CCF ChinaSoft 2022 CCF 中国软件大会

基于静态分析的Rust内存安全缺陷检测研究

报告人：徐辉

复旦大学

报告日期：2022.11.25

大纲

一、问题背景

二、Rust指针缺陷检测方法

三、实验结论

四、论文发表心得

大纲

一、问题背景

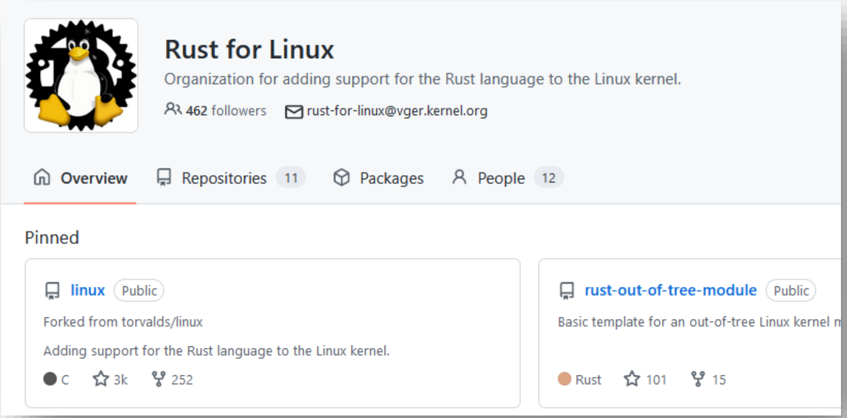
二、Rust指针缺陷检测方法

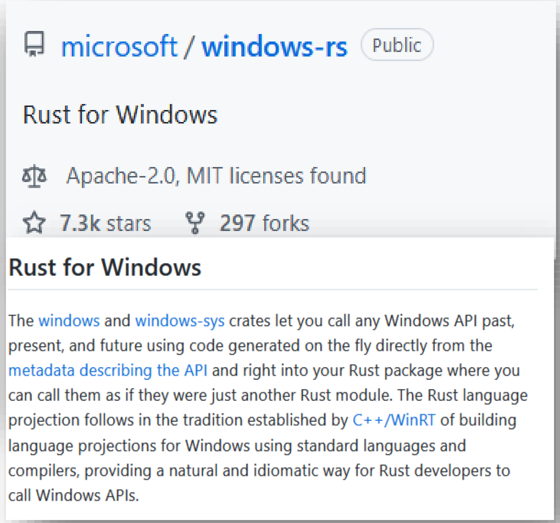
三、实验结论

四、论文发表心得

Rust语言

- 系统级安全编程语言
  - 内存安全
  - 并发安全
  - 效率





2006年	2011年	2015年	2020年	2021年
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

AWS, Huawei, Google, Microsoft, Mozilla...

Rust如何保障内存安全?

- 内存安全问题产生的主要原因之一是指针别名导致悬空指针
  - 手动释放内存或调用析构函数
  - 函数返回时发生的自动析构或内存释放
- Rust设计的目标之一是编译时检查指针别名（共享可变引用）
  - 但一般意义上的指针分析是NP-hard问题
  - 智能指针可行，但作为运行时方案，效率低
  - Rust在语法设计中引入所有权机制，简化指针分析问题

Rust所有权模型 => XOR Mutability

- 一个对象有且只有一个所有者
- 所有权可以转移给其它变量
  - 用完不用还
- 所有权可以被其它变量借用
  - 用完自动归还
  - 只读（immutable）借用：&
  - 可变（mutable ）借用：& mut

```
fn main(){
  let mut alice = Box::new(1);
  let bob = alice;
  println!("bob:{}", bob);
  println!("alice:{}", alice);
}
```

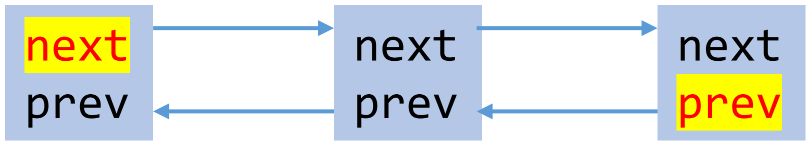
- `let mut alice = Box::new(1);` ← alice拥有Box对象
- `let bob = alice;` ← 转移所有权转移给bob，alice失去Box对象的所有权
- `println!("alice:{}", alice);` ✗

```
fn main(){
  let mut alice = Box::new(1);
  let bob = &alice;
  println!("alice:{}", alice);
  println!("bob:{}", bob);
  *alice = 2;
}
```

- `let bob = &alice;` ← bob只读借用Box对象，alice临时失去修改权，保留只读权
- `println!("alice:{}", alice);` ← alice可读
- `*alice = 2;` ← bob自动归还Box对象，alice恢复修改权
- ✓

如果需要违背XOR Mutability怎么办?

- 以双向链表为例，中间节点被前后两个节点访用
- Rust为了提升可用性所做的妥协
  - 智能指针（性能损失）
  - 允许使用裸指针（unsafe模式）
    - 逃逸编译器的借用检查 => 指针别名




```
struct List{
    val: u64,
    prev: Option<Rc<RefCell<List>>>,
    next: Option<Rc<RefCell<List>>>,
}
```

方法一：智能指针

```
struct List{
    val: u64,
    next: *mut List,
    prev: *mut List,
}
```

方法二：允许使用裸指针

Unsafe Rust

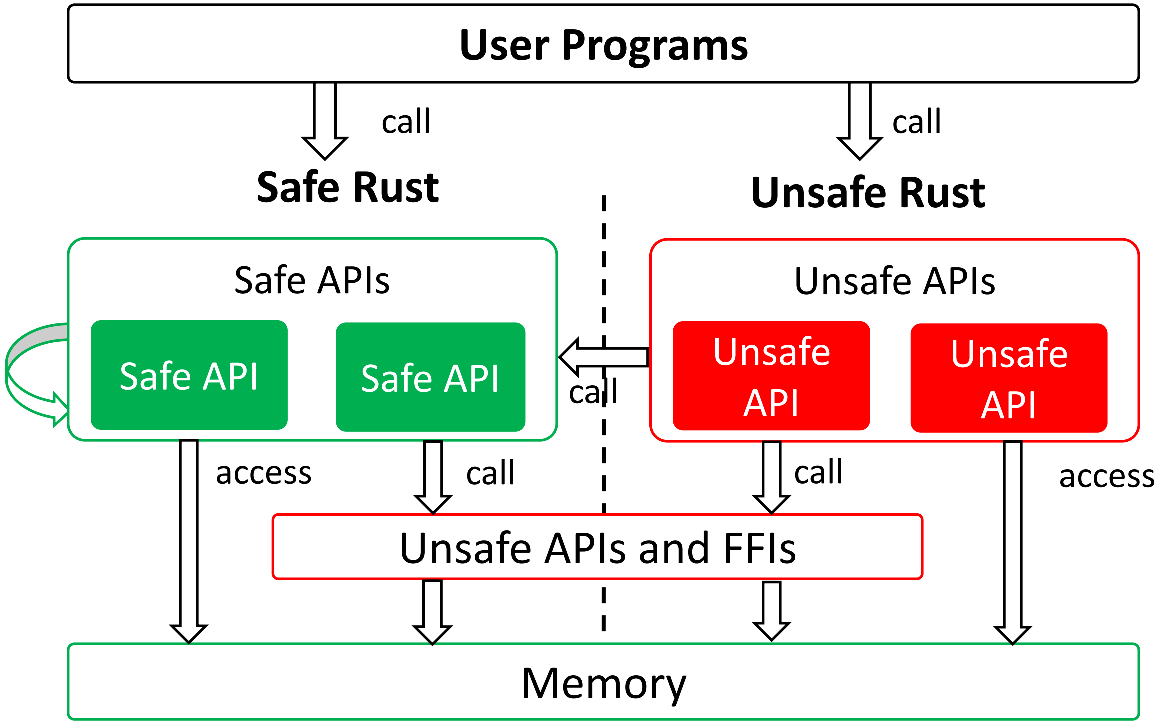
- Unsafe Rust功能：
  - 解引用裸指针
  - 调用unsafe函数
  - 调用FFI（其它语言接口）
- 使用条件：必须标注unsafe

```
let mut num = 5;
let r1 = &num as *const i32;
unsafe {
  println!("r1 is: {}", *r1);    ← 解引用裸指针
}
```

```
unsafe fn risky() {                ← 定义unsafe函数
    let address = 0x012345usize;
    let r = address as *const i32;
}
unsafe { risky(); }                ← 调用unsafe函数
```

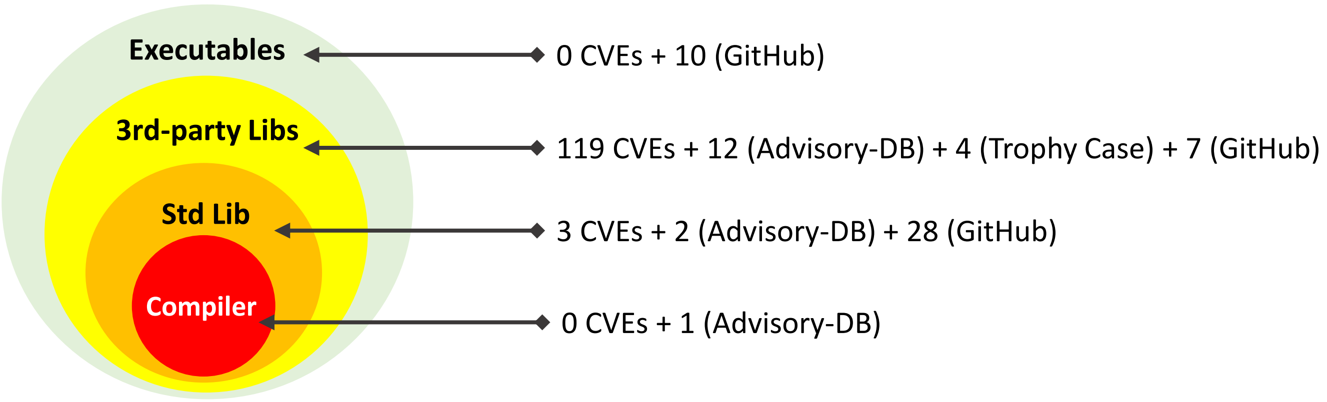
Rust的安全哲学

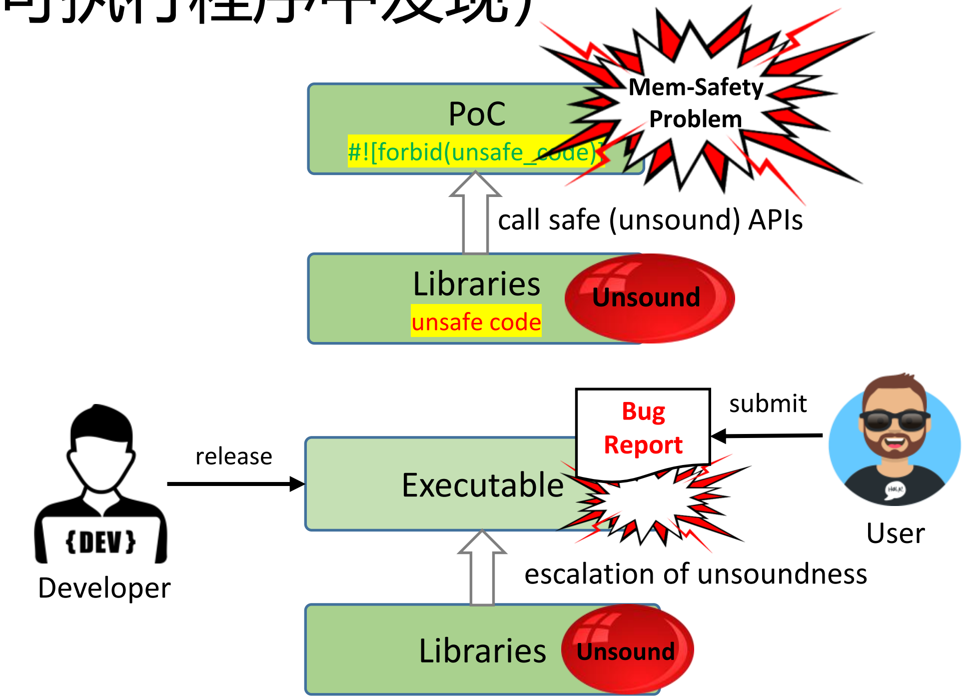
- Safe API无论如何被使用都不应带来未定义行为
- 程序员应避免直接使用unsafe code
- Interior unsafe：将unsafe code封装为safe API



Rust实际表现如何?

- 调研了2020年12月31日前报告的185个内存安全漏洞[TOSEM'21]
  - Rust在内存安全防护方面效果不错
  - 所有的漏洞（除了1个编译器漏洞）都需要unsafe code
  - 大部分CVEs都是 API soundness的问题（未在可执行程序中发现）





[TOSEM'21] “Memory-safety challenge considered solved? An in-depth study with all Rust CVEs”, TOSEM, 2021.

Rust项目中内存安全漏洞的特点

- ❑ Automatic Memory Reclaim
- ❑ Unsound Function
- ❑ Unsound Generic or Trait

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

Auto Memory Reclaim问题：示例1


```
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*/
}
```

创建一个临时字符串s

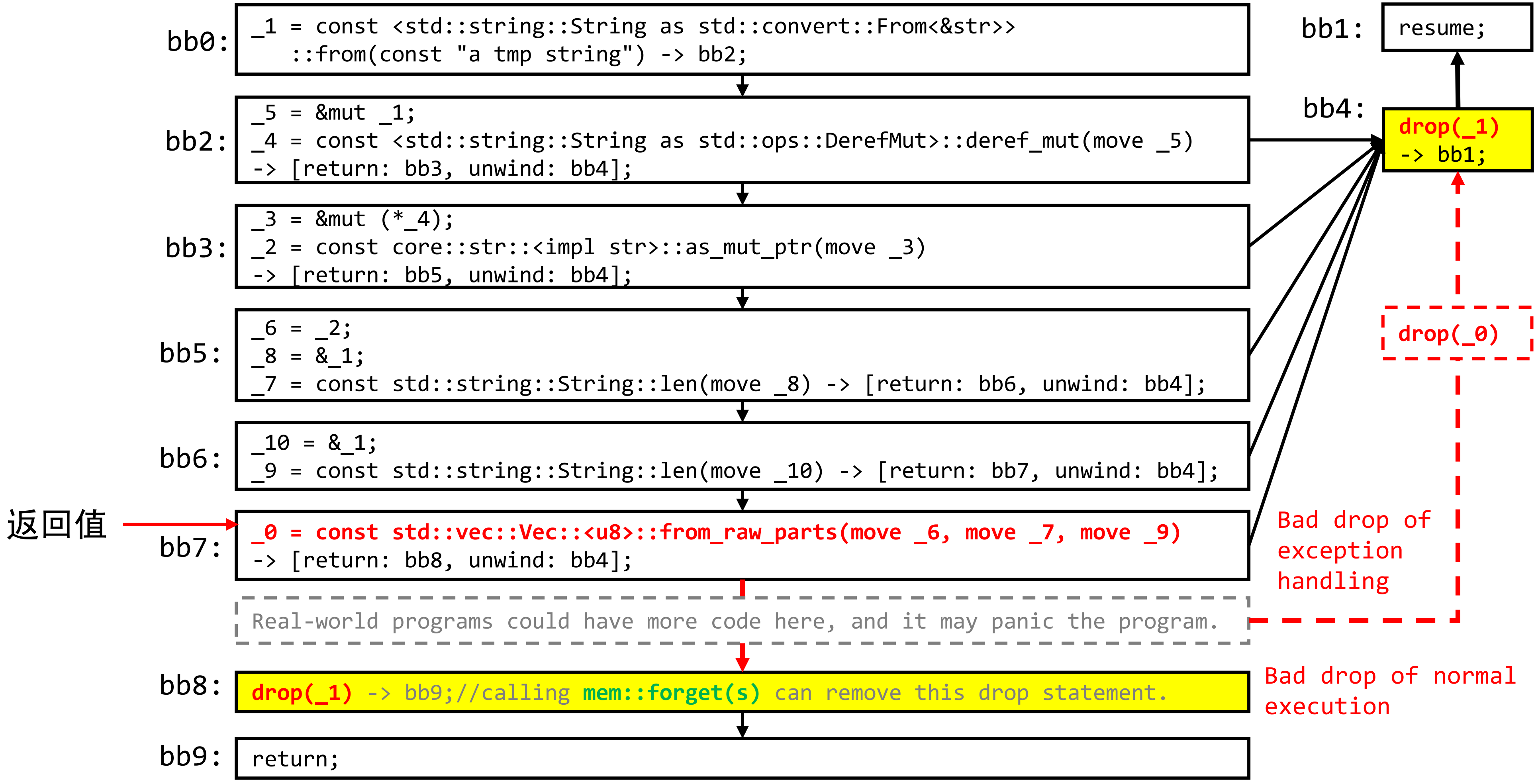
通过unsafe将v指向临时内存

返回v

自动析构s，造成悬空指针v

访问v造成use-after-free

从Rust MIR分析Auto Memory Reclaim问题



Auto Memory Reclaim问题：示例2

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

创建未初始化的变量foo

Panic将导致访问未初始化内存

大纲

一、问题背景

二、Rust指针缺陷检测方法

三、实验结论

四、论文发表心得

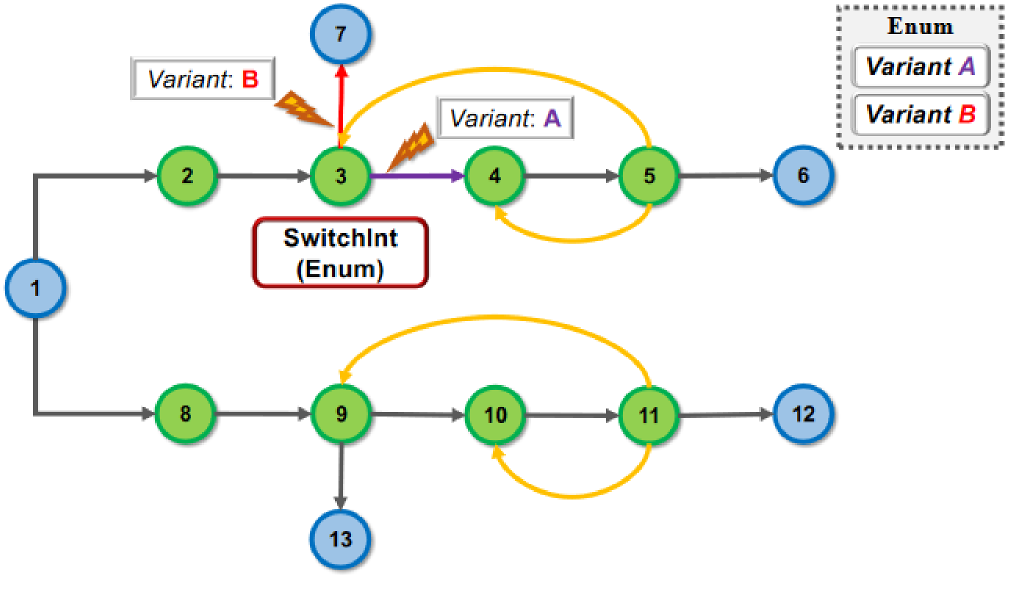
研究挑战和思路

- 研究挑战：指针分析是NP-hard问题
  - 准确性：应采用路径敏感的指针分析算法，避免过多误报
  - 分析效率：应基于Rust MIR的特点对算法进行优化，使其可行
- 整体思路：基于编译过程中的生成的MIR进行静态分析
  - 路径提取：控制流图=>生成树
  - 别名分析：分析指针之间的关联关系
  - 模式识别：根据预定义的缺陷模式检测指针漏洞

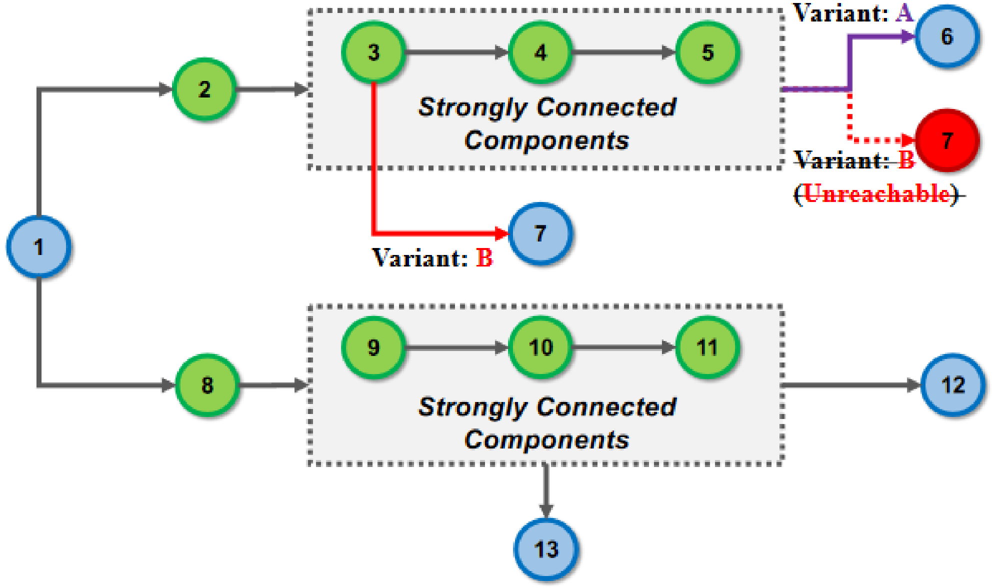
“SafeDrop: Detecting memory deallocation bugs of Rust programs via static data-flow analysis.” TOSEM, 2022.

路径提取

- 规律：同一个强联通分量（SCC）的may alias关系一般存在上界
  - 方法：基于tarjan算法进行SCC检测 => 生成树
  - 对SCC出口处的alias关系统一取上界
- 特殊情况特殊处理



控制流图



生成树

别名分析

路径提取 → 别名分析 → 模式识别

❑ 主要规则:

LValue := Use::Move(RValue)	: e.g., a = move b	=>	$S_a = S_a - a,\ S_b = S_b \cup a$
| := Use::Copy(RValue)	: e.g., a = b	=>	$S_a = S_a - a,\ S_b = S_b \cup a$
| := Ref/AddressOf(RValue)	: e.g., a = &b	=>	$S_a = S_a - a,\ S_b = S_b \cup a$ (近似处理multi-level pointers)
| := Deref(RValue)	: e.g., a = *(b)	=>	$S_a = S_a - a,\ S_b = S_b \cup a$ (近似处理multi-level pointers)
| := Fn(Move(RValue))	: e.g., a = Fn(move b)	=>	$Update(S_a, S_b)$ (过程间分析)
| := Fn(Copy(RValue))	: e.g., a = Fn(b)	=>	$Update(S_a, S_b)$ (过程间分析)

❑ 示例:

```
Statement 1:  _2 = &_1;         // alias set:{_1, _2}
Statement 2:  _1 = move _4;     // alias sets:{_1, _4}, {_2}
Statement 3:  _3 = &_1;         // alias sets:{_1, _3, _4}, {_2}
```

域敏感和过程间分析


```
enum E { A, B { ptr: *mut u8 } }
struct S { b: E }

fn foo(_1: &mut String) -> S:
    _3 = str::as_mut_ptr(_1);  // alias set: {_3, _1}
    ((_2 as B).0: *mut u8) = move _3;    // alias set: {_2.0, _3, _1}
    discriminant(_2) = 1;  // instantiate the enum type to variant B
    (_0.0: E) = move _2; // alias sets: {_0.0, _2}, {_0.0.0, _2.0, _3, _1}
    return;

fn main():
    _1 = String::from("string"); // alias set: {_1},
    _2 = &mut _1;    // alias set: {_2, _1},
    _3 = foo(move _2); // alias set: {_3.0.0, _2, _1}
    ...
```

域敏感

更新

过程间分析

缺陷模式

路径提取 → 别名分析 → 模式识别

生成新的所有者

```
Pattern 1: getPtr() -> unsafeConstruct() -> drop() -> use()  => UAF
Pattern 2: getPtr() -> unsafeConstruct() -> drop() -> drop() => DF
Pattern 3: getPtr() -> drop() -> unsafeConstruct() -> use()  => UAF
Pattern 4: getPtr() -> drop() -> unsafeConstruct() -> Drop() => DF
Pattern 5: getPtr() -> drop() -> use() => UAF
Pattern 6: uninitialized() -> use()  => IMA
Pattern 7: uninitialized() -> drop() => IMA
```

大纲

一、问题背景

二、Rust指针缺陷检测方法

三、实验结论

四、论文发表心得

实验效果

- 基于已知CVE评估检测能力

Crate			CVE		SafeDrop Report (TP/FP)						Rudra
Name	# Methods	LoC	CVE-ID	Type	UAF	DF	DP	IMA	Total	Recall	TP/FP
isahc	89	1304	2019-16140	UAF	-	0/1	1/0	-	1/1	100%	0/0
open-ssl	1188	20764	2018-20997	UAF	1/2	-	0/1	-	1/3	100%	0/0
linea	1810	24317	2019-16880	DF	-	1/0	-	10/0	11/0	100%	1/2
ordnung	145	2546	2020-35891	DF	0/1	-	3/0	-	3/1	100%	1/3
crossbeam	221	4184	2018-20996	IMA	-	0/1	-	2/0	2/1	100%	0/0
generator*	158	2608	2019-16144	IMA	-	-	-	1/0	1/0	100%	0/0
linkedhashmap	137	1974	2020-25573	IMA	-	-	-	1/0	1/0	100%	0/0
smallvec*	187	2297	2018-20991 2019-15551	DF DF	-	-	1/2	1/0	2/2	100%	2/1

高召回率
低误报率

- 基于GitHub上其它开源Rust项目的实验

Crate			SafeDrop Report (TP/FP)					Rudra	MirChecker	Miri
Name	# Methods	Loc	UAF	DF	DP	IMA	Total	TP/FP	TP/FP	TP/FP
stretch*	4280	78350	-	24/2	-	-	24/2	N.A.	0/0	N.A.
idroid*	5484	73856	-	7/0	-	-	7/0	N.A.	0/0	N.A.
rose-tools*	996	19160	-	27/3	-	-	27/3	N.	[illegible]	[illegible]
wasm-gb	165	5719	-	-	20/0	-	20/0	N.	[illegible]	[illegible]
rust-poker	113	2298	-	1/0	-	-	1/0	0,	[illegible]	[illegible]
teardown-tree	677	7258	-	0/2	1/0	-	1/2	0/3	0/4	N.A.
apres-bindings	18	1139	-	-	14/0	-	14/0	0/0	0/0	0/0
rust-workshop	13	1897	-	-	2/0	-	2/0	0/0	0/0	0/0
teraflops	40	606	-	-	2/0	-	2/0	0/0	0/3	0/0
rust-libcint	37	600	-	2/0	-	-	2/0	0/0	0/0	0/0
bzip2	20	170	-	-	2/1	-	2/1	0/0	0/0	0/0
rust-webassembly	12	118	-	-	1/0	-	1/0	0/0	0/1	0/0

double free和悬空指针问题比较多

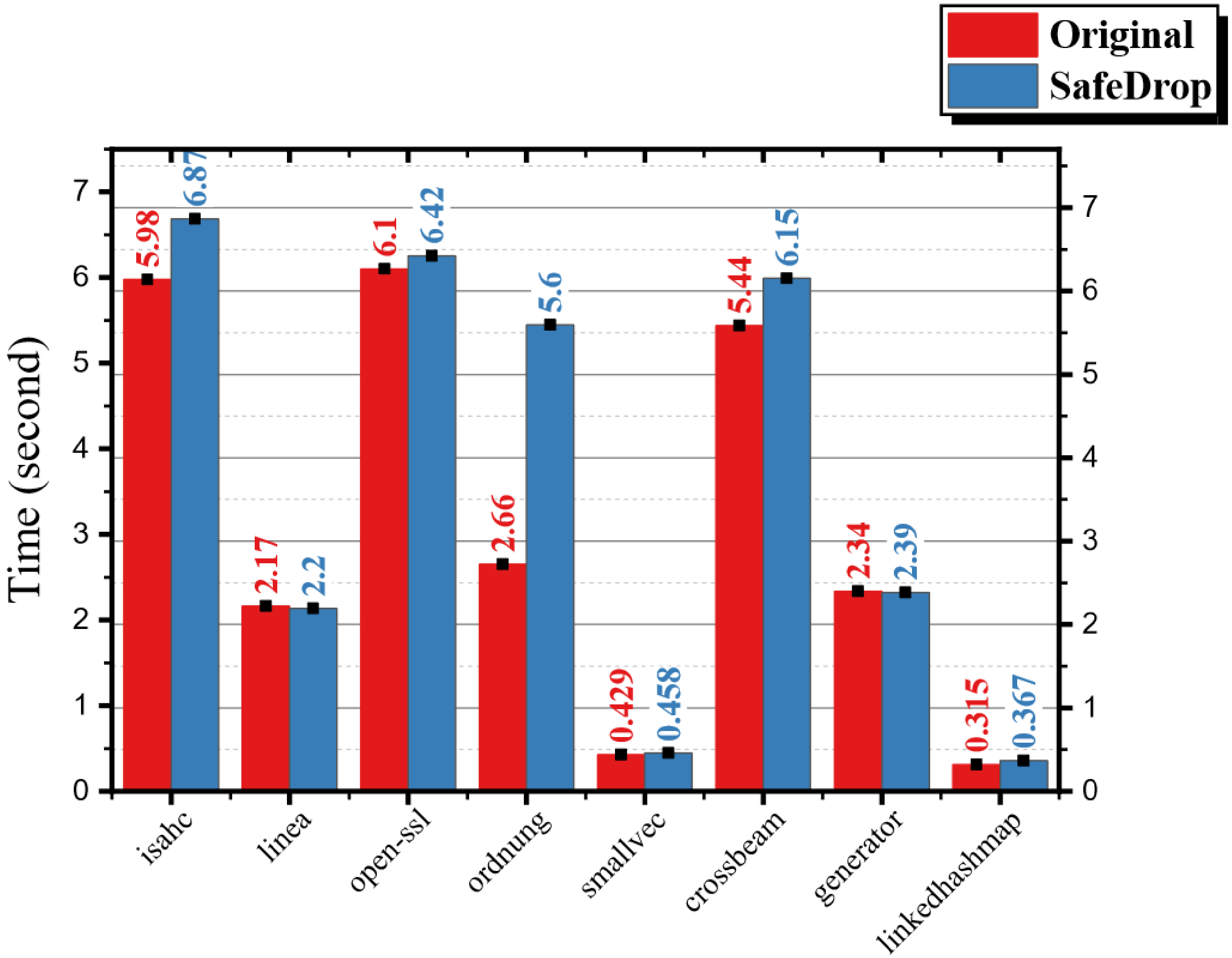
案例分析


```
#Real-world Bug Found by SafeDrop    // from crate: apres_bindings
pub fn get_ppqn( // interior unsafe function
    midi_ptr: *mut MIDI) -> u16 {
-   let mut midi = unsafe { Box::from_raw(midi_ptr) }; // unsafe constructor of Box<T>
+   let midi = mem::ManuallyDrop::new(Box::from_raw(midi_ptr));
+   // use smart pointer mem::ManauallyDrop<T> to avoid being dropped
    let output = midi.get_ppqn(); // double-free occurs if unwinding here
-   Box::into_raw(midi); // transfer to raw pointer to avoid being dropped
    output
}
```

提前

Panic将导致双重释放

性能：SafeDrop vs 原始Rust编译器



总结和思考

- 问题根源是Rust的自动析构机制
  - XOR Mutability保证自动析构的安全性
  - Unsafe会破坏安全性保证
  - 自动析构优于手动析构
  - 该问题的反面是内存泄露问题
- SafeDrop证明可在Rust编译器中适当增加相应的缺陷检测功能

大纲

论文发表心得

- 优先选择（发现）新场景：容易发表
  - Android系统(2010-2017)
  - 深度学习应用软件(2013-2020)
  - 考验平时技术和算法积累
- 坚持优化完善：审稿随机性
  - 重视论文评审意见，不断投稿
  - 预防scoop：arXiv
- 重视期刊论文：顶会论文期刊化
  - FSE/ASE等会议引入journal-first track或major revision机制

Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones
W Enck, P Gilbert, S Han, V Tendulkar... - ACM Transactions on ..., 2014 - dl.acm.org
Today's smartphone operating systems frequently fail to provide users with visibility into how third-party applications collect and share their private data. We address these shortcomings ...
Save Cite Cited by 4585 Related articles All 77 versions

Dynodroid: An input generation system for android apps
A Machiry, R Tahiliani, M Naik - Proceedings of the 2013 9th Joint ..., 2013 - dl.acm.org
... We present a system **Dynodroid** for generating relevant inputs to unmodified Android apps. **Dynodroid** views an ... This paper presents a system **Dynodroid** that satisfies the above criteria.
Save Cite Cited by 777 Related articles All 15 versions

Deepxplore: Automated whitebox testing of deep learning systems
K Pei, Y Cao, J Yang, S Jana - proceedings of the 26th Symposium on ..., 2017 - dl.acm.org
... We design, implement, and evaluate **DeepXplore**, to the best ... between DL systems, **DeepXplore** also supports adding ... We demonstrate that **DeepXplore** efficiently finds thousands
Save Cite Cited by 1073 Related articles All 22 versions

{**TVM**}: An automated {End-to-End} optimizing compiler for deep learning
T Chen, T Moreau, Z Jiang, L Zheng, E Yan... - ... USENIX Symposium on ..., 2018 - usenix.org
... This section describes **TVM** by using an example to walk through its components. Figure 2 summarizes execution steps in **TVM** and their corresponding sections in the paper. The ...
Save Cite Cited by 969 Related articles All 19 versions

ESEC/FSE 2023 (series) /

Research Papers

Call for Papers FAQ

Call for Papers

We invite high-quality submissions, from both industry and academia, describing original and unpublished results of theoretical, empirical, conceptual, and experimental software engineering research.

NEW THIS YEAR: *major revisions allowed!* The main novelty of this year's review process is that the initial output can be *accept, reject or major revision*. In case a paper is deemed publishable upon major revision, authors are granted **8 weeks** to perform major revisions, which might include additional experiments or new analyses of existing results; major rewriting of algorithms and explanations; clarifications, better scoping, and improved motivations. The same reviewers who requested major revisions will then assess whether the revised submission satisfies their requests adequately.

感谢观看

Thank You