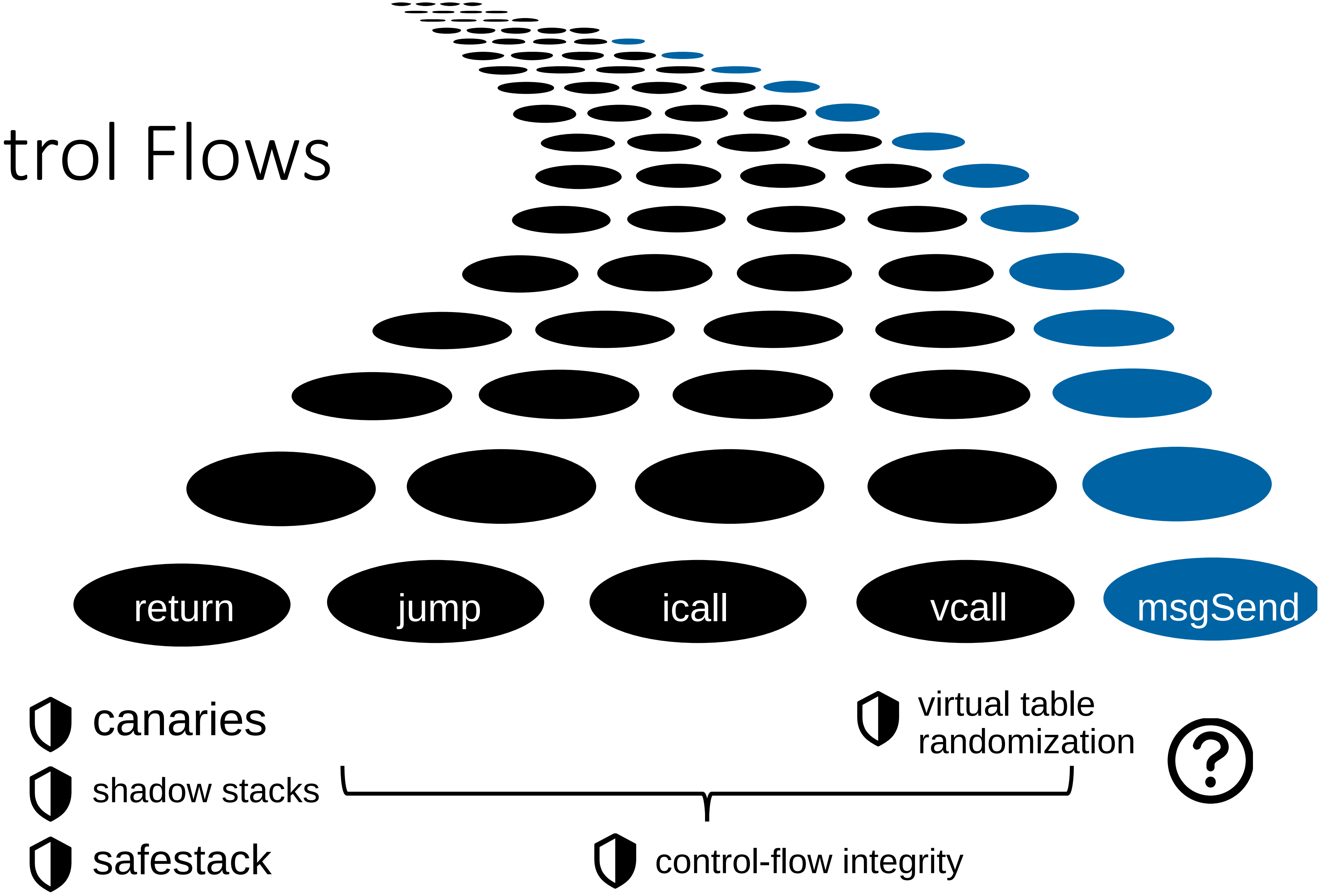
immunant

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Subversive-C: Abusing and Protecting Dynamic Message Dispatch

Julian Lettner, Benjamin Kollenda, **Andrei Homescu**,
Per Larsen, Felix Schuster, Lucas Davi,
Ahmad-Reza Sadeghi, Thorsten Holz, Michael Franz

Control Flows



return

jump

icall

vcall

msgSend

🛡 canaries

🛡 shadow stacks

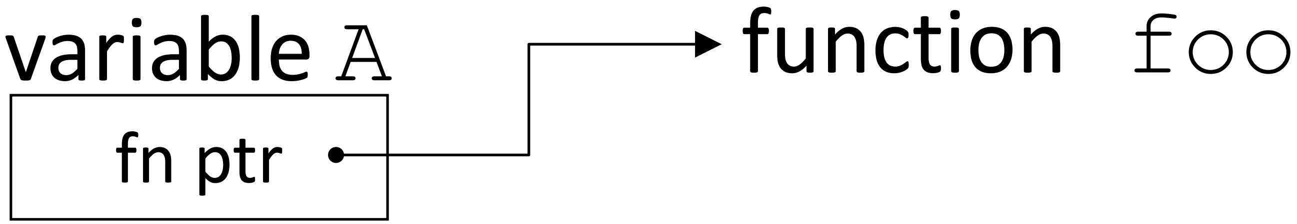
🛡 safestack

🛡 virtual table randomization

?

🛡 control-flow integrity

Control Flow Hijacking



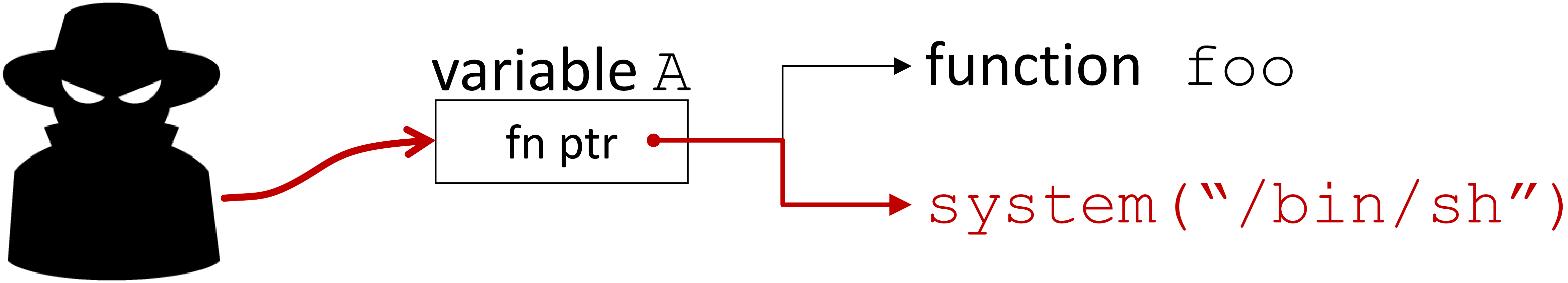
variable A

fn ptr

function foo

```
addr = Load(A);
Goto(addr);
```

Control Flow Hijacking



variable A

fn ptr

function foo

system("/bin/sh")

```
addr = Load(A);
Goto(addr);
```

Objective-C

Smalltalk-style object orientation

Good 'ol C

Message Dispatch

C++

```
A *obj = new A;
obj->foo();
```

- Caller "calls a method" in object
- Resolved using vtables
- Static class structure

Objective-C

```
A *obj = [[A alloc] init];
[obj foo];
```

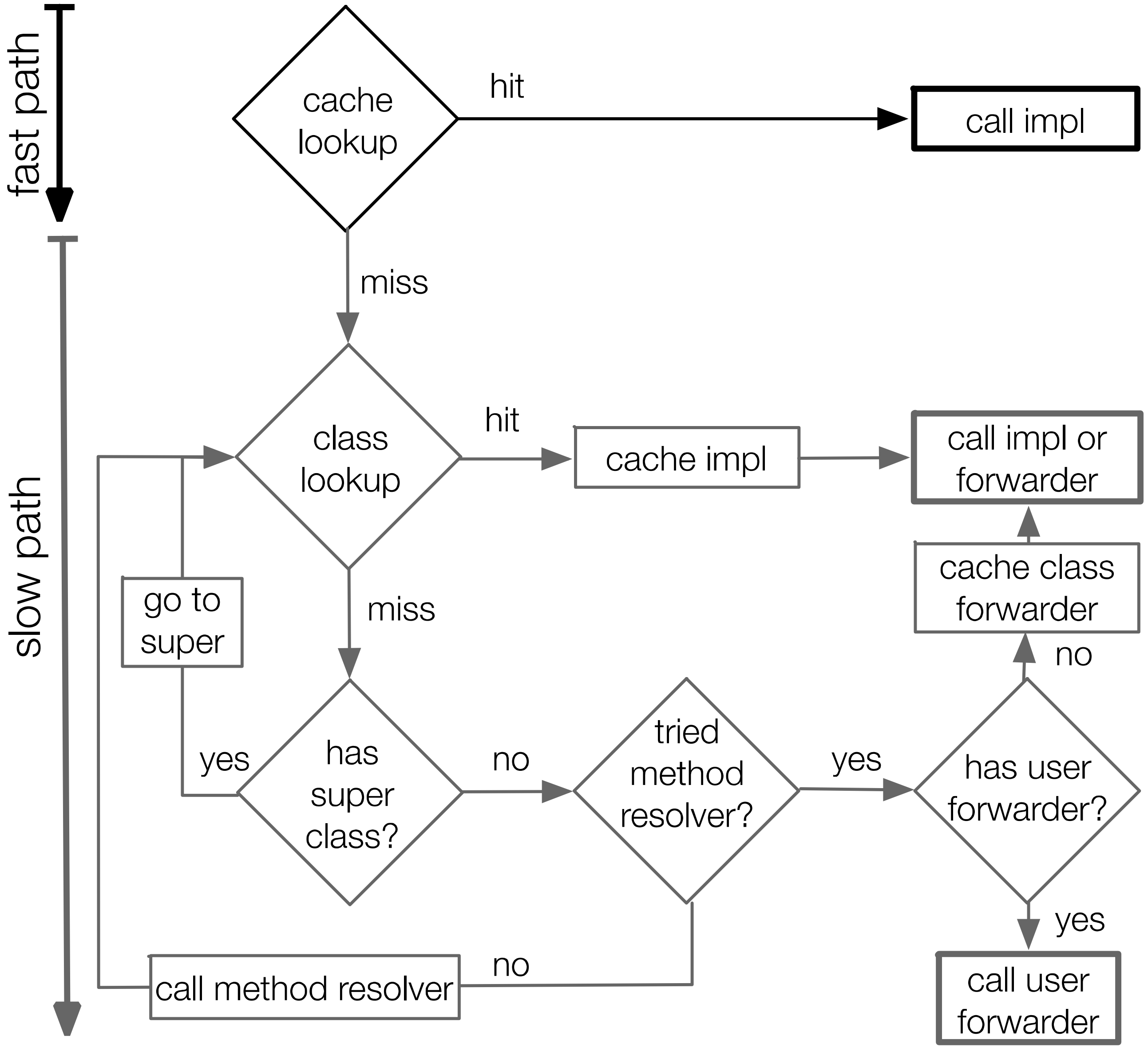
- Caller "sends a message" to object
- Resolved dynamically at run-time
- Dynamic class structure

Class Mutability

```
void fooIMP(id self, SEL _cmd) {}

A *obj = [[A alloc] init];
class_addMethod([obj class], @selector(foo),
                (IMP) fooIMP, “v@:”);
[obj foo];
```

Fast Path


fast path

slow path

cache lookup

hit

call impl

miss

class lookup

hit

cache impl

call impl or forwarder

miss

go to super

yes

has super class?

no

tried method resolver?

yes

has user forwarder?

no

cache class forwarder

yes

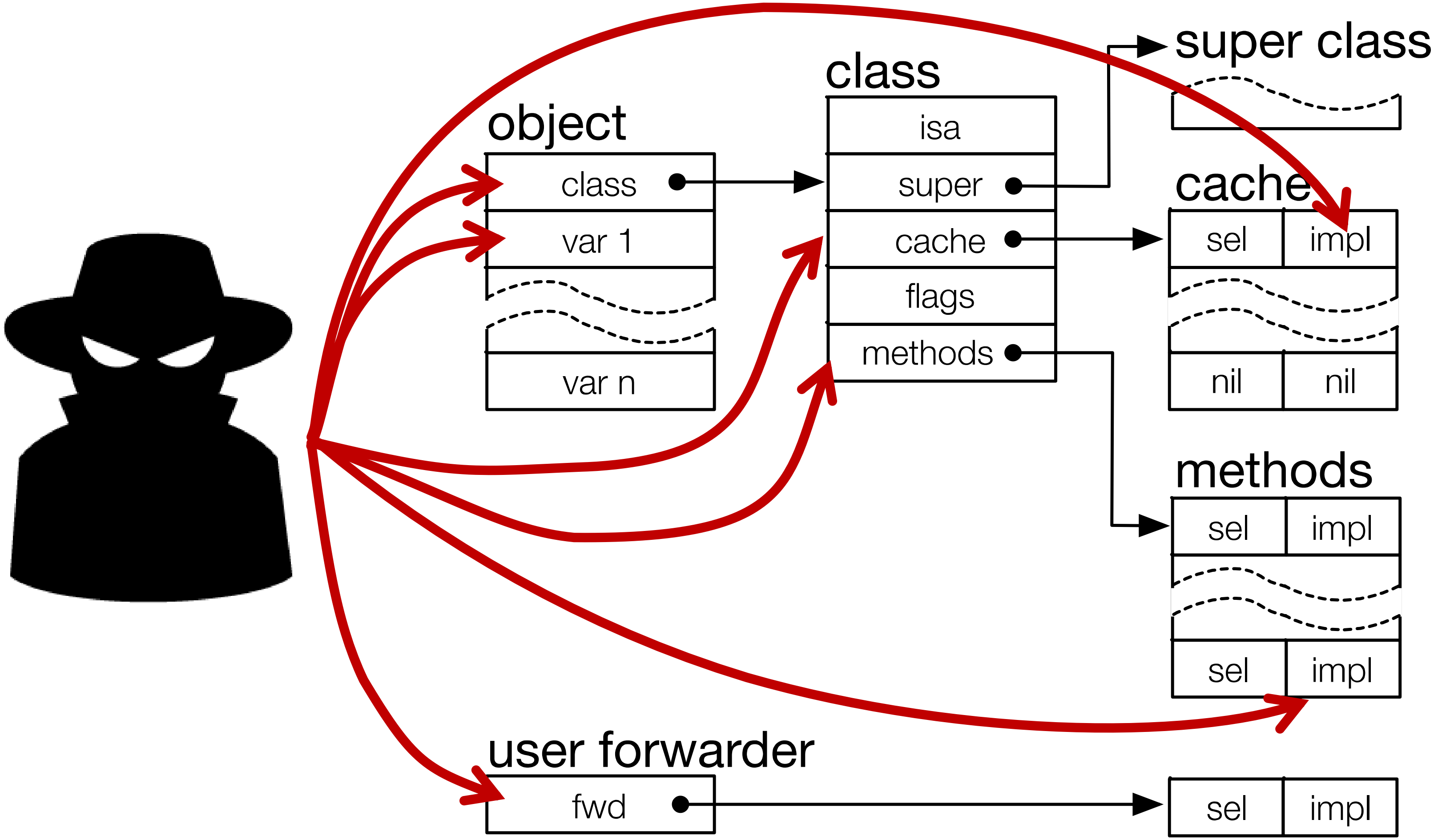
call user forwarder

no

call method resolver

Slow Path

Object Layout

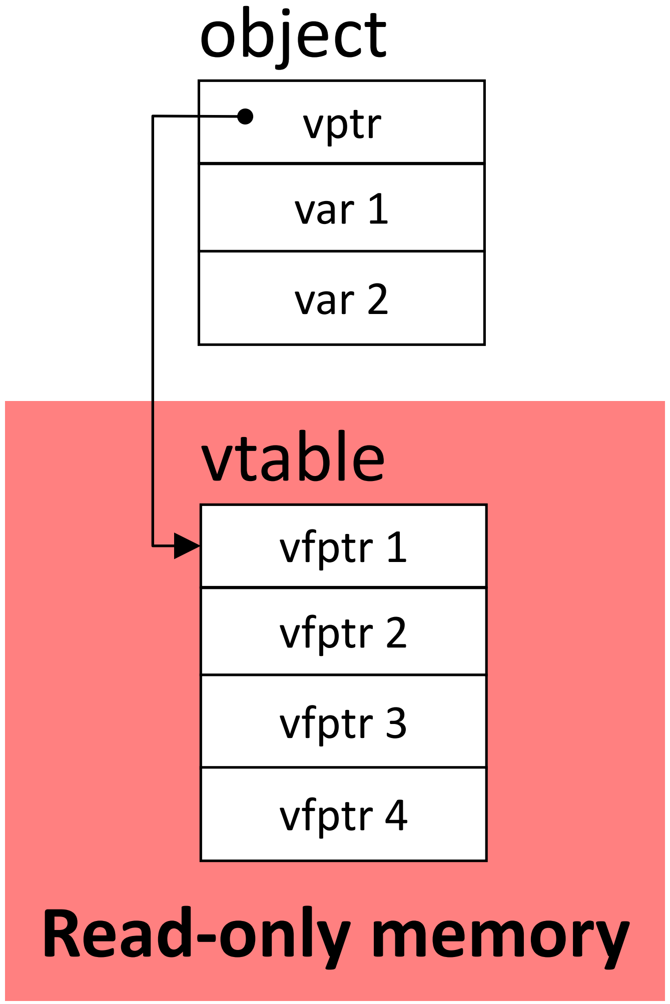


Attacker Model

- Arbitrary memory read (information disclosure)
- Arbitrary memory write
- No other control flow hijacking
    - No code injection
    - No code reuse (ROP, COOP, etc.)

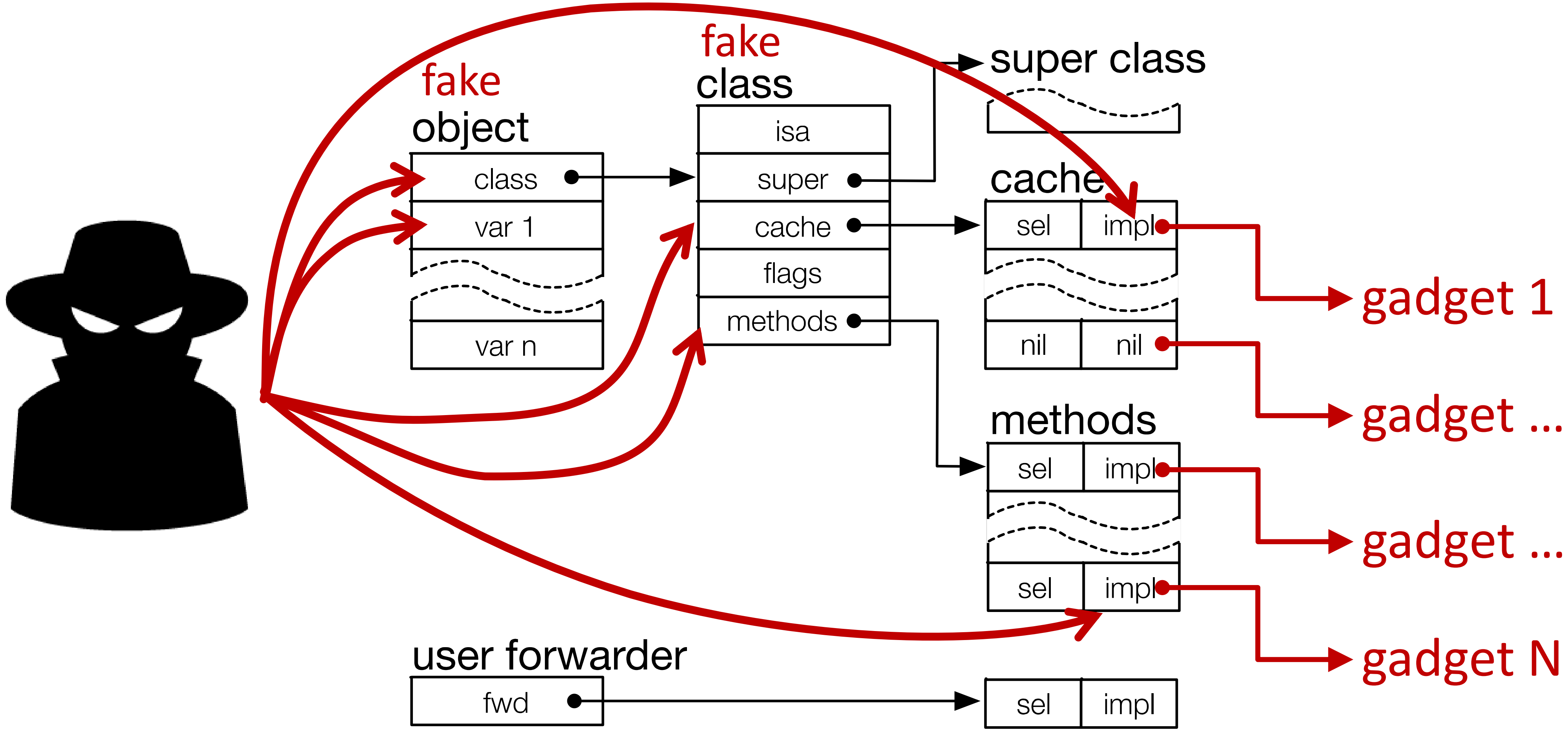
Previously: COOP

- COOP: Counterfeit Object-Oriented Programming
- Counterfeit objects attack for C++
- Reuses existing **vtables** (fully or partially)
- Reuses whole C++ functions



F. Schuster, Th. Tendyck, Ch. Liebchen, L. Davi, A.-R. Sadeghi, Th. Holz. **Counterfeit Object-oriented Programming: On the Difficulty of Preventing Code Reuse Attacks in C++ Applications.** IEEE S&P 2015.

Subversive-C



Subversive-C

- What we have
  - Arbitrary counterfeit Objective-C objects
  - Control flow hijacking

- What we want
  - Call malicious system call, e.g., `system(“/bin/sh”)`

Calling `system(“/bin/sh”)`

1. Find the address of `system()` in GOT

2. Set up function call arguments
   - Store `“/bin/sh”` in memory
   - Set up argument registers/stack

3. Invoke `system()` via computed address

Gadgets

Gadget	Description
`ML-G`	Dispatch execution to other gadgets
`LOAD-R64-G`	Load register from Objective-C object
`R-G`	Load register from memory
`ARITH-G`	Add two registers
`W-G`	Write result to Objective-C object
`INV-G`	Call function pointer from object

Example: Main Loop Gadget

- Used to invoke other gadgets repeatedly (gadget loops)
- Code from `dealloc` in `NSTextReplacementNode`

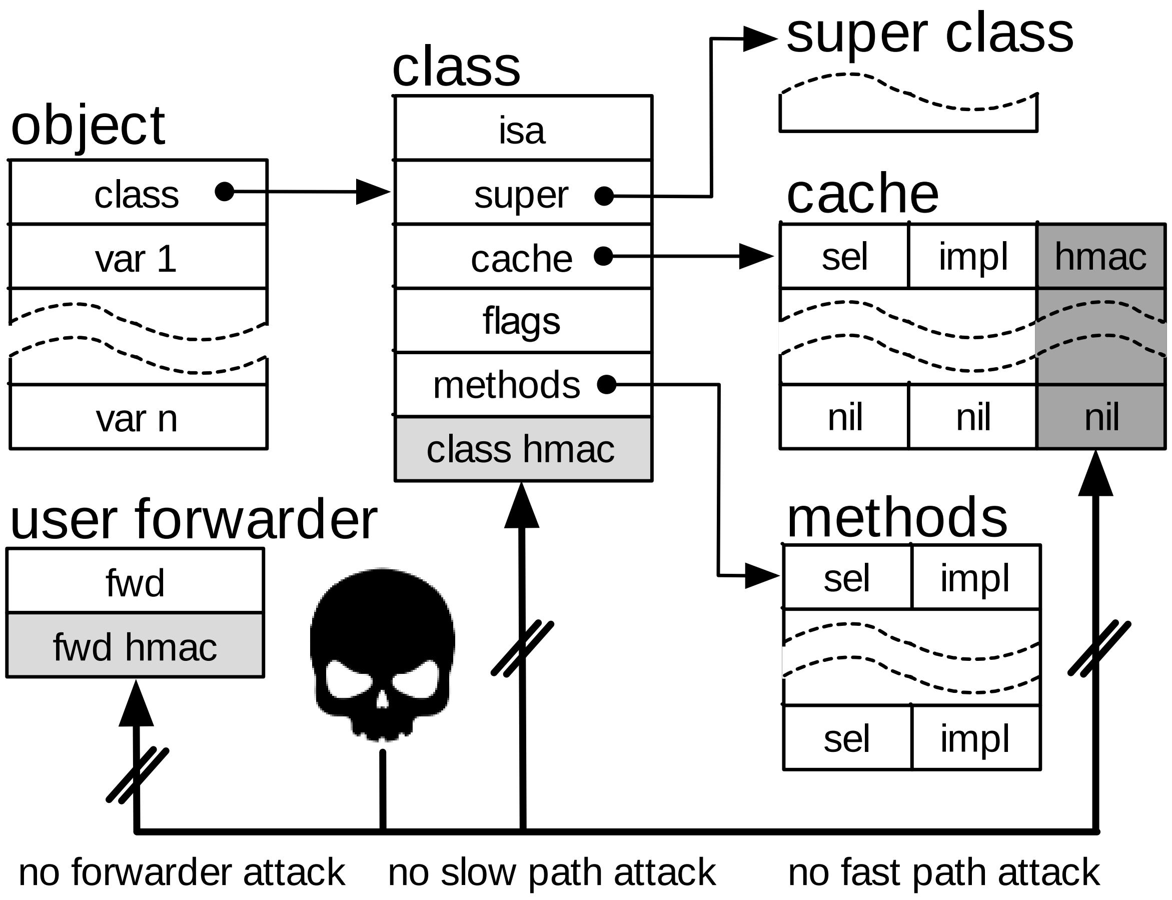
```
children = self->children;
counter = 0;
while (children[counter] != 0 && counter < 28) {
  [children[counter] release];
  counter++;
}
```

Results

- Successfully applied attack to AppKit on vulnerable PoC program

- AppKit is used by many popular Mac OS X apps

Defense: Object Layout Integrity



Securing the Slow Path

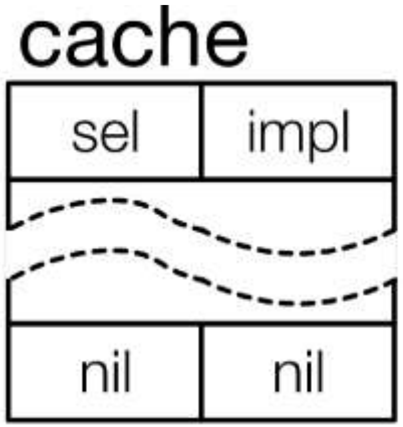
- $HMAC(K, m) = HMAC\text{-}MD5(K, m)$

- Checked on every slow path lookup

- K is a random 64-bit key stored in execute-only memory

- $m = \&class \parallel isa \parallel superclass \parallel flags \parallel method\ list\ elements$

Securing the Fast Path

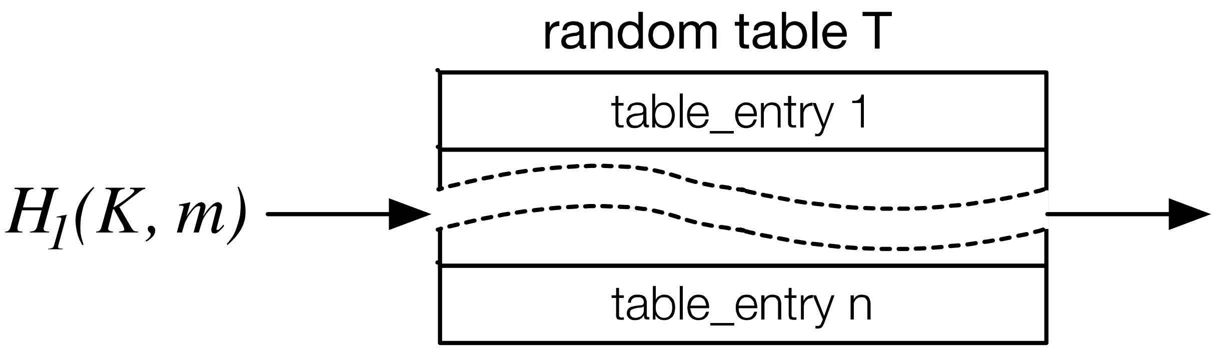

cache

sel	impl
nil	nil

- $UMAC$
  - $H_1(K,m) = \sum_{i=0}^{i<3}(m_L[i] + KL[i]) * (m_H[i] + KH[i])$

- K is a 192-bit random number stored in execute-only memory

- $m = \&class \parallel sel \parallel impl$


random table T

table_entry 1

table_entry n

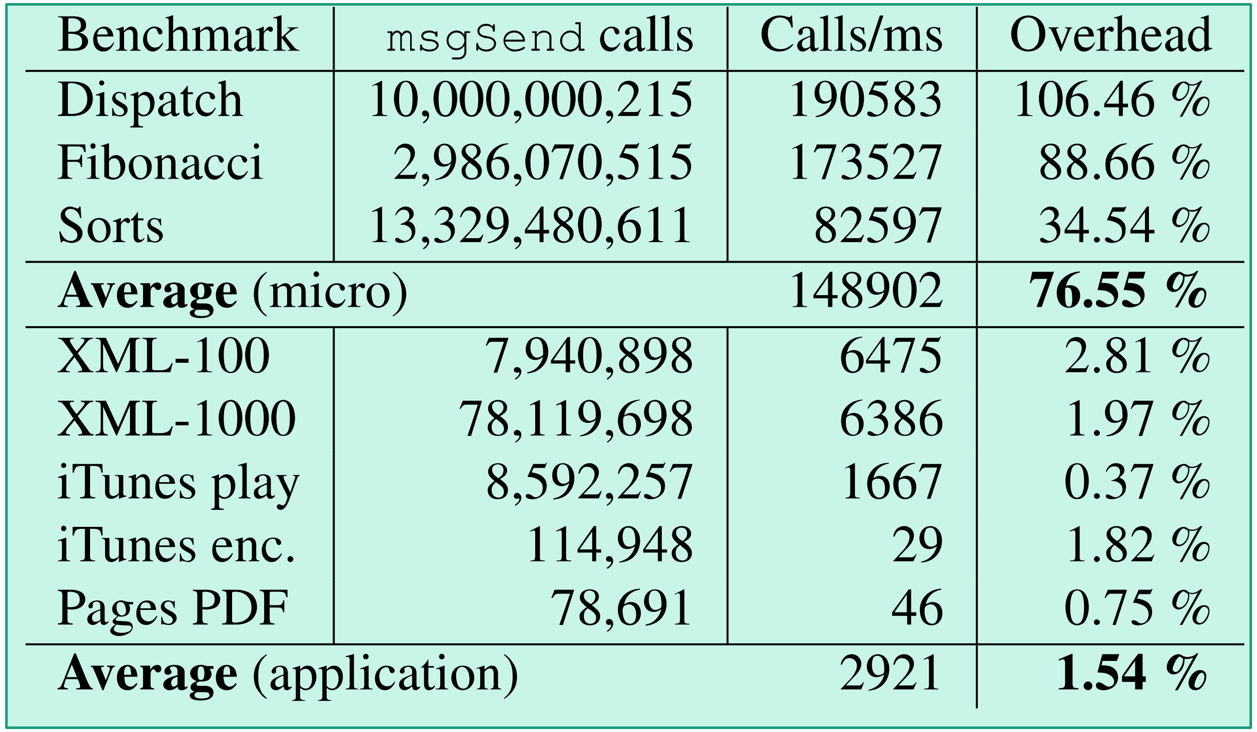
$H_1(K, m)$

eXecute-only Memory

- Crucial defense against information leaks
- Store HMAC keys in XoM (write-once or constant data)
- Access via execution
- Can be implemented in hardware or software
    - `mprotect()`-based mechanism
    - TLB splitting
    - EPT on x86
    - ARMv8 native support

Performance Evaluation

- Drop-in replacement for Objective-C runtime shipped by Apple!
  - Micro-benchmarks
  - iTunes, Pages, etc.



Benchmark	`msgSend` calls	Calls/ms	Overhead
Dispatch	10,000,000,215	190583	106.46 %
Fibonacci	2,986,070,515	173527	88.66 %
Sorts	13,329,480,611	82597	34.54 %
Average (micro)		148902	**76.55 %**
XML-100	7,940,898	6475	2.81 %
XML-1000	78,119,698	6386	1.97 %
iTunes play	8,592,257	1667	0.37 %
iTunes enc.	114,948	29	1.82 %
Pages PDF	78,691	46	0.75 %
Average (application)		2921	**1.54 %**

Summary

- Control flow hijacking attack on Objective-C message dispatch
- HMAC-based object integrity defense for Apple Objective-C runtime
- Low performance overhead (1.54% on real-world applications)

Questions?

Previous joint work open sourced and released into

Hardened Tor Browser for Linux

https://github.com/immunant/selfrando