RUHR UNIVERSITÄT BOCHUM RUB

VU UNIVERSITY AMSTERDAM

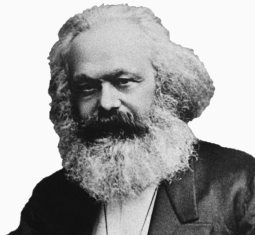
University of Cyprus

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

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

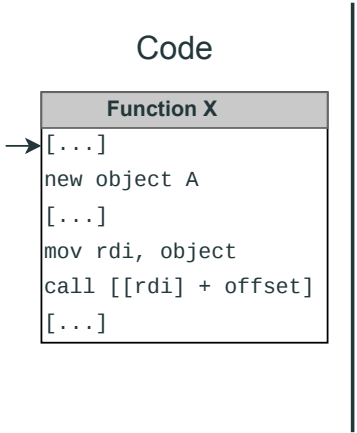
- C++ applications are hard to analyze **statically**.
  - Class inheritance.
  - Objects allocated on the heap.
  - Indirect function calls (polymorphism).

Motivation

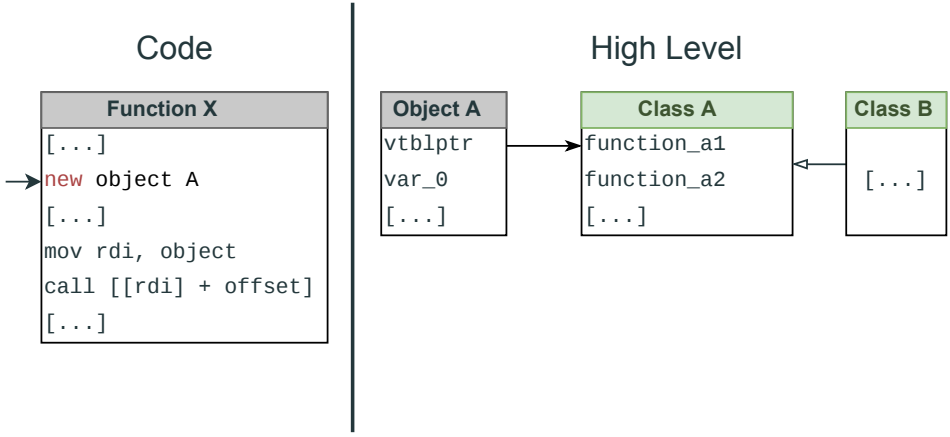
- C++ applications are hard to analyze **statically**.
  - Class inheritance.
  - Objects allocated on the heap.
  - Indirect function calls (polymorphism).

- 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.

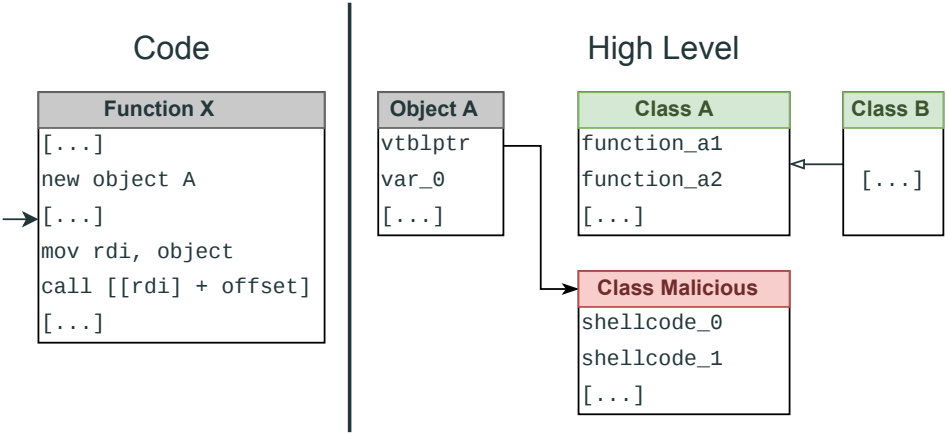
Control-Flow Integrity


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

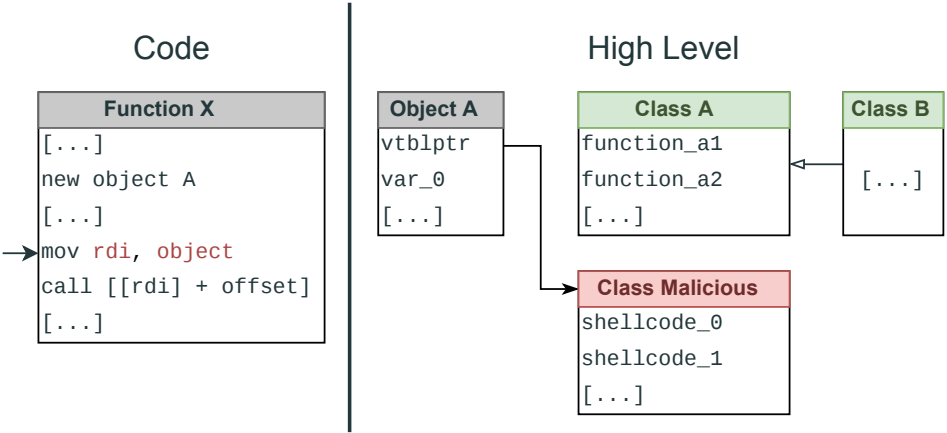
Control-Flow Integrity



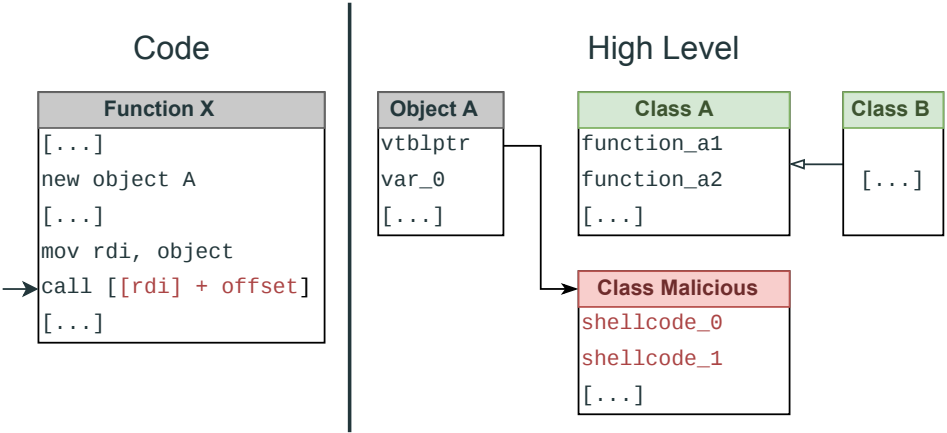
Control-Flow Integrity



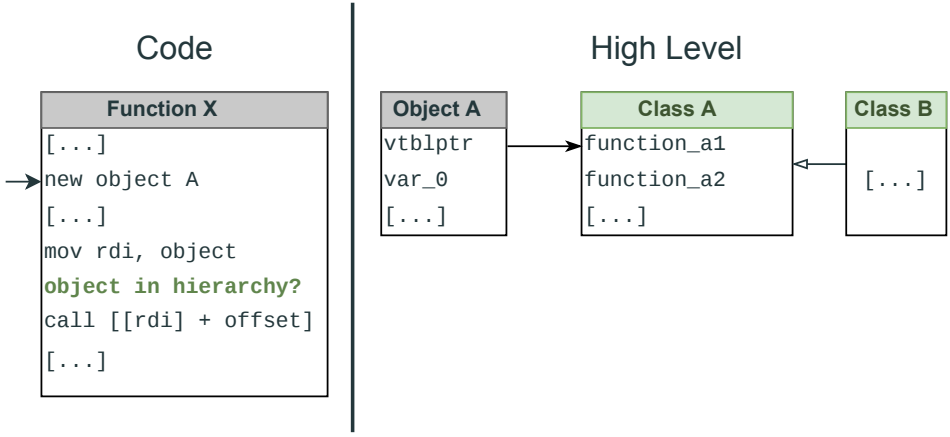
Control-Flow Integrity



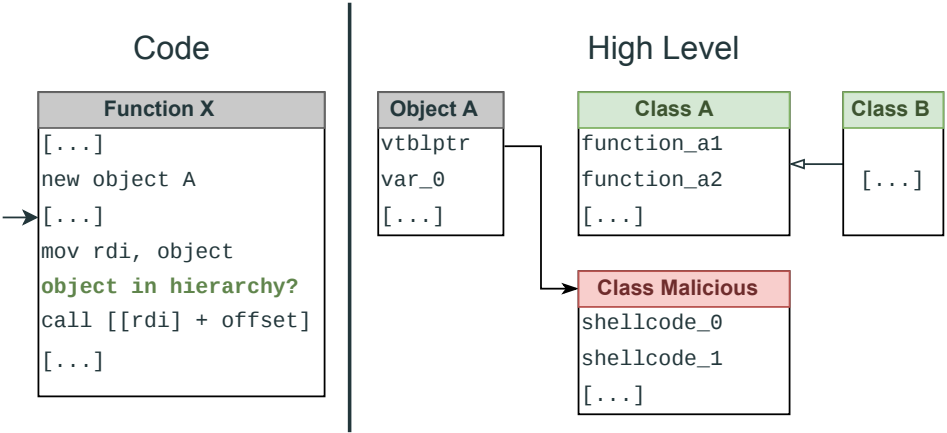
Control-Flow Integrity



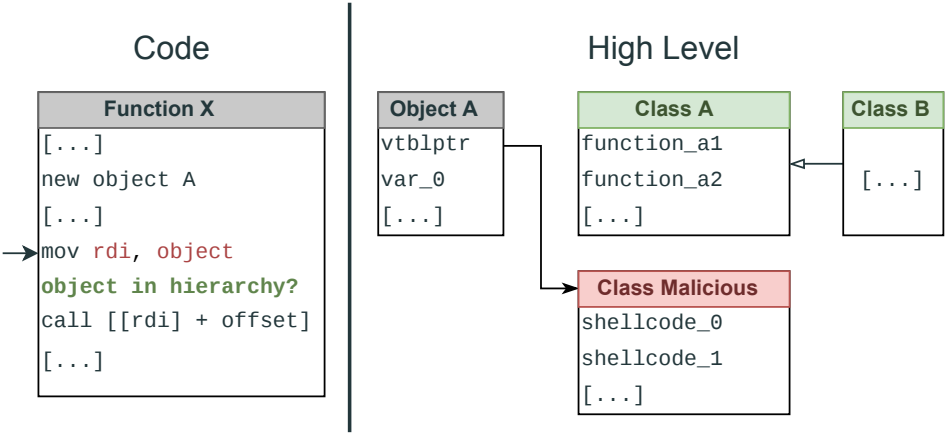
Control-Flow Integrity



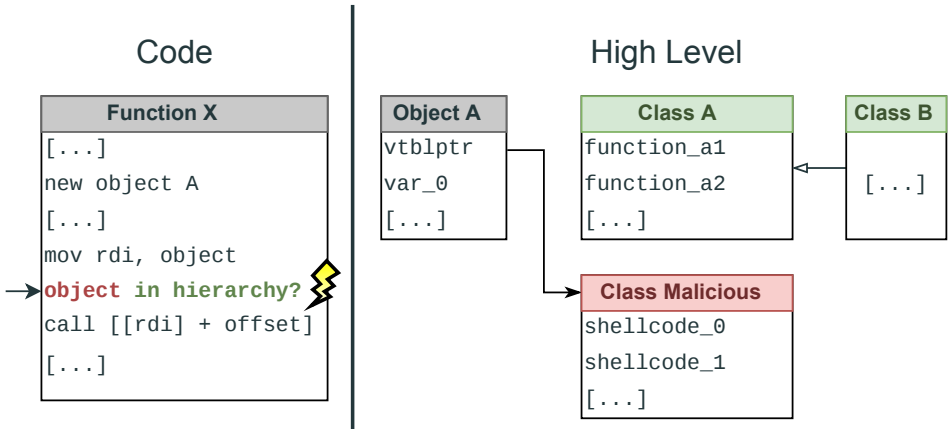
Control-Flow Integrity



Control-Flow Integrity

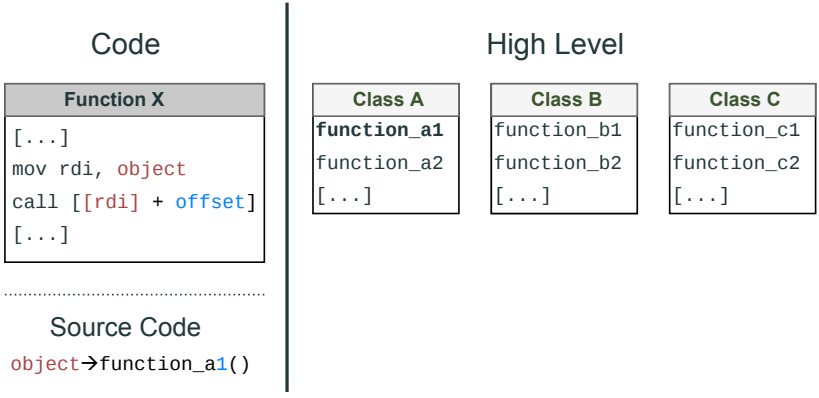


Control-Flow Integrity



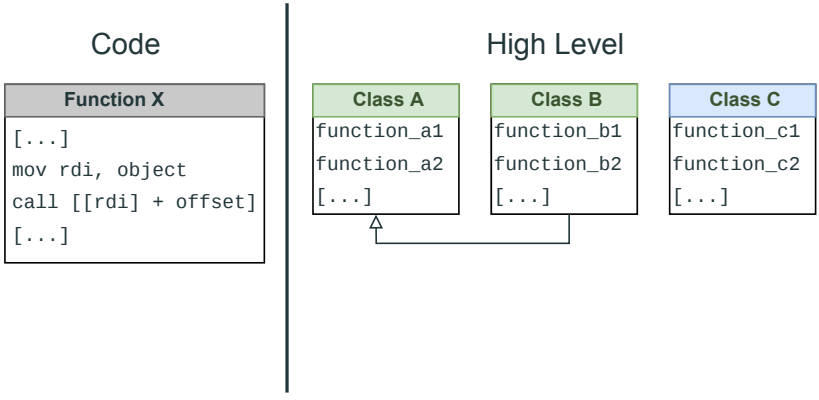
Approach

Approach



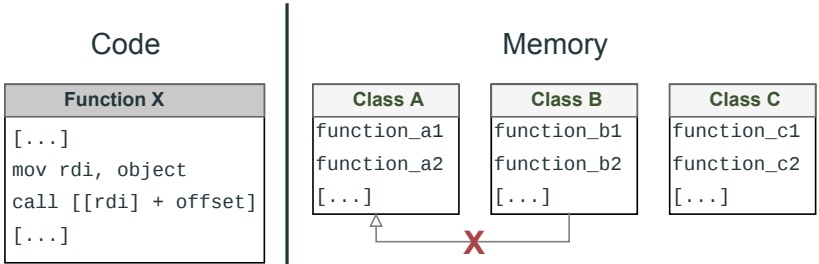
Indirect function calls are prominent C++ artifacts on binary level.

Approach



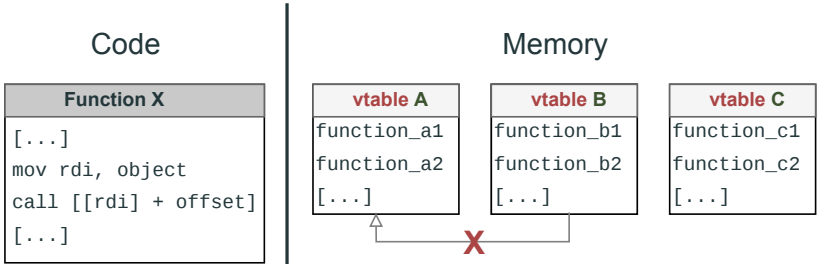
At high level, a class may *inherit* from another.

Approach



Inheritance relationships are not readily available in the binary.

Approach



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.

Approach

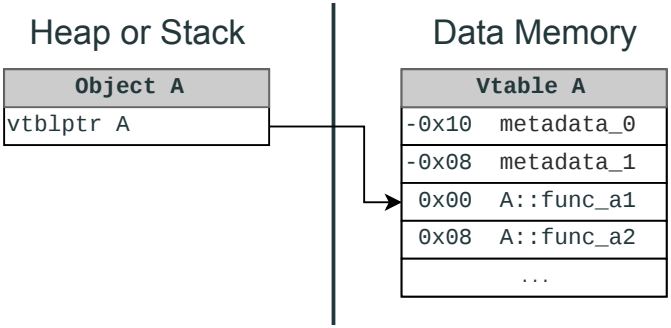
- MARX **statically** recovers the class hierarchy from an x86-64 binary (Itanium ABI).
  - Represents class hierarchy as set of vtables.
  - Ties set of vtables to indirect function call.

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

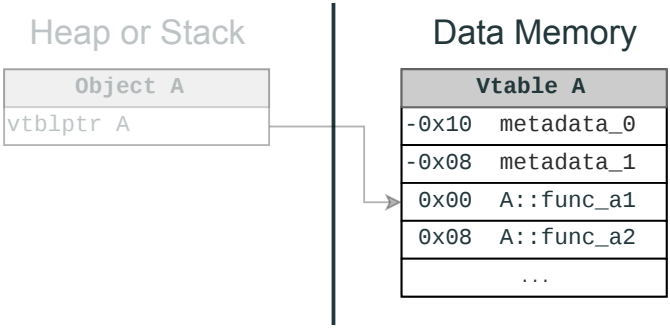
Approach

1. Vtable Candidates

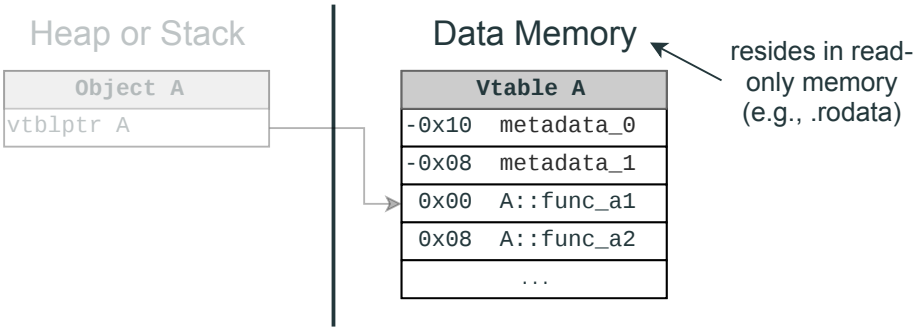
Approach – Vtable Identification



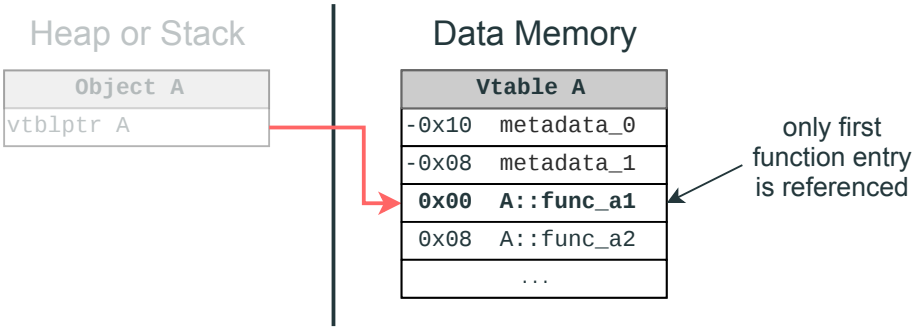
Approach – Vtable Identification



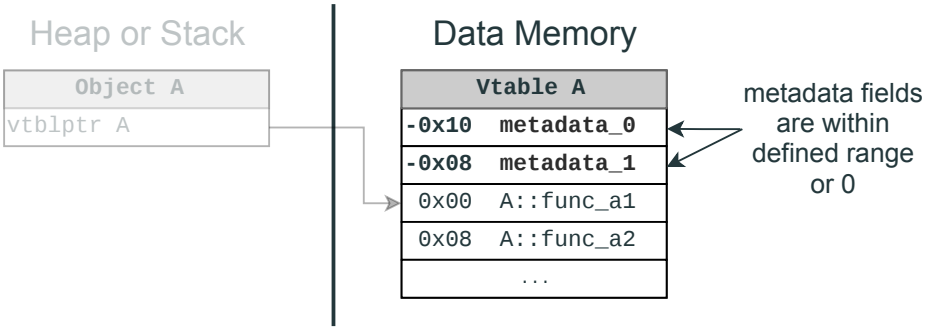
Approach – Vtable Identification



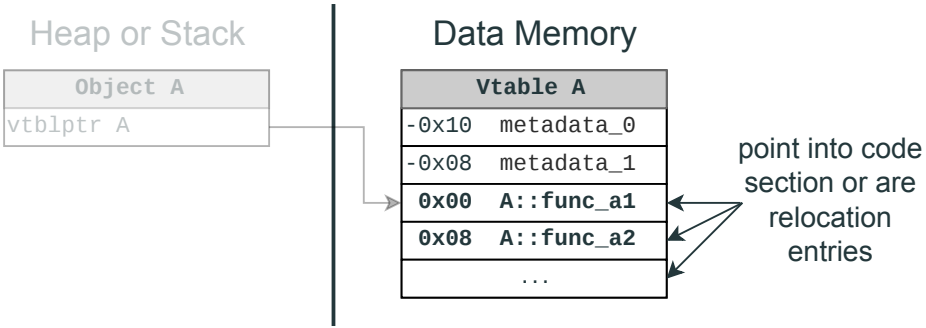
Approach – Vtable Identification



Approach – Vtable Identification



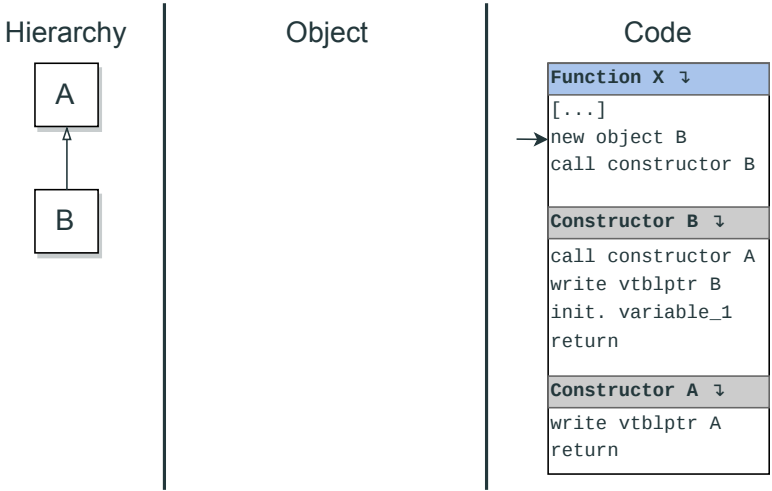
Approach – Vtable Identification



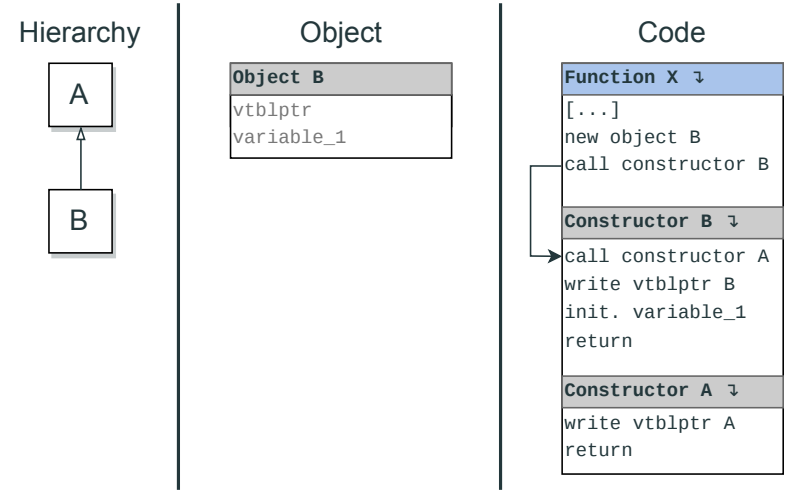
Approach

2. Vtable Usages

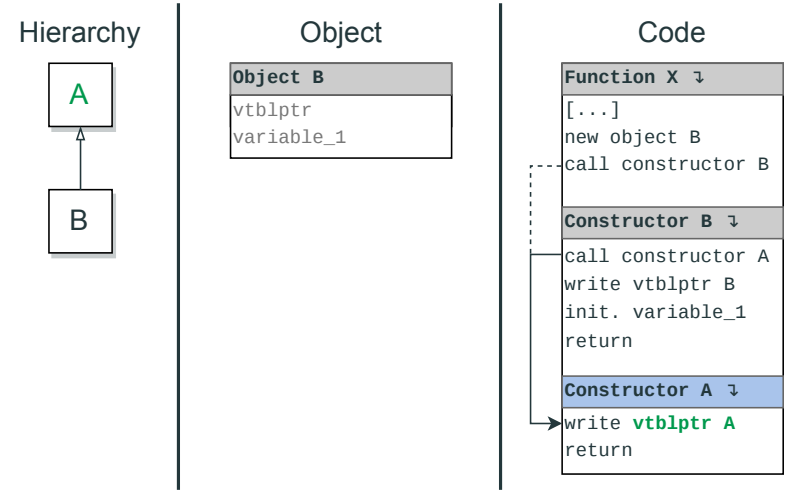
Approach – Overwrite Analysis



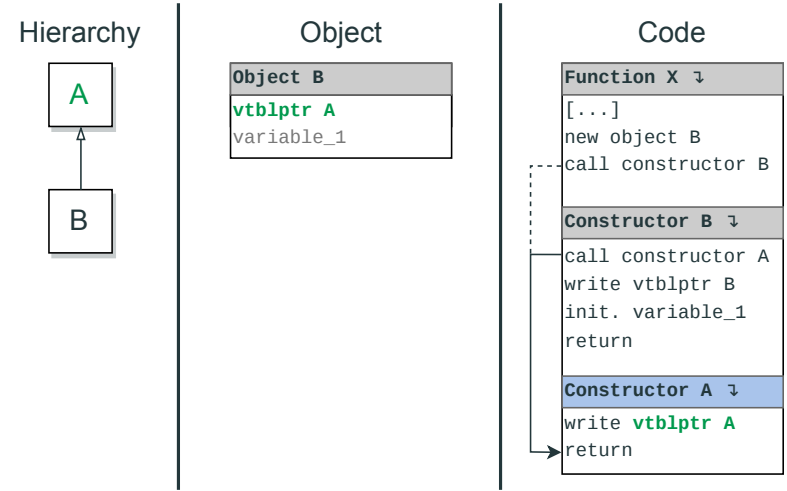
Approach – Overwrite Analysis



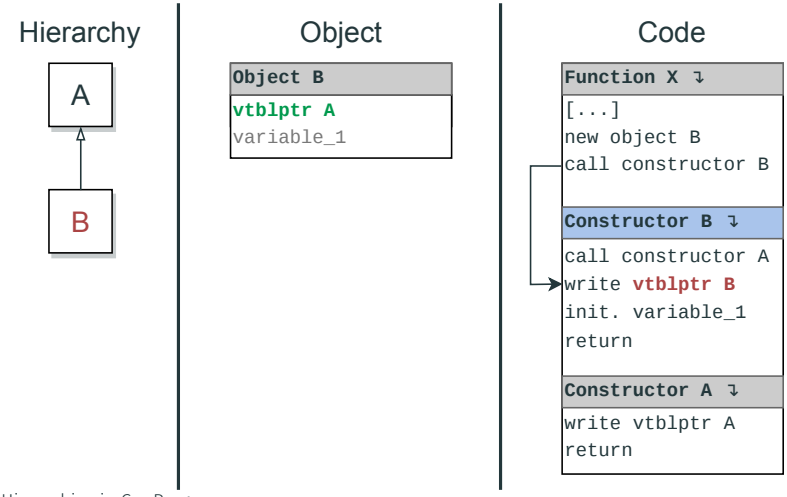
Approach – Overwrite Analysis



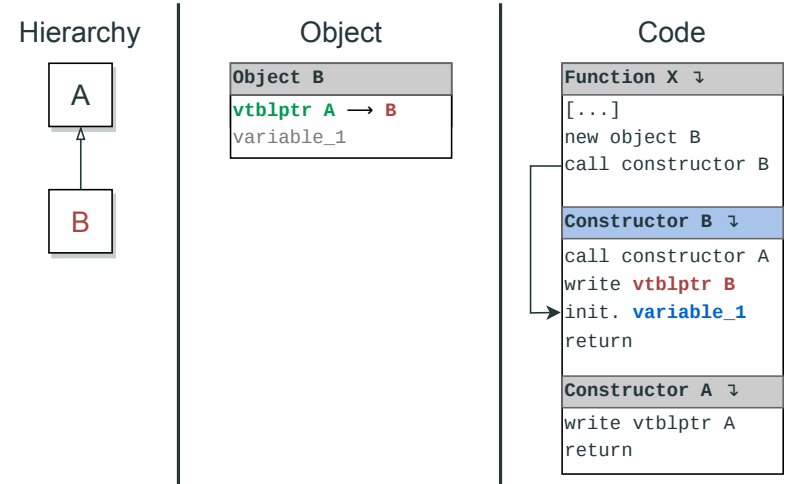
Approach – Overwrite Analysis



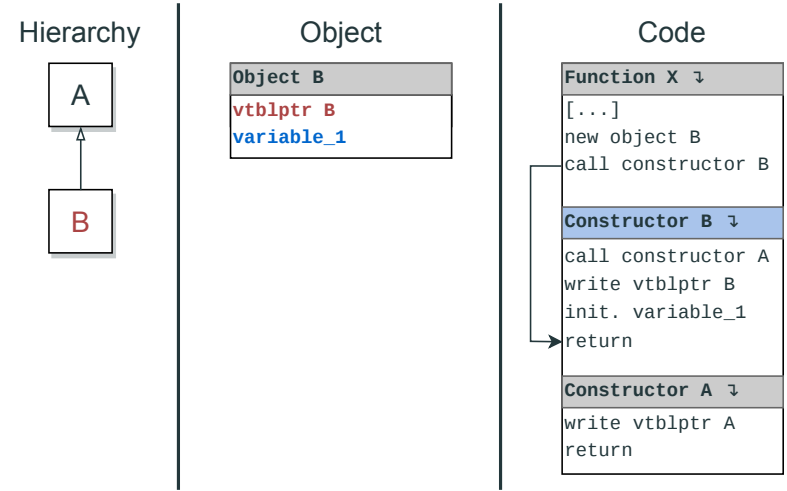
Approach – Overwrite Analysis



Approach – Overwrite Analysis



Approach – Overwrite Analysis



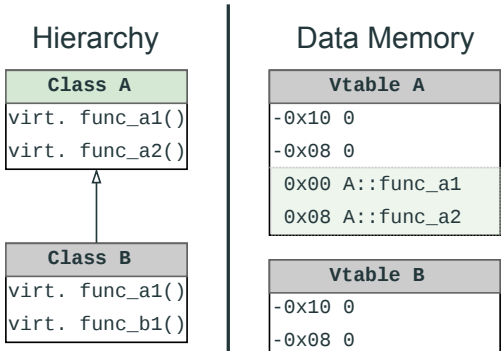
Approach – Path Generation

- 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.

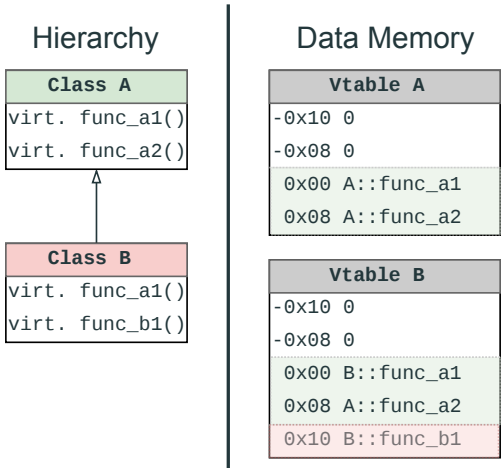
Approach

3. Vtable Heuristics

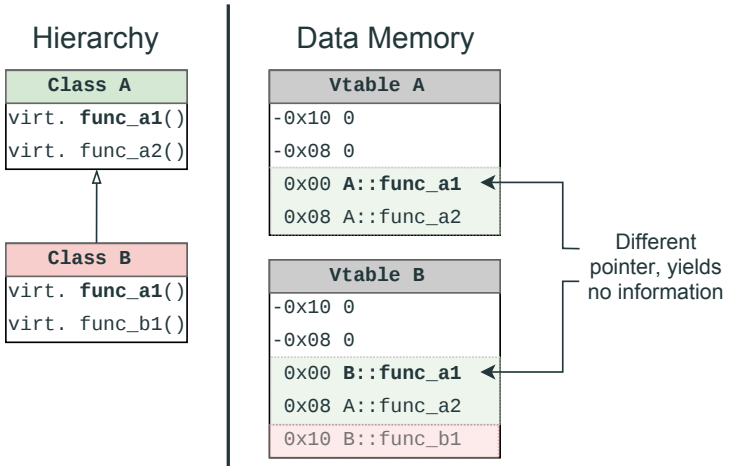
Approach – Function Heuristics



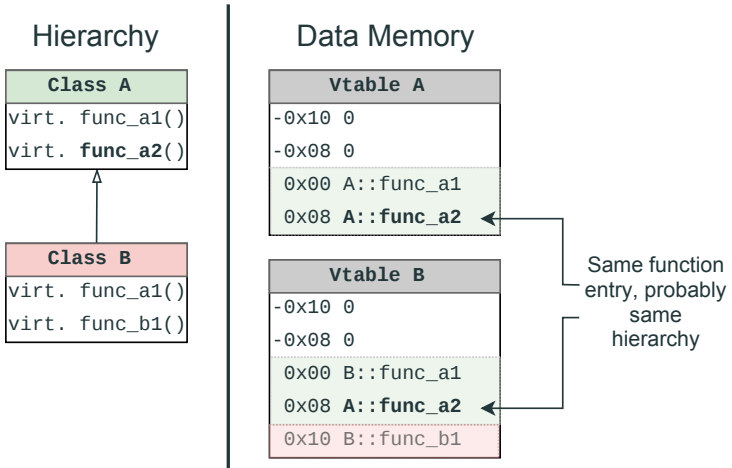
Approach – Function Heuristics



Approach – Function Heuristics



Approach – Function Heuristics

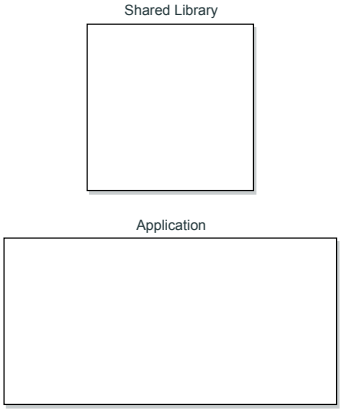


Approach

4. Inter-Modular Dependencies

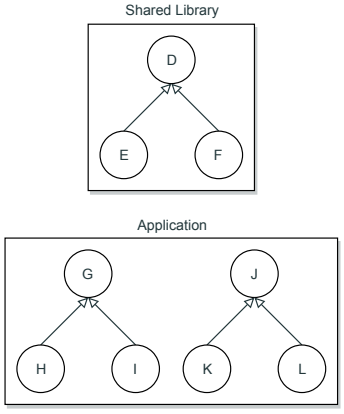
Approach – Inter-Modular Dependencies

Applications may consist of several modules.



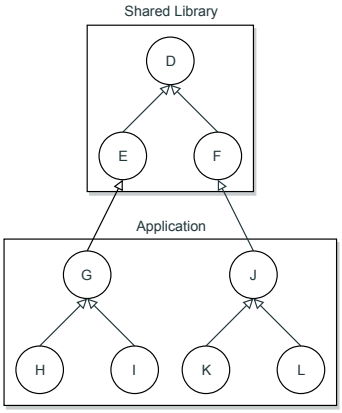
Approach – Inter-Modular Dependencies

Independent analysis yields, e. g., *three* distinct hierarchies.



Approach – Inter-Modular Dependencies

Unless we merge results across modules, we *underestimate* the hierarchy.



Evaluation

Evaluation – Class Hierarchy Reconstruction

Application	No. of Hierarchies	Exactly Reconstructed	Time (h)
MySQL Server	78	69 (88%)	11:36
MongoDB	158	137 (87%)	1:08
Node.js	59	55 (93%)	0:33

Evaluation – Class Hierarchy Reconstruction

Application	No. of Hierarchies	Exactly Reconstructed	Time (h)
MySQL Server	78	69 (88%)	11:36
MongoDB	158	137 (87%)	1:08
Node.js	59	55 (93%)	0:33

Evaluation – Class Hierarchy Reconstruction

Application	No. of Hierarchies	Exactly Reconstructed	Time (h)
MySQL Server	78	69 (88%)	11:36
MongoDB	158	137 (87%)	1:08
Node.js	59	55 (93%)	0:33

48 min analysis time on average for 5 real-world applications.

Evaluation – Class Hierarchy Reconstruction

Application	No. of Hierarchies	Exactly Reconstructed	Time (h)
MySQL Server	78	69 (88%)	11:36
MongoDB	158	137 (87%)	1:08
Node.js	59	55 (93%)	0:33

Application	Overestimated	Underestimated	Not Found
MySQL Server	1 (1%)	7 (9%)	1 (1%)
MongoDB	0	8 (5%)	13 (8%)
Node.js	2 (3%)	2 (3%)	0

48 min analysis time on average for 5 real-world applications.

Evaluation – Binary Hardening

- Use MARX' results to harden **legacy binaries**.

Evaluation – Binary Hardening

- 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?

Evaluation – Precision

- 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

Limitations

- 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).

Limitations

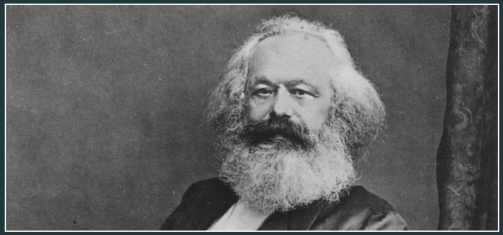
- Approach is **compiler-dependent**, optimizations may break assumptions.
  - Abstract base classes may not yield a vtable for us to discover.
- Analysis is prone to **path explosion** (inter/intra-procedural path generation).
- Applications building upon results have to **tolerate imprecision**.

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.



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`.