COMP 737011 - Memory Safety and Programming Language Design

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
  - Mohan Cui
  - Yan Dong

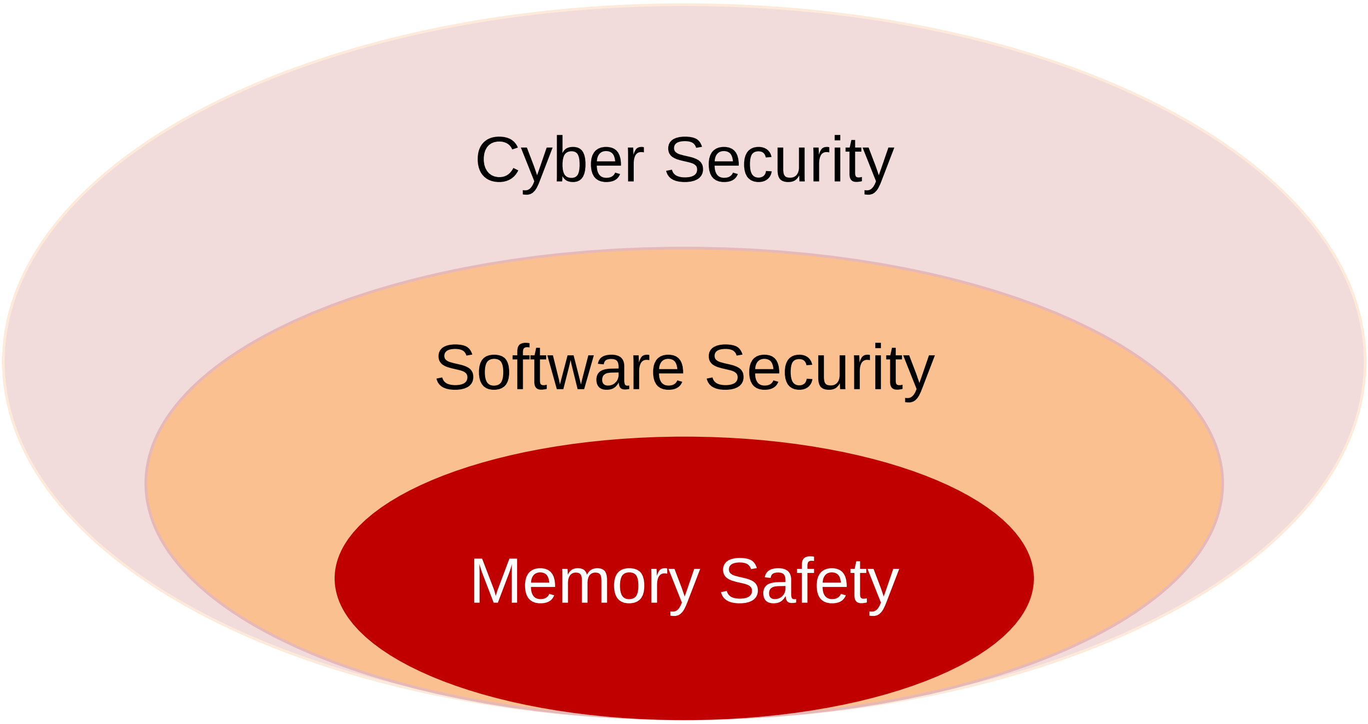
Zhicong Zhang

Mohan Cui

Yan Dong

Room D6010, X2 Building

Understand Memory-Safety Problems

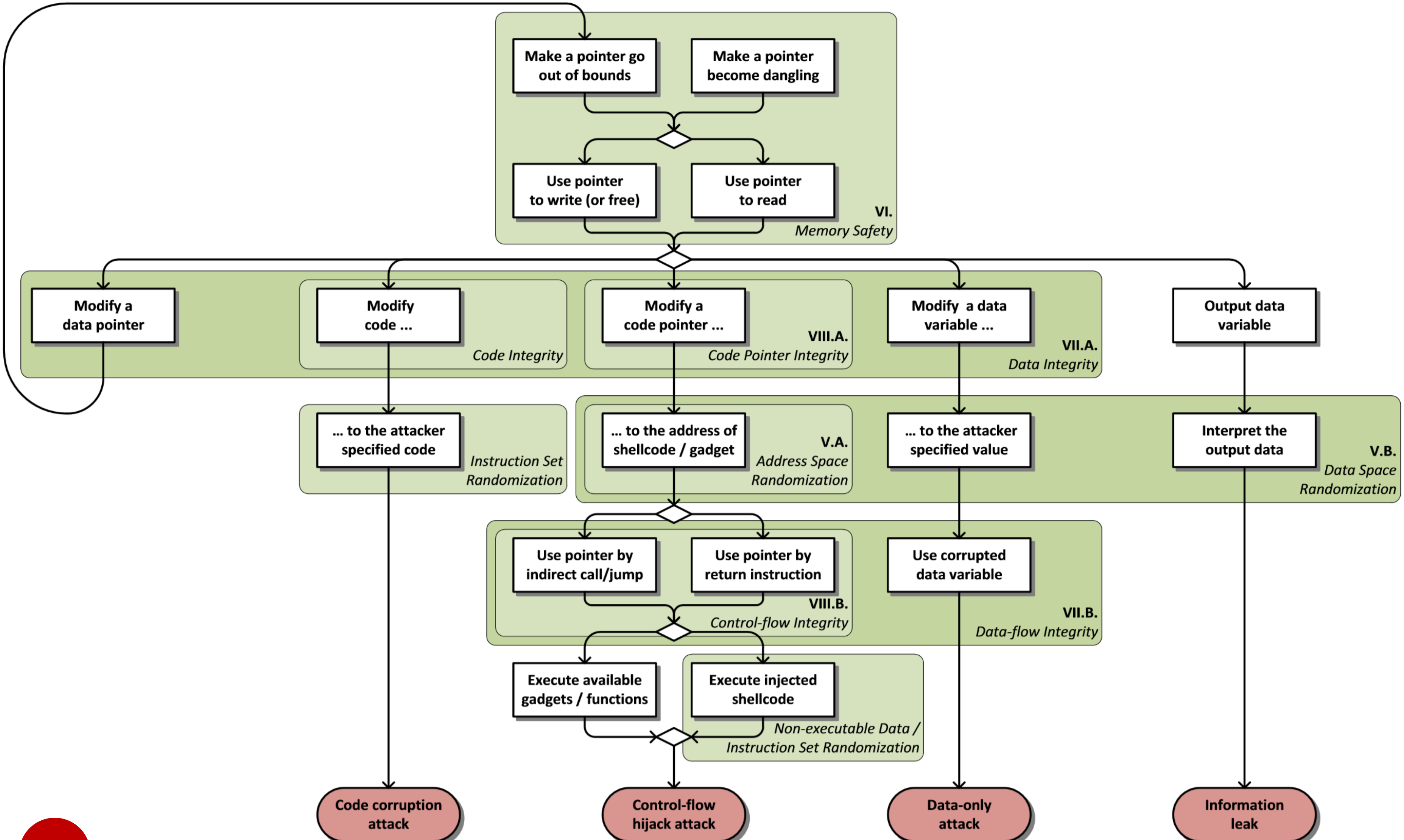


Top 25 Dangerous Software Errors

Rank	ID	Name	Score	KEV Count (CVEs)	Rank Change vs. 2021
1	CWE-787	Out-of-bounds Write	64.20	62	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.97	2	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	22.11	7	+3 ▲
4	CWE-20	Improper Input Validation	20.63	20	0
5	CWE-125	Out-of-bounds Read	17.67	1	-2 ▼
6	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	17.53	32	-1 ▼
7	CWE-416	Use After Free	15.50	28	0
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.08	19	0
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.53	1	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	9.56	6	0
11	CWE-476	NULL Pointer Dereference	7.15	0	+4 ▲
12	CWE-502	Deserialization of Untrusted Data	6.68	7	+1 ▲
13	CWE-190	Integer Overflow or Wraparound	6.53	2	-1 ▼
14	CWE-287	Improper Authentication	6.35	4	0
15	CWE-798	Use of Hard-coded Credentials	5.66	0	+1 ▲
16	CWE-862	Missing Authorization	5.53	1	+2 ▲
17	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	5.42	5	+8 ▲
18	CWE-306	Missing Authentication for Critical Function	5.15	6	-7 ▼
19	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	4.85	6	-2 ▼
20	CWE-276	Incorrect Default Permissions	4.84	0	-1 ▼
21	CWE-918	Server-Side Request Forgery (SSRF)	4.27	8	+3 ▲
22	CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	3.57	6	+11 ▲
23	CWE-400	Uncontrolled Resource Consumption	3.56	2	+4 ▲
[illegible]	CWE-611	Improper Restriction of XML External Entity Reference	3.38	0	-1 ▼
[illegible]	CWE-94	Improper Control of Generation of Code ('Code Injection')	3.32	4	+3 ▲

https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html

Eternal War in Memory



Laszlo Szekeres, *et al.* “Sok: Eternal war in memory.” *IEEE Symposium on Security and Privacy*, 2013.

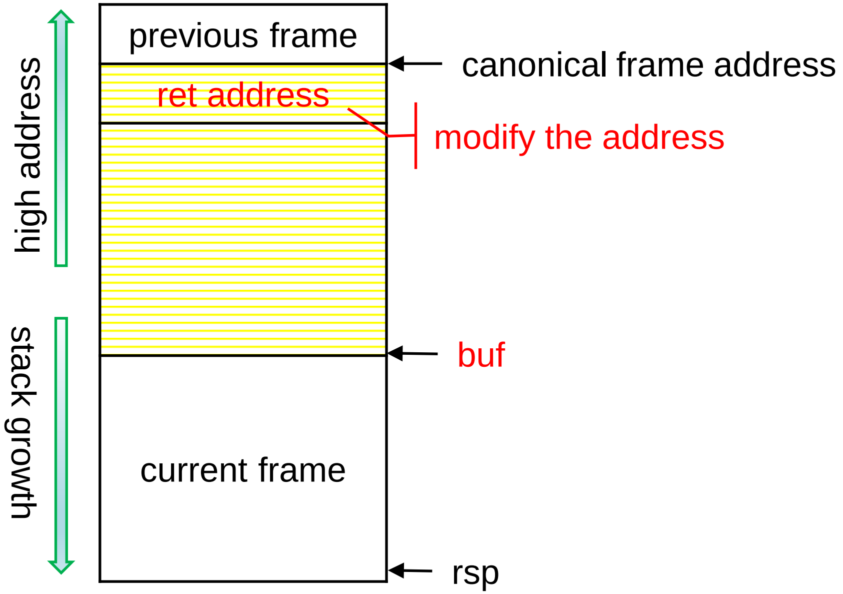
Memory Safety Issues

- Types of bugs:
  - Out-of-bound read
  - Out-of-bound write
    - stack smashing
    - heap overflow
  - Dangling pointer
    - use-after-free
    - double free
  - Concurrency issue
- Consequence:
  - Data leakage
  - Data integrity
  - Code integrity
  - Control-flow integrity
  - ...

Out-of-Bound Write

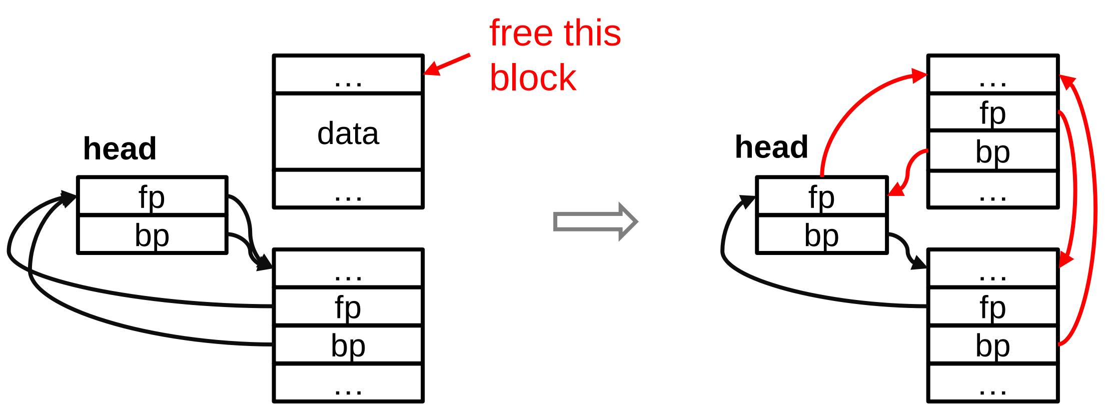
- Write beyond the allocated memory address
- Can happen either on stack or heap

```
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

- Memory blocks on heap are managed with linked lists
- Effects of freeing a memory block via free()
  - The block is added to a free list
  - The pointer still points to the address
- Writing to a dangling pointer could breach the list



Concurrency Issue

- Non-atomic code is vulnerable to race condition

```
void *inc(void *in) {
    int t = *(int *) in;
    sleep(1);
    *(int *) in = t+1;
}
```

```
void *dec(void *in) {
    int t = *(int *) in;
    sleep(1);
    *(int *) in = t-1;
}
```

```
int main(int argc, char** argv) {
    int x = 10;
    pthread_t tid[2];
    pthread_create(&tid[0], NULL, inc, (void *) &x);
    pthread_create(&tid[1], NULL, dec, (void *) &x);
    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);
    assert(x, 10);
}
```

More: Availability Issue

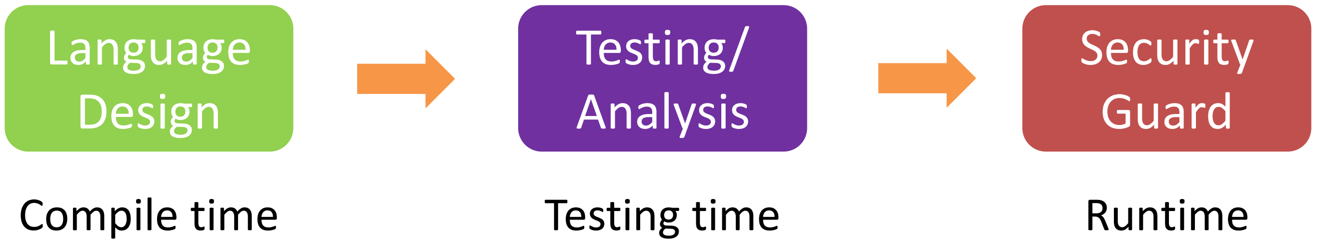
- This course also considers availability issues because it is closely related to memory safety

- Types of bugs:
  - Stack overflow
  - Heap exhaustion
  - Memory leakage

- Consequence:
  - Unexpected termination
  - Bad program state
  - May not be easy to recover

Methods to Protect Memory Safety

- We cannot trust developers
  - Developers are human, so errors cannot be avoided.
- Preventing bugs by programming language design
  - Type safety, smart pointer, *etc*.
- Preventing bugs by testing and program analysis
  - Address sanitizer, fuzz, symbolic execution, etc
- Preventing attacks via runtime security guard
  - Stack canary, shadow stack, *etc*.



Rust Language for Memory Safety

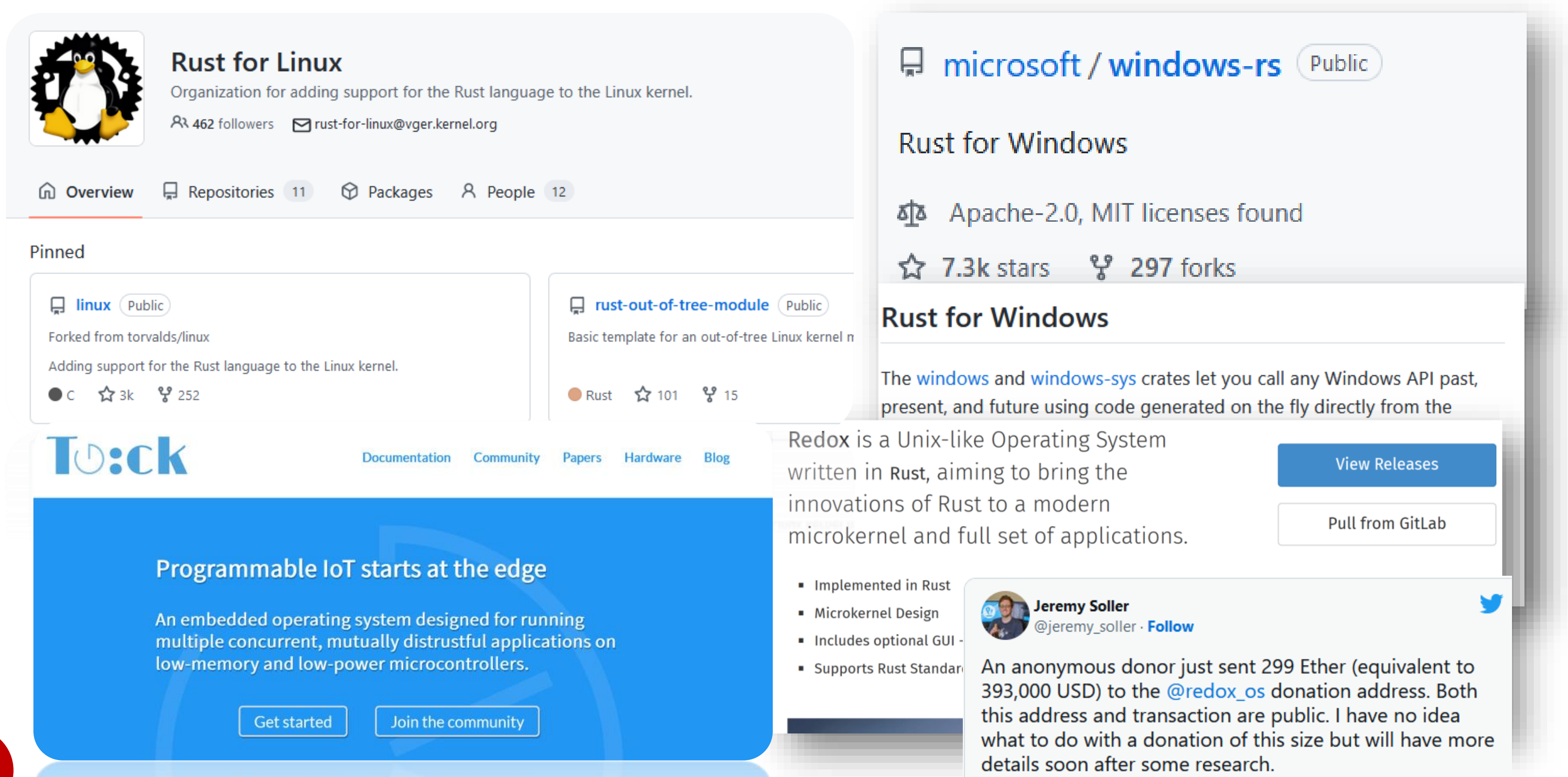
- Rust is a system programming language:
  - to prevent critical bugs via language design (memory safe)
  - while still offering adequate control flexibility (efficiency)

Mozilla员工 Graydon Hoare 的私人项目	Self-hosting	First stable release	Mozilla裁员 Servo团队	Rust Foundation 成立
2006年	2011年	2015年	2020年	2021年

AWS, Huawei, Google,
Microsoft, Mozilla…

Why Rust?

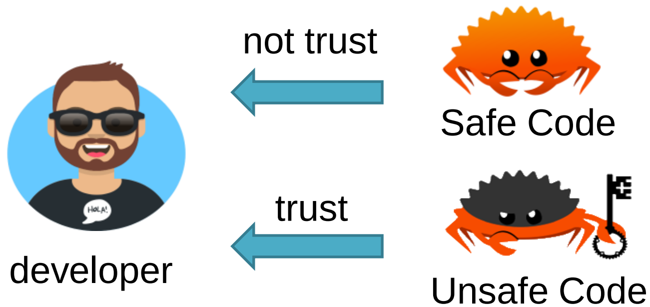
- State-of-the-art language for memory safety
- Most favorable according to stackoverflow
- Many companies and projects turn to Rust



https://insights.stackoverflow.com/survey/2022

Key Idea of Rust

- Interior safety:
  - wrap unsafe code into safe APIs
  - avoid using unsafe code directly



no undefined behaviors

```
struct List{
    val: u64,
    next: *mut List,
    prev: *mut List,
}
let l = List{...}; //construct a list
unsafe {
    *(l.next);
}
```

Dereference raw pointers

Objective of This Course

- After this course, the student shall know
  - the issues related to memory safety
  - some basic ideas and tools for memory safety protection
  - features of Rust
- Practice research and problem solving skills.

Tentative Schedule

Week	Subject		In-class Practice	Assignment
1	Foundations of Memory Safety	Stack Smash	Attack Experiment	
2		Memory Allocator	Coding Practice	
3		Heap Attack	Attack Experiment	
4		Auto Memory Management	Coding Practice	
5		Memory Exhaustion	Experiment	
6		Concurrent Access	Coding Practice	
7	Rust Programming Language	Rust OBRM	Coding Practice	1
8		Rust Type System	Coding Practice	
9		Rust Concurrent Programming	Coding Practice	
10		Rust Compiler and Review	Experiment	2
11		Guest Lecture	Discussion	
12	Advanced Topics	Static Program Analysis	Tool Experiment	
13		Testing and Fuzzing	Tool Experiment	
14		Formal Verification	Tool Experiment	
15		More Techniques		
16	Course Exam	Project Report		

Grading

- In-class practice: 30%
  - A report with at least three experiments
  - Due: week 15
- Two assignments: 30%
  - Case study related to Rust
  - Submit on elearning
  - Due: T+3 week
- Project report: 40%
  - 10 - 20min presentation
    - a research idea/one paper/multiple papers
  - PPT file is required for submission

Notice

- Plagiarism or cheating will not be tolerated
  - You cannot copy any sentence or paragraph
  - Rephrase it or *“quote it”*
  - You may use ChatGPT as an assistant tool
- Hard due date of assignments