Juan Gómez Mosquera

IBM **Research**

@Longor

Concurrencia en Rust

Sin miedo !!

Meetup

https://www.meetup.com/**MadRust**

IBM Research

@MadRustaceans

Qué vamos a ver hoy

- Introducción a Rust
- Concurrencia
- Herramientas

¿Qué es Rust?

- Previene seg-faults
- Es rapidísimo
- **Thread safety**

¿Por qué Rust?

- Soluciona un problema real:

Gestión manual de memoria => Problemas

IBM Research

¿Por qué Rust?

- Firefox

  ~ 35 millones de líneas de código

  C++

  Nuevos tiempos, más concurrencia

  Demasiado complejo

  Problemas graves de seguridad

¿Por qué Rust?

- Firefox

1	CVE-2018-5159	787	Overflow	2018-06-11	2018-10-20	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
An integer overflow can occur in the Skia library due to 32-bit integer use in an array without integer overflow checks, resulting in possible out-of-bounds writes. This could lead to a potentially exploitable crash triggerable by web content. This vulnerability affects Thunderbird < 52.8, Thunderbird ESR < 52.8, Firefox < 60, and Firefox ESR < 52.8.													
2	CVE-2018-5155	416		2018-06-11	2018-10-20	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
A use-after-free vulnerability can occur while adjusting layout during SVG animations with text paths. This results in a potentially exploitable crash. This vulnerability affects Thunderbird < 52.8, Thunderbird ESR < 52.8, Firefox < 60, and Firefox ESR < 52.8.													
3	CVE-2018-5154	416		2018-06-11	2018-10-20	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
A use-after-free vulnerability can occur while enumerating attributes during SVG animations with clip paths. This results in a potentially exploitable crash. This vulnerability affects Thunderbird < 52.8, Thunderbird ESR < 52.8, Firefox < 60, and Firefox ESR < 52.8.													
4	CVE-2018-5150	119	Overflow Mem. Corr.	2018-06-11	2018-10-20	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
Memory safety bugs were reported in Firefox 59, Firefox ESR 52.7, and Thunderbird 52.7. Some of these bugs showed evidence of memory corruption and we presume that with enough effort that some of these could be exploited to run arbitrary code. This vulnerability affects Thunderbird < 52.8, Thunderbird ESR < 52.8, Firefox < 60, and Firefox ESR < 52.8.													
5	CVE-2018-5148	416		2018-06-11	2018-08-09	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
A use-after-free vulnerability can occur in the compositor during certain graphics operations when a raw pointer is used instead of a reference counted one. This results in a potentially exploitable crash. This vulnerability affects Firefox ESR < 52.7.3 and Firefox < 59.0.2.													
6	CVE-2018-5147	787		2018-06-11	2018-08-14	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
The libtremor library has the same flaw as CVE-2018-5146. This library is used by Firefox in place of libvorbis on Android and ARM platforms. This vulnerability affects Firefox ESR < 52.7.2 and Firefox < 59.0.1.													
7	CVE-2018-5128	416		2018-06-11	2018-08-06	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
A use-after-free vulnerability can occur when manipulating elements, events, and selection ranges during editor operations. This results in a potentially exploitable crash. This vulnerability affects Firefox < 59.													

¿Por qué Rust?

- Servo

  Mozilla Research

  Rust

  Áltamente concurrente

  Seguro

  Integrándose con Firefox

¿Cómo lo hace?

- Ownership

    1. Sólo un propietario por valor

```
let v = vec![1,2,3,4];
let a = v;
println!("{:?}", v);
```

¿Cómo lo hace?

- Ownership

    1. Sólo un propietario por valor

```
let v = vec![1,2,3,4];
let a = v;
~~println!("{:?}", v);~~
```



```
error[E0382]: use of moved value: `v`
```

¿Cómo lo hace?

- Ownership

    1. Sólo un propietario por valor
    2. Si el propietario se sale de ámbito, se destruye el dato

```
fn func(){
    let heap = Box::new("Hola CommitConf");
}
```

¿Cómo lo hace?

- Borrowing (Referencias)
    1. Sólo un referencia mutable

```
let mut v = vec![1,2,3,4];
let a = &mut v;
let b = &mut v;
```

¿Cómo lo hace?

- Borrowing (Referencias)
    1. Sólo un referencia mutable

```rust
let mut v = vec![1,2,3,4];
let a = &mut v;
let b = &mut v;
```

```
error[E0499]: cannot borrow `v` as mutable more than once at a time
 --> src/main.rs:7:18
  |
6 |     let a = &mut v;
  |                 - first mutable borrow occurs here
7 |     let b = &mut v;
  |                 ^ second mutable borrow occurs here
```

¿Cómo lo hace?

- Borrowing (Referencias)
    1. Sólo un referencia mutable
    2. El tiempo de vida de una referencia nunca es mayor al del referenciado

```
let r : &Vec<i32>;
{
    let v = vec![1,2,3,4];
    r = &v;
}
println!("r = {:?}", r);
```

¿Cómo lo hace?

- Préstamos (Aliasing)
  1. Sólo un referencia mutable
  2. El tiempo de vida de una referencia nunca es mayor al del referenciado

```
let r : &Vec<i32>;
{
    let v = vec![1,2,3,4];
    r = &v;
}
println!("r = {:?}", r);
```

```
error[E0597]: `v` does not live long enough
```

Garantías

- Rust previene mutación + múltiples referencias

Garantías

- Rust previene mutación + múltiples referencias
- **Ownership** previene double-free

Garantías

- Rust previene mutación + múltiples referencias
- **Ownership** previene double-free
- **Borrowing** previene user-after-free

Garantías

- Rust previene mutación + múltiples referencias
- **Ownership** previene double-free
- **Borrowing** previene user-after-free
- **Garantías en tiempo de compilación**

Garantías

- Rust previene mutación + múltiples referencias
- **Ownership** previene double-free
- **Borrowing** previene user-after-free
- **Garantías en tiempo de compilación**

NO HAY CONDICIONES DE CARRERA EN DATOS

Concurrencia

Cuando varios cálculos se ejecutan simultáneamente, pudiendo interactuar entre ellos

Concurrencia no implica paralelismo

Paralelismo implica concurrencia

Concurrencia en Rust

Librerias

- Async/Await
- Futuros
- Corutinas
- ...

STD

- Threads
- ARC
- Atómicos
- Mutex

Lenguaje

- Trait Send
- Trait Sync

Concurrencia en Rust

Threads

```rust
use std::thread;

fn main(){
    let t = thread::spawn(||{
        println!("Hola CommitConf");
    });
    t.join();
}
```

Concurrencia en Rust

Threads

```
fn main(){
    let v = vec![1,2,3,4];
    let t = thread::spawn(||{
        println!("v = {:?}", v);
    });
    t.join();
}
```

Concurrencia en Rust

Threads

```
fn main(){
    let v = vec![1,2,3,4];
    let t = thread::spawn(||{
        ~~println!("v = {:?}", v);~~
    });
    t.join();
}
```

```
error[E0373]: closure may outlive the current function, but it borrows `v`,

to force the closure to take ownership of `v` (and any other referenced variables), use the `move` keyword
```

Concurrencia en Rust

Threads

```
fn main(){
    let v = vec![1,2,3,4];
    let t = thread::spawn(move||{
        println!("v = {:?}", v);
    });
    t.join();
}
```

Concurrencia en Rust

Threads

```
let v = vec![1,2,3,4];
for _ in 0..10 {
    thread::spawn(move||{
        println!("v = {:?}", v);
    });
}
```

Concurrencia en Rust

Threads

```
let v = vec![1,2,3,4];
for _ in 0..10 {
    thread::spawn(move||{
        ~~println!("v = {:?}", v);~~
    });
}
```

```
error[E0382]: capture of moved value: `v`
```

Concurrencia en Rust

std::sync::Arc

```
let v = Arc::new(vec![1,2,3,4]);
for _ in 0..10 {
    let v2 = v.clone();
    thread::spawn(move||{
        println!("v = {:?}", v2);
    });
}
```

Concurrencia en Rust

std::sync::Mutex

```
let num = Mutex::new(0);
{
    let mut guard = num.lock().unwrap();
    *guard = 5;
}
println!("{:?}", num);
```

Concurrencia en Rust

std::sync::{Arc, Mutex}

```
let v = Arc::new(Mutex::new(vec![]));
for i in 0..10 {
    let v2 = v.clone();
    thread::spawn(move||{
        let mut guard = v2.lock().unwrap();
        guard.push(i);
    });
}
```

Concurrencia en Rust

std::sync::atomic

```
let number = AtomicUsize::new(10);
let prev = number.fetch_add(1, SeqCst);
assert_eq!(prev, 10);
let prev = number.swap(2, SeqCst);
assert_eq!(prev, 11);
assert_eq!(number.load(SeqCst), 2);
```

Concurrencia en Rust

std::sync::mpsc

```
let (tx, rx) = mpsc::channel();

thread::spawn(move || {
    let text = "Hola";
    tx.send(text).unwrap();
});

let received = rx.recv().unwrap();
println!("{} CommitConf", received);
```

Concurrencia en Rust

std::sync::mpsc

```rust
let (tx, rx) = mpsc::channel();

for i in 0..10 {
    let tx2 = tx.clone();
    thread::spawn(move || {
        let text = format!("Hola, soy {}", i);
        tx2.send(text).unwrap();
    });
}

drop(tx);

for msg in rx {
    println!("Received = {}", msg);
}
```

Traits **Send** y **Sync**

- **Send** => ***transferencias*** entre hilos de forma segura
- **Sync** => ***referencias*** entre hilos de forma segura

`std::rc::Rc` **no** implementa **Send** y **Sync**
`std::sync::Arc` **sí**

Tokio

- La mejor librería para Async I/O
- Basada en Futuros
- Soporta HTTP/2 , HTTP, WebSockets, TCP, UDP, Unix Sockets, …

Rayon

- Tareas de uso intensivo de CPU
- Convierte iteradores secuenciales en paralelos

```
let v: Vec<u32> = (0..1_000).collect();
let sum_of_squares: u32 = v.par_iter()
    .map(|i| i * i)
    .sum();
```