Rust & Kernel Linux

Nicola De Filippo 25/10/2025

Cosa vedremo

1. Problemi
2. Rust
3. Nel kernel
4. Come
5. Conclusioni

Problemi

- 2025 già centinaia di vulnerabilità
- CVE-2024-1086 (use-after-free nel netfilter)
- CVE-2024-53104 (out of bounds write)
- Tipologie frequenti: memory leaks, use-after-free, double-free, buffer overflow / out-of-bounds

Rust

- 2006 Graydon Hoare
- 2009
- 2012 - 2015
- 2015 - 2020
- Ad oggi

Rust aiutaci tu

- RAII (**R**esource **A**cquisition **I**s **I**nitialization)
- Borrow checker
- Traits
- Altro

IniCiamo

- Tutto ok! Sicuro, sicuro?

```
[nicola@archbox c]$ gcc main.c
[nicola@archbox c]$ ./a.out
hello world!
[nicola@archbox c]$
```

```c
#include <stdio.h>
#include <string.h>

int main() {

    char s[20] = "ciao";

    char *p1 = s;

    char *p2 = s;


    strcpy(p1, "hello");

    strcat(p2, " world!"); // scrive nella stessa memoria

    printf("%s\n", s);

    return 0;
}
```

Borrow checker :(

- cannot borrow `s` as mutable more than once at a time

```
nicola@archbox linuxday25]$ cargo build
   Compiling linuxday25 v0.1.0 (/home/nicola/src/rust/linuxday25)
error[E0499]: cannot borrow `s` as mutable more than once at a time
  --> src/main.rs:5:14
   |
 4 |     let r1 = &mut s;
   |              ------ first mutable borrow occurs here
 5 |     let r2 = &mut s;
   |              ^^^^^^ second mutable borrow occurs here
...
12 |     r1.push_str(" mondo");
   |     -- first borrow later used here

For more information about this error, try `rustc --explain E0499`.
error: could not compile `linuxday25` (bin "linuxday25") due to 1 previous error
[nicola@archbox linuxday25]$
```

```rust
fn main() {

    let mut s = String::from("ciao");

    let r1 = &mut s;

    let r2 = &mut s;  // <-- Errore

    r1.push_str(" mondo");

    r2.push_str("!");
}
```

Borrow checker :)

- Ora funziona

```
[nicola@archbox linuxday25]$ cargo build
   Compiling linuxday25 v0.1.0 (/home/nicola/src/rust/linuxday25)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 7.87s
[nicola@archbox linuxday25]$ target/debug/linuxday25
ciao mondo!
[nicola@archbox linuxday25]$
```

```rust
fn main() {
    let mut s = String::from("ciao");

    {
        let r1 = &mut s;
        r1.push_str(" mondo");
        // r1 esce dallo scope
    }

    let r2 = &mut s; // Ok!!
    r2.push_str("!");
    println!("{}", r2);
}
```

Data race

- Guai in vista

```c
// Dati e lock sono separati
struct driver_stats {
    int counter;
};

struct driver_data {
    struct driver_stats stats;
    raw_spinlock_t lock; // Il lock è un campo separato
};

// Dimentica di acquisire il lock
void update_stats(struct driver_data *data) {
    // data race! Manca spin_lock(&data->lock);
    data->stats.counter++;
    // Manca spin_unlock(&data->lock);
}
```

Soluzione

- Trait

```rust
// Il lock e i dati sono uniti
use crate::locking::SpinLock; // Un lock ottimizzato per il kernel

struct DriverStats {
    counter: u32,
}

// Struttura che incapsula i dati con un lock.
// Il compilatore assicura che questo tipo sia `Sync`.
struct DriverData {
    stats: SpinLock<DriverStats>, // SpinLock "contiene" i dati
}

// Il compilatore forza l'uso del lock
fn update_stats(data: &DriverData) {
    // 1. Accedere a `stats` restituisce un guard (RAII).
    // Questo guarda ACQUISCE il lock.
    let mut stats_guard = data.stats.lock();

    // 2. L'accesso ai dati è possibile SOLO tramite il guard.
    stats_guard.counter += 1;

    // 3. Quando `stats_guard` esce dallo scope, il trait `Drop` viene chiamato.
    // Il trait `Drop` RILASCIA AUTOMATICAMENTE il lock.
}
```

Nel kernel

- Linux 6.1
- Stato attuale
- Moduli kernel in-bound e out bound (drivers)

Esempi

- https://github.com/Rust-for-Linux/linux/blob/rust-next/rust/kernel/fs/file.rs
- https://github.com/Rust-for-Linux/linux/blob/rust-next/rust/kernel/alloc/allocator.rs

```rust
// SPDX-License-Identifier: GPL-2.0

// Copyright (C) 2024 Google LLC.

//! Files and file descriptors.
//!
//! C headers: [`include/linux/fs.h`](srctree/include/linux/fs.h) and
//! [`include/linux/file.h`](srctree/include/linux/file.h)

use crate::{
    bindings,
    cred::Credential,
    error::{code::*, to_result, Error, Result},
    fmt,
    sync::aref::{ARef, AlwaysRefCounted},
    types::{NotThreadSafe, Opaque},
};
use core::ptr;

/// Flags associated with a [`File`].
pub mod flags {
    /// File is opened in append mode.
    pub const O_APPEND: u32 = bindings::O_APPEND;

    /// Signal-driven I/O is enabled.
    pub const O_ASYNC: u32 = bindings::FASYNC;

    /// Close-on-exec flag is set.
    pub const O_CLOEXEC: u32 = bindings::O_CLOEXEC;

    /// File was created if it didn't already exist.
    pub const O_CREAT: u32 = bindings::O_CREAT;

    /// Direct I/O is enabled for this file.
    pub const O_DIRECT: u32 = bindings::O_DIRECT;

    /// File must be a directory.
    pub const O_DIRECTORY: u32 = bindings::O_DIRECTORY;

    /// Like [`O_SYNC`] except metadata is not synced.
    pub const O_DSYNC: u32 = bindings::O_DSYNC;
```

Out bound

```
[nicola@archbox rust-out-of-tree-module]$ make
make -C /lib/modules/`uname -r`/build M=$PWD
make[1]: Entering directory '/usr/lib/modules/6.16.10-arch1-1/build'
make[2]: Entering directory '/home/nicola/src/kernel/rust-out-of-tree-module'
make[2]: Leaving directory '/home/nicola/src/kernel/rust-out-of-tree-module'
make[1]: Leaving directory '/usr/lib/modules/6.16.10-arch1-1/build'
[nicola@archbox rust-out-of-tree-module]$ sudo dmesg
.....
[ 1259.465423] rust_out_of_tree: Rust out-of-tree sample (init)
```

```rust
//! Rust out-of-tree sample

use kernel::prelude::*;

module! {
    type: RustOutOfTree,
    name: "rust_out_of_tree",
    author: "Rust for Linux Contributors",
    description: "Rust out-of-tree sample",
    license: "GPL",
}

struct RustOutOfTree {
    numbers: KVec<i32>,
}

impl kernel::Module for RustOutOfTree {
    fn init(_module: &'static ThisModule) -> Result<Self> {
        pr_info!("Rust out-of-tree sample (init)\n");

        let mut numbers = KVec::new();
        numbers.push(72, GFP_KERNEL)?;
        numbers.push(108, GFP_KERNEL)?;
        numbers.push(200, GFP_KERNEL)?;

        Ok(RustOutOfTree { numbers })
    }
}

impl Drop for RustOutOfTree {
    fn drop(&mut self) {
        pr_info!("My numbers are {:?}\n", self.numbers);
        pr_info!("Rust out-of-tree sample (exit)\n");
    }
}
```

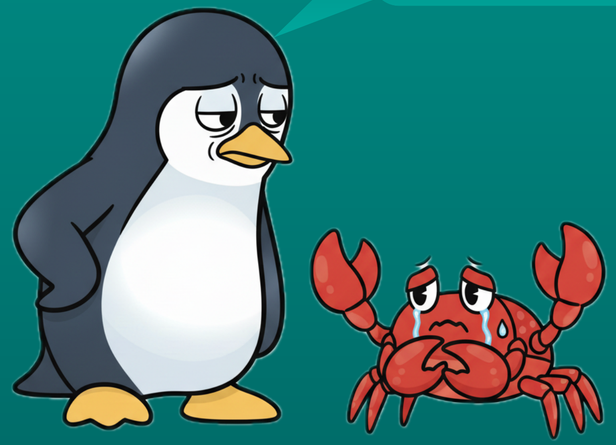
In sintesi

Problema	Rust
Leaks	Garantisce che la memoria allocata o i file aperti vengano sempre chiusi.
Deadlocks	Garantisce che un lock acquisito venga rilasciato.
Data race	Assicura che le operazioni di cleanup vengano eseguite in modo prevedibile.

Come iniziare

- Impara rust
- Quale distribuzione di GNU Linux?

Rust
Per oggi
Bast



Riferimenti

- https://www.technologyreview.com/2023/02/14/1067869/rust-worlds-fastest-growing-programming-language/
- https://stack.watch/product/linux/kernel/
- https://github.com/rust-for-linux
- https://rust-lang.org/it/
- https://github.com/google/comprehensive-rust
- https://github.com/mainmatter/100-exercises-to-learn-rust