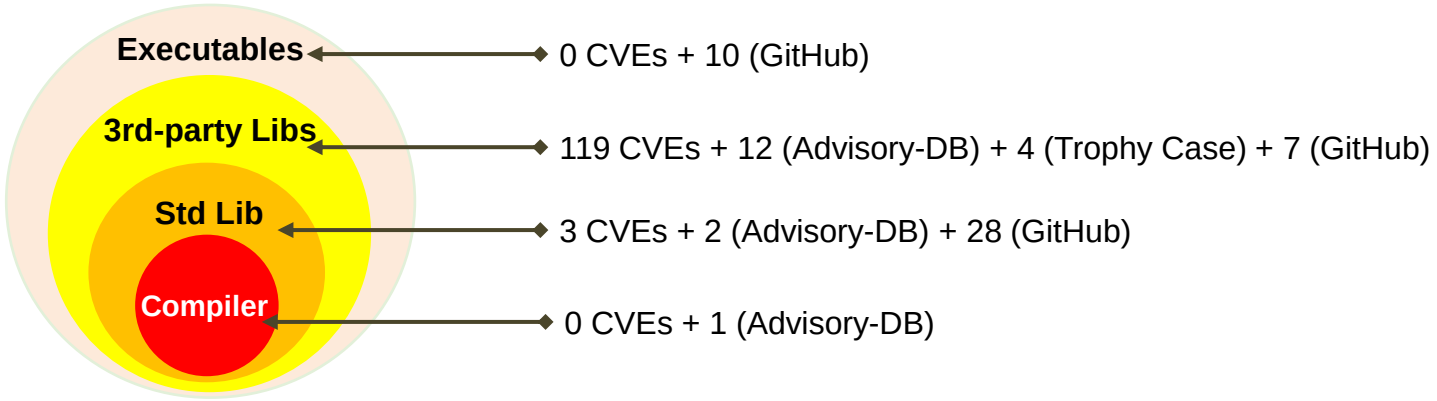
COMP 737011 - Memory Safety and Programming Language Design

Lecture 10: Rust Bugs

徐 辉
xuh@fudan.edu.cn

Case Study

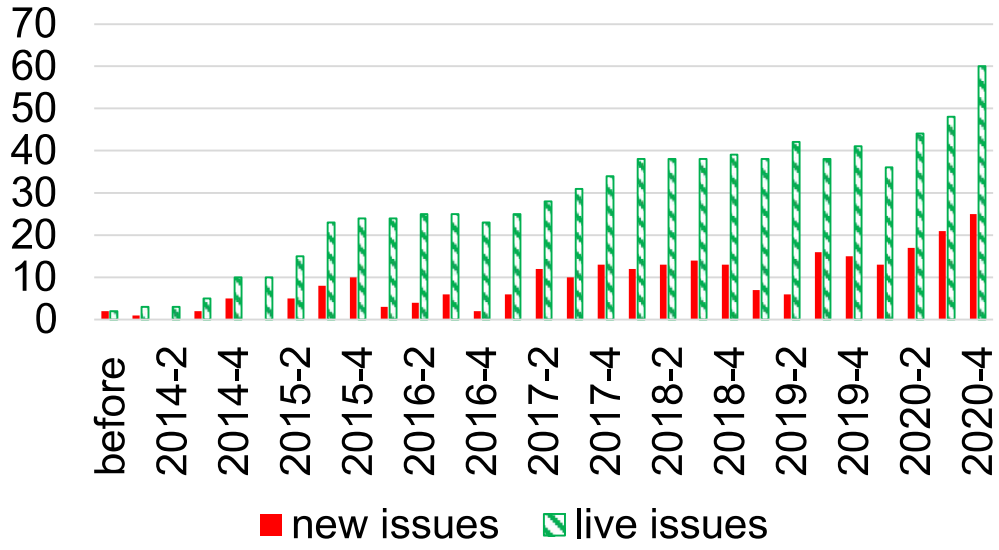
- Research Questions
  - How effective is Rust in preventing memory-safety bugs?
  - What are the characteristics of memory-safety bugs?
  - What lessons can we learn to make Rust more secure?
- Dataset of memory-safety bugs (186 in total)
  - Rust Advisory-DB(CVEs) till 2020-12-31
  - Trophy Case
  - Rust compiler
  - Other GitHub projects

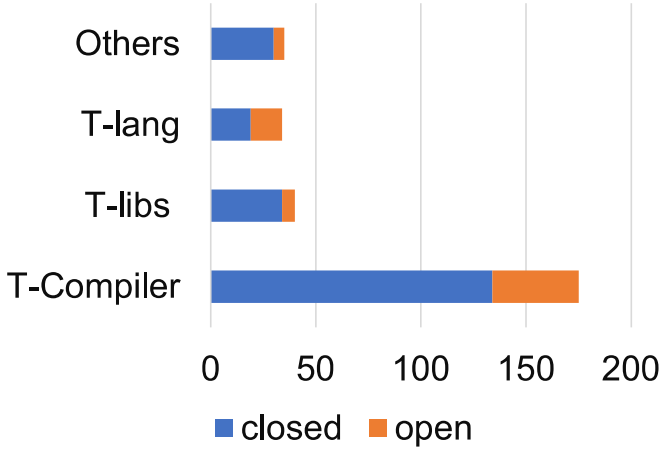


"Memory-safety challenge considered solved? An in-depth study with all Rust CVEs", TOSEM, 2021

How effective is Rust?

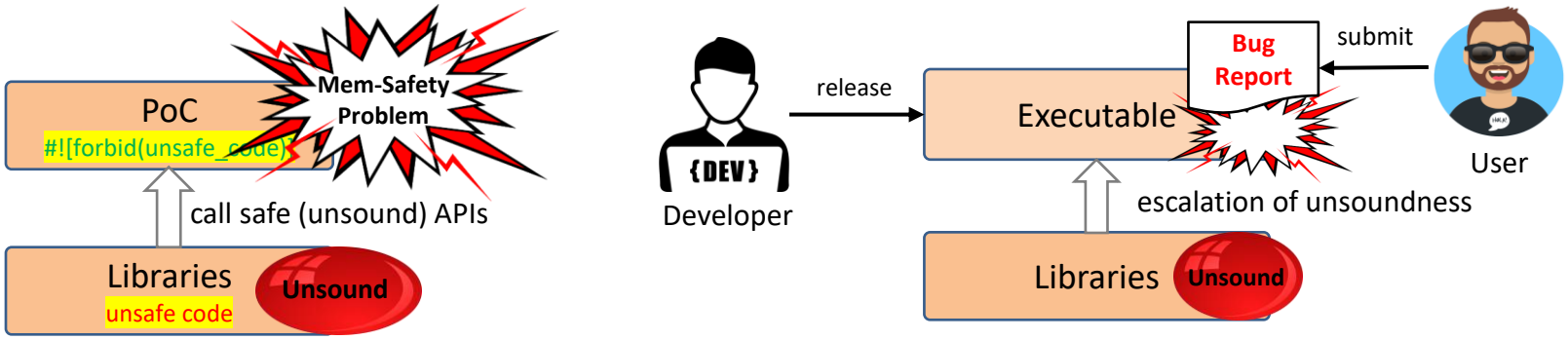
- Do all memory-safety bugs require unsafe code?
  - Yes, except the compiler bug
- How robust is Rust compiler?
  - Trend of unsoundness issues of the Rust compiler





How Severe are These Bugs?

- All CVEs are library bugs
  - Not involve security issues directly
  - Depend on how developers use the API



Characteristics of Bugs

- Automatic Memory Reclaim
- Unsound Function
- Unsound Generic or Trait
- Other Errors

Table 1. Distribution of memory-safety bugs in Rust std-lib + CVEs + others. For simplicity, we count the CVEs of Rust std-lib into Rust std-lib.

Culprit		Consequence					Total
		Buf. Over-R/W	Use-After-Free	Double Free	Uninit Mem	Other UB	
Auto Memory Reclaim	Bad Drop at Normal Block	0 + 0 + 0	1 + 9 + 6	0 + 2 + 1	0 + 2 + 0	0 + 1 + 0	22
	Bad Drop at Cleanup Block	0 + 0 + 0	0 + 0 + 0	1 + 7 + 0	0 + 5 + 0	0 + 0 + 0	13
Unsound Function	Bad Func. Signature	0 + 2 + 0	1 + 5 + 2	0 + 0 + 0	0 + 0 + 0	1 + 2 + 4	17
	Unsoundness by FFI	0 + 2 + 0	5 + 1 + 0	0 + 0 + 0	0 + 0 + 0	1 + 2 + 1	12
Unsound Generic or Trait	Insuff. Bound of Generic	0 + 0 + 1	0 + 33 + 2	0 + 0 + 0	0 + 0 + 0	0 + 0 + 0	36
	Generic Vul. to Spec. Type	3 + 0 + 1	1 + 0 + 0	0 + 0 + 0	1 + 0 + 1	1 + 2 + 0	10
	Unsound Trait	1 + 2 + 1	0 + 0 + 0	0 + 0 + 0	0 + 0 + 0	0 + 2 + 0	6
Other Errors	Arithmetic Overflow	3 + 1 + 0	1 + 0 + 0	0 + 0 + 0	0 + 0 + 0	0 + 0 + 0	5
	Boundary Check	1 + 9 + 0	1 + 0 + 0	0 + 0 + 0	0 + 0 + 0	1 + 0 + 0	12
	No Spec. Case Handling	2 + 2 + 1	0 + 0 + 0	0 + 0 + 0	0 + 0 + 0	2 + 1 + 1	9
	Exception Handling Issue	0 + 0 + 0	0 + 0 + 0	0 + 0 + 0	0 + 0 + 0	1 + 2 + 1	4
	Wrong API/Args Usage	0 + 3 + 0	1 + 4 + 0	0 + 0 + 0	0 + 1 + 1	0 + 5 + 2	17
	Other Logical Errors	0 + 4 + 1	2 + 3 + 4	0 + 0 + 1	0 + 1 + 0	1 + 4 + 1	22
Total		40	82	12	12	39	185

Case1: Auto Memory Reclaim

Code 1. PoC of use-after-free and double free bugs due to automatic memory reclaim.

```
fn genvec() -> Vec<u8> {
    let mut s = String::from("a tmp string");
    /*fix2: let mut s = ManuallyDrop::new(String::from("a tmp string"));*/
    let ptr = s.as_mut_ptr();
    unsafe {
        let v = Vec::from_raw_parts(ptr, s.len(), s.len());
        /*fix1: mem::forget(s);*/
        return v;
        /*s is freed when the function returns*/
    }
}
fn main() {
    let v = genvec();
    assert_eq!('a' as u8,v[0]); /*use-after-free*/
    /*double free: v is released when the function returns*/
}
```

Case2: Drop Uninitialized Memory

Code 4. PoC of dropping uninitialized memory during stack unwinding.

```
1  struct Foo { vec : Vec<i32>, }
2  impl Foo {
3     pub unsafe fn read_from(src: &mut Read) -> Foo {
4        let mut foo = mem::uninitialized::<Foo>();
5        //panic!(); /*panic here would recalim the uninitialized memory of type <Foo>*/
6        let s = slice::from_raw_parts_mut(&mut foo as *mut _ as *mut u8, mem::size_of::<Foo>());
7        src.read_exact(s);
8        foo
9     }
10 }
11 fn main() {
12    let mut v = vec![0,1,2,3,4,5,6];
13    let (p, len, cap) = v.into_raw_parts();
14    let mut u = [p as u64, len as _, cap as _];
15    let bp:*const u8 = &u[0] as *const u64 as *const _;
16    let mut b:&[u8] = unsafe { slice::from_raw_parts(bp, mem::size_of::<u64>()*3) };
17    let mut foo = unsafe{Foo::read_from(&mut b as _)};
18    println!("foo = {:?}", foo.vec);
19 }
```

Case 3: Insufficient Trait Bound

Code 5. PoC of lacking Send trait bound to generic.

```
struct MyStruct<T> {t:T}
unsafe impl<T> Send for MyStruct<T> {}
//fix: unsafe impl<T:Send> Send for MyStruct<T> {}
fn main() {
    let mut s = MyStruct { t:Rc::new(String::from("untouched␣data")) };
    for _ in 0..99{
        let mut c = s.clone();
        std::thread::spawn(move || {
            if !Rc::get_mut(&mut c.t).is_none(){
                (*Rc::get_mut(&mut c.t).unwrap()).clear();
            }
            println!("c.t␣=␣{:?}", c.t);
        });
    }
}
```

Case 4: Vulnerable Generic

Code 6. PoC of unsound generic that does not respect the memory alignment.

```
#[repr(align(128))]
struct LargeAlign(u8);
struct MyStruct<T> { v:Vec<u8>,  _marker:PhantomData<*const T>, }
impl<T:Sized> MyStruct<T> {
    fn from(mut value:T) -> MyStruct<T> {
        let size = size_of::<T>();
        let mut v = Vec::with_capacity(size_of::<T>());
        let src:*const T = &value;
        unsafe {
            ptr::copy(src, v.as_mut_ptr() as _, 1);
            v.set_len(size)
        }
        MyStruct { v, _marker:PhantomData }
    }
}
impl<T:Sized> ::std::ops::Deref for MyStruct<T> {
    type Target = T;
    fn deref(&self) -> &T{
        let p = self.v.as_ptr() as *const u8 as *const T;
        unsafe { &*p }
    }
}
fn main() {
    let s = MyStruct::from(LargeAlign(123));
    let v = &*s as *const _ as usize;
    assert!(v % std::mem::align_of::<LargeAlign>() == 0);
}
```

Case 5: Unsound Trait

Code 7. PoC of unsound Trait.

```
trait MyTrait {
    fn type_id(&self) -> TypeId where Self: 'static {
        TypeId::of::<Self>()
    }
}
impl dyn MyTrait {
    pub fn is<T:MyTrait + 'static>(&self) -> bool {
        TypeId::of::<T>() == self.type_id()
    }
    pub fn downcast<T:MyTrait + 'static>(self: Box<Self>) -> Result<Box<T>, Box<dyn MyTrait>> {
        if self.is::<T>(){ unsafe {
           let raw:*mut dyn MyTrait = Box::into_raw(self);
           Ok(Box::from_raw(raw as *mut T))
        }} else { Err(self) }
    }
}
impl<T> MyTrait for Box<T> {}
impl MyTrait for u128 {}
impl MyTrait for u8 {
    fn type_id(&self) -> TypeId where Self: 'static {
        TypeId::of::<u128>()
    }
}
fn main(){
    let s = Box::new(10u8);
    let r = MyTrait::downcast::<u128>(s);
}
```

Lessons Learnt?

- Best practice for code suggestion?
  - Generic Bound Declaration
  - Avoiding Bad Drop at Cleanup Block.
- Static analysis with unsafe code?
  - We will discuss more next week