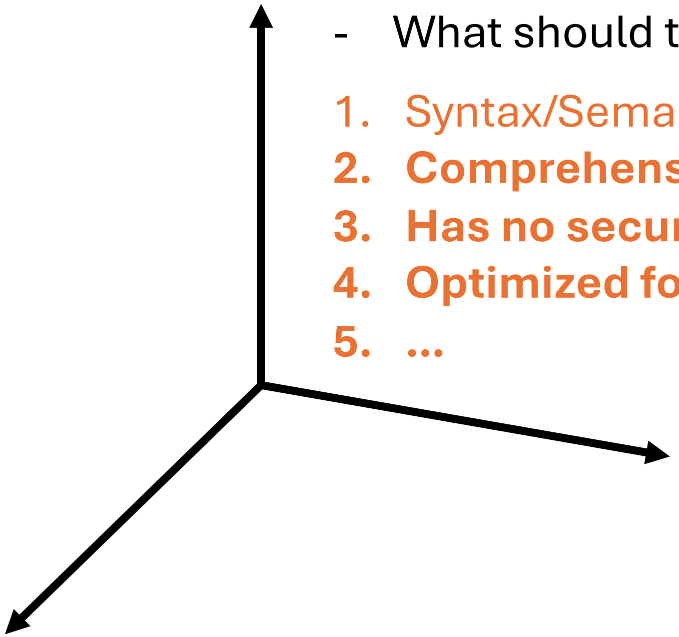
Machine Programming

Lecture 18 – Programming Languages for Software Security

Logistics – Week 10

- Oral Presentations
  - Emails are being sending out; plans established
- Final Projects
  - Final project proposal: 1 page PDF (due on Sunday)
  - Submit on GradeScope
  - Send email to the instructor questions

Module 3: Overview



Behavioral Specification

- What should the program do?

1. Syntax/Semantics/Functional correctness
2. **Comprehensive test coverage**
3. **Has no security flaw**
4. **Optimized for runtime speed**
5. **...**

Structural Specification

- What is the space of the programs?

General Purpose Programming Language

Python / Java / C / Rust / ...

Domain Specific Languages

SQL / LEAN / ROCQ / DATALOG / PDDL / ...

Synthesis Strategy

- How do we find such a program?

Enumeration: Bottom-up and Top-Down

Programming with Large Language Models

- Next token prediction, prompting, controlled decoding
- Iterative refinement, agentic frameworks and tool use
- Pre-training, fine-tuning, reinforcement learning

Fixed

Correct by Construction

Safe Programming Languages

Desirable Properties

Memory Safety

Side-channel Resistance

Termination

Injection-safety

Functional Assurance

Concurrency Safety

Type Safety

Capability Safety

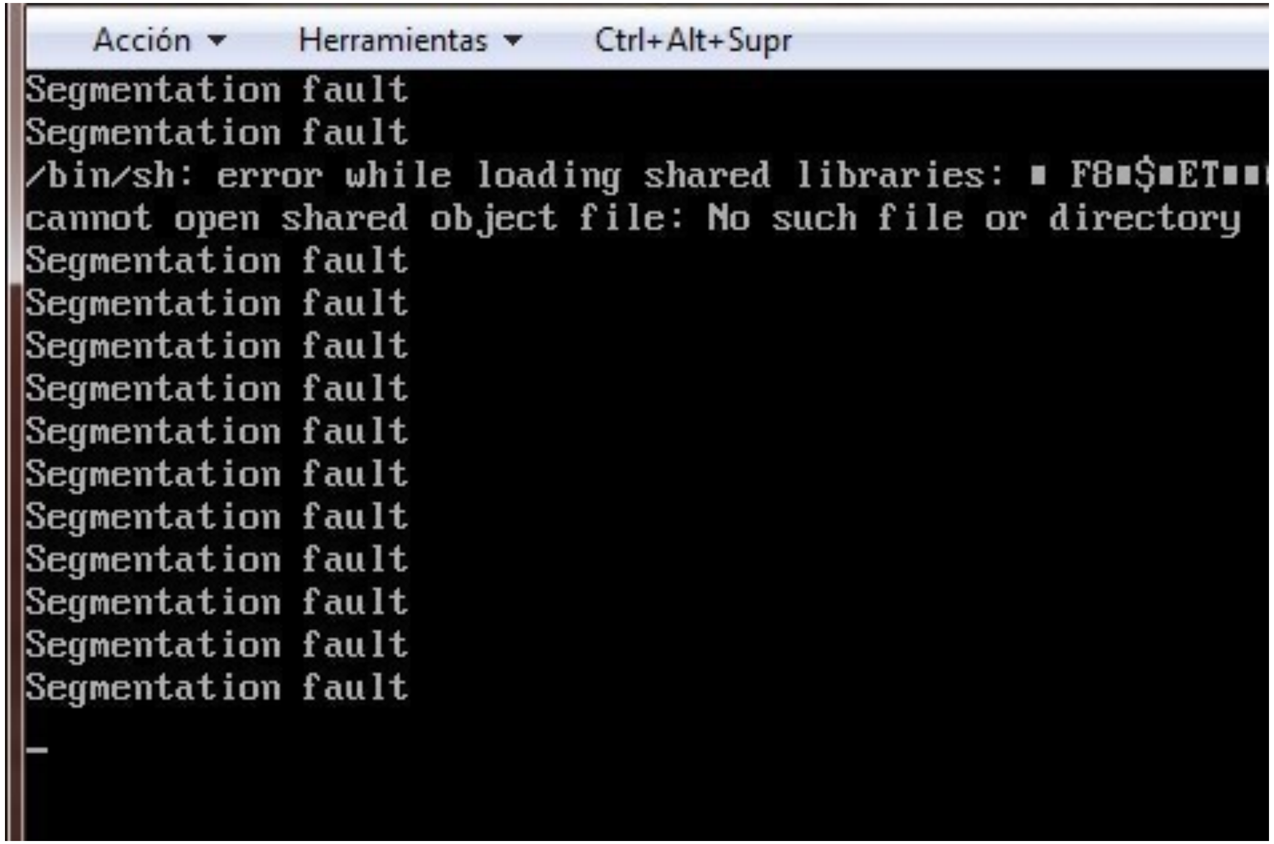
Smart-contract Safety

Control-flow Integrity

Resource Safety

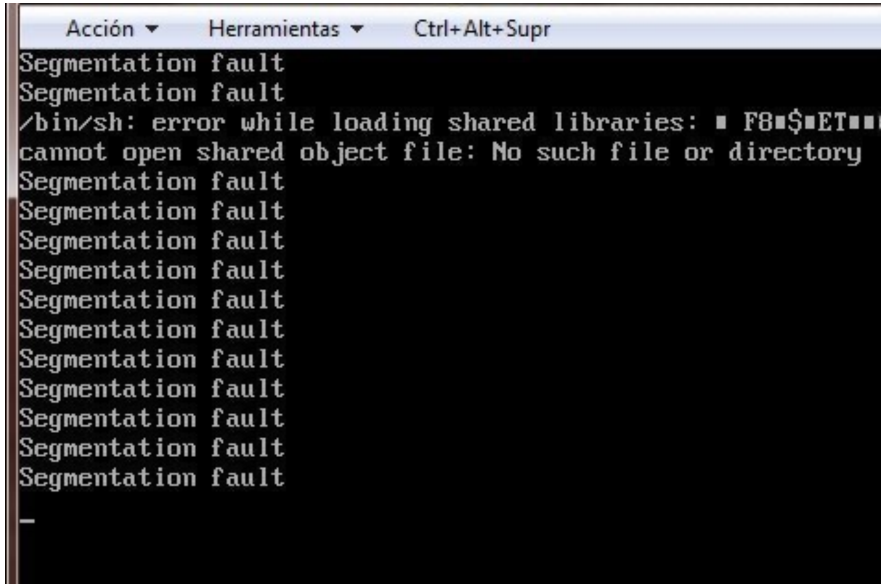
Data Integrity

Memory Safety

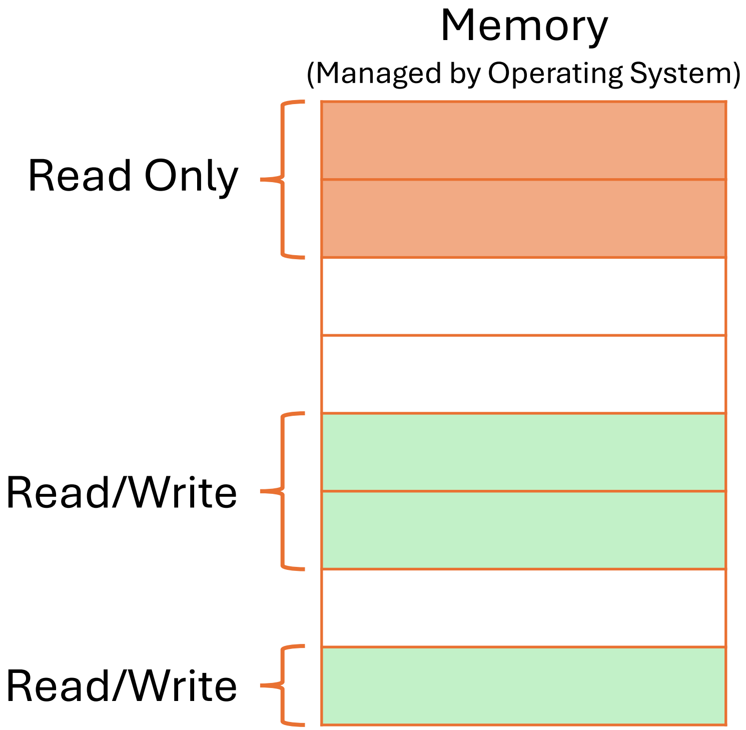


```
Segmentation fault
Segmentation fault
/bin/sh: error while loading shared libraries: ■ F8■$■ET■■
cannot open shared object file: No such file or directory
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
_
```

Memory Safety



```
Segmentation fault
Segmentation fault
/bin/sh: error while loading shared libraries: ■ F8■$■ET■■
cannot open shared object file: No such file or directory
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
_
```


Memory

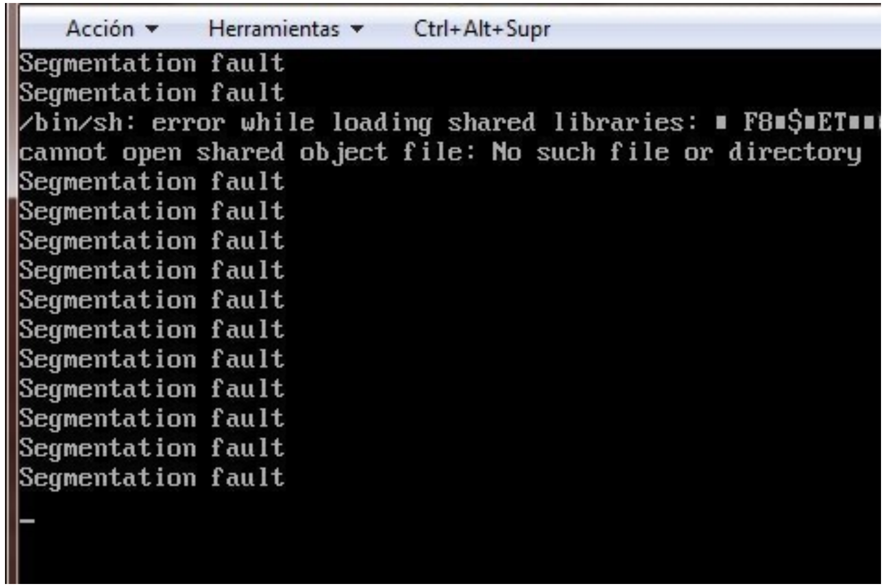
(Managed by Operating System)

Read Only

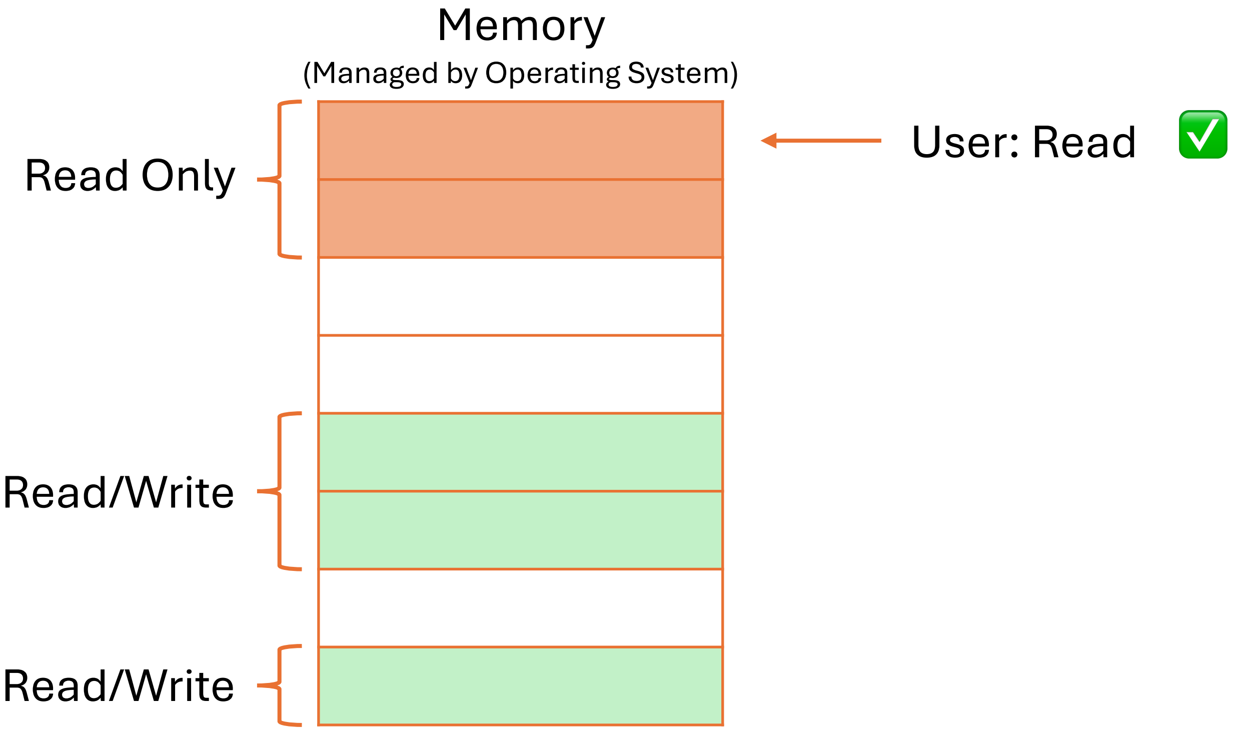
Read/Write

Read/Write

Memory Safety



```
Segmentation fault
Segmentation fault
/bin/sh: error while loading shared libraries: ▪ F8▪$▪ET▪▪
cannot open shared object file: No such file or directory
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
_
```



Memory

(Managed by Operating System)

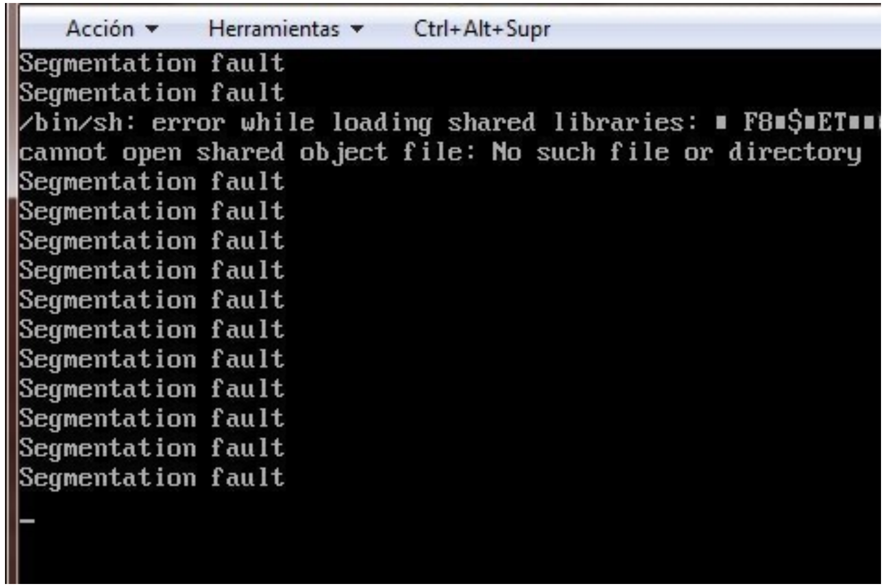
Read Only

Read/Write

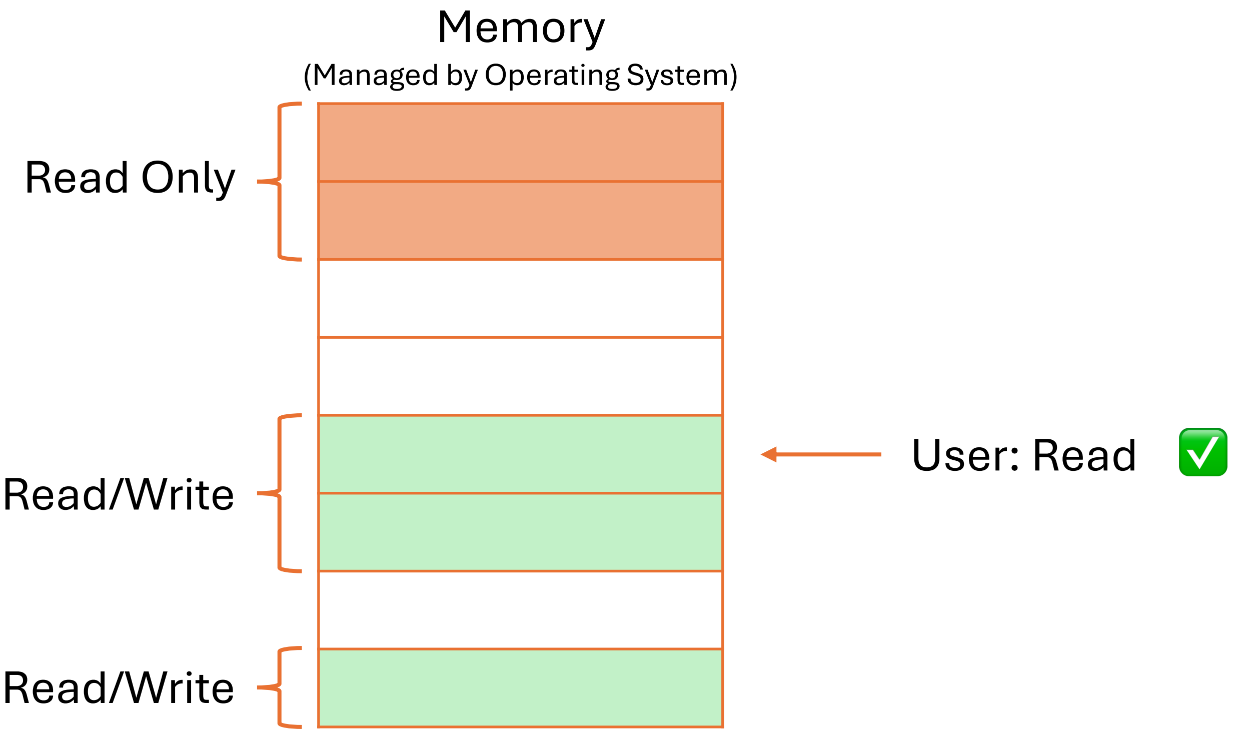
Read/Write

User: Read ✅

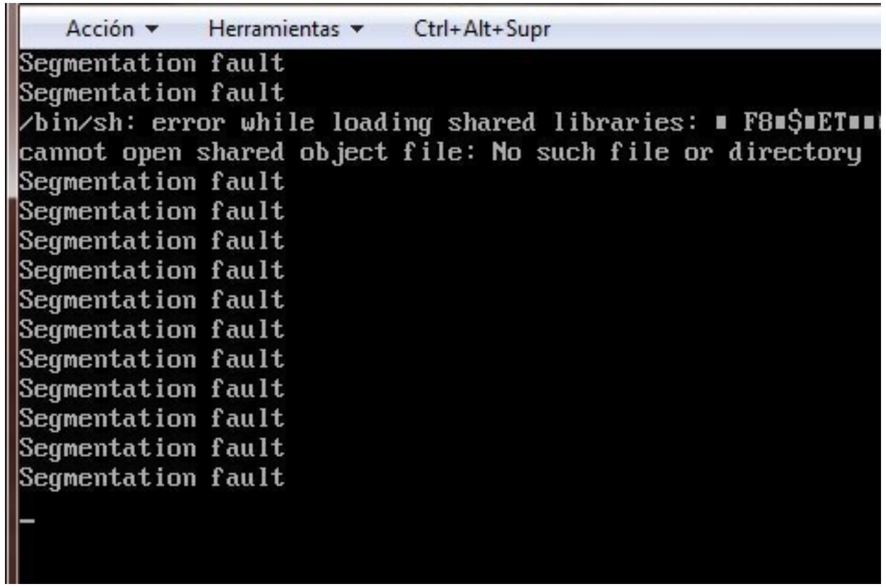
Memory Safety



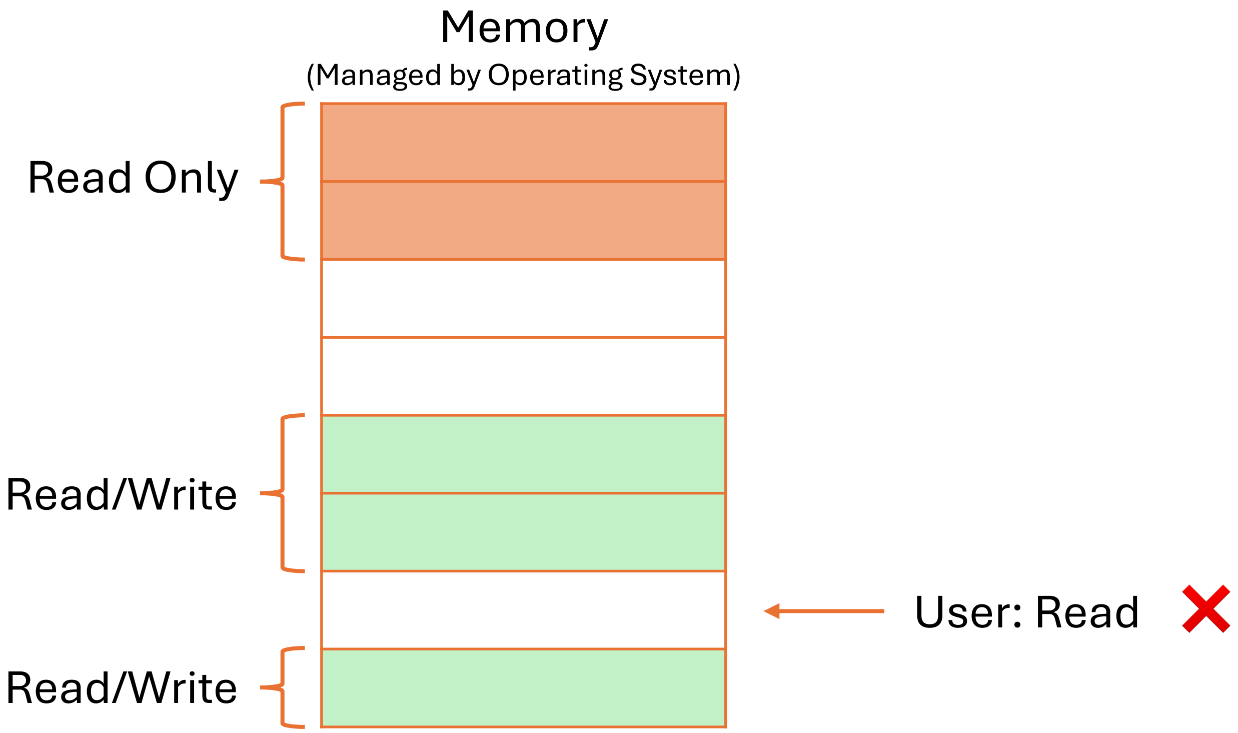
```
Segmentation fault
Segmentation fault
/bin/sh: error while loading shared libraries: ▪ F8▪$▪ET▪▪
cannot open shared object file: No such file or directory
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
_
```



Memory Safety



```
Segmentation fault
Segmentation fault
/bin/sh: error while loading shared libraries: ▪ F8▪$▪ET▪▪
cannot open shared object file: No such file or directory
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
_
```



Memory

(Managed by Operating System)

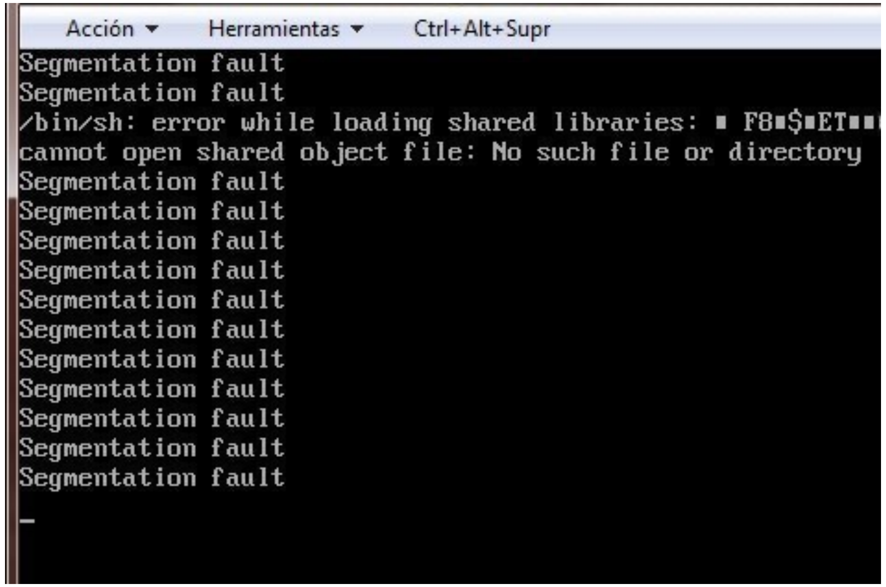
Read Only

Read/Write

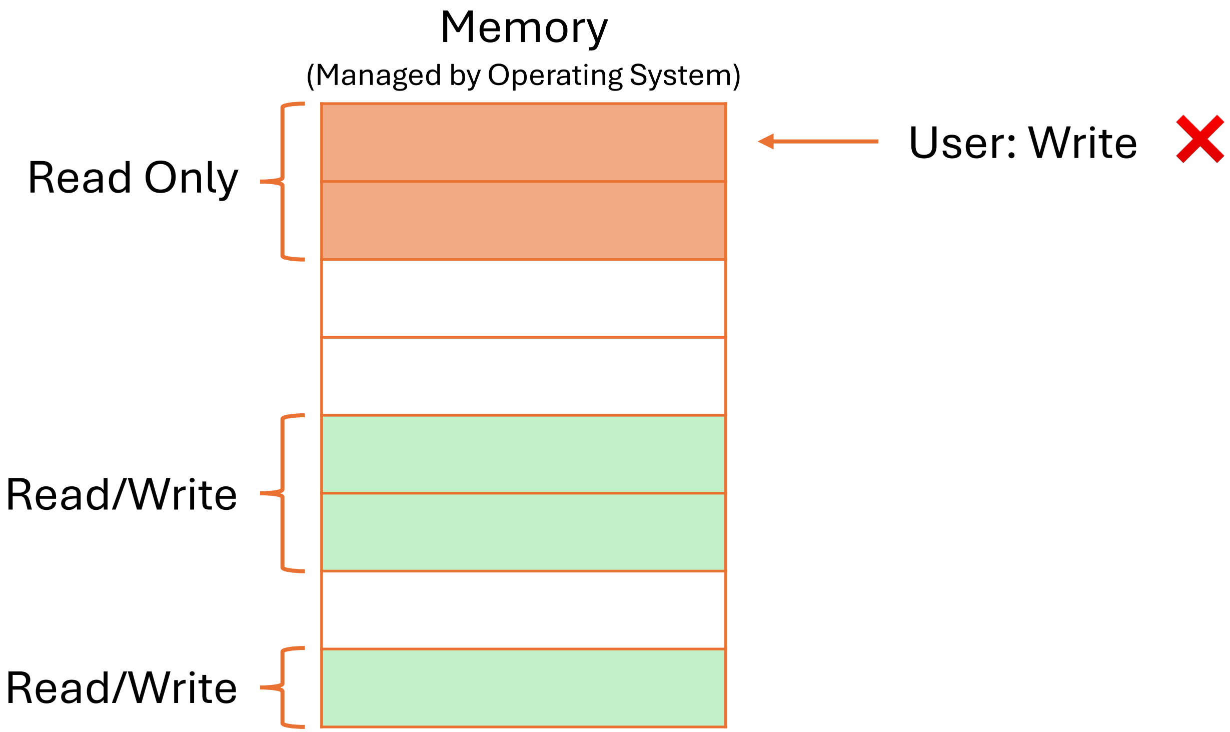
User: Read ✗

Read/Write

Memory Safety



```
Segmentation fault
Segmentation fault
/bin/sh: error while loading shared libraries: ▪ F8▪$▪ET▪▪
cannot open shared object file: No such file or directory
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
Segmentation fault
_
```

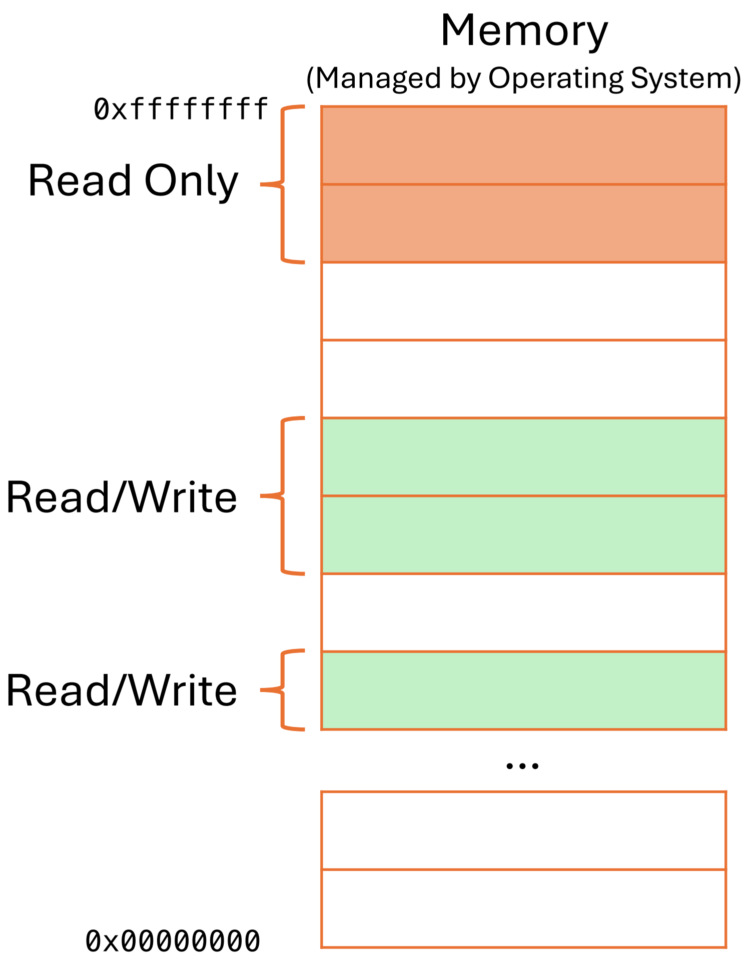


C Program that Breaks Memory Safety

```c
int main() {
  int *p = NULL;
  *p = 42;
}
```

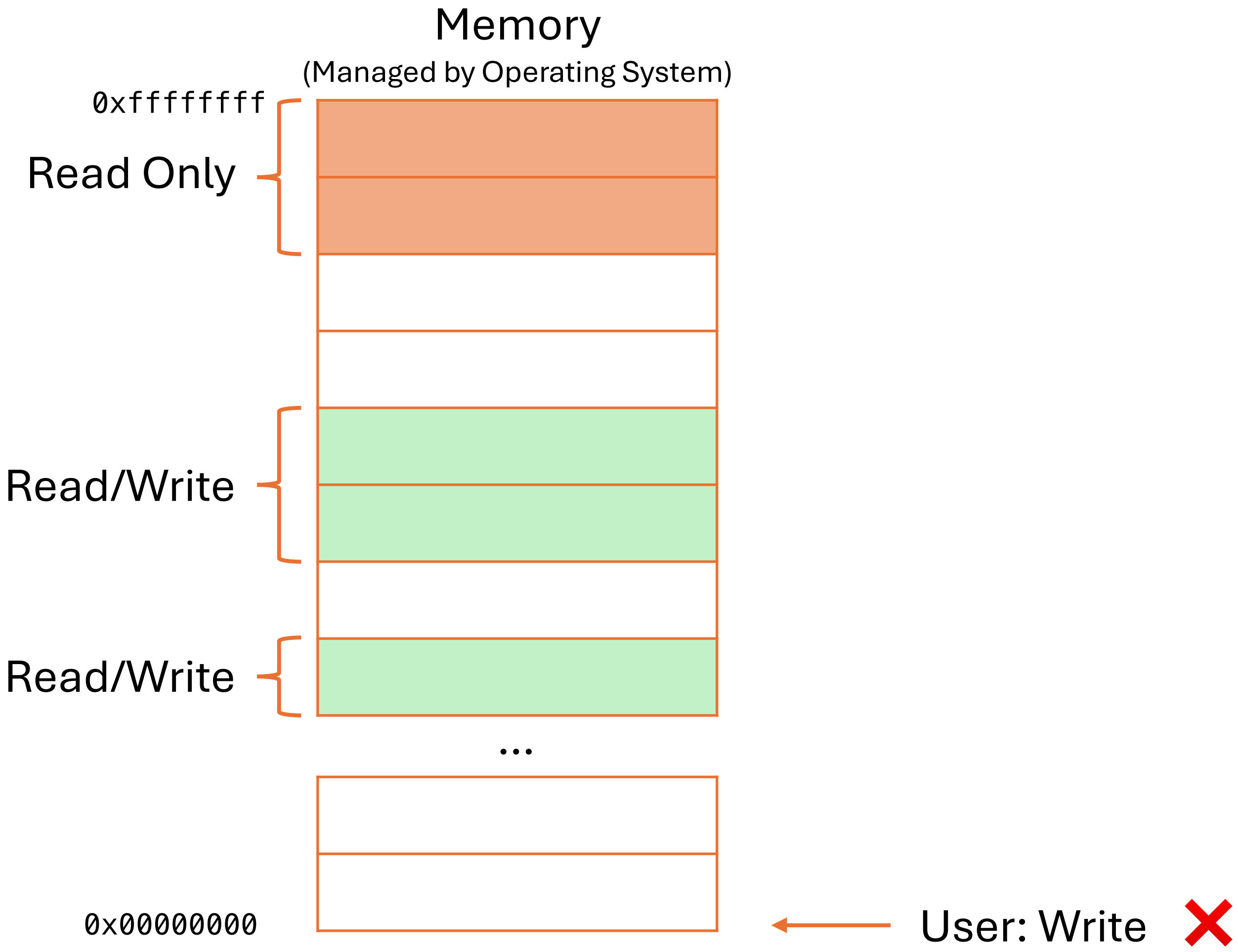
C Program that Breaks Memory Safety

```
int main() {
  int *p = NULL;
  *p = 42;
}
```



C Program that Breaks Memory Safety

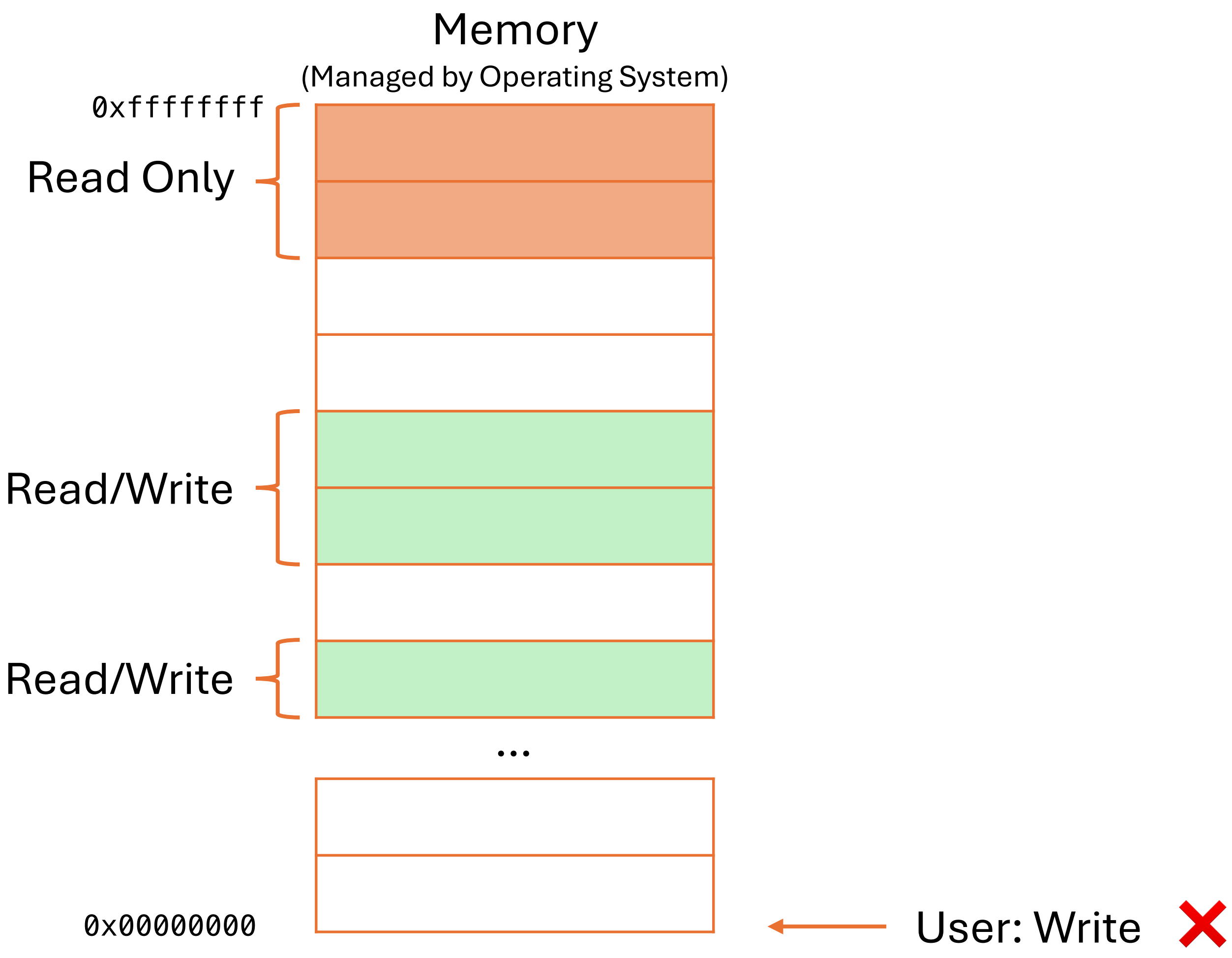
```
int main() {
  int *p = NULL;
  *p = 42;
}
```



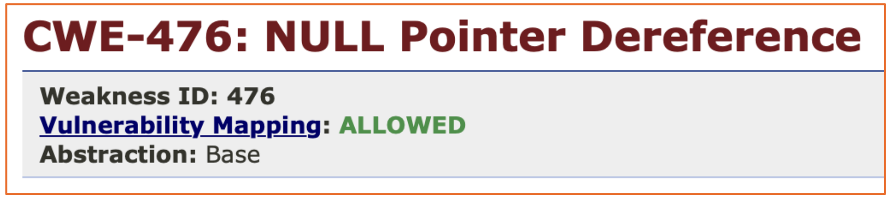
C Program that Noticeably Breaks Memory Safety

```c
int main() {
  int p[42];
  *p = 42;
}
```

```
liby@mac ~/L/P/Demo> gcc demo.c
liby@mac ~/L/P/Demo> ./a.out
fish: Job 1, './a.out' terminated
by signal SIGSEGV (Address boundary
error)
```



C Program that Noticeably Breaks Memory Safety



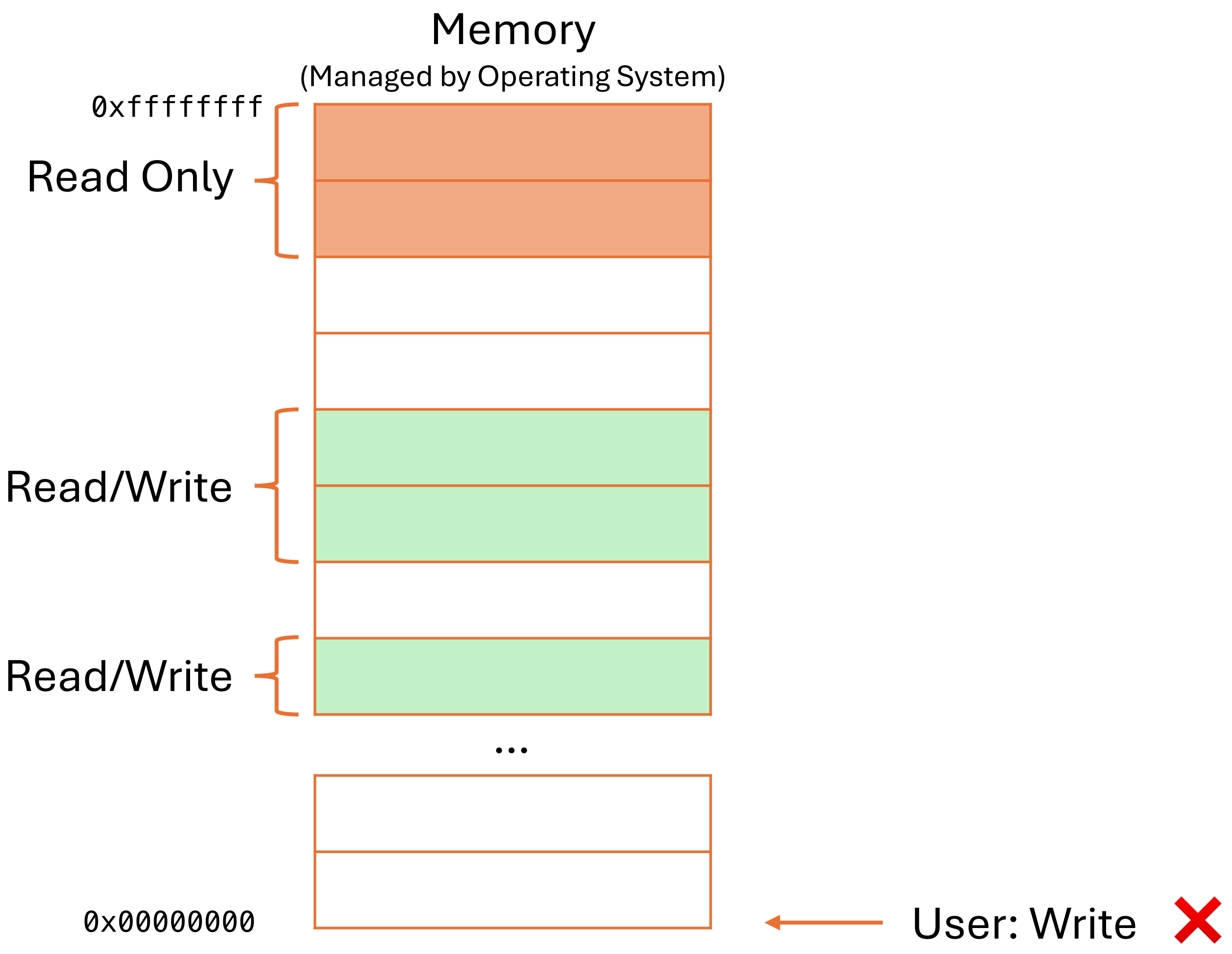
CWE-476: NULL Pointer Dereference

Weakness ID: 476
Vulnerability Mapping: ALLOWED
Abstraction: Base

NULL Pointer Dereference

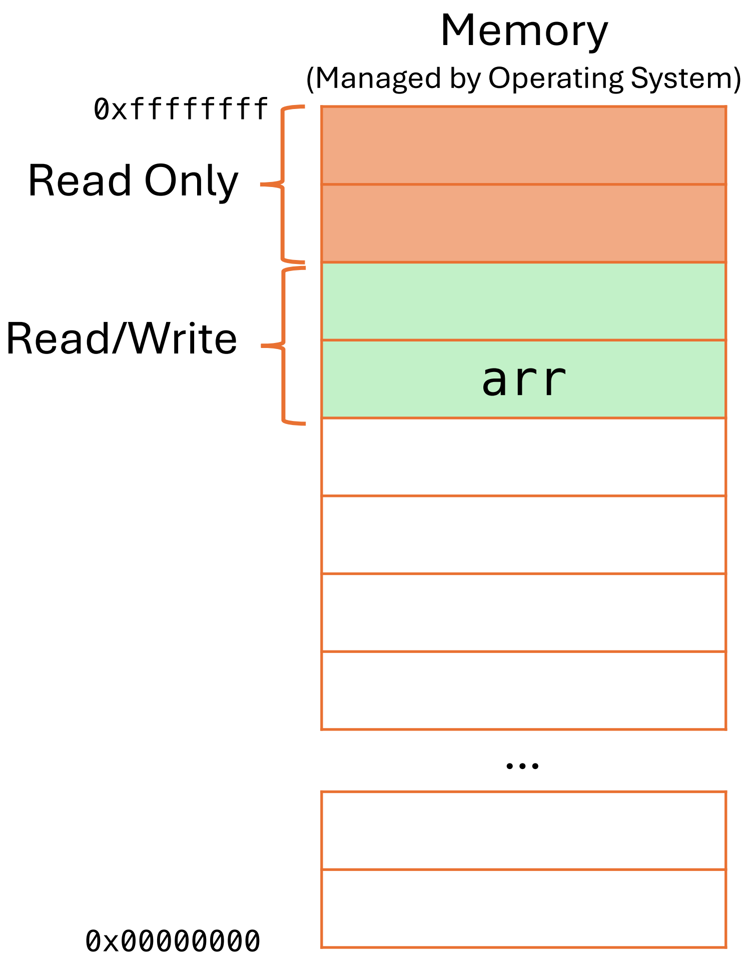
```
nt main() {
   int p[42];
   *p = 42;
}
```

```
liby@mac ~/L/P/Demo> gcc demo.c
liby@mac ~/L/P/Demo> ./a.out
fish: Job 1, './a.out' terminated
by signal SIGSEGV (Address boundary
error)
```



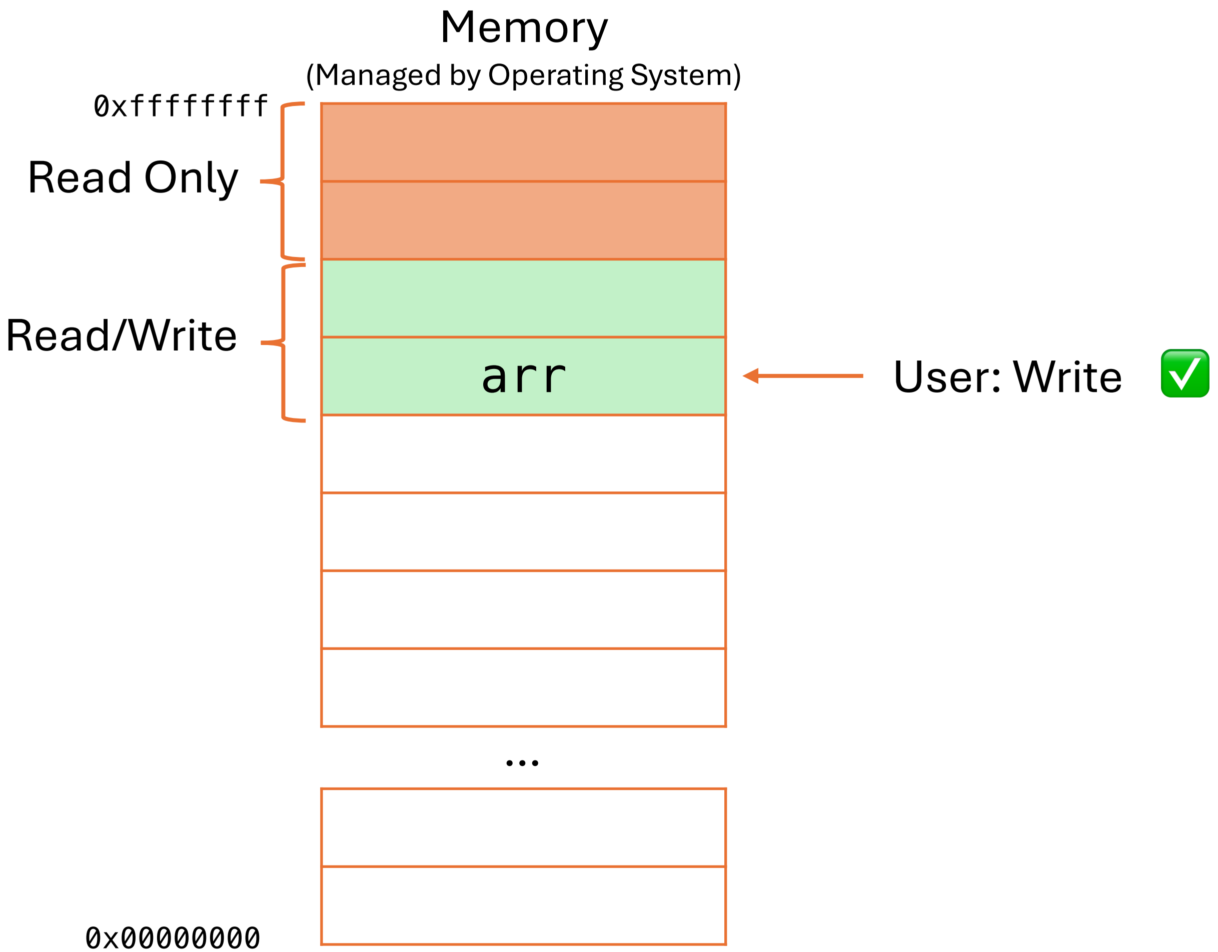
C Program that Silently Breaks Memory Safety

```
int main() {
  int arr[100];
  arr[182] = 42;
}
```



C Program that Silently Breaks Memory Safety

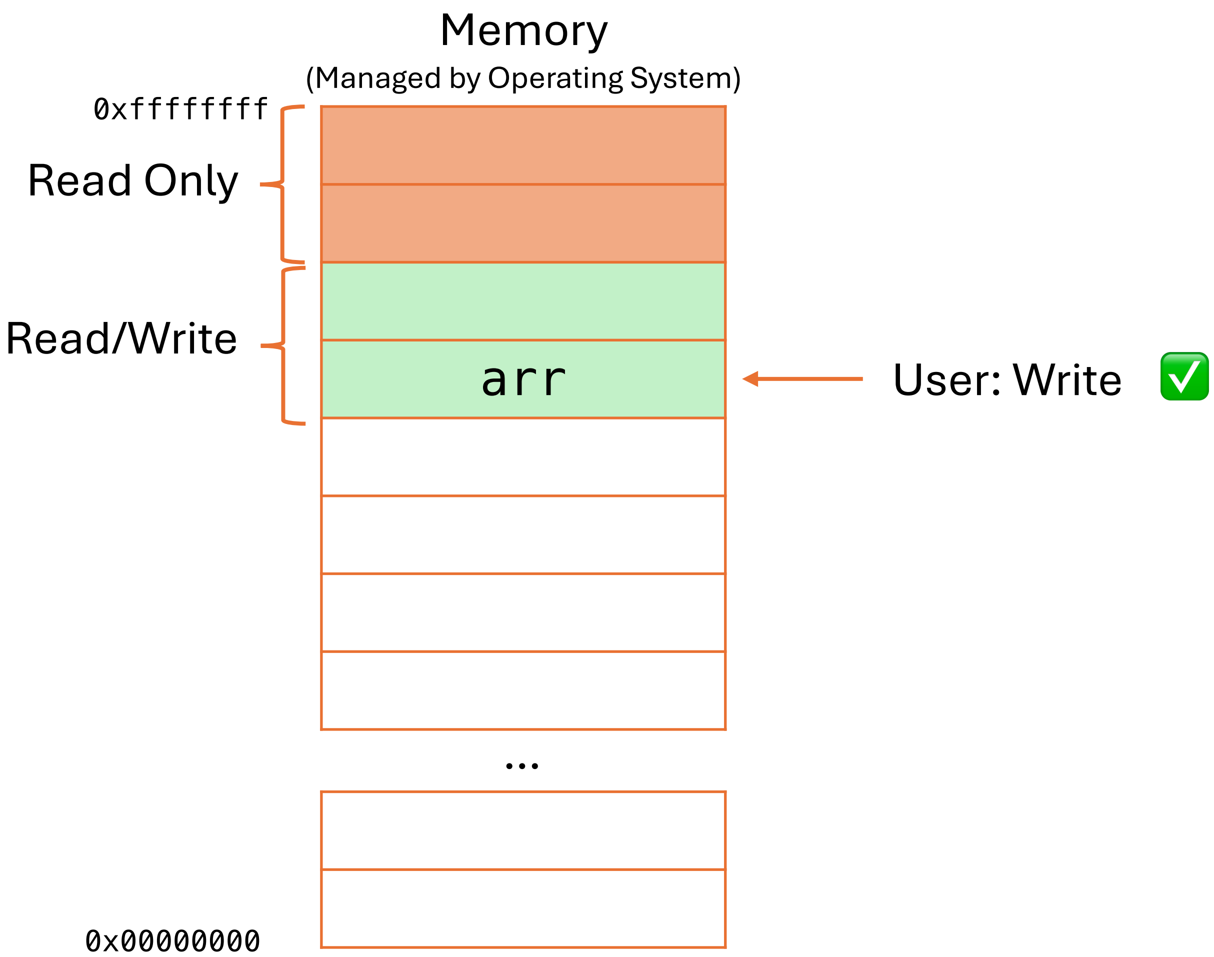
```c
int main() {
  int arr[100];
  arr[182] = 42;
}
```



C Program that Silently Breaks Memory Safety

```c
int main() {
  int arr[100];
  arr[182] = 42;
}
```

```
liby@mac ~/L/P/Demo> gcc demo.c
liby@mac ~/L/P/Demo> ./a.out
```



C Program that Silently Breaks Memory Safety

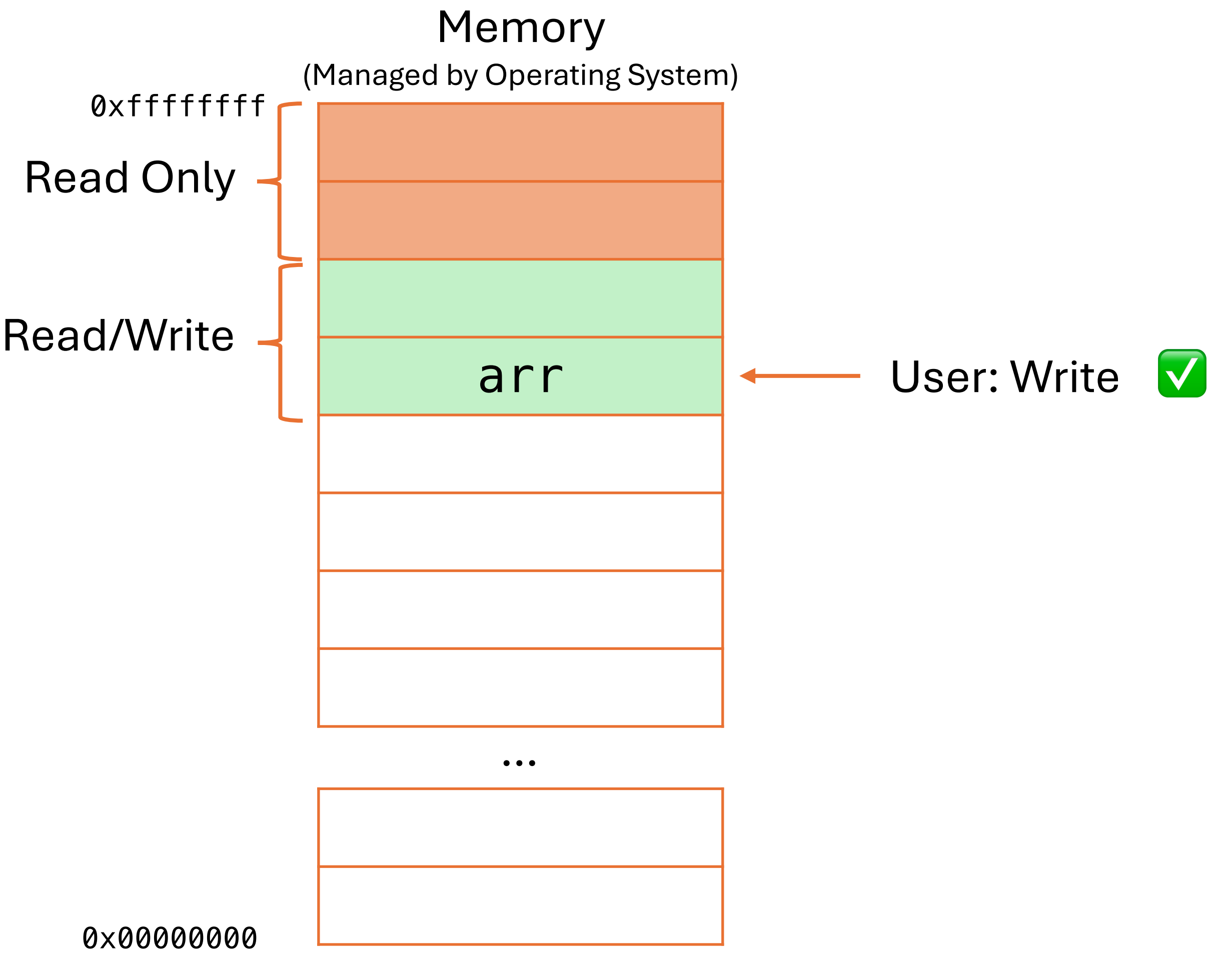
CWE-121: Stack-based Buffer Overflow

Weakness ID: 121
Vulnerability Mapping: ALLOWED
Abstraction: Variant

Buffer Overflow

```
int main() {
   int arr[100];
   arr[182] = 42;
}
```

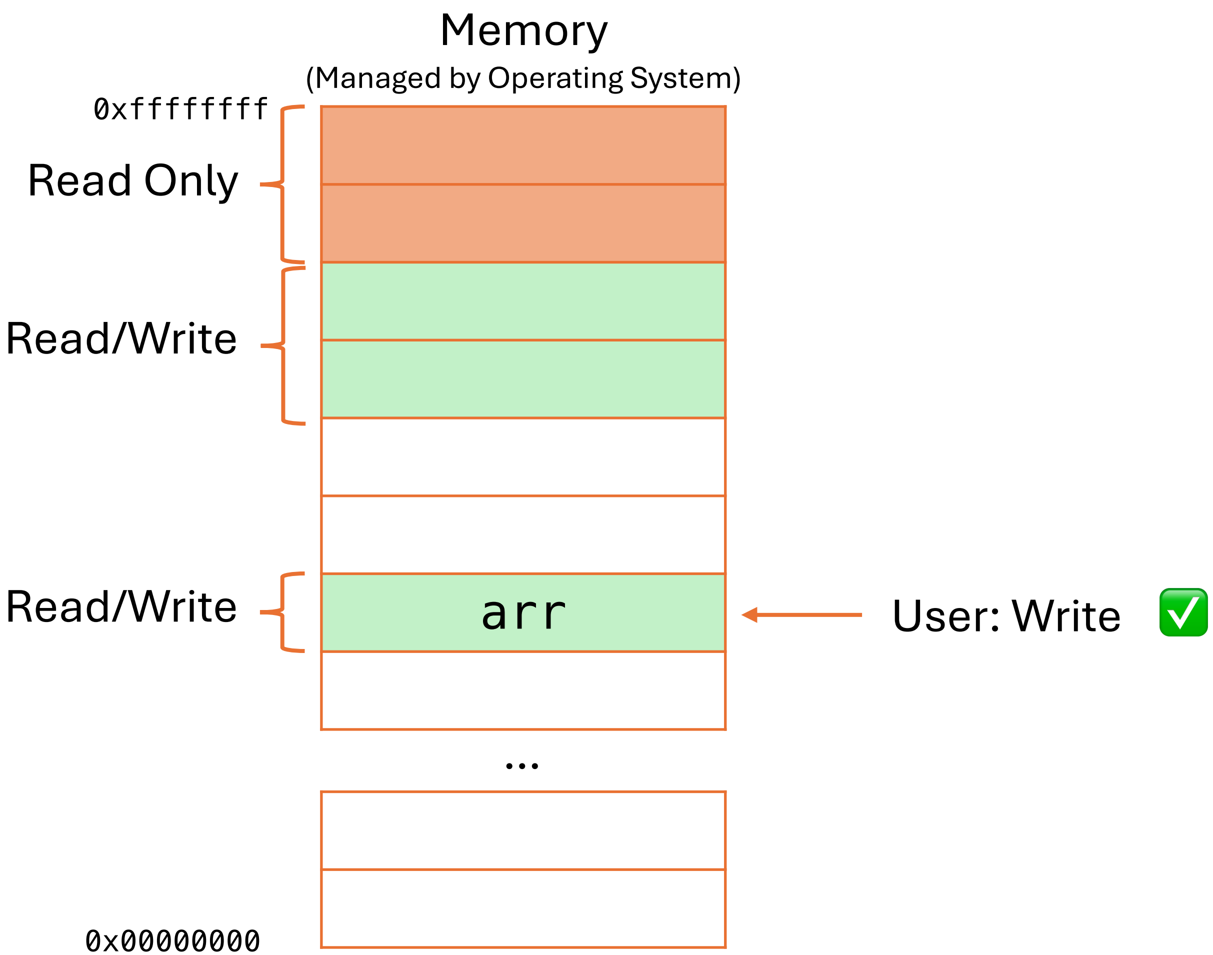
```
liby@mac ~/L/P/Demo> gcc demo.c
liby@mac ~/L/P/Demo> ./a.out
```



C Program that Silently Breaks Memory Safety

```c
int main() {
  int *arr = (int *)
    malloc(100 * sizeof(int));
  arr[182] = 42;
}
```

```
liby@mac ~/L/P/Demo> gcc demo.c
liby@mac ~/L/P/Demo> ./a.out
```



C Program that Silently Breaks Memory Safety

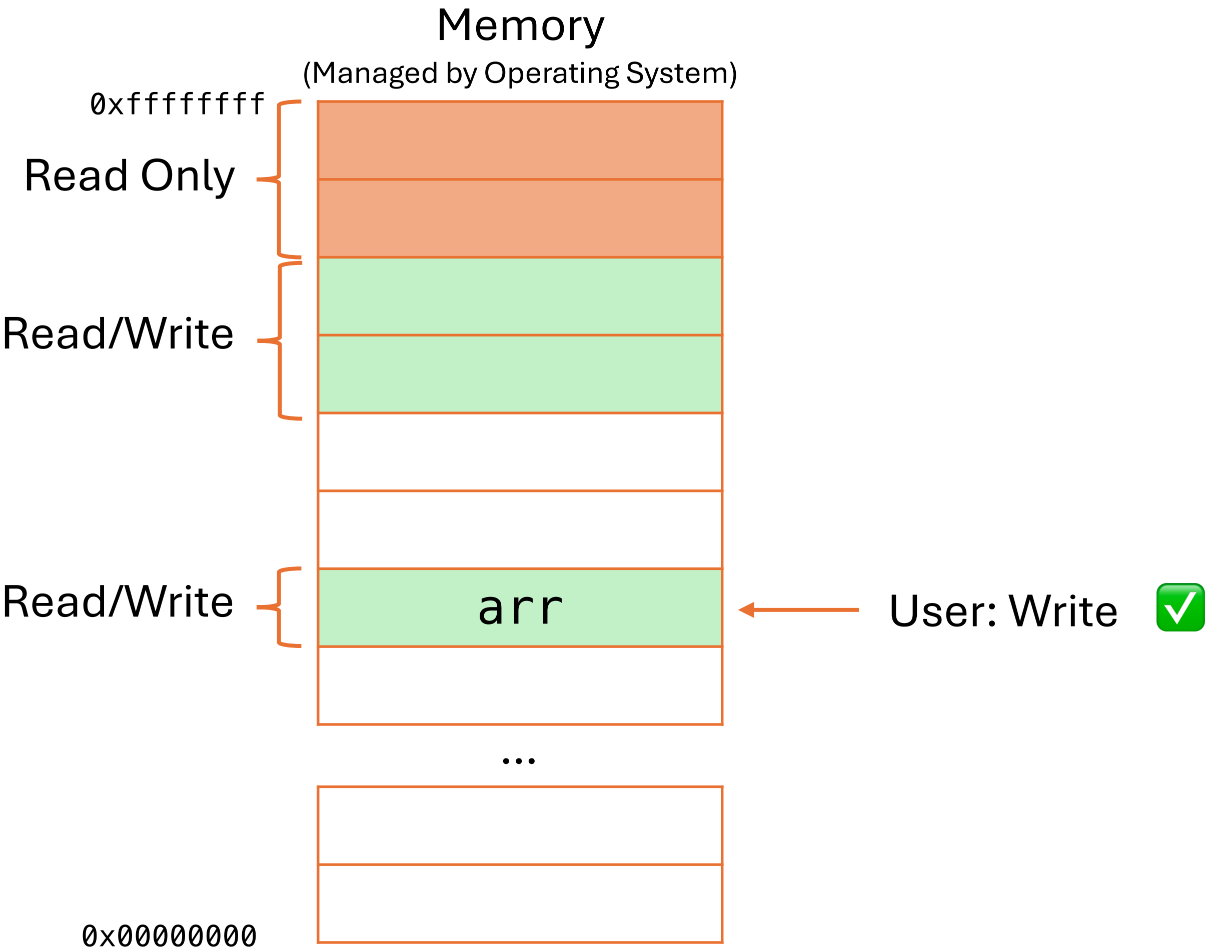
CWE-122: Heap-based Buffer Overflow

Weakness ID: 122
Vulnerability Mapping: ALLOWED
Abstraction: Variant

Buffer Overflow

```
int      () {
  int *arr = (int *)
    malloc(100 * sizeof(int));
  arr[182] = 42;
}
```

```
liby@mac ~/L/P/Demo> gcc demo.c
liby@mac ~/L/P/Demo> ./a.out
```

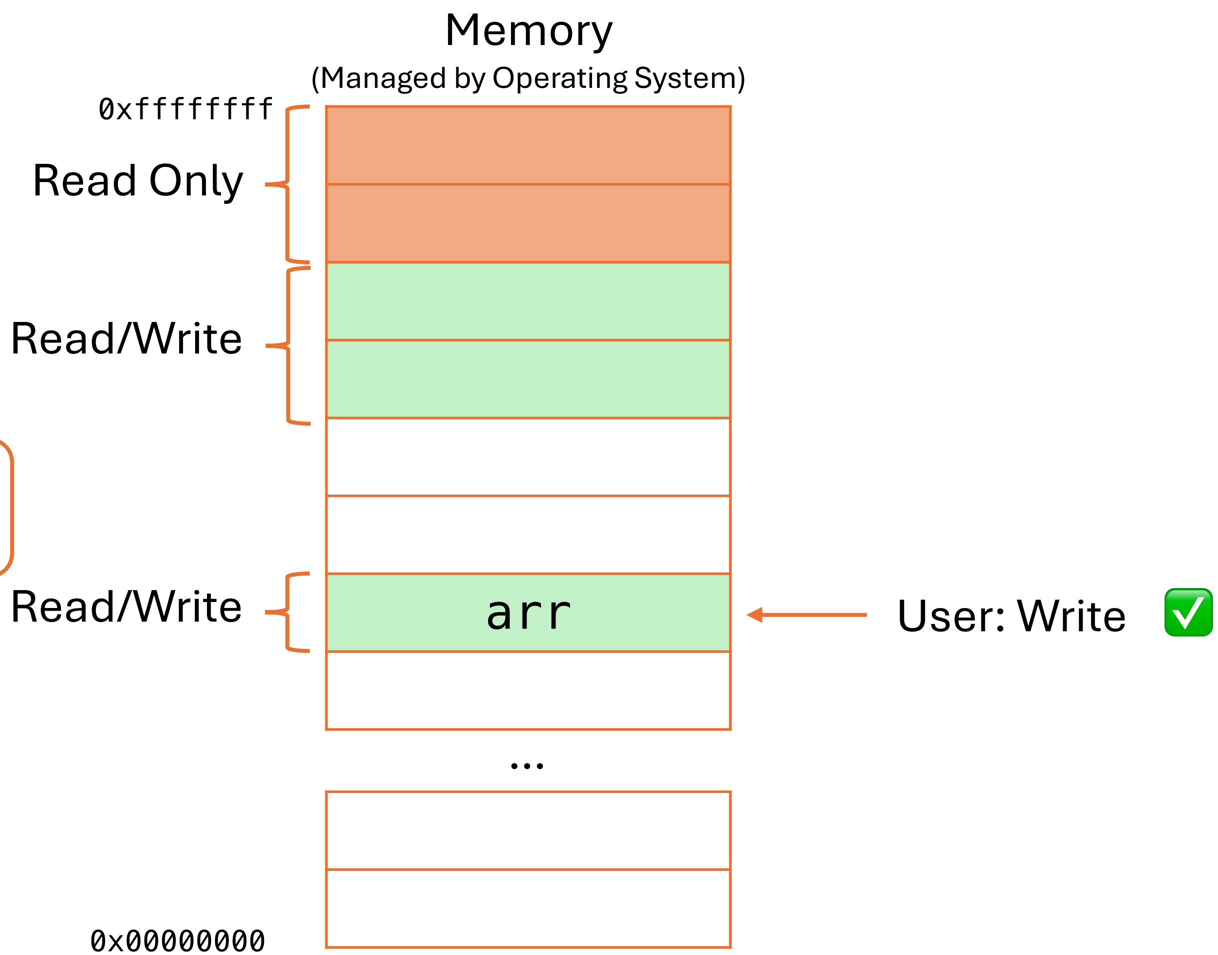


C Program that Silently Breaks Memory Safety

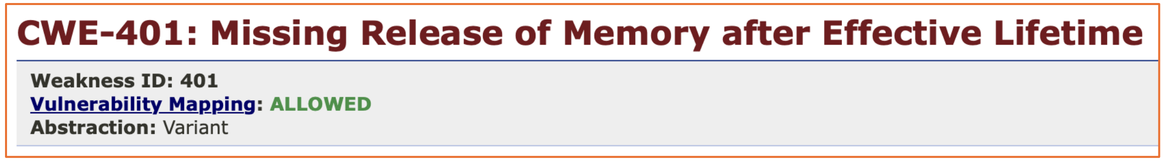
```
int main() {
  int *arr = (int *)
    malloc(100 * sizeof(int));
  arr[182] = 42;
}
```

Another issue with “arr”: Not free-ed

```
liby@mac ~/L/P/Demo> gcc demo.c
liby@mac ~/L/P/Demo> ./a.out
```



C Program that Silently Breaks Memory Safety



CWE-401: Missing Release of Memory after Effective Lifetime

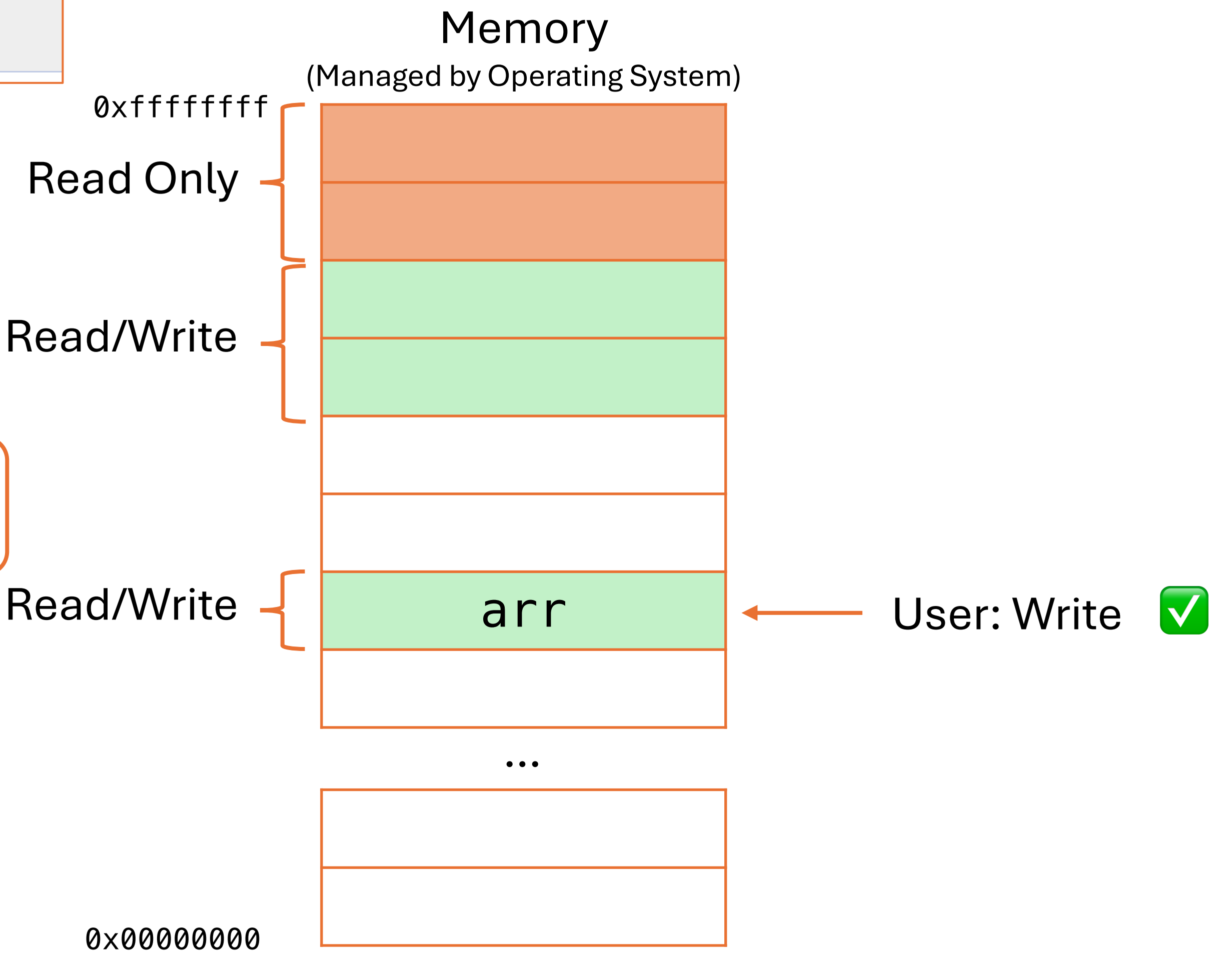
Weakness ID: 401
Vulnerability Mapping: ALLOWED
Abstraction: Variant

Memory Leak

```
int main() {
  int *arr = (int *)
    malloc(100 * sizeof(int));
  arr[182] = 42;
}
```

Another issue with “`arr`”: Not `free`-ed

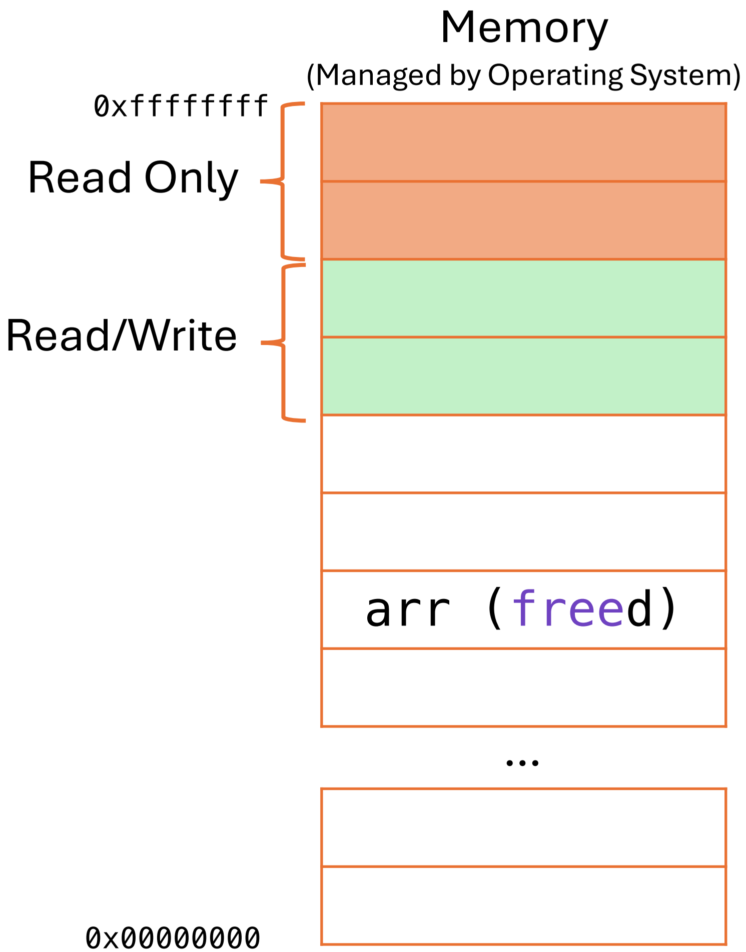
```
liby@mac ~/L/P/Demo> gcc demo.c
liby@mac ~/L/P/Demo> ./a.out
```



C Program that Silently Breaks Memory Safety

```
int main() {
  int *arr = (int *)
    malloc(100 * sizeof(int));
  arr[99] = 42;
+ free(arr);
}
```

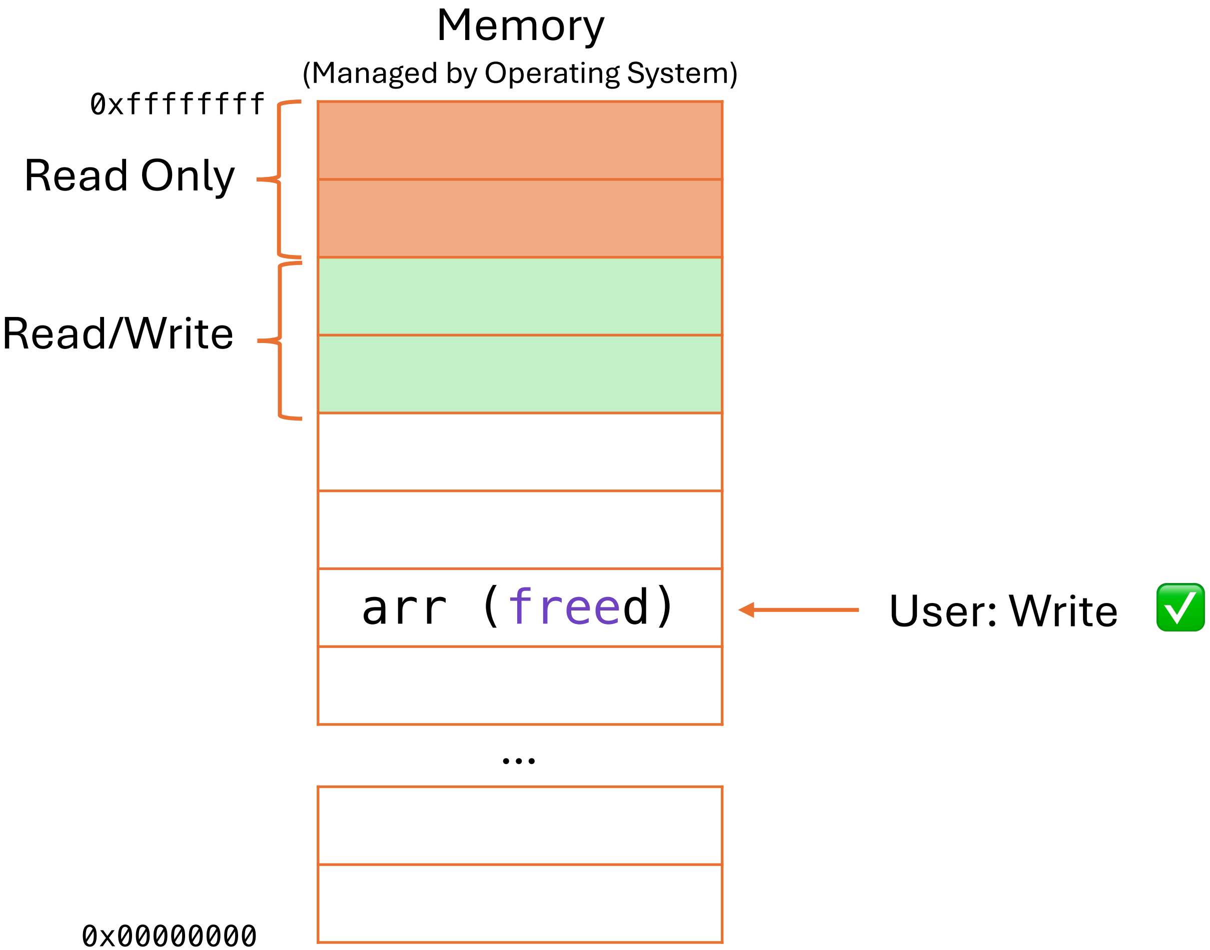
```
liby@mac ~/L/P/Demo> gcc demo.c
liby@mac ~/L/P/Demo> ./a.out
```



C Program that Silently Breaks Memory Safety

```
int main() {
  int *arr = (int *)
    malloc(100 * sizeof(int));
  arr[99] = 42;
  free(arr);
+ arr[3] = 27;
+ printf("%d\n", arr[3]);
}
```

```
liby@mac ~/L/P/Demo> gcc demo.c
liby@mac ~/L/P/Demo> ./a.out
27
```



Memory

(Managed by Operating System)

0xffffffff

Read Only

Read/Write

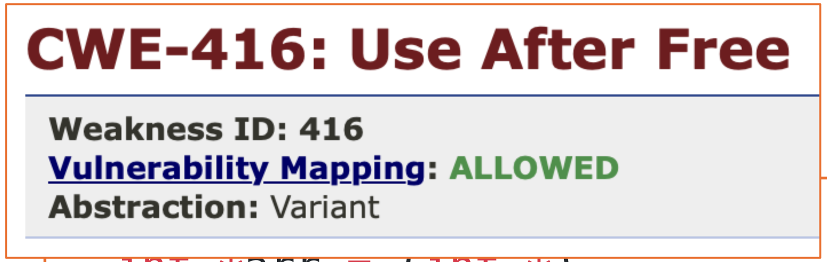
arr (freed)

User: Write ✅

...

0x00000000

C Program that Silently Breaks Memory Safety



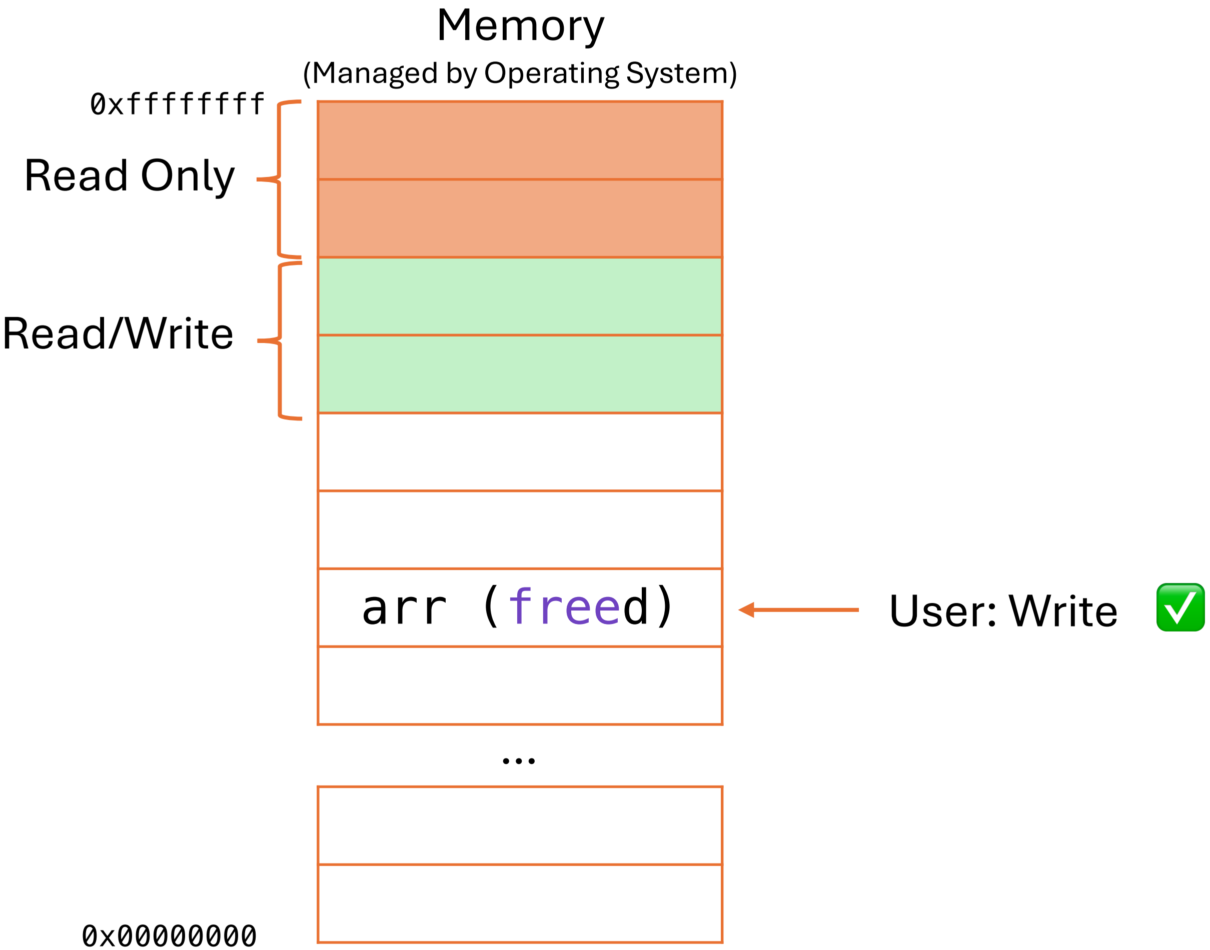
CWE-416: Use After Free

Weakness ID: 416
Vulnerability Mapping: ALLOWED
Abstraction: Variant

Use After Free

```
  int *arr = (int *)
                                     (int));

  free(arr);
+ arr[3] = 27;
+ printf("%d\n", arr[3]);
}
```

```
liby@mac ~/L/P/Demo> gcc demo.c
liby@mac ~/L/P/Demo> ./a.out
27
```



Takeaway

- C language does NOT have memory safety by-construct
- The responsibility of keeping memory safe is on the developers
  - If we ask LLMs to write C code, the responsibility is on the LLMs
- The unsafe memory operations may not be always noticeable
  - Silent undefined behavior is hard to catch
- Need extra tools to help catching silent issues
  - E.g., most memory related issues can be caught by valgrind

(Memory) Safe by Construct: Python

C/C++ Program

```c
int main() {
  int *arr = (int *)
    malloc(100 * sizeof(int));
  arr[182] = 42;
  free(arr);
  arr[3] = 27;
  printf("%d\n", arr[3]);
}
```

Python Program

```python
def main():
  arr = [()] * 100
  arr[182] = 42
```

```
 File ”demo.py", line 3, in main
   arr[182] = 42
   ~~~^^^^^^
IndexError: list assignment index
out of range
```

(Memory) Safe by Construct: Python

C/C++ Program

```c
int main() {
  int *arr = (int *)
    malloc(100 * sizeof(int));
  arr[182] = 42;
  free(arr);
  arr[3] = 27;
  printf("%d\n", arr[3]);
}
```

Python Program

```python
def main():
  arr = [()] * 100
+ if 182 > len(arr):
+   raise Exception(...)
  arr[182] = 42
```

```
 File ”demo.py", line 3, in main
   arr[182] = 42
   ~~~^^^^^
IndexError: list assignment index
out of range
```

(Memory) Safe by Construct: Python

C/C++ Program

```
int main() {
  int *arr = (int *)
    malloc(100 * sizeof(int));
  arr[99] = 42;
  free(arr);
  arr[3] = 27;
  printf("%d\n", arr[3]);
}
```

Python Program

```
def main():
  arr = [()] * 100
  arr[99] = 42
```

(Memory) Safe by Construct: Python

C/C++ Program

```
int main() {
  int *arr = (int *)
    malloc(100 * sizeof(int));
  arr[99] = 42;
  free(arr);
  arr[3] = 27;
  printf("%d\n", arr[3]);
}
```

Python Program

```
def main():
  arr = [()] * 100
  arr[99] = 42
```

In Python, this is done implicitly by memory management system

(Memory) Safe by Construct: Python

C/C++ Program

```c
int main() {
  int *arr1 = (int *)
    malloc(100 * sizeof(int));
  int *arr2 = arr1;
  free(arr1);
  arr2[3] = 27;
  printf("%d\n", arr2[3]);
}
```

Python Program

```python
def main():
  arr1 = [()] * 100
  arr2 = arr1
  del arr1
  arr2[3] = 27
  print(arr2[3])
```

(Memory) Safe by Construct: Python

C/C++ Program

```c
int main() {
  int *arr1 = (int *)
    malloc(100 * sizeof(int));
  int *arr2 = arr1;
  free(arr1);
  arr2[3] = 27;
  printf("%d\n", arr2[3]);
}
```

Python Program

```python
def main():
  arr1 = [()] * 100
  arr2 = arr1
  del arr1
  arr2[3] = 27
  print(arr2[3])
```

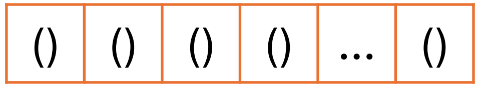
(Memory) Safe by Construct: Python

C/C++ Program

```
int main() {
  int *arr1 = (int *)
    malloc(100 * sizeof(int));
  int *arr2 = arr1;
  free(arr1);
  arr2[3] = 27;
  printf("%d\n", arr2[3]);
}
```

Python Program

```
def main():
  arr1 = [()] * 100
  arr2 = arr1
  del arr1
  arr2[3] = 27
  print(arr2[3])
```



Reference Count: 1

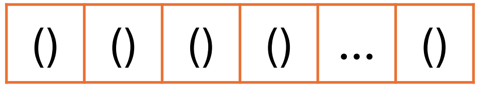
(Memory) Safe by Construct: Python

C/C++ Program

```c
int main() {
  int *arr1 = (int *)
    malloc(100 * sizeof(int));
  int *arr2 = arr1;
  free(arr1);
  arr2[3] = 27;
  printf("%d\n", arr2[3]);
}
```

Python Program

```python
def main():
  arr1 = [()] * 100
  arr2 = arr1
  del arr1
  arr2[3] = 27
  print(arr2[3])
```



Reference Count: 2 (+1)

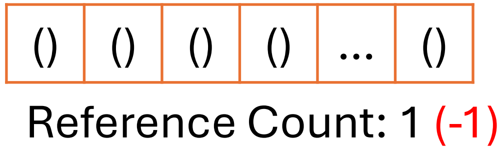
(Memory) Safe by Construct: Python

C/C++ Program

```c
int main() {
  int *arr1 = (int *)
    malloc(100 * sizeof(int));
  int *arr2 = arr1;
  free(arr1);
  arr2[3] = 27;
  printf("%d\n", arr2[3]);
}
```

Python Program

```python
def main():
  arr1 = [()] * 100
  arr2 = arr1
  del arr1
  arr2[3] = 27
  print(arr2[3])
```



()	()	()	()	…	()

Reference Count: 1 (-1)

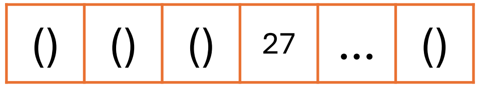
(Memory) Safe by Construct: Python

C/C++ Program

```c
int main() {
  int *arr1 = (int *)
    malloc(100 * sizeof(int));
  int *arr2 = arr1;
  free(arr1);
  arr2[3] = 27;
  printf("%d\n", arr2[3]);
}
```

Python Program

```python
def main():
  arr1 = [()] * 100
  arr2 = arr1
  del arr1
  arr2[3] = 27
  print(arr2[3])
```



Reference Count: 1

(Memory) Safe by Construct: Python

C/C++ Program

```c
int main() {
  int *arr1 = (int *)
    malloc(100 * sizeof(int));
  int *arr2 = arr1;
  free(arr1);
  arr2[3] = 27;
  printf("%d\n", arr2[3]);
}
```

Python Program

```python
def main():
  arr1 = [()] * 100
  arr2 = arr1
  del arr1
  arr2[3] = 27
  print(arr2[3])
```

()	()	()	27	…	()

Reference Count: 1

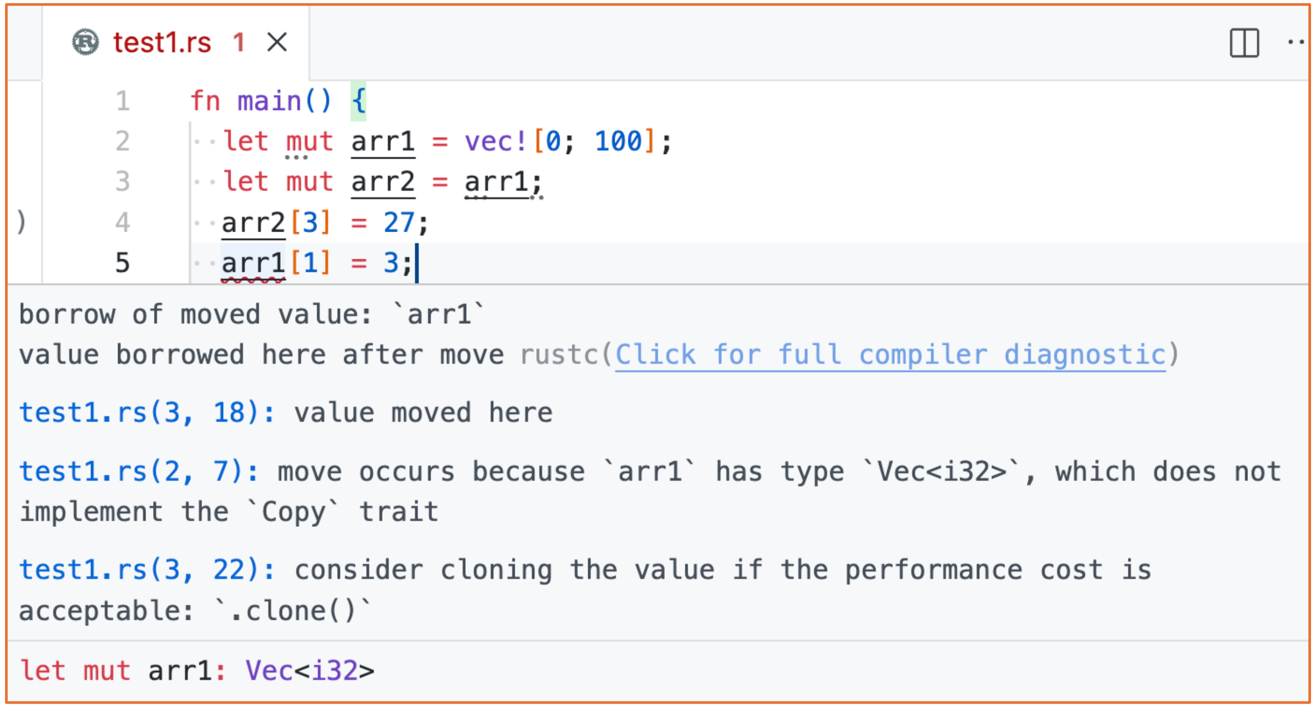
Garbage collection & “free”ing only happens when reference count (RC) of an object goes to 0

(Memory) Safe by Construct: Rust

C/C++ Program

```
int main() {
  int *arr1 = (int *)
    malloc(100 * sizeof(int));
  int *arr2 = arr1;
  free(arr1);
  arr2[3] = 27;
  printf("%d\n", arr2[3]);
}
```

Rust Program

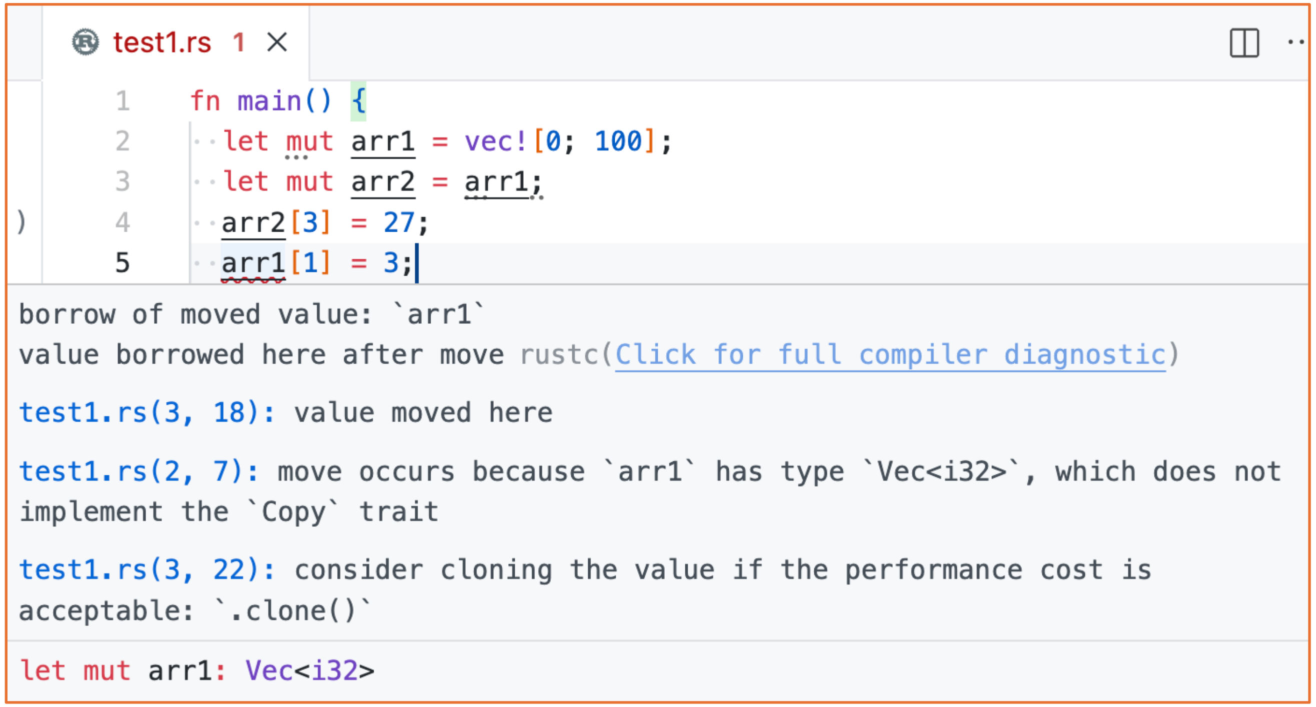


```
1  fn main() {
2    let mut arr1 = vec![0; 100];
3    let mut arr2 = arr1;
4    arr2[3] = 27;
5    arr1[1] = 3;
```

```
borrow of moved value: `arr1`
value borrowed here after move rustc(Click for full compiler diagnostic)

test1.rs(3, 18): value moved here

test1.rs(2, 7): move occurs because `arr1` has type `Vec<i32>`, which does not
implement the `Copy` trait

test1.rs(3, 22): consider cloning the value if the performance cost is
acceptable: `.clone()`

let mut arr1: Vec<i32>
```

(Memory) Safe by Construct: Rust

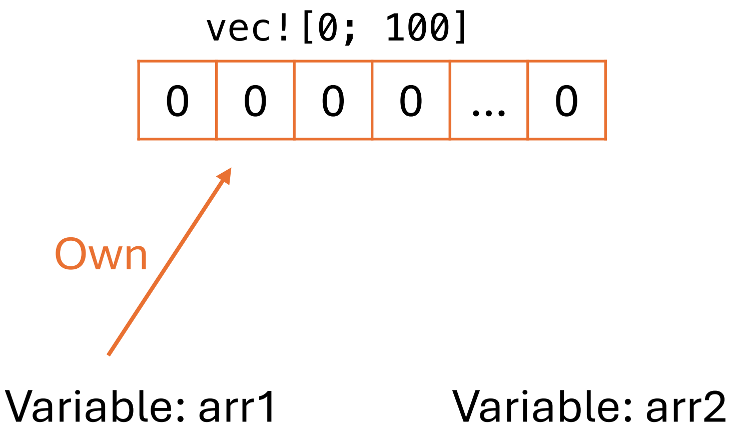
Rust Program



```
1  fn main() {
2    let mut arr1 = vec![0; 100];
3    let mut arr2 = arr1;
4    arr2[3] = 27;
5    arr1[1] = 3;
```

```
borrow of moved value: `arr1`
value borrowed here after move rustc(Click for full compiler diagnostic)

test1.rs(3, 18): value moved here

test1.rs(2, 7): move occurs because `arr1` has type `Vec<i32>`, which does not
implement the `Copy` trait

test1.rs(3, 22): consider cloning the value if the performance cost is
acceptable: `.clone()`

let mut arr1: Vec<i32>
```

Single Ownership


`vec![0; 100]`

0 0 0 0 … 0

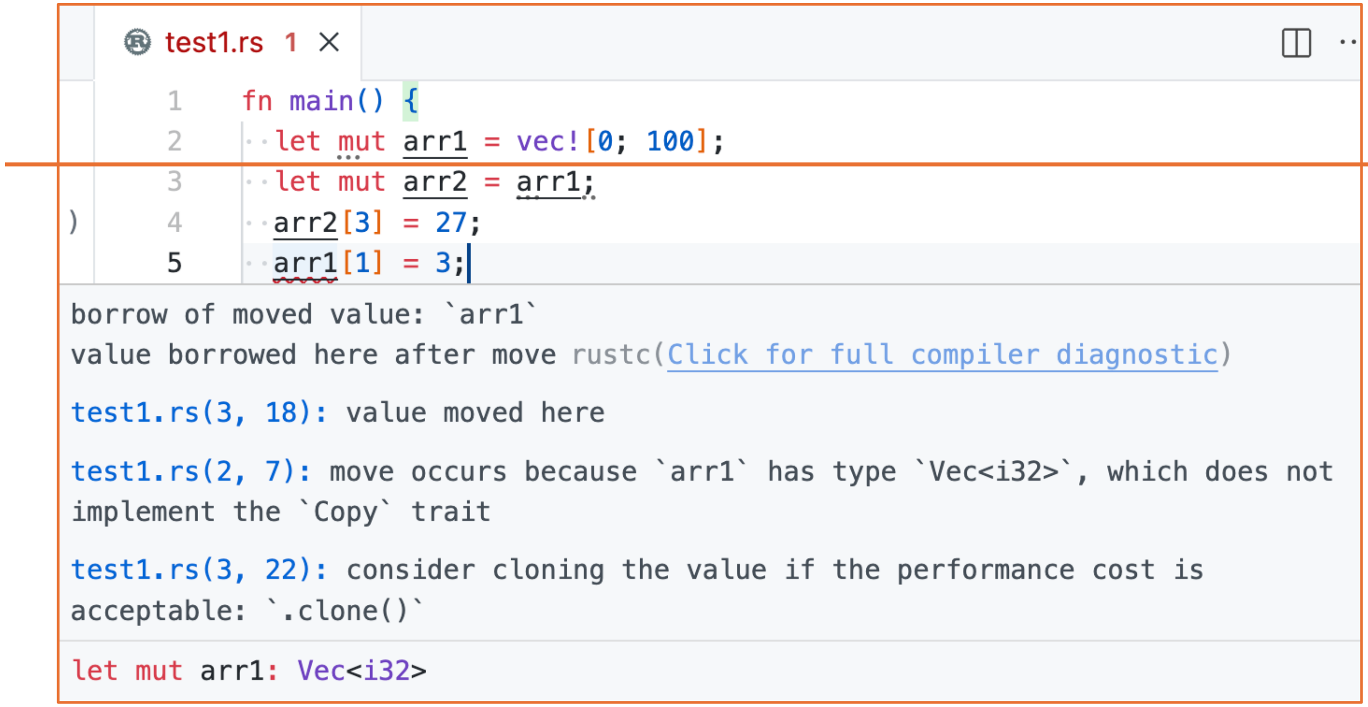
Own

Variable: arr1

Variable: arr2

(Memory) Safe by Construct: Rust

Rust Program



```
1  fn main() {
2      let mut arr1 = vec![0; 100];
3      let mut arr2 = arr1;
4      arr2[3] = 27;
5      arr1[1] = 3;
```

borrow of moved value: `arr1`
value borrowed here after move rustc(Click for full compiler diagnostic)

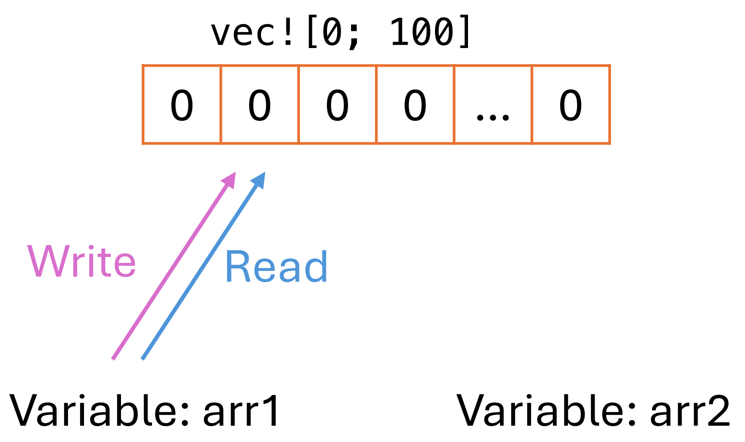
test1.rs(3, 18): value moved here

test1.rs(2, 7): move occurs because `arr1` has type `Vec<i32>`, which does not implement the `Copy` trait

test1.rs(3, 22): consider cloning the value if the performance cost is acceptable: `.clone()`

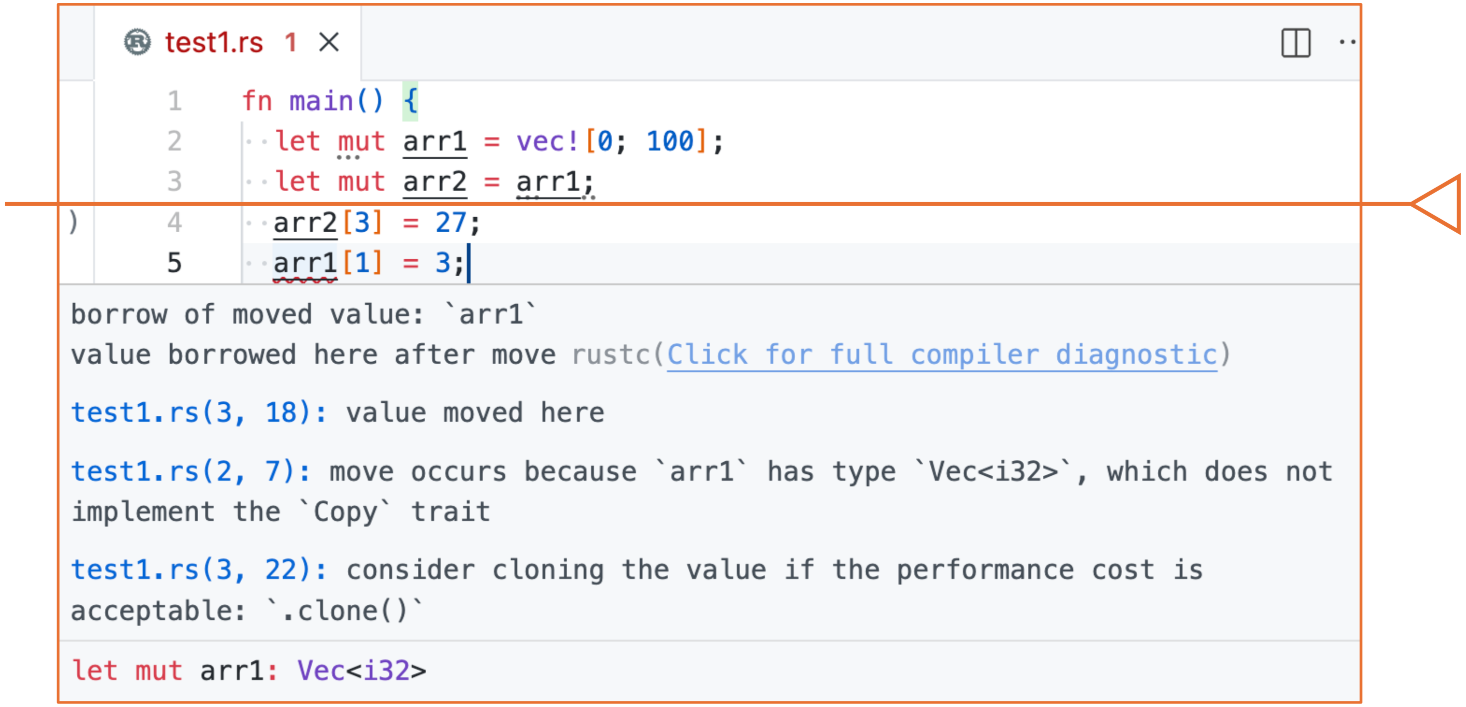
```
let mut arr1: Vec<i32>
```

Single Ownership



(Memory) Safe by Construct: Rust

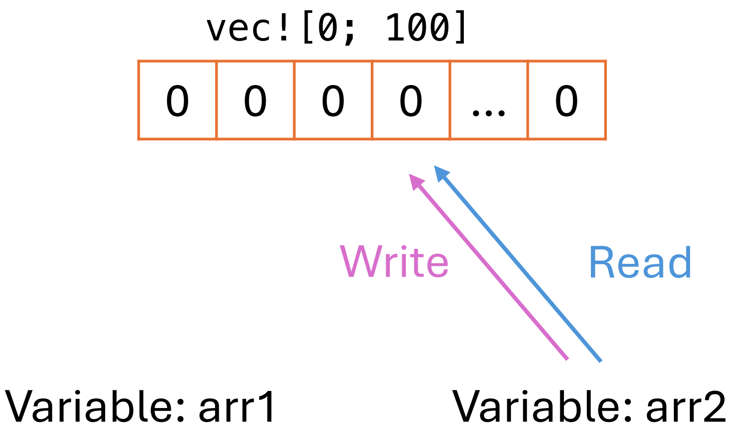
Rust Program



```
1  fn main() {
2      let mut arr1 = vec![0; 100];
3      let mut arr2 = arr1;
4      arr2[3] = 27;
5      arr1[1] = 3;
```

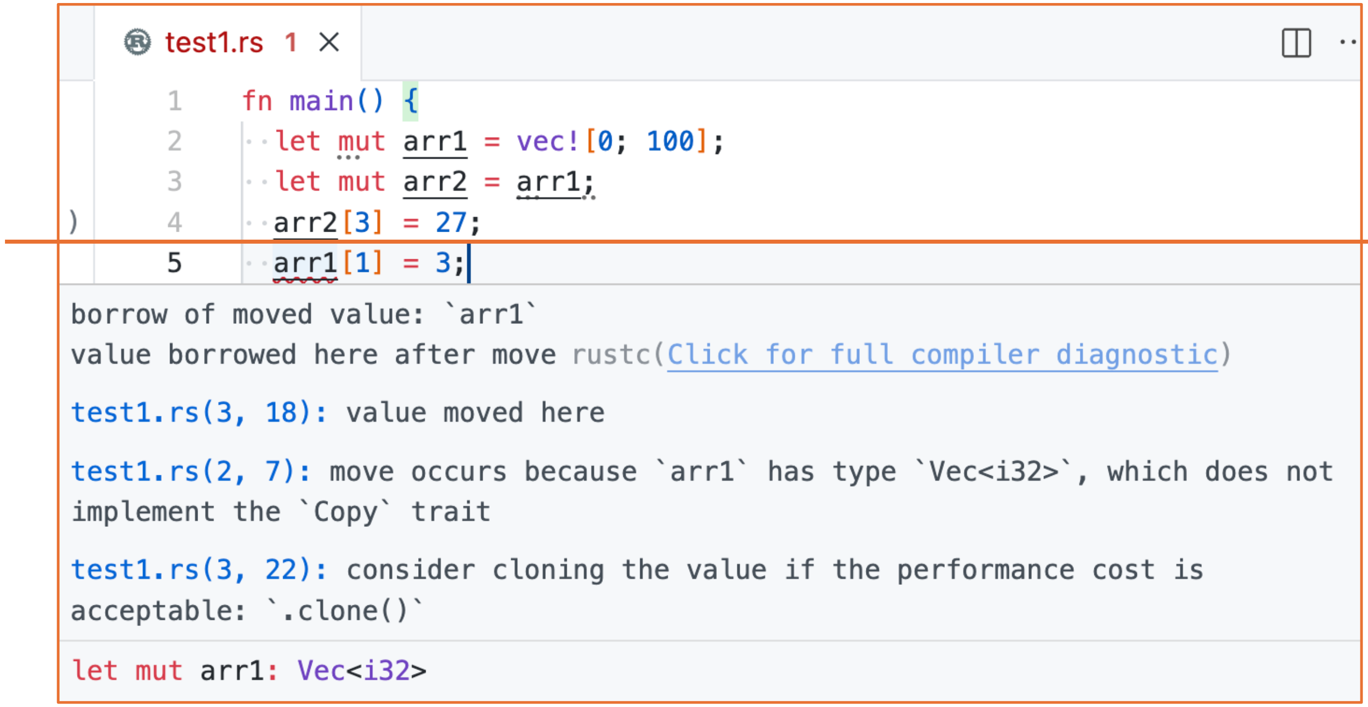
```
borrow of moved value: `arr1`
value borrowed here after move rustc(Click for full compiler diagnostic)

test1.rs(3, 18): value moved here

test1.rs(2, 7): move occurs because `arr1` has type `Vec<i32>`, which does not
implement the `Copy` trait

test1.rs(3, 22): consider cloning the value if the performance cost is
acceptable: `.clone()`

let mut arr1: Vec<i32>
```

Single Ownership



(Memory) Safe by Construct: Rust

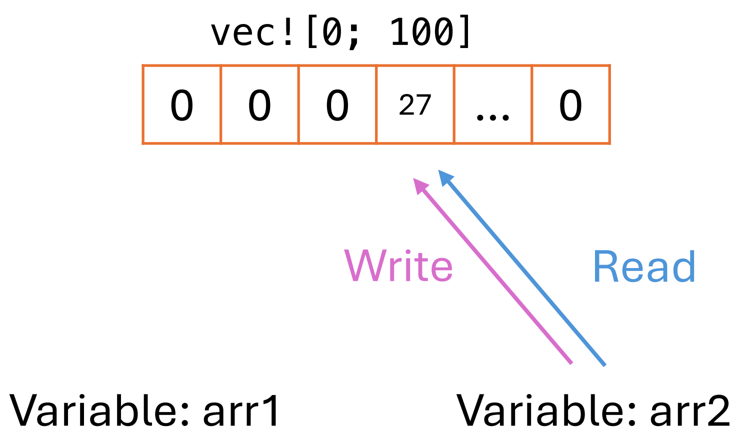
Rust Program



```
1  fn main() {
2    let mut arr1 = vec![0; 100];
3    let mut arr2 = arr1;
4    arr2[3] = 27;
5    arr1[1] = 3;
```

```
borrow of moved value: `arr1`
value borrowed here after move rustc(Click for full compiler diagnostic)

test1.rs(3, 18): value moved here

test1.rs(2, 7): move occurs because `arr1` has type `Vec<i32>`, which does not
implement the `Copy` trait

test1.rs(3, 22): consider cloning the value if the performance cost is
acceptable: `.clone()`

let mut arr1: Vec<i32>
```

Single Ownership

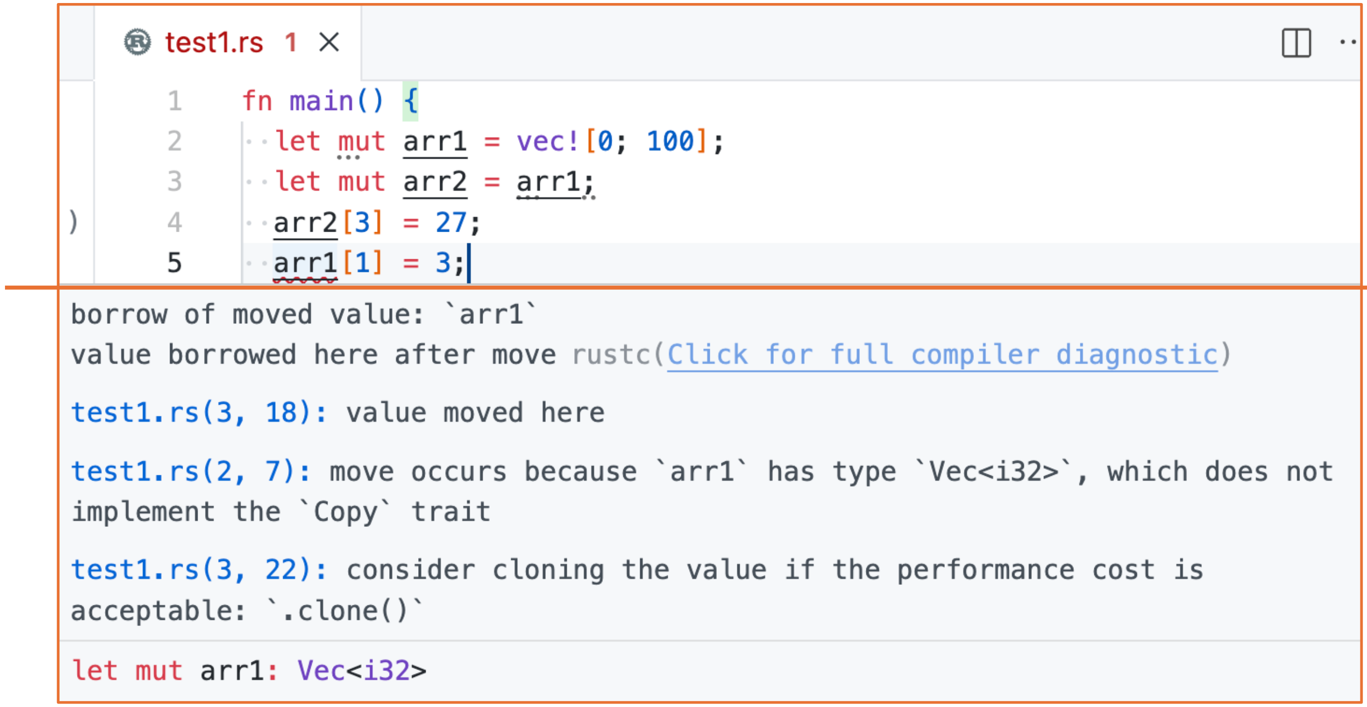


vec![0; 100]

0	0	0	27	...	0

(Memory) Safe by Construct: Rust

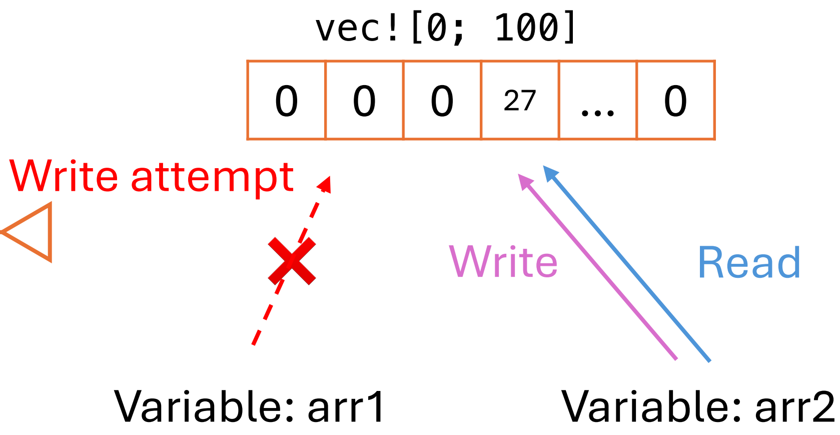
Rust Program



```
1  fn main() {
2      let mut arr1 = vec![0; 100];
3      let mut arr2 = arr1;
4      arr2[3] = 27;
5      arr1[1] = 3;
```

```
borrow of moved value: `arr1`
value borrowed here after move rustc(Click for full compiler diagnostic)

test1.rs(3, 18): value moved here

test1.rs(2, 7): move occurs because `arr1` has type `Vec<i32>`, which does not
implement the `Copy` trait

test1.rs(3, 22): consider cloning the value if the performance cost is
acceptable: `.clone()`

let mut arr1: Vec<i32>
```

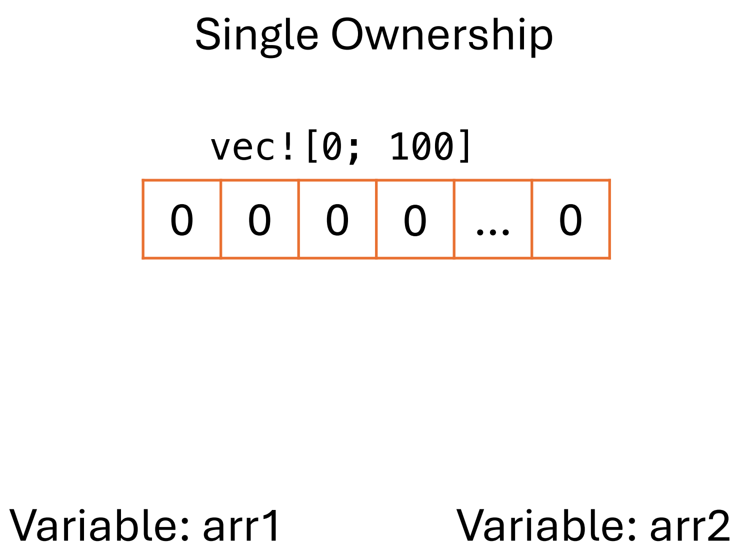
Single Ownership



(Memory) Safe by Construct: Rust

Rust Program

```
fn main() {
  let mut arr1 = vec![0; 100];
  let arr2 = &mut arr1;
  arr2[3] = 27;
  arr1[2] = 30;
}
```



Single Ownership

`vec![0; 100]`

0	0	0	0	...	0

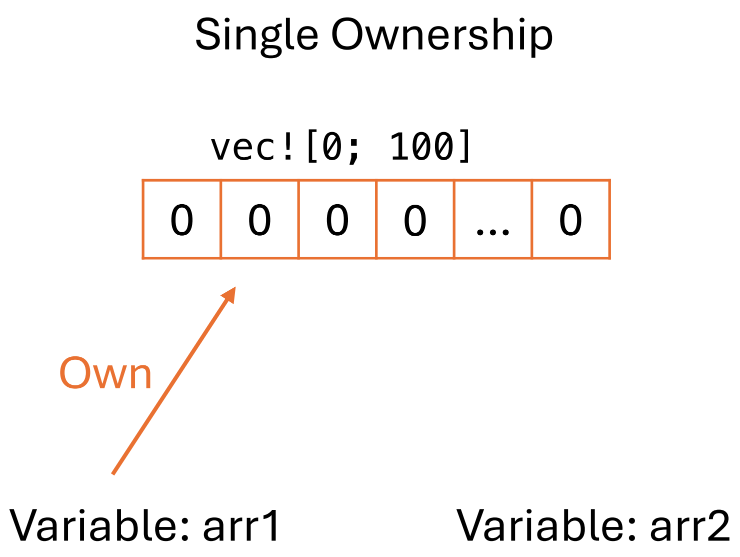
Variable: arr1

Variable: arr2

(Memory) Safe by Construct: Rust

Rust Program

```
fn main() {
  let mut arr1 = vec![0; 100];
  let arr2 = &mut arr1;
  arr2[3] = 27;
  arr1[2] = 30;
}
```



Single Ownership

vec![0; 100]

0	0	0	0	...	0

Own

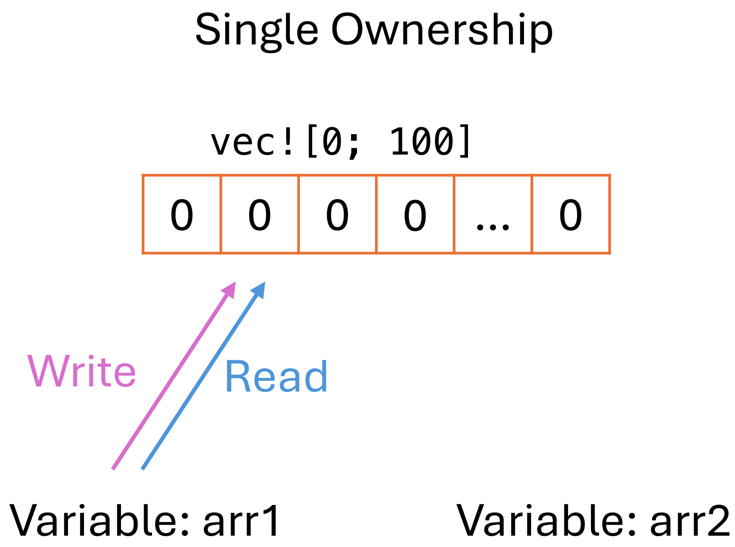
Variable: arr1

Variable: arr2

(Memory) Safe by Construct: Rust

Rust Program

```
fn main() {
  let mut arr1 = vec![0; 100];
  let arr2 = &mut arr1;
  arr2[3] = 27;
  arr1[2] = 30;
}
```



Single Ownership

vec![0; 100]

0	0	0	0	...	0

Variable: arr1

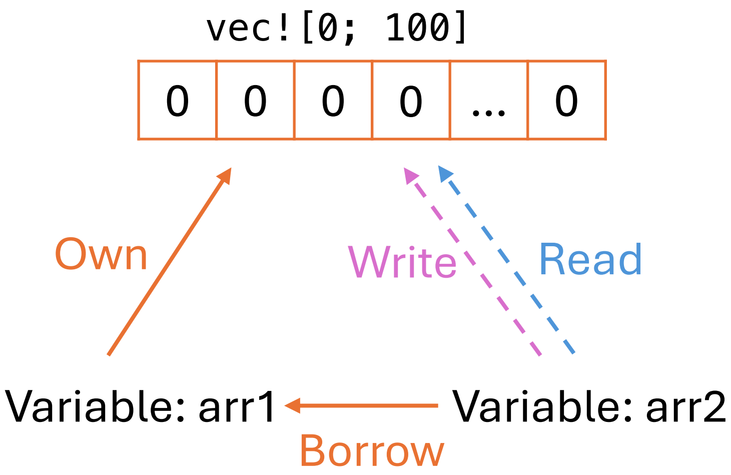
Variable: arr2

(Memory) Safe by Construct: Rust

Rust Program

```
fn main() {
  let mut arr1 = vec![0; 100];
  let arr2 = &mut arr1;
  arr2[3] = 27;
  arr1[2] = 30;
}
```

Single Ownership

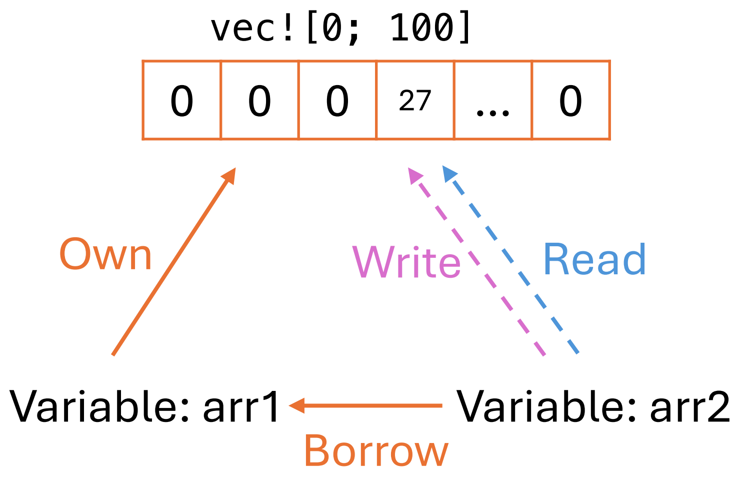


(Memory) Safe by Construct: Rust

Rust Program

```
fn main() {
   let mut arr1 = vec![0; 100];
   let arr2 = &mut arr1;
   arr2[3] = 27;
   arr1[2] = 30;
}
```

Single Ownership

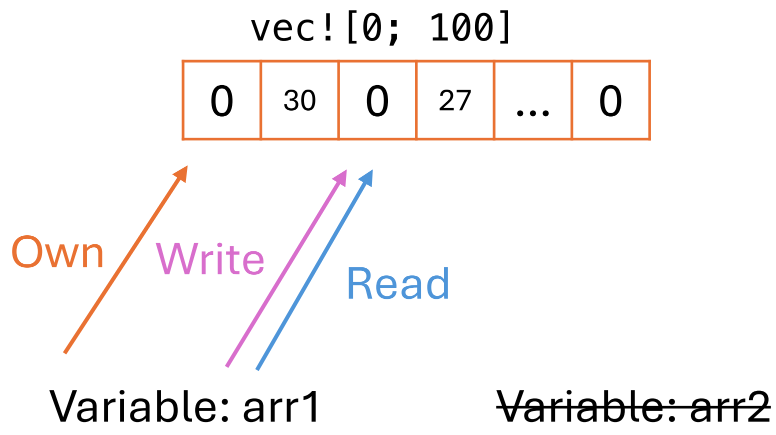


(Memory) Safe by Construct: Rust

Rust Program

```rust
fn main() {
  let mut arr1 = vec![0; 100];
  let arr2 = &mut arr1;
  arr2[3] = 27;
  arr1[2] = 30;
}
```

Single Ownership



Key Takeaway: Who is responsible for safety?

C/C++ Program

```c
int main() {
  int *arr1 = (int *)
    malloc(100 * sizeof(int));
  int *arr2 = arr1;
  free(arr1);
  arr2[3] = 27;
  printf("%d\n", arr2[3]);
}
```

Developer / LLM

Python Program

```python
def main():
  arr1 = [()] * 100
  arr2 = arr1
  del arr1
  arr2[3] = 27
  print(arr2[3])
```

Python Runtime

Memory Management

Reference Counting

Garbage Collection

Rust Program

```rust
fn main() {
  let mut arr1 = vec![0; 100];
  let arr2 = &mut arr1;
  arr2[3] = 27;
  arr1[2] = 30;
}
```

Rust Compiler

Linear type system

Ownership & borrow checker

Life-time resolver

Key Takeaway: Who can be trusted?

C/C++ Program

```c
int main() {
  int *arr1 = (int *)
    malloc(100 * sizeof(int));
  int *arr2 = arr1;
  free(arr1);
  arr2[3] = 27;
  printf("%d\n", arr2[3]);
}
```

Developer / LLM

NO

Python Program

```python
def main():
  arr1 = [()] * 100
  arr2 = arr1
  del arr1
  arr2[3] = 27
  print(arr2[3])
```

Python Runtime

Memory Management

Reference Counting

Garbage Collection

Maybe yes

Rust Program

```rust
fn main() {
  let mut arr1 = vec![0; 100];
  let arr2 = &mut arr1;
  arr2[3] = 27;
  arr1[2] = 30;
}
```

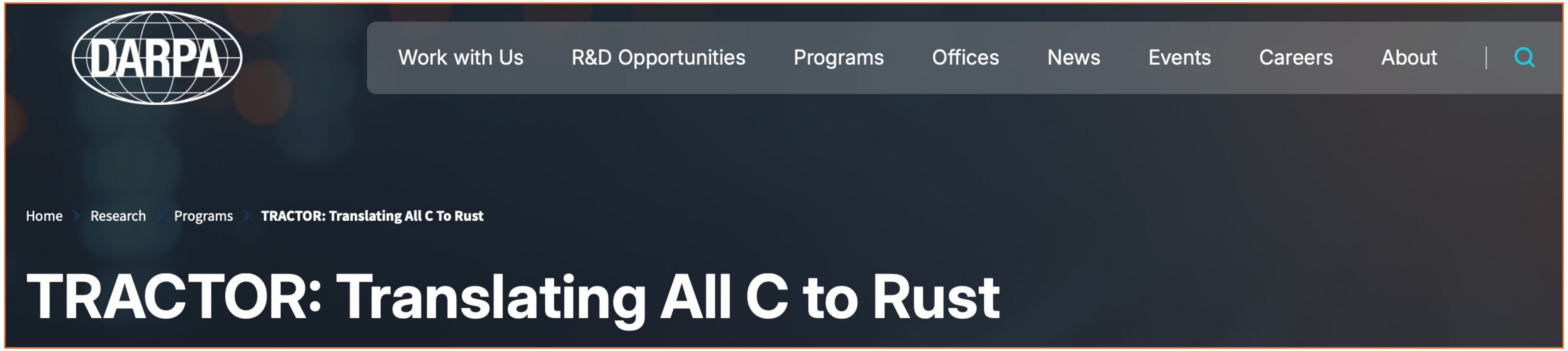
Rust Compiler

Linear type system

Ownership & borrow checker

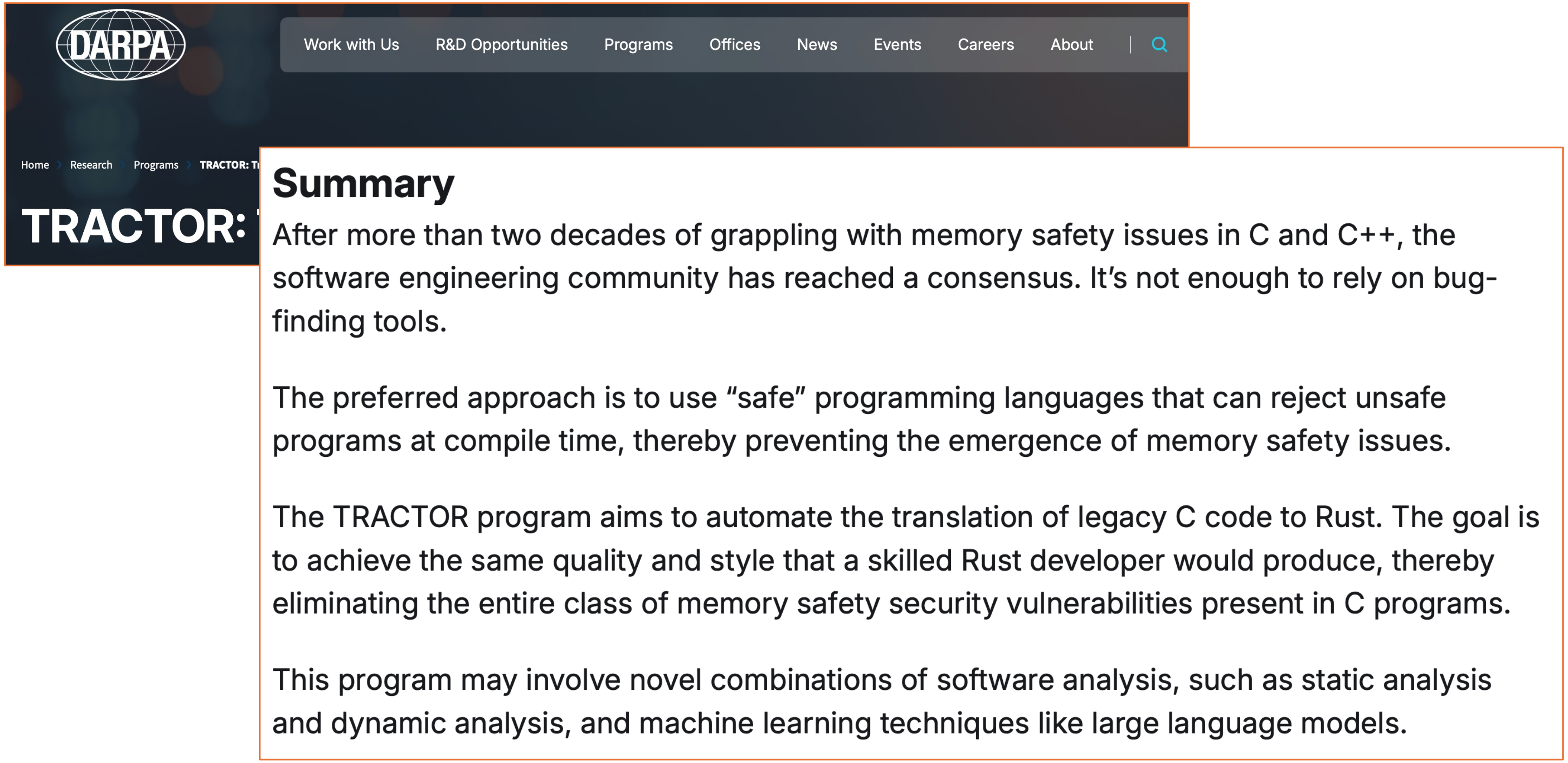
Life-time resolver

Maybe yes



Home › Research › Programs › TRACTOR: Translating All C To Rust

TRACTOR: Translating All C to Rust



TRACTOR:

Summary

After more than two decades of grappling with memory safety issues in C and C++, the software engineering community has reached a consensus. It's not enough to rely on bug-finding tools.

The preferred approach is to use "safe" programming languages that can reject unsafe programs at compile time, thereby preventing the emergence of memory safety issues.

The TRACTOR program aims to automate the translation of legacy C code to Rust. The goal is to achieve the same quality and style that a skilled Rust developer would produce, thereby eliminating the entire class of memory safety security vulnerabilities present in C programs.

This program may involve novel combinations of software analysis, such as static analysis and dynamic analysis, and machine learning techniques like large language models.

DARPA

Work with Us | R&D Opportunities | Programs | Offices | News | Events | Careers | About

Home › Resea

TRA

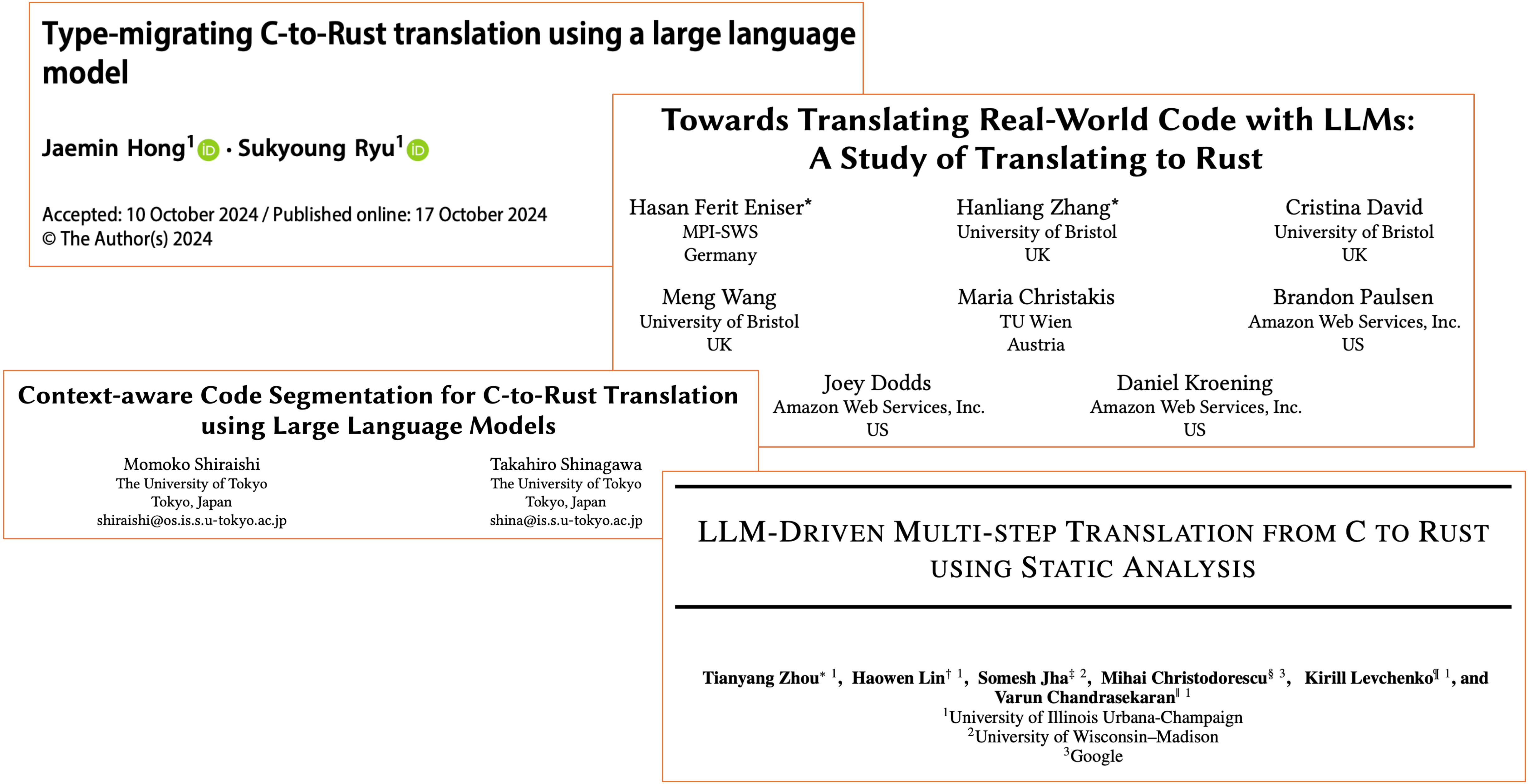
Summary

After more than two decades of grappling with memory safety issues in C and C++, the software enginee
finding tools.

The preferred ap
programs at com

The TRACTOR pr
to achieve the sa
eliminating the e

This program ma
and dynamic ana



Type-migrating C-to-Rust translation using a large language model

Jaemin Hong[1] · Sukyoung Ryu[1]

Accepted: 10 October 2024 / Published online: 17 October 2024
© The Author(s) 2024

Towards Translating Real-World Code with LLMs: A Study of Translating to Rust

Hasan Ferit Eniser*
MPI-SWS
Germany

Hanliang Zhang*
University of Bristol
UK

Cristina David
University of Bristol
UK

Meng Wang
University of Bristol
UK

Maria Christakis
TU Wien
Austria

Brandon Paulsen
Amazon Web Services, Inc.
US

Joey Dodds
Amazon Web Services, Inc.
US

Daniel Kroening
Amazon Web Services, Inc.
US

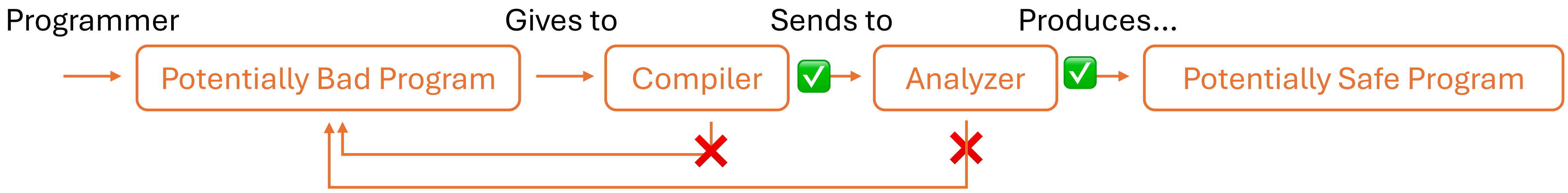
Context-aware Code Segmentation for C-to-Rust Translation using Large Language Models

Momoko Shiraishi
The University of Tokyo
Tokyo, Japan
shiraishi@os.is.s.u-tokyo.ac.jp

Takahiro Shinagawa
The University of Tokyo
Tokyo, Japan
shina@is.s.u-tokyo.ac.jp

LLM-Driven Multi-step Translation from C to Rust using Static Analysis

Tianyang Zhou* [1], **Haowen Lin**† [1], **Somesh Jha**‡ [2], **Mihai Christodorescu**§ [3], **Kirill Levchenko**¶ [1], **and Varun Chandrasekaran**‖ [1]
[1]University of Illinois Urbana-Champaign
[2]University of Wisconsin–Madison
[3]Google



Programmer

Potentially Bad Program

Gives to

Compiler

✓

Sends to

Analyzer

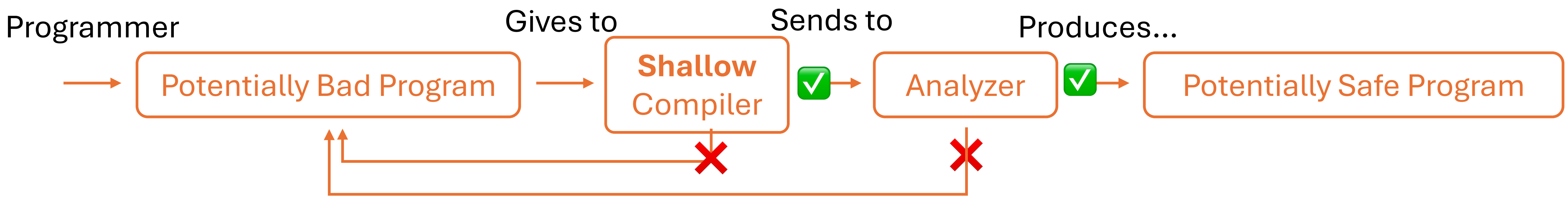
✓

Produces...

Potentially Safe Program

✗

✗



Programmer

Potentially Bad Program

Gives to

Shallow Compiler

✅

Sends to

Analyzer

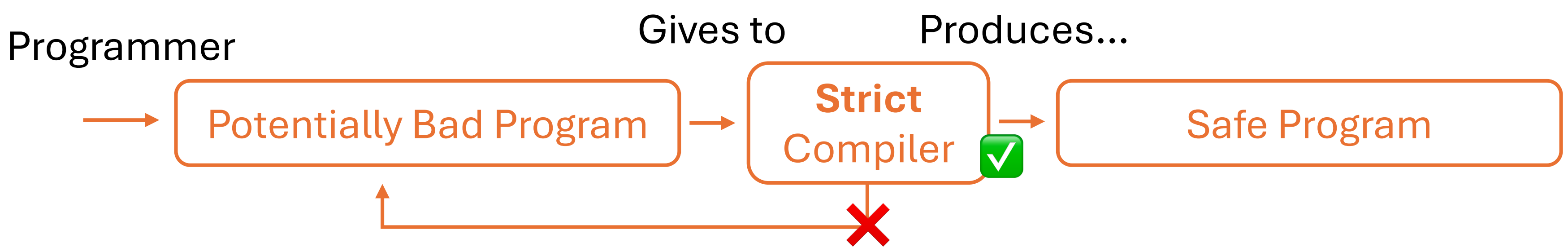
✅

Produces...

Potentially Safe Program

❌

❌



Programmer

Potentially Bad Program

Gives to

Strict Compiler

✅

Produces...

Safe Program

❌

Desirable Properties

Memory Safety

Side-channel Resistance

Termination

Injection-safety

Functional Assurance

Concurrency Safety

Type Safety

Capability Safety

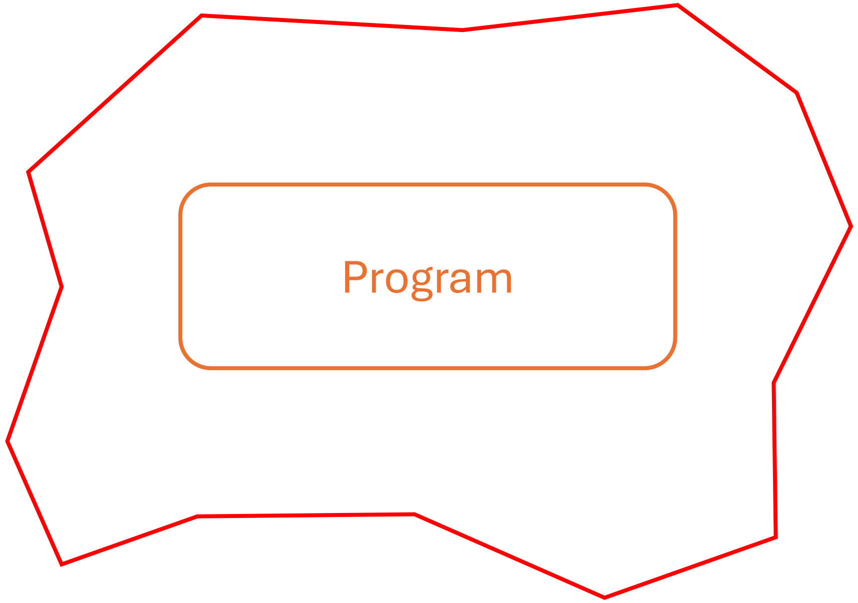
Smart-contract Safety

Control-flow Integrity

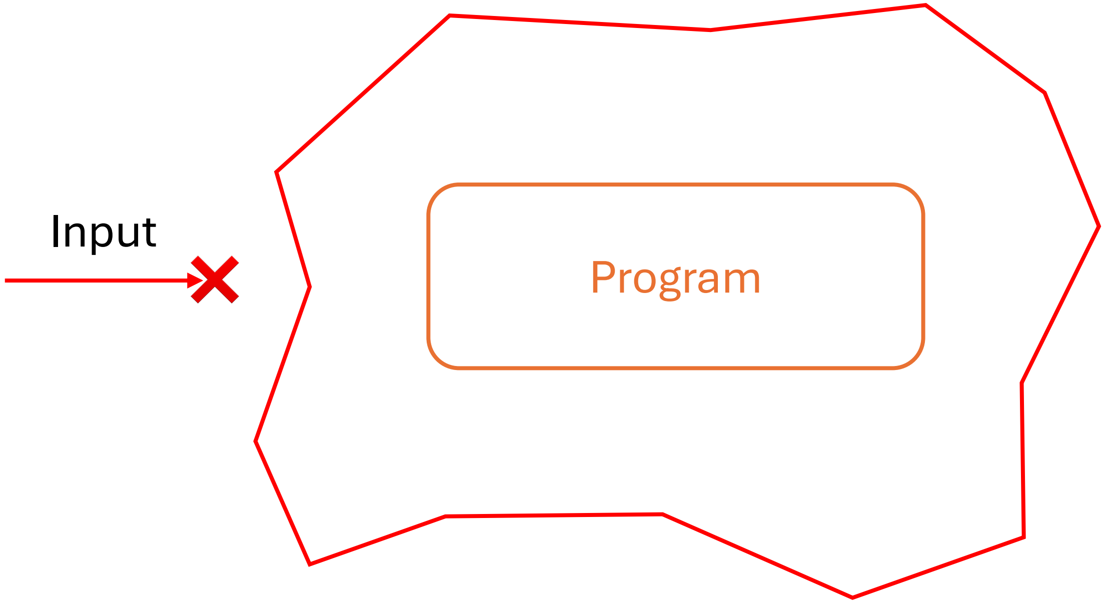
Resource Safety

Data Integrity

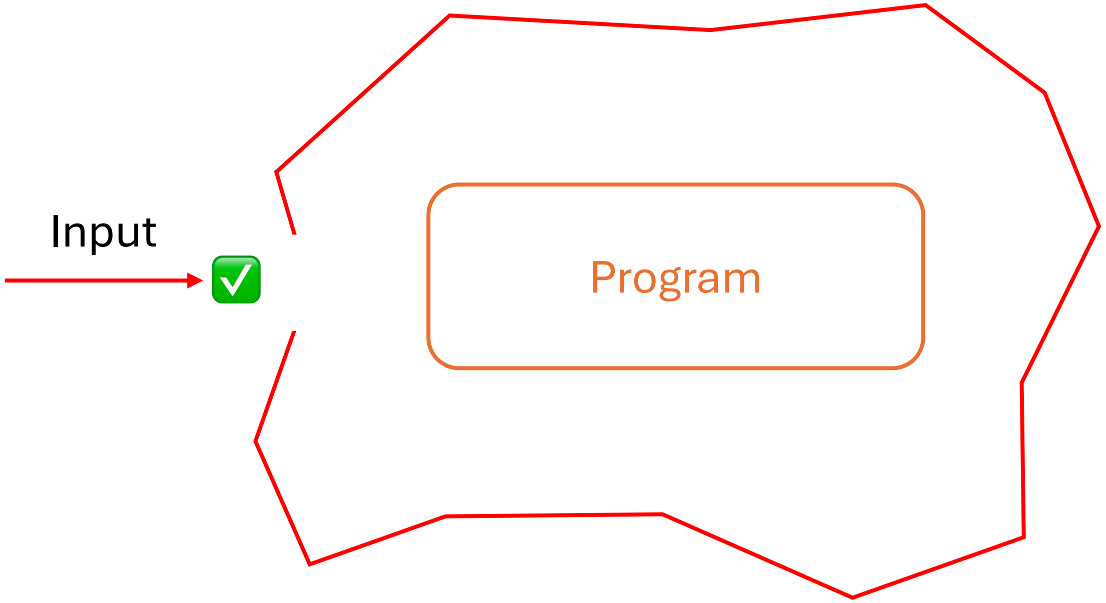
Safe (?) Program



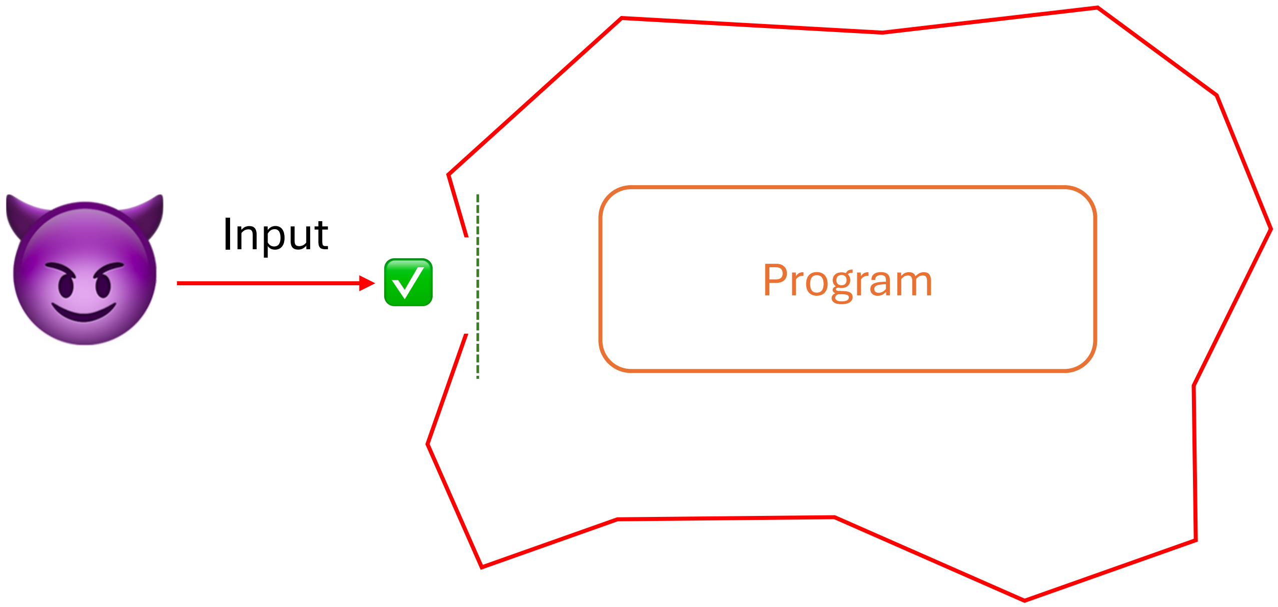
Safe Program is not interesting



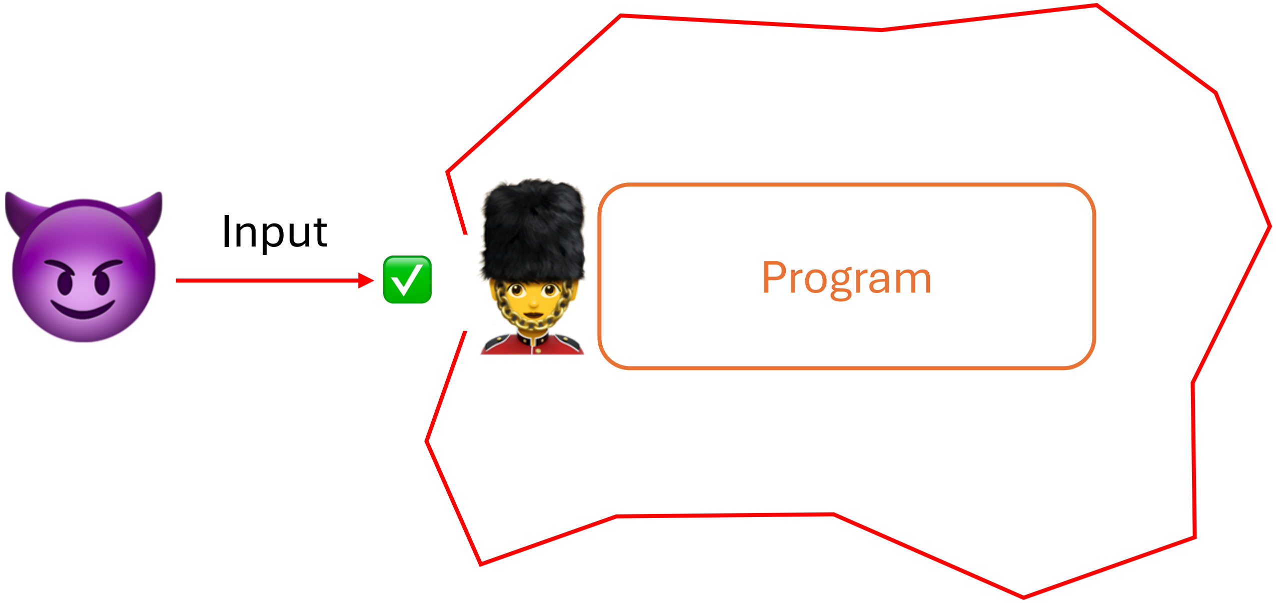
Programs Take Input…



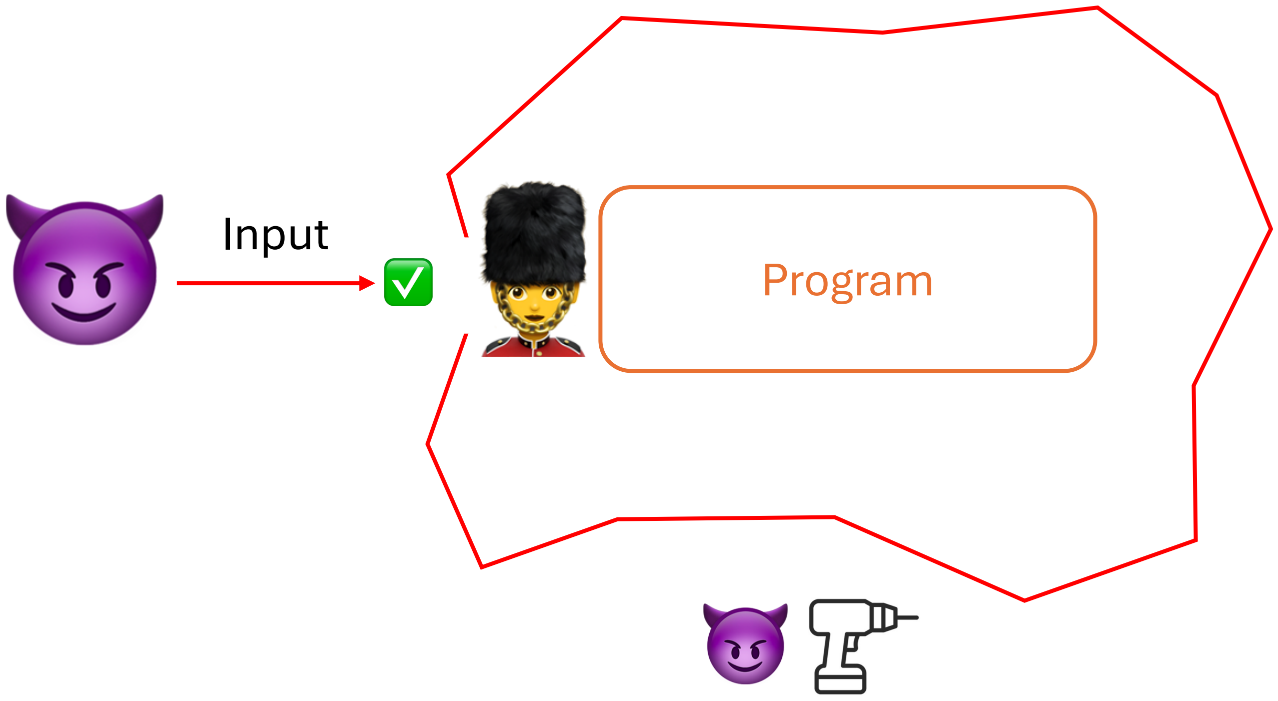
Attack Surface is Exposed...



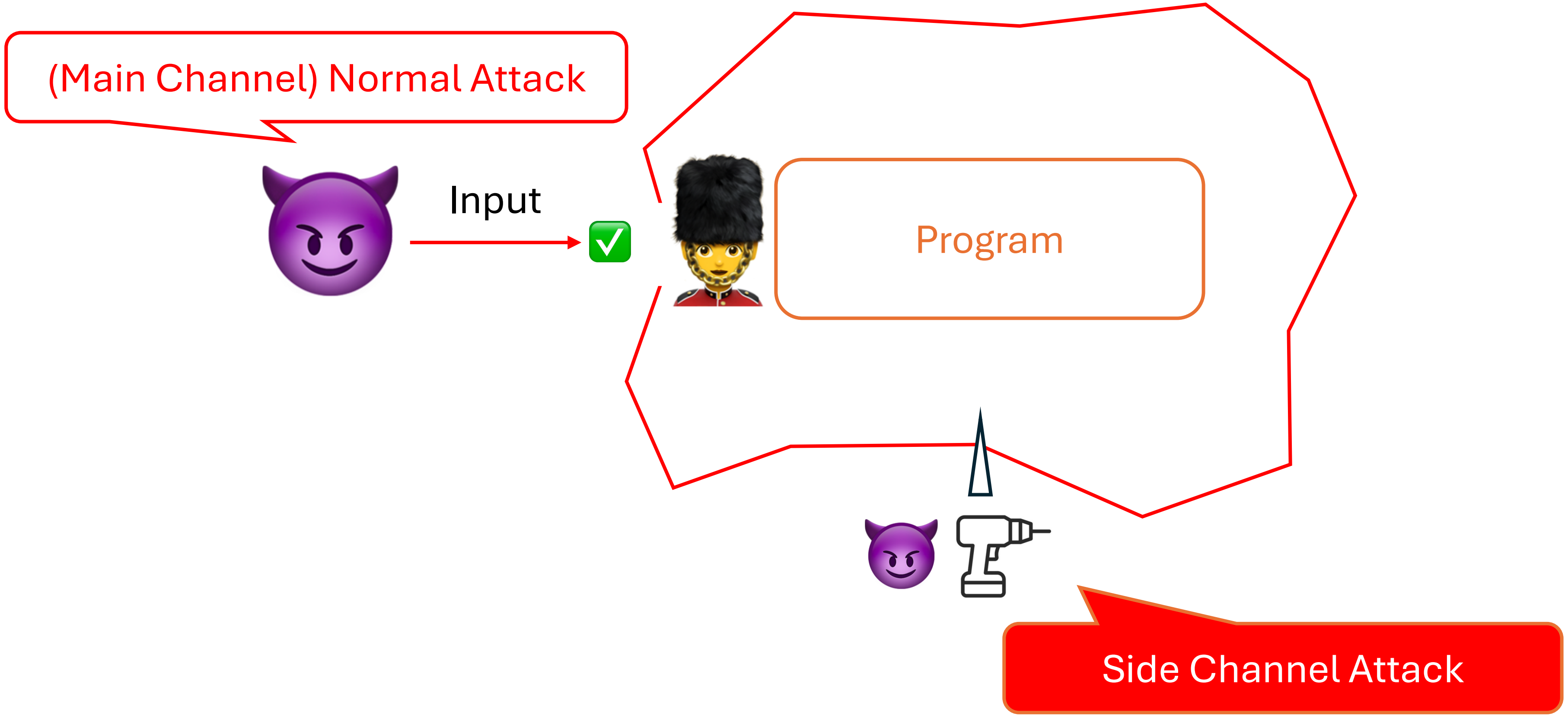
Defense is Setup...

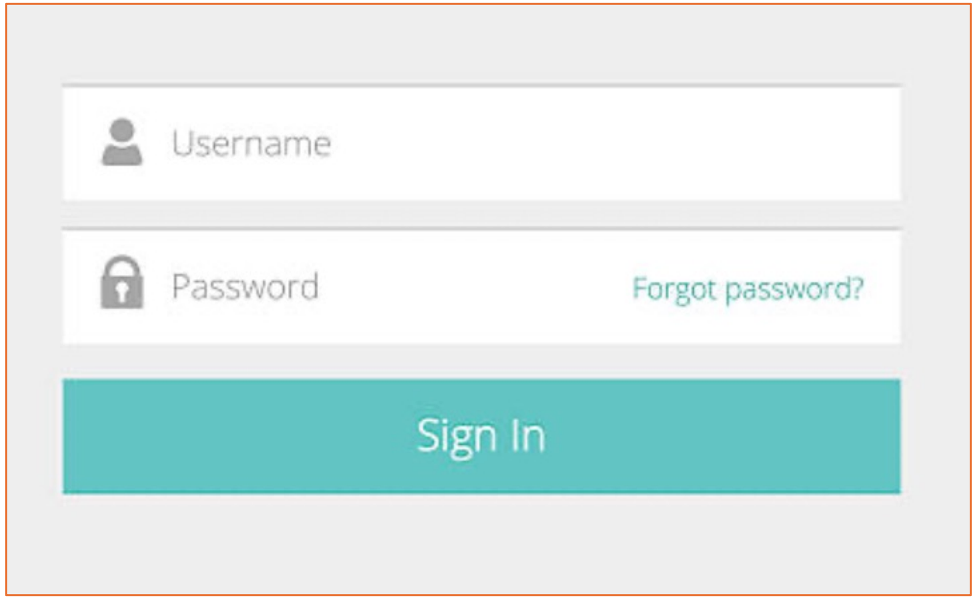


Defense is Setup, But



Defense is Setup, But...




Username
Password
Forgot password?
Sign In

```python
def check_password(expected_password, provided_password):
  if len(expected_password) != len(provided_password):
    return False
  for (expected_char, provided_char) in zip(expected_password, provided_password):
    if expected_char != provided_char:
      return False
  return True
```

```python
def check_password(expected_password, provided_password):
  if len(expected_password) != len(provided_password):
    return False
  for (expected_char, provided_char) in zip(expected_password, provided_password):
    if expected_char != provided_char:
      return False
  return True
```

Expected Password: 12345678

Attempt 1: 13579
✗

Attempt 2: 02468
✗

Attempt 3: 12345
✗

```python
def check_password(expected_password, provided_password):
  if len(expected_password) != len(provided_password):
    return False
  for (expected_char, provided_char) in zip(expected_password, provided_password):
    if expected_char != provided_char:
      return False
  return True
```

Expected Password: 12345678

Attempt 1: 13579

❌

Finishes in 4 CPU cycles

Estimation: 1 char match

Attempt 2: 02468

❌

Finishes in 2 CPU cycles

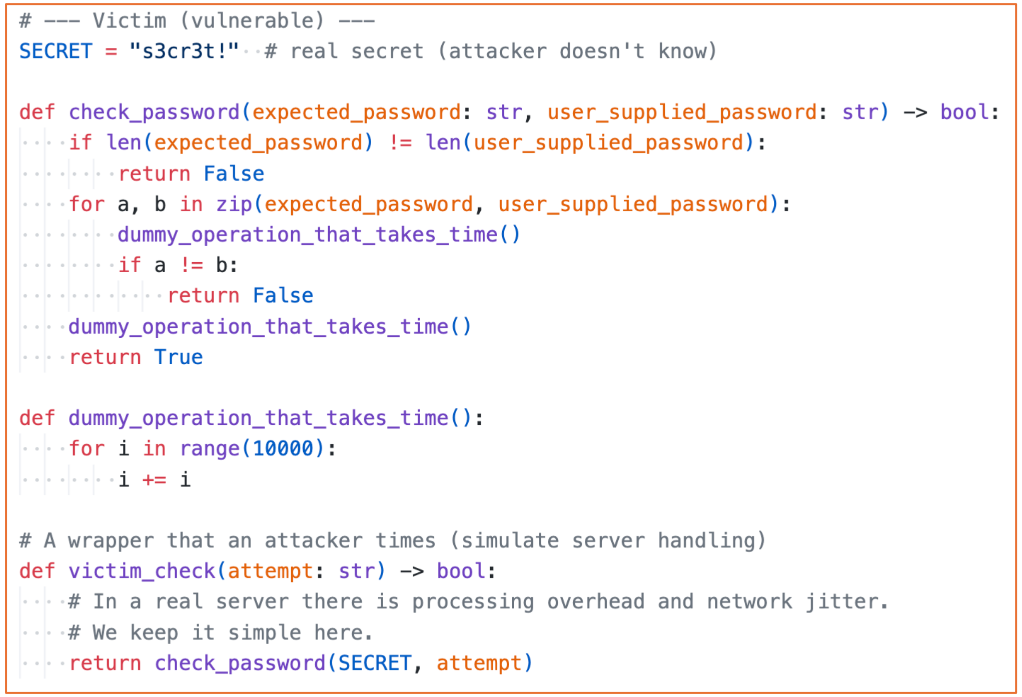
Estimation: 0 char match

Attempt 3: 12345

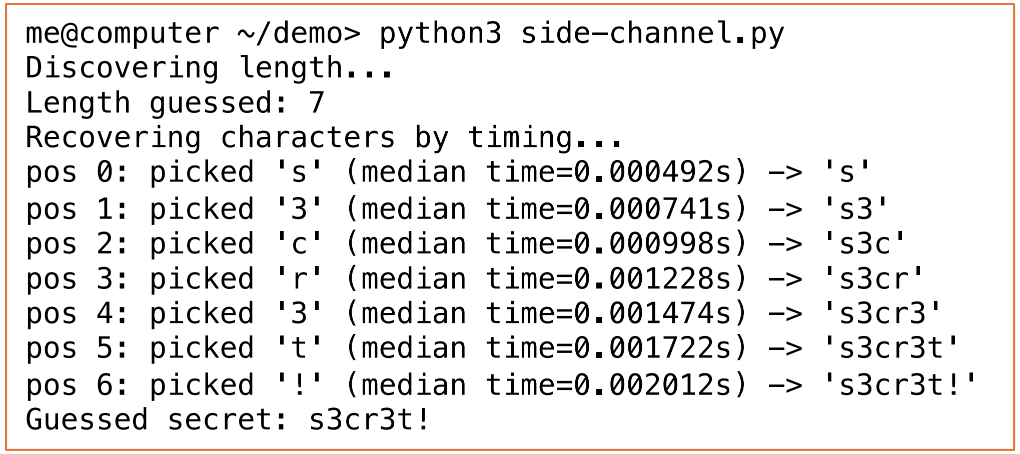
❌

Finishes in 12 CPU cycles

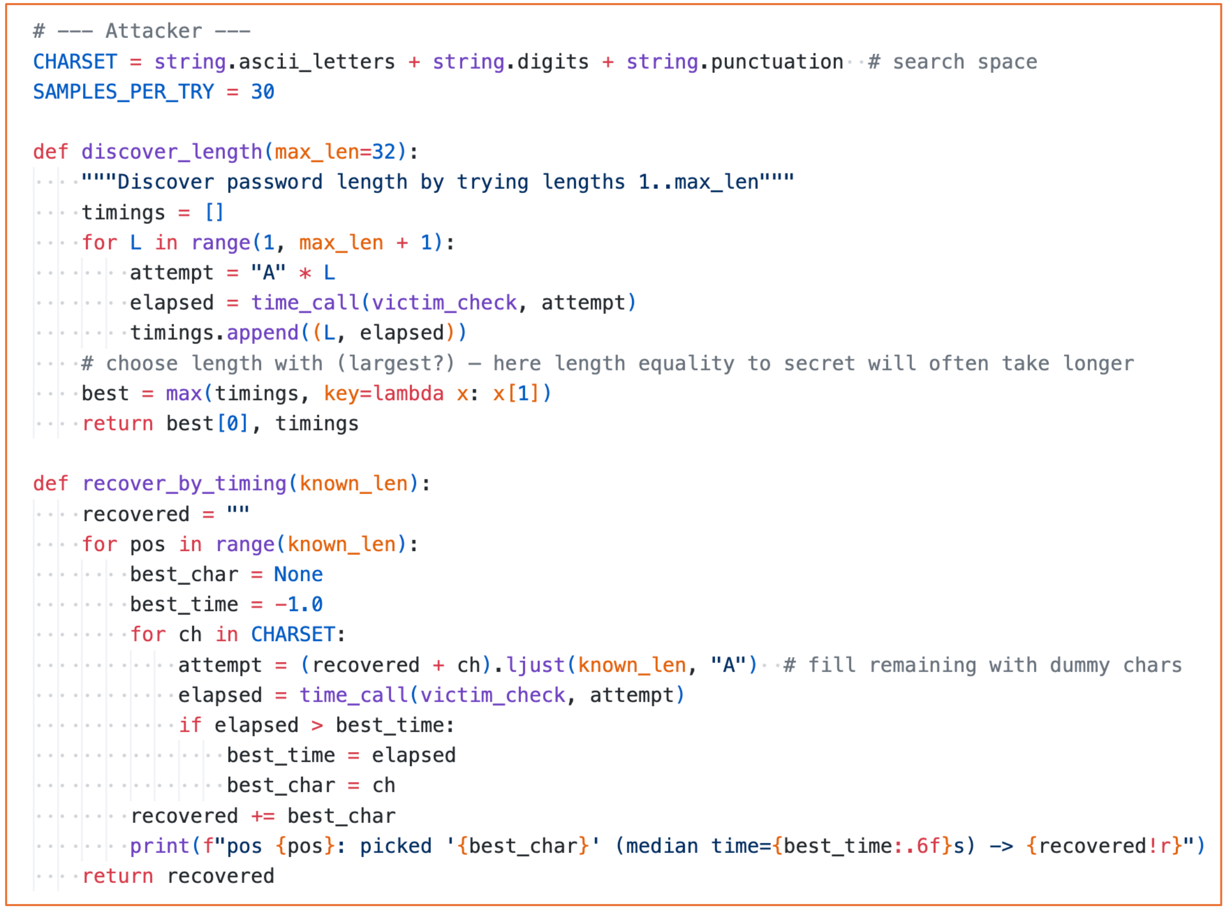
Estimation: 5 char match



```python
# --- Victim (vulnerable) ---
SECRET = "s3cr3t!"  # real secret (attacker doesn't know)

def check_password(expected_password: str, user_supplied_password: str) -> bool:
    if len(expected_password) != len(user_supplied_password):
        return False
    for a, b in zip(expected_password, user_supplied_password):
        dummy_operation_that_takes_time()
        if a != b:
            return False
    dummy_operation_that_takes_time()
    return True

def dummy_operation_that_takes_time():
    for i in range(10000):
        i += i

# A wrapper that an attacker times (simulate server handling)
def victim_check(attempt: str) -> bool:
    # In a real server there is processing overhead and network jitter.
    # We keep it simple here.
    return check_password(SECRET, attempt)
```

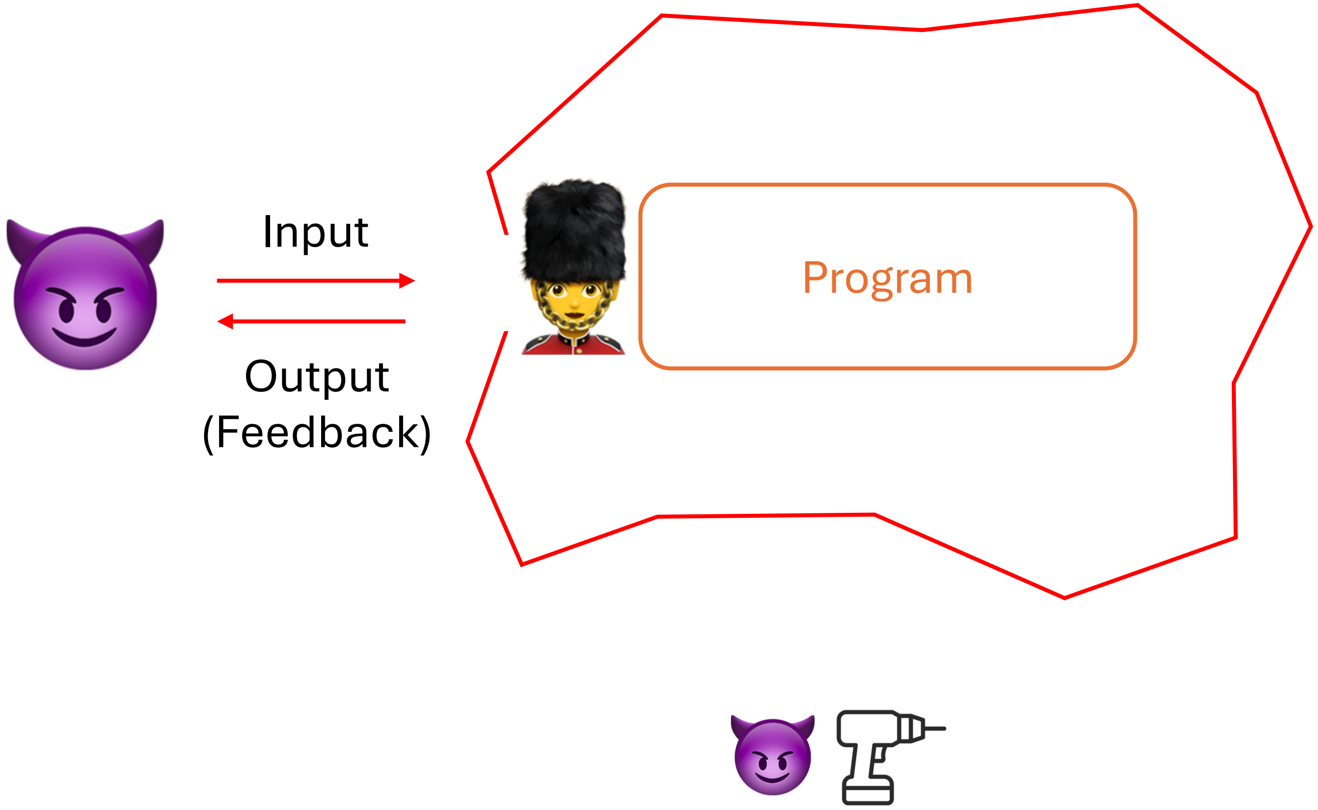


```
me@computer ~/demo> python3 side-channel.py
Discovering length...
Length guessed: 7
Recovering characters by timing...
pos 0: picked 's' (median time=0.000492s) -> 's'
pos 1: picked '3' (median time=0.000741s) -> 's3'
pos 2: picked 'c' (median time=0.000998s) -> 's3c'
pos 3: picked 'r' (median time=0.001228s) -> 's3cr'
pos 4: picked '3' (median time=0.001474s) -> 's3cr3'
pos 5: picked 't' (median time=0.001722s) -> 's3cr3t'
pos 6: picked '!' (median time=0.002012s) -> 's3cr3t!'
Guessed secret: s3cr3t!
```



```python
# --- Attacker ---
CHARSET = string.ascii_letters + string.digits + string.punctuation  # search space
SAMPLES_PER_TRY = 30

def discover_length(max_len=32):
    """Discover password length by trying lengths 1..max_len"""
    timings = []
    for L in range(1, max_len + 1):
        attempt = "A" * L
        elapsed = time_call(victim_check, attempt)
        timings.append((L, elapsed))
    # choose length with (largest?) — here length equality to secret will often take longer
    best = max(timings, key=lambda x: x[1])
    return best[0], timings

def recover_by_timing(known_len):
    recovered = ""
    for pos in range(known_len):
        best_char = None
        best_time = -1.0
        for ch in CHARSET:
            attempt = (recovered + ch).ljust(known_len, "A")  # fill remaining with dummy chars
            elapsed = time_call(victim_check, attempt)
            if elapsed > best_time:
                best_time = elapsed
                best_char = ch
        recovered += best_char
        print(f"pos {pos}: picked '{best_char}' (median time={best_time:.6f}s) -> {recovered!r}")
    return recovered
```

Side-Channel Attack: Non-Constant Time Op

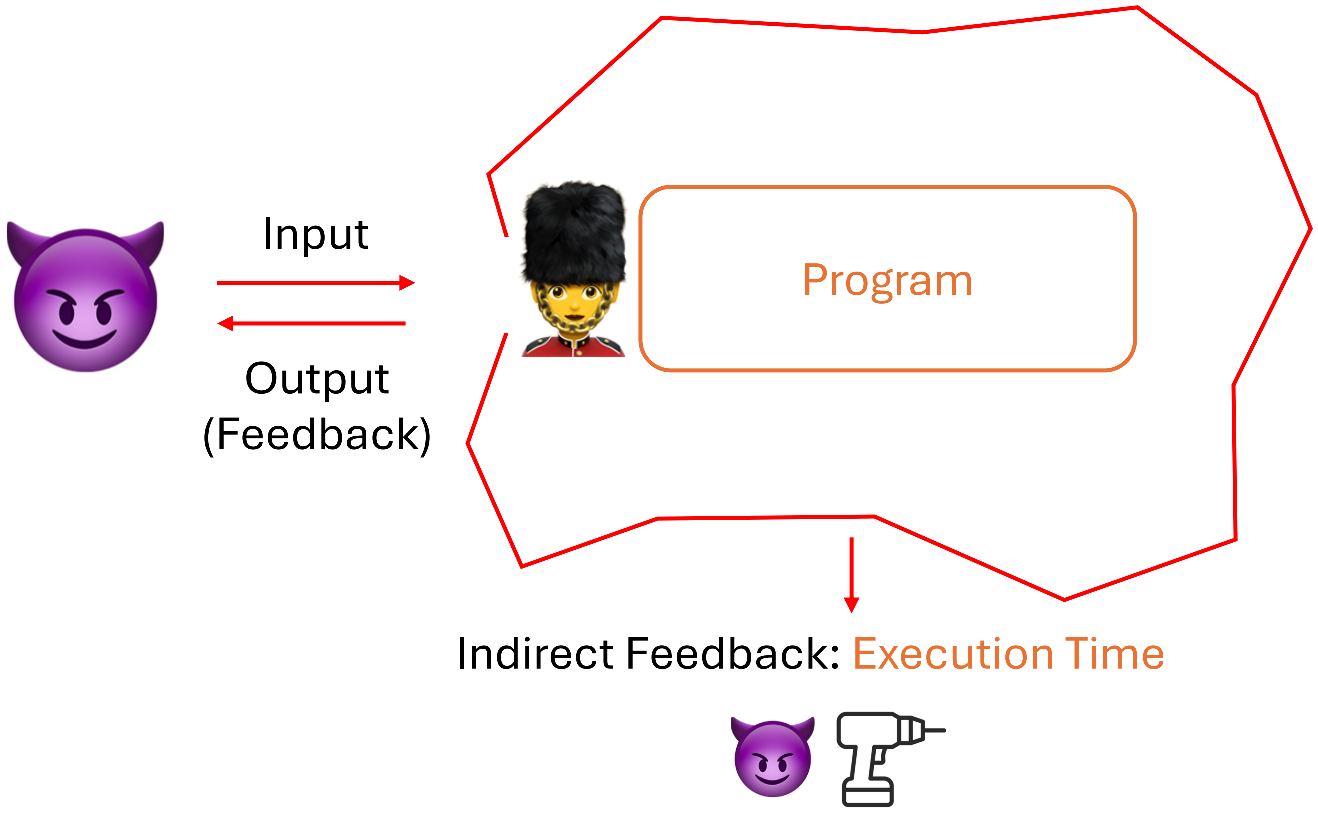


Input

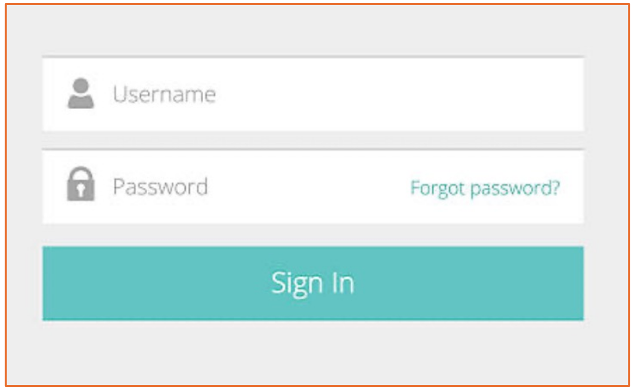
Output (Feedback)

Program

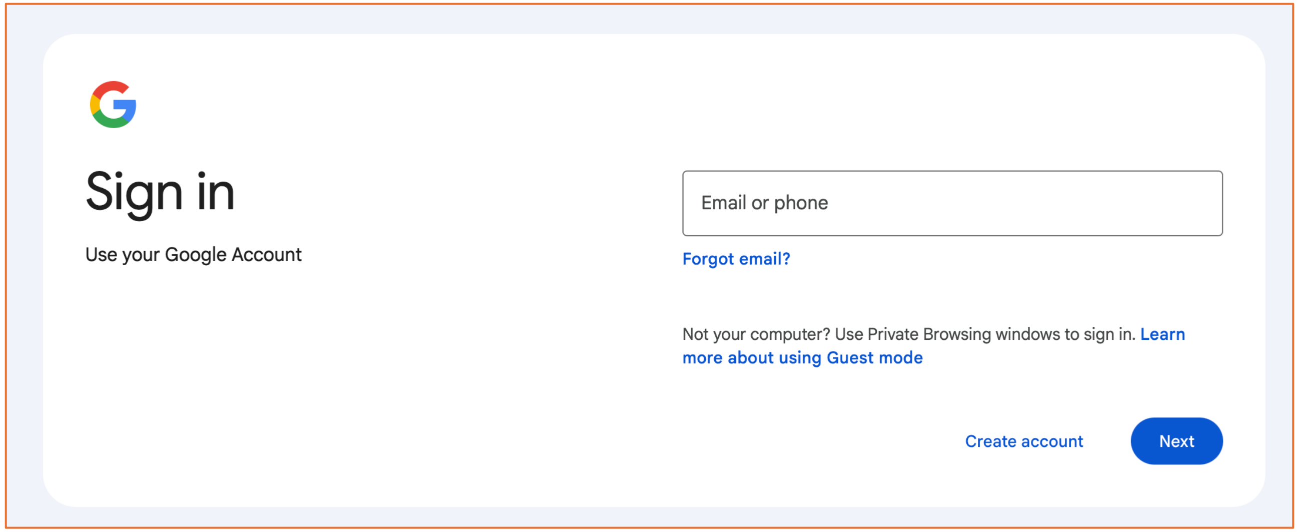
Side-Channel Attack: Non-Constant Time Op



Single Step Login

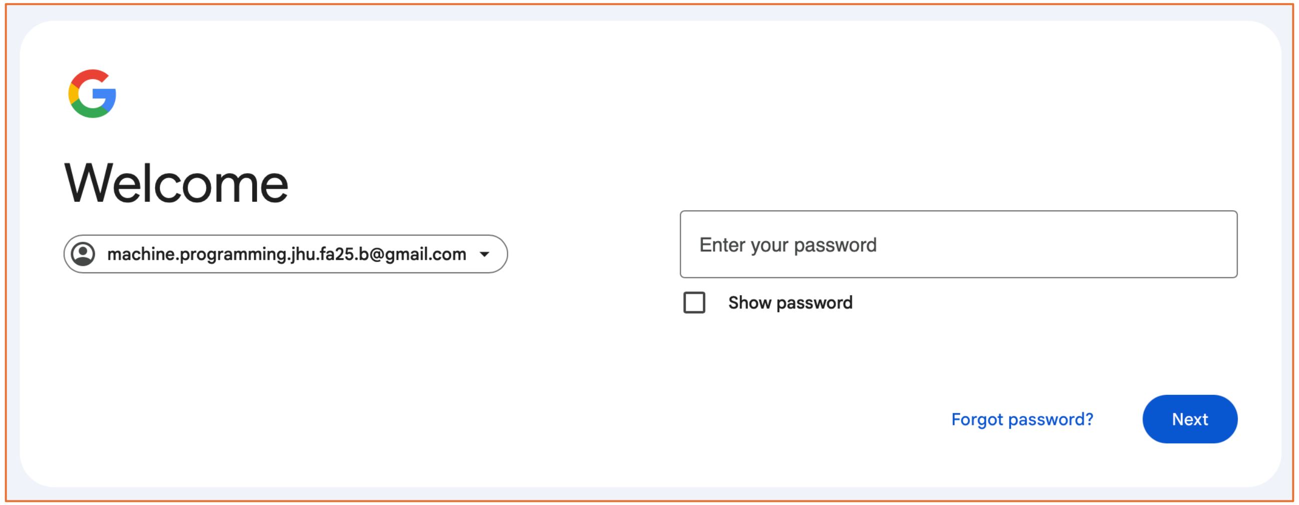


Two Stage Login: Step 1



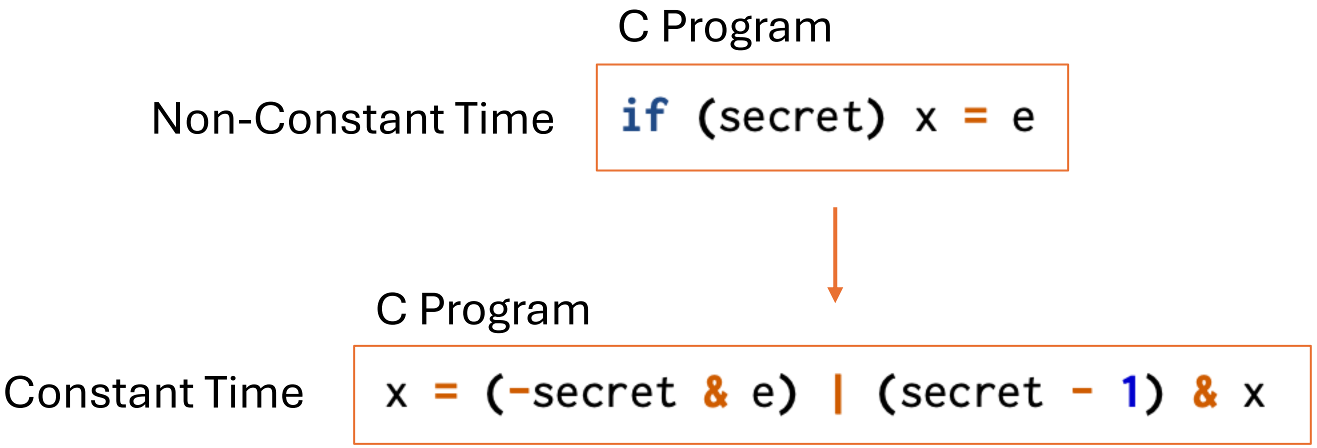
Not your computer? Use Private Browsing windows to sign in. Learn more about using Guest mode

Two Stage Login: Step 2



machine.programming.jhu.fa25.b@gmail.com

Mitigation: Constant-Time Operations



C Program

Non-Constant Time

```
if (secret) x = e
```

C Program

Constant Time

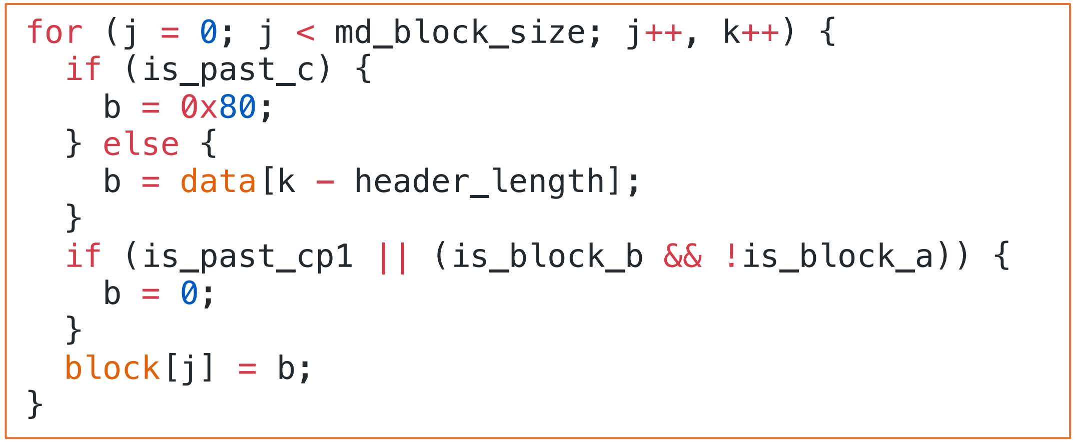
```
x = (-secret & e) | (secret - 1) & x
```

FaCT: A DSL for Timing-Sensitive Computation, Cauligi et. al., PLDI 2019

Mitigation: Constant-Time Operations

C Program

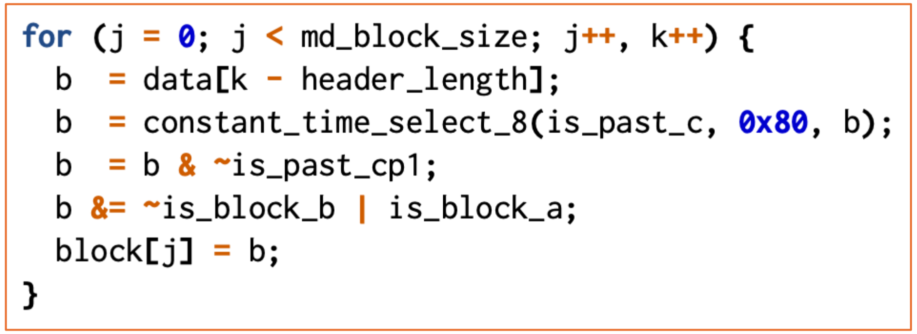
Non-Constant Time



```
for (j = 0; j < md_block_size; j++, k++) {
  if (is_past_c) {
    b = 0x80;
  } else {
    b = data[k - header_length];
  }
  if (is_past_cp1 || (is_block_b && !is_block_a)) {
    b = 0;
  }
  block[j] = b;
}
```

C Program

Constant Time



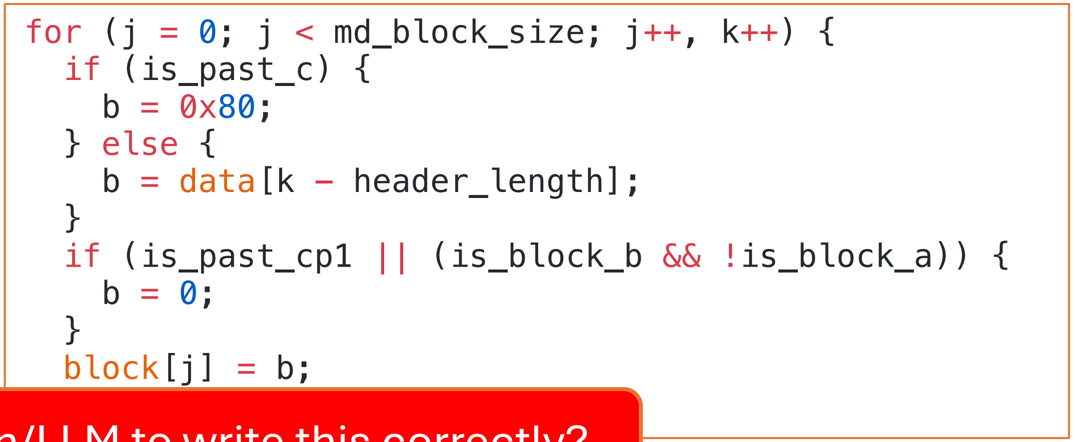
```
for (j = 0; j < md_block_size; j++, k++) {
  b  = data[k - header_length];
  b  = constant_time_select_8(is_past_c, 0x80, b);
  b  = b & ~is_past_cp1;
  b &= ~is_block_b | is_block_a;
  block[j] = b;
}
```

FaCT: A DSL for Timing-Sensitive Computation, Cauligi et. al., PLDI 2019

Mitigation: Constant-Time Operations

Non-Constant Time

C Program

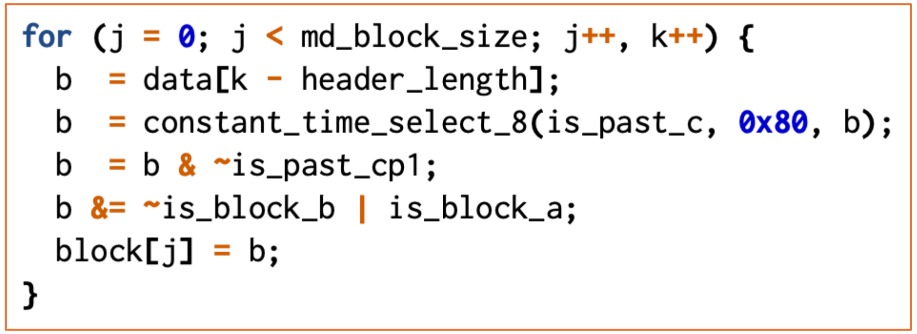


```
for (j = 0; j < md_block_size; j++, k++) {
  if (is_past_c) {
    b = 0x80;
  } else {
    b = data[k - header_length];
  }
  if (is_past_cp1 || (is_block_b && !is_block_a)) {
    b = 0;
  }
  block[j] = b;
```

Are we trusting Human/LLM to write this correctly?

Constant Time

C Program

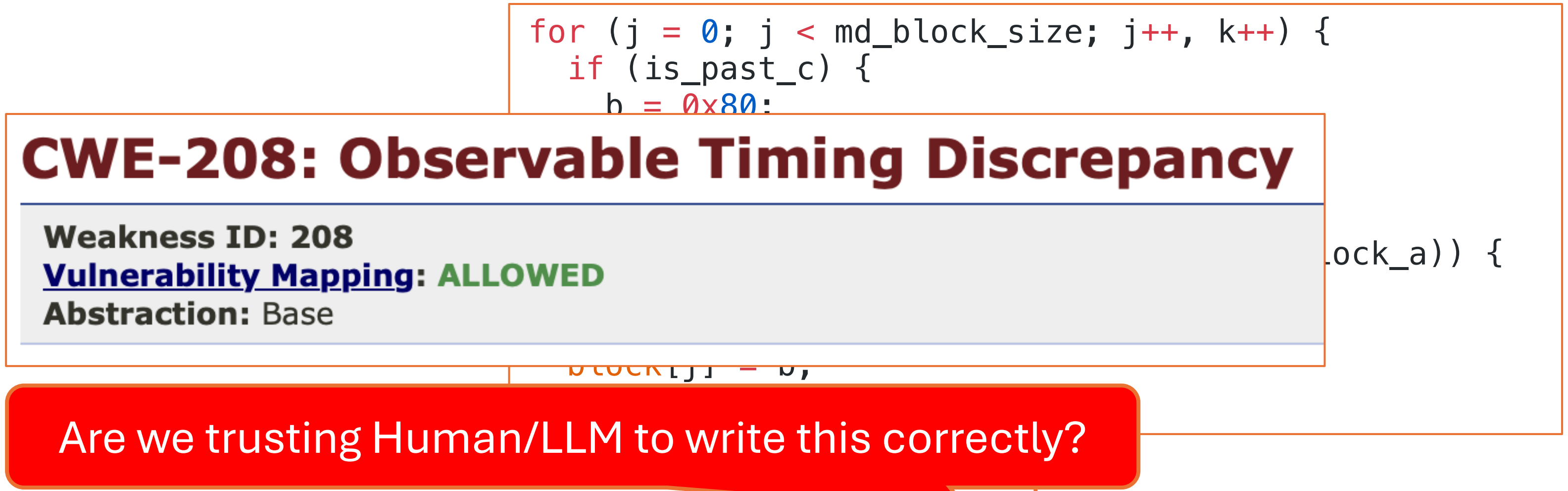


```
for (j = 0; j < md_block_size; j++, k++) {
  b  = data[k - header_length];
  b  = constant_time_select_8(is_past_c, 0x80, b);
  b  = b & ~is_past_cp1;
  b &= ~is_block_b | is_block_a;
  block[j] = b;
}
```

FaCT: A DSL for Timing-Sensitive Computation, Cauligi et. al., PLDI 2019

Mitigation: Constant-Time Operations

C Program



```
for (j = 0; j < md_block_size; j++, k++) {
  if (is_past_c) {
    b = 0x80;
```

```
ock_a)) {
```

```
  block[j] = b;
```

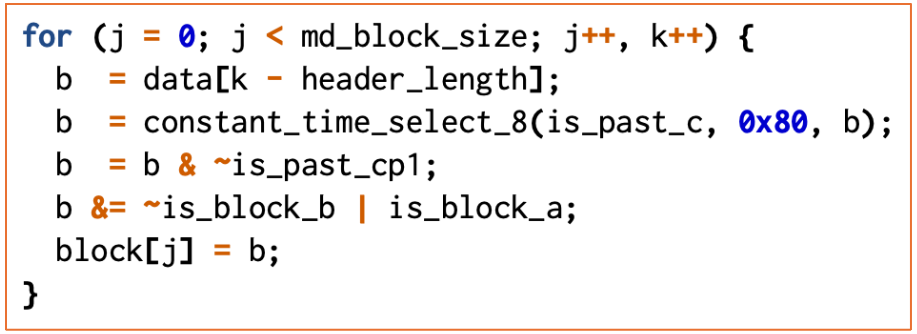
CWE-208: Observable Timing Discrepancy

Weakness ID: 208
Vulnerability Mapping: ALLOWED
Abstraction: Base

Are we trusting Human/LLM to write this correctly?

C Program

Constant Time



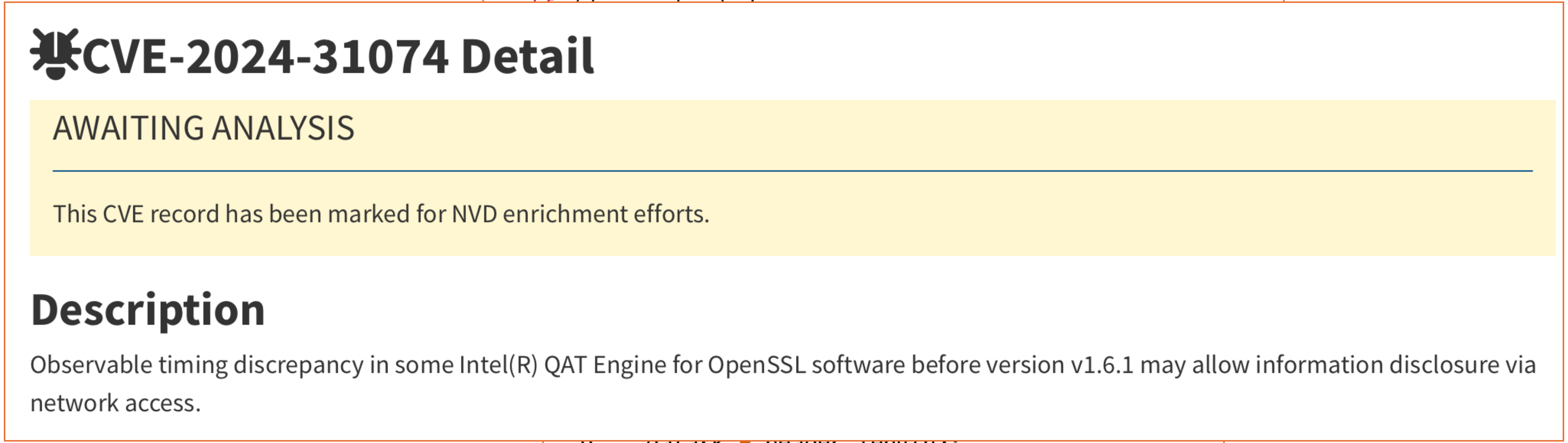
```
for (j = 0; j < md_block_size; j++, k++) {
  b  = data[k - header_length];
  b  = constant_time_select_8(is_past_c, 0x80, b);
  b  = b & ~is_past_cp1;
  b &= ~is_block_b | is_block_a;
  block[j] = b;
}
```

FaCT: A DSL for Timing-Sensitive Computation, Cauligi et. al., PLDI 2019

Mitigation: Constant-Time Operations

C Program

```
for (j = 0; j < md_block_size; j++, k++) {
```



CVE-2024-31074 Detail

AWAITING ANALYSIS

This CVE record has been marked for NVD enrichment efforts.

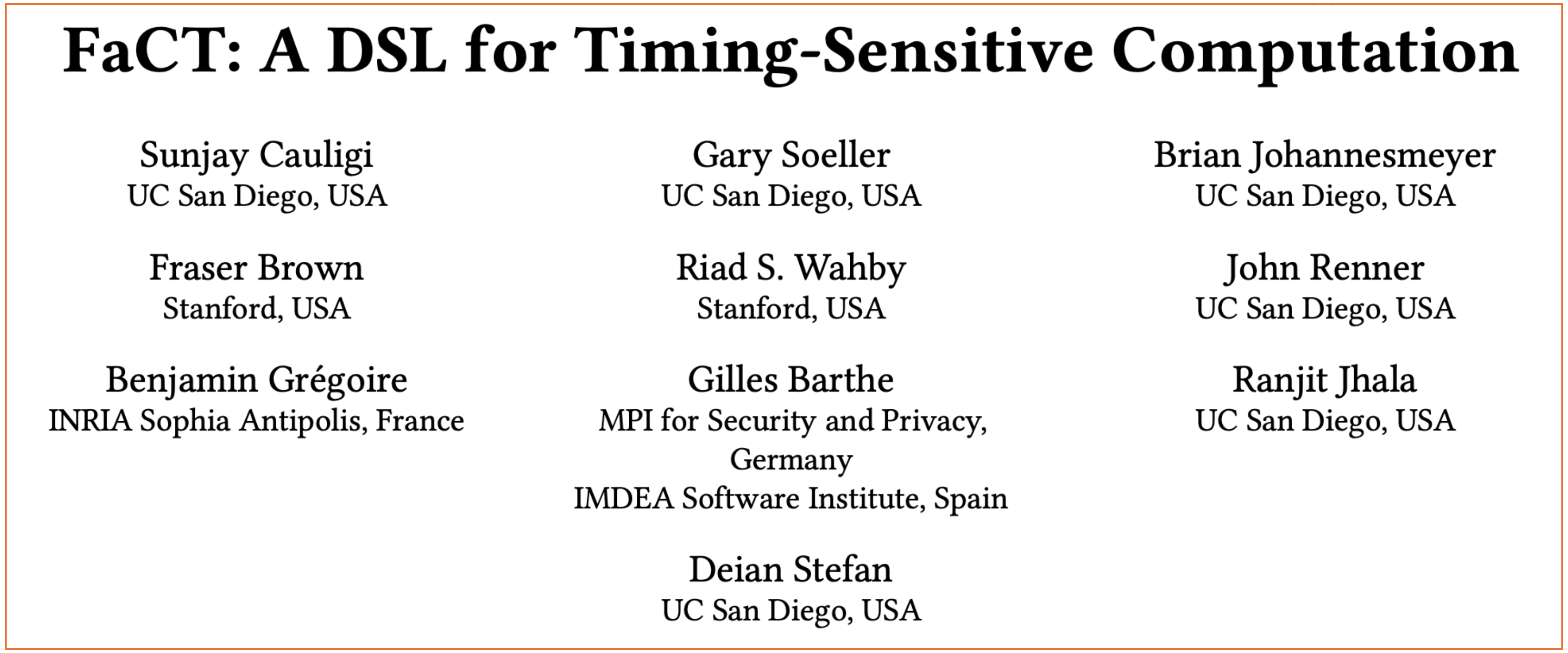
Description

Observable timing discrepancy in some Intel(R) QAT Engine for OpenSSL software before version v1.6.1 may allow information disclosure via network access.

Constant Time

```
  b  = data[k - header_length];
  b  = constant_time_select_8(is_past_c, 0x80, b);
  b  = b & ~is_past_cp1;
  b &= ~is_block_b | is_block_a;
  block[j] = b;
}
```

FaCT: A DSL for Timing-Sensitive Computation, Cauligi et. al., PLDI 2019



FaCT: A DSL for Timing-Sensitive Computation

Sunjay Cauligi
UC San Diego, USA

Gary Soeller
UC San Diego, USA

Brian Johannesmeyer
UC San Diego, USA

Fraser Brown
Stanford, USA

Riad S. Wahby
Stanford, USA

John Renner
UC San Diego, USA

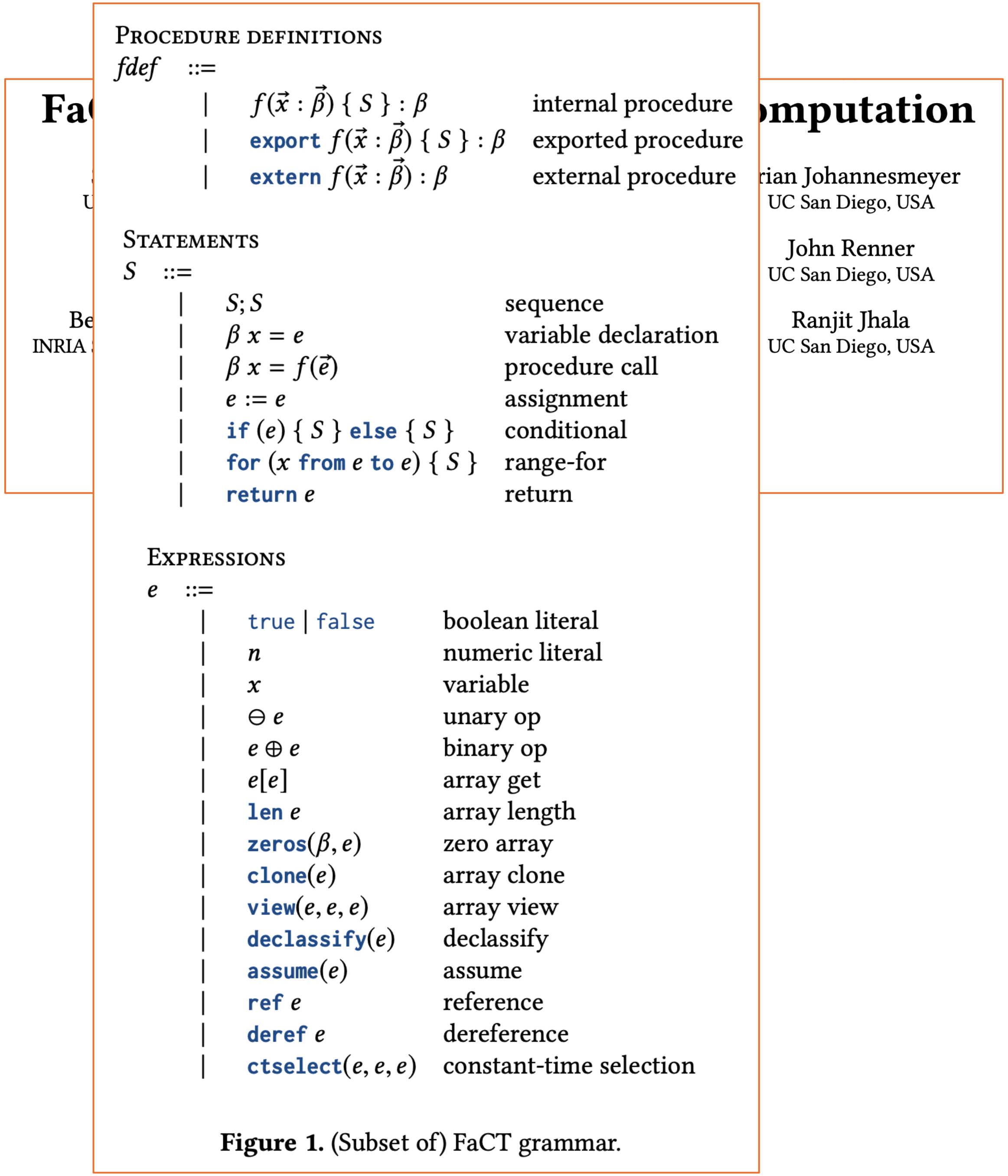
Benjamin Grégoire
INRIA Sophia Antipolis, France

Gilles Barthe
MPI for Security and Privacy, Germany
IMDEA Software Institute, Spain

Ranjit Jhala
UC San Diego, USA

Deian Stefan
UC San Diego, USA

FaCT: A DSL for Timing-Sensitive Computation, Cauligi et. al., PLDI 2019



Fa omputation

U

Be
INRIA

rian Johannesmeyer
UC San Diego, USA

John Renner
UC San Diego, USA

Ranjit Jhala
UC San Diego, USA

Procedure definitions

$fdef$ ::=

	Production	Description
|	$f(\vec{x} : \vec{\beta})\ \{\ S\ \} : \beta$	internal procedure
|	$\texttt{export}\ f(\vec{x} : \vec{\beta})\ \{\ S\ \} : \beta$	exported procedure
|	$\texttt{extern}\ f(\vec{x} : \vec{\beta}) : \beta$	external procedure

Statements

S ::=

	Production	Description
|	$S; S$	sequence
|	$\beta\ x = e$	variable declaration
|	$\beta\ x = f(\vec{e})$	procedure call
|	$e := e$	assignment
|	$\texttt{if}\ (e)\ \{\ S\ \}\ \texttt{else}\ \{\ S\ \}$	conditional
|	$\texttt{for}\ (x\ \texttt{from}\ e\ \texttt{to}\ e)\ \{\ S\ \}$	range-for
|	$\texttt{return}\ e$	return

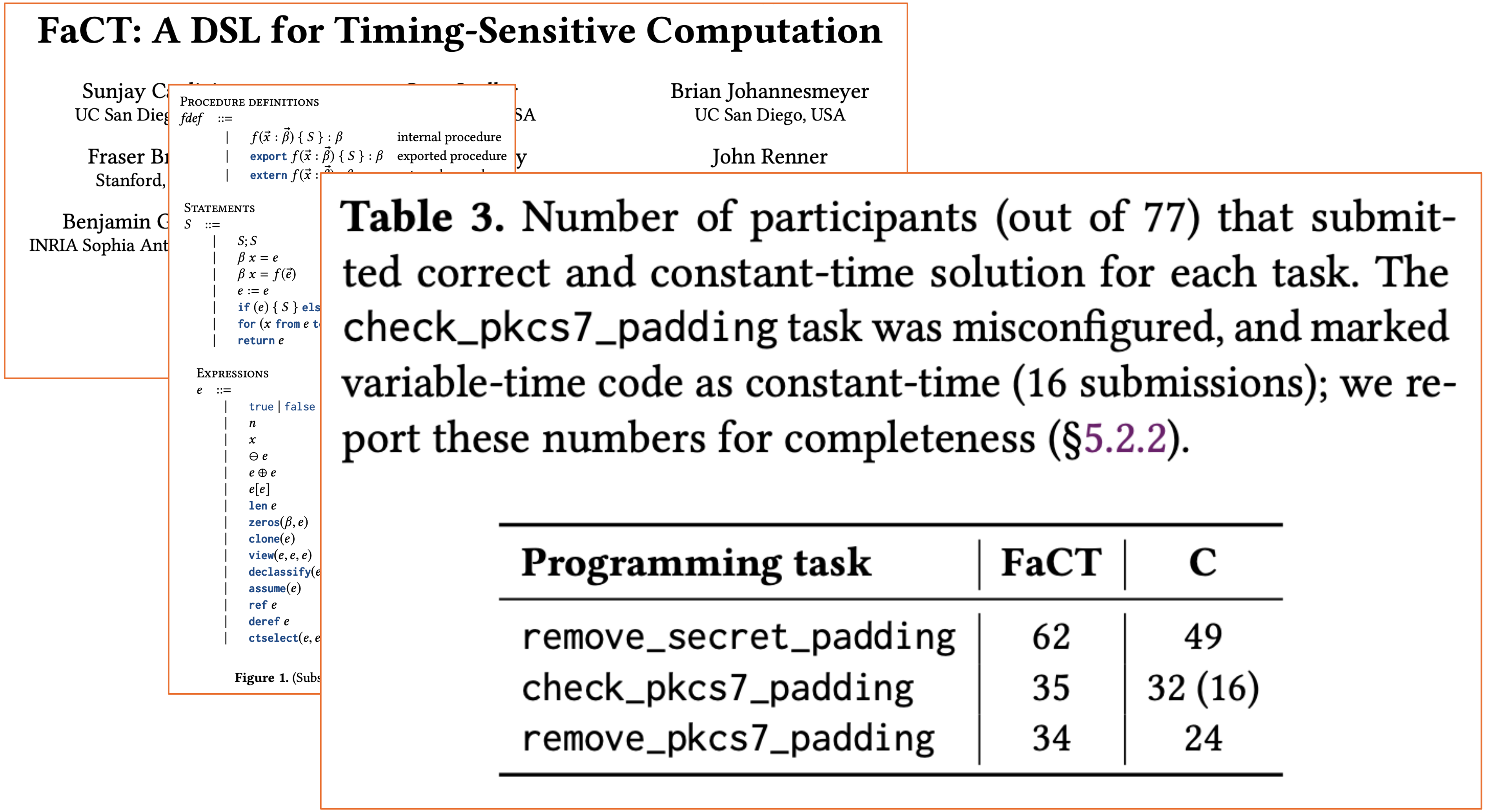
Expressions

e ::=

	Production	Description
|	$\texttt{true} \mid \texttt{false}$	boolean literal
|	n	numeric literal
|	x	variable
|	$\ominus\ e$	unary op
|	$e \oplus e$	binary op
|	$e[e]$	array get
|	$\texttt{len}\ e$	array length
|	$\texttt{zeros}(\beta, e)$	zero array
|	$\texttt{clone}(e)$	array clone
|	$\texttt{view}(e, e, e)$	array view
|	$\texttt{declassify}(e)$	declassify
|	$\texttt{assume}(e)$	assume
|	$\texttt{ref}\ e$	reference
|	$\texttt{deref}\ e$	dereference
|	$\texttt{ctselect}(e, e, e)$	constant-time selection

Figure 1. (Subset of) FaCT grammar.

FaCT: A DSL for Timing-Sensitive Computation, Cauligi et. al., PLDI 2019



FaCT: A DSL for Timing-Sensitive Computation

Sunjay C
UC San Dieg
SA

Brian Johannesmeyer
UC San Diego, USA

Fraser Br
Stanford,
y

John Renner

Benjamin G
INRIA Sophia Ant

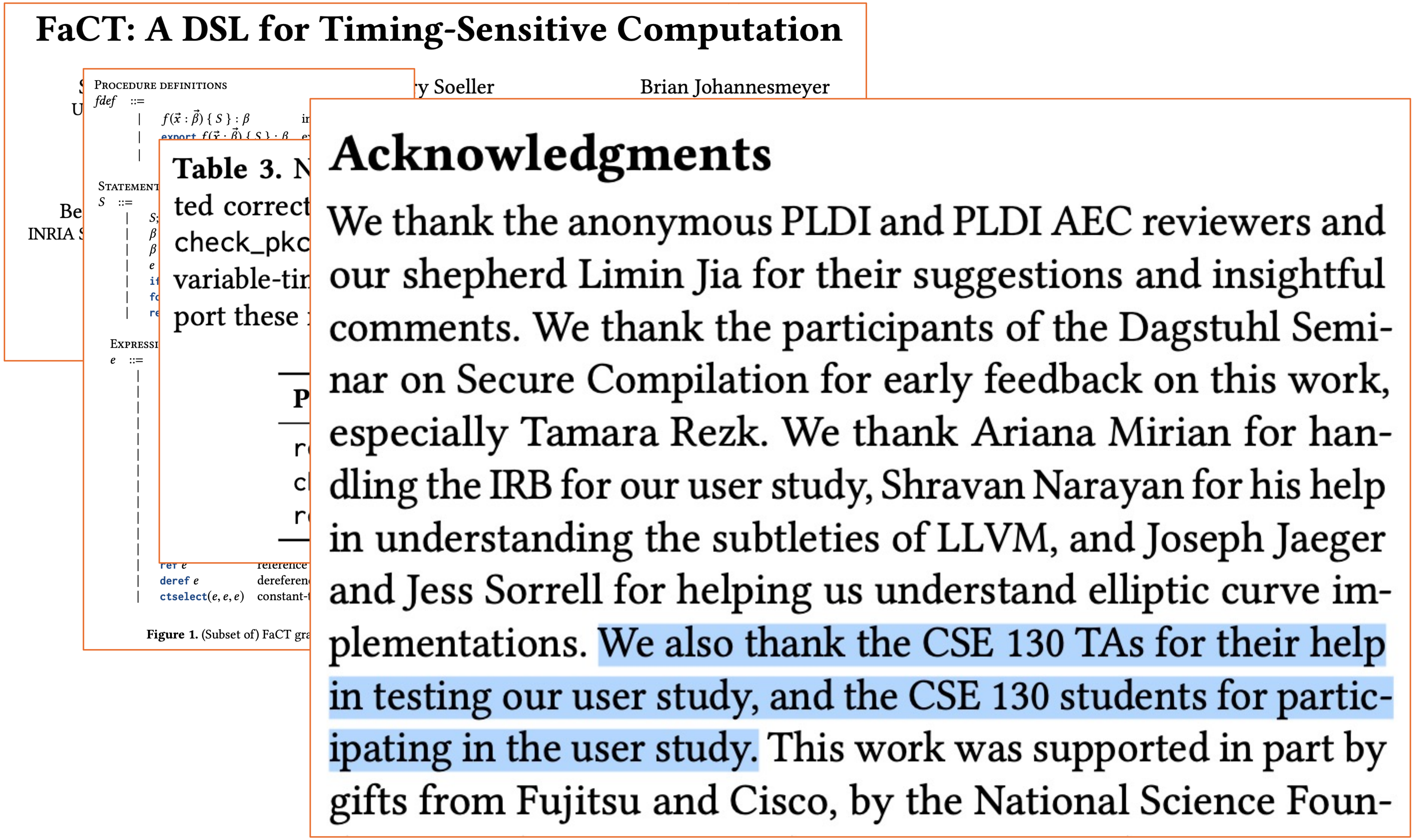
```
PROCEDURE DEFINITIONS
fdef  ::=
      |  f(x⃗ : β⃗) { S } : β          internal procedure
      |  export f(x⃗ : β⃗) { S } : β   exported procedure
      |  extern f(x⃗ :

STATEMENTS
S  ::=
   |  S; S
   |  β x = e
   |  β x = f(e⃗)
   |  e := e
   |  if (e) { S } els
   |  for (x from e t
   |  return e

EXPRESSIONS
e  ::=
   |  true | false
   |  n
   |  x
   |  ⊖ e
   |  e ⊕ e
   |  e[e]
   |  len e
   |  zeros(β, e)
   |  clone(e)
   |  view(e, e, e)
   |  declassify(e
   |  assume(e)
   |  ref e
   |  deref e
   |  ctselect(e, e
```

Figure 1. (Subs

Table 3. Number of participants (out of 77) that submitted correct and constant-time solution for each task. The `check_pkcs7_padding` task was misconfigured, and marked variable-time code as constant-time (16 submissions); we report these numbers for completeness (§5.2.2).

Programming task	**FaCT**	**C**
`remove_secret_padding`	62	49
`check_pkcs7_padding`	35	32 (16)
`remove_pkcs7_padding`	34	24

FaCT: A DSL for Timing-Sensitive Computation, Cauligi et. al., PLDI 2019



FaCT: A DSL for Timing-Sensitive Computation

S ... y Soeller   Brian Johannesmeyer

U

Be

INRIA S

Procedure definitions

```
fdef  ::=
      |  f(x⃗ : β⃗) { S } : β          in
      |  export f(x⃗ : β⃗) { S } : β   ex
      |
```

Statement

```
S  ::=
      |  S;
      |  β
      |  β
      |  e
      |  if
      |  fo
      |  re
```

Expressi

```
e  ::=
      |  ref e               reference
      |  deref e             dereferen
      |  ctselect(e, e, e)   constant-t
```

Figure 1. (Subset of) FaCT gra

Table 3. N

ted correct

check_pkc

variable-tin

port these

P

r

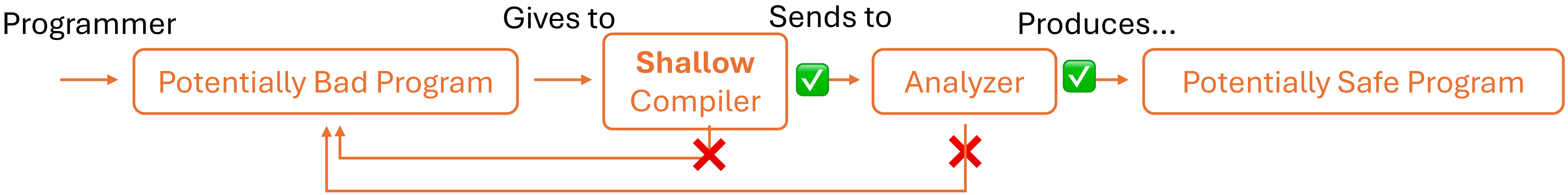
c

r

Acknowledgments

We thank the anonymous PLDI and PLDI AEC reviewers and our shepherd Limin Jia for their suggestions and insightful comments. We thank the participants of the Dagstuhl Seminar on Secure Compilation for early feedback on this work, especially Tamara Rezk. We thank Ariana Mirian for handling the IRB for our user study, Shravan Narayan for his help in understanding the subtleties of LLVM, and Joseph Jaeger and Jess Sorrell for helping us understand elliptic curve implementations. We also thank the CSE 130 TAs for their help in testing our user study, and the CSE 130 students for participating in the user study. This work was supported in part by gifts from Fujitsu and Cisco, by the National Science Foun-

FaCT: A DSL for Timing-Sensitive Computation, Cauligi et. al., PLDI 2019



Programmer

Potentially Bad Program

Gives to

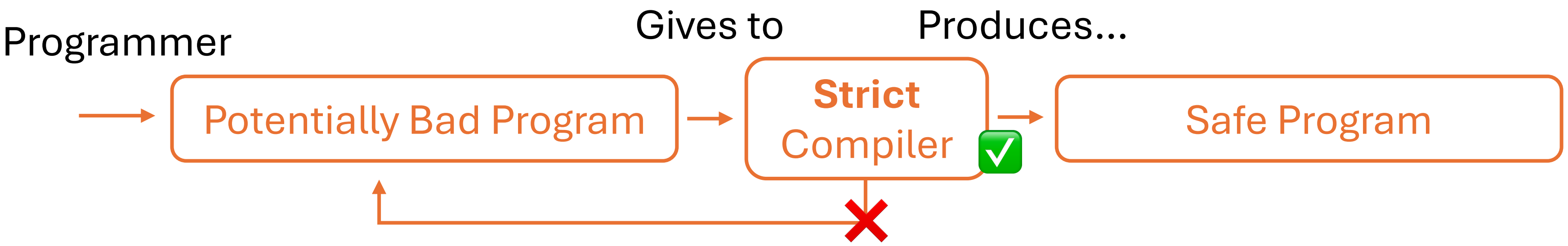
Shallow Compiler ❌ ✅

Sends to

Analyzer ❌ ✅

Produces...

Potentially Safe Program



Programmer

Potentially Bad Program

Gives to

Strict Compiler ❌ ✅

Produces...

Safe Program

Takeaway

- There are many generally used languages with different safety features: memory safety, concurrency safety, smart-contract safety, …
- Instead of writing buggy code and use analysis tools to detect and fix them afterwards, we may prefer employing a better language that is safe-by-construct
  - The language may be more limiting, but is safer
  - A good safe language mitigates the limitations well and is fast
- We want to ask LLM to write programs in safer languages
  - It maybe harder to get the compiler to compile the program, but the compiled program already has good and provable safety properties
  - E.g., Generate Rust > C
  - E.g., Generate TypeScript > JavaScript

Logistics – Week 10

- Oral Presentations
    - Emails are being sending out; plans established
- Final Projects
    - Final project proposal: 1 page PDF (due on Sunday)
    - Submit on GradeScope
    - Send email to the instructor questions