Control Flow Integrity for COTS Binaries

Mingwei Zhang and R. Sekar
Stony Brook University

---

Summarized by Navid Emamdoost
University of Minnesota
Outline

• Background
  – Control Flow attacks
  – Control Flow Integrity

• Control Flow Integrity for COTS Binaries
Control Flow

• The order of instruction execution
• A subset of possible paths are intended by program
• An attacker can change this order due to
  – Programming mistakes
  – Insufficient security primitives provided by PL
  – Intrinsic complexity of architecture
Control Flow attacks

• Code injection
  – Overflow a buffer on system stack
  – Overwrite the return address
  – Divert control to injected code
Control Flow attacks

• Return to Libc
  – Overflow a buffer on system stack
  – Overwrite the return address
  – Divert control to an existing module
    • system(/bin/sh)
Control Flow attacks

• Return Oriented Programming (ROP)
  – Overflow a buffer on system stack
  – Overwrite the return address
  – Divert control to start of gadget
    • inc eax; ret;
    • pop eax; ret;
Control Flow Integrity

• Protect program’s control flow integrity
  – Resist deviation from CFG
• Identify legal control transfer targets
• Prevent transfers to other targets
• Restrict program execution to the set of intended paths
Control Flow Integrity

- By Abadi et. al presented at 2005
- Computed control transfers are instrumented

<table>
<thead>
<tr>
<th>Opcode bytes</th>
<th>Source Instructions</th>
<th>Destination Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF E1</td>
<td>jmp ecx</td>
<td>mov eax, [esp+4] ; dst</td>
</tr>
<tr>
<td></td>
<td>; computed jump</td>
<td>...</td>
</tr>
<tr>
<td>81 39 78 56 34 12</td>
<td>cmp [ecx], 12345678h ; comp ID &amp; dst</td>
<td>78 56 34 12 ; data 12345678h ; ID</td>
</tr>
<tr>
<td>75 13</td>
<td>jne error_label</td>
<td>if != fail</td>
</tr>
<tr>
<td>8D 49 04</td>
<td>lea ecx, [ecx+4]</td>
<td>; skip ID at dst</td>
</tr>
<tr>
<td>FF E1</td>
<td>jmp ecx</td>
<td>; jump to dst</td>
</tr>
</tbody>
</table>

can be instrumented as (a):
CFI

- Unique IDs: the bit patterns chosen as IDs must not be present anywhere in the code memory except in IDs and ID-checks
- Non-Writable Code: It must not be possible for the program to modify code memory at runtime
- Non-Executable Data: It must not be possible for the program to execute data as if it were code
- One ID value for the start of functions and another ID value for valid destinations for function returns
CFI

- Is not vulnerable to information leakage attacks, unlike
  - Stack canary
  - ASLR
- Protect against existing code reuse
  - Return-to-libc
  - ROP
Control Flow Integrity for COTS Binaries

• Goal:
  – Enforce CFI on COTS binaries
    • There is no source-code
    • No assembly-level information
    • No relocation information (unlike ASLR on windows)
    • Like shared libraries
    • Operate with less information available
Control Flow Integrity for COTS Binaries

• Steps
  – Disassemble
    • Correctly identify instructions
  – ICF analysis
    • Provide missing information (instead of using relocation info)
  – Instrument the binary
    • Enforce CFI
Disassembly

• Linear
  – Start from the first instruction of the segment
  – Assume nest instruction starts from the end of previous one
  – Problem: gaps
    • Data
    • Instruction alignment
Disassembly

• Recursive
  – Depth-first approach
  – A set of entry points
  – Add target of each direct CF transfer to the set of EP
  – Continue linearly up to an unconditional CF transfer
  – Problem: can not indentify codes reachable via ICF
    • Available from relocation information
COTS Disassembly

- Combination of linear and recursive
- Use static analysis of ICF to identify gaps

Steps:
- Linearly disassemble entire binary
- Check for erroneous instructions
  - Invalid opcode
  - Direct CF transfer to outside of module
  - Direct CF transfer to the middle of another instruction
COTS Disassembly (cont’d)

• On an erroneous instruction
  – Move backward to reach a direct CF transfer
    • Mark as gap start
  – From ICF analysis find the first target after erroneous instruction
    • Mark as gap end
  – Repeat disassembly by avoiding gaps
ICF analysis

- **Code pointer constants (CK)**
  - consists of code addresses that are computed at compile-time.
- **Computed code addresses (CC)**
  - include code addresses that are computed at runtime.
- **Exception handling addresses (EH)**
  - include code addresses that are used to handle exceptions.
- **Exported symbol addresses (ES)**
  - include export function addresses.
- **Return addresses (RA)**
  - include the code addresses next of a call.
Code pointer constants (CK)

• In general, there is no way to distinguish a code pointer from other types of constants in code

• Every constant having properties
  – Be within the rage of code addresses
    • For shared libraries consider it as offset
    • Because there is no knowledge about base address at compile time
  – Is consistent with instruction boundaries
Computed code addresses (CC)

• Any arithmetic computation on pointers are possible in binary
• But they observed pointer arithmetic occurs just in jump tables
  – Switch case

• Properties of jump tables
  – Intra-function
  – Simple form: *(CE1+ Ind)+CE2
  – Within fixed sized window of instructions
    • 50 instructions
Computed code addresses (CC)

• Determine function boundaries
  – Exported functions

• Identify indirect jump and move backward to find the expression
  – CE1 and CE2 are constants

• Enumerate possible values of $Ind$
  – for every possible value if the result falls within the current region
Other code addresses

- Exception handling addresses (EH)
  - From ELF headers

- Exported symbol addresses (ES)
  - From ELF headers

- Return addresses (RA)
  - The address of instruction after the call
    - Computable after disassembly
CFI classes

• reloc-CFI
  – Types of ICF
    • Indirect Call
    • Indirect Jump
    • Return Address

• strict-CFI
  – Same as reloc-CFI
  – But uses static analysis instead of relocation info
  – Extensions for EH and Context switch

• bin-CFI
  – Has a new type of ICF: Program Linkage Table
bin-CFI

<table>
<thead>
<tr>
<th></th>
<th>Returns (RET), Indirect Jumps (IJ)</th>
<th>PLT targets, Indirect Calls (IC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return addresses (RA)</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Exception handling addresses (EH)</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Exported symbol addresses (ES)</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>Code pointer constants (CK)</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Computed code addresses (CC)</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Figure 2: Bin-CFI Property Definition
CFI Instrumentation

• After instrumenting the binary, new object file is generated
• The new object file is injected into ELF file
• Prepare new segment for execution
• Update Entry point
• Mark original code segments as un-executable
CFI Instrumentation

• New code is in different segment
  – Function pointers are invalid
• Keep a table for address translation
  <original address, new address>
• For each valid ICF target
• addr_trans: a trampoline code performing translation by a hash table
• If target is within current module
  – lookup the hash
  – If no entry found, an error is sent
• If not, use a global translation table loaded by ld.so
CFI Instrumentation

• Signals
  – Intercept \textit{signal} and \textit{sigaction} system calls
  – Store the handlers address
  – Update system calls arguments to point to a wrapper function
  – The wrapper performs redirection to instrumented code
Evaluation

- Disassembly

<table>
<thead>
<tr>
<th>Module</th>
<th>Package</th>
<th>Size</th>
<th># of Instructions</th>
<th># of Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>libxul.so</td>
<td>firefox-5.0</td>
<td>26M</td>
<td>4.3M</td>
<td>0</td>
</tr>
<tr>
<td>gimp-console-2.6</td>
<td>gimp-2.6.5</td>
<td>7.7M</td>
<td>385K</td>
<td>0</td>
</tr>
<tr>
<td>libc.so</td>
<td>glibc-2.13</td>
<td>8.1M</td>
<td>301K</td>
<td>0</td>
</tr>
<tr>
<td>libnss3.so</td>
<td>firefox-5.0</td>
<td>4.1M</td>
<td>235K</td>
<td>0</td>
</tr>
<tr>
<td>libmozsqlite3.so</td>
<td>firefox-5.0</td>
<td>1.8M</td>
<td>128K</td>
<td>0</td>
</tr>
<tr>
<td>libfreebl3.so</td>
<td>firefox-5.0</td>
<td>876K</td>
<td>66K</td>
<td>0</td>
</tr>
<tr>
<td>libsoftokn3.so</td>
<td>firefox-5.0</td>
<td>756K</td>
<td>50K</td>
<td>0</td>
</tr>
<tr>
<td>libnspr4.so</td>
<td>firefox-5.0</td>
<td>776K</td>
<td>41K</td>
<td>0</td>
</tr>
<tr>
<td>libssl3.so</td>
<td>firefox-5.0</td>
<td>864K</td>
<td>40K</td>
<td>0</td>
</tr>
<tr>
<td>libm.so</td>
<td>glibc-2.13</td>
<td>620K</td>
<td>35K</td>
<td>0</td>
</tr>
<tr>
<td>libnssdbm3.so</td>
<td>firefox-5.0</td>
<td>570K</td>
<td>34K</td>
<td>0</td>
</tr>
<tr>
<td>libsmime3.so</td>
<td>firefox-5.0</td>
<td>746K</td>
<td>30K</td>
<td>0</td>
</tr>
<tr>
<td>ld.so</td>
<td>glibc-2.13</td>
<td>694K</td>
<td>28K</td>
<td>0</td>
</tr>
<tr>
<td>gimpressionist</td>
<td>gimp-2.6.5</td>
<td>403K</td>
<td>21K</td>
<td>0</td>
</tr>
<tr>
<td>script-fu</td>
<td>gimp-2.6.5</td>
<td>410K</td>
<td>21K</td>
<td>0</td>
</tr>
<tr>
<td>libnssckbi.so</td>
<td>firefox-5.0</td>
<td>733K</td>
<td>19K</td>
<td>0</td>
</tr>
<tr>
<td>libtestcrasher.so</td>
<td>firefox-5.0</td>
<td>676K</td>
<td>17K</td>
<td>0</td>
</tr>
<tr>
<td>gfig</td>
<td>gimp-2.6.5</td>
<td>442K</td>
<td>17K</td>
<td>0</td>
</tr>
<tr>
<td>libpthread.so</td>
<td>glibc-2.13</td>
<td>666K</td>
<td>15K</td>
<td>0</td>
</tr>
<tr>
<td>libnsl.so</td>
<td>glibc-2.13</td>
<td>448K</td>
<td>15K</td>
<td>0</td>
</tr>
<tr>
<td>map-object</td>
<td>gimp-2.6.5</td>
<td>257K</td>
<td>15K</td>
<td>0</td>
</tr>
<tr>
<td>libresolv.so</td>
<td>glibc-2.13</td>
<td>275K</td>
<td>13K</td>
<td>0</td>
</tr>
<tr>
<td>libnssutil3.so</td>
<td>firefox-5.0</td>
<td>311K</td>
<td>13K</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>58M</td>
<td>5.84M</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 6: Disassembly Correctness
Evaluation

• CFI effectiveness:
  – Average Indirect target Reduction (AIR)
  – For $n$ ICF transfers, and $S$ initial targets for them

\[
\frac{1}{n} \sum_{j=1}^{n} \left( 1 - \frac{|T_j|}{S} \right)
\]
## Evaluation

<table>
<thead>
<tr>
<th>Name</th>
<th>Reloc CFI</th>
<th>Strict CFI</th>
<th>Bin CFI</th>
<th>Bundle CFI</th>
<th>Instr CFI</th>
</tr>
</thead>
<tbody>
<tr>
<td>perlbench</td>
<td>98.49%</td>
<td>98.44%</td>
<td>97.89%</td>
<td>95.41%</td>
<td>67.33%</td>
</tr>
<tr>
<td>bzip2</td>
<td>99.55%</td>
<td>99.49%</td>
<td>99.37%</td>
<td>95.65%</td>
<td>78.59%</td>
</tr>
<tr>
<td>gcc</td>
<td>98.73%</td>
<td>98.71%</td>
<td>98.34%</td>
<td>95.86%</td>
<td>80.63%</td>
</tr>
<tr>
<td>mcf</td>
<td>99.47%</td>
<td>99.37%</td>
<td>99.25%</td>
<td>95.91%</td>
<td>79.35%</td>
</tr>
<tr>
<td>gobmk</td>
<td>99.40%</td>
<td>99.40%</td>
<td>99.20%</td>
<td>97.75%</td>
<td>89.08%</td>
</tr>
<tr>
<td>hmmer</td>
<td>98.90%</td>
<td>98.87%</td>
<td>98.61%</td>
<td>95.85%</td>
<td>79.01%</td>
</tr>
<tr>
<td>sjeng</td>
<td>99.32%</td>
<td>99.30%</td>
<td>99.10%</td>
<td>96.22%</td>
<td>83.18%</td>
</tr>
<tr>
<td>libquantum</td>
<td>99.14%</td>
<td>99.07%</td>
<td>98.89%</td>
<td>95.96%</td>
<td>76.53%</td>
</tr>
<tr>
<td>h264ref</td>
<td>99.64%</td>
<td>99.60%</td>
<td>99.52%</td>
<td>96.25%</td>
<td>80.71%</td>
</tr>
<tr>
<td>omnetpp</td>
<td>98.26%</td>
<td>98.08%</td>
<td>98.89%</td>
<td>95.72%</td>
<td>82.03%</td>
</tr>
<tr>
<td>astar</td>
<td>99.18%</td>
<td>99.13%</td>
<td>98.95%</td>
<td>96.02%</td>
<td>78.00%</td>
</tr>
<tr>
<td>milc</td>
<td>98.89%</td>
<td>98.86%</td>
<td>98.65%</td>
<td>96.03%</td>
<td>79.74%</td>
</tr>
<tr>
<td>namd</td>
<td>99.65%</td>
<td>99.64%</td>
<td>99.59%</td>
<td>95.81%</td>
<td>76.37%</td>
</tr>
<tr>
<td>soplex</td>
<td>99.19%</td>
<td>99.10%</td>
<td>98.86%</td>
<td>95.50%</td>
<td>77.37%</td>
</tr>
<tr>
<td>povray</td>
<td>99.01%</td>
<td>98.99%</td>
<td>98.67%</td>
<td>95.87%</td>
<td>78.03%</td>
</tr>
<tr>
<td>lbm</td>
<td>99.60%</td>
<td>99.50%</td>
<td>99.46%</td>
<td>96.79%</td>
<td>80.92%</td>
</tr>
<tr>
<td>sphinx3</td>
<td>98.83%</td>
<td>98.80%</td>
<td>98.64%</td>
<td>96.06%</td>
<td>80.75%</td>
</tr>
<tr>
<td>average</td>
<td>99.13%</td>
<td>99.08%</td>
<td>98.86%</td>
<td>96.04%</td>
<td>79.27%</td>
</tr>
</tbody>
</table>

**Figure 8:** AIR metrics for SPEC CPU 2006.
Evaluation

• Gadget elimination

<table>
<thead>
<tr>
<th>Name</th>
<th>Reloc CFI</th>
<th>Strict CFI</th>
<th>Bin CFI</th>
<th>Instr CFI</th>
</tr>
</thead>
<tbody>
<tr>
<td>perlbench</td>
<td>96.62%</td>
<td>96.24%</td>
<td>93.23%</td>
<td>58.65%</td>
</tr>
<tr>
<td>bzip2</td>
<td>97.78%</td>
<td>95.56%</td>
<td>93.33%</td>
<td>44.44%</td>
</tr>
<tr>
<td>gcc</td>
<td>97.69%</td>
<td>97.69%</td>
<td>91.42%</td>
<td>66.67%</td>
</tr>
<tr>
<td>mcf</td>
<td>95.45%</td>
<td>90.91%</td>
<td>90.91%</td>
<td>36.36%</td>
</tr>
<tr>
<td>gobmk</td>
<td>98.84%</td>
<td>98.27%</td>
<td>97.69%</td>
<td>70.52%</td>
</tr>
<tr>
<td>hmmer</td>
<td>97.00%</td>
<td>96.00%</td>
<td>96.00%</td>
<td>58.00%</td>
</tr>
<tr>
<td>sjeng</td>
<td>92.75%</td>
<td>92.75%</td>
<td>91.30%</td>
<td>47.83%</td>
</tr>
<tr>
<td>libquantum</td>
<td>93.18%</td>
<td>90.91%</td>
<td>86.36%</td>
<td>40.91%</td>
</tr>
<tr>
<td>h264ref</td>
<td>98.26%</td>
<td>97.39%</td>
<td>96.52%</td>
<td>60.87%</td>
</tr>
<tr>
<td>omnetpp</td>
<td>97.12%</td>
<td>97.12%</td>
<td>93.42%</td>
<td>74.07%</td>
</tr>
<tr>
<td>astar</td>
<td>95.35%</td>
<td>93.02%</td>
<td>93.02%</td>
<td>46.51%</td>
</tr>
<tr>
<td>milc</td>
<td>95.77%</td>
<td>94.37%</td>
<td>90.14%</td>
<td>57.75%</td>
</tr>
<tr>
<td>namd</td>
<td>94.87%</td>
<td>92.31%</td>
<td>92.31%</td>
<td>53.85%</td>
</tr>
<tr>
<td>soplex</td>
<td>94.64%</td>
<td>93.75%</td>
<td>93.75%</td>
<td>54.46%</td>
</tr>
<tr>
<td>povray</td>
<td>96.75%</td>
<td>96.75%</td>
<td>95.45%</td>
<td>61.69%</td>
</tr>
<tr>
<td>lbm</td>
<td>94.12%</td>
<td>88.24%</td>
<td>88.24%</td>
<td>23.53%</td>
</tr>
<tr>
<td>sphinx3</td>
<td>95.00%</td>
<td>93.75%</td>
<td>92.50%</td>
<td>52.50%</td>
</tr>
</tbody>
</table>

Figure 10: Gadget elimination in different CFI implementation
Evaluation

• Performance overhead

Figure 11: SPEC CPU2006 Benchmark Performance
Evaluation

• Space overhead:
  – 139% increase in file size
  – 2.2% for resident memory use
Thank You