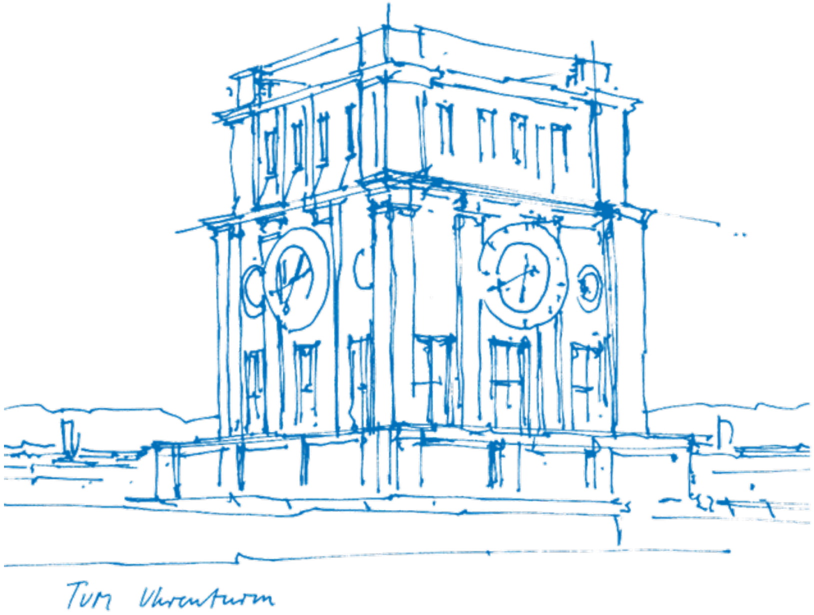
Protecting C++ Dynamic Dispatch Through VTable Interleaving

Benjamin Zanger

Technical University of Munich
Department of Informatics
Chair for IT Security

Munich, November 19, 2018



Chair of IT Security
Department of Informatics
Technical University of Munich

Outline

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

VTable Hijacking

Protecting Dynamic Dispatches
- VTable Ordering
- VTable Interleaving
- Multiple Inheritance
- Optimization

Benchmarks

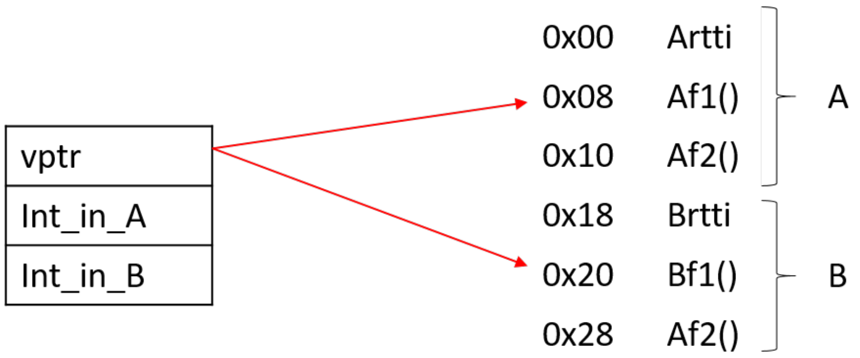
Comparison with other Approaches

Chair of IT Security
Department of Informatics
Technical University of Munich

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;
};
```

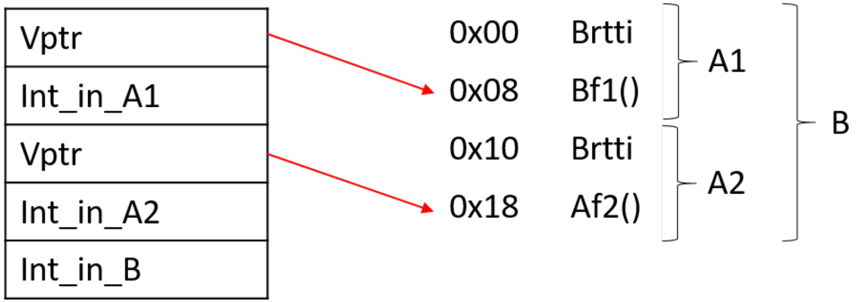


Multiple Inheritance

```
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

- Exploting 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:

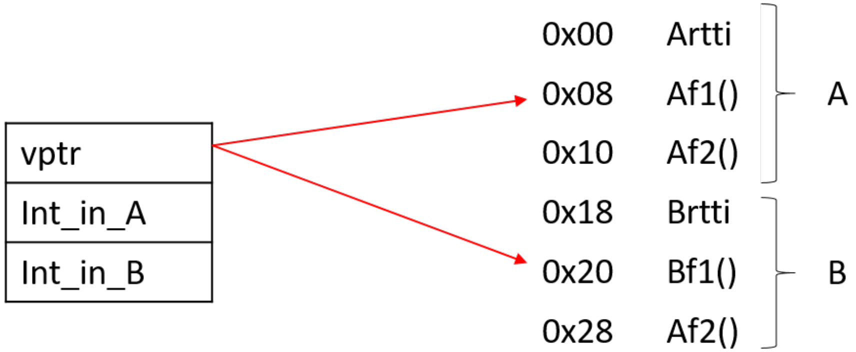
- Hacker capable of modifying the Heap.
- Registers are safe.

Protecting Dynamic Dispatches

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

$$vptr \in \{0x08, 0x20\} \tag{1}$$

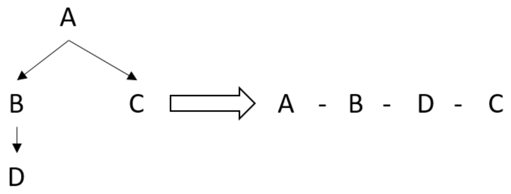
⇒ Differences are in the implementation.



Chair of IT Security
Department of Informatics
Technical University of Munich

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[1]:




```
0x18  Artti      ]
0x20  Afoo       ]  A
0x28  <padding>
0x30  <padding>
0x38  Brtti      ]
0x40  Bfoo       ]  B
0x48  Bbar       ]
0x50  <padding>
0x58  Drtti      ]
0x60  Dfoo       ]  D
0x68  Bbar       ]
0x70  Dboo       ]
0x78  Crtti      ]
0x80  Afoo       ]  C
0x88  Cbaz       ]
```

Class	Start	End	Alignment
A	0x20	0x80	0x20
B	0x40	0x60	0x20
C	0x80	0x80	0x20
D	0x60	0x60	0x20

[1] 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.
- Example[2]:

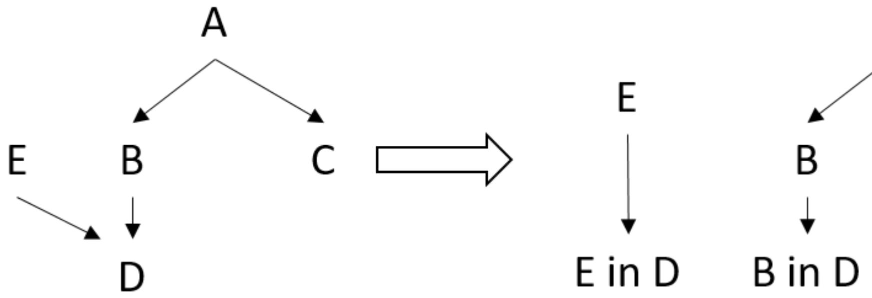
```
      A      B      D      C
0x00  Artti
0x08         Brtti
0x10                Drtti
0x18                       Crtti
0x20  Afoo
0x28         Bfoo
0x30                Dfoo
0x38                       Afoo
0x40         Bbar
0x48                Bbar
0x50                Dboo
0x58                       Cbaz
```

VTable Entry	Old Offset	New Offset
rtti	-0x8	-0x20
foo	0	0
bar	0x8	0x18
boo	0x10	0x20
baz	0x8	0x20

[2] Bounov, Kici, and Lerner 2016.

Handling Multiple Inheritance

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



Chair of IT Security
Department of Informatics
Technical University of Munich

Implementation of IRMs

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

Trivial implementation:

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

Enhanced implementation:

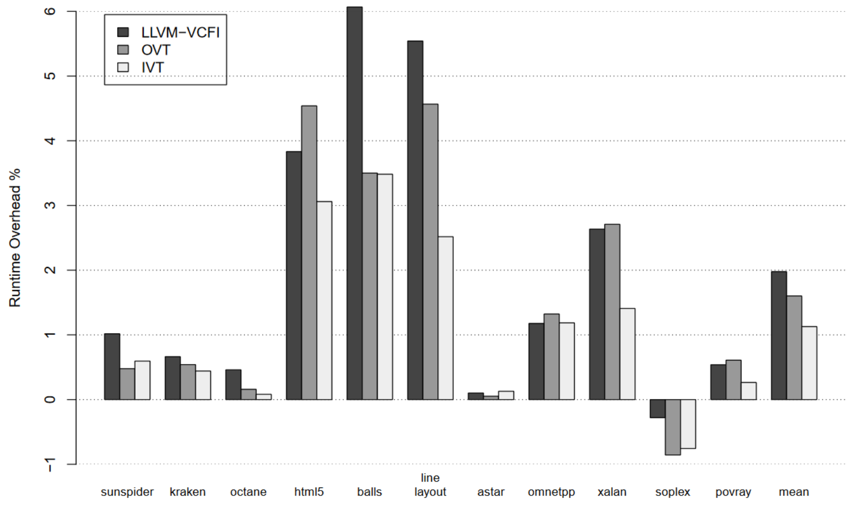
```
$diff = $vptr - $a
$diffR = rotr $diff,n
cmp $diffR, ($b-$a) >> n
jgt FAIL
... ;Success
```

Chair of IT Security
Department of Informatics
Technical University of Munich

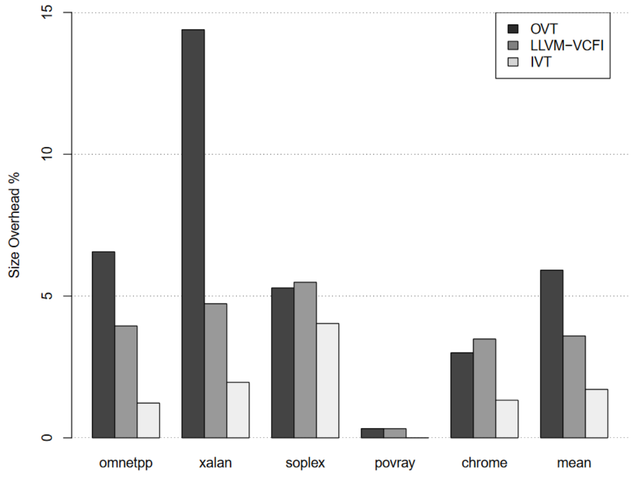
Benchmarks

- 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 (SafeDispatch[5], VTV[6]).
  → Similar runtime and binary overhead.
- General CFI which protect all control transfers (also normal function pointers and returns).
  → Bigger runtime overhead.

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

References

- Bounov, D., R. G. Kici, and S. Lerner (2016). “Protecting C++ Dynamic Dispatch Through VTable Interleaving.”. In: *NDSS*.
- Jang, D., Z. Tatlock, and S. Lerner (2014). “SafeDispatch: Securing C++ Virtual Calls from Memory Corruption Attacks.”. In: *NDSS*.
- Tice, C. et al. (2014). “Enforcing Forward-Edge Control-Flow Integrity in GCC & LLVM.”. In: *USENIX Security Symposium*, pp. 941–955.