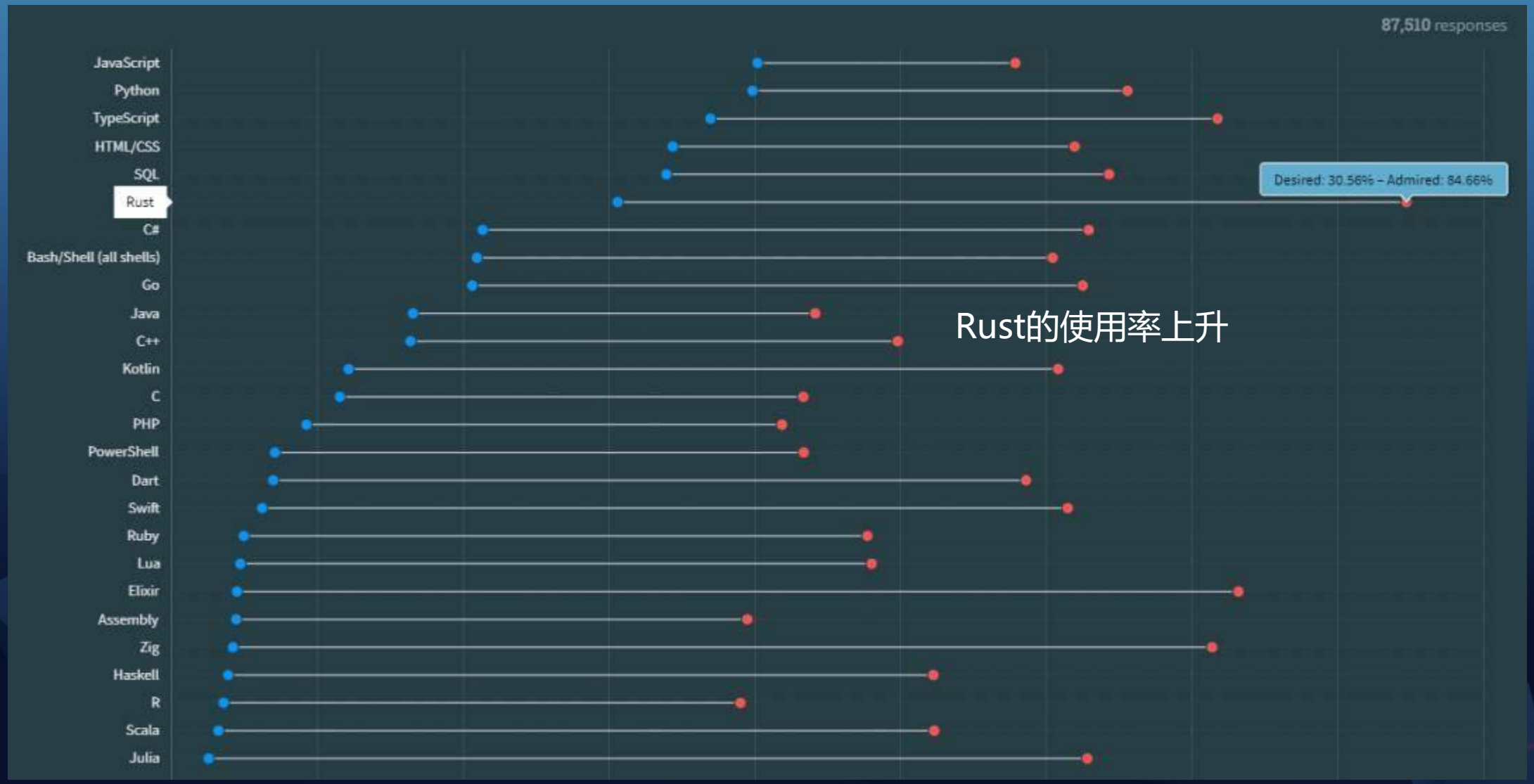
安全开发者峰会

Rust的安全幻影：语言层面的约束及其局限性

陈浩 奇安信 天工实验室

目录

目录


87,510 responses
JavaScript
Python
TypeScript
HTML/CSS
SQL
Rust
C#
Bash/Shell (all shells)
Go
Java
C++
Kotlin
C
PHP
PowerShell
Dart
Swift
Ruby
Lua
Elixir
Assembly
Zig
Haskell
R
Scala
Julia
Desired: 30.56% – Admired: 84.66%
Rust的使用率上升

Mark Russinovich

@markrussinovich · **Follow**

Speaking of languages, it's time to halt starting any new projects in C/C+
a non-GC langua
and reliability. th
languages as de

6:50 AM · Sep 20, 202

7.5K Repl

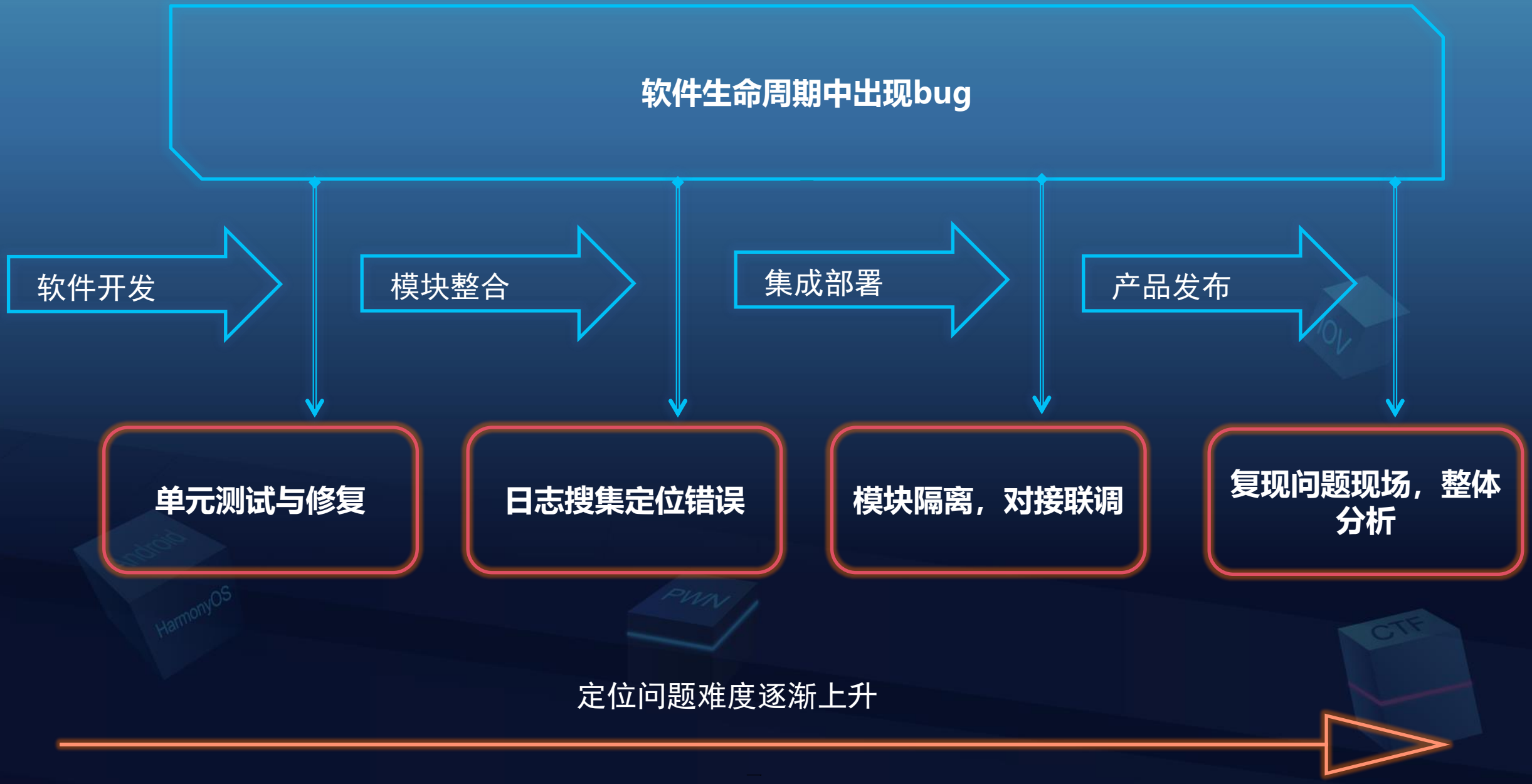
```
C:\Windows\System32>dir win32k*
 Volume in drive C has no label.
 Volume Serial Number is E60B-9A9E

 Directory of C:\Windows\System32

04/15/2023  09:50 PM           708,608 win32k.sys
04/15/2023  09:49 PM         3,424,256 win32kbase.sys
04/15/2023  09:49 PM           110,592 win32kbase_rs.sys
04/15/2023  09:50 PM         4,194,304 win32kfull.sys
04/15/2023  09:49 PM            40,960 win32kfull_rs.sys
04/15/2023  09:49 PM            69,632 win32kns.sys
04/15/2023  09:49 PM            98,304 win32ksgd.sys
               7 File(s)      8,646,656 bytes
               0 Dir(s)  116,366,049,280 bytes free
```

_rs = Rust!

Rust 安全防护

Rust 被誉为“专注于安全的编程语言”

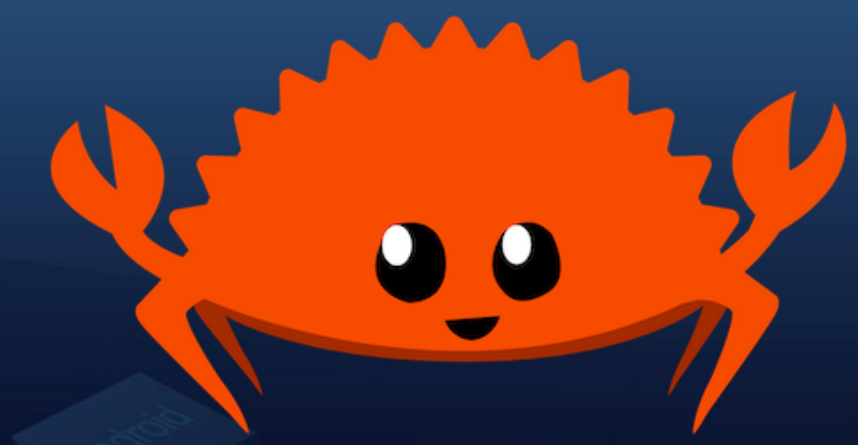
- 在编译阶段进行安全检查
- 尤其防护内存安全


软件生命周期中出现bug
软件开发
模块整合
集成部署
产品发布
单元测试与修复
日志搜集定位错误
模块隔离，对接联调
复现问题现场，整体分析
定位问题难度逐渐上升


软件生命周期中出现bug
软件开发
模块整合
继承部署
产品发布
单元测试与修复
日志搜集定位错误
模块隔离，对接联调
复现问题现场，整体分析
Rust 提供保护的时机

目录

Rust 安全防护



尽量在编译阶段解决大部分的安全问题

“只要能编译通过，就能解决绝大部分内存安全问题”

保证了运行时的效率

```rust
let s = String::from("Hello, Rust!");
let &mut my_struct;
// trigger drop structure

{
    let mut test = MyStruct::new(s);
    my_struct = &test;
}

println!("{:?}",my_struct);
```

Rust中一段常见可能发生UAF的代码

```
error[E0597]: `test` does not live long enough
  --> src/main.rs:14:21
   |
13 |         let mut test = MyStruct::new(s);
   |             -------- binding `test` declared here
14 |         my_struct = &test;
   |                     ^^^^^ borrowed value does not live long enough
15 |     }
   |     - `test` dropped here while still borrowed
16 |     // drop(my_struct);
17 |     println!("{:?}",my_struct);
   |                     --------- borrow later used here
```

编译器会及时检查错误

Rust 安全防护

Rust 提出了所有权的概念，在编译阶段进行检查

对象只能在一个时刻被一个变量引用

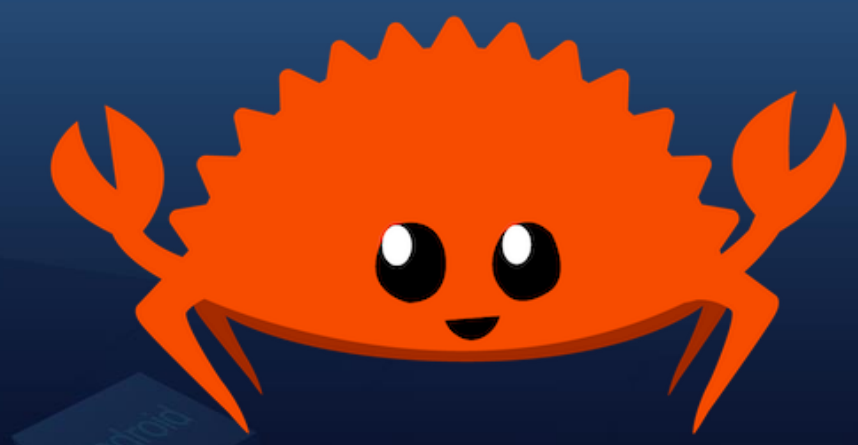
Rust 安全防护

支持高度抽象，保证接口的正确使用

合理使用泛型，闭包等特性，从设计层面杜绝漏洞

Rust 安全防护

静态无法检测的问题，使用动态检查确保安全



```rust
struct MyStruct {
    value: i32,
}

fn main() {
    let my_data = RefCell::new(MyStruct { value: 10 });
    let mut data1 = my_data.borrow_mut();
    let mut data2 = my_data.borrow_mut();
    data1.value += 10;
    data2.value += 20;
}
```

```
thread 'main' panicked at src/main.rs:10:29:
already borrowed: BorrowMutError
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

动态多次借用的检查

Rust 安全防护

即便对于发布的二进制库，依然能够提供保护机制

使用元数据机制

Rust 安全防护

编译期静态检查+动态运行时检查

最大限度地杜绝内存问题

最安全的编程语言

绝对不会有安全问题吗?

目录

Rust 安全边界与威胁

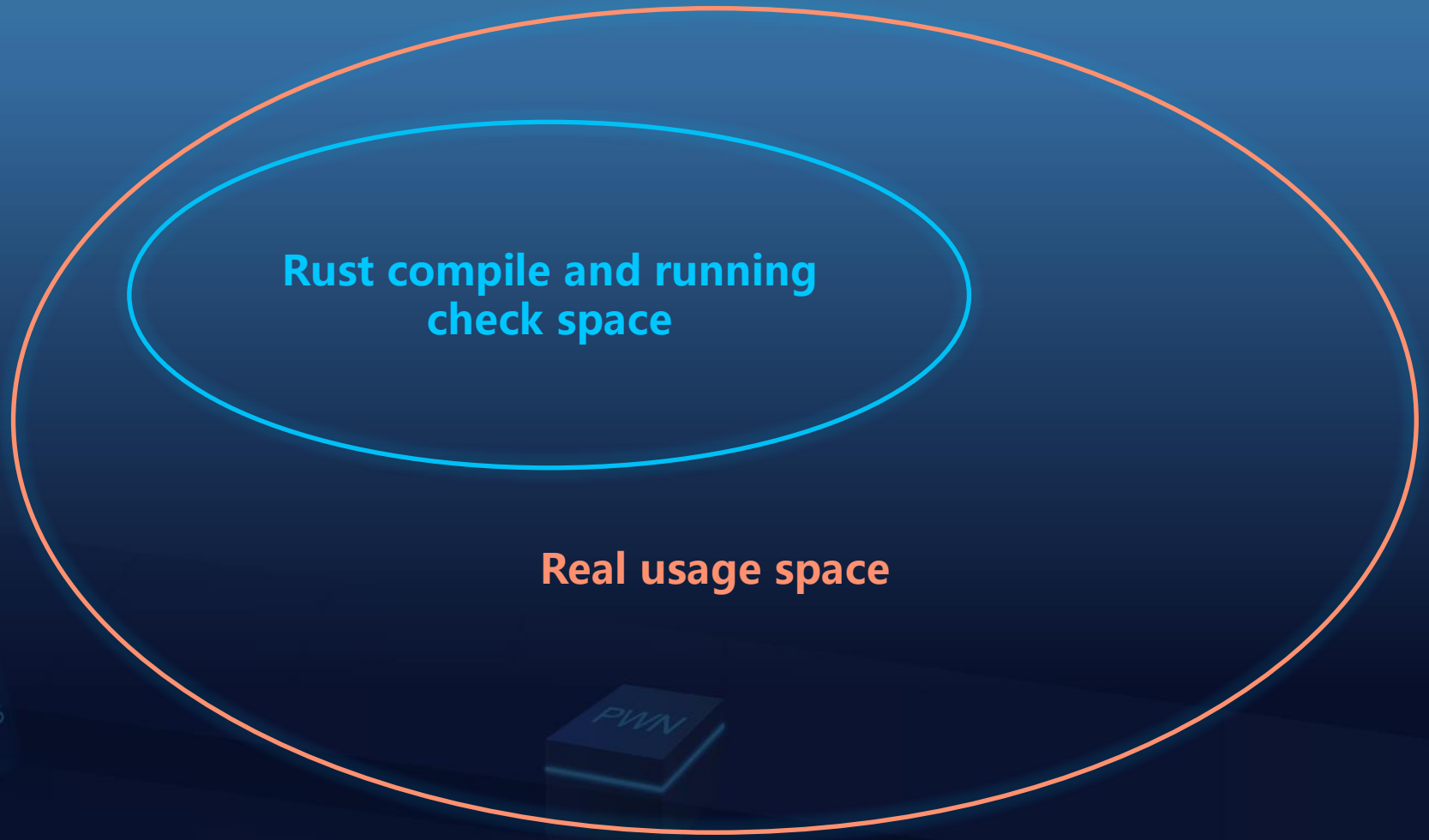
Rust 安全边界与威胁

并非指Rust本身存在安全问题

- 实际上，Rust确实无愧最安全的编程语言

这里讨论Rust的保护无法顾及的情况

【认知偏差】


Rust compile and running
check space
Real usage space

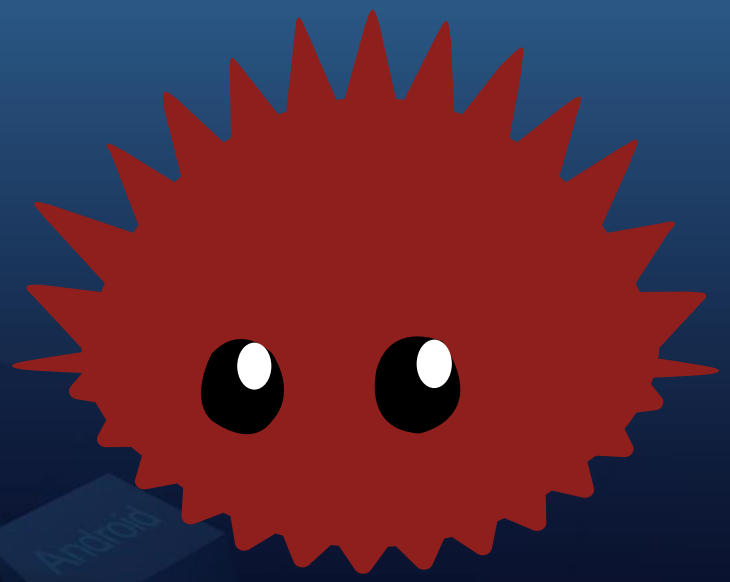
Rust 安全边界与威胁

编译期间会对所有权进行检查

运行期会完成对动态获取运行的结果检查

- 数组越界
- 整数溢出
- 悬挂引用
- etc

Rust 安全边界与威胁



运行时不再检查所有权

```rust
let s1 = String::from("hello");
let s2 = s1;
// 这里会报错
println!("{}, world!", s1);
```

所有权发生变更，String : s1 -> s2

二进制层面发生了什么?

```rust
let s1 = String::from("hello");
let s2 = s1;
// 这里会报错
println!("{}, world!", s1);
```

- S1 被当场销毁
- S2 为S1 新拷贝对象
- 从底层逻辑层面抑制UAF

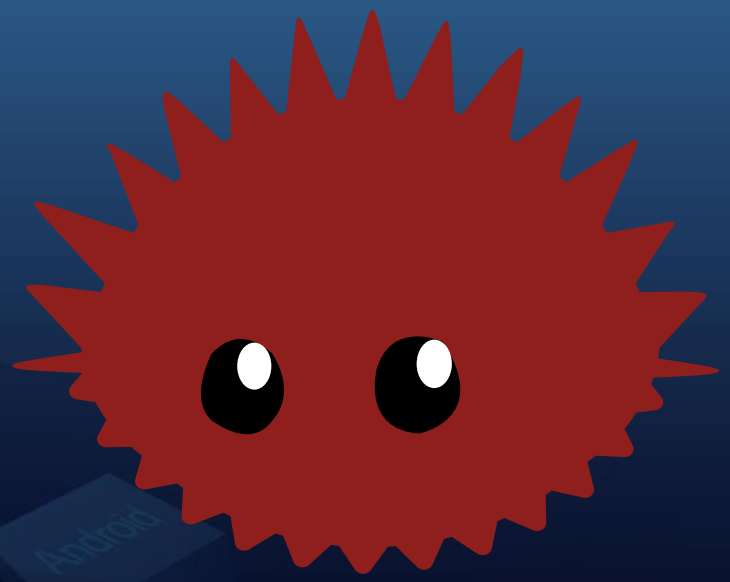
- S1 并未被销毁
- 实际上只存在一个变量
- 编译器不编译这类代码

```rust
let s1 = String::from("hello");
let s2 = s1;
// 这里会报错
println!("{}, world!", s1);
```

- S1 被当场销毁
- S2 为S1 新拷贝对象
- 从底层逻辑层面抑制UAF

- S1 并未被销毁
- 实际上只存在一个变量
- 编译器不编译这类代码

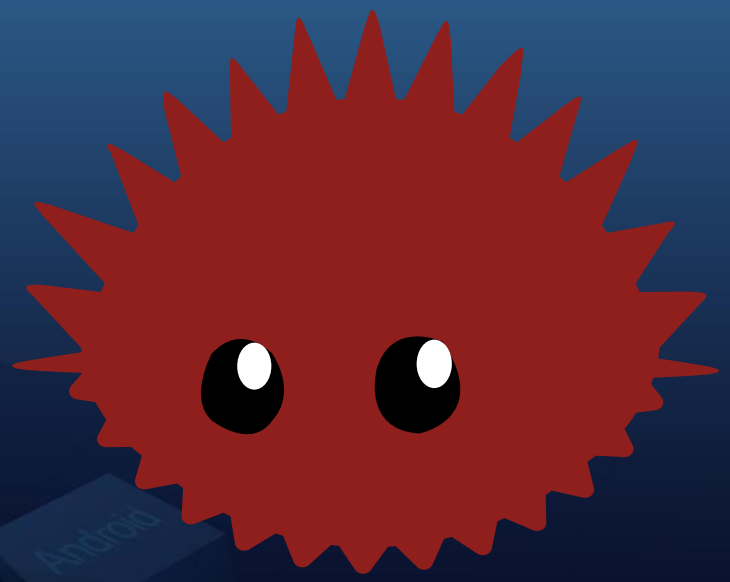
Rust 安全边界与威胁



用户使用的时候，更多关注的是功能

- 软件本身运行提供的功能
- 第三方库能力支持
- 底层操作能力
- etc

Rust 安全边界与威胁



Rust 无法保证安全的几种情况

- 作为API的场合（无法预知调用者行为）
- Rust 主动忽略的场合（unsafe）
- 逻辑处理问题（常见过滤错误等）

目录

CVE-2024-24576

BatBadBut

不至于直接导致RCE，所以不算最糟

由安全研究员RyotaK发现

影响众多编程语言，rust也不例外

CVE-2024-24576

```
CreateProcess(NULL, "test.bat 123");
```

```
cmd /c  “ “test.bat”  “123”  “
```

Windows CreateProcess 对脚本处理存在隐式行为

Windows支持直接运行.bat或者.cmd后缀的文件

会将运行指令转换成 cmd /c “ “each argument” ”

CVE-2024-24576

Windows 命令行中的 \ 并非Linux下的转义字符

- 仅可将回车和自身转义

Windows 等价的转义字符为^

CVE-2024-24576

Rust意识到了Windows的特性后，紧急进行了修复

然而特殊处理不够完善，添加\符号转义却不够完整

Rust

检测到参数为bat文件后，单独处理

```rust
pub(crate) fn make_bat_command_line(
    script: &[u16],
    args: &[Arg],
    force_quotes: bool,
) -> io::Result<Vec<u16>> {
    // Set the start of the command line to `cmd.exe /c "`
    // It is necessary to surround the command in an extra pair of quotes,
    // hence the trailing quote here. It will be closed after all arguments
    // have been added.
    let mut cmd: Vec<u16> = "cmd.exe /d /c \"".encode_utf16().collect();
    // skip some code
    for arg in args {
        cmd.push(' ' as u16);
        // Make sure to always quote special command prompt characters, including:
        // * Characters `cmd /?` says require quotes.
        // * `%` for environment variables, as in `%TMP%`.
        // * `|<>` pipe/redirect characters.
        const SPECIAL: &[u8] = b"\t &()[]{}^=;!'+,`~%|<>";
        let force_quotes = match arg {
            Arg::Regular(arg) if !force_quotes => arg.bytes().iter().any(|c| SPECIAL.contains(c)),
            _ => force_quotes,
        };
        append_arg(&mut cmd, arg, force_quotes)?;
```

参数拼接

```rust
fn append_arg()
    for x in arg.encode_wide() {
        if escape {
            if x == '\\' as u16 {
                backslashes += 1;
            } else {
                if x == '"' as u16 {
                    // Add n+1 backslashes to total 2n+1 before internal '"'.
                    cmd.extend((0..=backslashes).map(|_| '\\' as u16));
                }
                backslashes = 0;
            }
        }
        cmd.push(x);
    }
```

Rust

\符号无效防护

Rust

```
85900
cmd.exe /d /c ""D:\Daily\Rust\CVE\CVE-2024-24576\target\debug\test.bat" "\"&calc.exe""
```

```rust
// Loop through the string, escaping `\` only if followed by `"`.
    // And escaping `"` by doubling them.
    let mut backslashes: usize = 0;
    for x in arg.encode_wide() {
        if x == '\\' as u16 {
            backslashes += 1;
        } else {
            if x == '"' as u16 {
                // Add n backslashes to total 2n before internal `"`.
                cmd.extend((0..backslashes).map(|_| '\\' as u16));
                // Appending an additional double-quote acts as an escape.
                cmd.push(b'"' as u16);
            }
        }
    }
```

Rust

双引号防护

CVE-2024-24576

对操作系统特性理解出错导致的逻辑问题

- 超出编译器检测范围
- 属于【逻辑错误】


Rust 代码进行的参数检查
开发人员/用户对参数的认知

目录

RUSTSEC-2024-0346

Rust在内存布局中存在“潜规则”

- 结构体的顺序会发生重构
- 以最小空间占用作为依据

https://github.com/rust-lang/rust/pull/125360

RUSTSEC-2024-0346

```
struct test
{
  char a;
  int  b;
  short c;
};
```

C

12

```
struct TestStruct {
    field1: u8,
    field2: u32,
    field3: u16,
}
```

Rust

8

RUSTSEC-2024-0346

以#[repr] 字段指定当前结构体的行为

- #[repr(C)] 以C语言的理解方式进行排布
- #[repr(packed)] 以内存紧凑的方式进行排布（去掉padding）

RUSTSEC-2024-0346

Rust 的mod ZeroVec

- 能够方便的实现对各类切片引用
- ZeroVec能在Rust中辅助进行反序列化
- 【零拷贝】

RUSTSEC-2024-0346

```rust
use std::str::FromStr;

use icu_calendar::Date;
use icu_datetime::{options::length, DateFormatter};
use icu_locid::Locale;

fn main() {
    let locale = Locale::from_str("en-u-ca-japanese").unwrap();
    let formatter = DateFormatter::try_new_with_length(&locale.into(), length::Date::Full).unwrap();
    let date = Date::try_new_iso_date(2020, 5, 30).unwrap().to_any();
    dbg!(formatter.format_to_string(&date).unwrap());
}
```

POC

```rust
#[cfg_attr(feature = "serde", derive(serde::Deserialize))]
pub struct JapaneseErasV1<'data> {
    /// A map from era start dates to their era codes
    #[cfg_attr(feature = "serde", serde(borrow))]
    pub dates_to_eras: ZeroVec<'data, (EraStartDate, TinyStr16)>,
}
```

RUSTSEC-2024-0346

```rust
pub struct EraStartDate {
    /// The year the era started in
    pub year: i32,
    /// The month the era started in
    pub month: u8,
    /// The day the era started in
    pub day: u8,
}
```

大小：8

```rust
#[repr(transparent)]
#[derive(PartialEq, Eq, Ord, PartialOrd, Copy, Clone, Hash)]
pub struct TinyAsciiStr<const N: usize> {
    bytes: [AsciiByte; N],
}
```

大小：16

RUSTSEC-2024-0346

```rust
pub struct EraStartDate {
    /// The year the era started in
    pub year: i32,
    /// The month the era started in
    pub month: u8,
    /// The day the era started in
    pub day: u8,
}

// #[repr(C)]
#[repr(packed)]
// #[repr(packed,C)]
1 implementation
pub struct TestTuple(EraStartDate, [AsciiByte; 16]);
// pub struct TestTuple(EraStartDate, [u8; 16]);
```

```
Tuple Size: 24
EraStartDate Size: 8
EraStartDate Offset: 16
TinyAscii Size: 16
TinyAscii Offset: 0
```

- TestTuple 中有两个元素，EraStartDate 和 [AsciiByte]
- 当编译后运行，发现编译器将两个结构体成员顺序颠倒了
- Self.0 由 EraStartDate变成了 [AsciiByte]

RUSTSEC-2024-0346

Enforce C,packed, not just packed, on ULE types #5049

Merged **Manishearth** merged 1 commit into `unicode-org:main` from `Manishearth:cpacked` on Jun 17

Conversation 16 | Commits 1 | Checks 27 | Files changed 13

Changes from **all commits** | File filter | Conversations

Filter changed files

- components
  - calendar/src/provider
    - chinese_based.rs
    - islamic.rs
  - casemap/src/provider

2 components/calendar/src/provider/chinese_based.rs

```
@@ -143,7 +143,7 @@ impl<'data> ChineseBasedCacheV1<'data> {
143 143         derive(databake::Bake),
144 144         databake(path = icu_calendar::provider),
145 145     )]
146     -   #[repr(packed)]
    146 +   #[repr(C, packed)]
147 147     pub struct PackedChineseBasedYearInfo(pub u8, pub u8, pub u8);
148 148
```

Manishearth commented on Jun 14 • edited

Member

This broke ICU4X. It's our fault: we assumed `repr(packed)` meant `repr(C, packed)`, but this is an assumption I have seen quite commonly in the Rust ecosystem around networking and FFI code, since there has never really been a concept of "Rust packed" distinct from "C packed" till now, and 99% of code using packed will either be using it for networking (where they need `C, packed`, or for zero-copy stuff (where they need stability, which you get from `C, packed`)

So while I wouldn't go as far as to say it's a breaking change, I wonder if this should be done more carefully, perhaps requiring `packed` to be paired with `Rust` or `C`.

Or even having `repr(packed)` default to `C, packed` unless explicitly specified? `Rust, packed` does seem to be somewhat niche to me until `repr(Rust)` stabilizes.

(The fix for ICU4X is unicode-org/icu4x#5049)

RUSTSEC-2024-0346

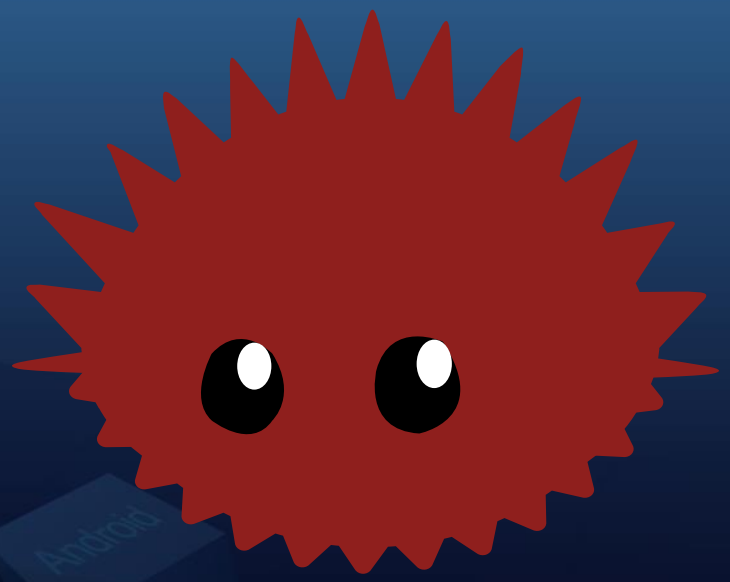
开发者对语言底层理解出错导致的逻辑问题

- 编译器认为是正确行为
- 处于【被忽略检查】的状态


Rust 设计接口的初衷
开发者对于特性的实际理解

目录

CVE-2024-27284

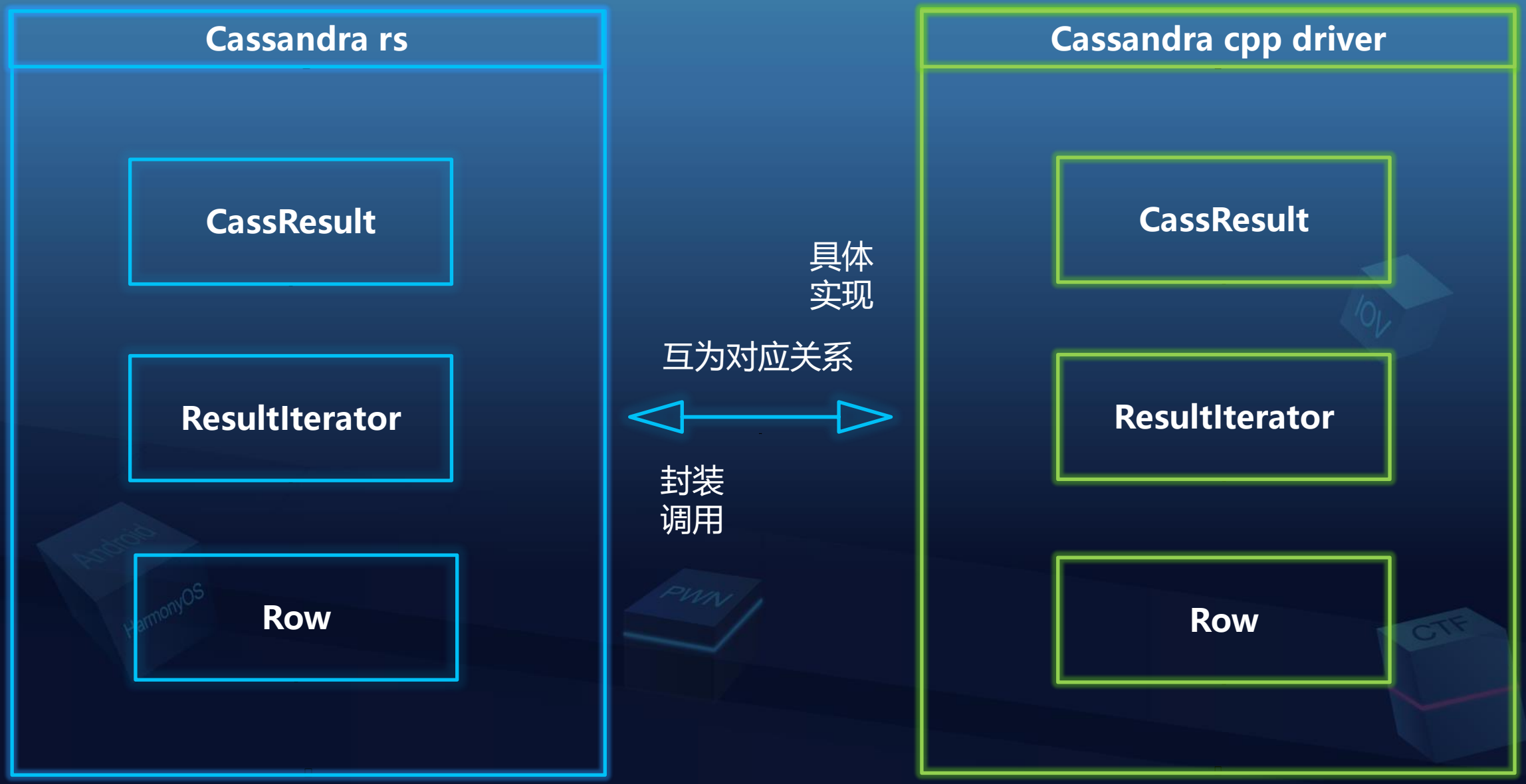


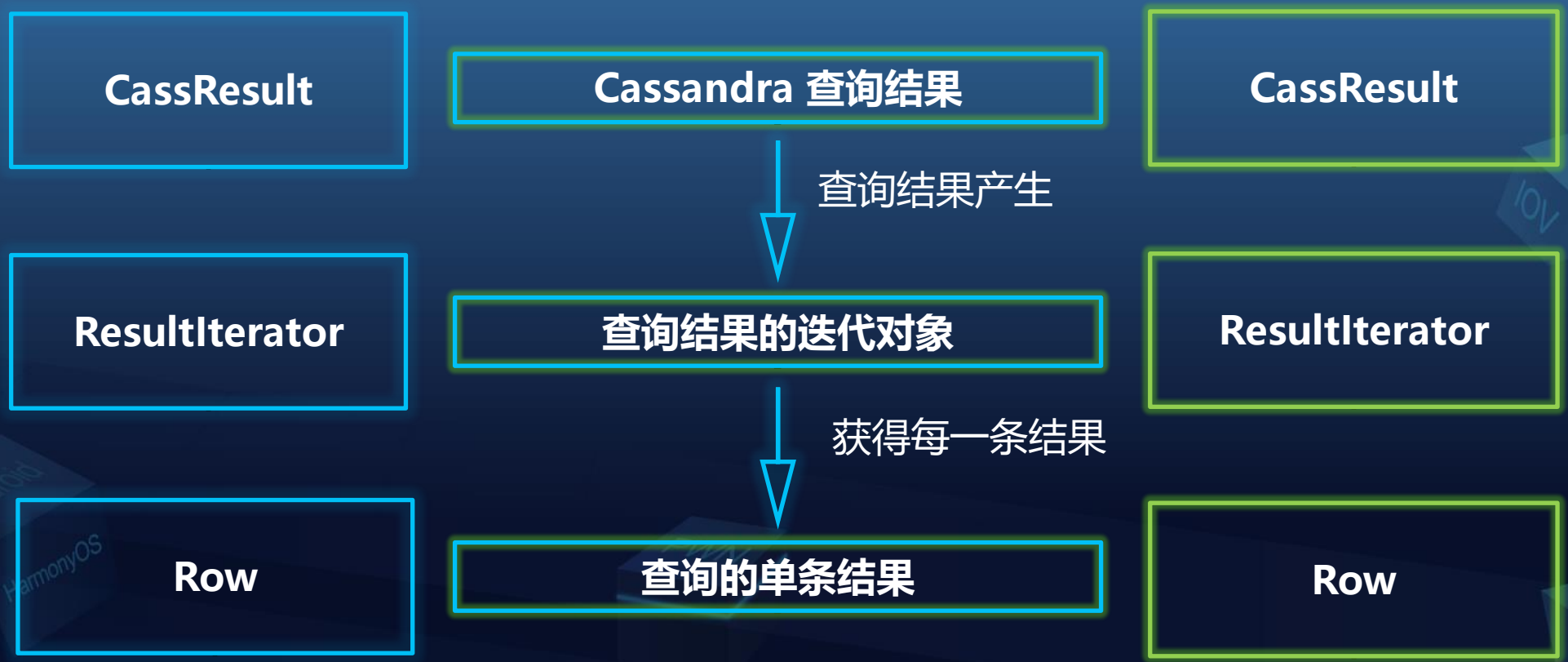
漏洞出现在Cassandra-rs 项目

- 分布式数据库
- 内部存在迭代器对象
- 存在unsafe block，对底层进行操作

问题本质为UAF漏洞

- 突破所有权管理（？）


Cassandra rs
CassResult
ResultIterator
Row
具体
实现
互为对应关系
封装
调用
Cassandra cpp driver
CassResult
ResultIterator
Row


CassResult
Cassandra 查询结果
CassResult
查询结果产生
ResultIterator
查询结果的迭代对象
ResultIterator
获得每一条结果
Row
查询的单条结果
Row

CVE-2024-27284

```
189         - impl<'a> Iterator for ResultIterator<'a> {
190         -     type Item = Row<'a>;
191         -     fn next(&mut self) -> Option<<Self as Iterator>::Item> {
        210 + impl LendingIterator for ResultIterator<'_> {
        211 +     type Item<'a> = Row<'a> where Self: 'a;
        212 +
        213 +     fn next(&mut self) -> Option<<Self as LendingIterator>::Item<'_>> {
192     214             unsafe {
193     215                 match cass_iterator_next(self.0) {
194     216                     cass_false => None,
195     217                     cass_true => Some(self.get_row()),
196     218                 }
197     219             }
198     220         }
199     221
```

审计Patch修复策略

CVE-2024-27284

```
189         - impl<'a> Iterator for ResultIterator<'a> {
190         -     type Item = Row<'a>;
191         -     fn next(&mut self) -> Option<<Self as Iterator>::Item> {
        210 + impl LendingIterator for ResultIterator<'_> {
        211 +     type Item<'a> = Row<'a> where Self: 'a;
        212 +
        213 +     fn next(&mut self) -> Option<<Self as LendingIterator>::Item<'_>> {
192     214             unsafe {
193     215                 match cass_iterator_next(self.0) {
194     216                     cass_false => None,
195     217                     cass_true => Some(self.get_row()),
196     218                 }
197     219             }
198     220         }
199     221
```

CVE-2024-27284

LendingIterator 本质实现了一种迭代器

可以将迭代器本身与迭代器对象生命周期关联

CVE-2024-27284

```
204  226
205      - impl<'a> ResultIterator<'a> {
206      -     /// Gets the next row in the result set
207      -     pub fn get_row(&mut self) -> Row<'a> {
         227  + impl ResultIterator<'_> {
         228  +     /// Gets the current row in the result set
         229  +     pub fn get_row(&self) -> Row {
208      230          unsafe { Row::build(cass_iterator_get_row(self.0)) }
209      231      }
210      232  }
```

此处提及的Row对象也发生变化

CVE-2024-27284

Row结构体中，包含一个指向cpp实现的_Row的裸指针，以及一个虚幻数据 PhantomData


```
29    /// A collection of column values. Read-only, so thread-safe.
    - pub struct Row<'a>(*const _Row, PhantomData<&'a CassResult>);
30  + //
31  + // Borrowed immutably.
32  + pub struct Row<'a>(*const _Row, PhantomData<&'a _Row>);
33
```

CVE-2024-27284

```
struct Tmp<'a>{
        index: &'a u32
    }
```

Rust无法声明结构体的生命周期，而是通过结构体本身成员来绑定声明生命周期

CVE-2024-27284

某些特定场合的结构体可能需要表明结构体的生命周期和某个对象强绑定

- Web 链接中的 WebContext 和其结构体中的 Session对象
- 迭代器Iterator以及其指向的对象Object
- etc

CVE-2024-27284

```rust
pub struct Test2<'a> {
    n1: u32,
    _marker: PhantomData<&'a Test1>,
}
```

Rust添加不占大小的PhantomData进行生命周期的声明

CVE-2024-27284

Row对象与CassResult 关联

```
29    /// A collection of column values. Read-only, so thread-safe.
-     pub struct Row<'a>(*const _Row, PhantomData<&'a CassResult>);
30  + //
31  + // Borrowed immutably.
32  + pub struct Row<'a>(*const _Row, PhantomData<&'a _Row>);
33
```

Row对象与_Row 关联

CVE-2024-27284

get_row 函数会返回一个Row::build的结果

来自C 语言层实现的cass_iterator_get_row，获取CassResult的提取结果

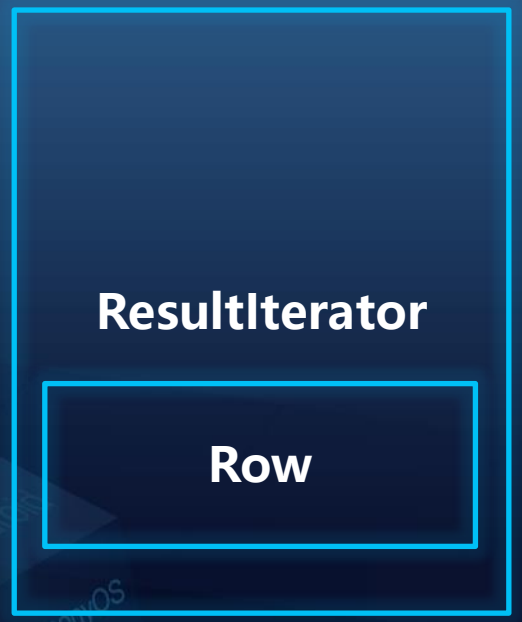
```
204  226
205      - impl<'a> ResultIterator<'a> {
206      -     /// Gets the next row in the result set
207      -     pub fn get_row(&mut self) -> Row<'a> {
         227 + impl ResultIterator<'_> {
         228 +     /// Gets the current row in the result set
         229 +     pub fn get_row(&self) -> Row {
208      230           unsafe { Row::build(cass_iterator_get_row(self.0)) }
209      231       }
210      232   }
```

CVE-2024-27284

- cass_iterator_get_row 返回iterator对象的row对象
- Iterator对象中包含完整Row对象

```
const CassRow* cass_iterator_get_row(const CassIterator* iterator) {
    if (iterator->type() != CASS_ITERATOR_TYPE_RESULT) {
      return NULL;
    }
    return CassRow::to(static_cast<const ResultIterator*>(iterator->from())->row());
  }

class ResultIterator : public Iterator {
  ResultIterator(const ResultResponse* result)
      , result_(result)
      , row_(result) {
      }  const Row* row() const {
    assert(index_ >= 0 && index_ < result_->row_count());
    if (index_ > 0) {
      return &row_;
    } else {
      return &result_->first_row();
    }
  }
  const ResultResponse* result_;
  Row row_;
};
```

CVE-2024-27284



结论1

ResultIterator和Row公用一段内存空间

CVE-2024-27284

```
189         - impl<'a> Iterator for ResultIterator<'a> {
190         -     type Item = Row<'a>;
191         -     fn next(&mut self) -> Option<<Self as Iterator>::Item> {
        210 + impl LendingIterator for ResultIterator<'_> {
        211 +     type Item<'a> = Row<'a> where Self: 'a;
        212 +
        213 +     fn next(&mut self) -> Option<<Self as LendingIterator>::Item<'_>> {
192     214             unsafe {
193     215                 match cass_iterator_next(self.0) {
194     216                     cass_false => None,
195     217                     cass_true => Some(self.get_row()),
196     218                 }
197     219             }
198     220         }
199     221
```

ResultIterator 和 Item 指代的Row<'a>类型生命周期长度一致

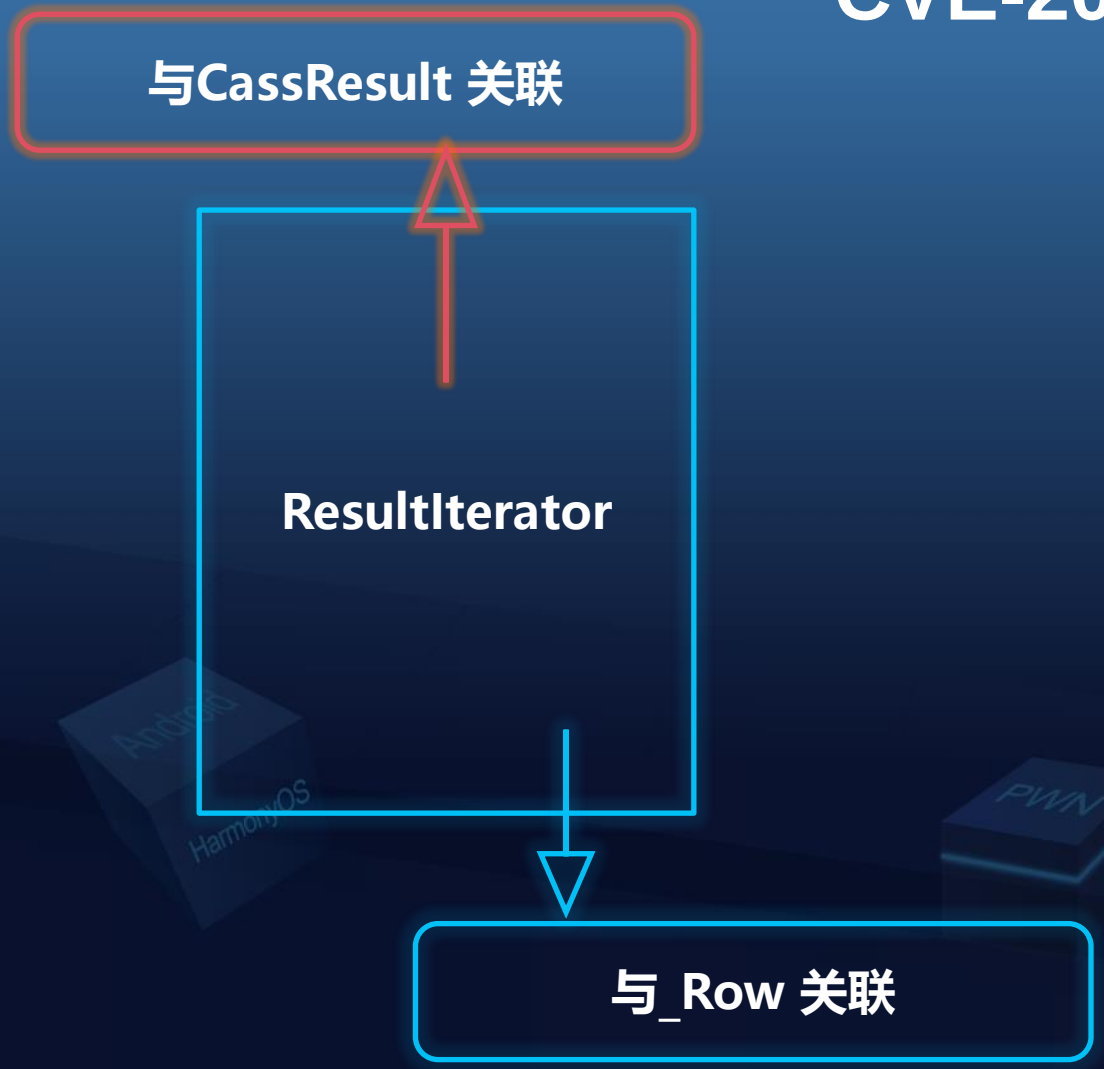
CVE-2024-27284

Row对象与CassResult 关联

```
29    /// A collection of column values. Read-only, so thread-safe.
-     pub struct Row<'a>(*const _Row, PhantomData<&'a CassResult>);
30  + //
31  + // Borrowed immutably.
32  + pub struct Row<'a>(*const _Row, PhantomData<&'a _Row>);
33
```

Row对象与_Row 关联

CVE-2024-27284



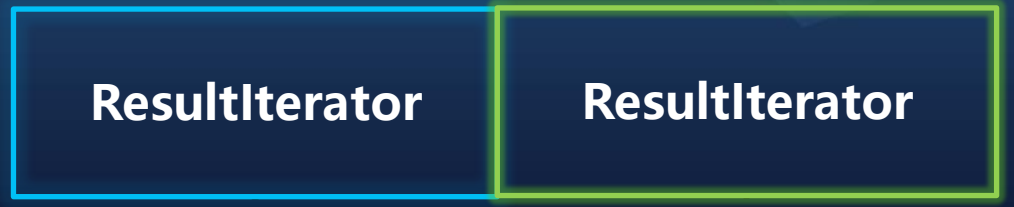
结论2

修复前 ResultIterator 关联CassResult的生命周期，

修复后ResultIterator 关联Row声明周期

CVE-2024-27284

修复前，CassResult和ResultIterator生命周期同步



修复后，CassResult和Row生命周期同步

CVE-2024-27284

```rust
impl CassResult {
    /// Gets the number of rows for the specified result.
    // ...
    /// Creates a new iterator for the specified result.
    /// This can be
    /// used to iterate over rows in the result.
    pub fn iter(&self) -> ResultIterator {
        unsafe {
            ResultIterator(
                cass_iterator_from_result(self.0),
                cass_result_row_count(self.0),
                PhantomData,
            )
        }
    }
}
```

```
  ResultIterator(const ResultResponse* result)
      : Iterator(CASS_ITERATOR_TYPE_RESULT)
      , result_(result)          // CaseResult pointer
      , index_(-1)
      , row_(result)             // CaseResult pointer
```

CassResult.iter 函数中，会尝试从CassResult，也就是中获取iterator对象，并且存入ResultIterator

结论3

迭代CassResult的时候，访问的是ResultIterator对象

ResultIterator 使用 result_指针指向CassResult

CVE-2024-27284

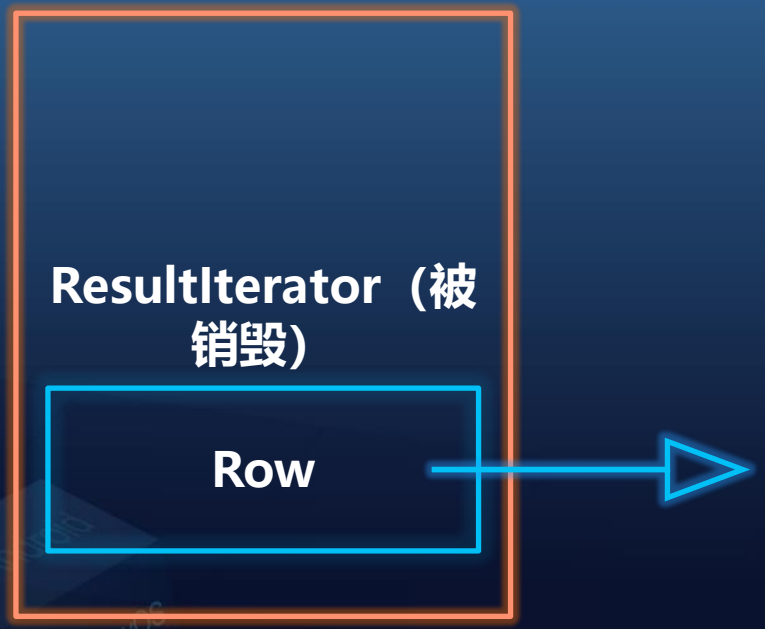
遍历查询结果CassResult，实际上在访问ResultIterator

ResultIterator 和 Row处在同一个内存空间

修复前没有显示的声明ResultIterator和Row的生命周期关系

CVE-2024-27284

CVE-2024-27284



即便被销毁，Row对象依然可以被使用（无法被Rust发现）

CVE-2024-27284

```
let mut tmp_row = None;
let result = function.get_result();
{
    for row in result.iter() {
        if condition.satisfied():
            tmp_row = Some(row)
            break;
    }
}
println!("here will cause problem {:?}", tmp_row);
```

当result.iter 离开循环，ResultIterator发生析构

Row因为生命周期未被声明，编译器未检查出问题

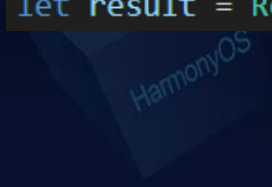
访问的时候会发生UAF

CVE-2024-27284

```
let result = ResultSet::new();
let mut taget_row = result.iter().get_row();
for row in result.iter() {
    taget_row = row;
    // println!("ks name = {}", col);
    break;
}

taget_row.visit();
println!("target iterator is {:?}", taget_row);
let result = ResultSet::new();
```

使用Demo模拟的当前模式的代码

复现出UAF的崩溃，并且无法被Rust运行时防护捕获



CVE-2024-27284

```
(1610.4508): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
rax=00000159ffffffff rbx=0000015996100000 rcx=0000000000000005
rdx=00007ff91e30d0a0 rsi=0000015996107ca0 rdi=0000015996107d30
rip=00007ff91e210240 rsp=0000000f8cd9f040 rbp=0000000f8cd9f339
 r8=0000015996109b90  r9=0000015996100320 r10=0000000000000003
r11=0000000f8cd9efd0 r12=0000000000000001 r13=0000000000000004
r14=0000015996107d20 r15=0000000000000003
iopl=0         nv up ei ng nz ac pe cy
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010293
ntdll!RtlpAllocateHeap+0x1130:
00007ff9`1e210240 488b08          mov     rcx,qword ptr [rax] ds:00000159`ffffffff=????????????????
```

CVE-2024-27284

错误的生命周期声明导致的逻辑问题

- 超出编译器检测范围
- 处于【API异常调用】范围


Rust 针对所有权的保护
未能正确保护的生命周期对象运行状态


Android
HarmonyOS

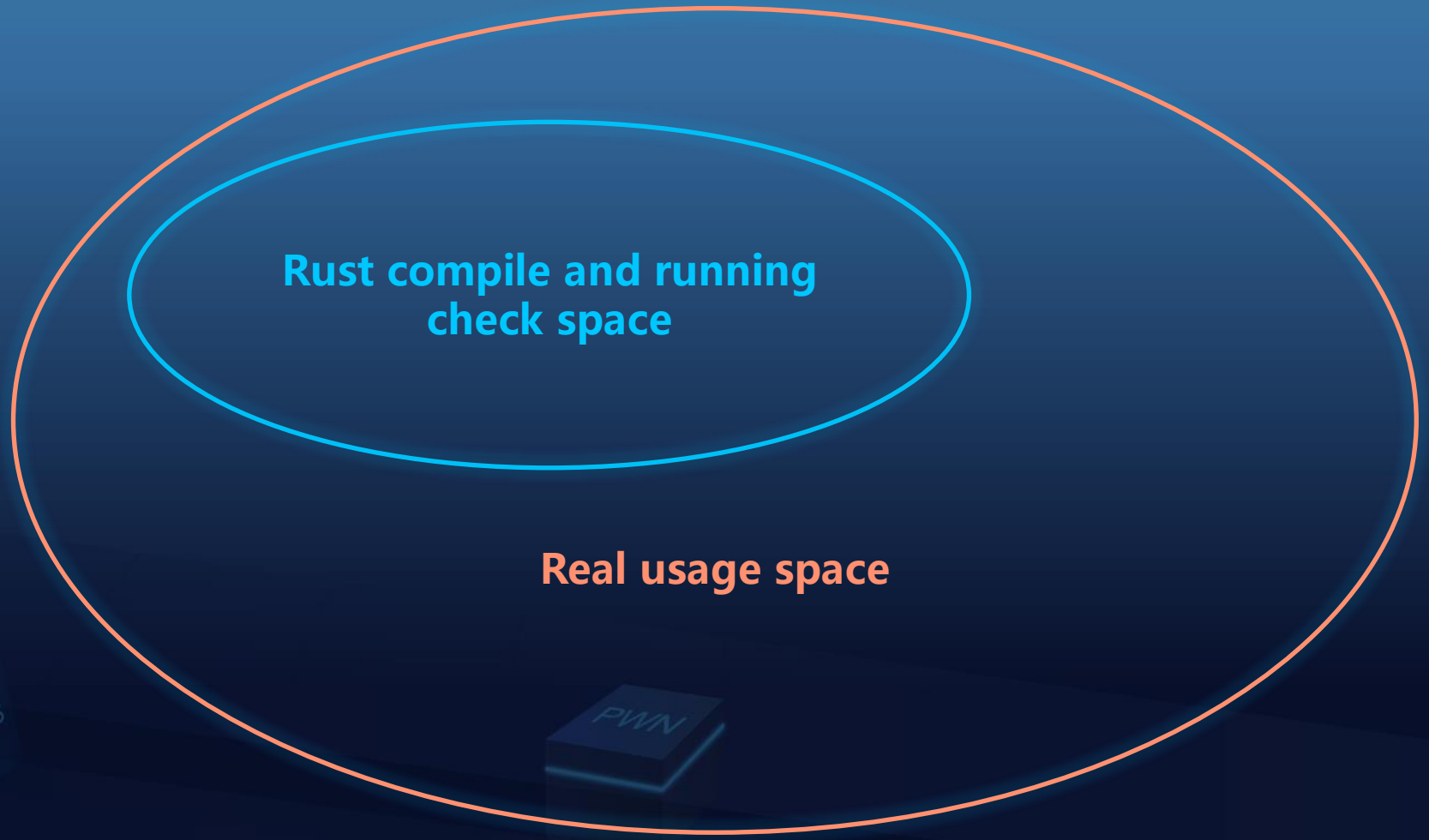
PWN


IOV


CTF

Rust 安全防护

总结相关


Rust compile and running
check space
Real usage space

Rust 安全防护

编译器检查范围仅发生在研发阶段

运行时检查用于防护特定错误

软件研发周期 ≠ 软件生命周期

Rust 安全防护

漏洞总结：

- 编译器检查范围外
- 特性类漏洞
- Unsafe 需要额外关注

感谢各位聆听