Unsafe Rust 代码治理

徐辉

复旦大学

2025年7月19日

大纲

- 一、Rust的关键问题：Unsafe代码
- 二、Unsafe代码治理：tag-std项目
- 三、Unsafe代码验证：RAPx项目

Unsafe代码治理：安全属性标注问题

❑ 当前unsafe代码最佳实践：文本形式的安全属性标注

- ➢ Unsafe函数声明处，通过doc说明安全使用条件，以防其他开发者用错
- ➢ Unsafe函数调用处，通过注释说明为何安全，防止引入安全缺陷

```
/// 安全文档：说明安全使用foo的条件
pub unsafe fn foo (p: *const u8) {
    ...
}
```

```
unsafe {
// 安全注释：解释为何使用foo是安全的
    foo(p);
}
```

❑ 存在问题：

- ➢ 标注繁琐：代码中存在大量、重复的文本描述
- ➢ 规范性差：漏标、错标的情况比较普遍（即便是Rust标准库）

Unsafe代码治理：严谨性问题（Soundness）

❑编译器无法验证：

- ❑Safe函数封装的严谨性，即无论如何使用不应造成内存安全问题
- ❑Unsafe函数安全属性标注的正确性（充分且必要）

```rust
fn get_slice<T>(s: &[T], l: usize) -> &[u32] {
    let ptr = s.as_ptr() as *const u32;
    let len = s.len();
    if l < len {
        unsafe { slice::from_raw_parts(ptr, l) }
    } else {
        unsafe { slice::from_raw_parts(ptr, len) }
    }
}
```

将泛型slice转换为u32 slice

无法排除越界访问

- 攻击举例：参数s为u8 slice

PoC of CVE-2021-45709

Rust标准库验证挑战：Rust基金会/AWS发起

Contest Structure: Tools

3 tools accepted, 2 under review

$25k reward for accepted application

Verification Tools

- Kani
- GOTO Transcoder
- VeriFast

Add Tool: Flux #362

Open

nilehmann opened last month

Add Tool: KMIR by Runtime Verification #296

Open #310

gregorymakodzeba opened last month

Contest Structure: Challenges

1: Verify core transmuting methods
2: Verify the memory safety of core intrinsics using raw pointers
3: Verifying Raw Pointer Arithmetic Operations
4: Memory safety of BTreeMap's btree::node module
5: Verify functions iterating over inductive data type: linked_list
6: Safety of NonNull
7: Safety of Methods for Atomic Types & Atomic Intrinsics
8: Contracts for SmallSort
9: Safe abstractions for core::time::Duration
10: Memory safety of String
11: Safety of Methods for Numeric Primitive Types
12: Safety of NonZero
13: Safety of CStr
14: Safety of Primitive Conversions
15: Contracts and Tests for SIMD Intrinsics
16: Verify the safety of Iterator functions
17: Verify the safety of slice functions
18: Verify the safety of slice iter functions
19: Safety of RawVec
20: Verify the safety of char-related functions in str::pattern
21: Verify the safety of substring-related functions in str::pattern
22: Verify the safety of str iter functions
23: Verify the safety of Vec functions part 1
24: Verify the safety of Vec functions part 2
25: Verify the safety of VecDeque functions

(... and growing!)

aws $5/10/15K reward for solutions

Challenges

25 challenges published, 5 resolved

1: Verify core transmuting methods
2: Verify the memory safety of core intrinsics using raw pointers
~~3: Verifying Raw Pointer Arithmetic Operations~~
4: Memory safety of BTreeMap's btree::node module
5: Verify functions iterating over inductive data type: linked_list
~~6: Safety of NonNull~~
7: Safety of Methods for Atomic Types & Atomic Intrinsics
8: Contracts for SmallSort
~~9: Safe abstractions for core::time::Duration~~
10: Memory safety of String
~~11: Safety of Methods for Numeric Primitive Types~~
12: Safety of NonZero
13: Safety of CStr
~~14: Safety of Primitive Conversions~~
15: Contracts and Tests for SIMD Intrinsics
16: Verify the safety of Iterator functions
17: Verify the safety of slice functions
18: Verify the safety of slice iter functions
19: Safety of RawVec
20: Verify the safety of char-related functions in str::pattern
21: Verify the safety of substring-related functions in str::pattern
22: Verify the safety of str iter functions
23: Verify the safety of Vec functions part 1
24: Verify the safety of Vec functions part 2
25: Verify the safety of VecDeque functions

aws

https://model-checking.github.io/verify-rust-std/intro.html

大纲

- ❑一、Rust的关键问题：Unsafe代码
- ❑二、Unsafe代码治理：tag-std项目
- ❑三、Unsafe代码验证：RAPx项目

tag-std：基于DSL的unsafe代码安全属性标注

❑ 使用DSL定义常用的安全属性，并在unsafe函数上下文中使用：

- Unsafe函数声明处，标注安全属性
- Unsafe函数调用处，discharge安全属性，并解释原因

```
#[safety::precond::Inbound(p, u32)]
pub unsafe fn foo (p: *const u8) {
    ...
}
```

```
unsafe {
    #[safety::discharges(Inbound,
        memo = "...")]
    foo(p);
}
```


项目地址：https://github.com/Artisan-Lab/tag-std

tag-std：编译时生成安全文档 + 安全合约

```
#[safety::precond::Inbound(p, u32)]
```

安全文档

```
/// The memory from p to p+3 should belong to a single allocated object.
```

安全合约

兼容不同验证工具

```
#[rapx::inner(property=Inbound(p, u32), kind = "precond")]
```

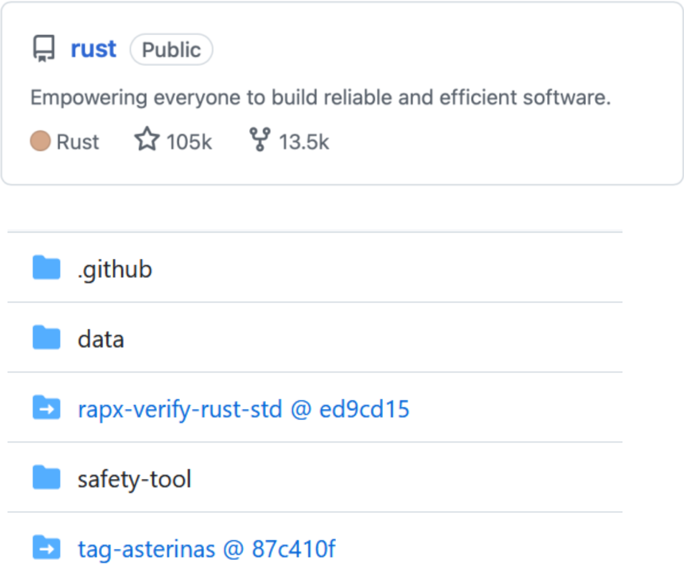
```
#[kani::require(...)]
```

DSL定义：标准库部分

<table>
<tr><th>Category</th><th>Safety Property</th><th>Meaning</th><th>Usage</th></tr>
<tr><td rowspan="4">Layout</td><td>Align(p, T)</td><td>p % alignment(T) = 0 && sizeof(T) % alignment(T) = 0</td><td>precondition</td></tr>
<tr><td>Sized(T)</td><td>sizeof(T) = const, const ≥ 0</td><td>option</td></tr>
<tr><td>ZST(T)</td><td>sizeof(T) = 0</td><td>precondition</td></tr>
<tr><td>!Padding(T)</td><td>Padding(T) = 0</td><td>precondition</td></tr>
<tr><td rowspan="5">Pointer</td><td>!Null(p)</td><td>p != 0</td><td>precondition</td></tr>
<tr><td>!Dangling(p)</td><td>allocator(p) != none</td><td>precond, hazard</td></tr>
<tr><td>Allocated(p, T, len, A)</td><td>∀ i ∈ 0..sizeof(T) * len, allocator(p+i) = A</td><td>precondition</td></tr>
<tr><td>InBound(p, T, len, arrage)</td><td>[p, p+(len+1)*sizeof(T)) ∈ arrage</td><td>precondition</td></tr>
<tr><td>!Overlap(dst, src, len, T)</td><td>|dst-src| > sizeof(T) * len</td><td>precondition</td></tr>
<tr><td rowspan="5">Content</td><td>ValidInt(exp, vrange)</td><td>exp ∈ vrange</td><td>precondition</td></tr>
<tr><td>ValidString(arange)</td><td>mem(arange) ∈ UTF-8</td><td>precond, hazard</td></tr>
<tr><td>ValidCStr(p, len)</td><td>mem(p+len, p+len+1) = null</td><td>precondition</td></tr>
<tr><td>Init(p, T, len)</td><td>∀i∈0..len, mem(p+i*sizeof(T), p+(i+1)*sizeof(T)) = validobj(T)</td><td>precond, hazard</td></tr>
<tr><td>Unwrap(x, T, target)</td><td>unwrap(x) = target, target ∈ {Ok(T), Err, Some(T), None}</td><td>precondition</td></tr>
<tr><td rowspan="3">Aliasing</td><td>Owning(p)</td><td>ownership(*p) = none</td><td>precondition</td></tr>
<tr><td>Alias(p1, p2)</td><td>p1 = p2</td><td>hazard</td></tr>
<tr><td>Alive(p, l)</td><td>lifetime(*p) ≥ l</td><td>precondition</td></tr>
<tr><td rowspan="4">Misc</td><td>Pinned(p)</td><td>p = &*p</td><td>hazard</td></tr>
<tr><td>!Volatile(p)</td><td>volatile(*p) = t, t ∈ {true, false}</td><td>precondition</td></tr>
<tr><td>Opened(fd)</td><td>opened(fd) = true</td><td>precondition</td></tr>
<tr><td>Trait(T, trait)</td><td>trait ∈ Trait(T), trait ∈ {Copy, Unpin, ...}</td><td>Option</td></tr>
</table>

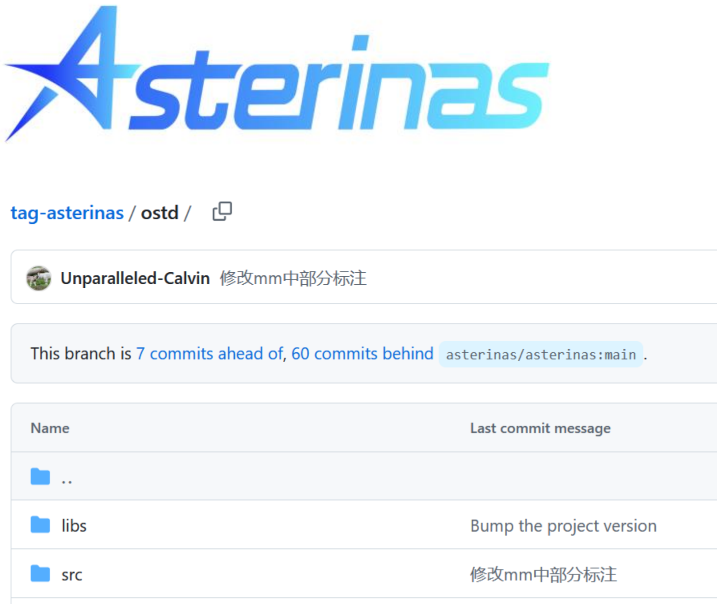
“Annotating and Auditing the Safety Properties of Unsafe Rust”, Zihao Rao, et al, arXiv:2504.21312, 2025.

tag-std应用（进行中）



标准库
（merge难度较大）

Rust for Linux
（安全属性抽象）



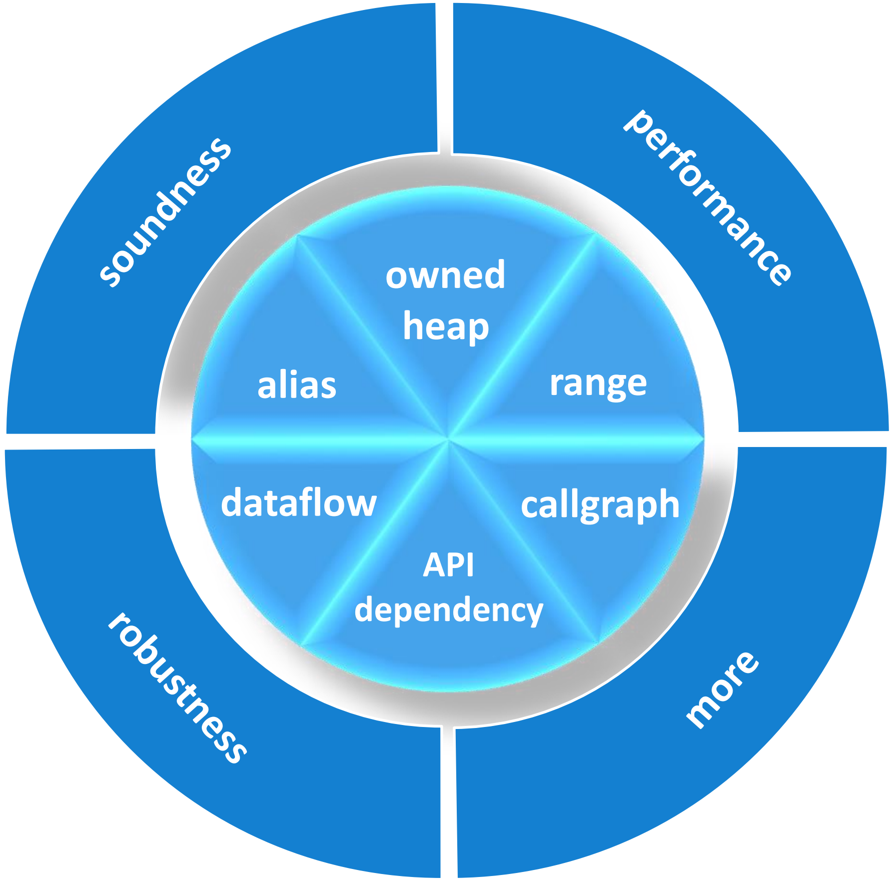
星绽操作系统
（OSTD）

大纲

- 一、Rust的关键问题：Unsafe代码
- 二、Unsafe代码治理：tag-std项目
- 三、Unsafe代码验证：RAPx项目

RAPx：Rust（静态）程序分析平台

- ❑目的：做rustc编译器“不能做”的事
- ❑特点：基础算法和应用分离，避免重复实现



集成多篇论文成果：

- 悬空指针：SafeDrop @ TOSEM；
- 内存泄漏：rCanary @ TSE
- 用例合成：RULF @ ASE 2021，RuMono @ TOSEM
- 安全属性：@ ICSE 2024；
- 内存预测：@ FSE 2023
- 性能优化：@ ISSRE 2025

项目地址：https://github.com/Artisan-Lab/RAPx

使用RAPx进行安全验证

第一步：使用tag-std标注安全属性

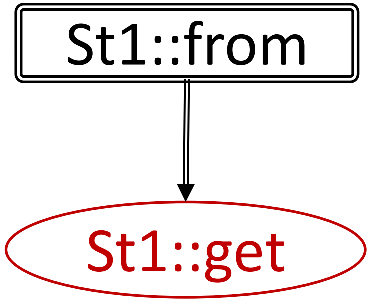
第二步：提取安全验证/审计单元

第三步：通过抽象解释进行验证

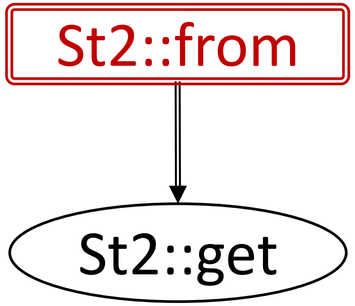
提取安全验证/审计单元：构造函数和方法的关系

❑如何声明安全性+标注安全属性？

```
struct St1 { ptr: *mut u8, len: usize }
impl St1 {
    pub fn from(p: *mut u8, l: usize) -> St1 {...}
    /// 标注什么？
    pub unsafe fn get(&self) -> &[u8] { unsafe {...} }
}
```

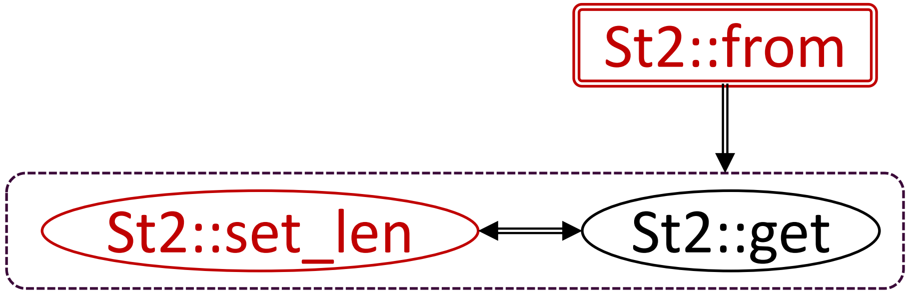



```
struct St2 { ptr: *mut u8, len: usize }
impl St2 {
    pub unsafe fn from(p: *mut u8, l: usize) -> St1 {...}
    pub fn get(&self) -> &[u8] { unsafe {...} }
}
```



提取安全验证/审计单元：方法之间的关系

❑调用一个方法可能影响其它方法的安全性

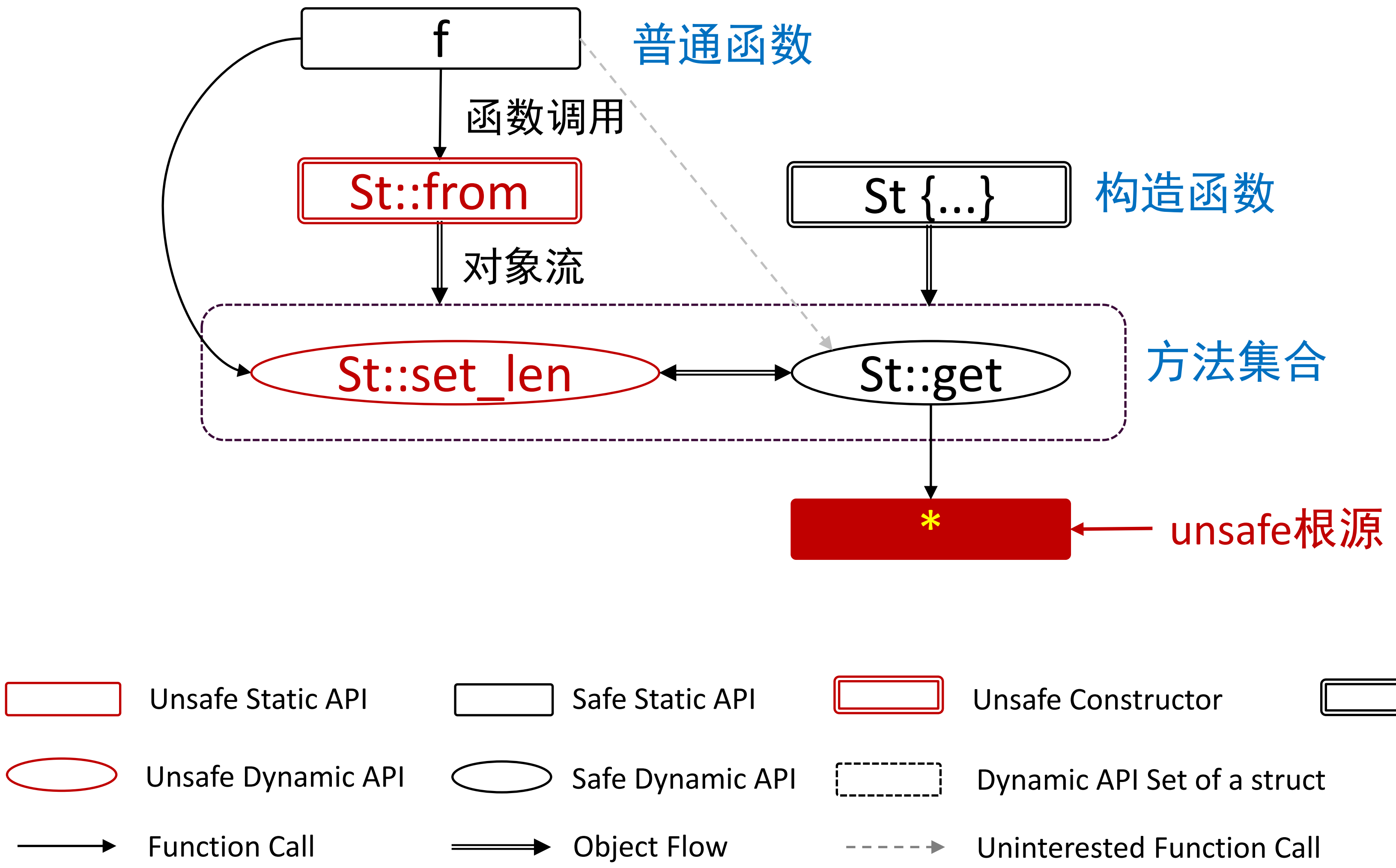



```
struct St2 { ptr: *mut u8, len: usize }

impl St2 {
    pub unsafe fn from(p: *mut u8, l: usize) -> St1 {...}
    pub fn get(&self) -> &[u8] { unsafe {...} }
    pub unsafe fn set_len(l: usize) {...}
}
```

步骤一：构建unsafety传导图

- 将所有unsafe函数的影响范围使用图表示



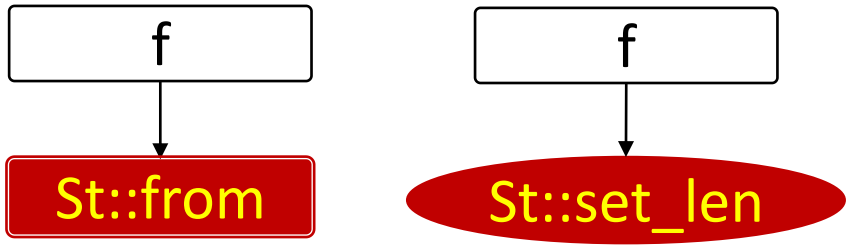
步骤二：提取验证单元

- ***RS:*** *Required Safety Property*
- ***VS:*** *Verified Safety Property*

单节点：应标注清楚安全属性

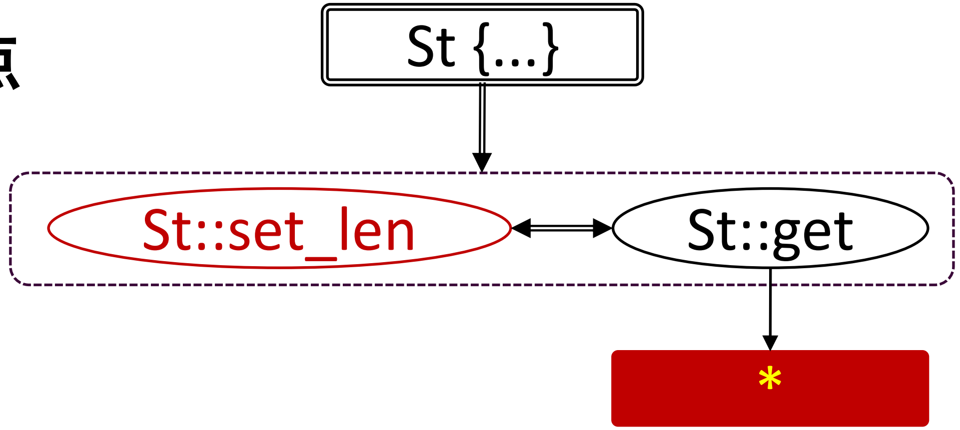
RS_{f_u} RS_{m_u}

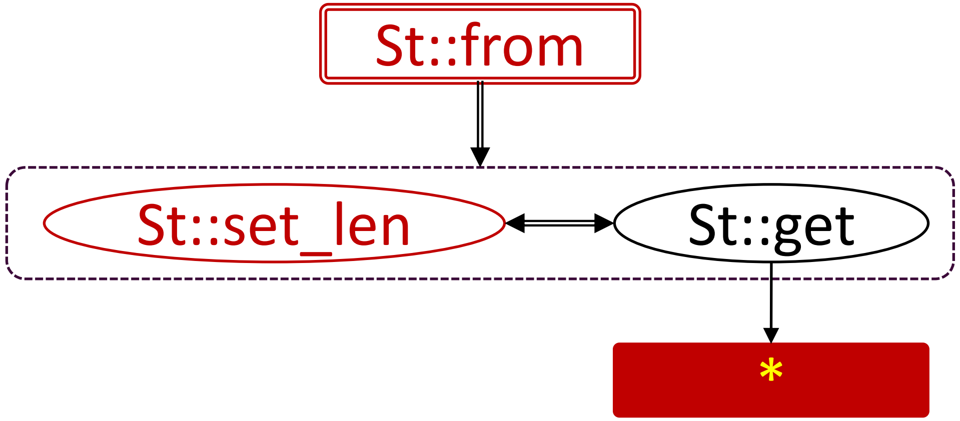
双节点



$RS_{c_u} \subseteq VS_{f_s}$ $RS_{m_u} \subseteq VS_{f_s}$

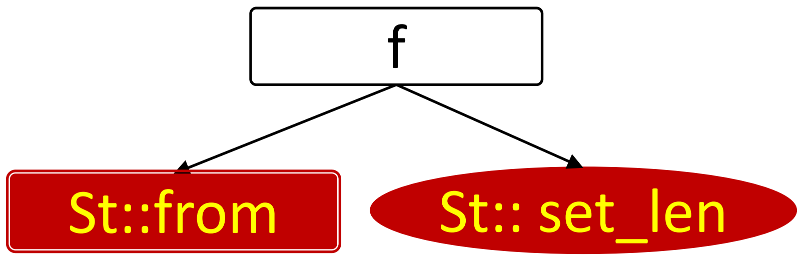
多节点

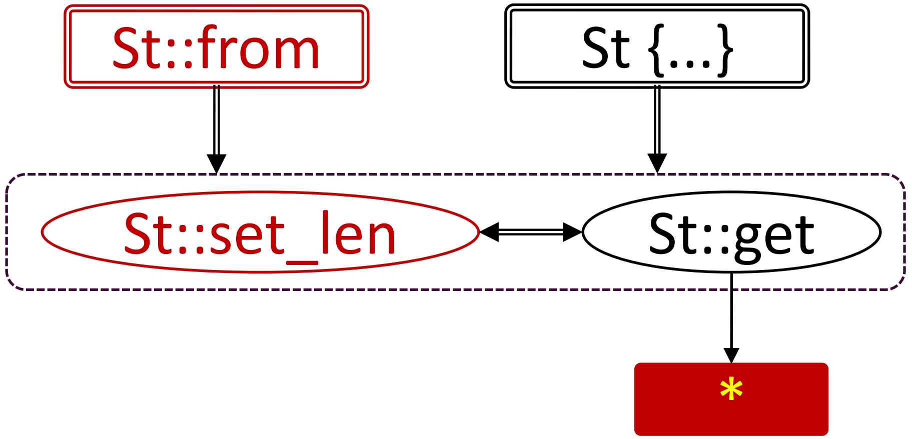




$RS_{f_u} \subseteq VS_{c_s} \cup VS_{m_s} - KS_M$ $RS_{f_u} \subseteq (RS_{c_u} \cup VS_{c_u}) \cup VS_{m_s} - KS_M$

步骤三：合并审计单元





$$RS_{c_u} \cup RS_{m_u} \subseteq VS_{f_s}$$

$$RS_{f_u} \subseteq ((RS_{c_u} \cup VS_{c_u}) \cap VS_{c_s}) \cup VS_{m_s} - KS_M$$

谢谢! Q&A