TypeArmor [S&P16]

A Tough **`call`**: Mitigating Advanced Code-Reuse Attacks At The Binary Level

Victor van der Veeny, Enes G¨oktas¸y, Moritz Contagz, Andre Pawlowskiz, Xi Cheny,
Sanjay Rawaty, Herbert Bosy, Thorsten Holzz, Elias Athanasopoulosy, and Cristiano Giuffriday

20203331 신명근 MyeongGeun Shin

2020.09.07 Mon @ IS893-2020-fall

What is TypeArmor exactly?

A **binary-level CFI** technique
(against **COOP** and it's advances)

CFI ?

- **C**ontrol-**F**low **I**ntegrity
- Code reuse attacks
- Much easier in source-level

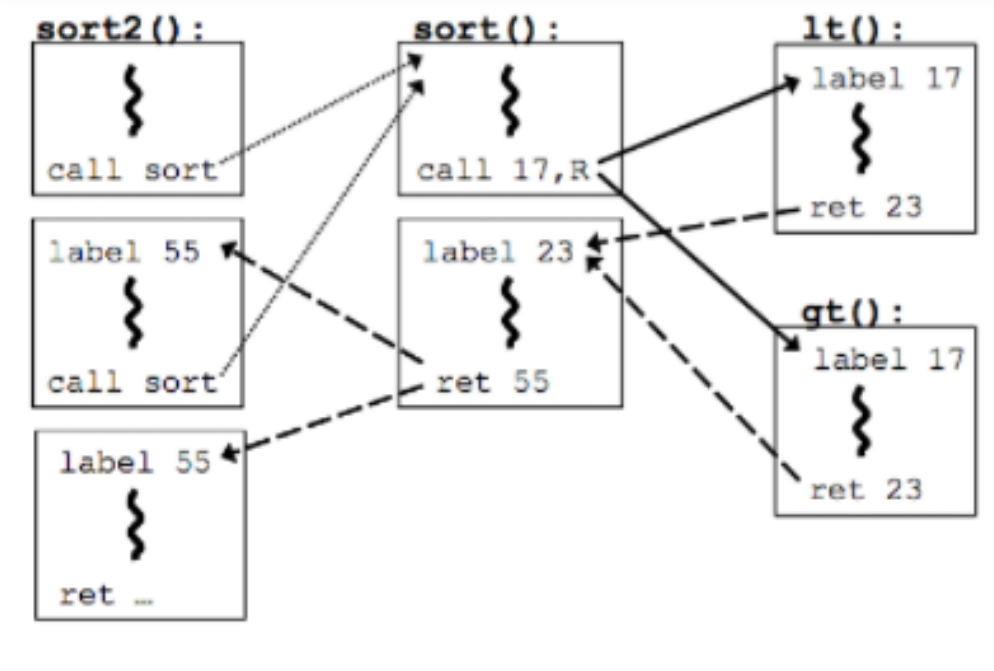


Image from https://dl.acm.org/doi/10.1145/1102120.1102165

COOP ?

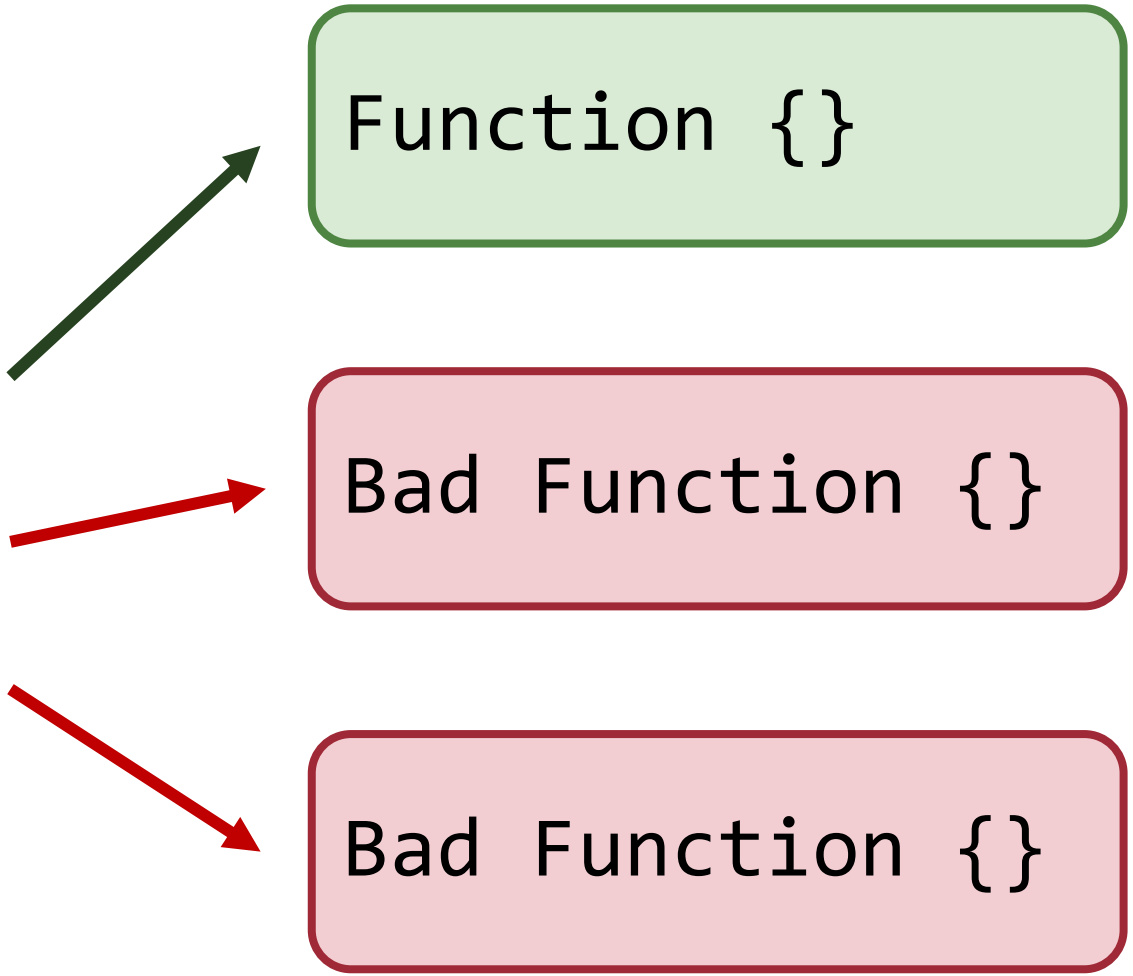
- **C**ounterfeit **O**bject-**O**riented **P**rogramming

- Effective method to break existing CFI mitigations

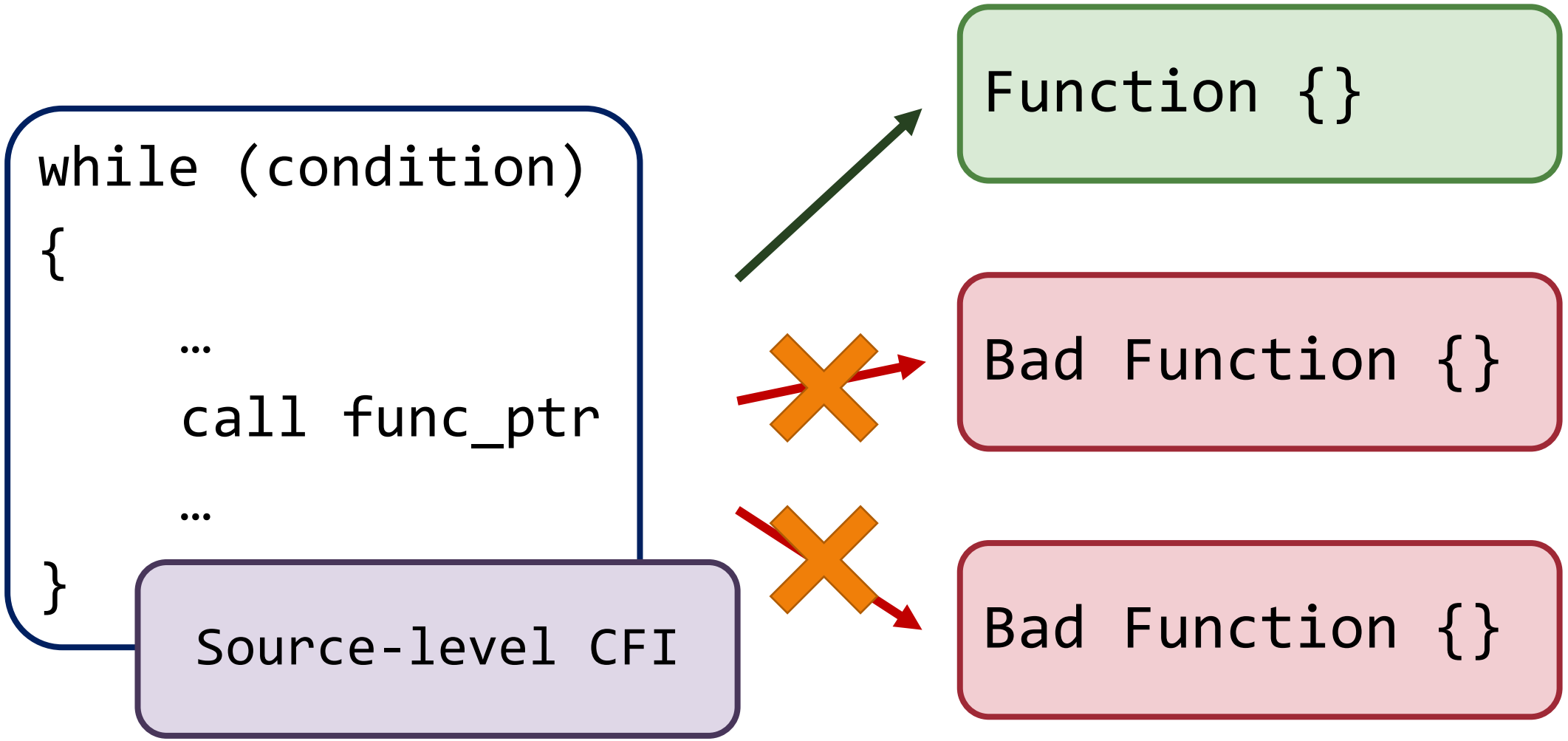
```
for index from src to dst {          // inside vf
    object[index]->do_something()
}   // example ML-Gadget
```

What TypeArmor does

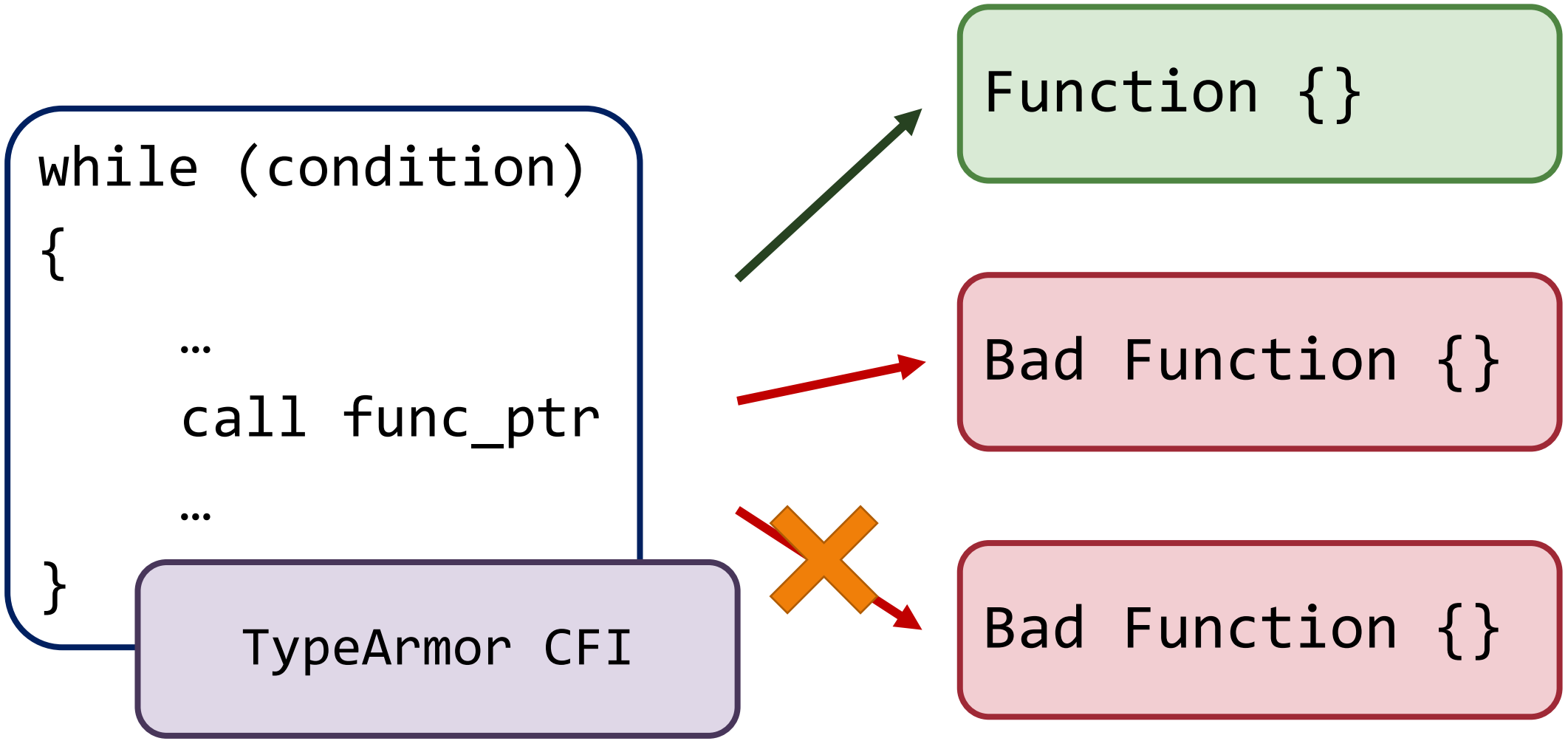
```
while (condition)
{
      …
      call func_ptr
      …
}
```



What TypeArmor does



What TypeArmor does

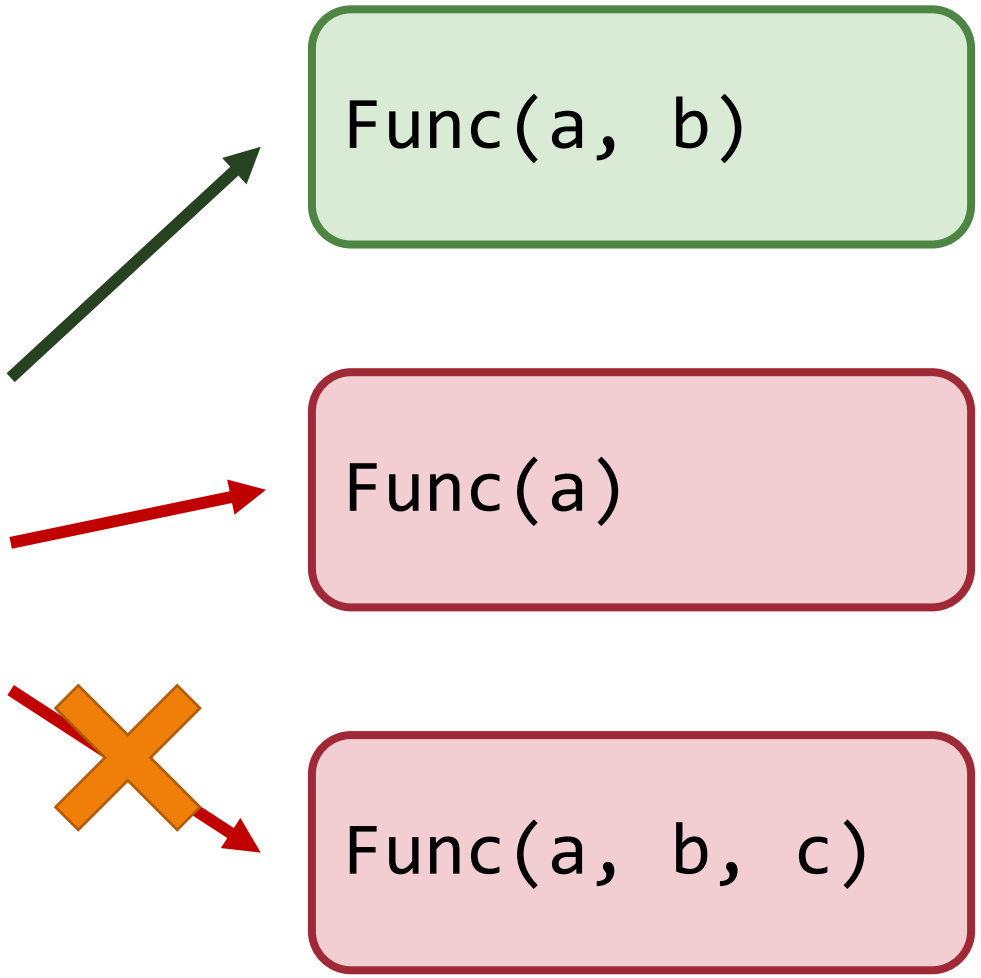


Making function signatures

- With function's argument count
- Whether function returns value or not
- Allow jmp when signature matches (allows)

What TypeArmor does

```
while (condition)
{
    …
    CHECK ARG_CNT <= 2
    call func_ptr(a,b)
    …
}
```



In x86_64 architecture

- Function arguments are passed through registers
- Check some register has read before write
- Check some register has written before read
- Check caller uses return value or not

Check function arguments

call with 3 arguments

```
movl %rax,%rdi
movl %rbx,%rsi
mov  0x123,%rdx
call *0x8(%rax)
... ...
... ...
movl 0x44,%rdi
call *%rax
```

call with 1 argument

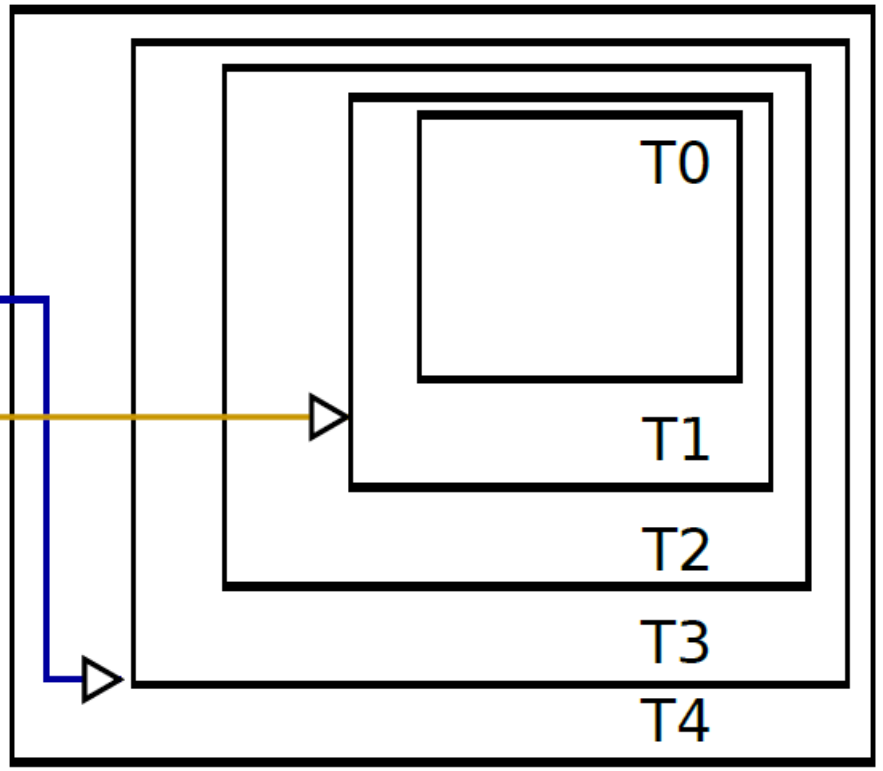


Image from paper

How about this?

```
foo(void)
      mov $0x1, %rdi

bar(int arg1)
      ...

main(void)
      foo()
      mov $0x1,%rdi
      call bar
```

(a)

```
bar(int arg1)
      ...

foo(int arg1)
      ...

main(void)
      mov $0x1,%rdi
      call foo
      mov $0x1,%rdi
      call bar
```

(b)

Static analysis (use-def)

- Check some register has read before write
- Check some register has written before read
- Merge basic blocks not harming functionality

Static analysis (cont')

- Register has read before write - mark **R**
- Register has written before read - mark **W**
- Register untouched (clear) - mark **C**
- Merge basic blocks not harming functionality

Static analysis (cont')

x86_64 calling convention

<< underestimates arg_count <<

Functionality conservativeness

Return value check

- Check whether caller uses $RAX or not
- Check whether callee sets $RAX or not
- Return-value-expected Caller can only call non-void Callee

CFC

- Control-Flow Containment

Undefined arguments at the callsite

Not used by any callee by design

Effectiveness?

Exploit	Stopped?	Notes
COOP ML-G [26]		
– IE (32-bit)	✗	Out of scope
– IE 1 (64-bit)	✓(CFI)	Argcount mismatch
– IE 2 (64-bit)	✓(CFI)	Argcount mismatch
– Firefox	✓(CFI)	Argcount mismatch
COOP ML-REC [13]		
– Chrome	✓(CFI)	Argcount mismatch, Void target where non-void was expected
Control Jujutsu [16]		
– Apache	✓(CFI)	Target function not AT
– Nginx	✓(CFI)	Void target where non-void was expected

Table from paper

Performance issues?

Server	*IC/sec*	*CFI*	*+CFC*
Exim	4,574	1.068	1.067
lighttpd	1,425,099	1.116	1.174
Memcached	72,519	1.014	1.017
Nginx	5,084,715	1.132	1.155
OpenSSH	78	1.021	1.013
ProFTPD	542,443	1.007	1.002
Pure-FTPd	17	1.020	1.013
vsftpd	24,024	1.025	1.051
PostgreSQL	18,024,485	1.160	1.205
MySQL	19,693,937	1.239	1.222
Node.js	1,965,955	1.061	1.055
geo-mean	110,157.9	1.076	1.086

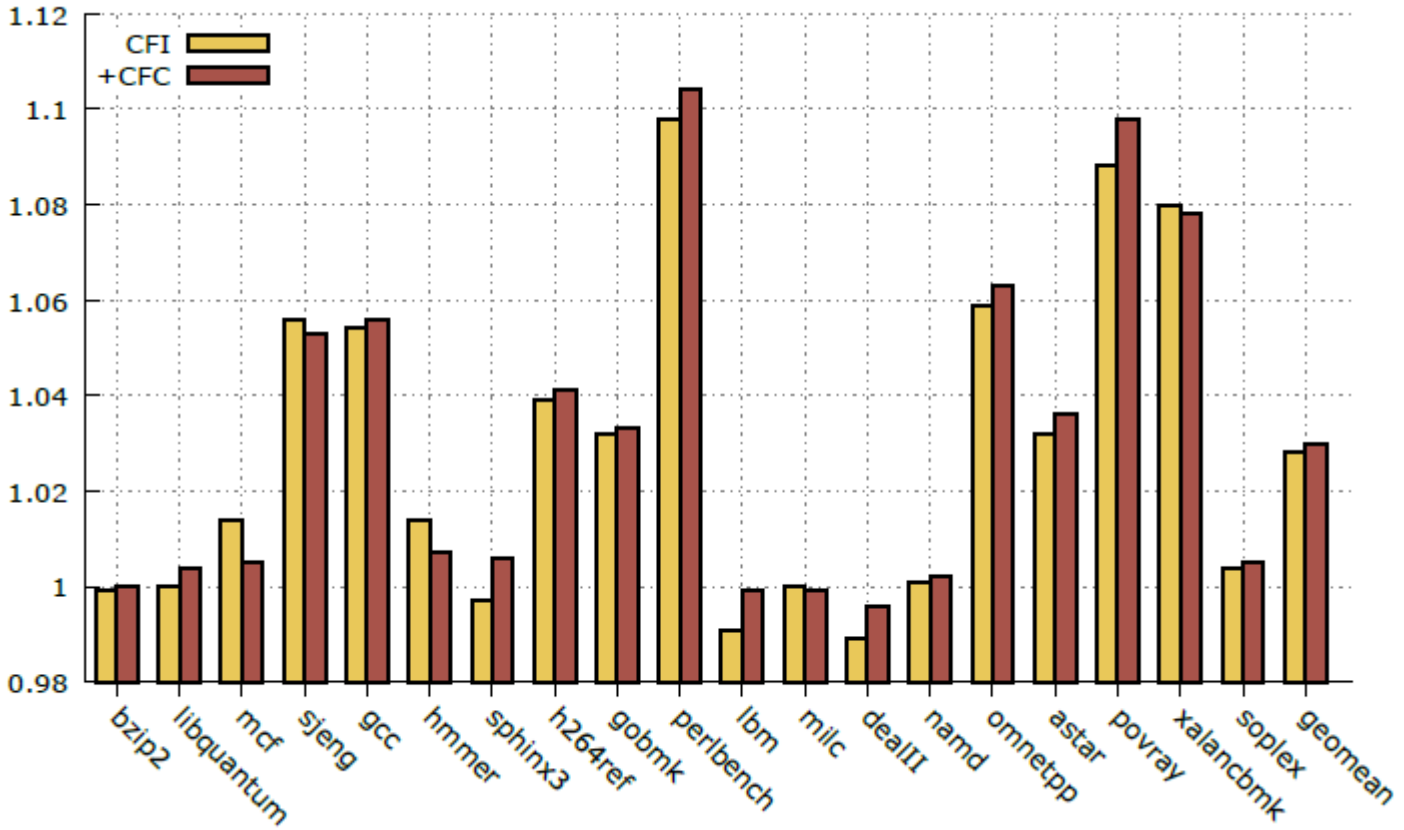


Table from paper

Conclusion

- Mitigated
  - Many code-reuse attacks
  - Most of COOP, COOP-like attacks
- Things to be improved
  - Other architectures (Calling conventions)
  - Data-only attacks

Questions?