

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

**Presenter : Mateo**



**The ELUG Event**

# About me

**Mateo , only a curious mind**

**Github : <https://github.com/mateo-rfz>**

**Email : [mahdifeyzolahy@gmail.com](mailto:mahdifeyzolahy@gmail.com)**

**Elug Github : <https://github.com/elugiran>**

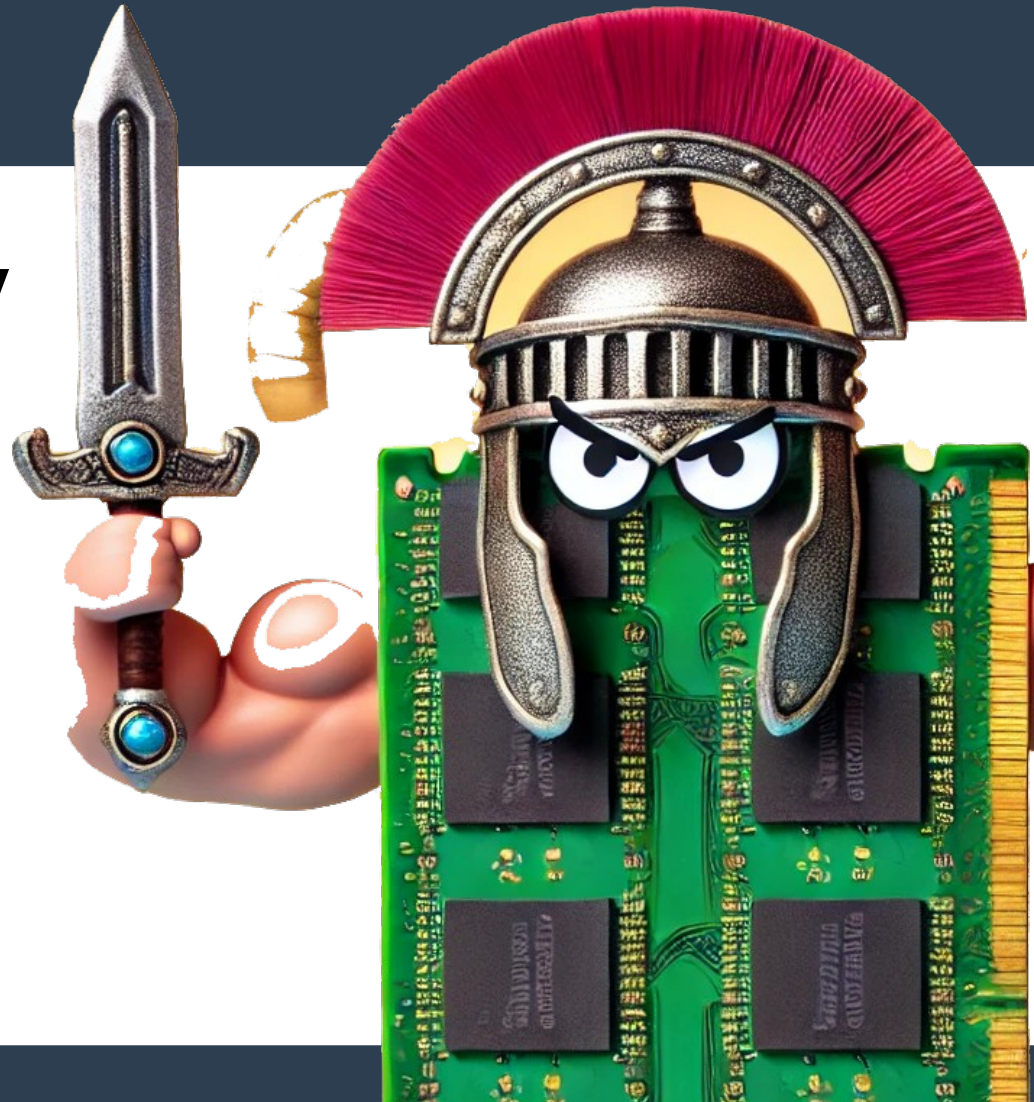


# Memory safe PR



<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

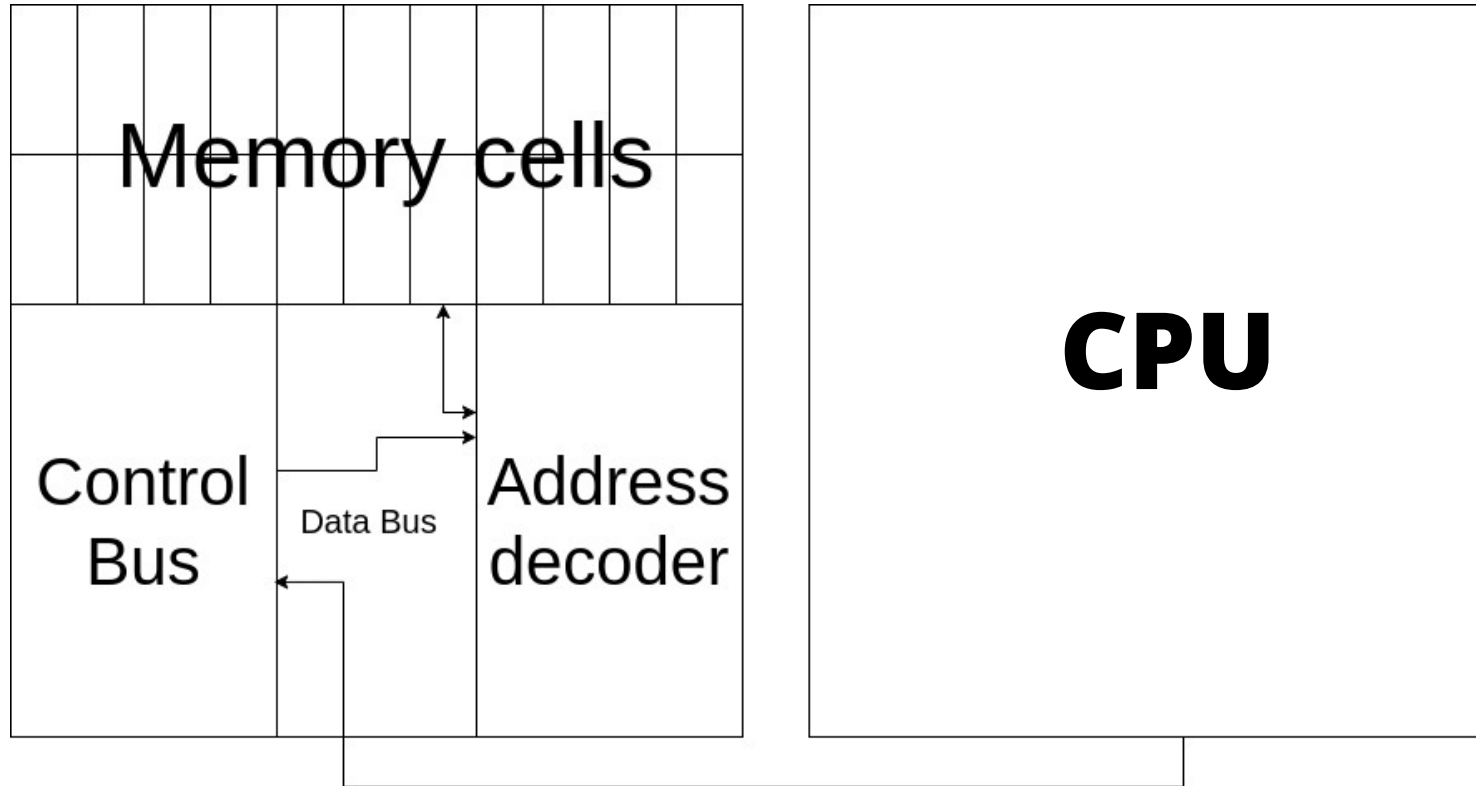
```
#include <iostream>
#include <cstring>
using namespace std;
```

```
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

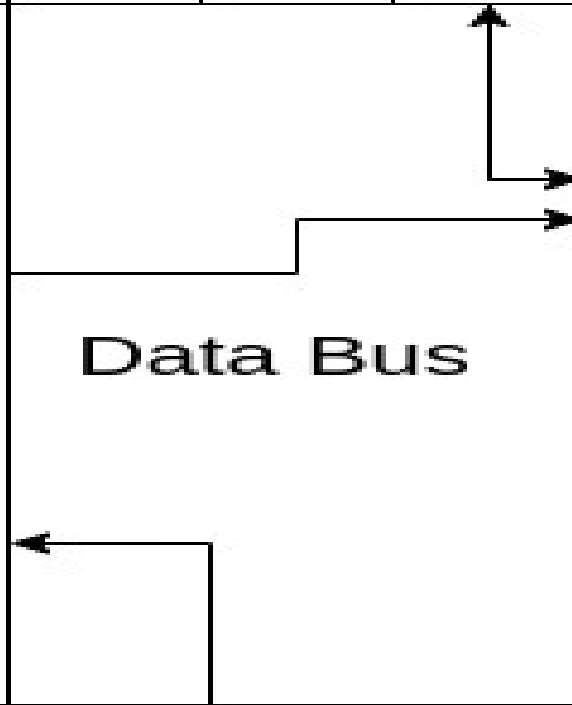


# Memory cells

Control  
Bus

Data Bus

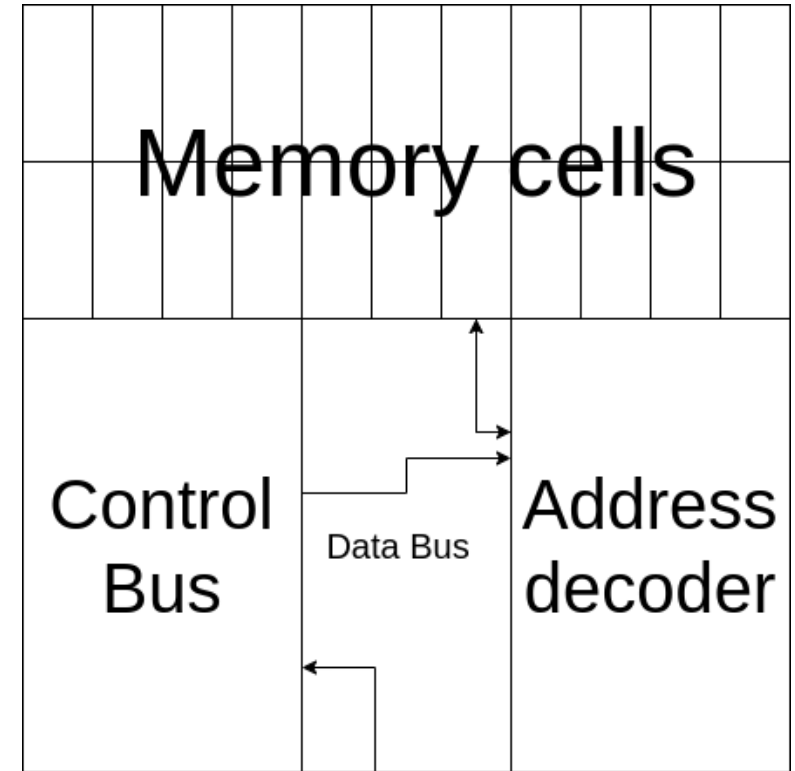
Address  
decoder



# Control Bus

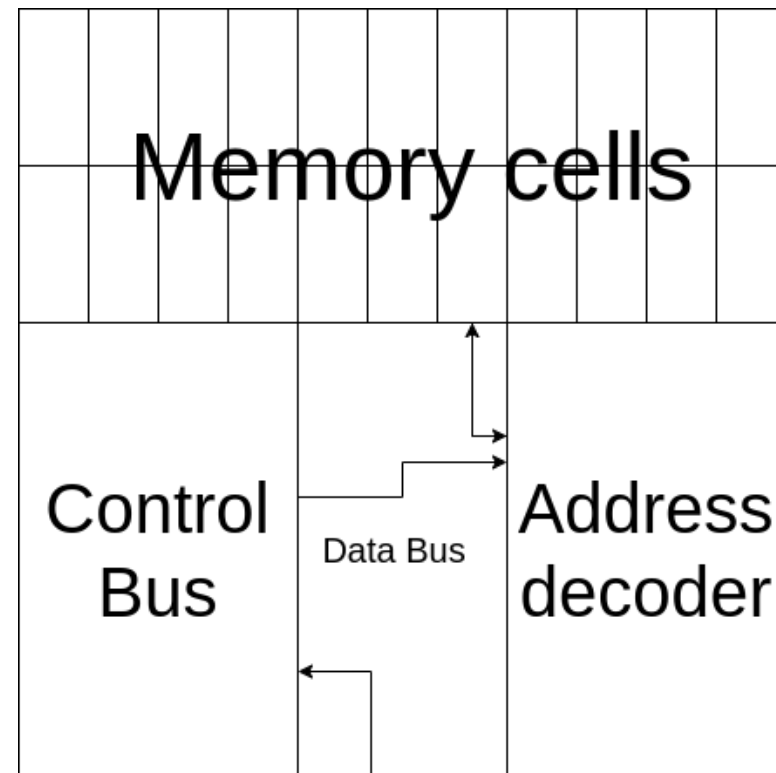
**Wrong command(Read/Write)**

**Ex.Use after Free**



# Address Decoder

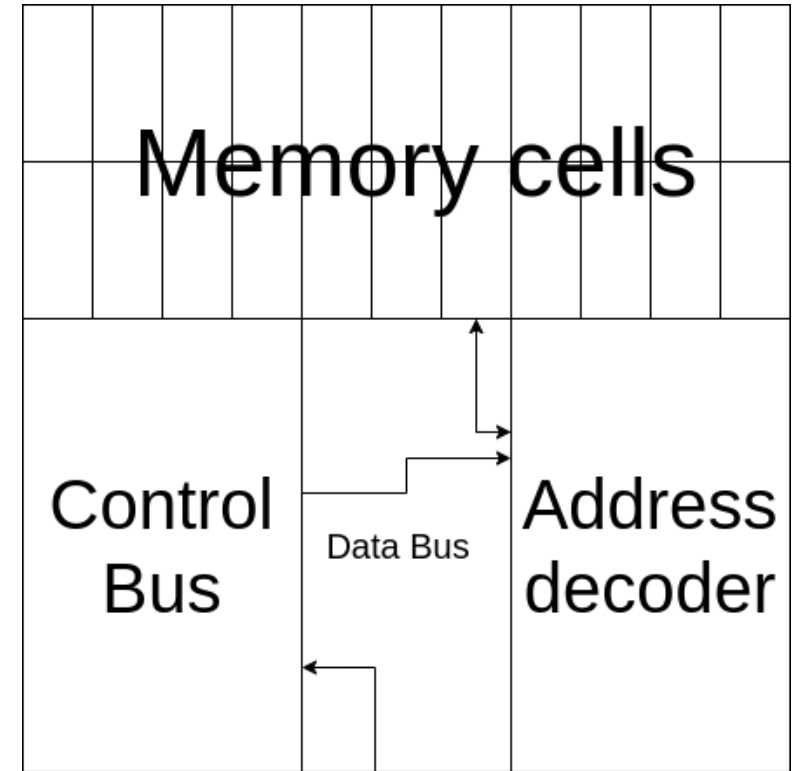
## Out of range Access



# 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
```

```
matee@funlan:~/Desktop$ ./test
Enter some text: TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
You entered: TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
Secret value: 1414812756
Segmentation fault (core dumped)
matee@funlan:~/Desktop$
```

```
mateo@tunlap: ~/Desktop$ ./7.c
Enter some text: Mateo
You entered: Mateo
Secret value: 42
```



# 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
<pre>Enter some text: AAAAAAAAAA... (payload) You got hacked!</pre>	



# 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++

```
1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4
5 void hacked() {
6     cout << "You got hacked!" << endl;
7 }
8
9 void safe() {
10     char buffer[10];
11     cout << "Enter some text (max 9 characters): ";
12     cin.width(10);
13     cin >> buffer;
14
15     cout << "You entered: " << buffer << endl;
16 }
17
18 int main() {
19     safe();
20     cout << "Back to main!" << endl;
21     return 0;
22 }
```

```
1 #include <iostream>
2 #include <memory>
3 #include <string>
4 using namespace std;
5
6 void tester() {
7     cout << "hey you on tester" << endl;
8 }
9
10 void safe() {
11     auto buffer = make_unique<string>();
12     cout << "Enter some text: ";
13     cin >> *buffer;
14
15     cout << "You entered: " << *buffer << endl;
16 }
17
18 int main() {
19     safe();
20     cout << "Back to main!" << endl;
21     return 0;
22 }
23
```

# Solution on this code for issues with rust

```
1 use std::io::{self, Write};
2
3 fn vulnerable_function() {
4     let mut buffer = String::with_capacity(10);
5     print!("Enter something: ");
6     io::stdout().flush().unwrap();
7     io::stdin().read_line(&mut buffer).unwrap();
8     println!("You entered: {}", buffer);
9 }
10
11 fn main() {
12     vulnerable_function();
13 }
14
```

# 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](https://en.wikipedia.org/wiki/Memory_safety)
- <http://valgrind.org/docs/>
- [https://duckboard.net/rust\\_in\\_linux/?utm\\_source=chatgpt.com](https://duckboard.net/rust_in_linux/?utm_source=chatgpt.com)



**Thanks for your time**