Protecting C++ Dynamic Dispatch Through VTable Interleaving

Benjamin Zanger
Technical University of Munich
Department of Informatics
Chair for IT Security

Munich, November 19, 2018
Outline

Dynamic Dispatch in C++
  Virtual Functions in C++

VTable Hijacking

Protecting Dynamic Dispatches
  VTable Ordering
  VTable Interleaving
  Multiple Inheritance
  Optimization

Benchmarks

Comparison with other Approaches
Virtual Functions in C++

class A {
    public:
    void f0() {}
    virtual void f1() {}
    virtual void f2() {}
    int int_in_A;
};
class B : public A {
    public:
    void f1() {} //override f1
    int int_in_B;
};
Multiple Inheritance

class A1 {
    public:
    void f0() {}
    virtual void f1() {}
    int int_in_A1;
};
class A2 {
    public:
    virtual void f2() {}
    int int_in_A2;
};
class B : public A1, public A2 {
    public:
    void f1() {} // override f1
    int int_in_B;
};
VTable Hijacking

- Exploiting memory corruption, for example use after free.
- VTables are stored in read only memory, vptr in writable memory.
- Changing vptr to take control over program flow.
- Either code injection or reuse attacks possible.

Assumptions made by the authors:
- Hacker capable of modifying the Heap.
- Registers are safe.
Protecting Dynamic Dispatches

- Most strategies use **Inline Reference Monitors** (IRMs) before dynamic dispatch calls.
- Example for semantic of IRMs:

\[ vptr \in \{0x08, 0x20\} \]  \hspace{1cm} (1)

⇒ Differences are in the implementation.
VTable ordering (OVT)

- Preorder traversal of the class hierarchy.
- Padding added, so that VTable addresses are $2^n$ Bytes aligned.
- Address point ranges are stored.
- Example:\n
  \[
  \begin{array}{c|c|c|c}
  \hline
  \text{Class} & \text{Start} & \text{End} & \text{Alignment} \\
  \hline
  A & 0x20 & 0x80 & 0x20 \\
  B & 0x40 & 0x60 & 0x20 \\
  C & 0x80 & 0x80 & 0x20 \\
  D & 0x60 & 0x60 & 0x20 \\
  \hline
  \end{array}
  \]

\[Bounov, Kici, and Lerner 2016.\]
VTable Ordering (OVT)

- Simple range check and alignment check before dispatch call.

Problems of VTable Ordering:
- Takes more memory than necessary because of padding.
- Especially an issue in systems with limited memory (embedded systems).

⇒ VTable Interleaving
VTable Interleaving (IVT)

- Interleaving of different VTables, by making them sparse, to save memory.
- Saving different functions offsets.
- Example²:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>D</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Artti</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x08</td>
<td></td>
<td>Brtti</td>
<td></td>
</tr>
<tr>
<td>0x10</td>
<td></td>
<td></td>
<td>Drtti</td>
</tr>
<tr>
<td>0x18</td>
<td></td>
<td></td>
<td>Crtti</td>
</tr>
<tr>
<td>0x20</td>
<td>Afoo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x28</td>
<td></td>
<td>Bfoo</td>
<td></td>
</tr>
<tr>
<td>0x30</td>
<td></td>
<td>Dfoo</td>
<td></td>
</tr>
<tr>
<td>0x38</td>
<td></td>
<td></td>
<td>Afoo</td>
</tr>
<tr>
<td>0x40</td>
<td>Bbar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x48</td>
<td></td>
<td>Bbar</td>
<td></td>
</tr>
<tr>
<td>0x50</td>
<td></td>
<td>Dboo</td>
<td></td>
</tr>
<tr>
<td>0x58</td>
<td></td>
<td></td>
<td>Cbaz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VTable Entry</th>
<th>Old Offset</th>
<th>New Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtti</td>
<td>-0x8</td>
<td>-0x20</td>
</tr>
<tr>
<td>foo</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bar</td>
<td>0x8</td>
<td>0x18</td>
</tr>
<tr>
<td>boo</td>
<td>0x10</td>
<td>0x20</td>
</tr>
<tr>
<td>baz</td>
<td>0x8</td>
<td>0x20</td>
</tr>
</tbody>
</table>

²Bounov, Kici, and Lerner 2016.
Handling Multiple Inheritance

- Multiple Inheritance can be decomposed into several single inheritances.
- Each single inheritance is managed individually.
Implementation of IRMs

- Checking necessary if \( vptr \in [a, b] \) and \( vptr \mod 2^n = 0 \).

### Trivial implementation:

```plaintext
cmp $vptr, $a
jlt FAIL
cmp $vptr, $b
jgt FAIL
and $vptr, 1111...n
cmp $vptr, 0
jne FAIL
... ;Success
```

### Enhanced implementation:

```plaintext
$diff = $vptr - $a
$diffR = rotr $diff, n
cmp $diffR, ($b-$a) >> n
jgt FAIL
... ;Success
```
Benchmarks

- Implemented approaches into the LLVM compiler.

Runtime overhead:\(^3\):

Binary size overhead:\(^4\):

---

\(^3\)Bounov, Kici, and Lerner 2016.

\(^4\)Bounov, Kici, and Lerner 2016.
Other Approaches

- Other compiler based techniques (SafeDispatch⁵, VTV⁶).
  → Similar runtime and binary overhead.
- General CFI which protect all control transfers (also normal function pointers and returns).
  → Bigger runtime overhead.

---

⁵Jang, Tatlock, and Lerner 2014.
⁶Tice et al. 2014.
References

