Binary Code Continent: Finer-Grained Control Flow Integrity for Stripped Binaries

Minghua Wang\textsuperscript{1,2,4}, Heng Yin\textsuperscript{2}, Abhishek Vasisht Bhaskar\textsuperscript{2}, Purui Su\textsuperscript{1,3}, Dengguo Feng\textsuperscript{1}

1. Chinese Academy of Sciences
2. Syracuse University
3. State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
4. University of Chinese Academy of Sciences
Control Flow Integrity

- It plays an important role in mitigating control flow hijacking attacks.
- It forces control flow transfers in the program to follow the policy represented by the CFG.
- Source code based solutions
  - Fine-grained CFI policy
- Binary-only solutions
  - Coarse-grained policy

Advanced ROP attacks can be launched by leveraging special gadgets

Propose a finer-grained control flow integrity for binaries
**Binary-Only Coarse-Gained CFI**

- A return is allowed to reach any callsite

```
main:
  ID_FUNC
  movl bar, %eax
  check_ID_FUNC
  call *%eax
  ID_RET
  call foo
  ID_RET
  ret

foo:
  ID_FUNC
  movl $1, %ebx
  check_ID_RET
  ret

bar:
  ID_FUNC
  movl foo, %eax
  check_ID_FUNC
  call *%eax
  ID_RET
  check_ID_RET
  ret
```
Bypass BinCFI/CCFIR

Utilize call-preceded gadgets to achieve ROP attacks


Challenges

- Coarse-Grained CFI: A function is allowed to return to any callsite
- After code duplication and instrumentation:
  - A directly called function will only reach direct callsites
  - An indirectly called function will only reach indirect callsites
Our Approach

- Duplicate code
- Construct mutually exclusive code continents, which imply the control flow dependency in the binary
- Classify indirect transfers into Intra-Continent and Inter-Continent transfers, and propose Intra-Continent and Inter-Continent CFI policies
- Enforce CFI policies through binary rewriting
Binary Code Continent

Code Duplication

- ICF: Indirect Called Function
  - Constants which imply entry points of functions
- DCF: Direct Called Function
  - Operands of direct callsites

ICFs

- Duplication

DCFs

- Only called by indirect calls
- Only called by direct calls
Binary Code Continent

### Code Duplication - Example

<table>
<thead>
<tr>
<th>main:</th>
<th>foo:</th>
<th>bar:</th>
<th>qux:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: movl bar, %eax</td>
<td>5: movl qux, %eax</td>
<td>8: movl foo, %eax</td>
<td>13: movl $0, %eax</td>
</tr>
<tr>
<td>3: call foo</td>
<td>7: ret</td>
<td>10: movl $1, %eax</td>
<td></td>
</tr>
<tr>
<td>4: ret</td>
<td></td>
<td>11: call foo</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>12: ret</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>foo’:</th>
</tr>
</thead>
<tbody>
<tr>
<td>5*: movl qux, %eax</td>
</tr>
<tr>
<td>6*: call *%eax</td>
</tr>
<tr>
<td>7*: ret</td>
</tr>
</tbody>
</table>

Original Functions

Duplicate Functions

ICF = { foo, qux, bar }
DCF = { foo }
Binary Code Continent

- Describe the control flow dependency in the binary
- Code Continent Construction
  - Construct from SuperCFGs of functions
  - Merge SuperCFGs based on common edges
  - Generate mutually exclusive continents
Binary Code Continent

Code Continent – Example

<table>
<thead>
<tr>
<th></th>
<th>main:</th>
<th>foo:</th>
<th>bar:</th>
<th>qux:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>movl bar%eax</td>
<td>5: movl qux%eax</td>
<td>8: movl foo%eax</td>
<td>13: movl $0, %eax</td>
</tr>
<tr>
<td>3:</td>
<td>call foo</td>
<td>7: ret</td>
<td>10: movl $1, %eax</td>
<td></td>
</tr>
<tr>
<td>4:</td>
<td>ret</td>
<td></td>
<td>11: call foo</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12: ret</td>
<td></td>
</tr>
</tbody>
</table>

Original Functions

<table>
<thead>
<tr>
<th></th>
<th>foo*:</th>
</tr>
</thead>
<tbody>
<tr>
<td>5*:</td>
<td>movl qux, %eax</td>
</tr>
<tr>
<td>6*:</td>
<td>call *%eax</td>
</tr>
<tr>
<td>7*:</td>
<td>ret</td>
</tr>
</tbody>
</table>

Duplicate Functions

Binary Code Continent

CC1

CC2

CC3
CFI Policy

- Continent Node Classification
- CFI Policy
  - Intra-Continent CFI Policy
  - Inter-Continent CFI Policy
CFI Policy

Continent Node Classification

- **Root node**: Entry points of ICFs
- **Boarder node**: Indirect transfers of which targets cannot be statically determined
- **Inner node**: Nodes except root nodes and boarder nodes

### Original Functions
- **main**: 1: movl bar%eax 2: call *%eax 3: call foo 4: ret
- **foo**: 5: movl qux%eax 6: call *%eax 7: ret
- **bar**: 8: movl foo%eax 9: call *%eax 10: movl $1, %eax 11: call foo 12: ret
- **qux**: 13: movl $0, %eax 14: ret

### Duplicated Functions
- **foo**: 5*: movl qux, %eax 6*: call *%eax 7*: ret

#### Continent Node Classification

- **Root Node**: Entry points of ICFs
- **Boarder Node**: Indirect transfers of which targets cannot be statically determined
- **Inner Node**: Nodes except root nodes and boarder nodes
CFI Policy

Intra-Continent CFI Policy

- This policy constrains inner nodes whose targets can be statically determined.
- The targets will be always present in current continent, and can be determined by the SuperCFG.

<table>
<thead>
<tr>
<th>main:</th>
<th>foo:</th>
<th>bar:</th>
<th>qux:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: movl bar, %eax</td>
<td>5: movl qux, %eax</td>
<td>8: movl foo, %eax</td>
<td>13: movl $0, %eax</td>
</tr>
<tr>
<td>3: call foo</td>
<td>7: ret</td>
<td>10: movl $1, %eax</td>
<td></td>
</tr>
<tr>
<td>4: ret</td>
<td></td>
<td>11: call foo</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>12: ret</td>
<td></td>
</tr>
</tbody>
</table>

Original Functions

<table>
<thead>
<tr>
<th>foo*:</th>
</tr>
</thead>
<tbody>
<tr>
<td>5*: movl qux, %eax</td>
</tr>
<tr>
<td>6*: call *%eax</td>
</tr>
<tr>
<td>7*: ret</td>
</tr>
</tbody>
</table>

Duplicated Functions

This policy constrains inner nodes whose targets can be statically determined. The targets will be always present in current continent, and can be determined by the SuperCFG.
This policy constrains the **border nodes**:

- Indirect call nodes can only reach all the root nodes
- Indirect ret nodes can only reach indirect call nodes
- Indirect jmp nodes can only reach all the root nodes and call nodes

---

**Original Functions**

<table>
<thead>
<tr>
<th>main:</th>
<th>foo:</th>
<th>bar:</th>
<th>qux:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: movl bar, %eax</td>
<td>5: movl qux, %eax</td>
<td>8: movl foo, %eax</td>
<td>13: movl $0, %eax</td>
</tr>
<tr>
<td>3: call foo</td>
<td>7: ret</td>
<td>10: movl $1, %eax</td>
<td></td>
</tr>
<tr>
<td>4: ret</td>
<td></td>
<td>11: call foo</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12: ret</td>
</tr>
</tbody>
</table>

**Duplicated Functions**

<table>
<thead>
<tr>
<th>foo*:</th>
</tr>
</thead>
<tbody>
<tr>
<td>5*: movl qux, %eax</td>
</tr>
<tr>
<td>6*: call *%eax</td>
</tr>
<tr>
<td>7*: ret</td>
</tr>
</tbody>
</table>

---

**Inter-Continent CFI Policy**

- **CC1**
- **CC2**
- **CC3**

---

**Graph representation**

1. Root Node
2. Border Node
3. Inner Node
CFI Enforcement

- **Instrumentation**
- **Address Checking Routine**
  - Check whether the target is legitimate
  - Each kind of indirect transfers has its own address checking routine
  - In call’s routine, the duplicated function will be actually executed if an ICF is called

```assembly
Instrumentation to indirect calls
```
```
call *%eax  movl %eax, %gs:0x40
jmp addr_trans_call
```
```
Instrumentation to indirect jmps
```
```
jmp *%eax  movl %eax, %gs:0x40
jmp addr_trans_jmp
```
```
Instrumentation to indirect returns
```
```
ret  movl (%esp), %gs:0x40
jmp addr_trans_ret
```
```
Instrumentation to direct returns
```
```
_addr: ret  _addr: prefetchnta start
prefetchnta size
movl _addr, %gs:0x50
movl %(%esp), %gs:0x40
jmp addr_trans_dret
```
Evaluation

- Implementation
  - BinCC (Binary Code Continent)
- Evaluate BinCC from four respects
  - Code Duplication Evaluation
  - Indirect Transfer Targets Deduction
  - ROP Attack Evaluation
  - Performance
Evaluation

Code Duplication Evaluation

Percentage of Duplicated Functions

Percentage of Duplicated Instructions
Evaluation

Indirect Transfer Targets Reduction

- **AIR (Average Indirect target Reduction)**, proposed by BinCFI:

\[
AIR = \frac{1}{N} \sum_{j=1}^{N} \left(1 - \frac{|T_j|}{S}\right)
\]

- \(T_j\) : the legitimate targets of indirect transfer \(i_j\)
- \(S\) : size of code section
- \(N\) : the number of all the indirect transfers

\textbf{BinCFI : 98.86\%, BinCC: 99.54\%}

- A new metric: **RAIR (Relative AIR)**, to evaluate extent to which BinCC refines the legitimate targets compared to BinCFI:

\[
RAIR = \frac{1}{N} \sum_{j=1}^{N} \left(1 - \frac{|T_j|}{|T_j'|}\right)
\]

- \(T_j\) : the legitimate targets of \(i_j\) in BinCFI
- \(T_j'\) : the legitimate targets of \(i_j\) in BinCC
- \(N\) : the number of all the indirect transfers

BinCC reduced legitimate targets by \textbf{81.34\%} from BinCFI
Evaluation

ROP Attacks Evaluation

GS (Gadget Survivability):
Suppose in a CFI enforced binary a return instruction was fully controlled by attackers, how likely this return could be used to reach a call preceded gadget in this binary

\[
GS = \frac{1}{|R|} \sum_{i=0}^{|R|} \frac{|C_i|}{|C|}
\]

- \(C_i\): the legitimate targets of \(\text{ret}_i\) in a CFI enforced binary
- \(R\): all the returns in the binary
- \(C\): all the call-preceded gadgets in the binary

\textbf{BinCFI: 100%}, \textbf{BinCC: 0.70%}
Evaluation

Performance

- **Space Overhead**
  - 125% of the original binary size
  - 14% lower than the size of BinCFI-enforced binaries

- **Runtime Overhead**
Conclusion

- Propose binary code continent as a way to enforce finer-grained CFI policy in x86 stripped ELF binaries

- Classify indirect transfers into Intra-Continent and Inter-Continent transfers, and apply separate, strict policies to constrain them

- Significantly reduce the legitimate transfer targets by 81.34% from BinCFI

- Significantly increase the difficulty for adversaries to bypass CFI to launch sophisticated ROP attacks
Thanks !!

Q & A