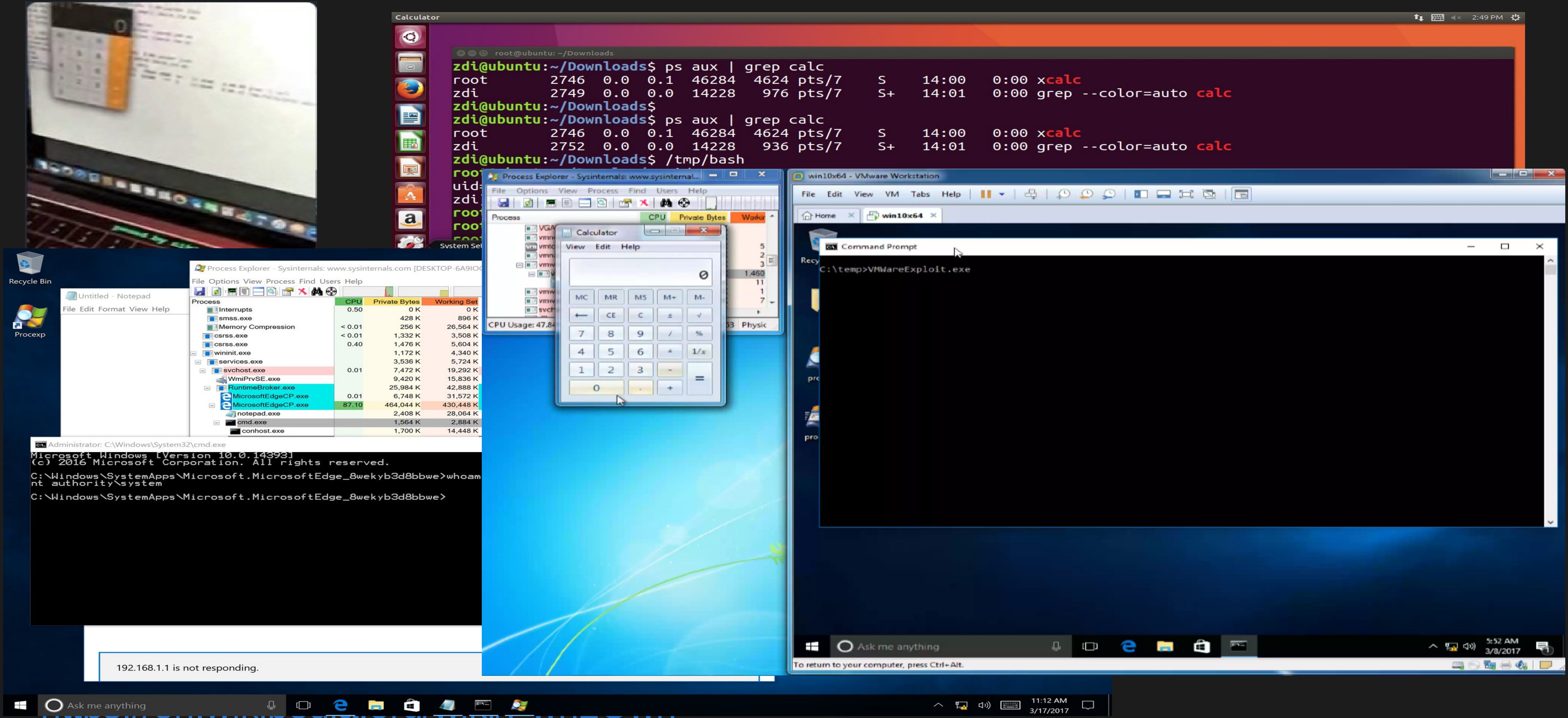
hexhive

Type Confusion: Discovery, Abuse, Protection

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

Type confusion leads to RCE



```
zdi@ubuntu:~/Downloads$ ps aux | grep calc
root       2746  0.0  0.1  46284  4624 pts/7    S    14:00   0:00 xcalc
zdi        2749  0.0  0.0  14228   976 pts/7    S+   14:01   0:00 grep --color=auto calc
zdi@ubuntu:~/Downloads$
zdi@ubuntu:~/Downloads$ ps aux | grep calc
root       2746  0.0  0.1  46284  4624 pts/7    S    14:00   0:00 xcalc
zdi        2752  0.0  0.0  14228   936 pts/7    S+   14:01   0:00 grep --color=auto calc
zdi@ubuntu:~/Downloads$ /tmp/bash
```

```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Windows\SystemApps\Microsoft.MicrosoftEdge_8wekyb3d8bbwe>whoami
nt authority\system

C:\Windows\SystemApps\Microsoft.MicrosoftEdge_8wekyb3d8bbwe>
```

192.168.1.1 is not responding.

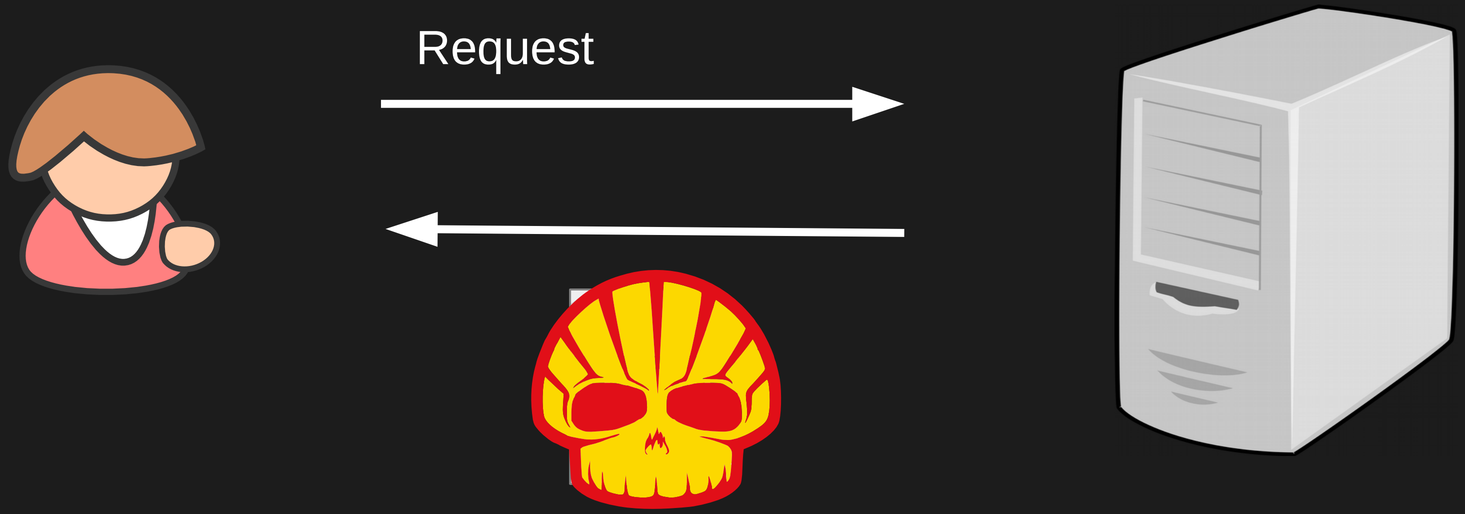
```
C:\temp>VMWareExploit.exe
```

Calculator
root@ubuntu: ~/Downloads
Process Explorer - Sysinternals: www.sysinternals.com
win10x64 - VMware Workstation
Command Prompt
Administrator: C:\Windows\System32\cmd.exe
Recycle Bin
Procexp
Untitled - Notepad
Ask me anything
To return to your computer, press Ctrl+Alt.
2:49 PM
11:12 AM 3/17/2017
5:52 AM 3/8/2017

Attack surface is huge

Google Chrome:	**76 MLoC**
Gnome:	**8.6 MLoC**
Xorg:	**1 MLoC**
glibc:	**1.5 MLoC**
Linux kernel:	**14 MLoC**
Total:	**>100 MLoC**

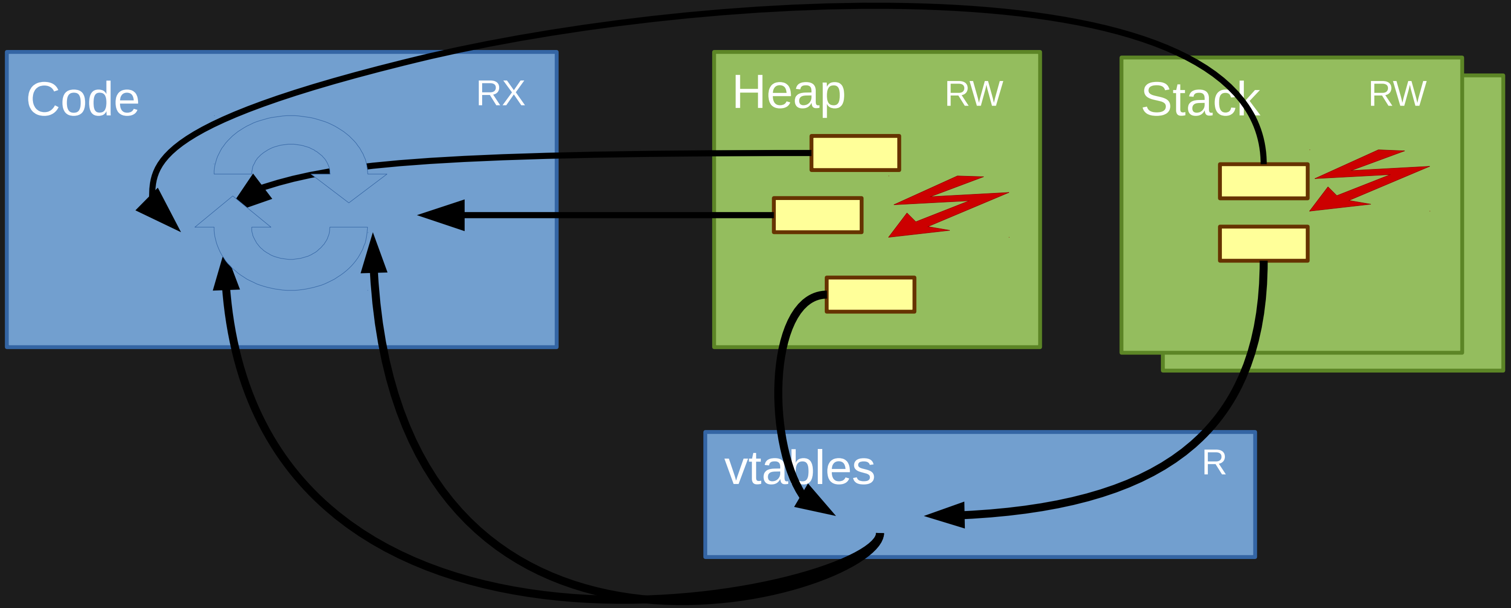
Attacker model



External user ⟶ User ⟶ Administrator

Control-Flow Hijack Attack

Attacker model: hijacking control-flow



C++ Casting

C++ casting operations

`static_cast<ToClass>(Object)`

- Compile time check
- No runtime type information

`dynamic_cast<ToClass>(Object)`

- Runtime check
- Requires Runtime Type Information (RTTI)
- Not used in performance critical code

Static cast, O0

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

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

Dynamic cast, O0

```
a = dynamic_cast<Greeter*>(b);
movq  -24(%rbp), %rax           # Load pointer
testq %rax, %rax                # Null check
je   .L7
movl  $0, %ecx
leaq  _ZTI7Greeter(%rip), %rdx
leaq  _ZTI4Base(%rip), %rsi
movq  %rax, %rdi
call  __dynamic_cast@PLT        # Type check
jmp .L8
.L7:
movl  $0, %eax
.L8:
movq  %rax, -40(%rbp)           # Store pointer
```

Dynamic cast, optimized

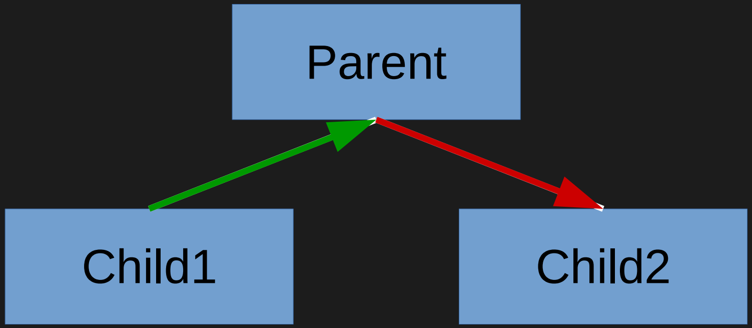
```
a = dynamic_cast<Greeter*>(b);
leaq  _ZTI7Greeter(%rip), %rdx
leaq  _ZTI4Base(%rip), %rsi
xorl  %ecx, %ecx
movq  %rbp, %rdi                     # Load pointer
call  __dynamic_cast@PLT             # Type check
```

Static cast, optimized

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

Type Confusion

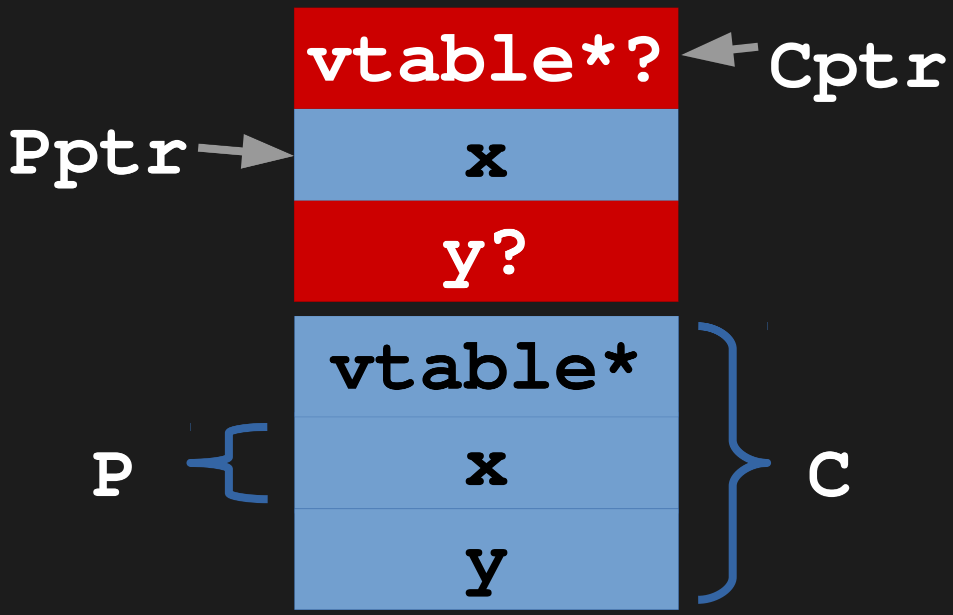
Type confusion arises through illegal downcasts



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

Type confusion

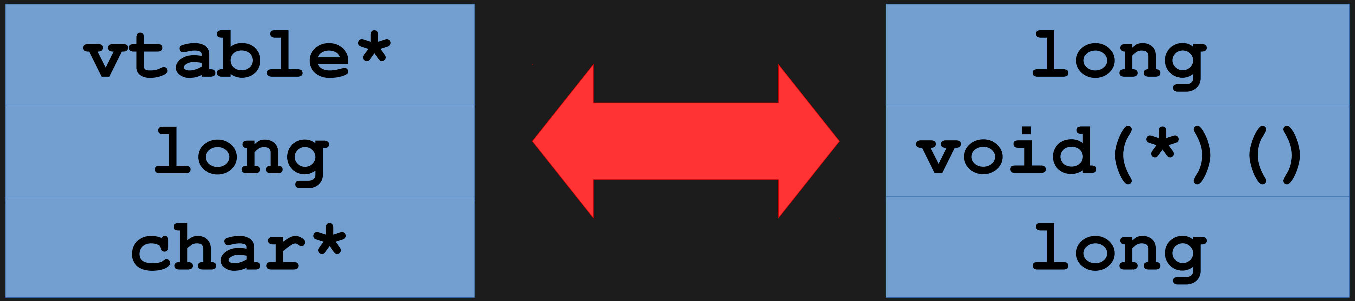
```
class P {
   int x;
};
class C: P {
   int y;
   virtual void print();
};
…
P *Pptr = new P;
C *Cptr = static_cast<C*>Pptr; // Type Conf.
Cptr->y = 0x43; // Memory safety violation!
Cptr->print();  // Control-flow hijacking
```



Exploit primitive

Control two pointers of different types to single memory area

Different interpretation of fields leads to “opportunities”



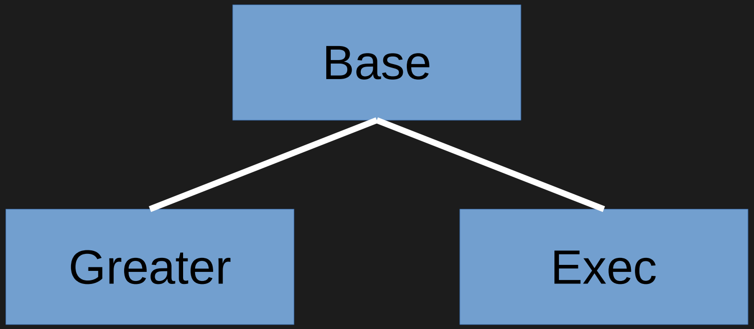
https://googleprojectzero.blogspot.ch/2015/07/one-perfect-bug-exploiting-type_20.html
https://blogs.technet.microsoft.com/mmpc/2015/06/17/understanding-type-confusion-vulnerabilities-cve-2015-0336/

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;
    }
};
```



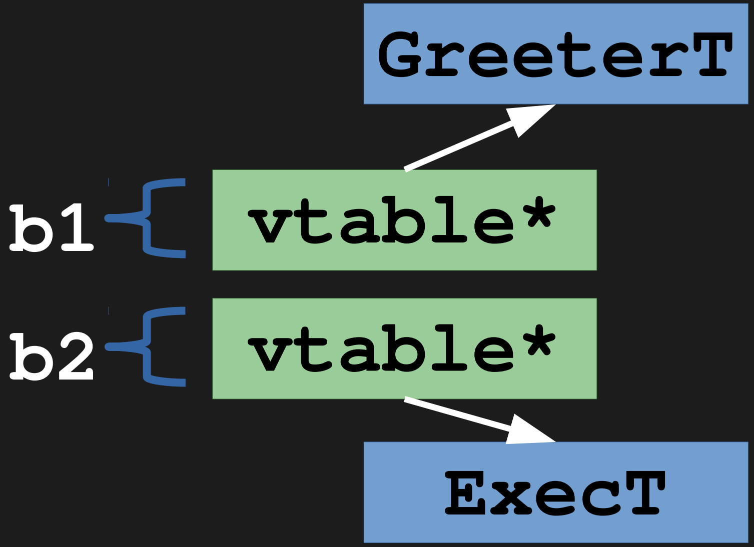
Simple exploitation demo

```
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;
}
```



Searching for type confusion bugs: SEGFAULT

Type Safety

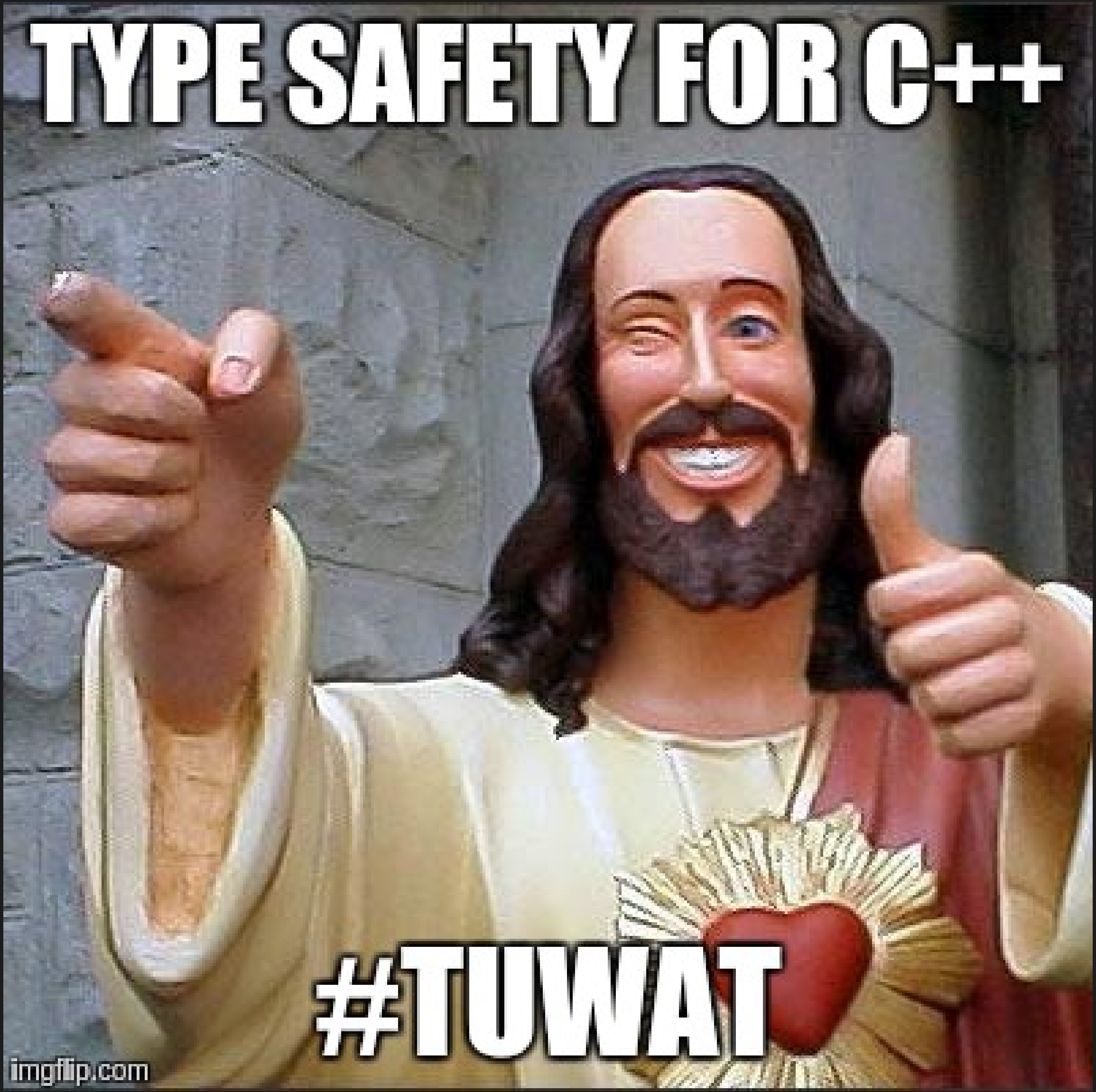
Type confusion detection

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


TYPE SAFETY FOR C++
#TUWAT
imgflip.com

Type confusion detection

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

HexType design:

- 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

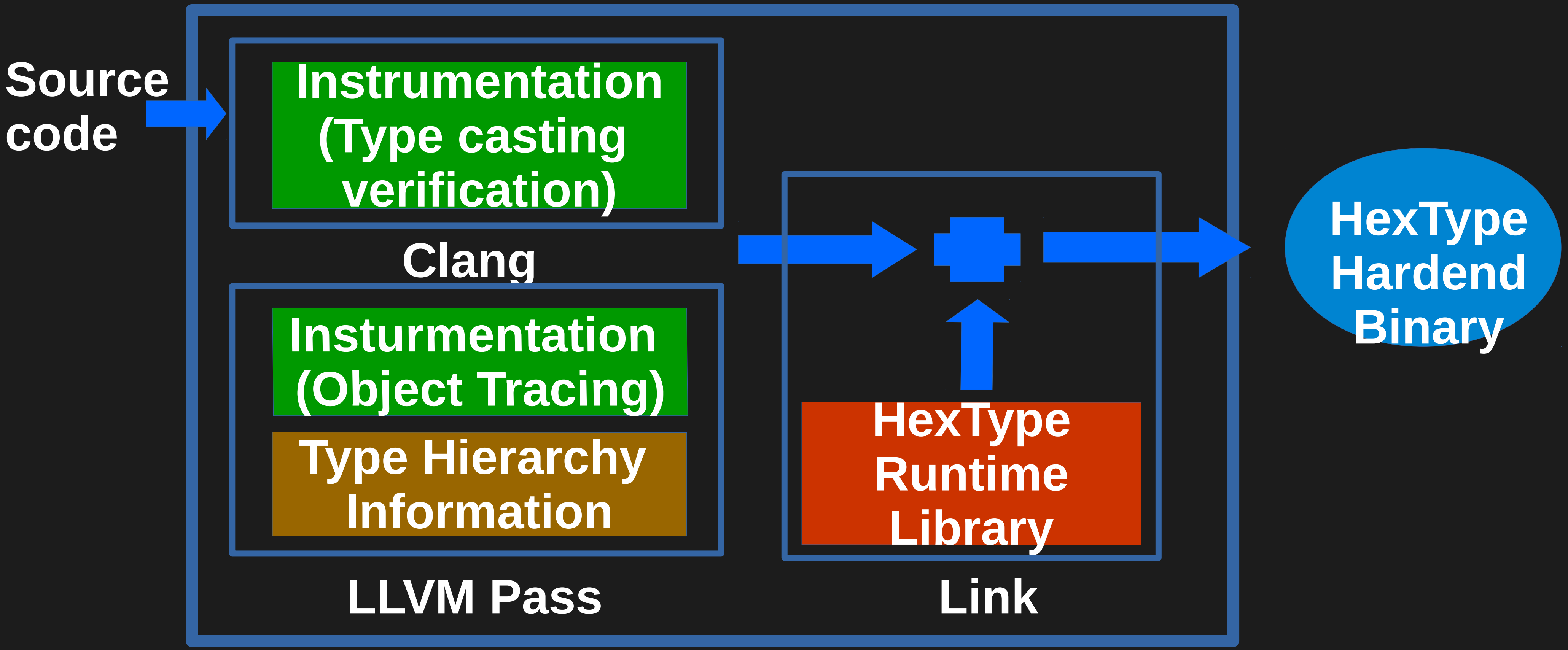
- `static_cast<ToClass>(Object)`
- `dynamic_cast<ToClass>(Object)`
- `reinterpret_cast<ToClass>(Object)`
- `(ToClass)(Object)`

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

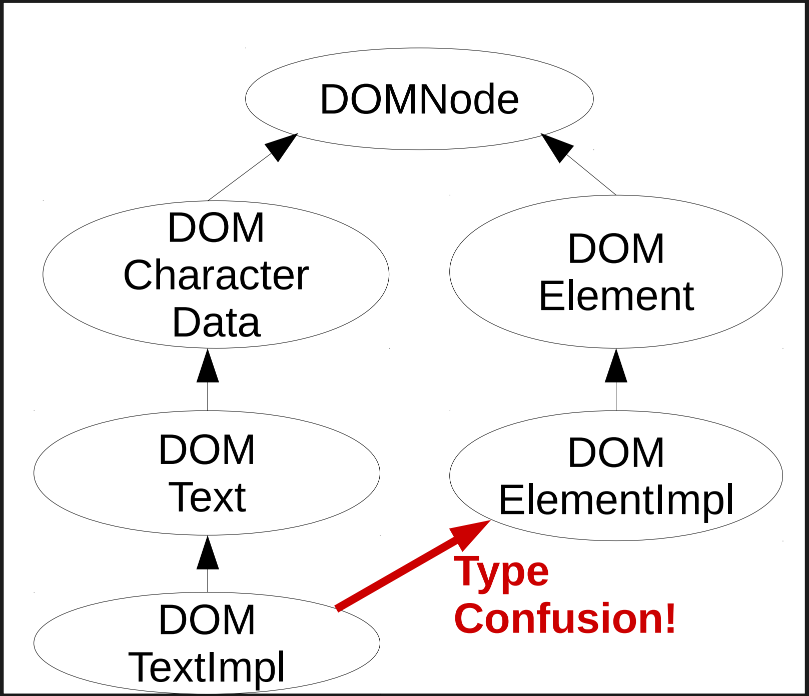
- Remove statically verifiable casts

No more RTTI for dynamic casts

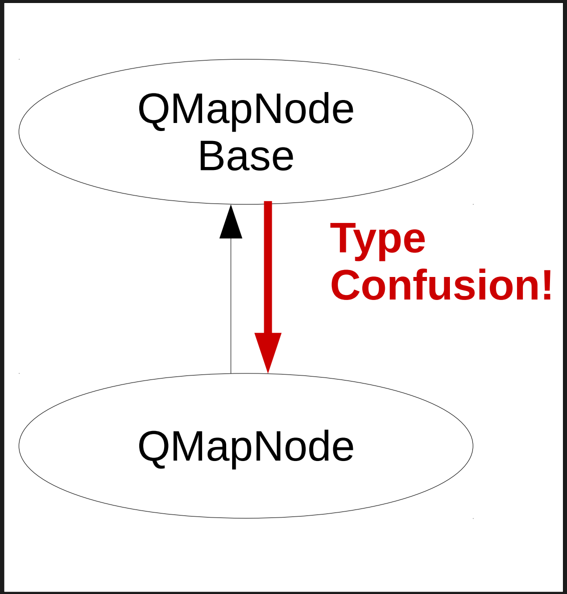
- Replace dynamic casts with fast lookup

Low hanging fruits: four new vulnerabilities

Apache Xerces C++



Qt base library



Fuzz all the Things!

Combine AFL with HexType

AFL and HexType play surprisingly well together

- Compile software with HexType, trap on type confusion
- Let AFL 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

Two weeks of fuzzing

QTcore: two new type confusion bugs (not exploitable)

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

Libsass: 7 reports (triaging in progress)

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

- We are working hard on triaging
- Firefox code is messy...

Conclusion

Future/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

Always on checks for polymorphic objects

- Enforce type integrity at low overhead

Conclusion

Type confusion fundamental in today's exploits

Existing solutions are incomplete, partial, slow

HexType

- Trap upon type confusion, not memory safety violation
- Reasonable overhead (Firefox: 0-0.5x slowdown)
- Integrated with AFL for broad bug discovery

https://github.com/HexHive/HexType

Twitter: @gannimo