and create a meaningful program
      - SW may instrument checks in intended gadgets
        - Neglectable performance overhead

Analysis & Summary

CET Security Analysis

Shadow Stack

- Enforces a function to return to its call site
- Prevents pivoting the shadow stack because its page and the SSP are non-writeable
- Prevents reusing old-frames in the shadow stack because you can only release frames
- Keeps stack ABI intact – no changes to data stack layout

- Doesn't require other check tools
- Can be applied to full stack (kernel to applications) and support dynamic linking
- Not limited to C/C++ programming languages
- Very low performance overhead

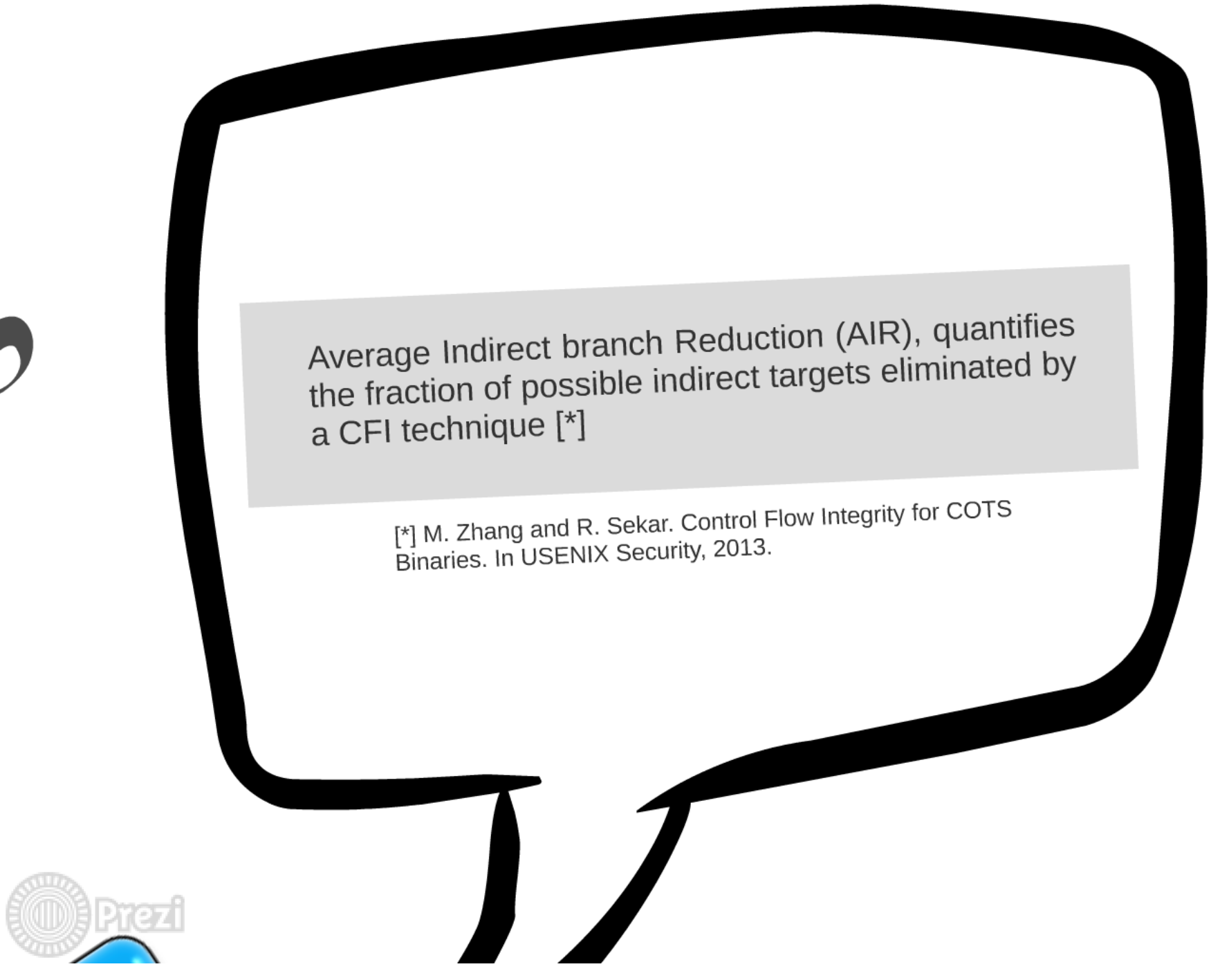
Indirect Branch Tracking

Indirect Branch Tracking

- Enforces indirect calling / jumping to valid addresses (no unintended gadgets)
- AIR: SPEC 2006 benchmarks are prone to attacks by 0.02%
- Very difficult to chain indirect branches to intended gadgets and create a meaningful program
- SW may instrument checks in intended gadgets
- Neglectable performance overhead

Prezi Analysis &

CFI Status and Future Work


Average Indirect branch Reduction (AIR), quantifies the fraction of possible indirect targets eliminated by a CFI technique [*]
[*] M. Zhang and R. Sekar. Control Flow Integrity for COTS Binaries. In USENIX Security, 2013.

Indirect Branch Tracking

- Enforces indirect calling / jumping to valid addresses (no unintended gadgets)
- AIR: SPEC 2006 benchmarks are prone to attacks by 0.02%
- Very difficult to chain indirect branches to intended gadgets and create a meaningful program
- SW may instrument checks in intended gadgets
- Neglectable performance overhead

Prezi Analysis &

CFI Status and Future Work

SW may instrument checks in intended gadgets

Neglectable performance overhead

Analysis & Summary

CET Status and Future Work

- LLVM already supports Shadow Stack and IBT (including optimizations)
- The architecture is enabled using **-mshstk** / **-mibt** flags
- Instrumentation is enabled using ***-fcf-protection = return/branch*** flag
- New attribute **nocf_check** is currently supported
- ICC / GCC implemented CET and updated corresponding libraries, program loader and linker (ld)
- MS Compiler is also being updated

- In the future a super set flag of -mibt & -mshstk called -**mcet** will be added
- A fix up for setJump / longJump is being promoted into LLVM
- LLVM IInker will also be updated to support new ABI flags and generating IBT-enabled PLT

Analysis & Summary

Summary

Control-flow Enforcement Technology (CET)

- Introduces new HW based Control Flow Integrity (CFI) mechanism

Shadow Stack and Indirect Branch Tracking (IBT)

- Shadow Stack protects against ROP attacks
- Indirect Branch Tracking protects against JOP/COP attacks

Low Overhead

- CET introduces competitive protection metric rates while maintaining very low performance overhead

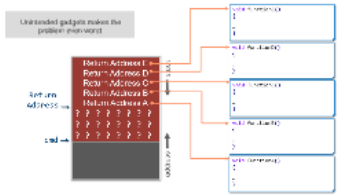
?

Protecting The Code

Control-flow Enforcement Technology

Oren Benita Ben Simhon Monday, April 16, 2018

ROP Attack
Manipulating The Stack



Overview & Agenda

ROP Attack
Using Unintended Gadgets

ROP Attack
Is It That Critical?

95%

Similar Attack Techniques

Defenses Against ROP/JOP/COP Attacks
Control-flow Enforcement Technology

- Control-flow Integrity (CFI) checks perform the following:
  - Indirect branches target only valid target addresses
  - Return instructions should only transfer control to the call site
- Intel® Control-flow Enforcement Technology (CET) is a CPU instruction set extension to implement CFI

Shadow Stack
- Prevents ROP attacks
- Saves control flow to a shadow stack

Indirect Branch Tracking
- Prevents JOP/COP
- Allows branching only to valid targets

Keeping Shadow Stack In Sync

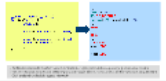
Protection #1: Shadow Stack
Protecting From ROP and Return Address Corruption On Stack

- Shadow stack is a separate stack used exclusively for control transfer operations and is separate from data stack
- Shadow stack supporting processors has a new register – Shadow Stack Pointer (SSP)
- Writes to the shadow stack are restricted to control transfer instructions and special protected instructions

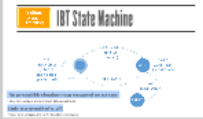
- Call -> Pushes return address on both stacks
  - No parameters passing on shadow stack
  - Far calls push code Segment (CS), Linear Instruction Pointer (LIP) and SSP
- Ret -> pops return address from both stacks
  - Control Flow Protection (#CP) exception in case return addresses don't match

Shadow Stack introduces mean Instruction Per Cycle loss of less than 2%

Fine-grained Indirect Branch Tracking
ARL TRACK Prefix and Legacy Compatibility

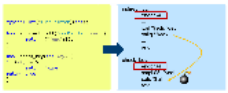


IBT State Machine



Protection #2: Indirect Branch Tracking
Protecting From JOP and COP Attacks

- Indirect Branch Tracking (IBT) detects and prevents attempts to redirect control flow to unintended targets
- IBT introduces new instructions
  - ENDBRANCH32 for 32 bit programs
  - ENDBRANCH64 for 64 bit programs
- ENDBRANCH instructions are NOP instructions on Intel® processors that do not support CET
- If a target instruction of indirect jump / call has no ENDBRANCH instruction a #CP exception is fired



- Compiler inserts ENDBRANCH instruction to:
  - Instructions that their address was taken
  - Global functions
- A new noct_check attribute was added to:
  - Disable ENDBRANCH instruction in the beginning of a function
  - Add no_track prefix to indirect jump/call to disable control flow check

CET Security Analysis

CET Status and Future Work

Summary

Control-flow Enforcement Technology (CET)
- Introduces new HW based Control Flow Integrity (CFI) mechanism

Shadow Stack and Indirect Branch Tracking (IBT)
- Shadow Stack protects against ROP attacks
- Indirect Branch Tracking protects against JOP/COP attacks

Low Overhead
- CET introduces competitive protection metric rates while maintaining very low performance overhead