MARX – Uncovering Class Hierarchies in C++ Programs

NDSS 2017, San Diego

Andre Pawlowski*, Moritz Contag*, Victor van der Veen†, Chris Ouwehand†, Thorsten Holz*, Herbert Bos†, Elias Anthonasopoulos‡, Cristiano Giuffrida†

* Ruhr-Universität Bochum † Vrije Universiteit Amsterdam ‡ University of Cyprus
Introduction
• We present MARX, a tool to extract class hierarchies from legacy binaries.

• This eases static analysis of C++ applications.

• We use the results for security-related use cases:
  1. Control-Flow Integrity
  2. Type-safe Object Reuse
Motivation

• C++ applications are hard to analyze \textit{statically}.
  • Class inheritance.
  • Objects allocated on the heap.
  • Indirect function calls (polymorphism).
Motivation

- C++ applications are hard to analyze **statically**.
  - Class inheritance.
  - Objects allocated on the heap.
  - Indirect function calls (polymorphism).

- Solving these problems aids in the **reverse engineering** process.
- **Exploit mitigations** would profit from solutions to these problems.
  - Counterfeit Object-oriented Programming.
  - Fine-grained control over forward edge.
Control-Flow Integrity
new object A

mov rdi, object
call [[rdi] + offset]
Control-Flow Integrity

Code

Function X

new object A

mov rdi, object
call [[rdi] + offset]

High Level

Object A

vtblptr
var_0

Class A

function_a1
function_a2

Class B

[...]
Control-Flow Integrity

**Code**

Function X

```c
[...
new object A
[...
mov rdi, object
call [[rdi] + offset]
[...
```

**High Level**

Object A

```c
vtblptr
tblptr
[...
```

Class A

```c
function_a1
function_a2
[...
```

Class B

```c
[...
```

Class Malicious

```c
shellcode_0
shellcode_1
[...
```
Control-Flow Integrity

Code

Function X

[...]
new object A
[...]
mov rdi, object
call [[rdi] + offset]
[...]

High Level

Object A

vtblptr
var_0
[...]

Class A

function_a1
function_a2
[...]

Class Malicious

shellcode_0
shellcode_1
[...]

Class B

[...]
Control-Flow Integrity

**Code**

```
Function X

[...]
new object A
[...]
mov rdi, object
call [rdi] + offset
[...]
```

**High Level**

```
Object A

vtblptr
var_0
[...]
```

```
Class A

function_a1
function_a2
[...]
```

```
Class B

[...]
```

```
Class Malicious

shellcode_0
shellcode_1
[...]
```

**MARX – Uncovering Class Hierarchies in C++ Programs**
**Control-Flow Integrity**

---

**Code**

<table>
<thead>
<tr>
<th>Function X</th>
</tr>
</thead>
<tbody>
<tr>
<td>[...]</td>
</tr>
<tr>
<td>new object A</td>
</tr>
<tr>
<td>[...]</td>
</tr>
<tr>
<td>mov rdi, object</td>
</tr>
<tr>
<td>object in hierarchy?</td>
</tr>
<tr>
<td>call [[rdi] + offset]</td>
</tr>
<tr>
<td>[...]</td>
</tr>
</tbody>
</table>

---

**High Level**

<table>
<thead>
<tr>
<th>Object A</th>
</tr>
</thead>
<tbody>
<tr>
<td>vtblptr</td>
</tr>
<tr>
<td>var_0</td>
</tr>
<tr>
<td>[...]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class A</th>
</tr>
</thead>
<tbody>
<tr>
<td>function_a1</td>
</tr>
<tr>
<td>function_a2</td>
</tr>
<tr>
<td>[...]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class B</th>
</tr>
</thead>
<tbody>
<tr>
<td>[...]</td>
</tr>
</tbody>
</table>
Control-Flow Integrity

Code

Function X

...]
new object A
...]
mov rdi, object
object in hierarchy?
call [[rdi] + offset]
...]

High Level

Object A

vtblptr
var_0
...]

Class A

function_a1
function_a2
...]

Class Malicious

shellcode_0
shellcode_1
...]

Class B

...]
Control-Flow Integrity

Code

Function X

[...]
new object A
[...]
mov rdi, object
object in hierarchy?
call [[rdi] + offset]
[...]

Object A

vtblptr
var_0
[...]

Class A

function_a1
function_a2
[...]

Class B

Class Malicious

shellcode_0
shellcode_1
[...]

MARX – Uncovering Class Hierarchies in C++ Programs
Control-Flow Integrity

Code

Function X

[...]
new object A
[...]
mov rdi, object

object in hierarchy?
call [[rdi] + offset]
[...]

Object A

vtblptr
var_0
[...]

High Level

Class A

function_a1
function_a2
[...]

Class B

[...]

Class Malicious

shellcode_0
shellcode_1
[...]
Approach
Indirect function calls are prominent C++ artifacts on binary level.
At high level, a class may *inherit* from another.
Inheritance relationships are not readily available in the binary.
Classes are (loosely) represented by *vtables* instead.
**Approach**

- **MARX statically** recovers the class hierarchy from an x86-64 binary (Itanium ABI).
  - Represents class hierarchy as set of vtables.
  - Ties set of vtables to indirect function call.

Analysis steps:
1. Identify vtables,
2. Inspect their usages,
3. Refine results using heuristics,
4. Merge results across modules.
• **MARX** *statically* recovers the class hierarchy from an x86-64 binary (Itanium ABI).
  
  • Represents class hierarchy as set of vtables.
  
  • Ties set of vtables to indirect function call.

• Analysis steps:
  
  1. Identify *vtables*,
  2. inspect their *usages*,
  3. refine results using *heuristics*,
  4. merge results across *modules*. 
Approach

1. Vtable Candidates
**Approach – Vtable Identification**

<table>
<thead>
<tr>
<th>Object A</th>
<th>Vtable A</th>
</tr>
</thead>
<tbody>
<tr>
<td>vtblptr A</td>
<td>-0x10 metadata_0</td>
</tr>
<tr>
<td></td>
<td>-0x08 metadata_1</td>
</tr>
<tr>
<td></td>
<td>0x00 A::func_a1</td>
</tr>
<tr>
<td></td>
<td>0x08 A::func_a2</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Heap or Stack

Data Memory

Marx – Uncovering Class Hierarchies in C++ Programs
Approach – Vtable Identification

Heap or Stack

Object A
vtblptr A

Data Memory

Vtable A

-0x10 metadata_0
-0x08 metadata_1
0x00 A::func_a1
0x08 A::func_a2
...

MARX – Uncovering Class Hierarchies in C++ Programs
Approach – Vtable Identification

Heap or Stack

Object A
vtblptr A

Data Memory

Vtable A
-0x10 metadata_0
-0x08 metadata_1
0x00 A::func_a1
0x08 A::func_a2
...

Data Memory resides in read-only memory (e.g., .rodata)
### Approach – Vtable Identification

#### Heap or Stack

- **Object A**
- `vtblptr A`

#### Data Memory

<table>
<thead>
<tr>
<th>Vtable A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10   metadata_0</td>
</tr>
<tr>
<td>0x08   metadata_1</td>
</tr>
<tr>
<td>0x00   A::func_a1</td>
</tr>
<tr>
<td>0x08   A::func_a2</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

*only first function entry is referenced*
**Objective**

- Heap or Stack
- vtblptr A
- Vtable A
  - 0x10 metadata_0
  - 0x08 metadata_1
  - 0x00 A::func_a1
  - 0x08 A::func_a2
  - ...

**Data Memory**

Metadata fields are within defined range or 0.
**APPROACH – Vtable Identification**

- **Object A**
  - `vtblptr A`
  - `Vtable A`
    - `-0x10` `metadata_0`
    - `-0x08` `metadata_1`
    - `0x00` `A::func_a1`
    - `0x08` `A::func_a2`
    - `...`

Point into code section or are relocation entries.
Approach

2. Vtable Usages
**Approach – Overwrite Analysis**

Hierarchy

- A
- B

Object

- Call constructor A
- Write vtblptr B

Code

```
Function X ↓
[...]
new object B
call constructor B

Constructor B ↓
call constructor A
write vtblptr B
init. variable_1
return

Constructor A ↓
write vtblptr A
return
```
**Approach – Overwrite Analysis**

**Hierarchy**

1. A
2. B

**Object**

- Object B
  - vtblptr
  - variable_1

**Code**

- Function X
  - [...]
  - new object B
  - call constructor B

- Constructor B
  - call constructor A
  - write vtblptr B
  - init. variable_1
  - return

- Constructor A
  - write vtblptr A
  - return

**Marx – Uncovering Class Hierarchies in C++ Programs**
**Approach – Overwrite Analysis**

**Hierarchy**

```
A
  ↓
B
```

**Object**

```
Object B
vtblptr
variable_1
```

**Code**

```
Function X ↓
[...]
new object B
call constructor B

Constructor B ↓
call constructor A
write vtblptr B
init. variable_1
return

Constructor A ↓
write vtblptr A
return
```
**Approach – Overwrite Analysis**

**Hierarchy**

```
Object B
vtblptr A
variable_1
```

**Code**

```
Function X ↓
[...]
new object B
call constructor B

Constructor B ↓
call constructor A
write vtblptr B
init. variable_1
return

Constructor A ↓
write vtblptr A
return
```
Hierarchy

Object

Code

Function X ↓

[...]

new object B

call constructor B

Constructor B ↓

call constructor A

write vtblptr B

init. variable_1

return

Constructor A ↓

write vtblptr A

return
**Approach – Overwrite Analysis**

### Hierarchy

- **A**
- **B**

### Object

- **Object B**
  - `vtblptr A → B`
  - `variable_1`

### Code

**Function X ↓**

- ` [...]`
- `new object B`
- `call constructor B`

**Constructor B ↓**

- `call constructor A`
- `write vtblptr B`
- `init. variable_1`
- `return`

**Constructor A ↓**

- `write vtblptr A`
- `return`
**Approach – Overwrite Analysis**

### Hierarchy

- **A**
- **B**

### Object

- **Object B**
- `vtblptr B`
- `variable_1`

### Code

**Function X**

```cpp
[...]
new object B
call constructor B
```

**Constructor B**

```cpp
call constructor A
write vtblptr B
init. variable_1
return
```

**Constructor A**

```cpp
write vtblptr A
return
```
**Approach – Path Generation**

- In a function, **build paths** visiting
  1. indirect calls,
  2. direct calls to `new`, and
  3. instructions operating on `vtables`.

- **Resolve** indirect function calls for known `vtables` in current context.

- Follows calls with a **call depth** of 2.
Approach

3. Vtable Heuristics
### Approach – Function Heuristics

#### Hierarchy

- **Class A**
  - `virt. func_a1()`
  - `virt. func_a2()`

- **Class B**
  - `virt. func_a1()`
  - `virt. func_b1()`

#### Data Memory

- **Vtable A**
  - `-0x10 0`
  - `-0x08 0`
  - `0x00 A::func_a1`
  - `0x08 A::func_a2`

- **Vtable B**
  - `-0x10 0`
  - `-0x08 0`
  - `0x00 B::func_a1`
  - `0x08 A::func_a2`
  - `0x10 B::func_b1`
**Approach – Function Heuristics**

### Hierarchy

**Class A**
- `virt. func_a1()`
- `virt. func_a2()`

**Class B**
- `virt. func_a1()`
- `virt. func_b1()`

### Data Memory

**Vtable A**
- `-0x10 0`
- `-0x08 0`
- `0x00 A::func_a1`
- `0x08 A::func_a2`

**Vtable B**
- `-0x10 0`
- `-0x08 0`
- `0x00 B::func_a1`
- `0x08 A::func_a2`
- `0x10 B::func_b1`
**Hierarchy**

- **Class A**
  - `virt. func_a1()`
  - `virt. func_a2()`

- **Class B**
  - `virt. func_a1()`
  - `virt. func_b1()`

**Data Memory**

- **Vtable A**
  - `0x10 0`
  - `0x08 0`
  - `0x00 A::func_a1`
  - `0x08 A::func_a2`

- **Vtable B**
  - `0x10 0`
  - `0x08 0`
  - `0x00 B::func_a1`
  - `0x08 A::func_a2`
  - `0x10 B::func_b1`
**Hierarchy**

- **Class A**
  - virt. func_a1()
  - virt. func_a2()

- **Class B**
  - virt. func_a1()
  - virt. func_b1()

**Data Memory**

- **Vtable A**
  - -0x10 0
  - -0x08 0
  - 0x00 A::func_a1
  - 0x08 A::func_a2

- **Vtable B**
  - -0x10 0
  - -0x08 0
  - 0x00 B::func_a1
  - 0x08 A::func_a2
  - 0x10 B::func_b1

Same function entry, probably same hierarchy
Approach

4. Inter-Modular Dependencies
Applications may consist of several modules.
Independent analysis yields, e.g., \textit{three} distinct hierarchies.
Unless we merge results across modules, we \textit{underestimate} the hierarchy.
Evaluation
## EVALUATION – Class Hierarchy Reconstruction

<table>
<thead>
<tr>
<th>Application</th>
<th>No. of Hierarchies</th>
<th>Exactly Reconstructed</th>
<th>Time (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL Server</td>
<td>78</td>
<td>69 (88%)</td>
<td>11:36</td>
</tr>
<tr>
<td>MongoDB</td>
<td>158</td>
<td>137 (87%)</td>
<td>1:08</td>
</tr>
<tr>
<td>Node.js</td>
<td>59</td>
<td>55 (93%)</td>
<td>0:33</td>
</tr>
</tbody>
</table>
## Evaluation – Class Hierarchy Reconstruction

<table>
<thead>
<tr>
<th>Application</th>
<th>No. of Hierarchies</th>
<th>Exactly Reconstructed</th>
<th>Time (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL Server</td>
<td>78</td>
<td>69 (88%)</td>
<td>11:36</td>
</tr>
<tr>
<td>MongoDB</td>
<td>158</td>
<td>137 (87%)</td>
<td>1:08</td>
</tr>
<tr>
<td>Node.js</td>
<td>59</td>
<td>55 (93%)</td>
<td>0:33</td>
</tr>
</tbody>
</table>

Analysis time on average for 5 real-world applications.
## Evaluation – Class Hierarchy Reconstruction

<table>
<thead>
<tr>
<th>Application</th>
<th>No. of Hierarchies</th>
<th>Exactly Reconstructed</th>
<th>Time (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL Server</td>
<td>78</td>
<td>69 (88%)</td>
<td>11:36</td>
</tr>
<tr>
<td>MongoDB</td>
<td>158</td>
<td>137 (87%)</td>
<td>1:08</td>
</tr>
<tr>
<td>Node.js</td>
<td>59</td>
<td>55 (93%)</td>
<td>0:33</td>
</tr>
</tbody>
</table>

48 min analysis time on average for 5 real-world applications.
### Evaluation – Class Hierarchy Reconstruction

<table>
<thead>
<tr>
<th>Application</th>
<th>No. of Hierarchies</th>
<th>Exactly Reconstructed</th>
<th>Time (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL Server</td>
<td>78</td>
<td>69 (88%)</td>
<td>11:36</td>
</tr>
<tr>
<td>MongoDB</td>
<td>158</td>
<td>137 (87%)</td>
<td>1:08</td>
</tr>
<tr>
<td>Node.js</td>
<td>59</td>
<td>55 (93%)</td>
<td>0:33</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application</th>
<th>Overestimated</th>
<th>Underestimated</th>
<th>Not Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL Server</td>
<td>1 (1%)</td>
<td>7 (9%)</td>
<td>1 (1%)</td>
</tr>
<tr>
<td>MongoDB</td>
<td>0</td>
<td>8 (5%)</td>
<td>13 (8%)</td>
</tr>
<tr>
<td>Node.js</td>
<td>2 (3%)</td>
<td>2 (3%)</td>
<td>0</td>
</tr>
</tbody>
</table>

48 min analysis time on average for 5 real-world applications.
• Use MARX’ results to harden legacy binaries.
• Use MARX’ results to harden legacy binaries.

• Implemented and tested two applications:
  1. Control-Flow Integrity  ...................... “binary-level Vtable Verification (VTV)”
  2. Type-safe Object Reuse  ...................... “binary-level Cling”, bucketing by hierarchy

• Lack of precision of analysis?
• **Type-safe Object Reuse** allows imprecision, lowered security.

• **Control-Flow Integrity** may break applications.
EVALUATION – Precision

- **Type-safe Object Reuse** allows imprecision, lowered security.

- **Control-Flow Integrity** may break applications.
  - Enrich static results with **dynamic analysis** (unit tests).
  - Fall-back to computationally intensive **slow path** (*PathArmor*).
EVALUATION – Precision

- **Type-safe Object Reuse** allows imprecision, lowered security.

- **Control-Flow Integrity** may break applications.
  - Enrich static results with *dynamic analysis* (unit tests).
  - Fall-back to computationally intensive *slow path* (*PathArmor*).

- Please refer to the paper for further details.
Conclusion
• Approach is **compiler-dependent**, optimizations may break assumptions.
  • Abstract base classes may not yield a vtable for us to discover.
LIMITATIONS

- Approach is **compiler-dependent**, optimizations may break assumptions.
  - Abstract base classes may not yield a vtable for us to discover.
- Analysis is prone to **path explosion** (inter/intra-procedural path generation).
· Approach is compiler-dependent, optimizations may break assumptions.
  · Abstract base classes may not yield a vtable for us to discover.

· Analysis is prone to path explosion (inter/intra-procedural path generation).

· Applications building upon results have to tolerate imprecision.
CONCLUSION

- MARX succeeds in **recovering the majority** of class hierarchies.
  - Large, real-world applications.
  - Promising results.

- Results are applicable to **security-related** use cases on the **binary level**.

- Our C++ **open-source implementation** based on **libVEX** is available at

  https://github.com/RUB-SysSec/Marx.
Thank you for your attention.
• Marx succeeds in **recovering the majority** of class hierarchies.
  • Large, real-world applications.
  • Promising results.

• Results are applicable to **security-related** use cases on the **binary level**.

• Our C++ **open-source implementation** based on **libVEX** is available at

  [https://github.com/RUB-SysSec/Marx](https://github.com/RUB-SysSec/Marx)