FineIBT

Fine-grain Control-flow Enforcement with Indirect Branch Tracking

Alexander J. Gaidis\textsuperscript{1} Joao Moreira\textsuperscript{2} Ke Sun\textsuperscript{2} Alyssa Milburn\textsuperscript{2}
Vaggelis Atlidakis\textsuperscript{1} Vasileios P. Kemerlis\textsuperscript{1}

\textsuperscript{1}Secure Systems Laboratory (SSL)
Department of Computer Science
Brown University

\textsuperscript{2}IPAS STORM
Intel
CFI Fundamentals

- CFI defends against control-flow hijacking by:

  1. Mapping legal control flows to CFI policy
  2. Enforcing the policy at runtime
CFI Fundamentals

- CFI defends against control-flow hijacking by:
  1. Mapping legal control flows $\leadsto$ CFI policy

```
  A
  /   \
 B   C
  \
   D
```
Control-flow Integrity

CFI Fundamentals

- CFI defends against control-flow hijacking by:
  1. Mapping legal control flows $\leadsto$ CFI policy
  2. Enforcing the policy at run time

A → B → C → D
CFI Fundamentals

- CFI defends against control-flow hijacking by:
  1. Mapping legal control flows \(\sim\) CFI policy
  2. Enforcing the policy at run time
- Attackers can tamper with code pointers
CFI Fundamentals

- CFI defends against control-flow hijacking by:
  1. Mapping legal control flows to CFI policy
  2. Enforcing the policy at run time

- Attackers can tamper with code pointers

- Computed control-flow transfers are confined e.g., `jmp *%reg`, `call *%reg`, `ret

Agreement: A

Violation: C

Validation: D

Target Domain: Where the scheme is applicable
Compatibility: What the scheme requires
Coverage: Forward edges and/or backward edges
Effectiveness: Size of indirect branch target sets

Fewer Targets: Fine-grain
More Targets: Coarse-grain
CFI Fundamentals

- CFI defends against control-flow hijacking by:
  1. Mapping legal control flows \( \sim \) CFI policy
  2. Enforcing the policy at run time
- Attackers can tamper with code pointers
- Computed control-flow transfers are confined e.g., `jmp *%reg`, `call *%reg`, `ret`

CFI Classification

- Target domain: where the scheme is applicable
Control-flow Integrity

CFI Fundamentals
- CFI defends against control-flow hijacking by:
  1. Mapping legal control flows $\rightsquigarrow$ CFI policy
  2. Enforcing the policy at run time
- Attackers can tamper with code pointers
- Computed control-flow transfers are confined e.g., jmp *%reg, call *%reg, ret

CFI Classification
- Target domain: where the scheme is applicable
- Compatibility: what the scheme requires
Control-flow Integrity

CFI Fundamentals
- CFI defends against control-flow hijacking by:
  1. Mapping legal control flows → CFI policy
  2. Enforcing the policy at run time
- Attackers can tamper with code pointers
- Computed control-flow transfers are confined
  e.g., jmp *%reg, call *%reg, ret

CFI Classification
- Target domain: where the scheme is applicable
- Compatibility: what the scheme requires
- Coverage: forward edges and/or backward edges
Control-flow Integrity

CFI Fundamentals

- CFI defends against control-flow hijacking by:
  1. Mapping legal control flows to CFI policy
  2. Enforcing the policy at run time
- Attackers can tamper with code pointers
- Computed control-flow transfers are confined e.g., jmp *%reg, call *%reg, ret

CFI Classification

- **Target domain**: where the scheme is applicable
- **Compatibility**: what the scheme requires
- **Coverage**: forward edges and/or backward edges
- **Effectiveness**: size of indirect branch target sets

agaidis@cs.brown.edu  (Brown University)
Control-flow Integrity

CFI Fundamentals
- CFI defends against control-flow hijacking by:
  1. Mapping legal control flows \( \rightarrow \) CFI policy
  2. Enforcing the policy at run time
- Attackers can tamper with code pointers
- Computed control-flow transfers are confined e.g., \texttt{jmp *\%reg}, \texttt{call *\%reg}, \texttt{ret}

CFI Classification
- Target domain: where the scheme is applicable
- Compatibility: what the scheme requires
- Coverage: forward edges and/or backward edges
- Effectiveness: size of indirect branch target sets
  - Fewer Targets \( \rightarrow \) Fine-grain
  - More Targets \( \rightarrow \) Coarse-grain
CET

- CET is a CFI hardware feature on modern Intel CPUs
CET

- CET is a CFI hardware feature on modern Intel CPUs
- Two components: shadow stack, indirect branch tracking
Control-flow Enforcement Technology

CET

- CET is a CFI hardware feature on modern Intel CPUs
- Two components: shadow stack, indirect branch tracking

Indirect Branch Tracking

- New 4-byte instruction: `endbr`
## CET
- CET is a CFI **hardware feature** on modern Intel CPUs
- Two components: **shadow stack, indirect branch tracking**

## Indirect Branch Tracking
- New 4-byte instruction: `endbr`
- Indirect branches must target an `endbr`
Control-flow Enforcement Technology

CET
- CET is a CFI hardware feature on modern Intel CPUs
- Two components: shadow stack, indirect branch tracking

Indirect Branch Tracking
- New 4-byte instruction: `endbr`
- Indirect branches must target an `endbr`

Target Domain: Anywhere HW feature is available
CET

- CET is a CFI hardware feature on modern Intel CPUs
- Two components: shadow stack, indirect branch tracking

Indirect Branch Tracking

- New 4-byte instruction: `endbr`
- Indirect branches must target an `endbr`

🌐 **Target Domain**: Anywhere HW feature is available

🛠️ **Compatibility**: No major requirements (just enable feature and add `endbrs`)
**CET**
- CET is a CFI *hardware feature* on modern Intel CPUs
- Two components: *shadow stack*, *indirect branch tracking*

**Indirect Branch Tracking**
- New 4-byte instruction: `endbr`
- Indirect branches must target an `endbr`

- **Target Domain**: Anywhere HW feature is available
- **Compatibility**: No major requirements (just enable feature and add `endbrs`)
- **Coverage**: Forward-edges
Control-flow Enforcement Technology

CET
- CET is a CFI hardware feature on modern Intel CPUs
- Two components: shadow stack, indirect branch tracking

Indirect Branch Tracking
- New 4-byte instruction: `endbr`
- Indirect branches must target an `endbr`

Target Domain: Anywhere HW feature is available
Compatibility: No major requirements (just enable feature and add `endbrs`)
Coverage: Forward-edges
Effectiveness: Coarse-grain
CET $\Rightarrow$ IBT

```
main
  ... 
  call func0 
  ... 
  call *%rcx # func2 
  ... 

func0
  ... 
  call *%rcx # func1 
  ... 

func1
  endbr 
  ... 

func2
  endbr 
  ... 
```

agaidis@cs.brown.edu  (Brown University)
... call func0
... call *%rcx # func2
...

main
endbr
...

call *%rcx # func1
...

func0

func1
endbr
...

func2
endbr
...

call *%rcx # func2
...
main
  endbr
  ...
  call func0
  ...
  call *%rcx # func2
  ...

func0
  ...
  call *%rcx # func1
  ...

func1
  endbr
  ...

func2
  endbr
  ...

agaidis@cs.brown.edu  (Brown University)
main
endbr
...
call func0
...
call *%rcx # func2
...

func0
...
call *%rcx # func1
...

func1
endbr
...

func2
endbr
...

agaidis@cs.brown.edu  (Brown University)
CET → IBT

main

endbr
...
call func0
...
call *%rcx # func2
...

func0

call *%rcx # func1
...

func1

endbr
...

func2

endbr
...

agaidis@cs.brown.edu (Brown University)
CET → IBT

main

endbr
...
call func0
...
call *%rcx # func2
...

func0

... call *%rcx # func1
...

func1

endbr
...

func2

endbr
...

agaidis@cs.brown.edu (Brown University)
IBT Summary

- **Performant**
**IBT Summary**

- **Performant**
- **Enforces a coarse-grain policy**
IBT Summary

+ Performant

- Enforces a coarse-grain policy
- Libraries greatly increase the number of indirect branch targets
## IBT Summary

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performant</td>
<td>Enforces a coarse-grain policy</td>
</tr>
<tr>
<td></td>
<td>Libraries greatly increase the number of indirect branch targets</td>
</tr>
<tr>
<td></td>
<td><em>e.g.</em>, GLIBC exports &gt; 2000 functions</td>
</tr>
</tbody>
</table>
**IBT Summary**

- **Performant**
- Enforces a coarse-grain policy
- Libraries greatly increase the number of indirect branch targets
  
  *e.g.*, GLIBC exports > 2000 functions

**FineIBT**

+ Retrofits IBT with light-weight instrumentation
+ Provides a mechanism to refine IBT's default policy
+ Supports coarse- and fine-grain CFI policies

*agaidis@cs.brown.edu* (Brown University)
**IBT Summary**

- **Performant**
- Enforces a coarse-grain policy
- Libraries greatly increase the number of indirect branch targets
  
  *e.g., GLIBC exports > 2000 functions*

**FineIBT**

- Retrofits IBT with light-weight instrumentation
**IBT Summary**

- **Performant**

- Enforces a coarse-grain policy

- Libraries greatly increase the number of indirect branch targets
  
  *e.g.*, GLIBC exports $> 2000$ functions

**FineIBT**

- Retrofits IBT with light-weight instrumentation

- Provides a *mechanism* to refine IBT’s default policy
### IBT Summary

- **Performant**
- Enforces a coarse-grain policy
- Libraries greatly increase the number of indirect branch targets
  
  *e.g.*, GLIBC exports > 2000 functions

### FineIBT

- Retrofits IBT with light-weight instrumentation
- Provides a mechanism to refine IBT’s default policy
- Supports coarse- and fine-grain CFI policies
IBT Summary

- Performant
- Enforces a coarse-grain policy
- Libraries greatly increase the number of indirect branch targets
  e.g., GLIBC exports > 2000 functions

FineIBT

- Retrofits IBT with light-weight instrumentation
- Provides a mechanism to refine IBT's default policy
- Supports coarse- and fine-grain CFI policies
  - Policy agnostic
```
main:

... 

call *%rcx  # func0

... 

func1:

... 
```
main:
...

call *%rcx  # func0
...

---

func0:
...

---

func1:
...

---

agaidis@cs.brown.edu (Brown University)
FineIBT ➔ Foundational Instrumentation

main:
  ...
  call  *%rcx  # func0
  ...
  func0:
    endbr
    ...
  func1:
    endbr
    ...

agaidis@cs.brown.edu  (Brown University)
main:
...
call *%rcx  # %ecx0
...

\[ \rightarrow \text{func0:} \]
\[ \quad \text{endbr} \]
\[ \quad \ldots \]

\[ \rightarrow \text{func1:} \]
\[ \quad \text{endbr} \]
\[ \quad \ldots \]
main:

...  

# SID: 0xc00010ff  
call *%rcx  # func0  
...  

func0:  # SID: 0xc00010ff  
endbr  
...  

func1:  # SID: Oxbaddcafe  
endbr  
...
FineIBT → Foundational Instrumentation

main:

...  

mov  $0xc00010ff, %eax

# SID:

call  *%rcx  # func0  

endbr  

...  

func0:  # SID: 0xc00010ff

...  

func1:  # SID: 0xbaddcafe

endbr  

...
main:

...  

mov $0xc00010ff, %eax

call *%rcx  # func0

...  

func0:

  endbr

  sub $0xc00010ff, %eax

  je func0_entry

  hlt

func0_entry:

  ...

func1:

  endbr

  sub $0xbaddcafe, %eax

  je func1_entry

  hlt

func1_entry:

  ...
main:

...  

mov $0xc00010ff, %eax  
call *%rcx  
...  

func0:

endbr  
sub $0xc00010ff, %eax  
je func0_entry  
hlt  

func0_entry:

...  

func1:

endbr  
sub $0xbaddcafe, %eax  
je func1_entry  
hlt  

func1_entry:

...
main:
... 
mov $0xc00010ff, %eax 
call *%rcx # func0 
... 

---

func0:

endbr
sub $0xc00010ff, %eax 
je func0_entry 
hlt

func0_entry:
...

func1:

endbr
sub $0xbaddcafe, %eax 
je func1_entry 
hlt

func1_entry:
...
main:
  ...
  mov $0xc00010ff, %eax
  call *%rcx  # func0
  ...
  call func1_entry  ---
  ...
  func0:
    endbr
    sub $0xc00010ff, %eax
    je func0_entry
    hlt
    func0_entry:
      ...
  func1:
    endbr
    sub $0xbaddcafe, %eax
    je func1_entry
    hlt
    func1_entry:
      ...

agaidis@cs.brown.edu (Brown University)
Custom Error-handling Code

- hlt is in the IRM’s hot-path
func0:
  endbr
sub $0xc00010ff, %eax
je func0_entry
hlte
  ...

**Custom Error-handling Code**

- `hlt` is in the IRM’s hot-path
- Only executed on SID-check failure
func0:
  endbr
sub $0xc00010ff, %eax
je func0_entry

hltr

func0_entry:
...
Custom Error-handling Code

- `hlt` is in the IRM's hot-path
- Only executed on SID-check failure
- We can move it outside the hot-path:
  - `s/je/jne/`

路人判断：

- `hlt` is in the IRM's hot-path
- Only executed on SID-check failure
- We can move it outside the hot-path:
  - `s/je/jne/`
Custom Error-handling Code

- **hlt** is in the IRM’s hot-path
- Only executed on SID-check failure
- We can move it outside the hot-path:
  → s/je/jne/
- The **hlt** can also be swapped for custom error-handlers without affecting performance
**Custom Error-handling Code**

- `hlt` is in the IRM’s hot-path
- Only executed on SID-check failure
- We can move it outside the hot-path: 
  - `s/je/jne/`
- The `hlt` can also be swapped for custom error-handlers without affecting performance

```
.func0_coldpath:
... /* set arg0, ..., argn */
call __fineibt_chk_fail@PLT
func0:
  endbr
sub $0xc00010ff, %eax
jne .func0_coldpath
func0_entry:
...```
FineIBT ➔ Performance Considerations

main:
... 
mov $0xc00010ff, %eax

CPU Front-end and I-Cache

- Direct calls bypass FineIBT’s instrumentation

```assembly
CPU Front-end and I-Cache

- Direct calls bypass FineIBT’s instrumentation

```
main:

... 

mov $0xc00010ff, %eax

call *%rcx  # func0

...

call func0_entry

...

.func0_coldpath:

hlt

func0:

endbr

sub $0xc00010ff, %eax

jne .func0_coldpath

func0_entry:

...

---

CPU Front-end and I-Cache

- Direct calls bypass FineIBT's instrumentation
- 4 instructions for fine-grain CFI
  - Different policies, same overhead
main:
...

mov $0xc00010ff, %eax

call *%rcx  # func0
...

call func0_entry
...

.func0_coldpath:
  hlt

func0:
  endbr
  sub $0xc00010ff, %eax
  jne .func0_coldpath

func0_entry:
...

---

**CPU Front-end and I-Cache**

- Direct calls bypass FineIBT's instrumentation
- 4 instructions for fine-grain CFI
  - Different policies, same overhead
  - FineIBT's hot-path uses **16 bytes** in total:
    - 5 bytes per indirect branch site
    - + 11 bytes per indirect branch target
    - **16 bytes**
main:

...  

\texttt{mov} \ $0xc00010ff, \ %eax  
\texttt{call} \ \ast\%rcx \quad \# \ \texttt{func0}  

...  
\texttt{call} \ \texttt{func0\_entry}  

...  
\texttt{.func0\_coldpath:}  
\texttt{hlt}  
\texttt{func0:}  
\texttt{endbr}  
\texttt{sub} \ $0xc00010ff, \ %eax  
\texttt{jne} \ \texttt{.func0\_coldpath}  

\texttt{func0\_entry:}  

...  

\textbf{CPU Front-end and I-Cache}  

- Direct calls bypass FineIBT's instrumentation  
- 4 instructions for fine-grain CFI  
  - Different policies, same overhead  
  - FineIBT's hot-path uses 16 bytes in total:  
    \begin{align*}
    &5 \text{ bytes}\quad \text{per indirect branch site} \\
    + &11 \text{ bytes}\quad \text{per indirect branch target} \\
    \hline \\
    &16 \text{ bytes}
    \end{align*}  
  - Clang-CFI's hot-path uses 25 bytes in total:  
    \begin{align*}
    &20 \text{ bytes}\quad \text{per indirect branch site} \\
    + &5 \text{ bytes}\quad \text{per trampoline entry} \\
    \hline \\
    &25 \text{ bytes}
    \end{align*}
.func0_coldpath:
  hlt
func0:
  endbr
sub $0xc00010ff, %eax
jne .func0_coldpath
func0_entry:
  ...
EXEC

.text

call func4@PLT
...
call func3@PLT
...
call *%rcx # func2@PLT
...

.plt

func2@PLT:
   endbr
   ...
func3@PLT:
   endbr
   ...
func4@PLT:
   endbr
   ...

DSO

.text

func0:
   endbr
   ...
func1:
   endbr
   ...
func2:
   endbr
   ...
func3:
   endbr
   ...
func4:
   endbr
   ...

agaidis@cs.brown.edu  (Brown University)
Security Considerations

```assembly
EXEC

.text

call func4@PLT
...
call func3@PLT
...
call *%rcx # func2@PLT
...

.plt

func2@PLT:
  endbr
...
func3@PLT:
  endbr
...
func4@PLT:
  endbr
...

.DSO

.text

func0:
  endbr
...
func1:
  endbr
...
func2:
  endbr
...
func3:
  endbr
...
func4:
  endbr
...```
FineIBT ➔ Security Considerations

```
call func4@PLT
...  
call func3@PLT
...  
call [%rcx] # func2@PLT
...  
func2@PLT:
   endbr
...  
func3@PLT:
   endbr
...  
func4@PLT:
   endbr
...  
func0:
   endbr
...  
func1:
   endbr
...  
func2:
   endbr
...  
func3:
   endbr
...  
func4:
   endbr
...
```

agaidis@cs.brown.edu  (Brown University)
FineIBT → Security Considerations

EXEC

.text

call func4@PLT
...
call func3@PLT
...
call %rcx # func2@PLT
...

.endbr

DSO

.text

.func0:
    endbr
    ...
    func1:
        endbr
        ...
        func2:
            endbr
            ...
            func3:
                endbr
                ...
                func4:
                    endbr
                    ...

.plt

func0@PLT:
    endbr
    ...
func1@PLT:
    endbr
    ...
func2@PLT:
    endbr
    ...
func3@PLT:
    endbr
    ...
func4@PLT:
    endbr
    ...

IBT PLT

Every entry prefixed with endbr in case they are AT

agaidis@cs.brown.edu  (Brown University)
FineIBT
Every entry prefixed with `endbr` in case they are AT

- Only AT entries prefixed with `endbr`
- Entries hardened with FineIBT IRM

agaidis@cs.brown.edu (Brown University)
FineIBT → Security Considerations

EXEC

.text

call func4@PLT
...
call func3@PLT
...
call *%rcx # func2@PLT
...

.PLT

func2@PLT:
   endbr ✔
   ...
func3@PLT:
   endbr ☠
   ...
func4@PLT:
   endbr ☠
   ...

DSO

.text

func0:
   endbr
   ...
func1:
   endbr
   ...
func2:
   endbr
   ...
func3:
   endbr
   ...
func4:
   endbr
   ...

IBT PLT

⚠ Every entry prefixed with endbr in case they are AT

agaidis@cs.brown.edu (Brown University)
Security Considerations

Every entry prefixed with `endbr` in case they are AT

Only AT entries prefixed with `endbr`

Entries hardened with FineIBT IRM

agaidis@cs.brown.edu (Brown University)
FineIBT → Security Considerations

Every entry prefixed with `endbr` in case they are AT

Only AT entries prefixed with `endbr`

Entries hardened with FineIBT IRM

agaidis@cs.brown.edu  (Brown University)
FineIBT → Security Considerations

Exported Functions

- Unused (exported) DSO functions still contain endbrs

agaidis@cs.brown.edu (Brown University)
**Security Considerations**

```
EXEC

.text

call func4@PLT
...
call func3@PLT
...
call *%rcx # func2@PLT
...

.plt

func2@PLT:
  endbr
...
func3@PLT:
  endbr
...
func4@PLT:
  endbr
...

.DSO

.text

func0:
  endbr
...
func1:
  endbr
...
func2:
  endbr
...
func3:
  endbr
...
func4:
  endbr
...
```

**Exported Functions**

- Unused (exported) DSO functions still contain endbrs

**NOPout**

- Runtime library that refines indirect branch targets by eliding endbrs in unused functions
Features

- Compatible with existing defenses (e.g., XOM)
- Supports custom error-handling
- Supports secure cross-DSO function calls
- Provides a tool (NOPout) to dynamically refine indirect branch target sets
Features

- Compatible with existing defenses (e.g., XOM)
Features

- Compatible with existing defenses (e.g., XOM)
- Supports custom error-handling
### Features

- Compatible with existing defenses (e.g., XOM)
- Supports custom error-handling
- Supports secure cross-DSO function calls
## Features

- Compatible with existing defenses (*e.g.*, XOM)
- Supports custom error-handling
- Supports secure cross-DSO function calls
- Provides a tool (*NOPout*) to dynamically refine indirect branch target sets
# FineIBT → Summary

## Features

- Compatible with existing defenses (e.g., XOM)
- Supports custom error-handling
- Supports secure cross-DSO function calls
- Provides a tool (NOPout) to dynamically refine indirect branch target sets

## Classification

---

*agaidis@cs.brown.edu*  
(Brown University)
FineIBT ➔ Summary

Features

- Compatible with existing defenses (e.g., XOM)
- Supports custom error-handling
- Supports secure cross-DSO function calls
- Provides a tool (NOPout) to dynamically refine indirect branch target sets

Classification

- Target Domain: works in multiple domains (e.g., user/kernel space)
Features

- Compatible with existing defenses (e.g., XOM)
- Supports custom error-handling
- Supports secure cross-DSO function calls
- Provides a tool (NOPout) to dynamically refine indirect branch target sets

Classification

- **Target Domain**: works in multiple domains (e.g., user/kernel space)
- **Compatibility**: uses source code to increase precision, but can be applied to binaries
Features

- Compatible with existing defenses (e.g., XOM)
- Supports custom error-handling
- Supports secure cross-DSO function calls
- Provides a tool (NOPout) to dynamically refine indirect branch target sets

Classification

- **Target Domain**: works in multiple domains (e.g., user/kernel space)
- **Compatibility**: uses source code to increase precision, but can be applied to binaries
- **Coverage**: forward-edge protection (we assume a shadow stack for backward edges)
**Features**

- Compatible with existing defenses (e.g., XOM)
- Supports custom error-handling
- Supports secure cross-DSO function calls
- Provides a tool (NOPout) to dynamically refine indirect branch target sets

**Classification**

- **Target Domain**: works in multiple domains (e.g., user/kernel space)
- **Compatibility**: uses source code to increase precision, but can be applied to binaries
- **Coverage**: forward-edge protection (we assume a shadow stack for backward edges)
- **Effectiveness**: policy agnostic, supports coarse- and fine-grain CFI policies
FineIBT ➔ Implementation

FineIBT

- FineIBT is a set of modifications to the LLVM v12 toolchain
  - ≈700 C++ LOC added to the LLVM compiler (llvm-{gcc, g++})
  - ≈200 C++ LOC added to the LLVM linker (lld)
FineIBT Implementation

**FineIBT**
- FineIBT is a set of modifications to the LLVM v12 toolchain
  - ≈700 C++ LOC added to the LLVM compiler (llvm-{gcc, g++})
  - ≈200 C++ LOC added to the LLVM linker (lld)

**NOPout**
- NOPout is a DSO (libnopout.so) implemented in ≈800 C LOC
FineIBT ➔ Implementation

- **FineIBT**
  - FineIBT is a set of modifications to the LLVM v12 toolchain
    - ≈700 C++ LOC added to the LLVM compiler (llvm-{gcc, g++})
    - ≈200 C++ LOC added to the LLVM linker (lld)

- **NOPout**
  - NOPout is a DSO (libnopout.so) implemented in ≈800 C LOC

- **CFI Policy**
  - FineIBT is policy agnostic
  - Demonstrated FineIBT’s policy flexibility by testing different policies:
    - Arity-based CFI
    - Type-based CFI
    - MLTA-based CFI
## SPEC CPU 2017

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>IBT</th>
<th>FineIBT !EH</th>
<th>FineIBT EH</th>
<th>Clang-CFI</th>
</tr>
</thead>
<tbody>
<tr>
<td>600.perlbench</td>
<td>1.46%</td>
<td>1.51%</td>
<td>≈0%</td>
<td>1.17%</td>
</tr>
<tr>
<td>602.gcc</td>
<td>0.03%</td>
<td>0.12%</td>
<td>0.05%</td>
<td>0.6%</td>
</tr>
<tr>
<td>605.mcf</td>
<td>0.51%</td>
<td>3.67%</td>
<td>1.94%</td>
<td>2.04%</td>
</tr>
<tr>
<td>620.omnetpp</td>
<td>≈0%</td>
<td>0.91%</td>
<td>≈0%</td>
<td>6.57%</td>
</tr>
<tr>
<td>623.xalancbmk</td>
<td>≈0%</td>
<td>0.99%</td>
<td>≈0%</td>
<td>7.89%</td>
</tr>
<tr>
<td>625.x264</td>
<td>0.05%</td>
<td>0.29%</td>
<td>≈0%</td>
<td>0.93%</td>
</tr>
<tr>
<td>631.deepsjeng</td>
<td>0.03%</td>
<td>0.17%</td>
<td>≈0%</td>
<td>≈0%</td>
</tr>
<tr>
<td>641.leela</td>
<td>≈0%</td>
<td>≈0%</td>
<td>≈0%</td>
<td>≈0%</td>
</tr>
<tr>
<td>657.xz</td>
<td>≈0%</td>
<td>≈0%</td>
<td>≈0%</td>
<td>0.06%</td>
</tr>
</tbody>
</table>

- FineIBT exhibited ≈0%–1.94% slowdown
- Results are on-par with IBT (≈0%–1.46%)
- FineIBT outperforms Clang-CFI
## SPEC CPU 2017

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>IBT</th>
<th>FineIBT</th>
<th>Clang-CFI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>EH</td>
<td>EH</td>
</tr>
<tr>
<td>600.perlbench</td>
<td>1.46%</td>
<td>1.51%</td>
<td>≈0%</td>
</tr>
<tr>
<td>602.gcc</td>
<td>0.03%</td>
<td>0.12%</td>
<td>0.05%</td>
</tr>
<tr>
<td>605.mcf</td>
<td>0.51%</td>
<td>3.67%</td>
<td>1.94%</td>
</tr>
<tr>
<td>620.omnetpp</td>
<td>≈0%</td>
<td>0.91%</td>
<td>≈0%</td>
</tr>
<tr>
<td>623.xalancbmk</td>
<td>≈0%</td>
<td>0.99%</td>
<td>≈0%</td>
</tr>
<tr>
<td>625.x264</td>
<td>0.05%</td>
<td>0.29%</td>
<td>≈0%</td>
</tr>
<tr>
<td>631.deepsjeng</td>
<td>0.03%</td>
<td>0.17%</td>
<td>≈0%</td>
</tr>
<tr>
<td>641.leela</td>
<td>≈0%</td>
<td>≈0%</td>
<td>≈0%</td>
</tr>
<tr>
<td>657.xz</td>
<td>≈0%</td>
<td>≈0%</td>
<td>≈0%</td>
</tr>
</tbody>
</table>

- FineIBT exhibited ≈0%–1.94% slowdown
- Results are on-par with IBT (≈0%–1.46%)
- FineIBT outperforms Clang-CFI
## SPEC CPU 2017

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>CB</th>
<th>FineIBT</th>
<th>Clang-CFI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IBT</td>
<td>EH</td>
<td>EH</td>
</tr>
<tr>
<td>600.perlbench</td>
<td>1.46%</td>
<td>1.51%</td>
<td>≈0%</td>
</tr>
<tr>
<td>602.gcc</td>
<td>0.03%</td>
<td>0.12%</td>
<td>0.05%</td>
</tr>
<tr>
<td>605.mcf</td>
<td>0.51%</td>
<td>3.67%</td>
<td>1.94%</td>
</tr>
<tr>
<td>620.omnetpp</td>
<td>≈0%</td>
<td>0.91%</td>
<td>≈0%</td>
</tr>
<tr>
<td>623.xalancbmk</td>
<td>≈0%</td>
<td>0.99%</td>
<td>≈0%</td>
</tr>
<tr>
<td>625.x264</td>
<td>0.05%</td>
<td>0.29%</td>
<td>≈0%</td>
</tr>
<tr>
<td>631.deepsjeng</td>
<td>0.03%</td>
<td>0.17%</td>
<td>≈0%</td>
</tr>
<tr>
<td>641.leela</td>
<td>≈0%</td>
<td>≈0%</td>
<td>≈0%</td>
</tr>
<tr>
<td>657.xz</td>
<td>≈0%</td>
<td>≈0%</td>
<td>≈0%</td>
</tr>
</tbody>
</table>

- FineIBT exhibited ≈0%–1.94% slowdown
- Results are on-par with IBT (≈0%–1.46%)
- FineIBT outperforms Clang-CFI

### Real-world Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>IBT</th>
<th>FineIBT (EH)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nginx (1KB)</td>
<td>≈0%</td>
<td>≈0%</td>
</tr>
<tr>
<td>Nginx (100KB)</td>
<td>0.77%</td>
<td>1.92%</td>
</tr>
<tr>
<td>Nginx (1MB)</td>
<td>≈0%</td>
<td>0.11%</td>
</tr>
<tr>
<td>Redis (GET)</td>
<td>0.39%</td>
<td>1.17%</td>
</tr>
<tr>
<td>Redis (SET)</td>
<td>0.39%</td>
<td>1.17%</td>
</tr>
<tr>
<td>MariaDB</td>
<td>0.55%</td>
<td>0.60%</td>
</tr>
<tr>
<td>SQLite</td>
<td>≈0%</td>
<td>0.36%</td>
</tr>
</tbody>
</table>

- FineIBT exhibited ≈0%–1.92% tput/runtime slowdown
- Real-world results confirmed those of SPEC CPU 2017

agaidis@cs.brown.edu  (Brown University)
**NOPout**

- Reduced valid targets by up to 17291
- Additional memory required ranges between 184KB–7.84MB
**NOPout**

- Reduced valid targets by up to 17291
- Additional memory required ranges between 184KB–7.84MB

**ConFIRM**

- Passed 17/18 compatibility/security tests in ConFIRM
- Only failed JIT because JIT engine was not FineIBT-aware
Takeaways

- CFI enforcement mechanism that is performant, effective, and compatible
- Policy agnostic: supports coarse- and fine-grain policies
- Run-time target refinement via NOPout library
- A version of FineIBT was recently added to Linux kernel v6.2

Availability: https://gitlab.com/brown-ssl/fineibt
Backup Slides
Programs written in memory unsafe languages (e.g., C/C++, ASM) are prone to memory errors.

**Example**

```c
int main(int argc, char *argv[])
{
    int (*fptr)(void) = &foo;
    char arr[10];
    ...
    strcpy(arr, argv[1]);
    ...
    fptr();
    ...
}
```

**Repercussions**
- Attackers can corrupt memory to perform control-flow hijacking.
- Control-flow hijacking can be escalated to achieve arbitrary code execution.
CFI schemes can be classified according to 4 properties:

- Effectiveness
- Coverage
- Compatibility
- Target Domain
CFI schemes can be classified according to 4 properties:

- **Effectiveness**: How many (code) locations an indirect branch can target.
  - **Fine-grain** schemes refine the set of allowed targets.
  - **Coarse-grain** schemes are overly permissive.

- **Coverage**: Fewer Locations → More Locations
  - Fine-grain
  - Coarse-grain
Control-flow Integrity Scheme Classifications

- CFI schemes can be classified according to 4 properties:
  - Effectiveness
  - Coverage
  - Compatibility
  - Target Domain

**Coverage**
- The type of control-flow transfers covered
  - **Forward edges**
    - *e.g.*, indirect calls/jumps
  - **Backward edges**
    - *e.g.*, returns
Control-flow Integrity Scheme Classifications

CFI schemes can be classified according to 4 properties:

- Effectiveness
- Coverage
- Compatibility
- Target Domain

Compatibility

- What the scheme requires
  - Source code provides useful information
  - Binary data avoids recompilation
  - Different languages require different considerations
Control-flow Integrity Scheme Classifications

CFI schemes can be classified according to 4 properties:

- Effectiveness
- Coverage
- Compatibility
- Target Domain

Target Domain
- Where the scheme is applicable
  - User space
  - Kernel space
  - Embedded systems
## ConFIRM Results (Full)

<table>
<thead>
<tr>
<th>Test</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>callback</td>
<td>✔</td>
<td>Callbacks support</td>
</tr>
<tr>
<td>code_coop</td>
<td>✔</td>
<td>COOP attack resilience</td>
</tr>
<tr>
<td>convention</td>
<td>✔</td>
<td>Different x86 calling conv. support</td>
</tr>
<tr>
<td>cppeh</td>
<td>✔</td>
<td>C++ exception handling support</td>
</tr>
<tr>
<td>data_symbl</td>
<td>✔</td>
<td>Import/export data sym. handling</td>
</tr>
<tr>
<td>fptr</td>
<td>✔</td>
<td>Indirect function call support</td>
</tr>
<tr>
<td>jit</td>
<td>❌</td>
<td>Runtime-generated code support</td>
</tr>
<tr>
<td>load_time_dynlnk</td>
<td>✔</td>
<td>Load-time function resolution</td>
</tr>
<tr>
<td>mem</td>
<td>✔</td>
<td>Memory mgmt. API support</td>
</tr>
<tr>
<td>multithreading</td>
<td>✔</td>
<td>Concurrent thread exec. support</td>
</tr>
<tr>
<td>pic</td>
<td>✔</td>
<td>PIC/PIE support</td>
</tr>
<tr>
<td>ret</td>
<td>✔</td>
<td>Return-address validation</td>
</tr>
<tr>
<td>run_time_dynlnk</td>
<td>✔</td>
<td>Run-time function resolution</td>
</tr>
<tr>
<td>signal</td>
<td>✔</td>
<td>Signal handling support</td>
</tr>
<tr>
<td>switch</td>
<td>✔</td>
<td>switch-based CF support</td>
</tr>
<tr>
<td>tail_call</td>
<td>✔</td>
<td>Tail-call optimizations</td>
</tr>
<tr>
<td>unmatched_pair</td>
<td>✔</td>
<td>Unmatched call/ret pairs</td>
</tr>
<tr>
<td>vtbl_call</td>
<td>✔</td>
<td>Virtual function support</td>
</tr>
</tbody>
</table>
### NOPout Results (Full)

#### NOPout

<table>
<thead>
<tr>
<th>Application</th>
<th>AT-elided</th>
<th>Pages</th>
<th>KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>redis-server</td>
<td>858 (19.05%)</td>
<td>206</td>
<td>844</td>
</tr>
<tr>
<td>sqlite</td>
<td>896 (34.07%)</td>
<td>165</td>
<td>676</td>
</tr>
<tr>
<td>nginx</td>
<td>2873 (19.33%)</td>
<td>606</td>
<td>2482</td>
</tr>
<tr>
<td>mariadb</td>
<td>17291 (44.98%)</td>
<td>1915</td>
<td>7844</td>
</tr>
<tr>
<td>600.perlbench</td>
<td>942 (35.19%)</td>
<td>246</td>
<td>1008</td>
</tr>
<tr>
<td>602.gcc</td>
<td>3646 (67.71%)</td>
<td>665</td>
<td>2724</td>
</tr>
<tr>
<td>605.mcf</td>
<td>83 (4.57%)</td>
<td>45</td>
<td>184</td>
</tr>
<tr>
<td>620.omnetpp</td>
<td>5623 (71.03%)</td>
<td>405</td>
<td>1659</td>
</tr>
<tr>
<td>623.xalanckmk</td>
<td>8023 (77.78%)</td>
<td>644</td>
<td>2638</td>
</tr>
<tr>
<td>625.x264</td>
<td>302 (14.82%)</td>
<td>76</td>
<td>311</td>
</tr>
<tr>
<td>631.deepsjeng</td>
<td>1690 (42.45%)</td>
<td>189</td>
<td>774</td>
</tr>
<tr>
<td>641.leela</td>
<td>1722 (42.91%)</td>
<td>194</td>
<td>795</td>
</tr>
<tr>
<td>657.xz</td>
<td>176 (9.17%)</td>
<td>59</td>
<td>242</td>
</tr>
</tbody>
</table>

- Reduces valid targets by up to 17291
- Additional memory required ranges between 184KB–7.84MB
Clang-CFI Instrumentation

0x401140 <func>:
...
40116a: mov $0x401250,%ecx
40116f: mov %rax,%rdx
401172: sub %rcx,%rdx
401175: rol $0x3d,%rdx
401179: cmp $0x2,%rdx
40117d: jae 4011c5
40117f: callq *%rax
...
4011a3: callq 4011d0 <f0.cfi>
...
4011c5: ud2
...
0x4011d0 <f0.cfi>:
...
0x4011f0 <f1.cfi>:
...
0x401250 <f0>:
401250: jmpq 4011d0 <f0.cfi>
401255: int3
401256: int3
401257: int3
0x401258 <f1>:
401258: jmpq 4011f0 <f1.cfi>
40125d: int3
40125e: int3
40125f: int3
...
Non-IBT PLT

1  PLT0: pushq  GOT+8(%rip)  /* GOT[1] */
2       jmp       *GOT+16(%rip)  /* GOT[2] */
3       nopl      0x0(%rax)    /* PAD */
4  ...
5  PLT3: jmp  *fsym3@GOT(%rip)  /* GOT[5] */
6       pushq    $0x2
7       jmp      PLT0
8  PLT4: jmp  *fsym4@GOT(%rip)  /* GOT[6] */
9       pushq    $0x3
10      jmp      PLT0
11  ...
12  PLTn: jmp  *fsymn@GOT(%rip)  /* GOT[n+2] */
13      pushq    $0xn-1
14      jmp      PLT0
IBT PLT

1 PLT0: pushq GOT+8(%rip)  /* GOT[1] */
2          jmp       *GOT+16(%rip)  /* GOT[2] */
3          nopl      0x0(%rax)    /* PAD */
4          ...
5 PLT3:   endbr64
6          pushq $0x2
7          jmp       PLT0
8          xchg     %ax,%ax     /* PAD */
9 PLT4:   endbr64
10         pushq $0x3
11         jmp       PLT0
12         xchg     %ax,%ax     /* PAD */
13         ...
14 PLTn:  endbr64
15         pushq $0xn-1
16         jmp       PLT0
17         xchg     %ax,%ax     /* PAD */
18         ...
19         ...
20 SPLT3: endbr64
21         jmp       *fsym3@GOT(%rip)  /* GOT[5] */
22         nopw      0x0(%rax,%rax,1) /* PAD */
23 SPLT4: endbr64
24         jmp       *fsym4@GOT(%rip)  /* GOT[6] */
25         nopw      0x0(%rax,%rax,1) /* PAD */
26         ...
27 SPLTn: endbr64
28         jmp       *fsymn@GOT(%rip)  /* GOT[n+2] */
29         nopw      0x0(%rax,%rax,1) /* PAD */
FineIBT PLT

PLT0:  shl   $0x20, %rax
        or    $SID, %rax
        pushq GOT+8(%rip)       /* GOT[1] */
        jmp   *GOT+16(%rip)      /* GOT[2] */
        nopw  %cs:0x0(%rax,%rax,1) /* PAD */
...
PLT4:  endbr64
        cmp   $SID, %eax
        pushq $0x3
        xchg  %ax, %ax           /* PAD */
        je    PLT0
        hlt
        nopw  0x0(%rax,%rax,1)    /* PAD */
...
FPLT4: mov   $SID, %eax
        jmp   *fsym4@GOT(%rip)    /* GOT[6] */
        nopl  0x0(%rax,%rax,1)    /* PAD */
...
ATFPLT4: endbr64
        sub   $SID, %eax
        je    FPLT4
        hlt
        data16 nopw %cs:0x0(%rax,%rax,1) /*PAD*/
        nopl  (%rax)              /* PAD */
main:                        /* caller */
...

movz w9, #0x3a, lsl #16    /* SID = 0x3a0000 */
blr x0                    /* x0 = &func */
...
bl func_entry
...

.func_finebti_coldpath:
...
/* arg0, ..., argn */
bl __finebti_chk_fail@PLT

func:                      /* callee */

bti c

subs w9, w9, #0x3a0, lsl #12 /* SID=0x3a0000 */
bne .func_finebti_coldpath

func_entry:
...
