CFIXX: Object Type Integrity

Nathan Burow, Derrick McKee, Scott A. Carr, Mathias Payer

PURDUE UNIVERSITY

hexhive

Control-Flow Hijacking Attacks

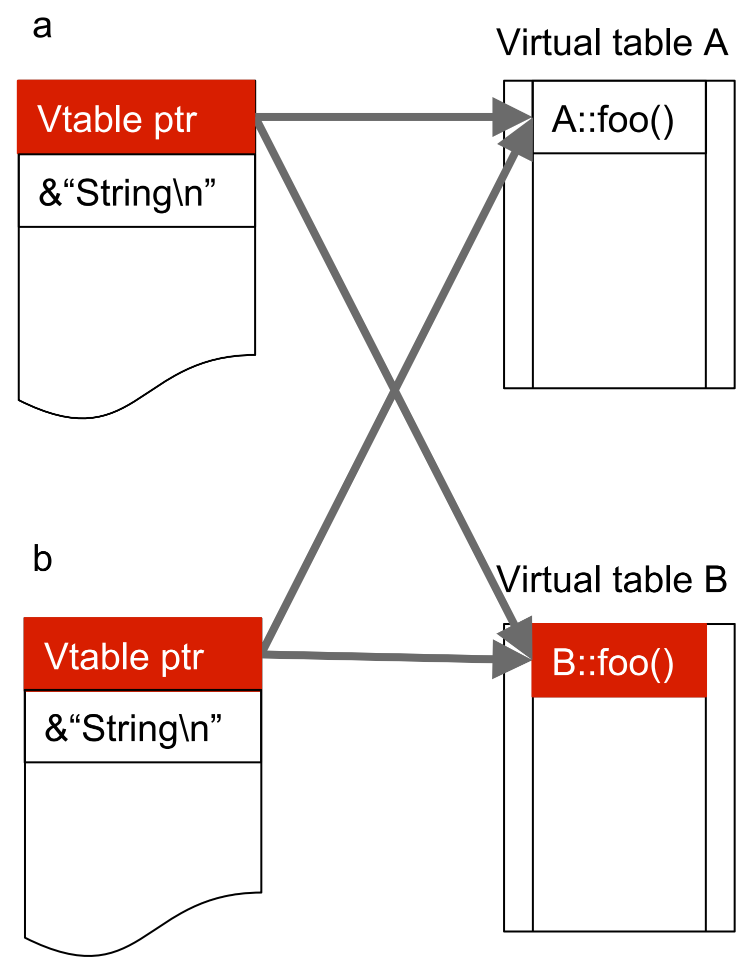
- C / C++ are ubiquitous and insecure
  - Browsers: Chrome, Firefox, Internet Explorer
  - Servers: Apache, memcached, MySQL, NodeJS
- 14,646 code execution CVEs in 2017 alone
- Allow attackers to control ***your*** systems

C++ Vulnerabilities

- Modern control-flow hijacks target indirect control-flow transfers
  - Back edges (returns) are symmetric -- defender knows correct target
  - Forward edges (indirect calls) are harder to protect
- C++ virtual calls have strict semantics at language level
  - Virtual calls rely on the object's ***allocated*** type
  - Virtual calls map to indirect calls, losing semantic information
  - Attackers can change the type associated with an object

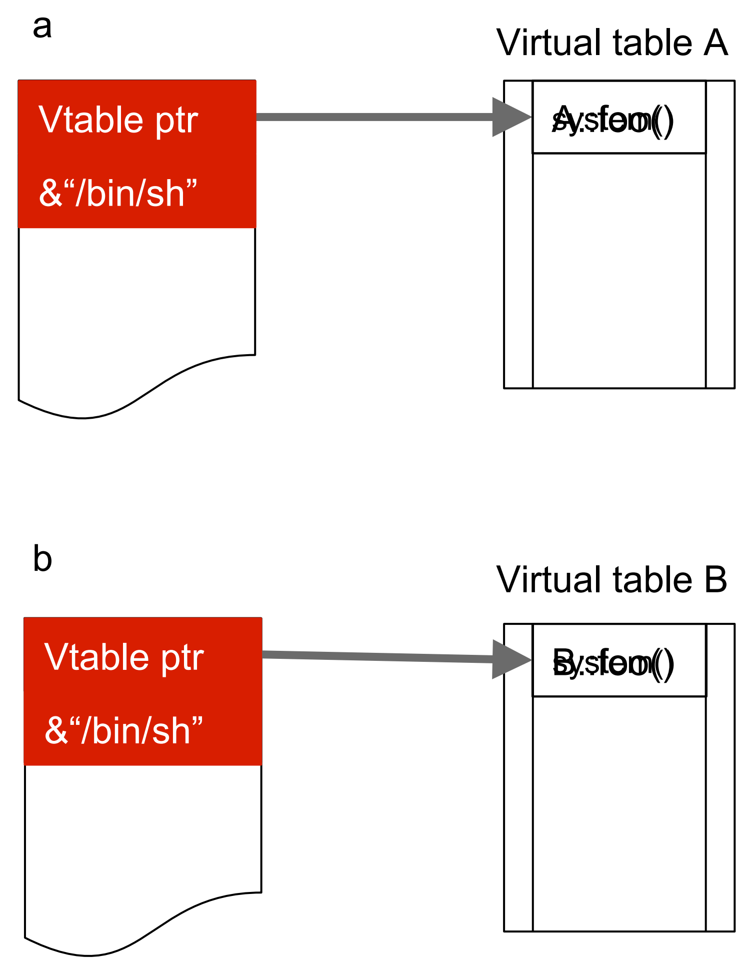
Class Hierarchy Attack

```
class A {
  char *s;
  virtual void foo(char *s) { … }
};
class B : public A {
  void foo(char *s) override { … }
};
void dispatch(A *a){
  a->foo(a->s);
}
int main(int argc, char **argv){
  A *a = new A("String\n");
  B *b = new B("String\n");

  // Arbitrary write for attacker
  vuln();

  dispatch(a);
 }
```



Synthetic Objects

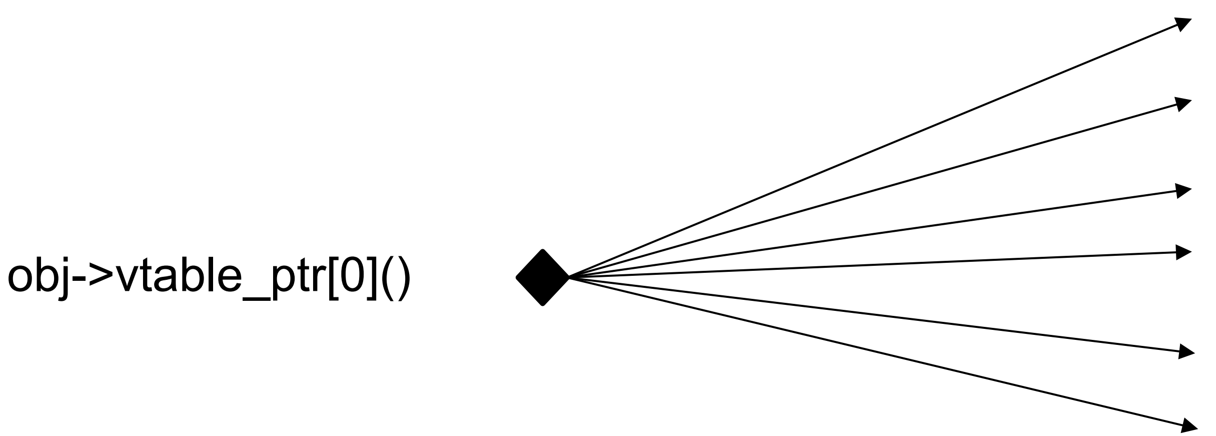
```
class A {
  char *s;
  virtual void foo(char *s) { … }
};
class B : public A {
  void foo(char *s) override { … }
};
void dispatch(A *a){
  a->foo(a->s);
}
int main(int argc, char **argv){
  A *a = new A("String\n");
  B *b = new B("String\n");

  // Arbitrary write for attacker

  dispatch(a);
}
```



Schuster, Felix, Thomas Tendyck, Christopher Liebchen, Lucas Davi, Ahmad-Reza Sadeghi, and Thorsten Holz. "Counterfeit object-oriented programming: On the difficulty of preventing code reuse attacks in C++ applications." *Oakland* 2015

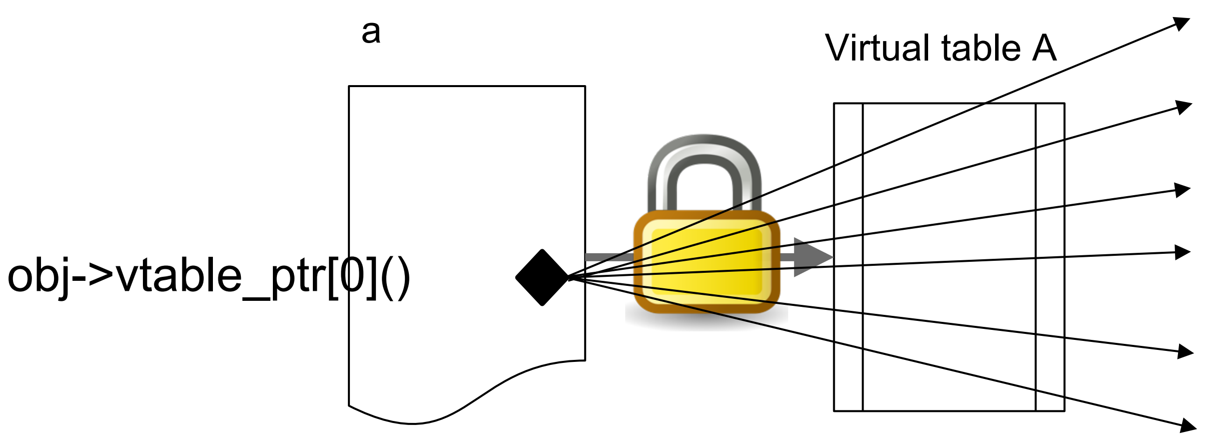
Control-Flow Integrity

- Control-Flow Integrity (CFI)
  - Leverages Control-Flow Graph (CFG)
  - Over-approximation -- allowed target set per indirect callsite
  - Low overhead -- 10% or less



Object Type Integrity (OTI)

- OTI is a new class of defense policies for C++
  - Protects objects by dynamically tracking their allocated type
  - OTI protects ***objects***, CFI protects ***callsites***
- OTI requires objects to have a known type -- can detect synthetic objects!
- OTI is extensible -- dynamic casts, type safety, use-after-free



CFIXX -- OTI Enforcement Mechanism

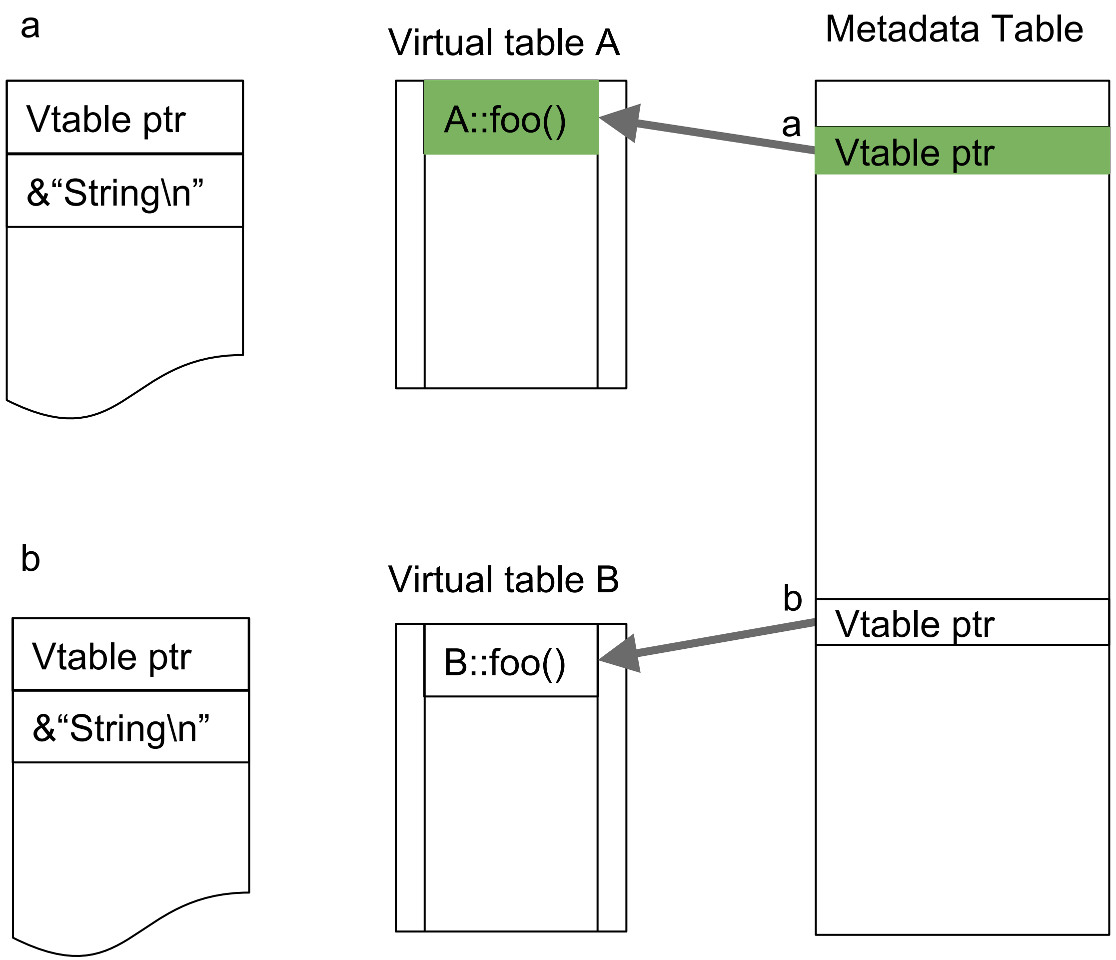
- Enforces C++ object type semantics at machine level
- Instruments dynamic dispatch to enforce defense policy:
  - ✔ Prevention -- dynamic dispatch uses protected object type
  - Detection -- dynamic dispatch compares object type in metadata and object

CFIXX Design

- Compile-time transformation that instruments program
  - Record type assigned by C++ semantics in constructor
  - Use protected type for dynamic dispatch
- Runtime library that maintains object type information
  - Metadata table indexed by *this* pointer
  - Metadata table protected by hardware
- Implemented on LLVM 3.9.1

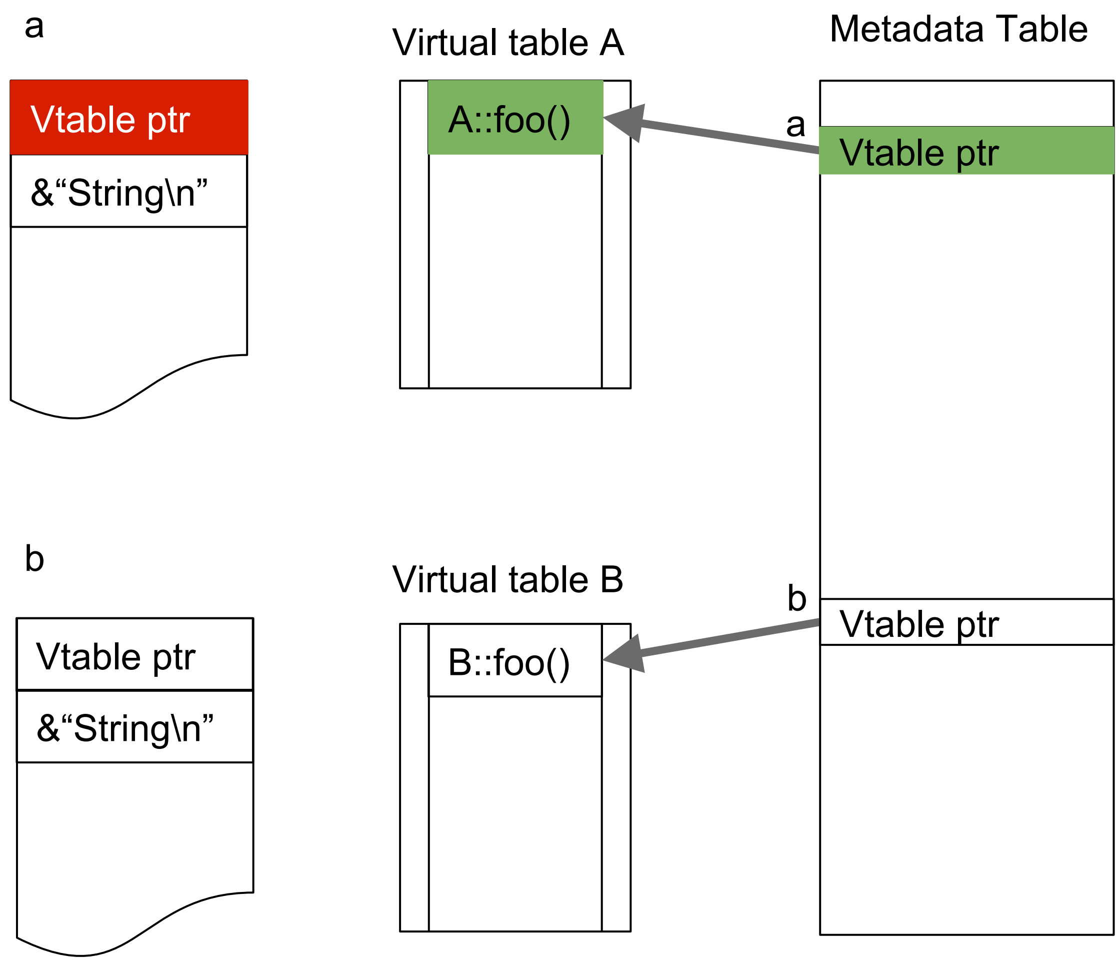
CFIXX Dynamic Dispatch

```
class A {
  char *s;
  virtual void foo(char *s) { … }
};
class B : public A {
  void foo(char *s) override { … }
};
void dispatch(A *a){
  a->foo(a->s);
}
int main(int argc, char **argv){
  A *a = new A("String\n");
  B *b = new B("String\n");

  dispatch(a);
}
```



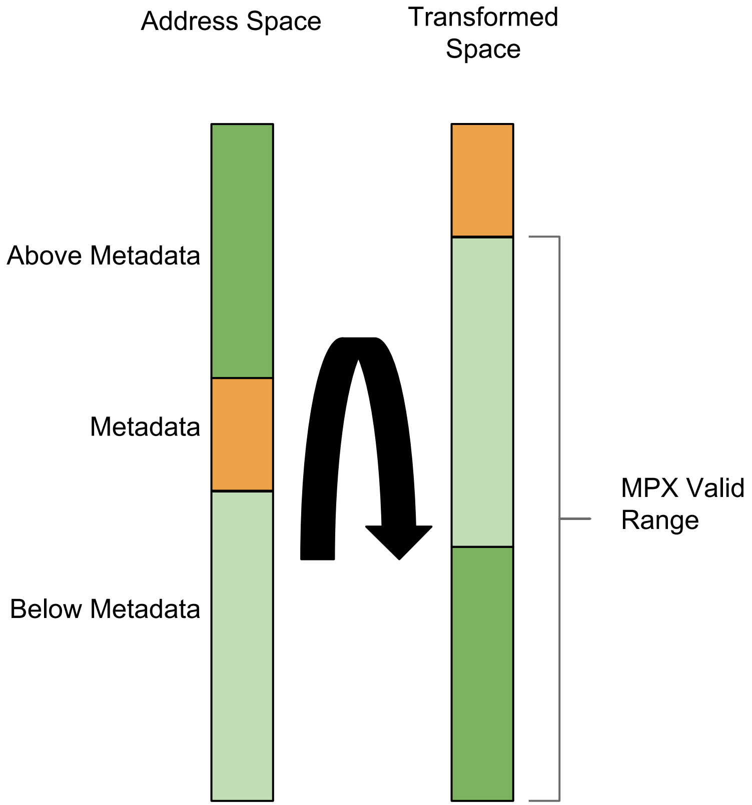
CFIXX vs Attacks

```
class A {
  char *s;
  virtual void foo(char *s) { … }
};
class B : public A {
  void foo(char *s) override { … }
};
void dispatch(A *a){
  a->foo(a->s);
}
int main(int argc, char **argv){
  A *a = new A("String\n");
  B *b = new B("String\n");

  // Arbitrary write for attacker

  dispatch(a);
}
```



Metadata Protection

- Must ***integrity*** protect metadata
- Use Intel Memory Protect Extensions (MPX)
- Check all non-CFIXX writes
- Perform checks on rotated address space
    - MPX requires valid range
    - One instruction to bounds check - bndcu



Evaluation

- Security
  - Microbenchmarks for all known attacks
  - Can combine with CFI to mitigate data flow attacks
- Performance
  - Chromium JS Benchmarks:
    - Octane - 2.03%
    - Kraken - 1.99%
    - JetStream - 2.80%
  - SPEC CPU2006

Security - Existing Defenses

```
class A {
  char *s;
  virtual void foo(char *s) { … }
};
class B : public A {
  void foo(char *s) override { … }
};

class Z {
  char *s;
 virtual void foo(char *s) { … }
};

void dispatch(A *a){
  a->foo(a->s);
}

int main(int argc, char **argv){
  A *a = new A("String\n");
  B *b = new B("String\n");

  dispatch(a);
}
```

- LLVM CFI
  - Static Analysis based CFI
  - Exact Policy evolves over time
- VTrust
  - Static Analysis based CFI
  - Leverages C++ class hierarchy
- CPS
  - Moves code pointers to safe region
  - Does not protect pointers to code pointers

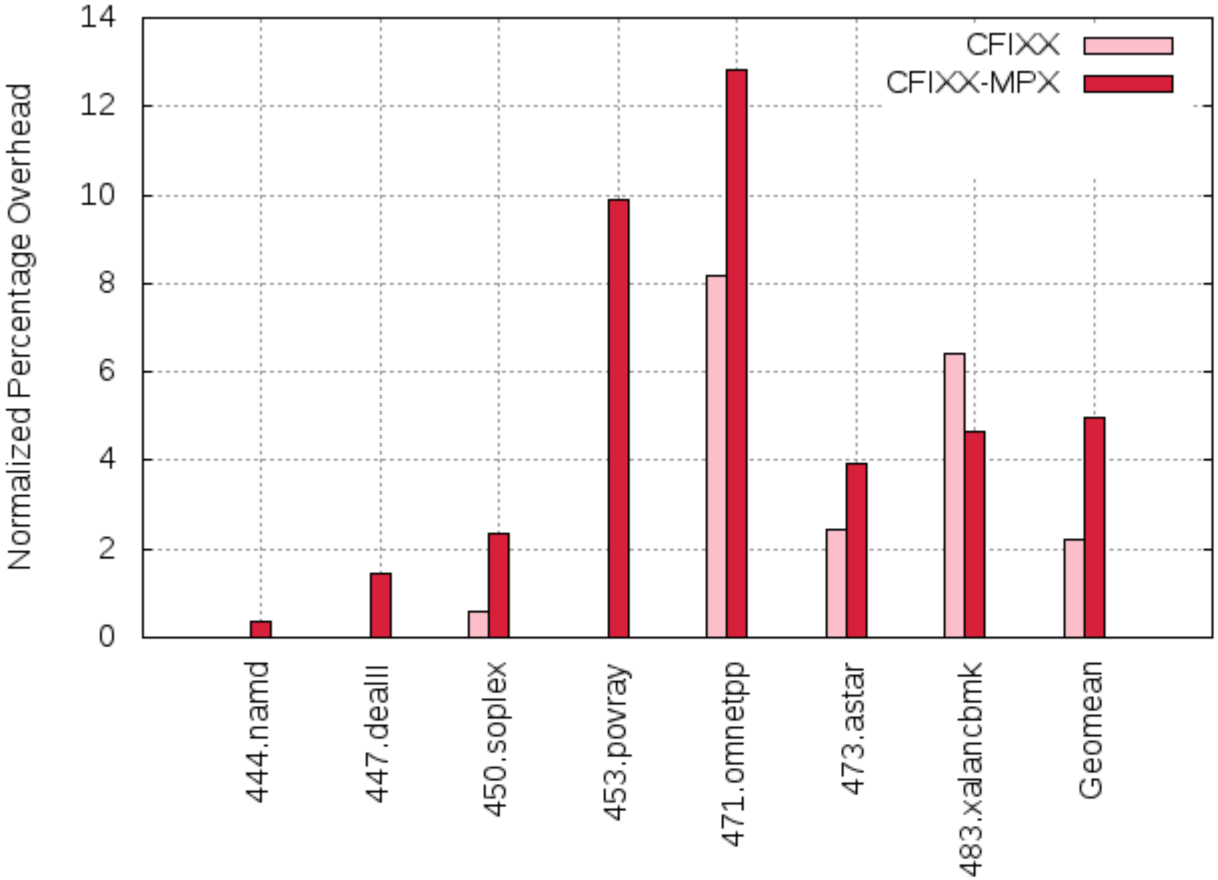
Security Microbenchmarks

	LLVM CFI	VTrust	CPS	CFIXX
FakeVT -- Inject vtable	✔	✔	✔	✔
FakeVT-Sig -- Inject vtable with correct prototypes	✔	✔	✔	✔
VTxchg -- Existing Vtable	✔	✔	✔	✔
VTxchg-hier -- Vtable of related class	✘	✘	✔	✔
COOP -- Synthetic objects	✘	✘	✘	✔

OTI vs CFI

- OTI can be combined with CFI
- OTI protects ***objects***, CFI protects ***callsites***
- CFI is over-approximate
  - Target sets based on static analysis
  - OTI uses dynamic information per object
- Data flow attacks that change object used at callsite
  - Not caught by OTI
  - Mitigated by adding CFI

SPEC CPU2006



Conclusion

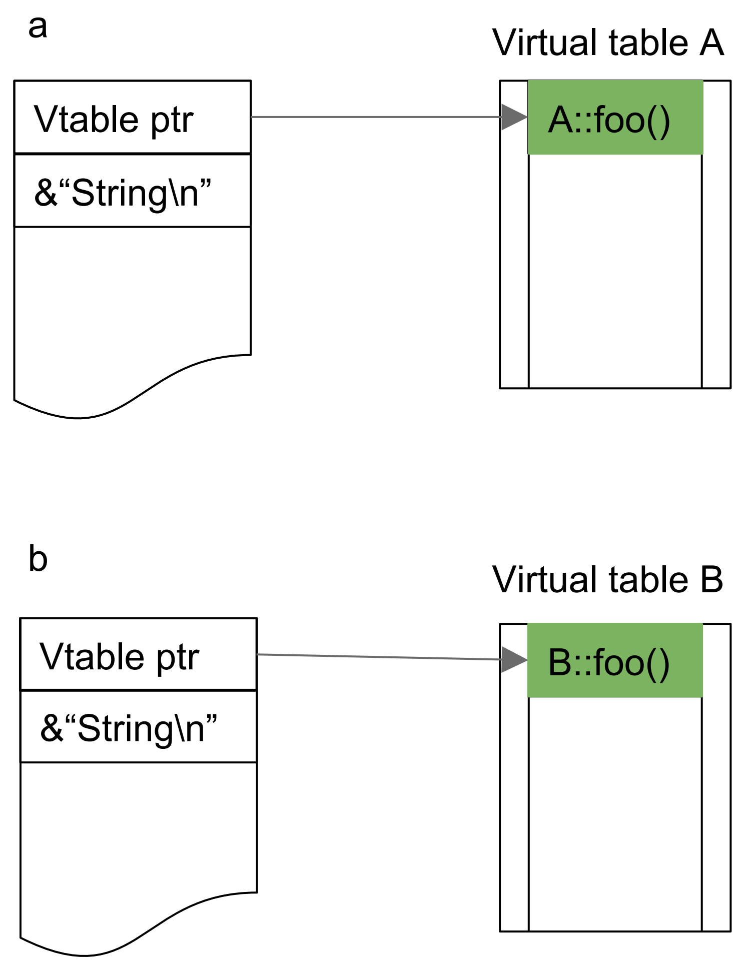
- OTI is a new class of defense policy
  - CFIXX mechanism guarantees correctness of dynamic dispatch per object
  - Can be extended to dynamic type safety, UaF
- Low performance overhead -- 2% on Chrome
- Can be combined with CFI to mitigate data flow attacks
- CFIXX implementation is open source

https://github.com/HexHive/CFIXX

Questions?

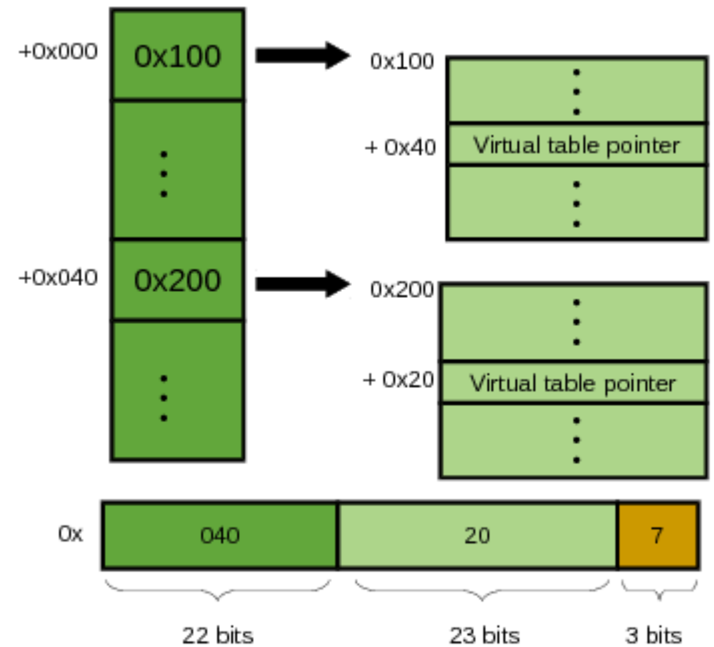
Dynamic Dispatch

```
class A {
  char *s;
  virtual void foo() { … }
};
class B : public A {
  void foo(char *s) override { … }
};
void dispatch(A *a){
  a->foo(a.s);
}
int main(int argc, char **argv){
  A *a = new A("String\n");
  B *b = new B("String\n");

  dispatch(a);
  dispatch(b);
}
```

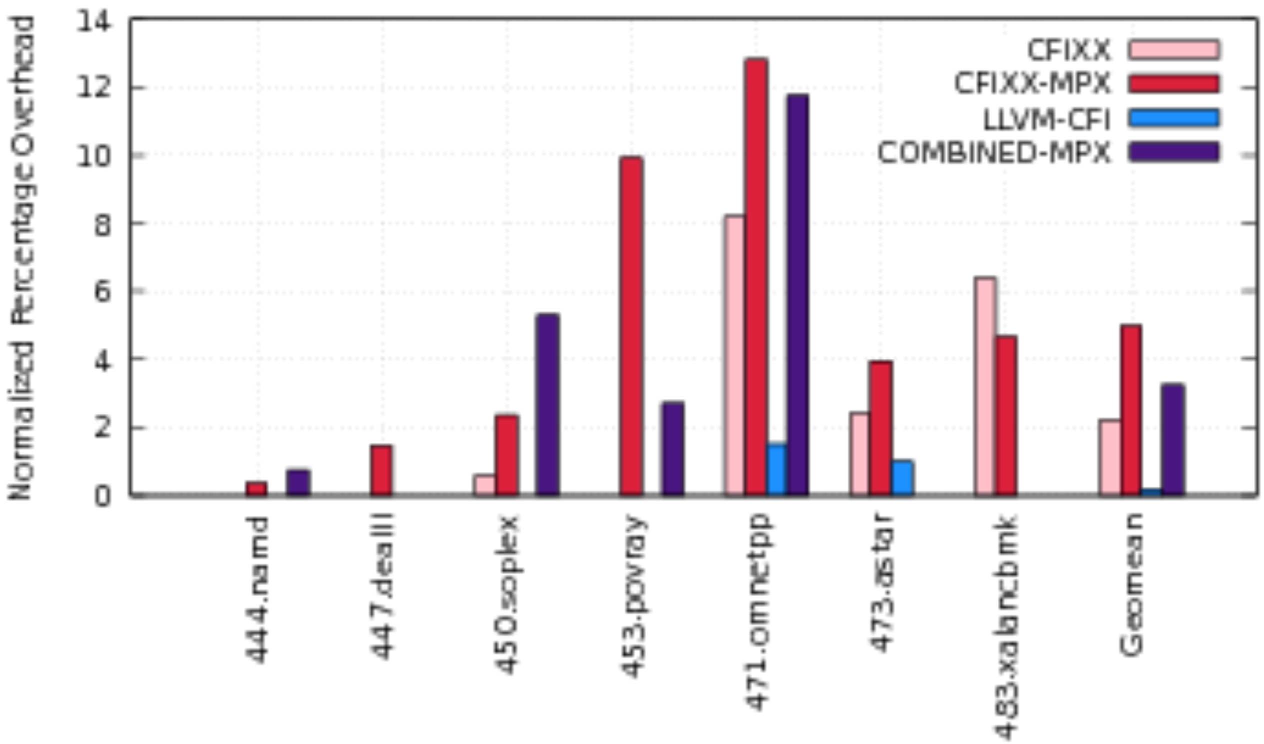


Metadata Structure

- Two Level Page Table
- 48 bit pointers:
  - 22 bits used as index in first level
  - 23 bits used as index in second level
  - 3 bits unused
- Fixed number of second level tables
- 1x memory overhead on SPEC



Full Performance Results



Attack Vector

- Attackers subvert dynamic dispatch to hijack control flow
- Attackers seek to control the vtable pointer
- Vtable pointer determines an object’s type
- Attacker’s control object types => control-flow hijacking

Object Type Integrity -- Prevent attackers from controlling an object’s type by integrity protecting vtable pointer