

Prohibiting Type Confusion With Inline Type Information

Nicolas Badoux, Flavio Toffalini, Yuseok Jeon & Mathias Payer

EPFL

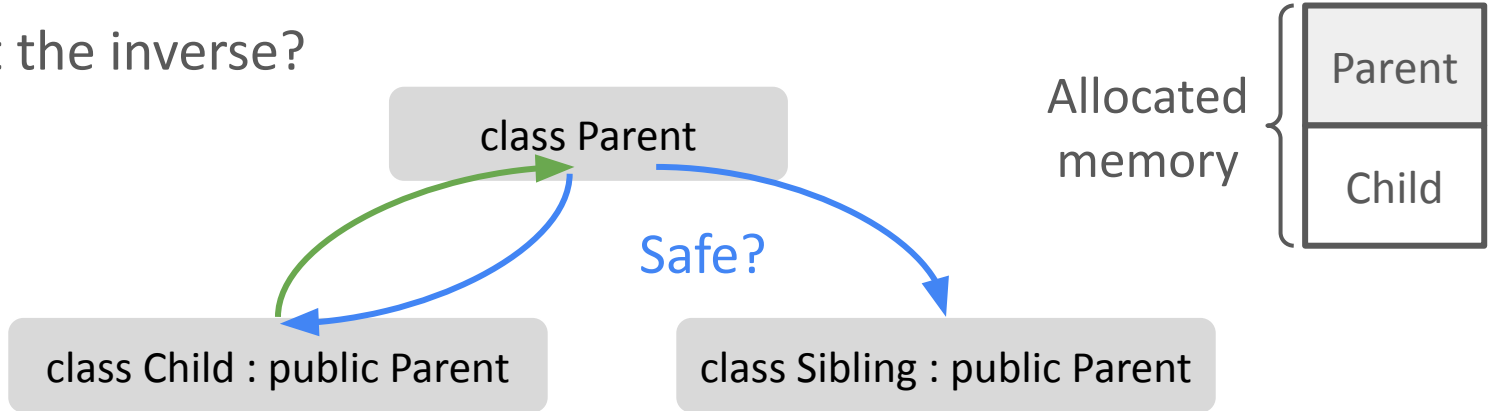


UNIST

Motivation: Derived Type Confusion in C++

Inheritance allows to use a *Child* object as a *Parent* (upcast)

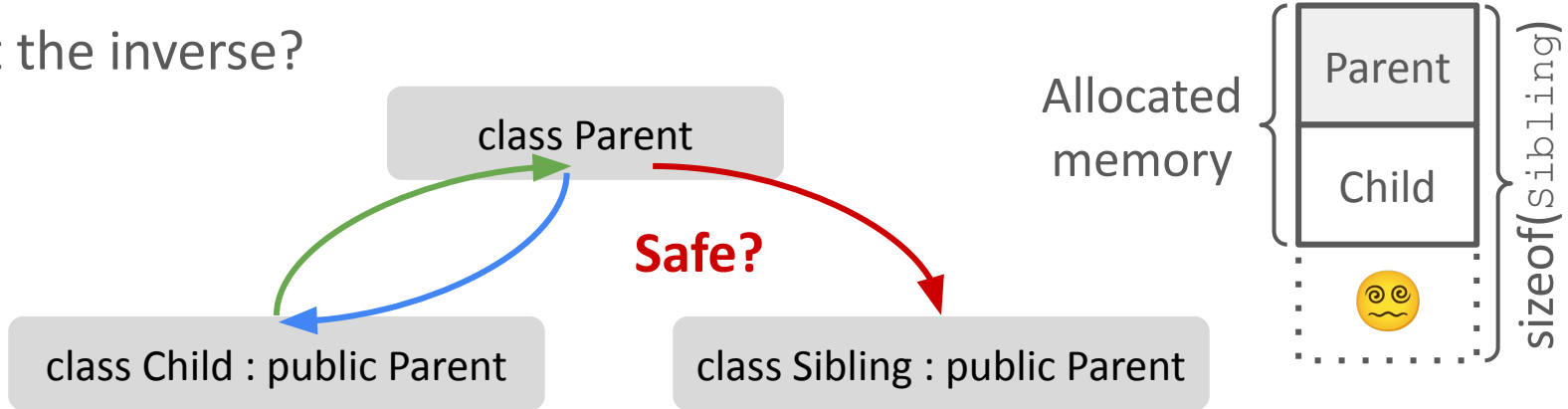
What about the inverse?



Motivation: Derived Type Confusion in C++



Inheritance allows to use a *Child* object as a *Parent* (upcast)

What about the inverse?



Possible with cast operators BUT **not guaranteed correct.**

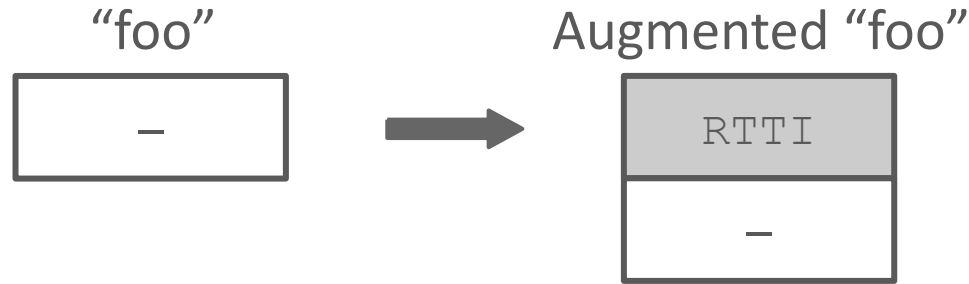
May lead to memory corruption

Still common today, e.g.,  


type++: A C++ Dialect Free of Derived Type Confusion

Goal: Enforce *runtime checks for all casts*

How: Adding inline type information to **all** objects involved in derived cast



Implications:

- Polymorphic types already have Runtime Type Information (RTTI) 
- Changes in object layout
- All the other classes/structs require **initialization**

Automatic Type Information Initialization

Setup RTTI through constructor calls

- Transparently defines a default constructor for all the classes

`new` 

`malloc` & `co`

- Explicit call to the default constructor
- Careful handling of `calloc/realloc`

Allow-list for custom memory allocators (e.g., pool allocator, ASan)

Object Layout: Required Adaptations

Since type++ imposes inlined RTTI for all derived cast classes

Change in layout is **incompatible** with the C++ ABI

Automatic wrappers/macros

- External libraries
- Headers shared with C/C++

Warnings for incompatible code

Limited code adaptations:

- <0.04% of LoC in SPEC CPU

C++ vs type++: Example of Incompatible Idioms

Comparison between `sizeof`:

```
sizeof(X) == 16
```

Implicit `placement_new`:

```
class X { /* other fields */ };  
class Y {  
    char __blob_[sizeof(X)];  
};  
  
...  
X x;  
Y* y = reinterpret_cast<Y*>(&x);
```

Evaluation: Porting Effort

We observed 179 warnings across 16 programs in SPEC CPU2006 & CPU2017

We modified 314 LoC (out of 2M LoC, < 0.04%)

Case study: Blender

Undefined behavior due to tagged pointers (an old-school hack)

```
#define unalignRayFace(o) ((Node *) (((intptr_t)o)|1))  
#define isRayFace(o) (((intptr_t)o)&3) == 1
```



We cannot find RTTI at the unaligned address!

Evaluation: Security & Performance

type++ protects **16x** more casts than the HexType sanitizer

	HexType		LLVM-CFI		type++	
	derived	unrelated	derived	unrelated	derived	unrelated
SPEC CPU2006	5.6B	0	2.1B	0	31B	1.5B
SPEC CPU2017	-	-	1.7B	0	52B	5.5B

Average overhead: **0.94%**, in line with the LLVM-CFI mitigation

	HexType		LLVM-CFI		type++	
	average	max	average	max	average	max
SPEC CPU2006	8.27%	29.21%	0.49%	3.43%	1.19%	4.11%
SPEC CPU2017	-	-	0.33%	3.22%	0.82%	4.58%

Case study: Chromium

We support 92% of Chromium's required classes

- Class support breakdown:

1,102 polymorphic	1,928 ported to type++	171 unsupp.
----------------------	---------------------------	----------------

- 3,339 warnings for 230 LoC changes
- One minor adaptation to `protoc`

JetStream2: 1.42% overhead

89.7% of derived casts protected, double those of LLVM-CFI



type++: Prohibiting Type Confusion With Inline Type Information

 Runtime type information for all classes involved in derived casts

 ABI change resulting in 314 patched LoC (out of 2MLoC)

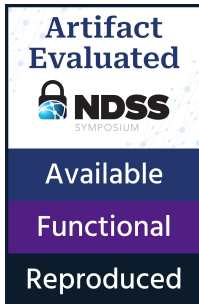
 All derived casts are verified at runtime

 **Less than 1% overhead for 90B casts protected (23x > SotA)**

 **14 new type confusions identified**

 type   paper: hexhive.epfl.ch/

 Artifact: github.com/HexHive/typepp



Security Impact

122 type confusions identified

- 14 new bugs

All have been fixed in more recent software versions

- Use of `dynamic_cast`
- Use of a proper type hierarchy

```
-typedef struct InstanceRayObject {  
-   RayObject rayobj;  
+typedef struct InstanceRayObject : RayObject {  
    RayObject *target;
```