What is memory safe and why Linux needs to be memory safe

Presenter : Mateo

E-LUG IRAN The ELUG Event

About me

Hi I'm Mateo — a passionate and curious programmer. I'm interested in software engineering, networking, and Linux, and I have a love for algorithms.

My work experience includes backend web development and building network-related tools.

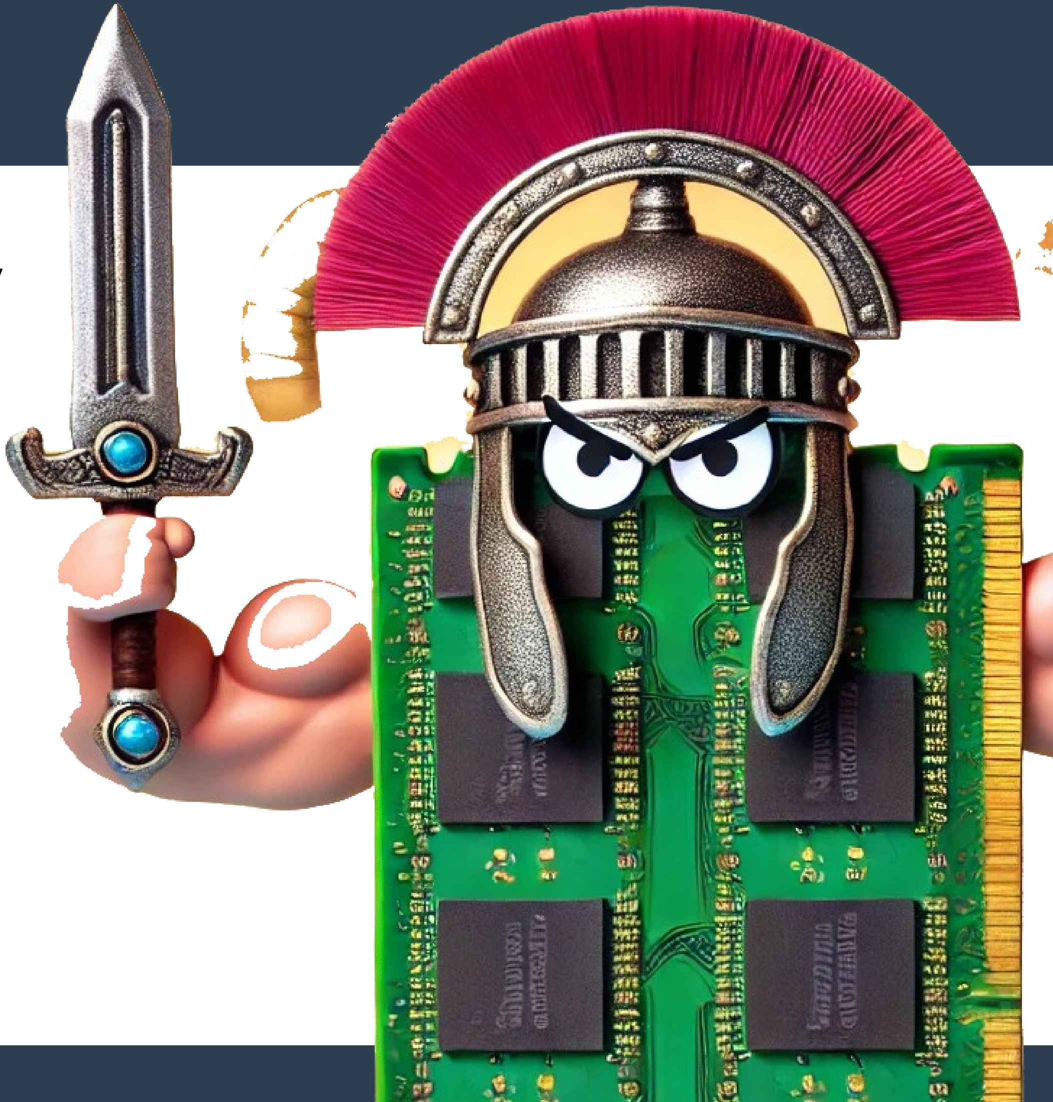
In this conference, I aim to talk about memory safety in linux and hope this presentation will be an opportunity for knowledge exchange and further learning.

Github : https://github.com/mateo-rfz
Email : mahdifeyzolahy@gmail.com
Elug Github : https://github.com/elugiran

Memory safe PR

http://linketobede.ir/elug

What is memory safe ?



A memory-safe program:

Accesses control

Free controller

Resource management

Accesses control

```
#include <iostream>
using namespace std;

int main() {
    int* ptr = new int(5);
    delete ptr;  // free the memory
    cout << *ptr << endl;  // use after free
    return 0;
}
```

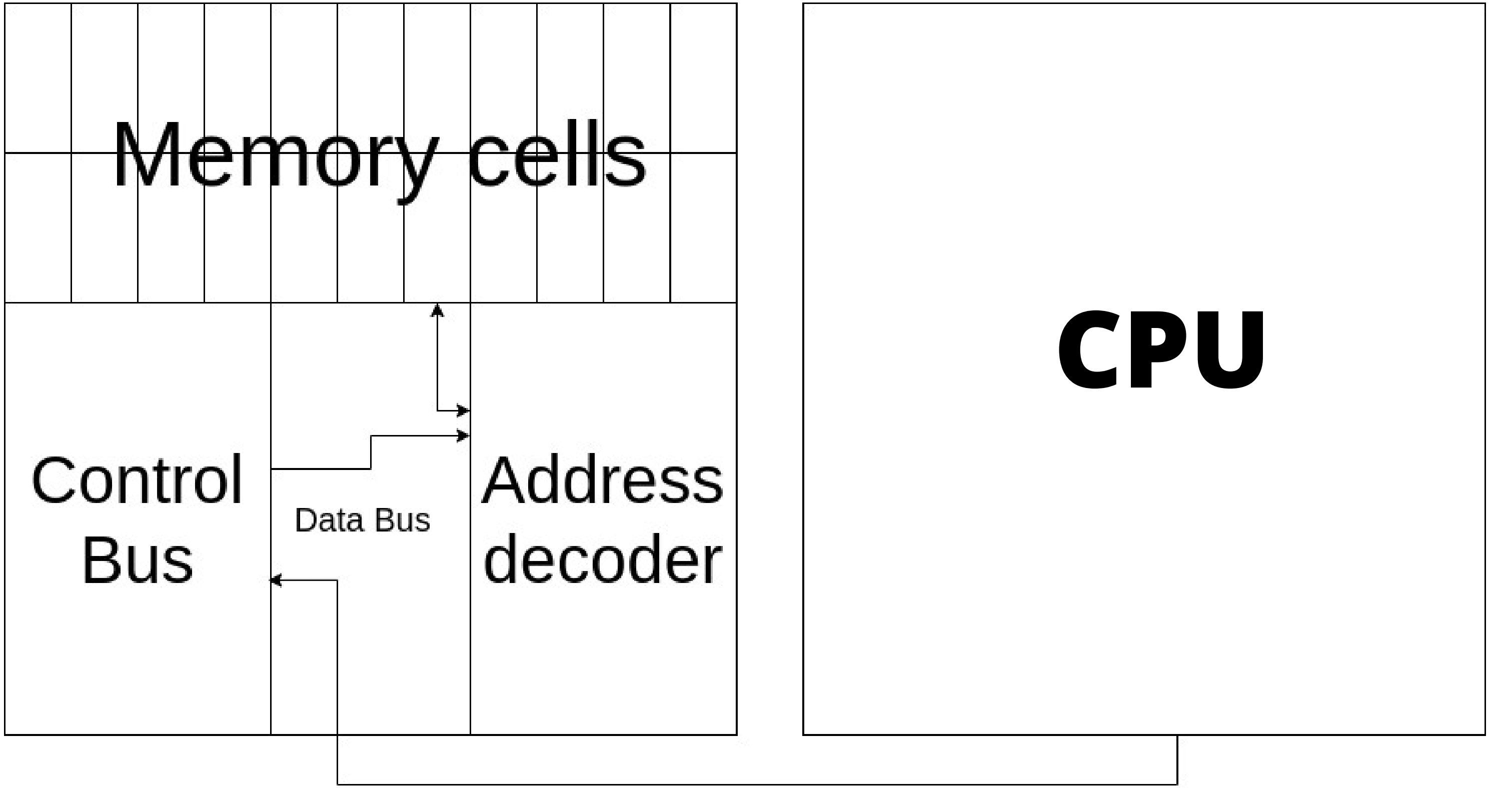
Garbage value

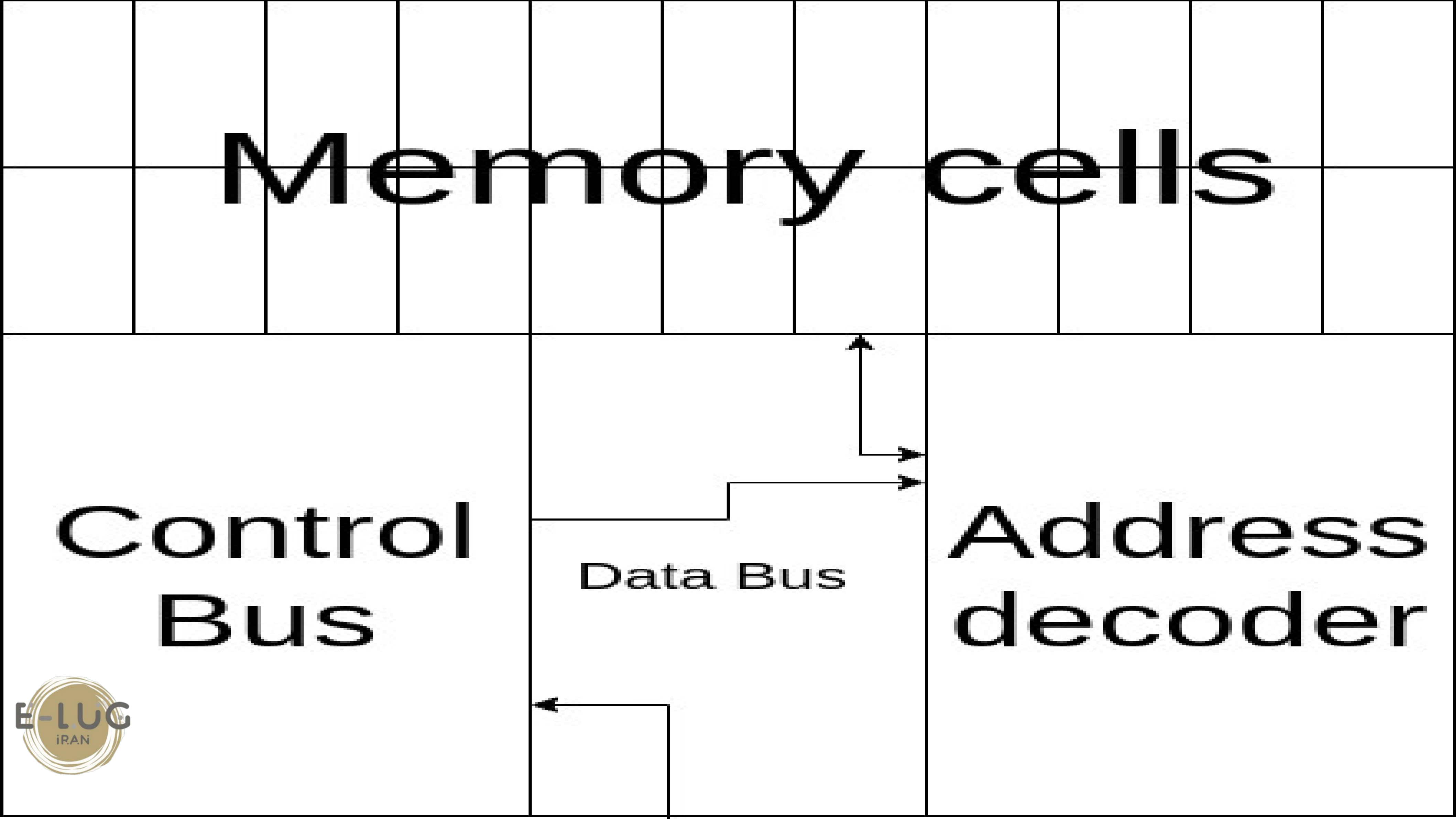
Memory management

```
#include <iostream>
#include <cstring>
using namespace std;

int main() {
    char buffer[10];
    cout << "Enter a string: ";
    cin >> buffer;  //buffer overflow
    cout << "You entered: " << buffer << endl;
    return 0;
}
```

Crash , Garbage value

Ram diagram





Memory cells

Control Bus

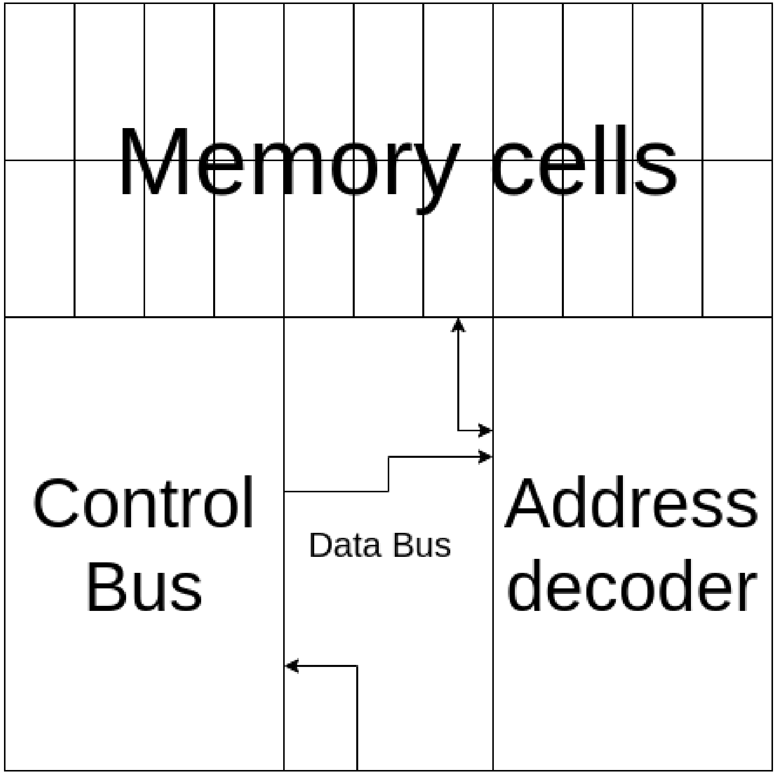
Data Bus

Address decoder

Control Bus

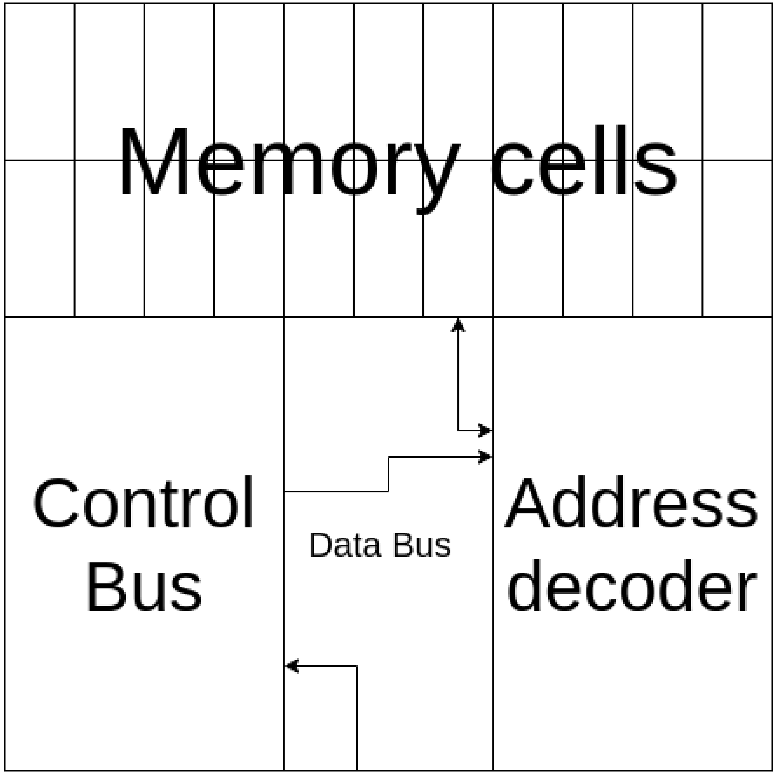
Wrong command(Read/Write)

Ex.Use after Free



Address Decoder

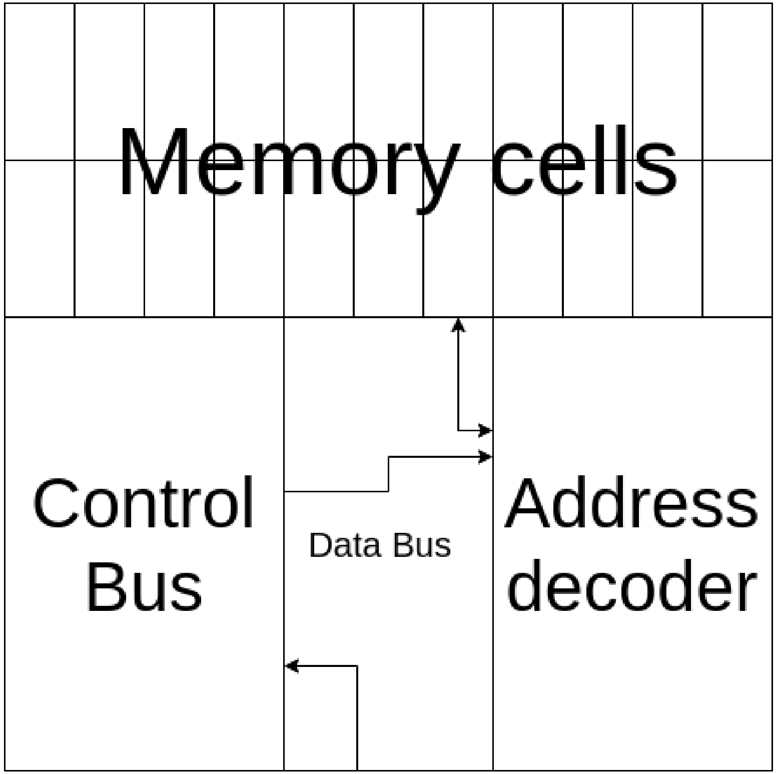
Out of range Access



Memory cells

Memory address space

ex. Buffer overflow



Unsafe issues

- Security
- Stability
- Resource management

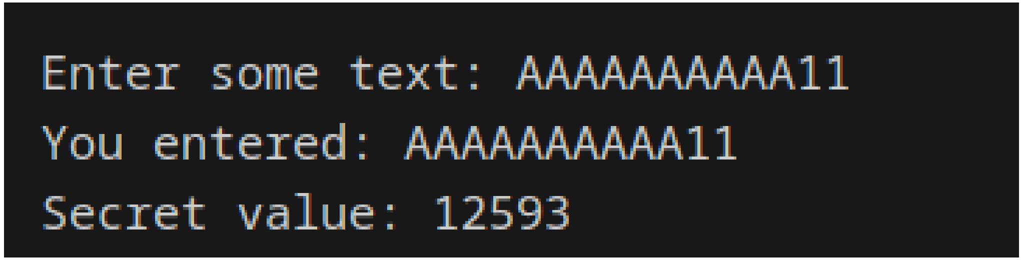
Big issues for unsafe languages

- Buffer overflow
- Use after-free
- Memory leak
- Stack smashing

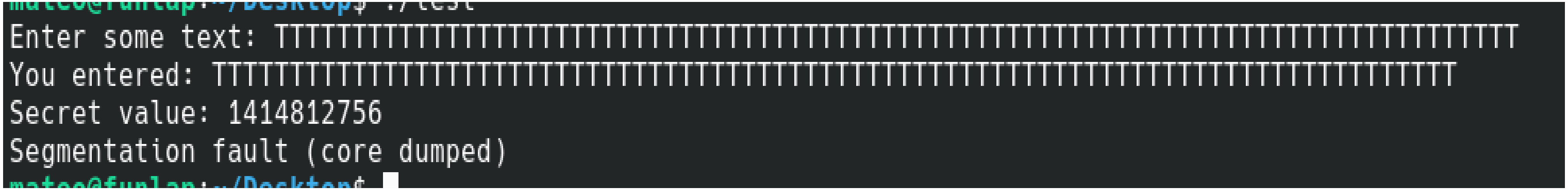
Buffer overflow

```
#include <iostream>
using namespace std;

int main() {
    char buffer[10];
    int secret = 42;

    cout << "Enter some text: ";
    cin >> buffer;
    cout << "You entered: " << buffer << endl;
    cout << "Secret value: " << secret << endl;

    return 0;
}
```

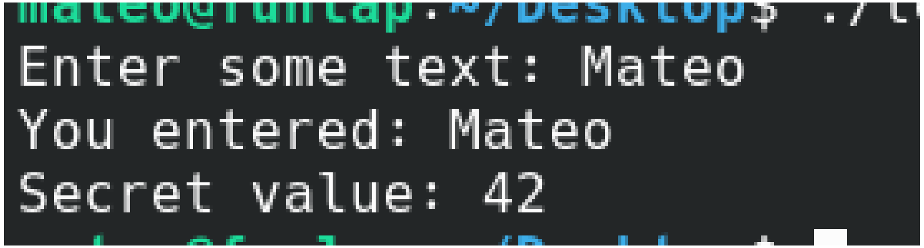
OUTPUT



```
Enter some text: AAAAAAAAAA11
You entered: AAAAAAAAAA11
Secret value: 12593
```



```
mateo@funlap:~/Desktop$ ./test
Enter some text: TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
You entered: TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
Secret value: 1414812756
Segmentation fault (core dumped)
mateo@funlap:~/Desktop$
```

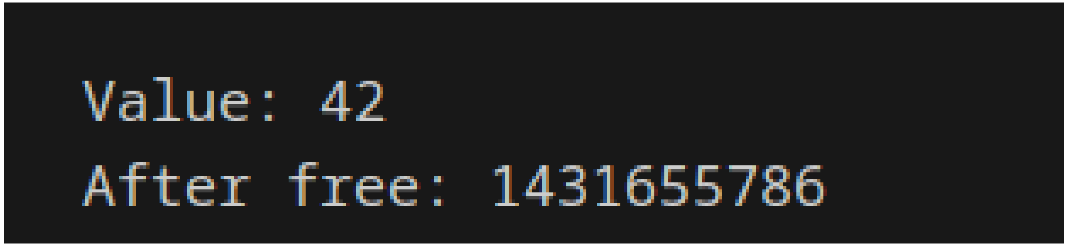


```
mateo@funlap:~/Desktop$ ./t
Enter some text: Mateo
You entered: Mateo
Secret value: 42
mateo@funlap:~/Desktop$
```

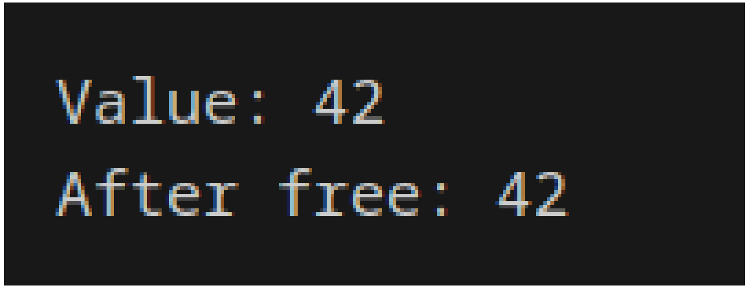
Use after-free

```
#include <iostream>
using namespace std;

int main() {
    int* ptr = new int(42);
    cout << "Value: " << *ptr << endl;

    delete ptr;

    cout << "After free: " << *ptr << endl;

    return 0;
}
```

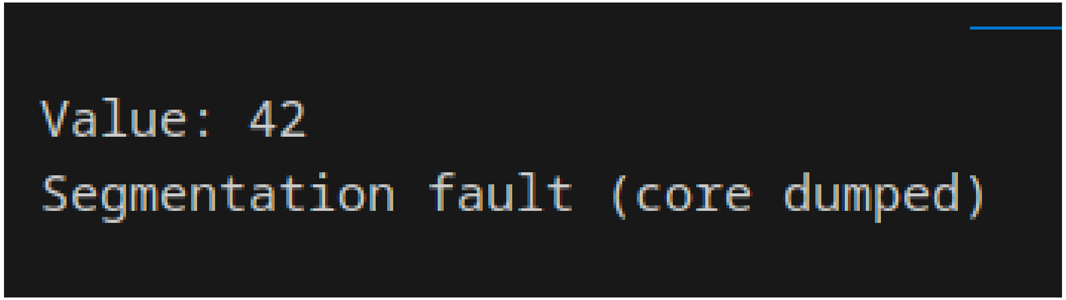
OUTPUT



```
Value: 42
After free: 1431655786
```



```
Value: 42
After free: 42
```



```
Value: 42
Segmentation fault (core dumped)
```

Memory leak

```
#include <iostream>
using namespace std;

void leak() {
    int* ptr = new int(42);
}

int main() {
    while (true) {
        leak();
    }
    return 0;
}
```

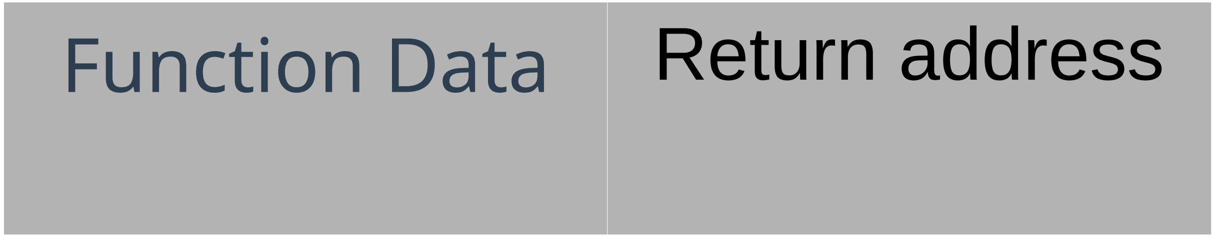
OUTPUT

Stack smashing

```
#include <iostream>
using namespace std;

void hacker() {
    cout << "Hey you on tester" << endl;
}

void vulnerable() {
    char buffer[10];
    cout << "Enter some text: ";
    cin >> buffer;
}
```

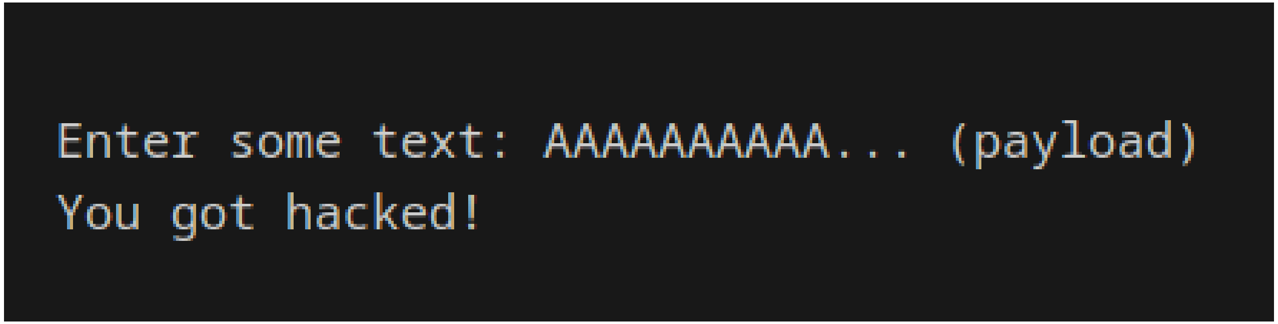
```
int main() {
    vulnerable();
    cout << "Back to main!" << endl;
    return 0;
}
```

OUTPUT



Function Data

Return address



```
Enter some text: AAAAAAAAAA... (payload)
You got hacked!
```

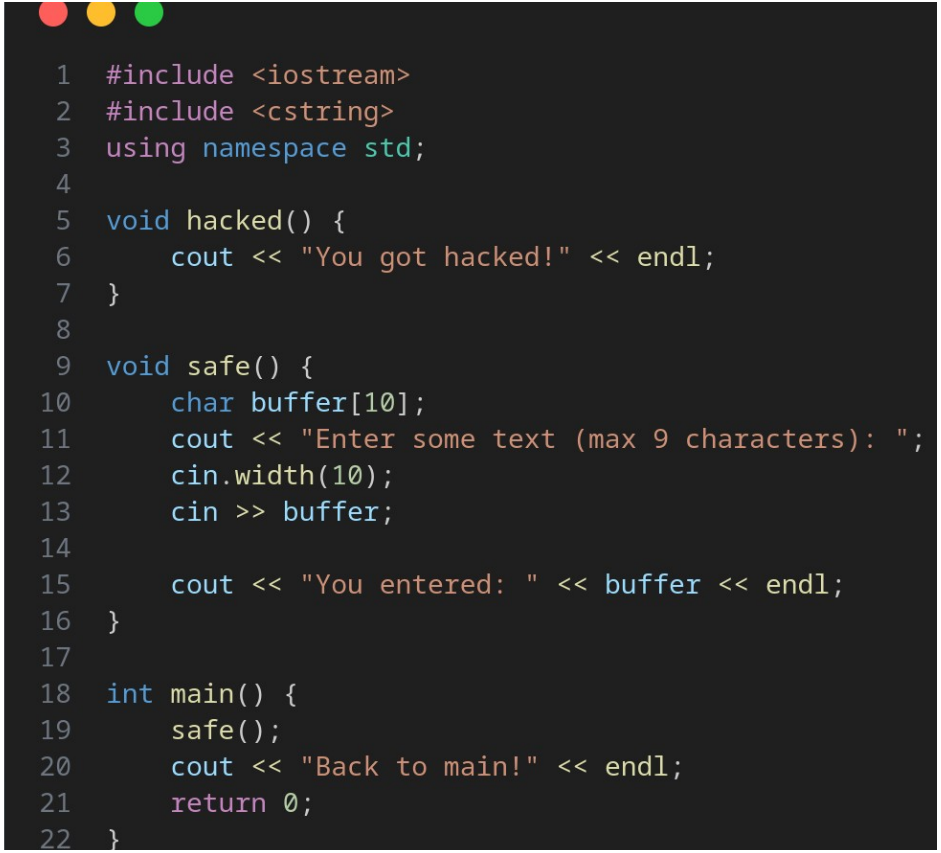
Solutions for Security Issues

1 – Using memory safe Language

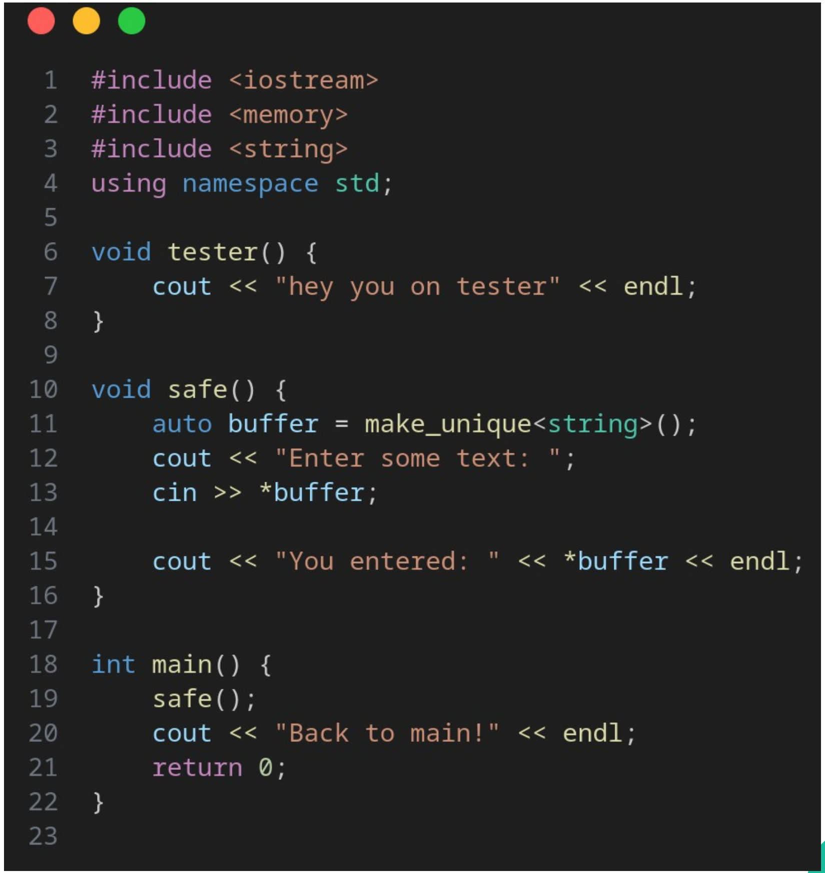
2 – Memory safe tools and libraries

3 – Static and Dynamic Analysis

Use modules in c/c++

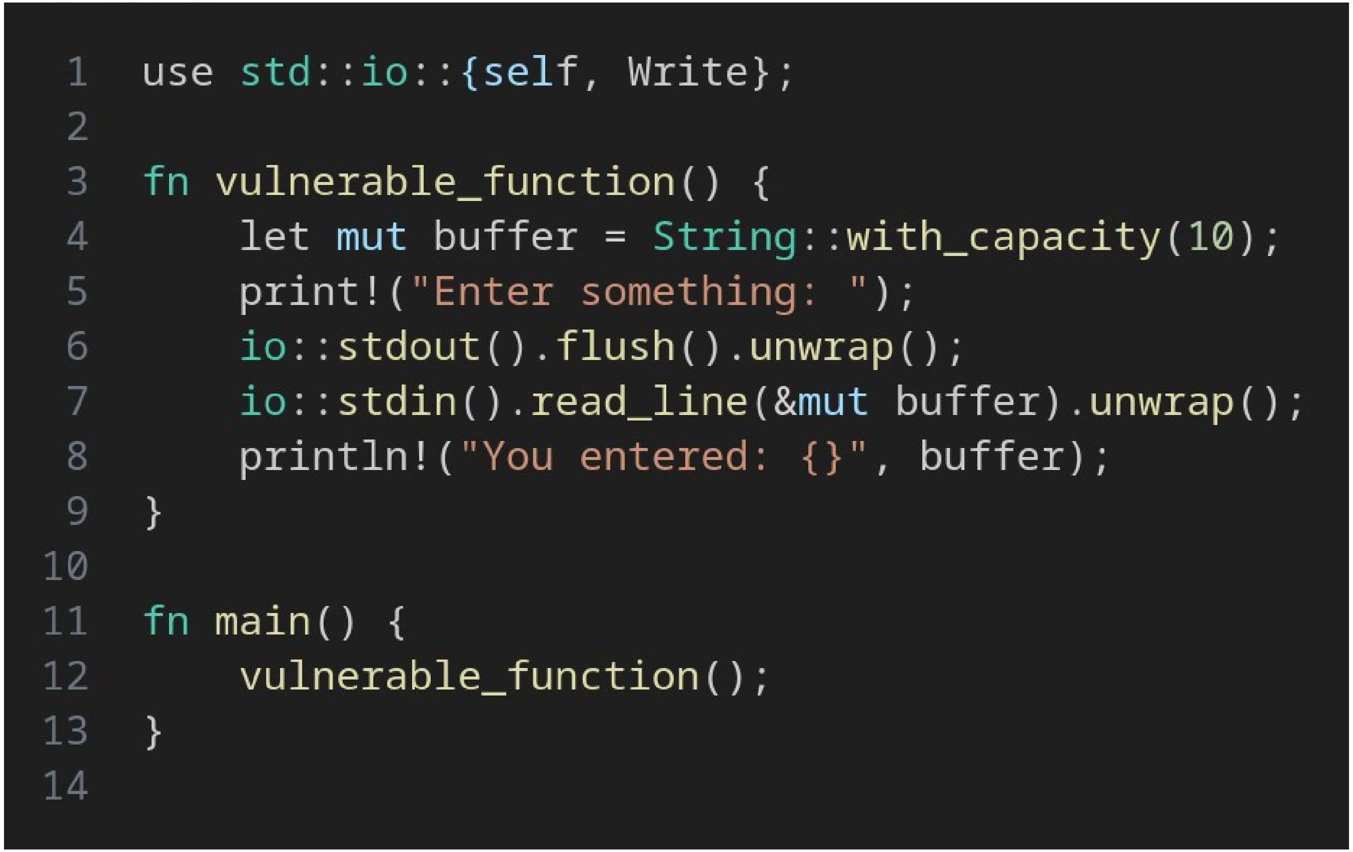


```cpp
#include <iostream>
#include <cstring>
using namespace std;

void hacked() {
    cout << "You got hacked!" << endl;
}

void safe() {
    char buffer[10];
    cout << "Enter some text (max 9 characters): ";
    cin.width(10);
    cin >> buffer;

    cout << "You entered: " << buffer << endl;
}

int main() {
    safe();
    cout << "Back to main!" << endl;
    return 0;
}
```



```cpp
#include <iostream>
#include <memory>
#include <string>
using namespace std;

void tester() {
    cout << "hey you on tester" << endl;
}

void safe() {
    auto buffer = make_unique<string>();
    cout << "Enter some text: ";
    cin >> *buffer;

    cout << "You entered: " << *buffer << endl;
}

int main() {
    safe();
    cout << "Back to main!" << endl;
    return 0;
}
```

Solution on this code for issues with rust



```rust
use std::io::{self, Write};

fn vulnerable_function() {
    let mut buffer = String::with_capacity(10);
    print!("Enter something: ");
    io::stdout().flush().unwrap();
    io::stdin().read_line(&mut buffer).unwrap();
    println!("You entered: {}", buffer);
}

fn main() {
    vulnerable_function();
}

```

Why Linux Should Be Memory Safe

Challenges of Transitioning Linux to Rust

- 1. Compatibility with Existing Code
- 2. Performance Considerations
- 3. Learning Curve
- 4. Community Resistance

If Linux Does Not Transition to Rust

Sources :

- https://www.rust-lang.org/
- https://www.kernel.org/doc/html/latest/
- https://en.wikipedia.org/wiki/Memory_safety
- http://valgrind.org/docs/
- https://duckboard.net/rust_in_linux/?utm_source=chatgpt.com

Thanks for your time