

COMP 737011 - Memory Safety and Programming Language Design

Lecture 0: Course Introduction

徐 辉

xuh@fudan.edu.cn



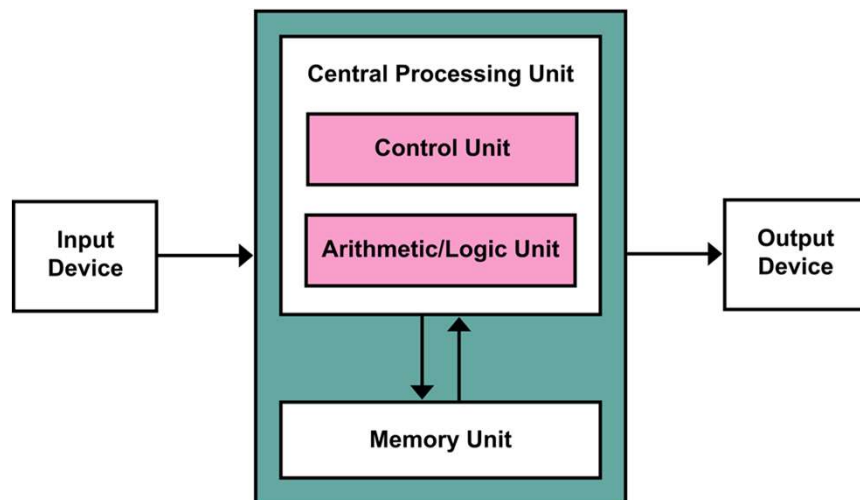
Instructor

- Xu, Hui
 - Ph. D. degree from CUHK
 - Research Interests: program analysis, software reliability
 - Email: xuh@fudan.edu.cn
 - Office: Room D6023, X2 Building, Jiangwan Campus
- Teaching assistants
 - Zhicong Zhang (Head TA)
 - Chengjun Chen
 - Mohan Cui
 - ...

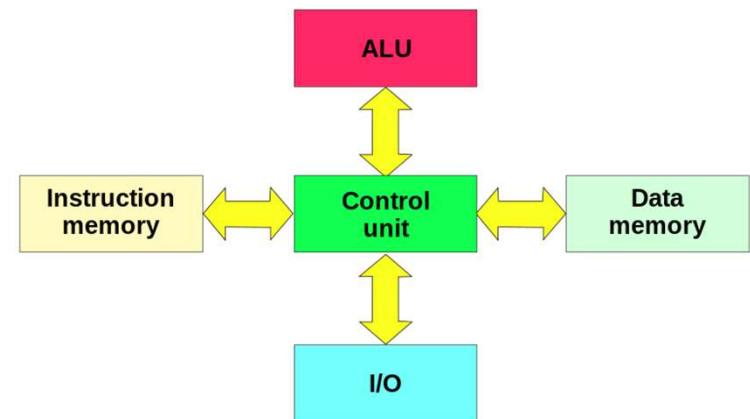


Computer Architecture

- von Neymann Architecture
 - Instructions and data share the same memory unit
 - Widely used
- Harvard Architecture
 - Instructions and data are separated
 - Mainly used in DSP or microcontrollers



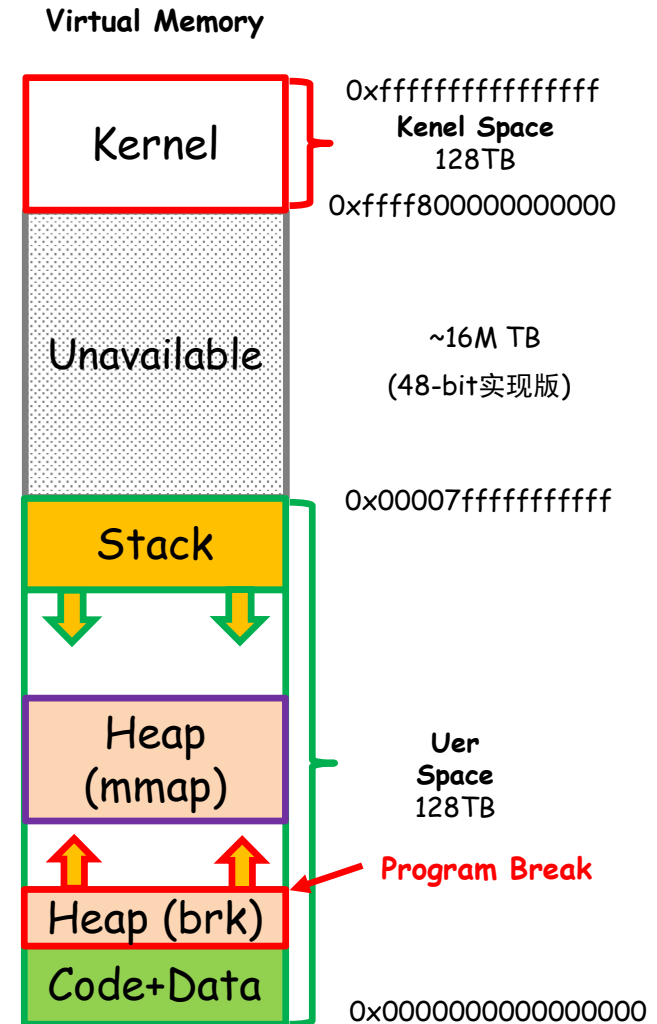
von Neymann Architecture



Harvard Architecture

Operating System

- Multi-process
 - Concurrently executed
 - May or may not share the same address space
 - An OS kernel for global control
 - Syscall for executing kernel code
- Process Memory Layout
 - Kernel space (ring 0)
 - Kernel code + data
 - Kernel runtime
 - User space (ring 3)
 - User code + data
 - User runtime: stack + heap



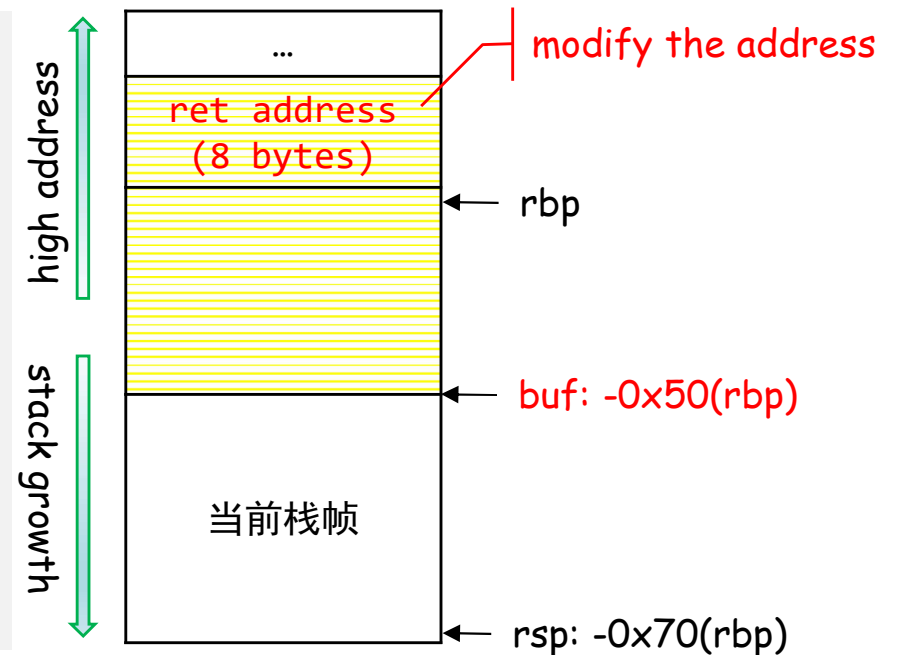
Memory Safety Issues

- Types of bugs:
 - Buffer overflow
 - Dangling pointer
 - Concurrency issue
- Consequence:
 - Data consistency
 - Code integrity
 - Code injection

Buffer Overflow

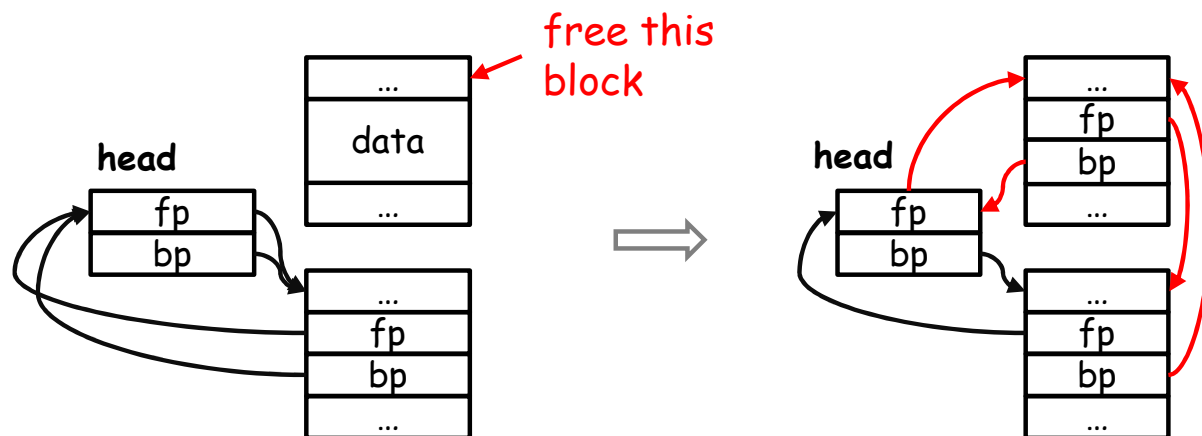
- Write beyond the allocated memory address;
- Can happen on either stack or heap.

```
char buf[64];
read(STDIN_FILENO, buf, 160);
if(strcmp(buf,LICENCE_KEY)==0){
    write(STDOUT_FILENO,
        "Key verified!\n", 14);
}else{
    write(STDOUT_FILENO,
        "Wrong key!\n", 11);
}
```



Dangling Pointer

- Heap are managed with linked lists;
- Effects of free a memory slot on heap via free():
 - The memory is not reclaimed by the OS;
 - The memory is add to a free list;
 - The pointer still points to the address;
- Write to a dangling pointer could breach the list.



Concurrency Issue

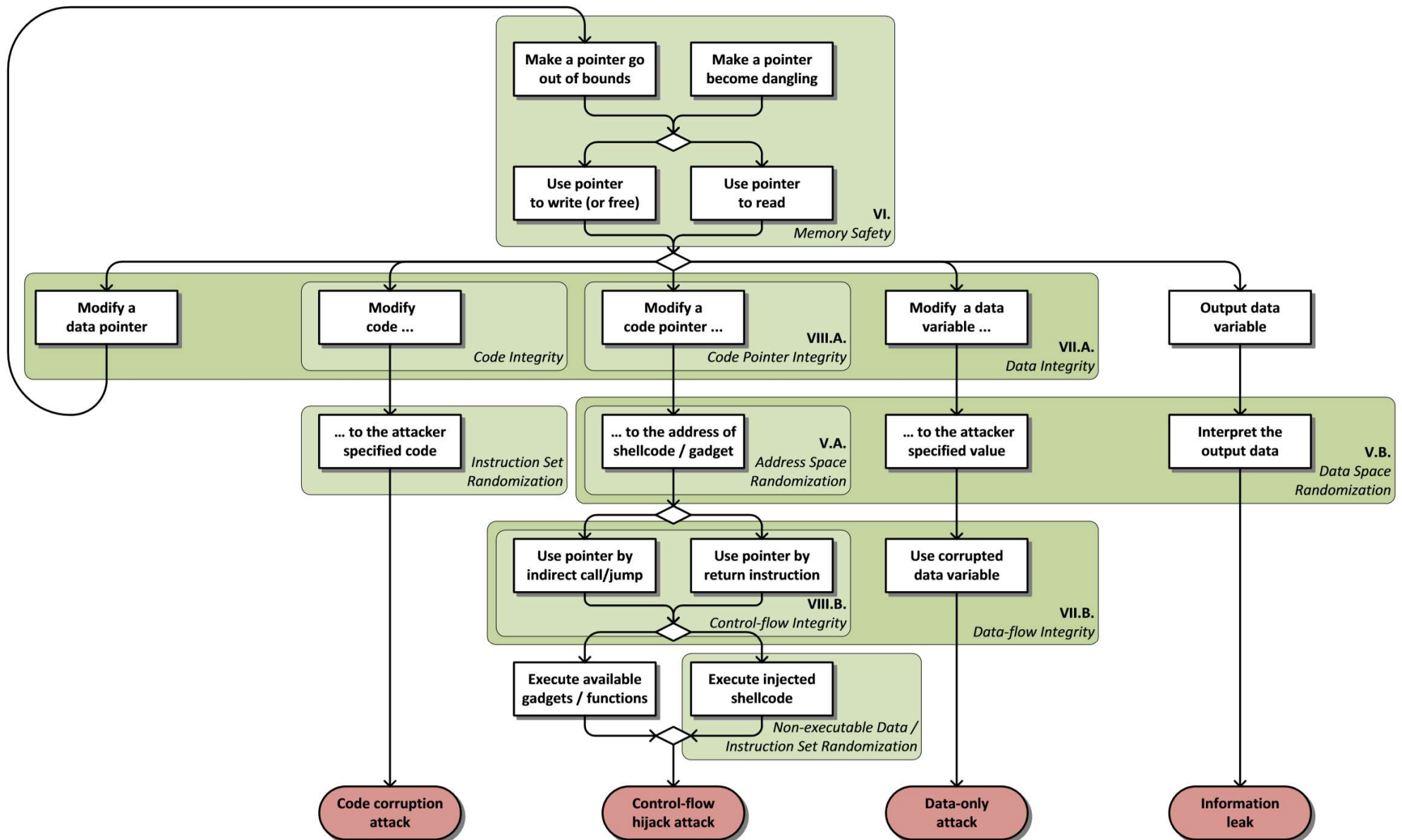
- Race condition or data race
- How to attack the following program?
 - Another thread modifies /tmp/X concurrently
 - Before access(/tmp/X, W_OK), the file /tmp/X is indeed /tmp/X
 - After access(/tmp/X, W_OK), change /tmp/X to /etc/passwd (via symbolic link)

```
if (!access("/tmp/X", W_OK)) {  
    /* the real user ID has access right */  
    f = open("/tmp/X", O_WRITE);  
    write_to_file(f);  
}  
else {  
    /* the real user ID does not have access right */  
    fprintf(stderr, "Permission denied\n");  
}
```


Top 25 Dangerous Software Errors

| Rank | ID | Name | Score | 2020 Rank Change |
|------|-------------------------|--|-------|------------------|
| [1] | CWE-787 | Out-of-bounds Write | 65.93 | +1 |
| [2] | CWE-79 | Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') | 46.84 | -1 |
| [3] | CWE-125 | Out-of-bounds Read | 24.9 | +1 |
| [4] | CWE-20 | Improper Input Validation | 20.47 | -1 |
| [5] | CWE-78 | Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') | 19.55 | +5 |
| [6] | CWE-89 | Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') | 19.54 | 0 |
| [7] | CWE-416 | Use After Free | 16.83 | +1 |
| [8] | CWE-22 | Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') | 14.69 | +4 |
| [9] | CWE-352 | Cross-Site Request Forgery (CSRF) | 14.46 | 0 |
| [10] | CWE-434 | Unrestricted Upload of File with Dangerous Type | 8.45 | +5 |
| [11] | CWE-306 | Missing Authentication for Critical Function | 7.93 | +13 |
| [12] | CWE-190 | Integer Overflow or Wraparound | 7.12 | -1 |
| [13] | CWE-502 | Deserialization of Untrusted Data | 6.71 | +8 |
| [14] | CWE-287 | Improper Authentication | 6.58 | 0 |
| [15] | CWE-476 | NULL Pointer Dereference | 6.54 | -2 |
| [16] | CWE-798 | Use of Hard-coded Credentials | 6.27 | +4 |
| [17] | CWE-119 | Improper Restriction of Operations within the Bounds of a Memory Buffer | 5.84 | -12 |
| [18] | CWE-862 | Missing Authorization | 5.47 | +7 |
| [19] | CWE-276 | Incorrect Default Permissions | 5.09 | +22 |
| [20] | CWE-200 | Exposure of Sensitive Information to an Unauthorized Actor | 4.74 | -13 |
| [21] | CWE-522 | Insufficiently Protected Credentials | 4.21 | -3 |
| [22] | CWE-732 | Incorrect Permission Assignment for Critical Resource | 4.2 | -6 |
| [23] | CWE-611 | Improper Restriction of XML External Entity Reference | 4.02 | -4 |
| [24] | CWE-918 | Server-Side Request Forgery (SSRF) | 3.78 | +3 |
| [25] | CWE-77 | Improper Neutralization of Special Elements used in a Command ('Command Injection') | 3.58 | +6 |

Eternal War in Memory



Methods to Protect Memory Safety

- Developers are human, so errors cannot be avoided
- Preventing bugs by programming language design.
 - Type safety, intelligent pointer, etc.
- Preventing bugs by testing
 - Address sanitizer, fuzz, symbolic execution, etc
- Preventing attack during runtime
 - Stack canary, shadow stack, etc.

Availability Issue

- Types of bugs:
 - Stack overflow
 - Heap exhaustion
 - Memory leakage
- Consequence:
 - Unexpected termination
 - May not be easy to recover
- This course also considers availability issues because it is closely related to memory safety

Rust Language for Memory Safety

- A system programming language focusing on:
 - Memory safety
 - Concurrency safety
 - Efficiency
- Timeline
 - Personal project started in 2006 by Mozilla employee Graydon Hoare.
 - Self-hosting since 2011
 - First stable version was released in 2015



Why Rust?

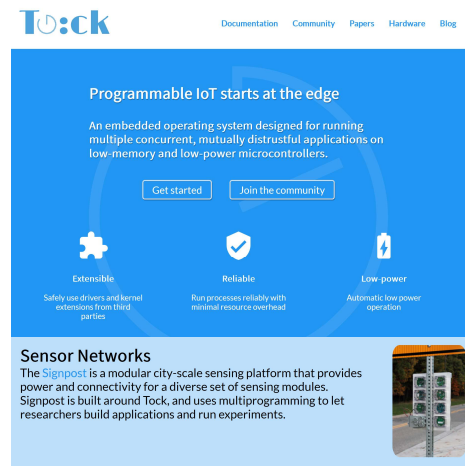
- State-of-the-art language for memory safety
- Most favorable according to stackoverflow
- Many large companies turn to Rust

Using Rust in Windows

Security Research & Defense / By MSRC Team / November 7, 2019 / Memory Safety, Rust, Safe Systems Programming Languages, Secure Development

This Saturday 9th of November, there will be a [keynote](#) from Microsoft engineers Ryan Levick and Sebastian Fernandez at RustFest Barcelona. They will be talking about why Microsoft is exploring Rust adoption, some of the challenges we've faced in this process, and the future of Rust adoption in Microsoft. If you want to talk with some of the people working on how Microsoft is evolving its code practices for better security, be sure to attend the keynote and talk to Ryan and Sebastian afterwards!

This blog describes part of the story of Rust adoption at Microsoft. Recently, I've been tasked with an experimental rewrite of a low-level system component of the Windows codebase (sorry, we can't say which one yet). Instead of rewriting the code in C++, I was asked to use Rust, a memory-safe alternative. Though the project is not yet finished, I can say that my experience with Rust has been generally positive. It's a good choice for those looking to avoid common mistakes that often lead to security vulnerabilities in C++ code bases.



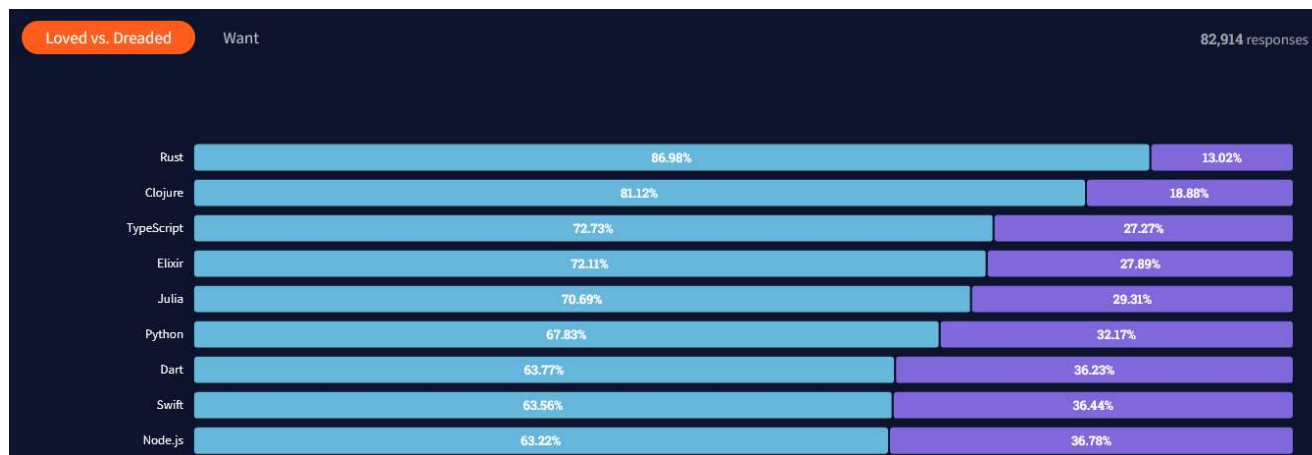
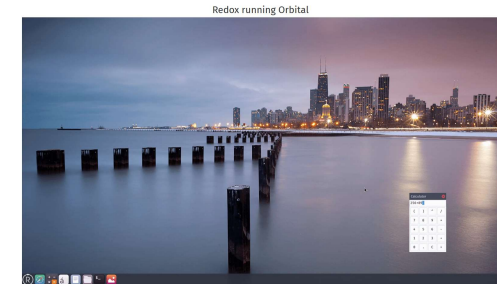
The screenshot shows the Tock website, which is an embedded operating system for IoT. The header includes links for Documentation, Community, Papers, Hardware, and Blog. The main content area features the title 'Programmable IoT starts at the edge' and a description: 'An embedded operating system designed for running multiple concurrent, mutually distrustful applications on low-memory and low-power microcontrollers.' Below this are two buttons: 'Get started' and 'Join the community'. The page is divided into three columns: 'Extensible' (Safety use drivers and kernel extensions from third parties), 'Reliable' (Run processes reliably with minimal resource overhead), and 'Low power' (Automatic low power operation). At the bottom, there is a section for 'Sensor Networks' with a description of the Signpost platform and a small image of a sensor node.

Redox is a Unix-like Operating System written in Rust, aiming to bring the innovations of Rust to a modern microkernel and full set of applications.

[View Releases](#)

[Pull from GitLab](#)

- Implemented in Rust
- Microkernel Design
- Includes optional GUI - Orbital
- Supports Rust Standard Library
- MIT Licensed
- Drivers run in Userspace
- Includes common Unix commands
- Custom libc written in Rust (relibc)



Key Idea of Rust

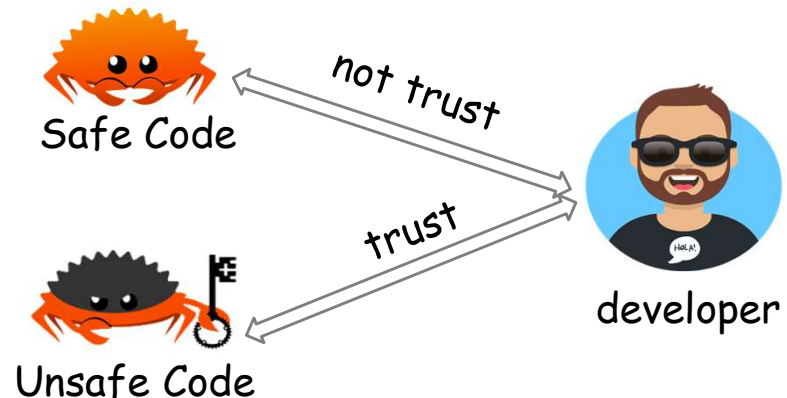
- Security zones of Rust Code:
 - unsafe code: dereference raw pointer, FFI, etc
 - safe code: without undefined behaviors
- Developers should
 - avoid using unsafe code as best as they can
 - wrap unsafe code into safe APIs so that they can use safe APIs instead

```
let mut num = 5;  
let r1 = &num as *const i32;  
let r2 = &mut num as *mut i32;  
unsafe {  
    println!("r1 is: {}", *r1);  
    println!("r2 is: {}", *r2);  
}
```

Dereference raw pointers

```
unsafe fn dangerous() {  
    let address = 0x012345usize;  
    let r = address as *const i32;  
}  
unsafe {  
    dangerous();  
}
```

Call unsafe functions



Objective of This Course

- Practice the ability in research and solving problems.
- After this course, the student shall
 - understand the issues related to memory safety and have the ability to demonstrate them;
 - know some basic ideas and tools for solving memory safety problems;
 - understand the advanced features of Rust.

Schedule

| Week | Subject | | In-class Practice |
|------|-----------------------------------|--|-------------------|
| 1 | Problems related to Memory-Safety | Buffer Overflow | Attack Experiment |
| 2 | | Memory Allocator | Coding Practice |
| 3 | | Memory Exhaustion | Attack Experiment |
| 4 | | Dangling Pointers | Attack Experiment |
| 5 | | Concurrent Memory Access | Coding Practice |
| 6 | Rust Programming Language | Rust Ownership-based Memory Management | Coding Practice |
| 7 | | Rust Generics and Traits | Coding Practice |
| 8 | | Rust Concurrency Programming | Coding Practice |
| 9 | | Rust Compiler Theory | Tool Experiment |
| 10 | | Rust Compiler Techniques | Tool Experiment |
| 11 | Advanced Topic for Memory Safety | Effectiveness of Rust | Discussion |
| 12 | | Testing and Fuzzing | Tool Experiment |
| 13 | | Address Sanitizer | Tool Experiment |
| 14 | | Static Program Analysis | Tool Experiment |
| 15 | | Symbolic Execution | Tool Experiment |
| 16 | | Isolation of Unsafe Code | Discussion |
| 17 | | Code Search and Recommendation | Coding Practice |
| 18 | Course Exam | Project Report | |

Grading

- In-class Practice: Max 60%
 - 10-15 experiments
 - 6% for each experiment
 - You may get all the marks by doing 10 experiments
 - Submit simple experiment reports on elearning
 - Due: T+1 week
- Discussion: 10%
 - Two classes
- Project: 30%
 - 20min presentation
 - one paper or multiple papers
 - PPT file is required for submission

Important Policy

- Plagiarism or cheating will not be tolerated
 - You cannot copy any sentence or paragraph
 - Rephrase it or *"quote it"*
- Hard due date of assignments