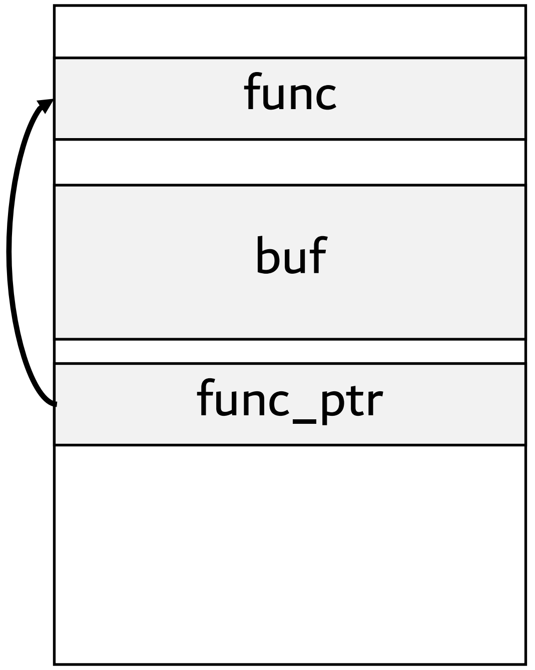
Code-Pointer Integrity

Volodymyr Kuznetsov, László Szekeres, Mathias Payer
George Candea, R. Sekar; Dawn Song

Hyewon Ryu

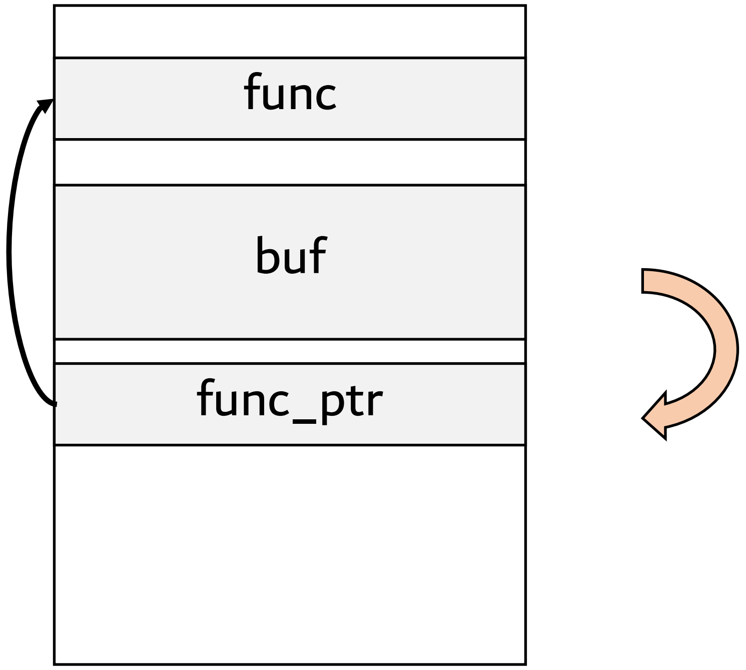
September 9, 2020 @ IS893

Control-flow Hijack Attack



C/C++: exploit memory safety bug -> divert control flow

Control-flow Hijack Attack



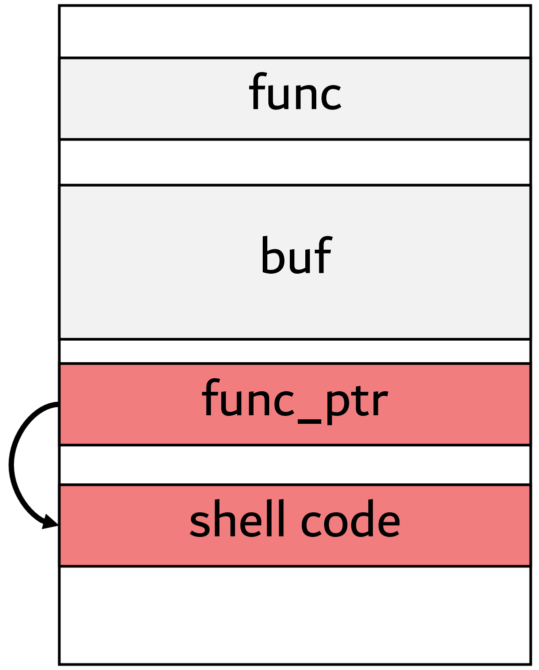
buffer overflow

use after free

...

C/C++: exploit memory safety bug -> divert control flow

Control-flow Hijack Attack



C/C++: exploit memory safety bug -> divert control flow

Code-Pointer Integrity (CPI)

: Memory safety for code pointers only

- NO sanity check (like CFI)

- Prevent from corrupt

Key idea of CPI

- Focus on **sensitive pointer**

- **Separate** safe and regular memory

- Enforce memory safety only for safe region (**isolation**)

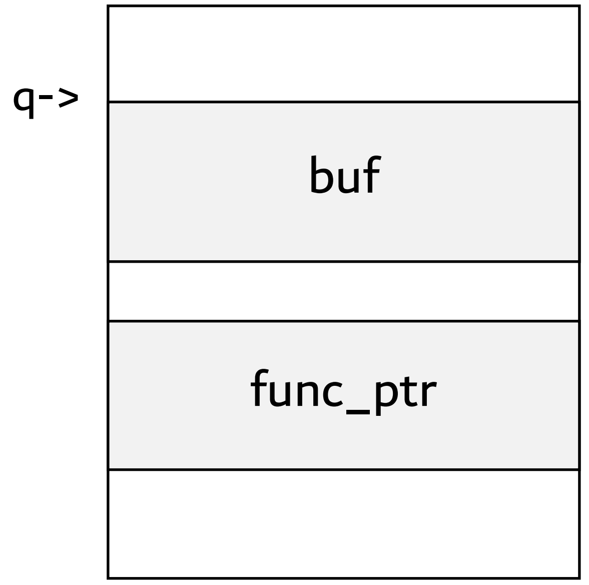
Practical Protection (CPS)

Sensitive pointers = Code pointers

(function pointers, return addresses)

CPS - heap

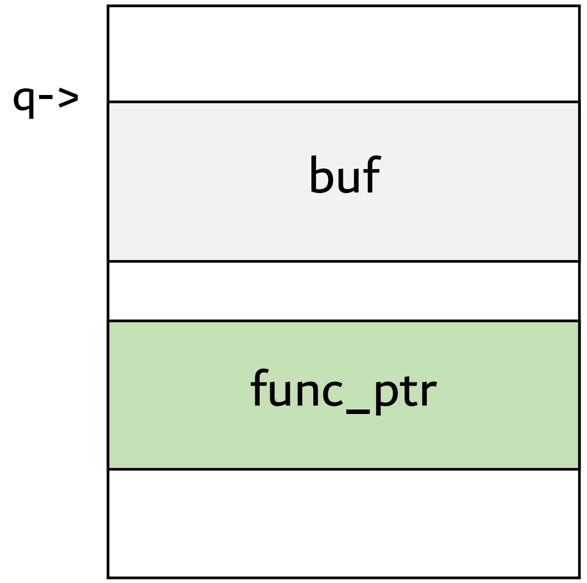
```
int *q = buf + input;
*q = input2;
…
(*func_ptr)();
```



CPS - heap

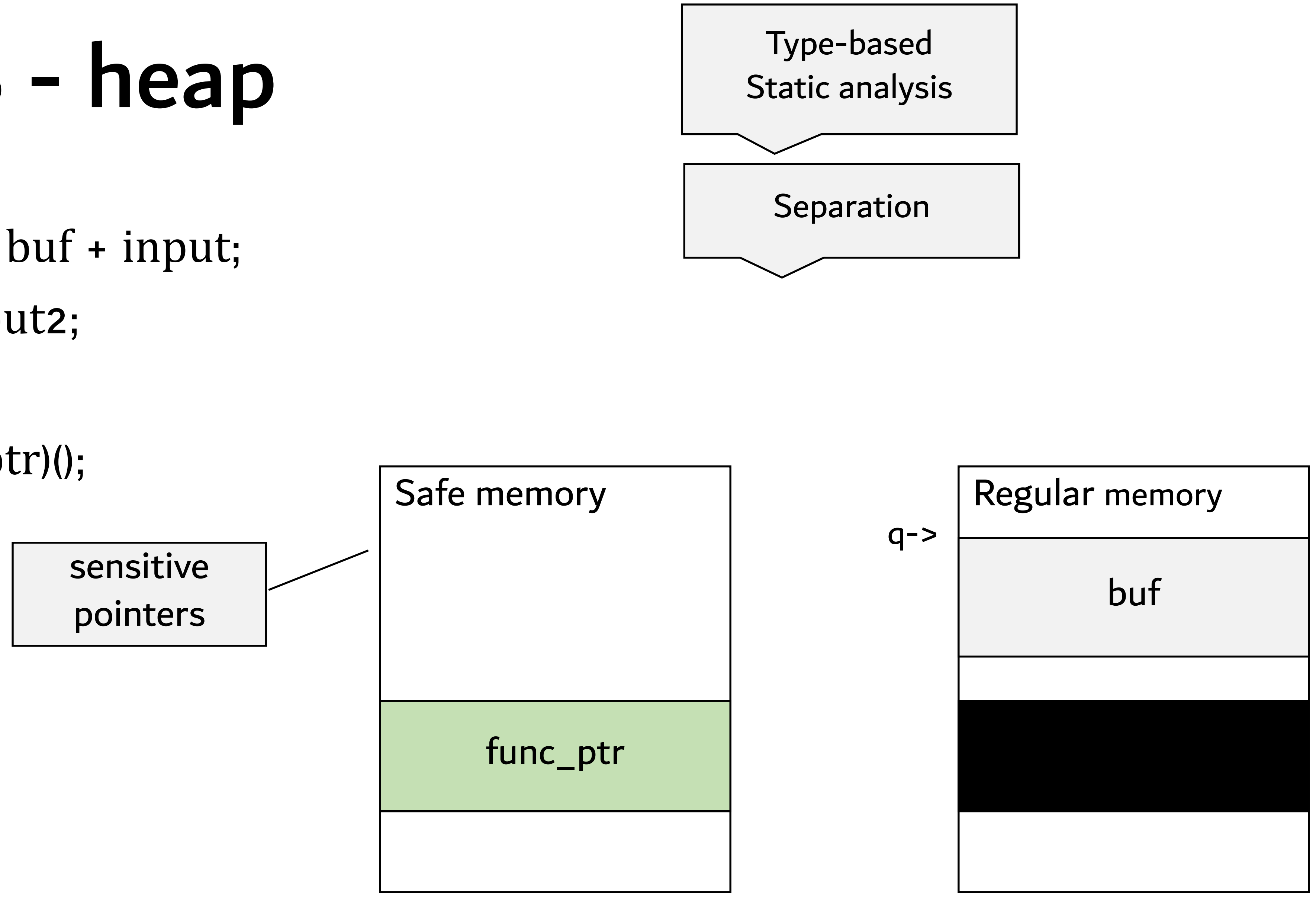
Type-based
Static analysis

```
int *q = buf + input;
*q = input2;
…
(*func_ptr)();
```



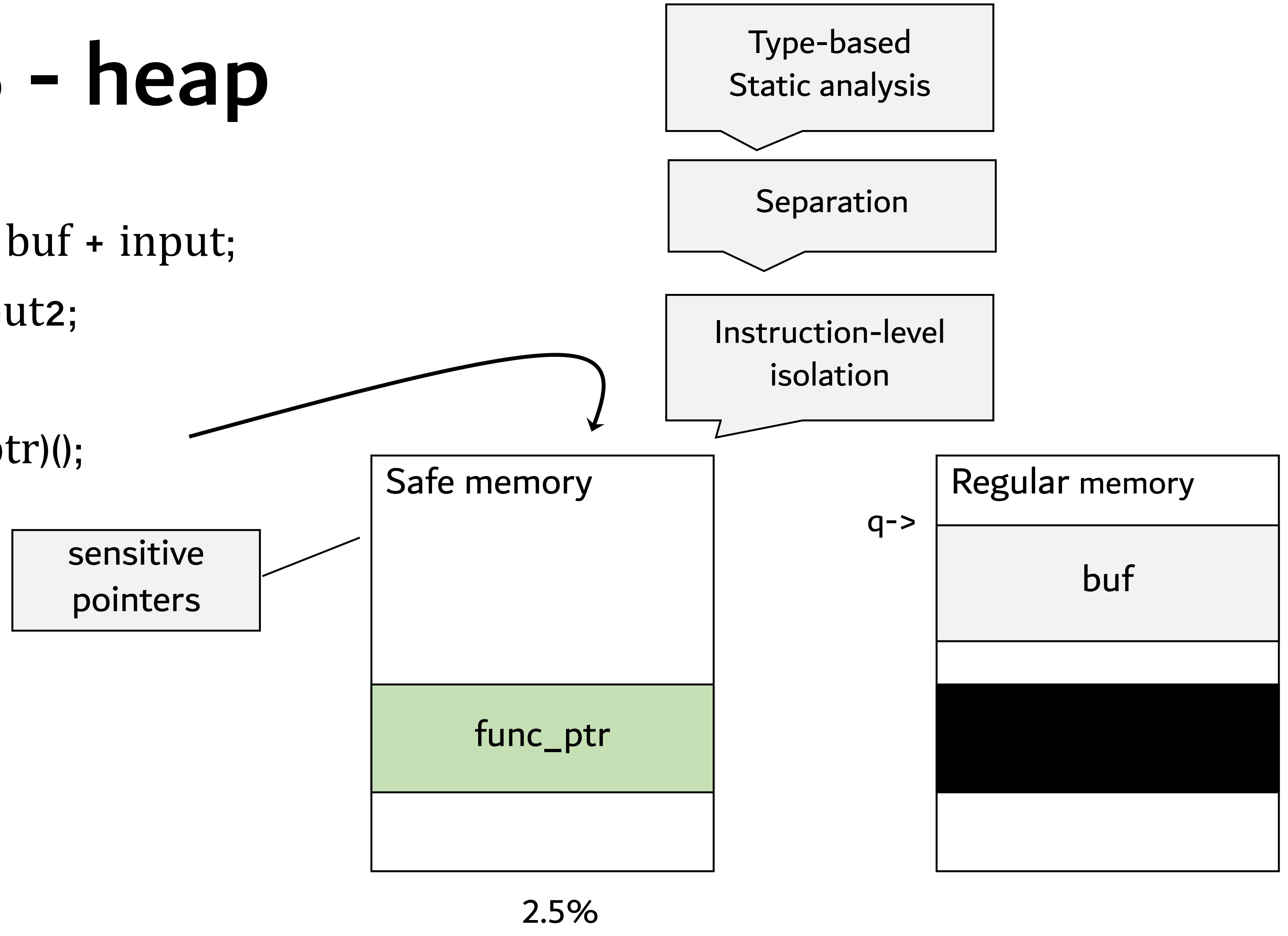
CPS - heap

```
int *q = buf + input;
*q = input2;
…
(*func_ptr)();
```



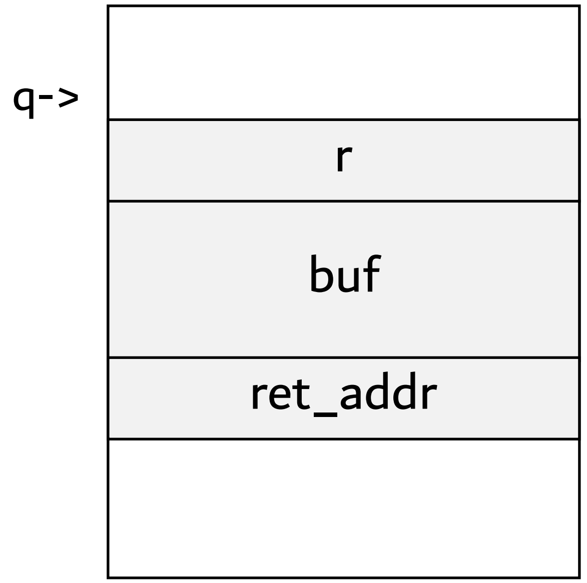
CPS - heap

```
int *q = buf + input;
*q = input2;
…
(*func_ptr)();
```



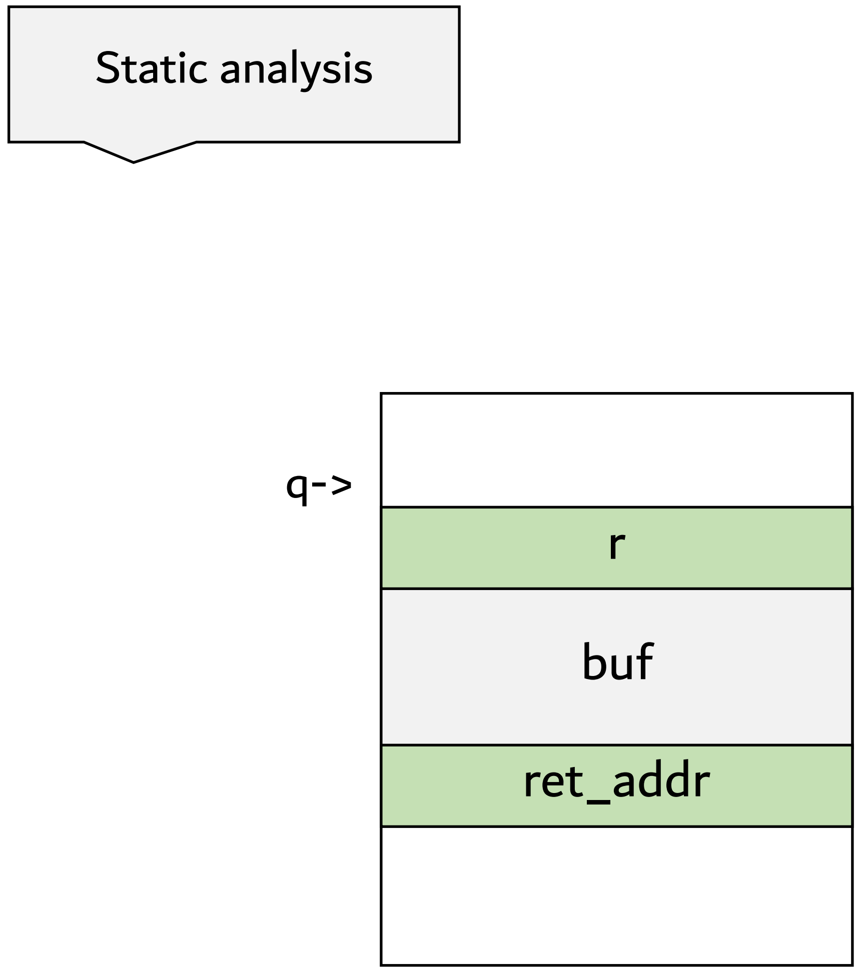
CPS - Stack

```
int foo() {
    char buf[16];
    int r;
    r = scanf("%s", buf);
    return r;
}
```



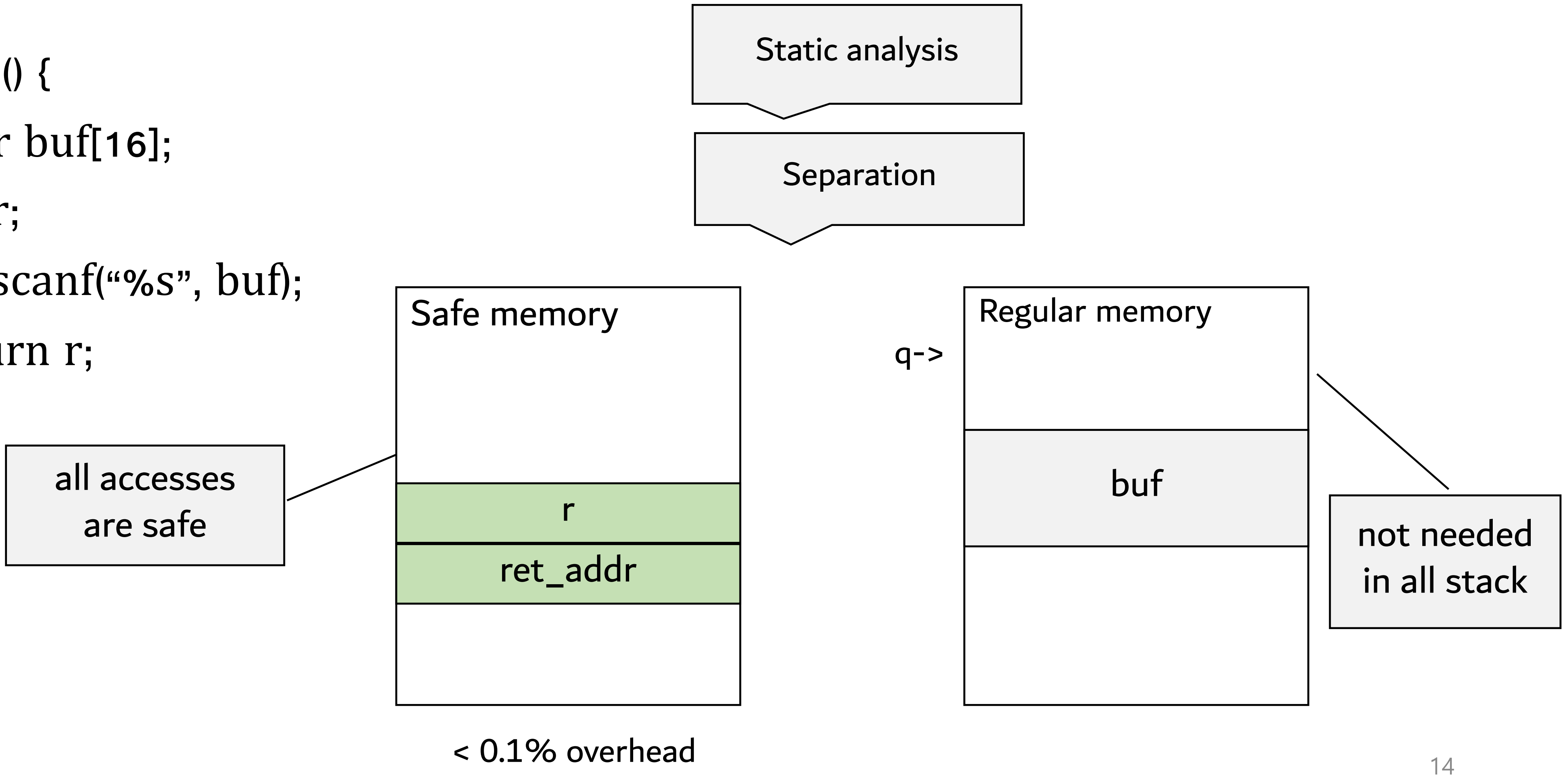
CPS - Stack

```
int foo() {
    char buf[16];
    int r;
    r = scanf("%s", buf);
    return r;
}
```

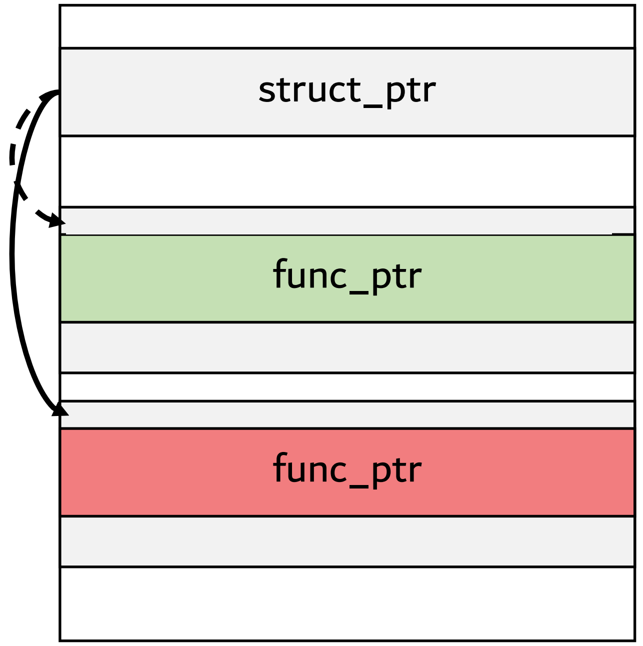


CPS - Stack

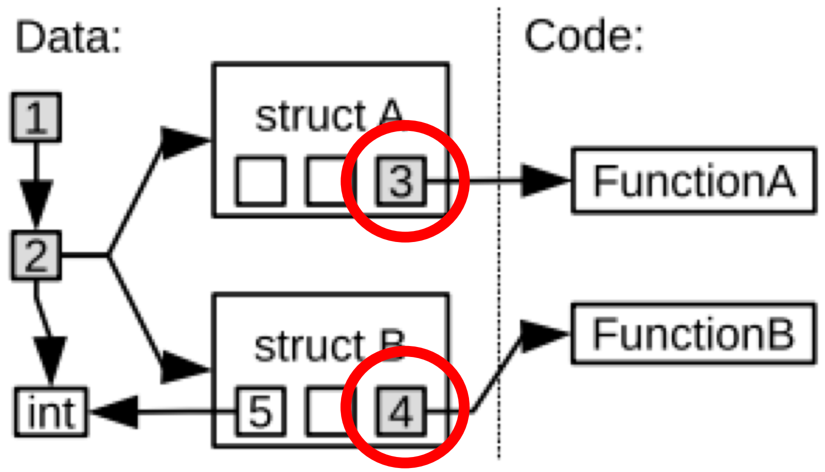
```
int foo() {
    char buf[16];
    int r;
    r = scanf("%s", buf);
    return r;
}
```



Are code pointers enough?



Define sensitive pointer (CPI)

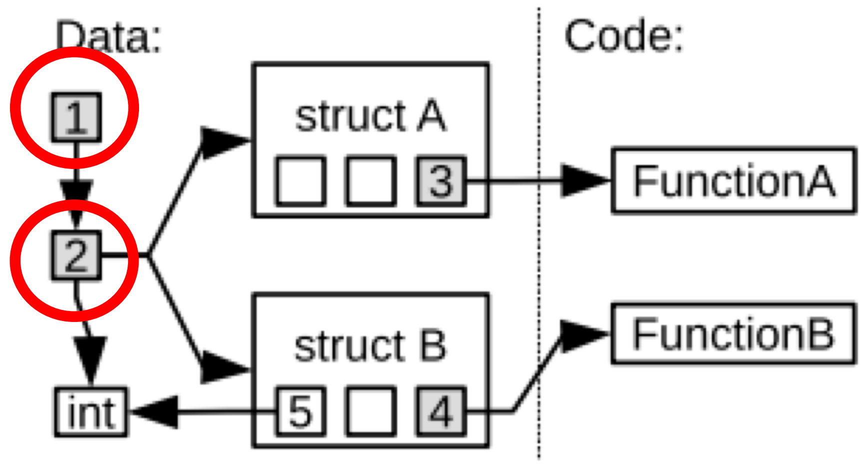


Code pointers

+

Pointers
access sensitive pointer
indirectly

Define sensitive pointer (CPI)

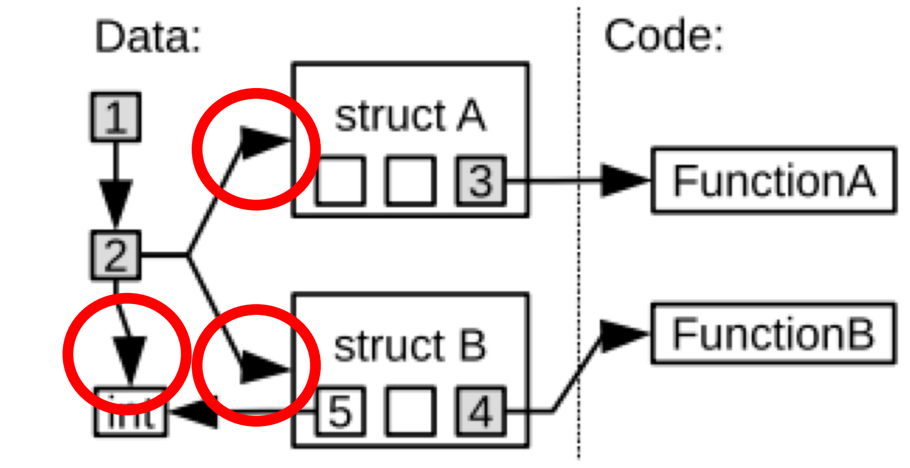


Code pointers

+

Pointers
access sensitive pointer
indirectly

Define sensitive pointer (CPI)



Code pointers

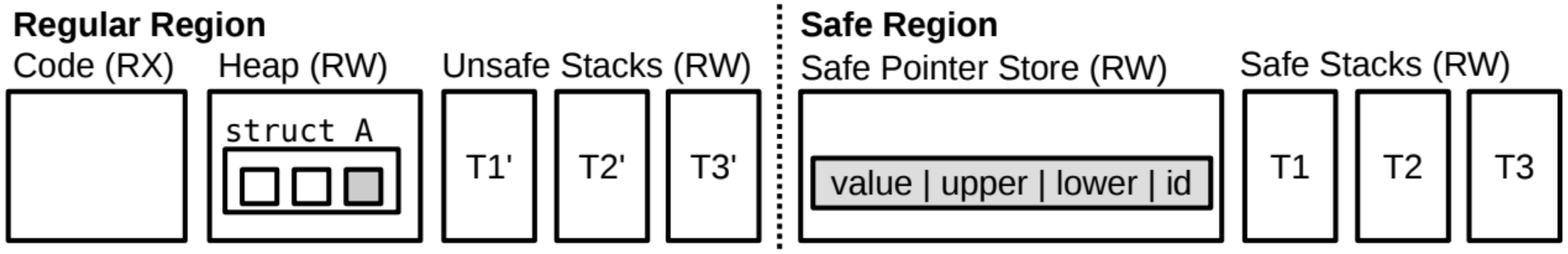
+

Pointers
access sensitive pointer
indirectly

Dynamic!

-> over-approximation using type-based static analysis

Separate memory (CPI)



- Safe Region = Safe Pointer Store + Safe Stacks

- If p = sensitive pointer

  Safe Pointer Store : &p -> p, metadata

  Heap : &p -> empty

CPS vs CPI

	CPS	CPI
sensitive pointers	code pointers	code pointers **+ indirect pointer to sensitive pointers**
safe region	separation	separation **+ runtime check**
regular region	nothing (instruction-level isolation)	

CPS vs CPI

Change code pointer
to location
stored in safe region

	CPS	CPI
sensitive pointers	code pointers	code pointers **+ indirect pointer to sensitive pointers**
safe region	separation	separation **+ runtime check**
regular region	nothing (instruction-level isolation)	

Implementation

Based on LLVM 3.3 compiler

Pass flags to enable CPI(-fcpi), CPS(-fcps)

How secure is it?

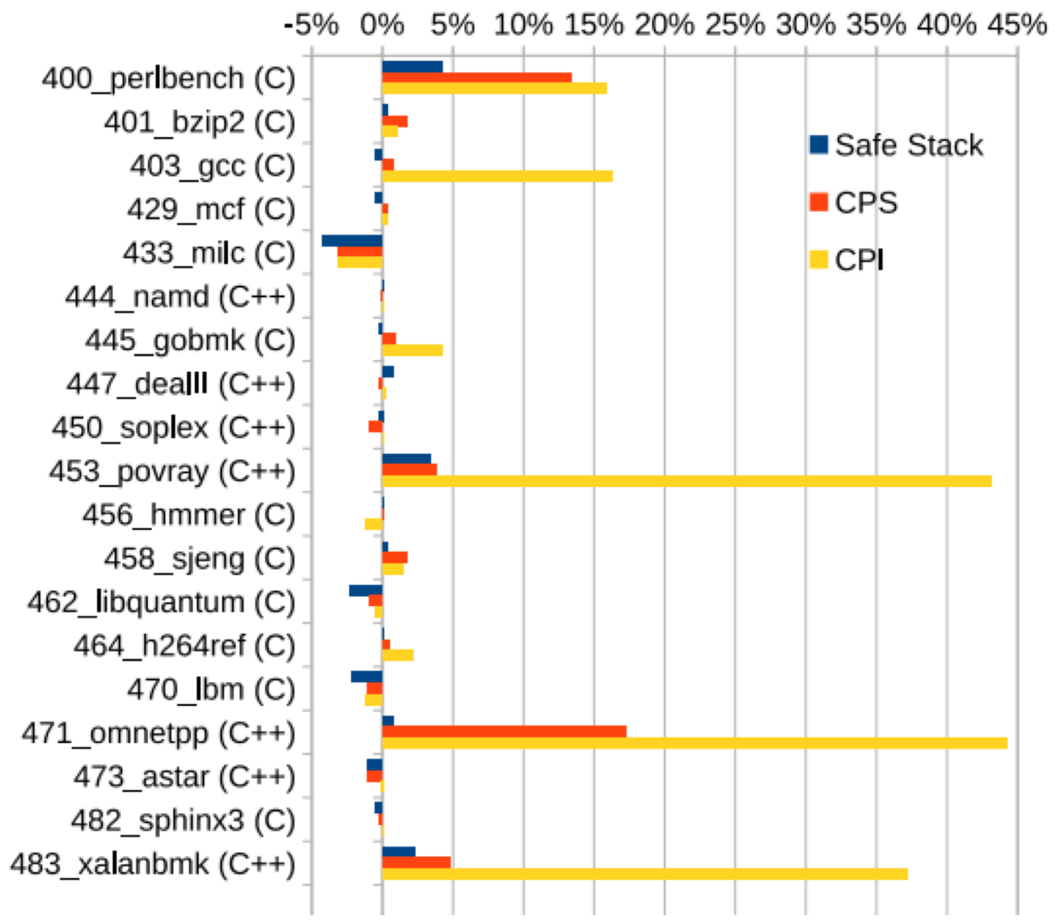
- With RIPE benchmark

  : Both CPI and CPS prevent all attack

- Future attacks

  : CPI correctness proof in paper

Performance



SPEC CPU2006 performance overhead

Average overhead

CPI: 8.4%

CPS: 1.9%

Safe Stack: < 0.1%

vs

Avg. CFI: 21%

SoftBound: > 60~200%

Performance

FreeBSD

+ more than 100 packages

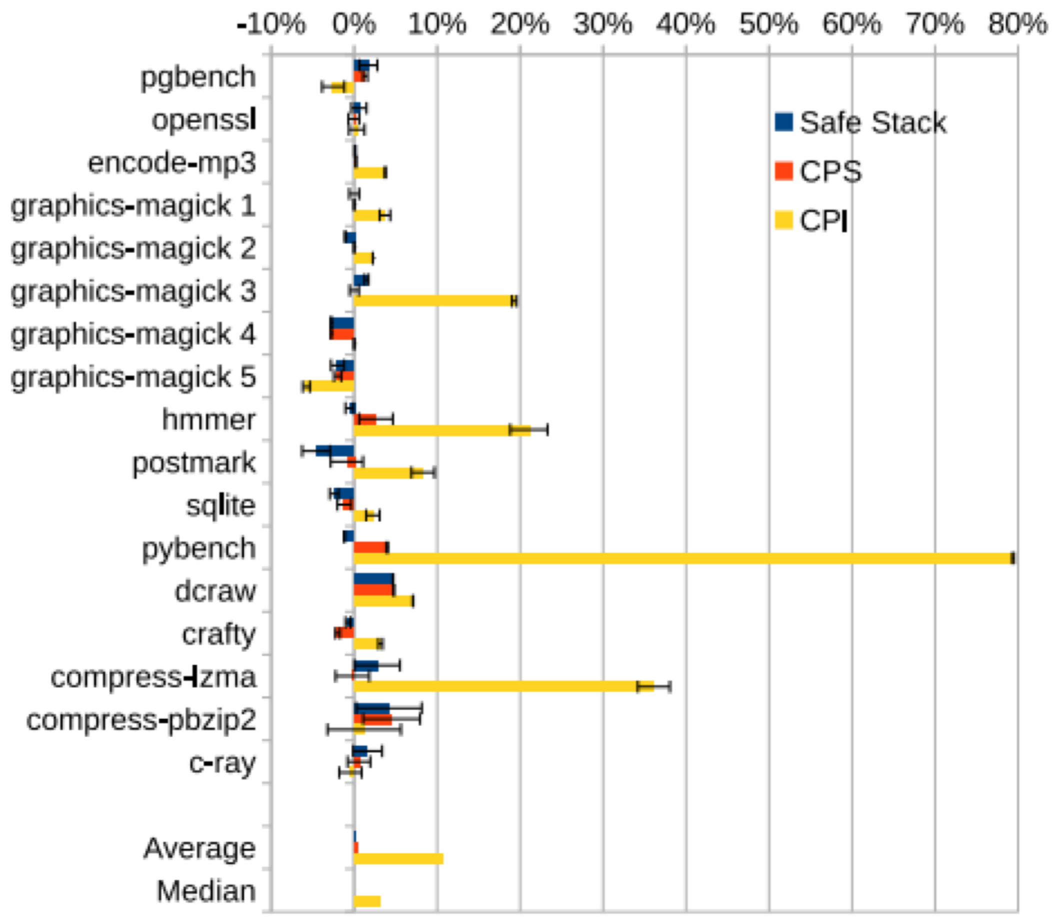
python™

THE APACHE™ SOFTWARE FOUNDATION

SQLite

django

Performance



FreeBSD (Phoronix) performance overhead

Average overhead

CPI: 10.5%

CPS: 0.5%

Safe Stack: 0.01%

Conclusion

: focus on code pointer only

- Secure against all control-flow hijacks
- No change in source code
- Low overhead (0.5-1.9 %, CPS)
  (8.4-10.5 %, CPI)

Question?