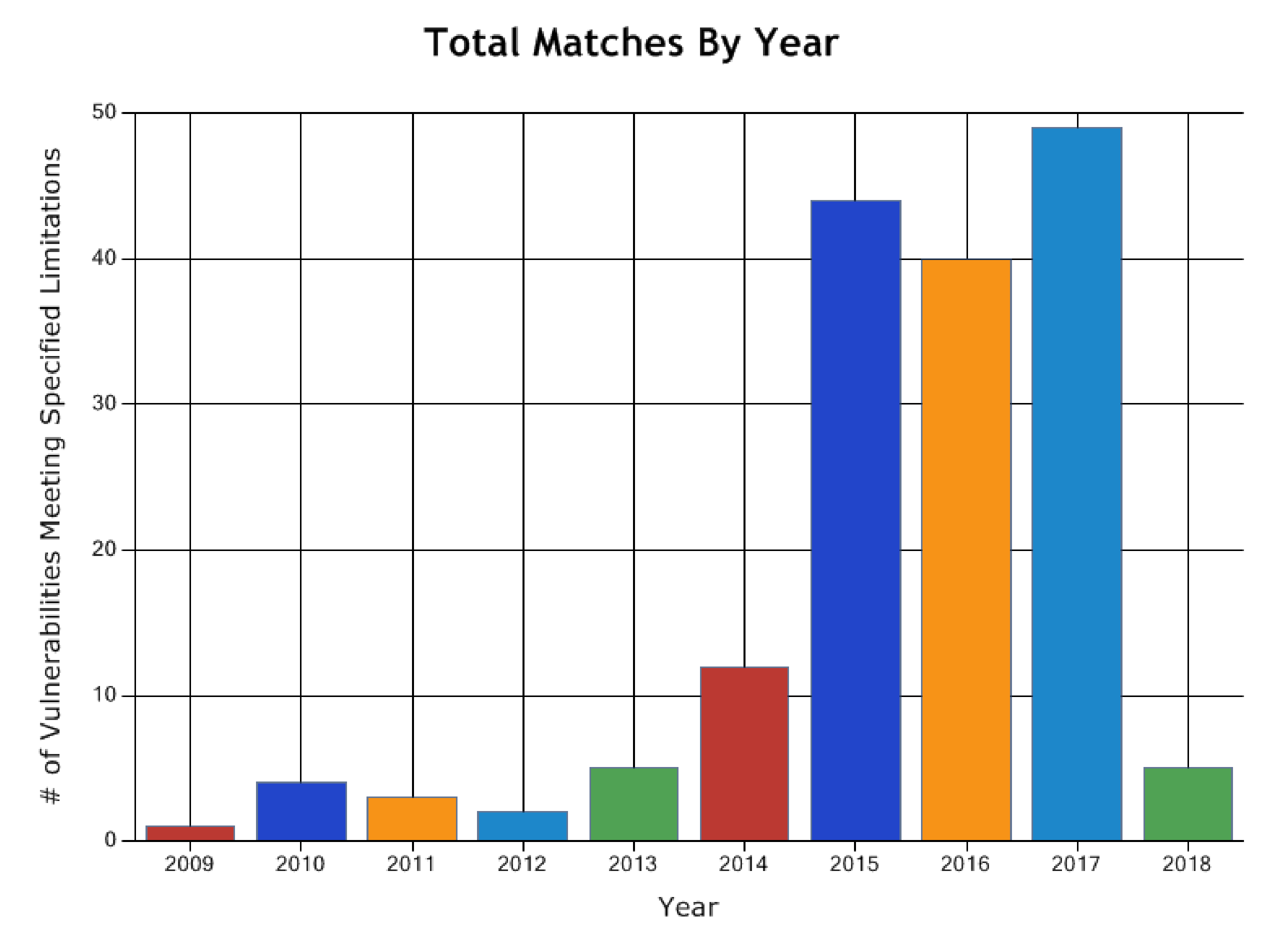
hexhive

# Type Confusion: Discovery, Abuse, Protection

Mathias Payer, @gannimo
http://hexhive.github.io

# Type confusion leads to RCE



# Attack surface is huge

| | |
|---|---|
| Google Chrome: | 76 MLoC |
| Gnome: | 9 MLoC |
| Xorg: | 1 MLoC |
| glibc: | 2 MLoC |
| Linux kernel: | 17 MLoC |

# Control-Flow Hijack Attack (and CFI)

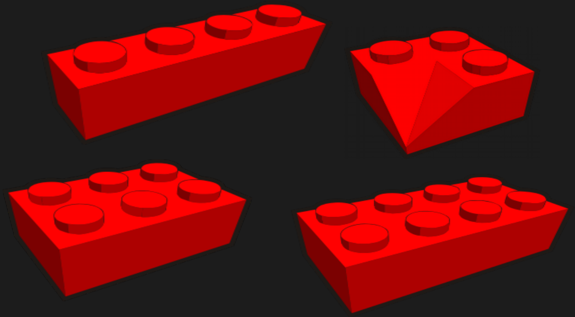
# Problem: broken abstractions?

C/C++

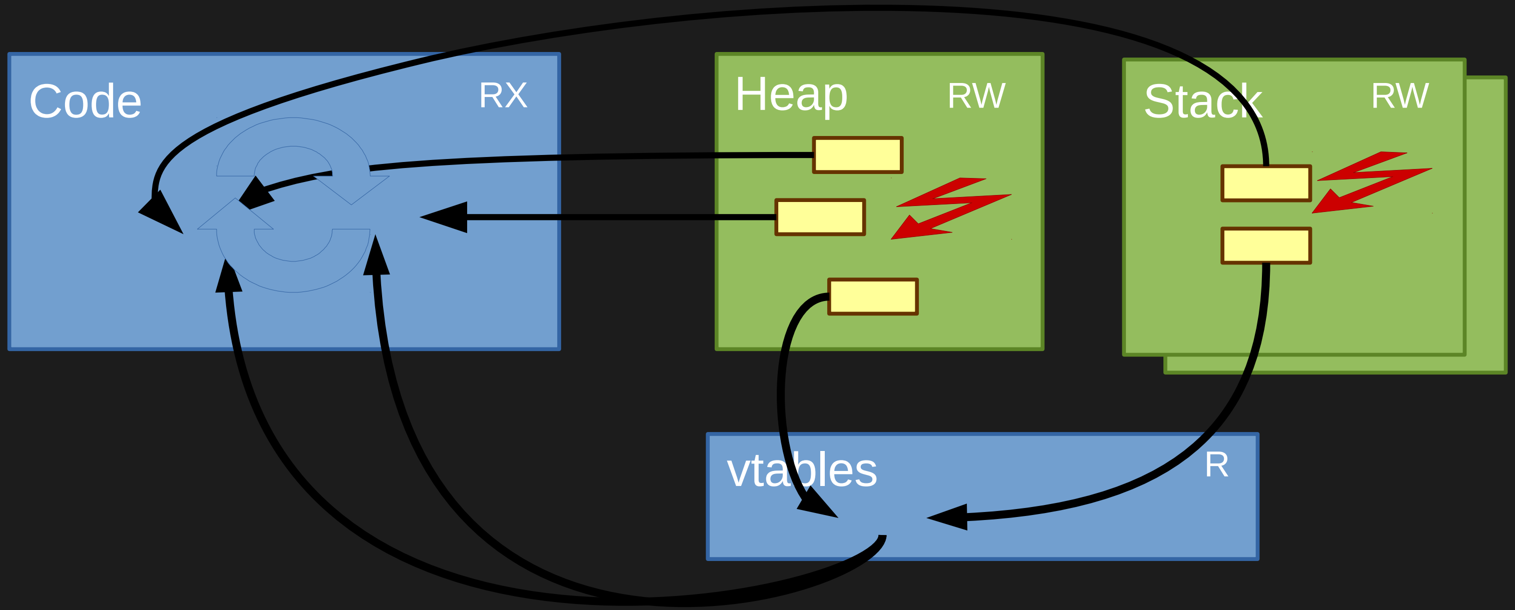
```c
void log(int a) {
  printf("Log: %d", a);
}
void (*fun)(int) = &log;
void init() {
  fun(15);
}
```

ASM

```asm
log:
  ...
fun:
  .quad log
init:
  ...
  movl $15, %edi
  movq  fun(%rip), %rax
  call  *%rax
```



# Attacker model: hijacking control-flow

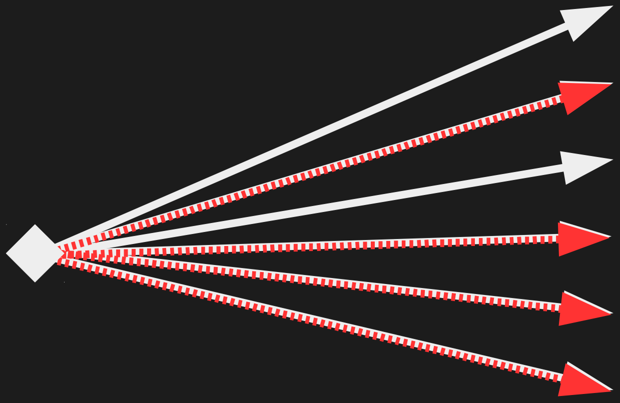


# Control-Flow Integrity (CFI)*

Restrict a program’s dynamic control-flow to the static CFG

- Requires static analysis
- Dynamic enforcement mechanism

```
CHECK(fn);
(*fn)(x);
```



* ***Control-Flow Integrity.*** Martin Abadi, Mihai Budiu, Ulfar Erlingsson, Jay Ligatti. ACM CCS ‘05

**** Control-Flow Integrity: Protection, Security, and Performance.*** Nathan Burow, Scott A. Carr, Joseph Nash, Per Larsen, Michael Franz, Stefan Brunthaler, Mathias Payer. ACM CSUR ‘17

# Class hierarchy depth

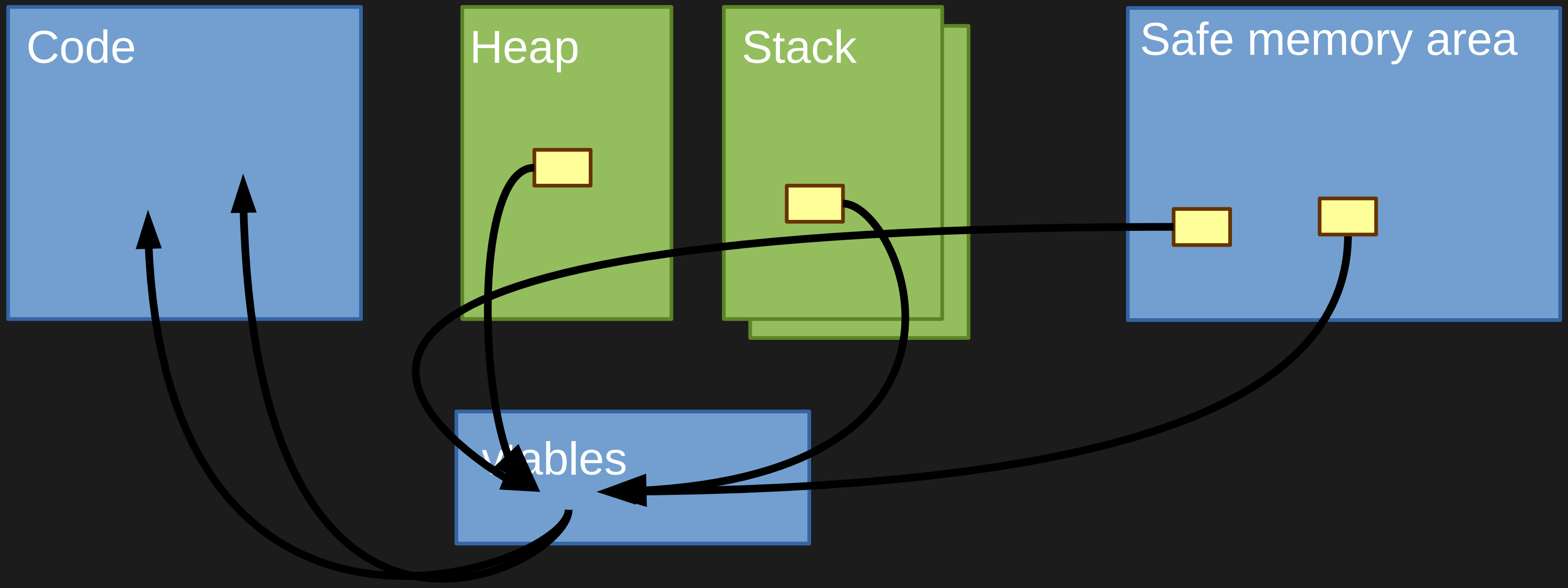
| Impl. Count | Chromium | | Firefox | |
|---|---|---|---|---|
| [1-10] | 13,751 | (99.33%) | 4,632 | (99.90%) |
| >10 | 78 | (0.57%) | 47 | (0.10%) |
| Max | 78 | | 107 | |

CFI prohibits use of corrupted pointer.
Can we do better?

# Object Type Integrity

# Object Type Integrity (OTI)*

Enforce integrity of vtable pointer, use protected dispatch



* ***CFIXX: Object Type Integrity for C++ Virtual Dispatch.*** Nathan Burow, Derrick McKee, Scott A. Carr, and Mathias Payer. In ISOC NDSS '18

# CFIXX instrumentation

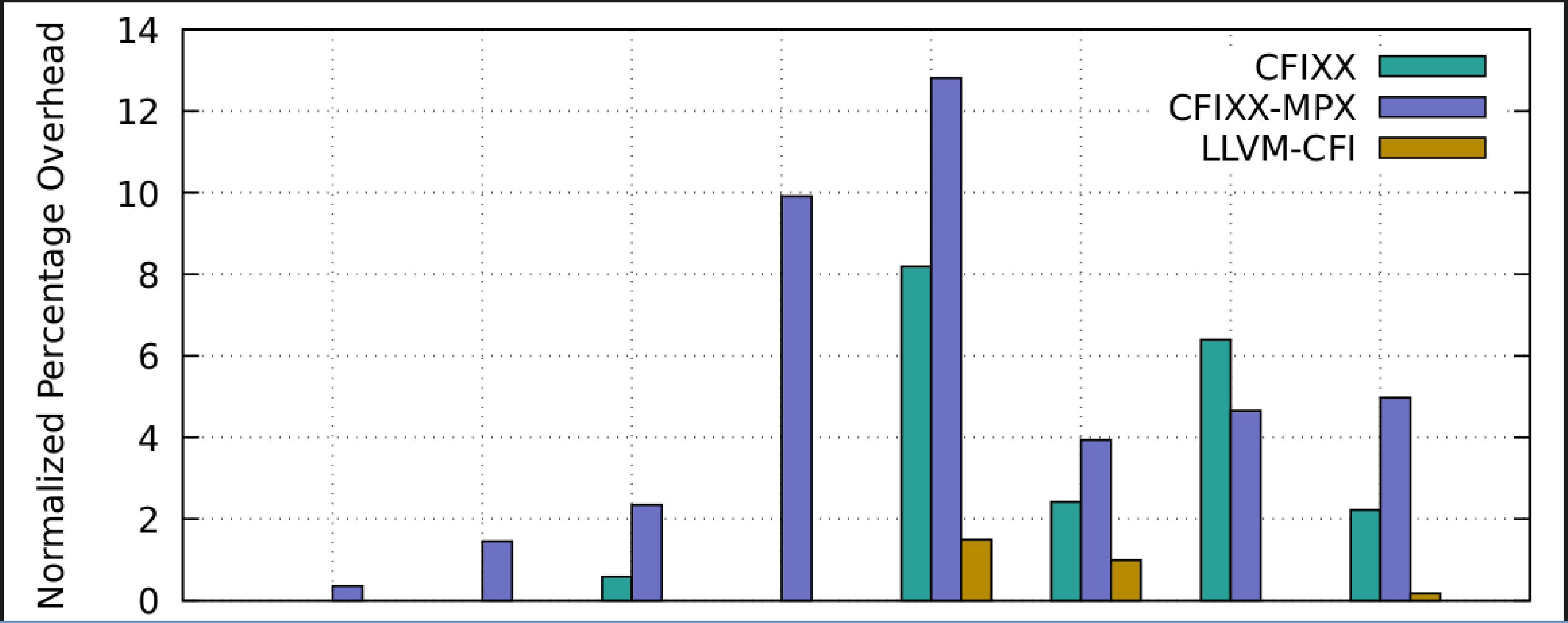
C++ dynamic dispatch has single target

- Only constructor allowed to write vtable pointer
- Deallocation invalidates vtable pointer
- Dispatch uses vtable pointer

Enforcing OTI protects against

- VTable injection even with correct method signature
- Swap vtable even in the same hierarchy
- Fake object creation (COOP)

# CFIXX performance



Chromium: 2.03% (Octane), 1.99% (Kraken)

# CFI and CFIXX summary

CFI makes attacks harder

- Effectiveness depends on analysis and complexity
- Deployed in Microsoft Edge, Google Chrome on Linux
- Limitation: large equivalence classes

Object Type Integrity (CFIXX)

- Protect object instead of dispatch
- Single valid target per object

Future direction: type check / overhead

Source: https://github.com/HexHive/CFIXX

LEFT
EXIT 12
GUARDING APPS AGAINST BUGS
FUZZING FOR THAT ONE ODAY
imgflip.com

# C++ Casting

# C++ casting operations

`dynamic_cast<ToClass>(Object)`

- Runtime check based on allocated type (vtable pointer)
- Not used in performance critical code

`static_cast<ToClass>(Object)`

- Class hierarchy check at compile time

`(ToClass)(Object)`

- C-style cast, no check at all

# Static cast

```
a = static_cast<Greeter*>(b);
```

```
movq  -24(%rbp), %rax    # Load pointer
                         # Type “check”
movq  %rax, -40(%rbp)    # Store pointer
```

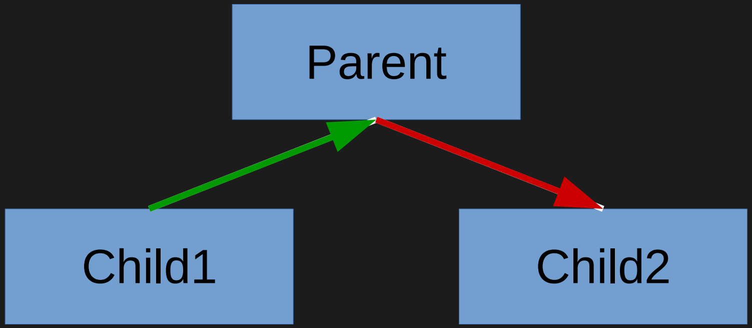
# Dynamic cast, optimized

```
a = dynamic_cast<Greeter*>(b);
```

```
leaq  _ZTI7Greeter(%rip), %rdx
leaq  _ZTI4Base(%rip), %rsi
xorl  %ecx, %ecx
movq  %rbp, %rdi                # Load pointer
call  __dynamic_cast@PLT        # Type check
```

# Type Confusion

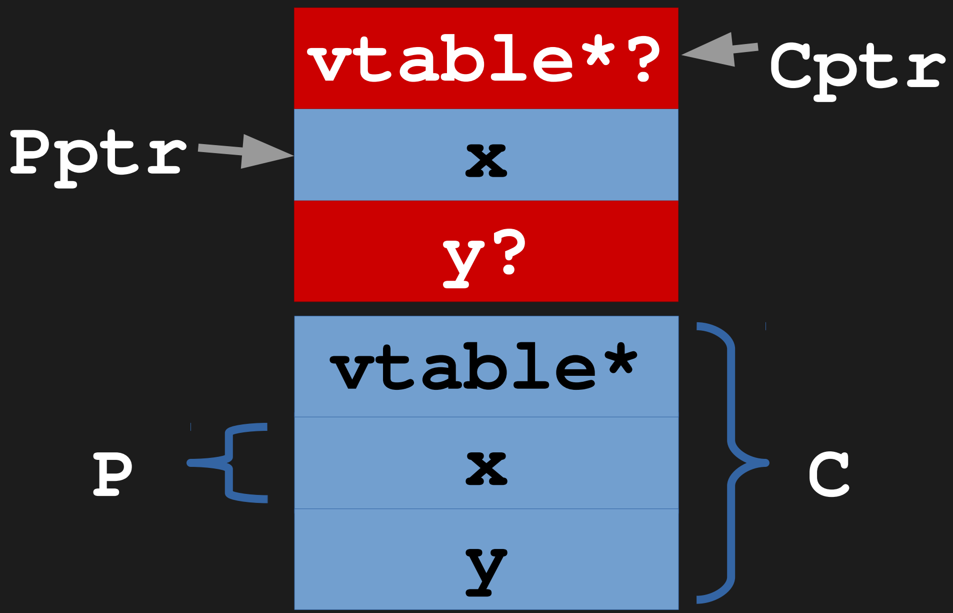
# Type confusion arises through illegal downcasts



```
Child1 *c = new Child1();
Parent *p = static_cast<Parent*>(c); ✓
Child2 *d = static_cast<Child2*>(p); ✗
```

# Type confusion

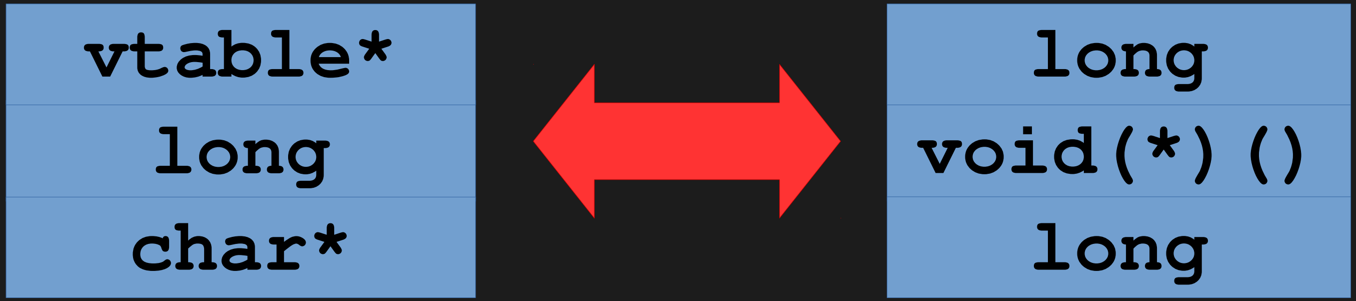
```
class P {
   int x;
};
class C: P {
   int y;
   virtual void print();
};
…
P *Pptr = new P;
C *Cptr = static_cast<C*>Pptr; // Type Conf.
Cptr->y = 0x43; // Memory safety violation!
Cptr->print();  // Control-flow hijacking
```



# Exploit primitive

Control two pointers of different types to single memory area

Different interpretation of fields leads to “opportunities”



vtable*
long
char*

long
void(*)()
long

https://googleprojectzero.blogspot.ch/2015/07/one-perfect-bug-exploiting-type_20.html
https://blogs.technet.microsoft.com/mmpc/2015/06/17/understanding-type-confusion-vulnerabilities-cve-2015-0336/

# Searching for type confusion bugs: SEGFAULT

# Type Sanitization

# Type safety for C++

A static cast is checked only at compile time

- Fast but no runtime guarantees

Dynamic casts are checked at runtime

- High overhead, limited to polymorphic classes

Our core idea:

- Conceptually check *all* casts dynamically
- Aggressively optimize design and implementation

* TypeSanitizer: Practical Type Confusion Detection. Istvan Haller, Yuseok Jeon, Hui Peng, Mathias Payer, Herbert Bos, Cristiano Giuffrida, Erik van der Kouwe. In CCS'16
* HexType: Efficient Detection of Type Confusion Errors for C++. Yuseok Jeon, Priyam Biswas, Scott A. Carr, Byoungyoung Lee, and Mathias Payer. In CCS'17

# Making type checks explicit

Enforce runtime check at all cast sites

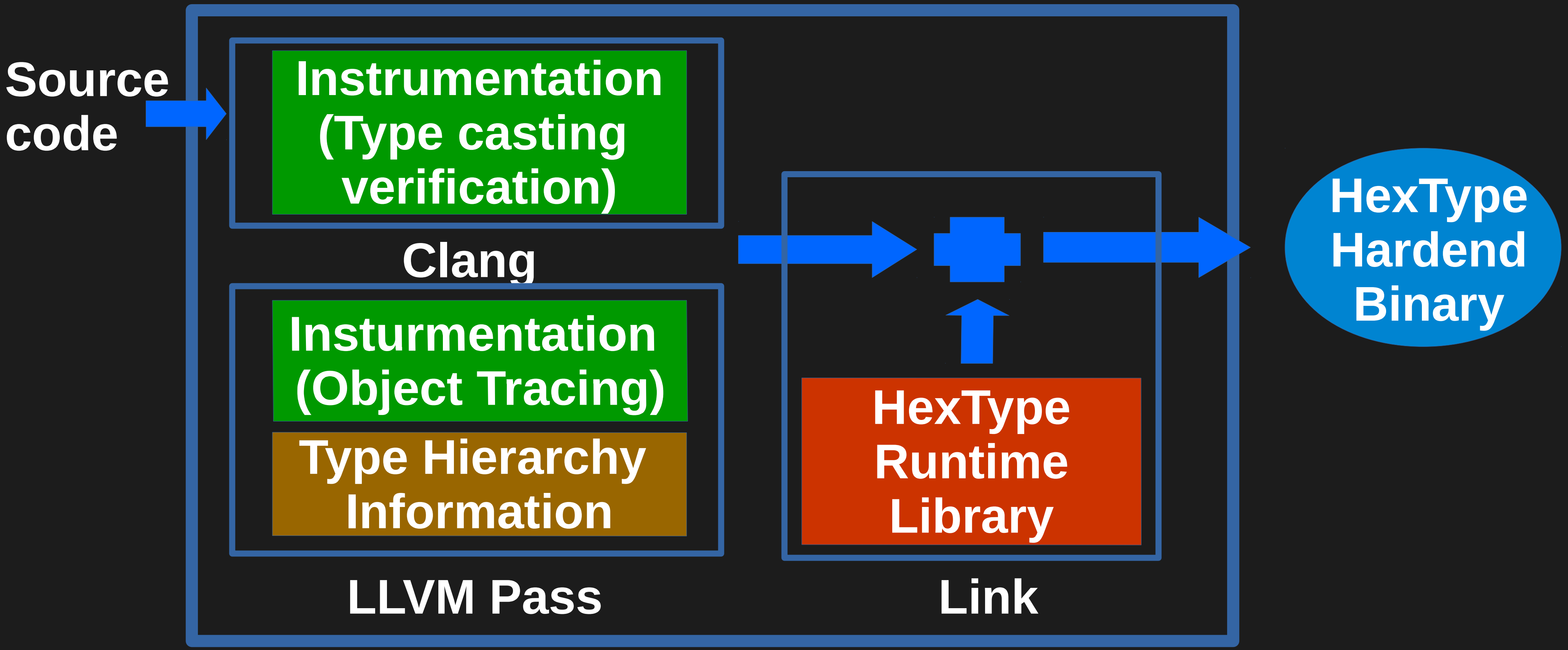
- `static_cast<ToClass>(Object)`
- `dynamic_cast<ToClass>(Object)`
- `reinterpret_cast<ToClass>(Object)`
- `(ToClass)(Object)`

Build global type hierarchy

Keep track of the allocation type of each object

- Must instrument all forms of allocation
- Requires disjoint metadata

# HexType: design


Source code
Instrumentation (Type casting verification)
Clang
Insturmentation (Object Tracing)
Type Hierarchy Information
LLVM Pass
HexType Runtime Library
Link
HexType Hardend Binary

# HexType: go full coverage!

Cover “**`new`**” object allocations

- Obscure allocation cases for, e.g., arrays, stack

Support **`placement_new`**

- Custom allocators don’t call malloc/new

Support **`reinterpret_cast`**

- Repurpose and revive existing objects

# HexType: aggressive optimization

Limit tracing to unsafe types

- Remove tracing of types that are never cast

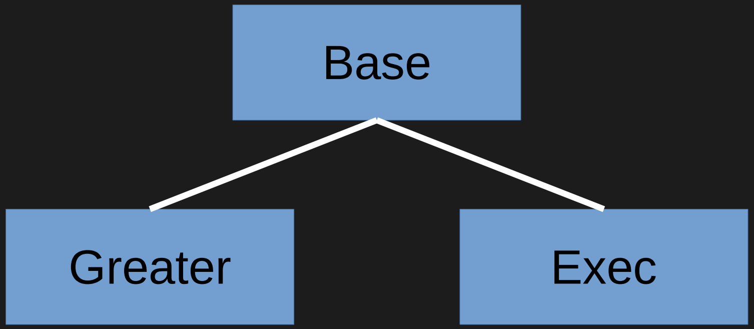
Limit checking to unsafe casts

- Remove statically verifiable casts

No more RTTI for dynamic casts

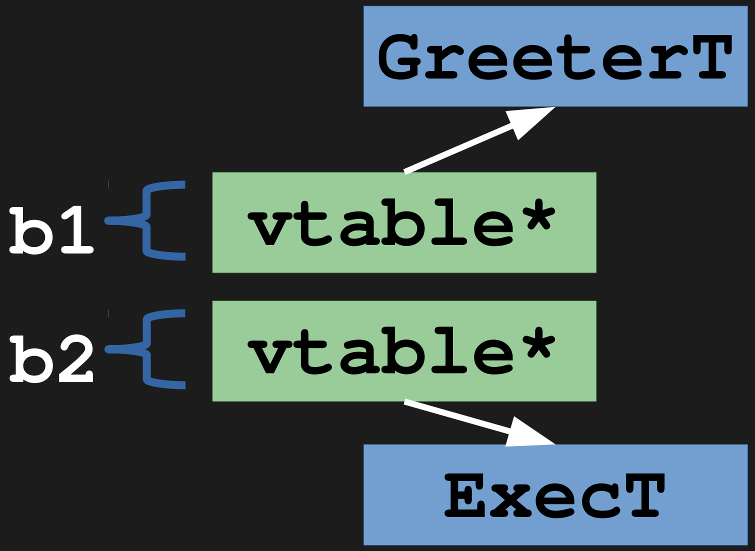
- Replace dynamic casts with fast lookup

# Simple exploitation demo



```
class Base { … };

class Exec: public Base {
  public:
    virtual void exec(const char *prg) {
      system(prg);
    }
};

class Greeter: public Base {
  public:
    virtual void sayHi(const char *str) {
      std::cout << str << std::endl;
    }
};
```

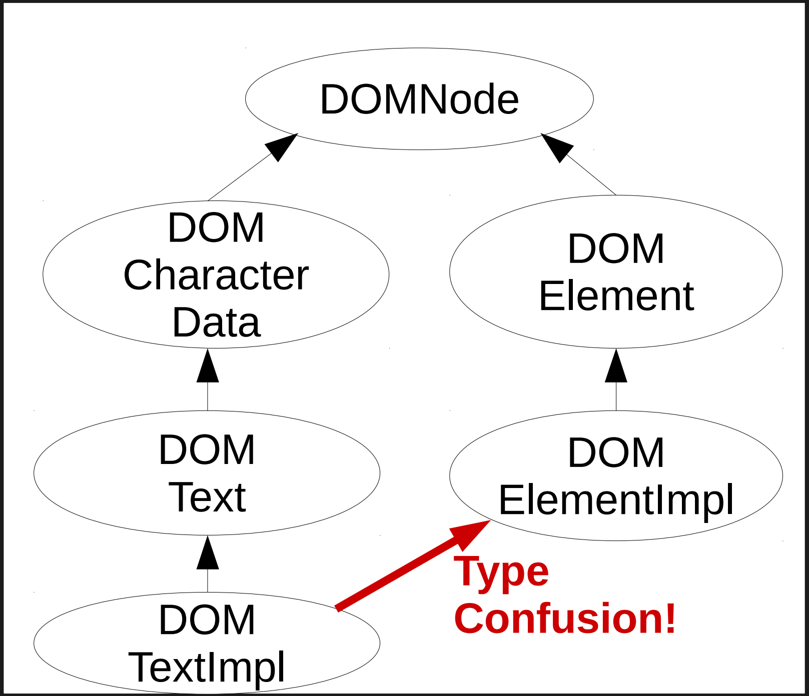
# Simple exploitation demo



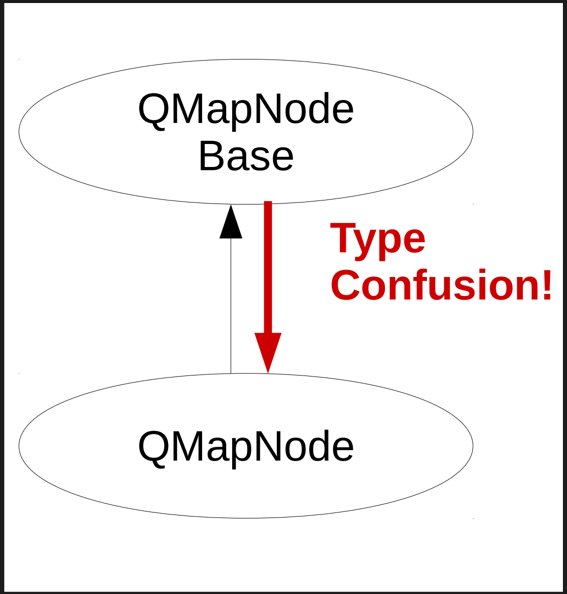
```
int main() {
  Base *b1 = new Greeter();
  Base *b2 = new Exec();
  Greeter *g;

  g = static_cast<Greeter*>(b1);
  g->sayHi("Greeter says hi!");   // g[0][0](str);

  g = static_cast<Greeter*>(b2);
  g->sayHi("/usr/bin/xcalc");     // g[0][0](str);

  delete b1;
  delete b2;
  return 0;
}
```

# Low hanging fruits: four new vulnerabilities

## Apache Xerces C++



## Qt base library



# Fuzz all the Things!

# Combine AFL/libFuzzer with HexType

AFL/libFuzzer and HexType play surprisingly well together

- Compile software with HexType, trap on type confusion
- Let fuzzing do its magic
- Triage type confusion reports
- $$$

TP-LINK
V·I·V·O
V·I·V·O
V·I·V·O
V·I·V·O
V·I·V·O

# First two weeks of fuzzing

QTcore: two new type confusion bugs (not exploitable, fixed)

Xerces C++: one new type confusion (reported, fixed)

# One more week of fuzzing with libFuzzer

ChakraCore: two cases of type confusion (reported)

MySQL 5.7: five cases of type confusion (reported, serious)

Other C++ software: evaluation ongoing

- Let us know what we should target next
- Have spare fuzzing capacity? Let’s team up!

# But what about Firefox?

| | |
|---|---|
| FF-Octane: | 5,506,850 type confusion reports |
| FF-Dramaeo-JS: | 15,216,798 type confusion reports |
| FF-Dramaeo-dom: | 7,240,272,959 type confusion reports |

Large amount of duplicates and false positives

- Firefox code is messy, few actual bugs but lots of code smell

# Conclusion

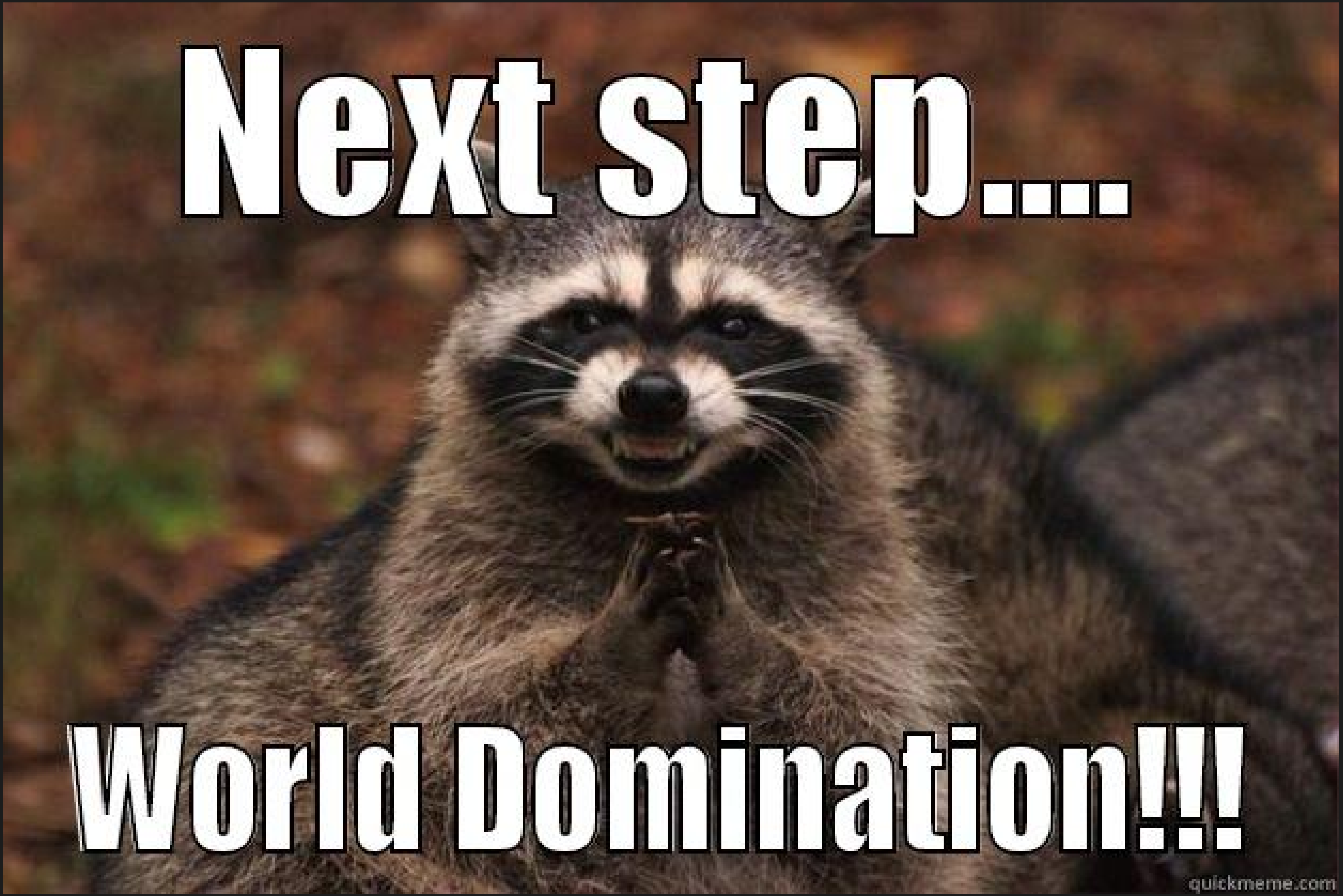
# Ongoing work

Fuzz all the things!

- More software, better test cases, deeper coverage

Selective fuzzing

- Select which types to test (DOM anyone?)
- Extend type check to dereference


Next step.....
World Domination!!!
quickmeme.com

# Conclusion

Type confusion is fundamental in today’s exploits

Existing solutions are incomplete, partial, slow

HexType: type sanitizer for C++

- Trap upon type confusion, not memory safety violation
- Reasonable slowdown for testing (~50%)
- Integrated with AFL/libFuzzer for broad bug discovery

https://github.com/HexHive/HexType

Twitter: @gannimo