

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY  
**SCHOOL OF INFORMATION AND COMMUNICATION  
TECHNOLOGY**



# **BÁO CÁO PROJECT 2**

**Chủ đề: Ngôn ngữ Rust và Bảo mật**

Giảng viên hướng dẫn: Nguyễn Đức Toàn

Sinh viên: Phạm Đặng Tấn Dũng

Mã số sinh viên: 20225569

Ngày 25 tháng 3 năm 2025

# Mục lục

<b>1</b>	<b>Week 1</b>	<b>4</b>
1.1	Rust Basics . . . . .	4
1.1.1	Variables in Rust . . . . .	4
1.1.2	Constants in Rust . . . . .	4
1.1.3	Shadowing in Rust . . . . .	4
1.1.4	Data Types in Rust . . . . .	5
1.1.5	Console Input in Rust . . . . .	5
1.1.6	Arithmetic and Type Casting in Rust . . . . .	5
1.1.7	Condition If Else in Rust . . . . .	5
1.1.8	Functions in Rust . . . . .	6
1.1.9	Expressions and Statements in Rust . . . . .	6
1.1.10	Heap and Stack in Rust . . . . .	6
1.2	Types of Attacks . . . . .	6
1.2.1	Phases of an Attack . . . . .	7
1.2.2	Profiles of Attackers . . . . .	7
1.2.3	Rust Error Handling . . . . .	8
1.2.4	Reading Files in Rust . . . . .	8
1.2.5	Lifetime Annotation . . . . .	9
1.2.6	Rc (Reference Counting) . . . . .	9
1.2.7	Arc (Atomic Reference Counting) . . . . .	9
<b>2</b>	<b>Week 2</b>	<b>11</b>
2.1	Multi-threading port scanning program . . . . .	11
2.1.1	Giới thiệu . . . . .	11
2.1.2	Dependencies . . . . .	11
2.2	Main execution thread (main.rs) . . . . .	11
2.3	Details . . . . .	14
2.3.1	subdomains::enumerate(subdomains.rs) . . . . .	14
2.3.2	subdomains::resolves (subdomains.rs) . . . . .	15
2.3.3	ports::scan_ports (ports.rs) . . . . .	16
2.3.4	ports::scan_port (ports.rs) . . . . .	16

2.4 Conclusion . . . . . 17

# Chương 1

## Week 1

### 1.1 Rust Basics

#### 1.1.1 Variables in Rust

Biến trong Rust là bất biến (immutable) theo mặc định, nghĩa là không thể thay đổi sau khi được gán giá trị. Để khai báo biến có thể thay đổi, ta dùng từ khóa `mut`.

```
1 let x = 5;    // immutable
2 let mut y = 10; // mutable
3 y = 15;      // ok
```

#### 1.1.2 Constants in Rust

Hằng số trong Rust được khai báo bằng từ khóa `const` và phải có kiểu dữ liệu rõ ràng. Giá trị của hằng số không thể thay đổi.

```
1 const MAX_POINTS: u32 = 100_000;
```

#### 1.1.3 Shadowing in Rust

Shadowing cho phép khai báo lại một biến với cùng tên, che giấu biến cũ.

```
1 let x = 5;
2 let x = x + 1; // x is 6
```

## 1.1.4 Data Types in Rust

Rust có các kiểu dữ liệu cơ bản như số nguyên (i32, u32), số thực (f32, f64), boolean (bool), và ký tự (char). Ngoài ra còn có tuple và array.

```
1 let integer: i32 = 42;
2 let float: f64 = 3.14;
3 let boolean: bool = true;
4 let character: char = 'a';
5 let tuple: (i32, f64, char) = (500, 6.4, 'b');
6 let array: [i32; 3] = [1, 2, 3];
```

## 1.1.5 Console Input in Rust

Để nhận input từ console, ta sử dụng `std::io::stdin`.

```
1 use std::io;
2 let mut input = String::new();
3 io::stdin().read_line(&mut input).expect("Failed to read line")
  );
```

## 1.1.6 Arithmetic and Type Casting in Rust

Rust hỗ trợ các phép toán số học cơ bản. Type casting được thực hiện bằng từ khóa `as`.

```
1 let sum = 5 + 10;
2 let product = 4 * 30;
3 let quotient = 56.7 / 32.2;
4 let remainder = 43 % 5;
5 let casted = 5 as f64;
```

## 1.1.7 Condition If Else in Rust

Câu lệnh điều kiện trong Rust tương tự như các ngôn ngữ khác.

```
1 let number = 3;
2 if number < 5 {
3     println!("condition was true");
4 } else {
5     println!("condition was false");
6 }
```

### 1.1.8 Functions in Rust

Hàm trong Rust được khai báo bằng từ khóa `fn`. Hàm có thể trả về giá trị.

```
1 fn add(x: i32, y: i32) -> i32 {  
2     x + y  
3 }
```

### 1.1.9 Expressions and Statements in Rust

Expressions trả về giá trị, trong khi statements không. Trong Rust, khối mã có thể là expression.

```
1 let y = {  
2     let x = 3;  
3     x + 1 // expression, no semi-colon  
4 };
```

### 1.1.10 Heap and Stack in Rust

Stack lưu trữ dữ liệu có kích thước cố định, trong khi heap lưu trữ dữ liệu có kích thước động. Rust quản lý bộ nhớ heap thông qua ownership.

```
1 // Stack  
2 let s = 5; // i32, default  
3  
4 // Heap  
5 let s = String::from("hello"); // String, dynamic size
```

## 1.2 Types of Attacks

Cyberattacks là các hành vi tấn công vào hệ thống máy tính hoặc mạng nhằm gây hại, đánh cắp thông tin, hoặc làm gián đoạn dịch vụ. Dưới đây là các loại tấn công phổ biến:

- **Tấn công không có mục tiêu rõ ràng:** Thường do thanh thiếu niên tò mò thực hiện, không có mục đích cụ thể nhưng vẫn gây thiệt hại tài chính.
- **Tấn công chính trị:** Nhằm truyền tải thông điệp chính trị, ví dụ: thay đổi nội dung website (defacement) hoặc tấn công từ chối dịch vụ (DoS).

- **Pentest (Penetration Testing):** Kiểm tra bảo mật hệ thống bằng cách mô phỏng tấn công, đôi khi chỉ để đáp ứng yêu cầu tuân thủ mà không thực sự hiệu quả.
- **Red Team:** Mô phỏng tấn công thực tế với phạm vi rộng hơn (như phishing, xâm nhập vật lý) để đánh giá khả năng phòng thủ.
- **Bug Bounty:** Chương trình thưởng cho việc phát hiện và báo cáo lỗ hổng bảo mật, tuy nhiên đôi khi chỉ mang tính hình thức.
- **Tội phạm mạng (Cybercrime):** Bao gồm đánh cắp dữ liệu, ransomware, gian lận thẻ tín dụng; nổi bật với vụ rò rỉ công cụ NSA năm 2017.
- **Gián điệp công nghiệp:** Đánh cắp bí mật thương mại để giành lợi thế cạnh tranh.
- **Chiến tranh mạng (Cyberwar):** Tấn công mạng trong chiến tranh, ví dụ: worm Stuxnet – vũ khí số đầu tiên trên thế giới.

### 1.2.1 Phases of an Attack

Một cuộc tấn công mạng thường trải qua các giai đoạn sau:

- **Reconnaissance (Thu thập thông tin):** Tìm hiểu về mục tiêu qua dữ liệu công khai (passive) hoặc quét mạng trực tiếp (active).
- **Exploitation (Khai thác):** Xâm nhập ban đầu bằng cách sử dụng lỗ hổng (exploits) hoặc kỹ thuật xã hội (social engineering).
- **Lateral Movements (Di chuyển bên trong):** Duy trì truy cập và mở rộng quyền hạn trong hệ thống, thường bằng các công cụ như RATs (Remote Access Tools).
- **Data Exfiltration (Trích xuất dữ liệu):** Đánh cắp dữ liệu từ hệ thống, cần thực hiện cẩn thận để tránh bị phát hiện.
- **Clean Up (Dọn dẹp):** Xóa dấu vết (logs, file tạm, cơ sở hạ tầng) để tránh bị truy ra.

### 1.2.2 Profiles of Attackers

Những kẻ tấn công có thể thuộc nhiều nhóm với vai trò khác nhau trong một đội tấn công:

- **The Hacker:** Người có kỹ năng tấn công cao, chịu trách nhiệm thu thập thông tin và khai thác lỗ hổng.

- **The Exploit Writer:** Phát triển công cụ và exploits để xâm nhập hệ thống.
- **The Developer:** Xây dựng các công cụ tùy chỉnh (credential dumpers, proxies, implants) để tránh bị phát hiện.
- **The System Administrator:** Quản lý và bảo mật cơ sở hạ tầng của kẻ tấn công, hỗ trợ trong các giai đoạn khai thác và di chuyển.
- **The Analyst:** Cung cấp kiến thức chuyên môn để phân tích dữ liệu và ưu tiên mục tiêu.

### 1.2.3 Rust Error Handling

Trong Rust, error handling được thực hiện một cách rõ ràng và an toàn, thường sử dụng kiểu dữ liệu `Result<T, E>` để biểu diễn kết quả có thể thành công hoặc thất bại. Ví dụ:

```
1 fn main() -> Result<(), Box<dyn std::error::Error>> {
2     // Code maybe return error
3     Ok(())
4 }
```

Trong đó, `Box<dyn std::error::Error>` cho phép trả về bất kỳ loại lỗi nào implement trait `std::error::Error`.

### 1.2.4 Reading Files in Rust

Đọc file trong Rust có thể được thực hiện bằng cách sử dụng `std::fs::File` và `std::io::BufReader`. Ví dụ:

```
1 use std::fs::File;
2 use std::io::{BufRead, BufReader};
3
4 let file = File::open("filename.txt")?;
5 let reader = BufReader::new(file);
6 for line in reader.lines() {
7     println!("{}", line?);
8 }
```



## 1.2.5 Lifetime Annotation

Lifetime annotation trong Rust được sử dụng để chỉ định phạm vi mà các tham chiếu hợp lệ, đảm bảo rằng các tham chiếu không tồn tại lâu hơn dữ liệu mà chúng trỏ đến. Ví dụ:

```
1 fn longest<'a>(x: &'a str, y: &'a str) -> &'a str {  
2     if x.len() > y.len() { x } else { y }  
3 }
```

Trong ví dụ này, 'a là lifetime annotation chung cho cả hai tham chiếu đầu vào và đầu ra.

Trong Rust, lifetime annotation được sử dụng để chỉ định phạm vi mà các tham chiếu hợp lệ, đảm bảo rằng các tham chiếu không tồn tại lâu hơn dữ liệu mà chúng trỏ đến, từ đó ngăn chặn các tham chiếu lơ lửng. Tuy nhiên, việc sử dụng quá nhiều lifetime annotation có thể làm cho mã nguồn trở nên phức tạp và khó đọc hơn, đặc biệt đối với người mới học. Hơn nữa, trong nhiều trường hợp, trình biên dịch có thể tự động suy ra lifetime, do đó không cần thiết phải ghi chú rõ ràng. Vì vậy, nên hạn chế sử dụng lifetime annotation và chỉ dùng khi thực sự cần thiết, chẳng hạn trong các tình huống phức tạp mà trình biên dịch không thể suy ra đúng lifetime.

## 1.2.6 Rc (Reference Counting)

Rc (Reference Counting) là một smart pointer trong Rust cho phép chia sẻ quyền sở hữu của dữ liệu giữa nhiều phần của chương trình. Nó đếm số lượng tham chiếu đến dữ liệu và tự động giải phóng khi không còn tham chiếu nào. Ví dụ mã giả:

```
1 use std::rc::Rc;  
2  
3 let data = Rc::new(5);  
4 let clone1 = Rc::clone(&data);  
5 let clone2 = Rc::clone(&data);  
6 println!("{}", *clone1); // 5
```

## 1.2.7 Arc (Atomic Reference Counting)

Arc (Atomic Reference Counting) tương tự như Rc nhưng an toàn cho việc sử dụng trong môi trường đa luồng. Nó sử dụng atomic operations để đếm số lượng tham chiếu. Ví dụ mã giả:

```
1 use std::sync::Arc;
```

```
2 use std::thread;
3
4 let data = Arc::new(5);
5 let clone1 = Arc::clone(&data);
6 thread::spawn(move || {
7     println!("{}", *clone1); // 5
8 });
```

# Chương 2

## Week 2

### 2.1 Multi-threading port scanning program

#### 2.1.1 Giới thiệu

Trong tuần 2, em sẽ tập trung vào khả năng đa luồng của Rust và viết chương trình quét cổng đa luồng. Một chương trình quét cổng tương tự đã được viết và so sánh để làm nổi bật tốc độ khi sử dụng đa luồng trong Rust.

#### 2.1.2 Dependencies

```
1 [dependencies]
2 thiserror = "1.0"
3 anyhow = "1.0"
4 rayon = "1.5"
5 trust-dns-resolver = "0.21"
6 request = { version = "0.11", default-features = false,
7             features = ["json", "blocking", "rustls-tls"] }
8 serde = { version = "1", features = ["derive"] }
```

Listing 2.1: Dependencies

### 2.2 Main execution thread (main.rs)

#### 1. Lấy đối số dòng lệnh:

```

1 let args: Vec<String> = env::args().collect();
2
3 if args.len() != 2 {
4     return Err(Error::CliUsage.into());
5 }
6
7 let target = args[1].as_str();
8

```

Listing 2.2: Lấy đối số dòng lệnh (main.rs)

- `env::args().collect()`: Thu thập các đối số dòng lệnh vào một vector `args`.
- `if args.len() != 2`: Kiểm tra số lượng đối số.
- `let target = args[1].as_str();`: Lấy tên miền mục tiêu.

## 2. Tạo HTTP client:

```

1 let http_timeout = Duration::from_secs(5);
2 let http_client = Client::builder()
3     .redirect(redirect::Policy::limited(4))
4     .timeout(http_timeout)
5     .build()?;
6

```

Listing 2.3: Tạo HTTP client (main.rs)

- `Duration::from_secs(5)`: Tạo `Duration` 5 giây.
- `Client::builder()`: Bắt đầu xây dựng `request::blocking::Client`.
- `.redirect(redirect::Policy::limited(4))`: Cấu hình chuyển hướng tối đa 4 lần (Có thể do trang web đã được di chuyển, máy chủ muốn sử dụng phiên bản HTTPS thay vì HTTP, cần đăng nhập trước khi xem nội dung, URL ban đầu là một URL rút gọn..).
- `.timeout(http_timeout)`: Đặt thời gian chờ 5 giây.
- `.build()?`: Hoàn tất và trả về `Result`.

## 3. Tạo thread pool (rayon):

```

1 let pool = rayon::ThreadPoolBuilder::new()
2     .num_threads(256)
3     .build()
4     .unwrap();

```

### Listing 2.4: Tạo thread pool (main.rs)

- `rayon::ThreadPoolBuilder::new()`: Bắt đầu xây dựng `rayon::ThreadPool`.
- `.num_threads(256)`: Cấu hình 256 luồng.
- `.build().unwrap()`: Hoàn tất và trả về `ThreadPool`.

#### 4. Thực hiện quét (trong thread pool):

```

1 pool.install(|| {
2     let scan_result: Vec<Subdomain> = subdomains::
      enumerate(&http_client, target)
3         .unwrap()
4         .into_par_iter()
5         .map(ports::scan_ports)
6         .collect();
7 });
8

```

### Listing 2.5: Thực hiện quét (main.rs)

- `pool.install(|| ... )`: Đăng ký closure để thực thi trong thread pool.
- `subdomains::enumerate(&http_client, target).unwrap()`: Lấy danh sách subdomain.
- `.into_par_iter()`: Sử dụng parallel iterator.
- `.map(ports::scan_ports)`: Áp dụng `scan_ports` cho từng subdomain (song song).
- `.collect()`: Thu thập kết quả vào `Vec<Subdomain>`.

#### 5. In kết quả:

```

1     for subdomain in scan_result {
2         println!("{}", &subdomain.domain);
3         for port in &subdomain.open_ports {
4             println!("{}", port.port);
5         }
6         println!();
7     }

```

### Listing 2.6: In kết quả (main.rs)

## 2.3 Details

### 2.3.1 subdomains::enumerate(subdomains.rs)

#### 1. Truy vấn crt.sh:

```
1 let entries: Vec<CrtShEntry> = http_client
2   .get(&format!("https://crt.sh/?q=%25.{}&output=json",
3     target))
4   .send()?
5   .json()?;
```

Listing 2.7: Truy vấn crt.sh (subdomains.rs)

- `http_client.get(...)`: Tạo yêu cầu GET đến `crt.sh`.
- `.send()?`: Gửi yêu cầu.
- `.json()?`: Phân tích phản hồi JSON thành `Vec<CrtShEntry>`.

#### 2. Xử lý và lọc dữ liệu:

```
1   let mut subdomains: HashSet<String> = entries
2     .into_iter()
3     .flat_map(|entry| {
4       entry
5         .name_value
6         .split('\n')
7         .map(|subdomain| subdomain.trim()).
8     to_string())
9     .collect::<Vec<String>>()
10    .filter(|subdomain: &String| subdomain !=
11      target)
12    .filter(|subdomain: &String| !subdomain.
13      contains('*'))
14    .collect();
15    subdomains.insert(target.to_string());
```

Listing 2.8: xử lý và lọc (subdomains.rs)

- `entries.into_iter()`: Chuyển vector `entries` thành iterator.
- `.flat_map(...)`: Biến đổi mỗi `CrtShEntry` thành một iterator.

- `entry.name_value.split('\n')`: Tách `name_value`.
- `.map(|subdomain| subdomain.trim().to_string())`: Loại bỏ khoảng trắng, chuyển String
- `.collect()`: Đưa các subdomain vào vector
- `.filter(|subdomain: &String| subdomain != target)`: Loại bỏ tên miền.
- `.filter(|subdomain: &String| !subdomain.contains('*'))`: Loại bỏ subdomain chứa `*`.
- `.collect()`: Thu thập vào `HashSet<String>`.

### 3. Tạo `Vec<Subdomain>` và kiểm tra DNS:

```

1      let subdomains: Vec<Subdomain> = subdomains
2          .into_iter()
3          .map(|domain| Subdomain {
4              domain,
5              open_ports: Vec::new(),
6          })
7          .filter(resolves)
8          .collect();
9

```

Listing 2.9: Tạo `Vec<Subdomain>` và kiểm tra DNS(subdomains.rs)

## 2.3.2 subdomains::resolves (subdomains.rs)

### 1. Cấu hình và tạo DNS resolver:

```

1      let mut opts = ResolverOpts::default();
2      opts.timeout = Duration::from_secs(4);
3      let dns_resolver = Resolver::new(ResolverConfig::
4          default(), opts).expect("resolver");

```

Listing 2.10: Cấu hình và tạo(subdomains.rs)

### 2. Thực hiện tra cứu DNS:

```

1      dns_resolver.lookup_ip(domain.domain.as_str()).is_ok()
2

```

Listing 2.11: Thực hiện tra cứu(subdomains.rs)

### 2.3.3 ports::scan\_ports (ports.rs)

#### 1. Tạo SocketAddr:

```
1 let socket_addresses: Vec<SocketAddr> = format!("{}", 1024,
    subdomain.domain)
2     .to_socket_addrs()
3     .expect("port scanner: Creating socket address")
4     .collect();
5
6 if socket_addresses.is_empty() {
7     return subdomain;
8 }
9
```

Listing 2.12: Tạo SocketAddr(ports.rs)

#### 2. Quét các cổng (song song):

```
1 subdomain.open_ports = MOST_COMMON_PORTS_100
2     .into_par_iter()
3     .map(|port| scan_port(socket_addresses[0], *port))
4     .filter(|port| port.is_open)
5     .collect();
6
```

Listing 2.13: quét cổng(ports.rs)

### 2.3.4 ports::scan\_port (ports.rs)

#### 1. Đặt cổng và thử kết nối:

```
1 let timeout = Duration::from_secs(3);
2 socket_address.set_port(port);
3
4 let is_open = TcpStream::connect_timeout(&socket_address,
    timeout).is_ok();
5
```

Listing 2.14: Đặt cổng(port.rs)

#### 2. Trả về Port:

```
1 Port { port, is_open }
```



### Listing 2.15: trả về(port.rs)

## 2.4 Conclusion

Qua thực nghiệm, thời gian thực hiện việc quét một địa chỉ khi sử dụng đa luồng khá nhanh(khoảng 3-5s) trong khi nếu thực thi trên một luồng duy nhất, thời gian khoảng 10 phút(tất nhiên còn phụ thuộc vào việc đặt thời gian chờ đợi phản hồi). Tuy nhiên, việc lập trình đa luồng giúp giảm đáng kể tốc độ thời gian thực thi chương trình.

Tuy nhiên việc mỗi luồng vẫn phải chờ đợi thời gian trả về của request khiến chương trình chưa đạt tốc độ thực thi tối đa. Tuần 3 sẽ nghiên cứu về áp dụng async/await(lập trình bất đồng bộ) cải thiện tốc độ của chương trình.

# Tài liệu tham khảo

- [1] Sylvain Kerkour, *Black Hat Rust*.
- [2] Tech With Tim, *Rust Programming Tutorial*.
- [3] Let's Get Rusty, *Rust Survival Guide*.

Link mã nguồn ở: [Project 2 Rust](#)