CS 6V81-05

Control-Flow Integrity Principles, Implementations, and Applications

Sureshbabu Murugesan

Department of Computer Science
University of Texas at Dallas

February 29th, 2012
1 Overview
   • Introduction

2 CFI by instrumentation
   • CFI by instrumentation
   • Example of CFI instrumentation
   • Assembly code Example
   • Assumptions

3 CFI implementation

4 Conclusion
1. Overview
   - Introduction

2. CFI by instrumentation
   - CFI by instrumentation
   - Example of CFI instrumentation
   - Assembly code Example
   - Assumptions

3. CFI implementation

4. Conclusion
Software attacks often build on exploits that subvert machine-code execution.

Control-flow integrity (CFI), can prevent such attacks from arbitrarily controlling program behavior.

CFI enforcement is practical and simple.

It is compatible with existing software and can be done efficiently using software rewriting in commodity systems.
Existing Vulnerability mitigations

**Existing Implementations**

- Many ingenious vulnerability mitigations have been proposed for defending against these attacks; these include stack canaries, runtime elimination of buffer overflows, randomization and artificial heterogeneity.

- Some of these mitigations are widely used, while others may be impractical because of hardware limitations and most of these techniques got circumvented by attackers.

- Mitigation techniques should be applicable to existing code (preferably even to legacy binaries) and incur low overhead.
CFI

The Control-Flow Graph (CFG) is the key here. Which can be obtained by source code analysis or binary code analysis.

Techniques

- CFGs derived from static binary analysis is discussed here.
- CFI is enforced by instrumentation.
- Control-flow transfers is enforced with a dynamic check
1. Overview
   - Introduction

2. CFI by instrumentation
   - CFI by instrumentation
   - Example of CFI instrumentation
   - Assembly code Example
   - Assumptions

3. CFI implementation

4. Conclusion
1. Overview
   - Introduction

2. CFI by instrumentation
   - CFI by instrumentation
   - Example of CFI instrumentation
   - Assembly code Example
   - Assumptions

3. CFI implementation

4. Conclusion
Techniques of CFI instrumentation

1. Machine-code rewriting presents an apparently straightforward strategy for implementing dynamic checks.

2. Machine code rewriting has many problems and it is not easy. Ex. Rewritten machine have different memory layout and all the addresses has to be changed.

3. CFI may be enforced by dynamic checks that compare the target address of each computed control-flow transfer to a set of allowed destination addresses.

4. Since the set of allowed destination addresses may be large, this naive strategy will lead to unacceptable overhead.
CFI by hardware model

Machine-code instructions

- label ID - label a CFG block with id 'ID'
- call ID,DST - jump to address contained in register DST only if the destination contains label ID
- Ret ID - Return to label with id 'ID'
CFI by instrumentation

CFI enforcement provides protection even against powerful adversaries that have full control over the entire data memory of the executing program.

CFI can be done in several ways. This paper rely on a combination of lightweight static verification and machine-code rewriting that instruments software with runtime checks.

The runtime checks dynamically ensure that control flow remains within a given CFG.

Exploits within the bounds of the allowed CFG are not prevented. E.g. certain exploits that rely on incorrect argument-string parsing to cause the improper launch of a dangerous executable.

CFI techniques described in paper either directly or indirectly, constrain control flow.
Outline

1. Overview
   - Introduction

2. CFI by instrumentation
   - CFI by instrumentation
   - Example of CFI instrumentation
   - Assembly code Example
   - Assumptions

3. CFI implementation

4. Conclusion
bool lt(int x, int y) {
    return x < y;
}
bool gt(int x, int y) {
    return x > y;
}

sort2(int a[], int b[], int len) {
    sort(a, len, lt);
    sort(b, len, gt);
}
Outline

1 Overview
   • Introduction

2 CFI by instrumentation
   • CFI by instrumentation
   • Example of CFI instrumentation
   • Assembly code Example
   • Assumptions

3 CFI implementation

4 Conclusion
Assembly code Example

<table>
<thead>
<tr>
<th>Opcode bytes</th>
<th>Source Instructions</th>
<th>Destination Opcode bytes</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF E1</td>
<td>jmp ecx</td>
<td>8B 44 24 04</td>
<td>mov eax, [esp+4]</td>
</tr>
<tr>
<td></td>
<td>; computed jump</td>
<td>...</td>
<td>; dst</td>
</tr>
</tbody>
</table>

can be instrumented as (a):

<table>
<thead>
<tr>
<th>Opcode bytes</th>
<th>Source Instructions</th>
<th>Destination Opcode bytes</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>81 39 78 56 34 12</td>
<td>cmp [ecx], 12345678h</td>
<td>78 56 34 12</td>
<td>; data 12345678h</td>
</tr>
<tr>
<td>75 13</td>
<td>jne error_label</td>
<td>8B 44 24 04</td>
<td>mov eax, [esp+4]</td>
</tr>
<tr>
<td>8D 49 04</td>
<td>lea ecx, [ecx+4]</td>
<td>...</td>
<td>; dst</td>
</tr>
<tr>
<td>FF E1</td>
<td>jmp ecx</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

or, alternatively, instrumented as (b):

<table>
<thead>
<tr>
<th>Opcode bytes</th>
<th>Source Instructions</th>
<th>Destination Opcode bytes</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>B8 77 56 34 12</td>
<td>mov eax, 12345677h</td>
<td>3E 0F 18 05</td>
<td>prefetchnta</td>
</tr>
<tr>
<td>40</td>
<td>inc eax</td>
<td>78 56 34 12</td>
<td>[12345678h]</td>
</tr>
<tr>
<td>39 41 04</td>
<td>cmp [ecx+4], eax</td>
<td>8B 44 24 04</td>
<td>mov eax, [esp+4]</td>
</tr>
<tr>
<td>75 13</td>
<td>jne error_label</td>
<td>...</td>
<td>; dst</td>
</tr>
<tr>
<td>FF E1</td>
<td>jmp ecx</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2: Example CFI instrumentations of a source x86 instruction and one of its destinations.
Outline

1. Overview
   - Introduction

2. CFI by instrumentation
   - CFI by instrumentation
   - Example of CFI instrumentation
   - Assembly code Example
   - Assumptions

3. CFI implementation

4. Conclusion
Assumptions

- **Unique IDs** - Don't conflict with opcodes. Done by making ID 32 bit number.
- **Non-Writable Code** - Code segment must be write protected.
- **Non-Executable Data** - Data segment is not executable.
- The assumptions can be somewhat problematic in the presence of self-modifying code, runtime code generation, and the unanticipated dynamic loading of code.
- Fortunately, most software is rather static - either statically linked or with a statically declared set of dynamic libraries.
Outline

1. Overview
   - Introduction

2. CFI by instrumentation
   - CFI by instrumentation
   - Example of CFI instrumentation
   - Assembly code Example
   - Assumptions

3. CFI implementation

4. Conclusion
## Assembly code instrumentation

<table>
<thead>
<tr>
<th>Opcode bytes</th>
<th>Instructions</th>
<th>Function Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF 53 08</td>
<td>call [ebx+8]</td>
<td>; call fptr</td>
</tr>
</tbody>
</table>

are instrumented using `prefetchnta` destination IDs, to become

<table>
<thead>
<tr>
<th>Opcode bytes</th>
<th>Instructions</th>
<th>Function Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2 10 00</td>
<td>ret 10h</td>
<td>; return</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Opcode bytes</th>
<th>Instructions</th>
<th>Function Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>8B 43 08</td>
<td>mov eax, [ebx+8]</td>
<td>; load fptr</td>
</tr>
<tr>
<td>3E 81 78 04 78 56 34 12</td>
<td>cmp [eax+4], 12345678h</td>
<td>; comp w/ID</td>
</tr>
<tr>
<td>75 13</td>
<td>jne error_label</td>
<td>; if != fail</td>
</tr>
<tr>
<td>FF D0</td>
<td>call eax</td>
<td>; call fptr</td>
</tr>
<tr>
<td>3E 0F 18 05 DD CC BB AA</td>
<td>prefetchnta [AABBCCDDh]</td>
<td>; label ID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Opcode bytes</th>
<th>Instructions</th>
<th>Function Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>8B 0C 24</td>
<td>mov ecx, [esp]</td>
<td>; load ret</td>
</tr>
<tr>
<td>83 C4 14</td>
<td>add esp, 14h</td>
<td>; pop 20</td>
</tr>
<tr>
<td>3E 81 79 04</td>
<td>cmp [ecx+4],</td>
<td>; compare</td>
</tr>
<tr>
<td>DD CC BB AA</td>
<td>AABBCCDDh</td>
<td>; w/ID</td>
</tr>
<tr>
<td>75 13</td>
<td>jne error_label</td>
<td>; if!=fail</td>
</tr>
<tr>
<td>FF E1</td>
<td>jmp ecx</td>
<td>; jump ret</td>
</tr>
</tbody>
</table>
Figure 4: Execution overhead of inlined CFI enforcement on SPEC2000 benchmarks.
Outline

1. Overview
   - Introduction

2. CFI by instrumentation
   - CFI by instrumentation
   - Example of CFI instrumentation
   - Assembly code Example
   - Assumptions

3. CFI implementation

4. Conclusion
Conclusion

- Use of high level programming language implies that only certain control flow has to be executed during software execution.
- The absence of runtime control-flow guarantees has a pervasive impact on all software analysis, processing, and optimization and it also enables many of today’s exploits.
- CFI instrumentation aims to change that by embedding runtime checks within software executable too prevent from many exploits.
- Inlined CFI enforcement is practical on modern processors, is compatible with most existing software, and has little performance overhead.
- CFI is simple, verifiable, and amenable to formal analysis, yielding strong guarantees even in the presence of a powerful adversary.