COMP 737011 - Memory Safety and Programming Language Design

Course Introduction

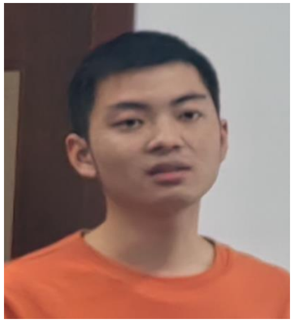
Hui Xu

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
  - Zihao Rao
  - Chenhao Cui

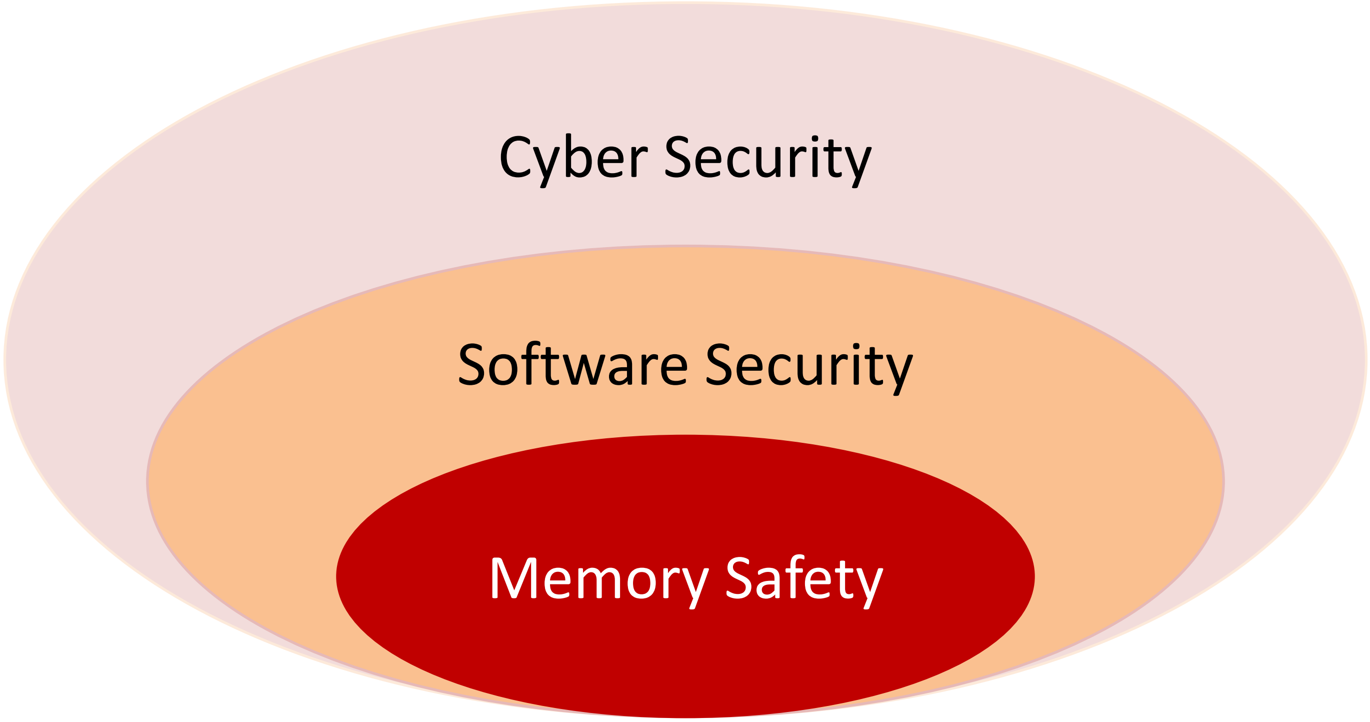
Zihao Rao



Chenhao Cui

Room E4006, X2 Building

Understand Memory-Safety Problems

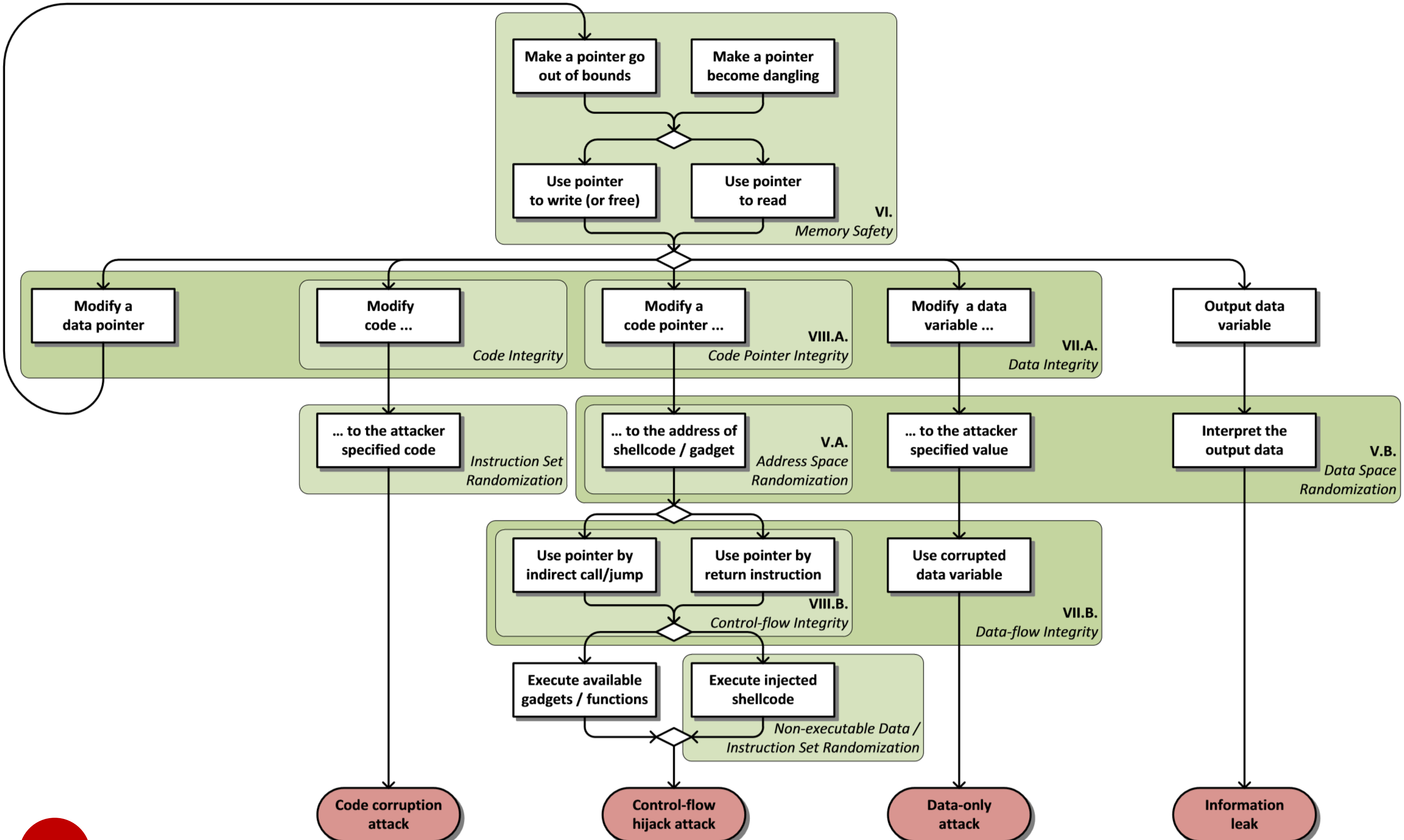


Top 25 Dangerous Software Errors

Rank	ID	Name	Score	CVEs in KEV	Rank Change vs. 2023
1	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	56.92	3	+1
2	CWE-787	Out-of-bounds Write	45.20	18	-1
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	35.88	4	0
4	CWE-352	Cross-Site Request Forgery (CSRF)	19.57	0	+5
5	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	12.74	4	+3
6	CWE-125	Out-of-bounds Read	11.42	3	+1
7	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	11.30	5	-2
8	CWE-416	Use After Free	10.19	5	-4
9	CWE-862	Missing Authorization	10.11	0	+2
10	CWE-434	Unrestricted Upload of File with Dangerous Type	10.03	0	0
11	CWE-94	Improper Control of Generation of Code ('Code Injection')	7.13	7	+12
12	CWE-20	Improper Input Validation	6.78	1	-6
13	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	6.74	4	+3
14	CWE-287	Improper Authentication	5.94	4	-1
15	CWE-269	Improper Privilege Management	5.22	0	+7
16	CWE-502	Deserialization of Untrusted Data	5.07	5	-1
17	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	5.07	0	+13
18	CWE-863	Incorrect Authorization	4.05	2	+6
19	CWE-918	Server-Side Request Forgery (SSRF)	4.05	2	0
20	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	3.69	2	-3
21	CWE-476	NULL Pointer Dereference	3.58	0	-9
22	CWE-798	Use of Hard-coded Credentials	3.46	2	-4
23	CWE-190	Integer Overflow or Wraparound	3.37	3	-9
24	CWE-400	Uncontrolled Resource Consumption	3.23	0	+13
25	CWE-306	Missing Authentication for Critical Function	2.73	5	-5

https://cwe.mitre.org/top25/archive/2024/2024_cwe_top25.html

Eternal War in Memory



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

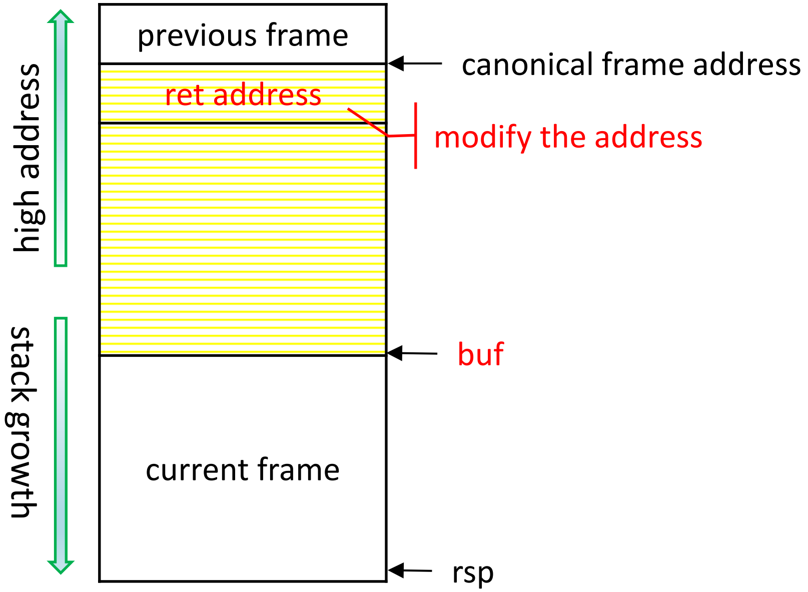
Memory Safety Issues

- 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

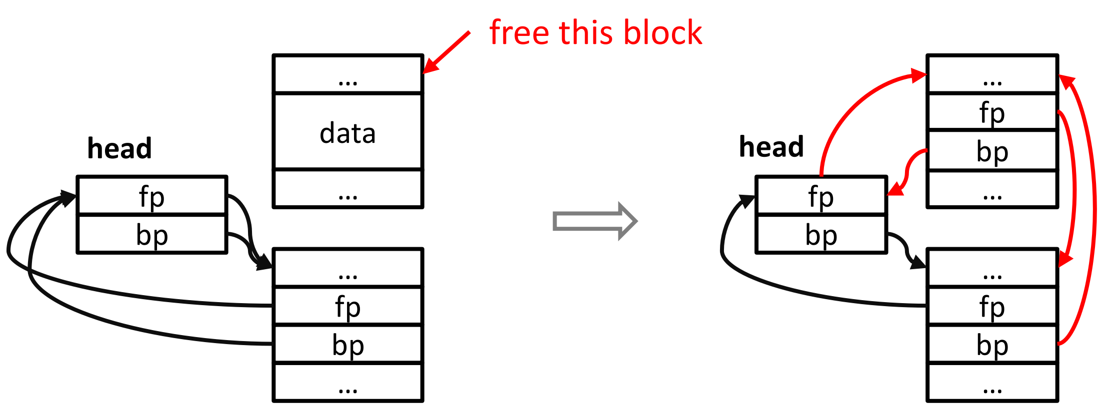
- Write beyond the allocated memory address.
- The issue can happen either on stack or heap.

```
char buf[64];
read(STDIN, buf, 160);
if(strcmp(buf,LICENCE_KEY)==0){
    write(STDOUT, "Verified!\n", 10);
}else{
    write(STDOUT, "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

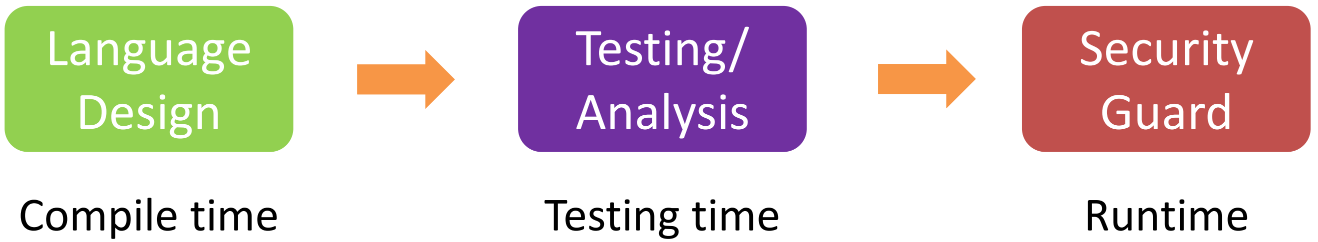
- This course also considers availability issues because it is closely related to memory safety.

- 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
  - Static 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:
  - Prevent critical bugs via language design (memory safe),
  - While still offering adequate control flexibility (efficiency).

Stable Version

Rust Foundation

Rust for Linux/Windows

Linux 6.0

NSA (US) with CAN/UK/AUS…


The Case for Memory Safe Roadmaps

Why Both C-Suite Executives and Technical Experts Need to Take Memory Safe Coding Seriously

White House


BACK TO THE BUILDING BLOCKS:

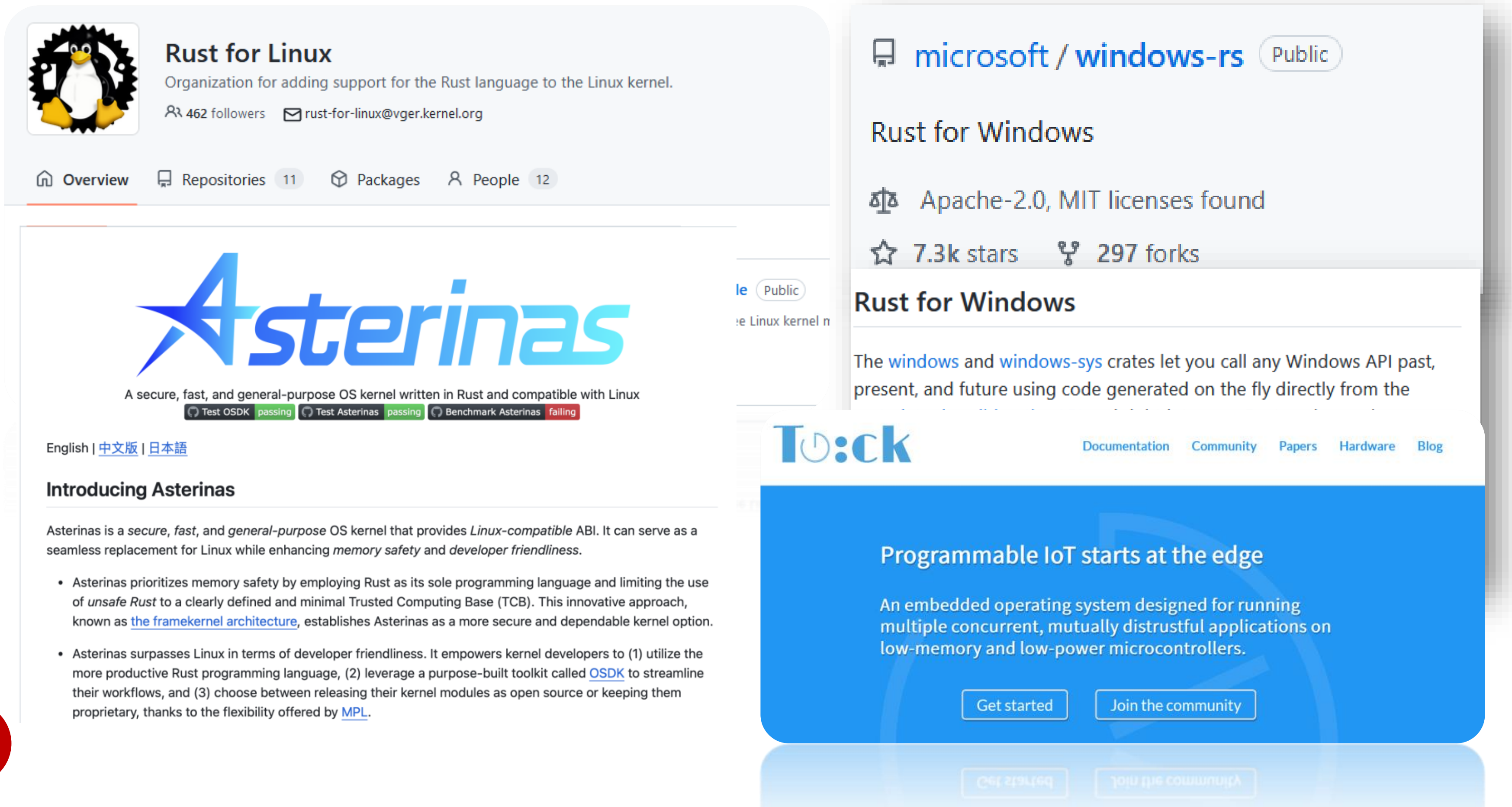
A PATH TOWARD SECURE AND MEASURABLE SOFTWARE

FEBRUARY 2024

2006 2015 2021 2022 2023 2024

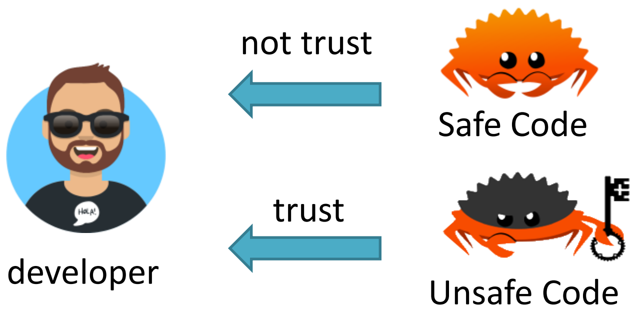
Why Rust?

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



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;
  - Key language features of Rust;
- Practice research and problem solving skills

Tentative Schedule

Week	Date	Subject		In-class Exercise
1	Feb-18	Foundations of Memory Safety	Stack Smash	Attack Experiment
2	Feb-25		Memory Allocator	Coding Practice
3	Mar-4		Heap Attack	Attack Experiment
4	Mar-11		Memory Exhaustion	Coding Practice
5	Mar-18		Concurrent Access	Experiment
6	Mar-25	Rust Programming Language	OBRM	Coding Practice
7	Apr-1		Type System	Coding Practice
8	Apr-8		Concurrency Programming	Coding Practice
9	Apr-15		Functional Programming	Coding Practice
10	Apr-22		Rust Compiler	Experiment
11	Apr-29	Advanced Topics	Pointer Analysis	Open Topic/ Tool Experiment
12	May-6		Dataflow Analysis	
13	May-13		Control Analysis	
14	May-20		Concurrency Analysis	
15	May-27	Project Report	I: Open Topic	
16	Jun-3		II: Tool Development	

Grading

- In-class practice: 50%
  - Three exercise reports in Part I: Foundations of Memory Safety (30%)
  - Coding practice in Part II: Rust Programming Language (20%)
  - Due: week 15 (May-27)
- Project: 50%
  - Cargo plugin: develop a tool for Rust program analysis
  - Open topic related to this course:
    - Choice I. Develop a language tool
    - Choice II. A research idea/survey of multiple papers
  - 20 min presentation
    - PPT file (and source code) is required for submission

Notice

- Plagiarism or cheating will not be tolerated:
  - You cannot copy any sentence or paragraph;
  - Rephrase or “quote”;
  - Using GPT as an assistant tool is allowed.
- Hard due date of assignments