**HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY**

School of Information, Communication and Technology

\*\*\*\*\*

A picture containing text, sign

Description automatically generated

**FINAL PROJECT REPORT**

**Front-running attack in EVM blockchain**

***Course: Network Security***

***Group 12***

**Instructor:** ***PhD. Nguyễn Đức Toàn***

**Team member:** Lê Đình Huy 20194776

Phạm Minh Đăng 20194737

Đặng Minh Khôi 20194781

**Mục lục**

Contents

[Yêu cầu của project 4](#_Toc155061445)

[Mở đầu 5](#_Toc155061446)

[Thành viên nhóm và nhiệm vụ 6](#_Toc155061447)

[I. Blockchain là gì 7](#_Toc155061448)

[**1. Khái niệm về blockchain** 7](#_Toc155061449)

[**2. Khái niệm về transaction** 8](#_Toc155061450)

[**2.1. Khái niệm** 8](#_Toc155061451)

[**2.2. Một transaction được tạo ra như thế nào** 8](#_Toc155061452)

[**3. Khái niệm về mempool** 9](#_Toc155061453)

[**4. Cơ chế đồng thuận trong mạng blockchain** 9](#_Toc155061454)

[**4.1. Khái niệm cơ chế đồng thuận** 9](#_Toc155061455)

[**4.2. Các loại cơ chế đồng thuận** 9](#_Toc155061456)

[II. Smart contract 11](#_Toc155061457)

[**1. Cơ bản về smart contract** 11](#_Toc155061458)

[**1.1. Khái niệm smart contract** 11](#_Toc155061459)

[**1.2. Cấu trúc smart contract** 11](#_Toc155061460)

[**1.3. Lợi ích và hạn chế** 12](#_Toc155061461)

[**1.4. Một số ứng dụng** 13](#_Toc155061462)

[**2. Cách triển khai smart contract lên mạng blockchain** 13](#_Toc155061463)

[III. Front-running mempool attack 15](#_Toc155061464)

[**1. Front-running attack là gì** 15](#_Toc155061465)

[**2. Cách thức hoạt động của front-running attack** 15](#_Toc155061466)

[**3. Các cuộc tấn công áp dụng front-running mempool attack hiện nay** 16](#_Toc155061467)

[**4. Các giải pháp:** 17](#_Toc155061468)

[IV. Demo front-running attack 19](#_Toc155061469)

[**1. Tổng quan về ứng dụng** 19](#_Toc155061470)

[**2. Cách dựng private EVM blockchain** 21](#_Toc155061471)

[**3. Viết Password Attack contract** 26](#_Toc155061472)

[**4. Cách dựng web và giao diện ứng dụng** 28](#_Toc155061473)

[**5. Các bước thực hiện demo** 35](#_Toc155061474)

[**6. Giải pháp:** 41](#_Toc155061475)

[V. Tài liệu tham khảo 45](#_Toc155061476)

# **Yêu cầu của project**

1. Mục tiêu của project:

Project này nhóm sẽ tập trung vào giới thiệu khái niệm, phương thức hoạt động của front-running attack và các loại tấn công áp dụng front-running phổ biến hiện nay. Sau đấy, nhóm sẽ phát triển một ứng dụng demo front-running attack để cung cấp cho người đọc một cái nhìn tổng quan về loại hình tấn công này.

2. Các nhiệm vụ chính:

Đầu tiên, nhóm sẽ đưa ra các khái niệm và giải thích cơ bản về blockchain, smart contract. Sau đó, nhóm sẽ đưa ra khái niệm, cơ chế hoạt động cũng như các dạng tấn công áp dụng front-running attack hiện nay. Cuối cùng, nhóm sẽ xây dựng một ứng dụng demo về loại hình tấn công này cũng như đưa ra giải pháp để hạn chế, ngăn chặn front-running attack.

# **Mở đầu**

Blockchain ra đời như một công nghệ mang lại tính minh bạch và phi tập trung lưu lại các giao dịch và dữ liệu một cách an toàn. Với sự ra đời của hợp đồng thông minh (smart contract) trên blockchain, phổ biến như Ethereum, rất nhiều loại ứng dụng đã được phát triển trên blockchain như game, defi (decentralized exchange), did (decentralize identity),…. Tuy nhiên, việc phát triển hợp đồng thông minh vẫn luôn tồn tại những lỗ hổng mà kẻ tấn công có thể khai thác được nếu như lập trình viên không có kiến thức về bảo mật trong blockchain. Rất nhiều cuộc tấn công ở trên blockchain, đặc biệt là Ethereum, đã diễn ra và nguyên nhân chủ yếu do sự thiếu hiểu biết về bảo mật ở lập trình viên.

Hiện nay, front-running attack là một kiểu tấn công khá phổ biến trên mạng Ethereum nhưng lại rất ít lập trình viên biết và hiểu rõ về mô hình tấn công này. Loại tấn công này thường gây ra thiệt hại rất lớn về mặt tiền bạc cho người sử dụng [1]. Vì vậy, việc nghiên cứu và hiểu sâu về cách thức hoạt động của front-running attack là vô cùng quan trọng, nhất là đối với lập trình viên blockchain.

# **Thành viên nhóm và nhiệm vụ**

Các nhiệm vụ tìm hiểu và triển khai project đều được phân công cho các thành viên một cách công bằng và không có sự chênh lệch đáng kể:

1. Lê Đình Huy – 20194776

- Tìm hiểu và viết contract (password attack contract và solution contract), viết script để deploy contract lên mạng blockchain.

- Viết final report.

2. Phạm Minh Đăng – 20194737

- Tìm hiểu và dựng private EVM blockchain, viết sample code lắng nghe các transaction được thêm vào mempool cũng như trong block mới được tạo.

- Viết final report.

3. Đặng Minh Khôi – 20194781

- Phát triển web để demo front-running attack, tìm hiểu về thư viện ethers.js và wagmi cho việc tương tác với contract cũng như kết nối với ví metamask.

- Làm slide báo cáo.

# **I. Blockchain là gì**

## **1. Khái niệm về blockchain**

Blockchain là danh sách các block được nối tiếp nhau thông qua hàm băm. Mỗi block sẽ tham chiếu đến block trước nó (thông qua mã băm của block trước được lưu trong block header). Mỗi block sẽ chứa thông tin về block trước đó nên nếu một block trước đó thay đổi, thì các block phía sau nó cũng sẽ bị thay đổi theo. Do đó, các giao dịch blockchain là không thể đảo ngược.

Trong một block của blockchain bao gồm block header (chứa các metadata) và các giao dịch (transaction). Block header gồm 3 phần: [2]

- Mã băm tham chiếu đến block trước đó (dùng để kết nối block này đến block trước nó trong chuỗi).

- Các metadata sẽ phụ thuộc vào cơ chế đồng thuận của blockchain, với thuật toán PoW (proof of work) thì sẽ có các trường như sau: độ khó, timestamp, số nonce (một biến counter được sử dụng trong thuật toán này, miner sẽ tăng dần số nonce từ 0 đến khi tìm được số nonce thỏa mãn theo yêu cầu của thuật toán và người đầu tiên tìm được số nonce này thì sẽ có quyền thêm block mới vào blockchain).

+ Merkle tree root: là một mã băm độc nhất, dùng để kiểm tra tính hợp lệ của các transaction trong một block, được thực hiện thông qua thuật toán merkle tree lên các transactions trong block đó. Merkle tree được dùng để băm đệ quy các mã băm của transaction trong một block đến khi tạo ra được một mã băm duy nhất được gọi là merkle root, thuật toán băm được dùng là double-SHA256 [2]

A diagram of a diagram

Description automatically generated

*Link ảnh tham khảo theo bitcoin book* [2]

## **2. Khái niệm về transaction**

### **2.1. Khái niệm**

Transaction là bản ghi hoạt động của một người dùng trên blockchain, thường là một giao dịch chuyển tiền giữa các bên tham gia vào mạng. Trên các mạng EVM như Ethereum, một transaction có thể là một hành động thực thi một hàm nào đấy của một smart contract hoặc là một hành động deploy contract của người dùng. Thông tin chi tiết của một transaction sẽ phụ thuộc vào kiểu mạng blockchain mà nó được triển khai, thường thì sẽ có các trường chung như: transaction hash (mã băm giao dịch), from (địa chỉ người gửi), to (địa chỉ người nhận), phí giao dịch.

### **2.2. Một transaction được tạo ra như thế nào**

Thông thường, trên các mạng EVM như Ethereum, một transaction sẽ được tạo ra thông qua các bước: khởi tạo, phát tán, xác minh, được thêm vào block và được thực thi. Đầu tiên, khi người dùng thực hiện một transaction (ví dụ như giao dịch chuyển tiền), họ sẽ tạo ra một transaction và kí lên transaction đó bằng private key của mình. Tiếp theo, transaction này sẽ được gửi đến một node, node sẽ kiểm tra xem transaction này có hợp lệ hay không. Nếu transaction không hợp lệ, nó sẽ bị loại bỏ. Ngược lại, node sẽ phát tán nó lên các node khác trong mạng. Sau khi các node trên mạng nhận được transaction, các node đó sẽ thêm nó vào mempool. Khi miner được chọn là người có khả năng thêm block mới vào blockchain, họ sẽ lấy các transaction trong mempool để thêm vào block mới. Sau khi block mới được tạo ra thì có nghĩa là các transaction này đã được thực thi. [3]

## **3. Khái niệm về mempool**

Mempool là một bể chứa toàn bộ các transactions chưa được thực hiện (pending transactions) trong blockchain. Khi một người thực hiện transaction, transaction của họ sẽ chưa được thực thi ngay, thay vào đó sẽ được đưa vào mempool.

Khi miner xây dựng một block mới, họ sẽ chọn các transactions từ mempool và đưa vào block mới. Các transactions sẽ được ưu tiên chọn trước phụ thuộc vào cơ chế của từng blockchain (ví dụ: transaction fee với Bitcoin và gas price với Ethereum). Sau đó, miner sẽ cố gắng chứng minh tính hợp lệ của block mới đó để thêm block đó vào blockchain.

## **4. Cơ chế đồng thuận trong mạng blockchain**

### **4.1. Khái niệm cơ chế đồng thuận**

Cơ chế đồng thuận có ý nghĩa vô cùng quan trọng trong lĩnh vực công nghệ blockchain vì nó đảm bảo sự đồng thuận và tính hợp lệ của giao dịch trên toàn mạng lưới. Cơ chế này bao gồm quá trình mà các node trong mạng lưới đồng thuận về trạng thái hiện tại của blockchain. Trong phần này, nhóm sẽ giải thích về hai thuật toán đồng thuận phổ biến là proof of work (PoW) và proof of stake (PoS).

### **4.2. Các loại cơ chế đồng thuận**

***a. Proof of work (PoW)***

Thuật toán đồng thuận PoW, được tiên phong bởi Bitcoin, là một cơ chế để xác nhận giao dịch và duy trì tính toàn vẹn của mạng lưới blockchain. Theo thuật toán này, người tham gia được gọi là miner, họ tham gia vào quá trình cạnh tranh nhằm giải quyết các câu đố toán học phức tạp. Miner đầu tiên thành công trong việc giải mã câu đố phức tạp sẽ được công nhận bời các miner khác và được nhận phần thưởng là native token và phí của các transaction, sau đó thêm block mới vào blockchain. Quá trình trên đòi hỏi một lượng lớn tài nguyên tính toán và gây ra mức độ tiêu thụ năng lượng đáng kể. An ninh của hệ thống blockchain được duy trì thông qua việc sử dụng PoW, giúp ngăn chặn kẻ tấn công can thiệp vào các giao dịch trước đó. Điều này được thực hiện bằng cách yêu cầu một lượng tính toán vô cùng lớn đối với bất kỳ cố gắng sửa đổi các bản ghi lịch sử của blockchain. [4] [5]

***b. Proof of stake (PoS)***

Thuật toán đồng thuận PoS được xem như là một phương pháp thay thế cho PoW, người được chọn để thêm block mới sẽ được gọi là validator thay vì miner. Trong cơ chế đồng thuận PoS, việc chọn validator để tạo ra các block mới được xác định dựa trên số lượng tiền (native token) mà họ stake như là một phí cọc để được tham gia vào quá trình đào block, nếu một validator có những hành vi không trung thực trong quá trình đào thì họ sẽ bị mất đi khoản phí này. Việc chọn lựa validator thường được thực hiện thông qua một quy trình xác định trước, bao gồm các yếu tố như số lượng tiền họ đã stake và thời gian họ đã stake nó. Cơ chế đồng thuận PoS được công nhận rộng rãi vì khả năng tiết kiệm năng lượng và giảm thiểu tài nguyên tính toán so với cơ chế PoW. [4] [5]

# **II. Smart contract**

## **1. Cơ bản về smart contract**

### **1.1. Khái niệm smart contract**

Smart contract hay còn được gọi là hợp đồng thông minh, là một chương trình chạy trên mạng blockchain, được biểu diễn bằng mã máy và tự động thực thi các điều khoản được đáp ứng bởi vì chúng đã được biên dịch ở trong code. Khác với các hợp đồng thông thường, smart contract không cần đến một bên thứ ba đáng tin cậy làm trung gian, cung cấp phương pháp an toàn và phi tập trung để tạo điều kiện thuận lợi thực thi các thỏa thuận giao dịch. Trên các mạng blockchain như Ethereum, smart contract có một địa chỉ xác định, là nơi lưu trữ chương trình, ngoài ra nó còn có số dư và cho phép nhận các giao dịch được gửi đến. [6]

### **1.2. Cấu trúc smart contract**

Mỗi loại blockchain sẽ có các quy chuẩn riêng cho smart contract. Ví dụ như: với Ethereum, smart contract sẽ được viết bằng ngôn ngữ Solidity. Mặc khác, ở Cosmos, smart contract sẽ được viết bằng ngôn ngữ Rust. Các ngôn ngữ trên sẽ có cách triển khai khác nhau, tuy nhiên thì chúng thường có chung các phần như tên hợp đồng, biến, các hàm thực thi và các hàm truy vấn để lấy ra giá trị được lưu trên hợp đồng. Ở đây, do nhóm làm về tấn công trong EVM blockchain, nên nhóm sẽ lấy một ví dụ về cấu trúc hợp đồng Solidity:

A screenshot of a computer program

Description automatically generated

Ở đoạn code trên, dòng code thứ 3 sẽ định nghĩa về phiên bản được chọn để compile đoạn code Solidity ra bytecode. Dòng thứ 10 để định nghĩa tên của contract, các biến sẽ được định nghĩa như dòng thứ 12, gần giống như cách mà định nghĩa biến trong một class. Bên cạnh đó, dòng thứ 18 là một hàm thực hiện với chức năng của nó là lưu một giá trị vào biến trong đoạn code và dòng thứ 26 là một hàm để lấy ra kết qua giá trị number đã được lưu trên hợp đồng thông minh.

### **1.3. Lợi ích và hạn chế**

Smart contract ngày càng được áp dụng rộng rãi hơn vì tính phi tập trung, không cần bên trung gian thứ ba của nó. Ngoài ra, smart contract còn mang lại nhiều lợi ích như tự động thực thi vì so với hợp đồng thông thường thì smart contract sẽ tự thực thi các điều khoản khi mà các điều kiện được đáp ứng. Smart contract còn có tính công khai và minh bạch do nó được lưu trên mạng blockchain, nên ai cũng có thể xem được nội dung của smart contract, các giao dịch liên quan đến nó nhằm đảm bảo không có một ý đồ nào xấu trong smart contract được thực hiện mà người dùng không biết. Smart contract đưa ra kết quả rõ ràng vì các điều kiện thực thi được biên dịch từ code nên các kết quả của hợp đồng thông minh là chính xác. Smart contract còn bảo vệ quyền riêng tư do các giao dịch trên liên quan đến smart contract được hiển thị bằng địa chỉ của người thực hiện giao dịch, nên nó vẫn đảm bảo được quyền riêng tư của người dùng khi tham gia vào mạng blockchain.

Bên cạnh những lợi ích mà smart contract đem lại, nó cũng còn tồn đọng một số hạn chế về an toàn bảo mật và vẫn chưa nhận được sự chấp nhận của pháp luật. Ngoài ra, smart contract còn bị hạn chế về mặt kích thước, dẫn đến các chức năng của nó cũng bị giới hạn. Bên cạnh đó, smart contract còn có tính thụ động, nó không thể tự kết nối và lấy thông tin từ thế giới bên ngoài vì cấu trúc thiết kế và những thông tin đó có thể phá vỡ sự đồng thuận.

### **1.4. Một số ứng dụng**

Hiện nay, vô vàn các loại ứng dụng khác nhau đã được triển khai trên blockchain. Tuy nhiên, ở đây nhóm sẽ đề cập đến một vài ứng dụng phổ biến như sau:

- Token: là một loại hợp đồng thông minh đề cập đến một loại tiền nào đó trong blockchain. Ví dụ: USDT token là một loại token được dùng để đại diện cho đồng USD ở trên mạng blockchain.

- NFT: là một loại hợp đồng thông minh để định danh cho loại tài sản quý giá, có số lượng hữu hạn. Ví dụ: tranh, thẻ hội viên, vật phẩm trong game như vũ khí, áo giáp, thú cưng,…

- Voting System: là một loại hợp đồng thông minh dùng để bỏ phiếu bầu cử cho một quyết định nào đó, đồng thời danh tính của người bầu cử được giữ bí mật bởi tính ẩn danh của blockchain.

## **2. Cách triển khai smart contract lên mạng blockchain**

Mỗi loại blockchain sẽ có các bước triển khai smart contract khác nhau. Tuy nhiên, chúng đều có bước chung là đoạn code smart contract sẽ được biên dịch ra bytecode phù hợp với virtual machine mà mạng blockchain đó triển khai. Vì các bước để tải bytecode này lên các mạng blockchain là khác nhau, nên trong project này, nhóm chỉ giải thích cách mà các mạng EVM như Ethereum thực hiện.

Sau khi người dùng compile đoạn code smart contract ra bytecode thành công, người đó sẽ thực hiện một transaction và trong trường data là bytecode của contract cùng với các biến khởi tạo nếu có. Khi transaction này được thực hiện, đồng nghĩa với việc contract đã được triển khai lên mạng blockchain thành công. Contract này sẽ được gắn một địa chỉ phụ thuộc vào cơ chế mà blockchain đó triển khai.

A screenshot of a chat

Description automatically generated

# **III. Front-running mempool attack**

## **1. Front-running attack là gì**

Trên các mạng blockchain, đặc biệt là Ethereum, front-running attack là một mô hình tấn công mà trong đó kẻ tấn công lợi dụng tính minh bạch, công khai của mempool, nơi mà các pending transaction được lưu trữ, nhằm chèn các transaction của mình vào trước transaction mục tiêu để chuộc lợi cho bản thân [7]. Bởi vì đặc tính sắp xếp thứ tự các transaction dựa vào gas price trên Ethereum, nên transaction nào có gas price cao hơn sẽ được ưu tiên thực thi trước. Do đó, front-running attack ngày càng trở nên nguy hiểm cho người dùng nếu như lập trình viên smart contract không có kiến thức về loại tấn công này.

Dưới đây là một ví dụ về một transaction mua token ở ví 0x438…25a2 đã bị front-running bởi kẻ tấn công có địa chỉ ví là 0xf9c…be68, được nhóm chụp lại ở trên dextool. Ở đây, người dùng bỏ 1 ETH vào để mua 1,281,753 tokens, khi kẻ tấn công thấy có thể thực hiện front-running attack để kiếm lời. Họ đã thực hiện một lệnh mua 1,618,927 tokens ở phía trước và bán ngay sau transaction của người mua để thu lợi được 1.2477 – 1.1954 = 0.0523 ETH.

A screenshot of a phone

Description automatically generated

*Link ảnh tham khảo:* [*https://www.dextools.io/app/en/ether/pairs*](https://www.dextools.io/app/en/ether/pairs)

## **2. Cách thức hoạt động của front-running attack**

Trên mạng blockchain, khi một người dùng tạo một transaction, transaction đó sẽ không được thực thi ngay mà nó sẽ được thêm vào mempool nơi chứa các transaction chưa được thực thi trên mạng. Do các transaction trong mempool đều minh bạch và được công khai, nên điều này vô tình tạo điều kiện cho các kẻ tấn công thực hiện front-running attack để chuộc lợi cho bản thân. Một cuộc tấn công front-running được diễn ra như sau:

Đầu tiên, kẻ tấn công sẽ quan sát các transaction đang được lưu trữ trong mempool. Vì các transaction trong mempool là công khai nên kẻ tấn công có thể biết được thông tin chi tiết của nó, bao gồm địa chỉ người gửi, người nhận, số lượng tiền (native token) và phí giao dịch. Nếu transaction đấy là một transaction gọi đến một chức năng của smart contract, thì cả phương thức gọi và tham số của nó cũng đều được hiển thị.

Sau khi quan sát các transaction trong mempool, kẻ tấn công nhận thấy có transaction có lợi cho mình, hắn sẽ ngay lập tức thực hiện một transaction có nội dung giống hệt với transaction mục tiêu nhưng với gas price cao hơn. Do transaction khi được thêm vào block được sắp xếp theo gas price theo thứ tự giảm dần, nên transaction của kẻ tấn công sẽ được ưu tiên thực thi trước, transaction mục tiêu sẽ được thực thi sau.

Sau khi một block mới được đào thành công, kẻ tấn công sẽ nhận được mọi lợi nhuận vì transaction của hắn được thực thi trước. Nạn nhân – người tạo ra transaction bị front-running sẽ bị mất phần lợi nhuận mà đáng lẽ họ được nhận. Sau đó, kẻ tấn công lại tiếp tục các hành động trên thông qua việc giám sát mempool.

## **3. Các cuộc tấn công áp dụng front-running mempool attack hiện nay**

Hiện nay, một cuộc tấn công áp dụng front-running attack vẫn đang diễn ra có tên gọi là sandwich attack. Đầu tiên, kẻ tấn công sẽ lắng nghe toàn bộ các transaction ở trong mempool trong thời gian thực. Với mỗi transaction mới được thêm vào, kẻ tấn công sẽ kiểm tra xem transaction đó có phải là transaction mua token không. Nếu như transaction đó là transaction mua token, chúng sẽ kiểm tra tiếp xem liệu họ sẽ phải mua bao nhiêu token trước người dùng đó để làm chệch tỉ giá token phục vụ cho việc kiếm lời. Sau khi tính toán được số lượng token họ cần mua, kẻ tấn công sẽ đặt một transaction mua ở đằng trước transaction mua của người dùng và transaction bán ở phía sau transaction mua này. Do transaction mua của người dùng bị kẹp giữa 2 transaction này nên được gọi là “sandwich attack”. Cuối cùng, khi block mới này được xác thực, người dùng sẽ nhận được ít token hơn do với ước tính ban đầu mà không biết đến sự xuất hiện của kẻ tấn công này. [8]

A diagram of a sandwich attack

Description automatically generated

*Link ảnh tham khảo theo trang web matcha* [8]

## **4. Các giải pháp:**

Vì miner luôn chọn các transaction có gas price cao hơn ở trong mempool để thêm vào block mới, nên chúng ta không thể thay đổi cơ chế này của miner trên mạng blockchain. Do đó, front-running attack gần như không có cách khắc phục hoàn toàn, thay vào đó sẽ phụ thuộc vào lập trình viên smart contract để có thể hạn chế khả năng kẻ tấn công kiếm lợi nhuận từ phương pháp này.

Hiện nay, để có thể ngăn chặn được “sandwich attack” đã nêu ở trên, lập trình viên smart contract nên thêm điều kiện trước khi người dùng thực hiện mua token, chính là cơ chế chống trượt giá (slippage). Khi người dùng mua token, hệ thống sẽ tính trước số lượng token mà họ sẽ nhận được. Người dùng có thể hoàn toàn tùy chỉnh thông số trượt giá này (slippage), để quyết định xem số lượng token tối thiểu mà họ chấp nhận thực hiện mua là bao nhiêu. Nếu số lượng token mà người dùng nhận được ít hơn số lượng token mà họ mong đợi, thì transaction sẽ bị lỗi. Ví dụ: nếu như người dùng muốn mua USDT từ ETH, giả sử họ nhập 1 ETH thì hệ thống báo là họ sẽ nhận được 4000 USDT sau khi giao dịch thực hiện thành công. Nếu họ đặt slippage là 5%, khi transaction được thực thi mà họ nhận được số lượng token lớn hơn hoặc bằng 4000 \* (1 – 0.05) = 3800 USDT thì transaction đó sẽ có trạng thái thành công. Ngược lại, transaction đó sẽ thất bại.

A screenshot of a phone

Description automatically generated

*Link ảnh tham khảo:* [*https://app.uniswap.org/swap?chain=mainnet*](https://app.uniswap.org/swap?chain=mainnet)

A screenshot of a computer

Description automatically generated

*Link ảnh tham khảo:* [*https://etherscan.io/address/0x7a250d5630b4cf539739df2c5dacb4c659f2488d*](https://etherscan.io/address/0x7a250d5630b4cf539739df2c5dacb4c659f2488d)

# **IV. Demo front-running attack**

## **1. Tổng quan về ứng dụng**

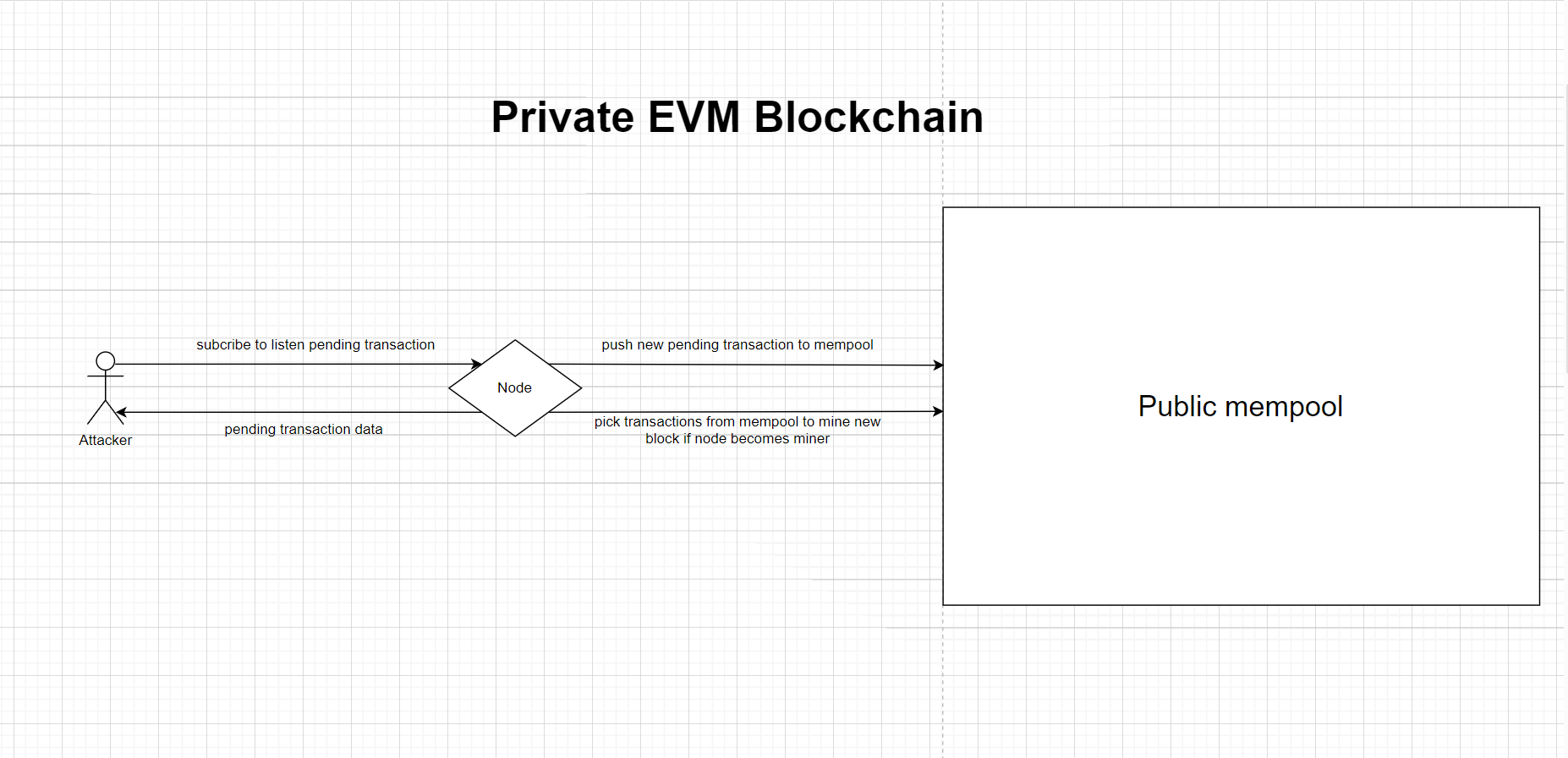
Front-running attack là một dạng tấn công khá phổ biến trên mạng blockchain. Tuy nhiên, những lập trình viên smart contract chưa có kinh nghiệm thường không biết đến dạng tấn công này, chỉ có những lập trình viên lâu năm và có kinh nghiệm mới biết. Trong thực tế, một mạng blockchain như Ethereum sẽ có rất nhiều node được chạy. Để kẻ tấn công có thể thực hiện được front-running attack thì họ sẽ phải tự chạy một full-node trên mạng đó, để có thể có được độ trễ thấp nhất khi lắng nghe các transaction trong mempool.

A diagram of a blockchain

Description automatically generated

*Ethereum blockchain thông thường với nhiều node*

Do điều kiện môn học không cho phép, nhóm sẽ mô phỏng lại mô hình tấn công này bằng cách tự dựng một mạng private EVM blockchain và chỉ có một node duy nhất. Bên cạnh đó, nhóm còn viết một smart contract có tên là password attack để áp dụng tấn công front-running lên contract đó. Ngoài ra, nhóm còn dựng một web UI để tương tác với smart contract (password attack) cũng như hiển thị quá trình tấn công front-running lên contract này qua giao diện để cho tường minh. Cuối cùng, nhóm sẽ đưa ra các giải pháp để hạn chế front-running attack đối với password attack contract.



*Private EVM blockchain với một node duy nhất*

Các bước để áp dụng mô hình tấn công front-running lên password attack contract được thực hiện như sau:

- Đầu tiên người tạo ra passwork attack contract sẽ deploy contract lên mạng private EVM. Trong contract sẽ có một biến chứa mật khẩu dưới dạng mã băm sử dụng thuật toán keccak256 và một hàm đoán mật khẩu (withdraw) cho phép người chơi nhập mật khẩu dưới dạng plain text, nếu mật khẩu đó sau khi được mã hóa với hàm băm keccak256 mà giống với mật khẩu được lưu trên contract thì người chơi sẽ được nhận tiền thưởng là số dư của contract.

- Sau đấy người tạo contract sẽ thực hiện deposit (chuyển tiền) vào contract 1 lượng native token (ví dụ 50 ETH) để tạo ra số dư, phục vụ việc trao thưởng sau này cho người chơi nào đoán đúng mật khẩu của contract.

- Tiếp theo sẽ có một người chơi tham gia đoán mật khẩu (plain password) của contract thông qua việc thực hiện một transaction là gọi hàm withdraw trong password attack contract. Sau khi transaction này được đưa vào mempool, tất cả mọi người đều có thể nhìn thấy nó cũng như trường data của nó.

- Sau đấy có một kẻ tấn công tham gia. Kẻ tấn công sẽ xem trong mempool và thấy transaction của người chơi đã được đẩy lên. Sau khi decode trường data của transaction, hắn thấy mật khẩu mà người chơi nhập là mật khẩu đúng và có mưu đồ chuộc lợi cho mình. Hắn sẽ thực hiện một transaction withdraw mới với mật khẩu giống mật khẩu người chơi đã nhập và gas price cao hơn gas price của transaction người chơi.

- Sau khi kẻ tấn công thực hiện transaction của họ, transaction đó sẽ được đẩy vào mempool. Sau một thời gian, miner sẽ thêm một block mới vào mạng blockchain, trong block đó có chứa cả giao dịch của người chơi và kẻ tấn công nhưng thứ tự sẽ là giao dịch của kẻ tấn công trước vì nó có gas price cao hơn. Sau khi block mới được thêm thành công, transaction của attacker sẽ được thực thi trước và số dư của contract sẽ thuộc về kẻ tấn công.

A diagram of a diagram

Description automatically generated

*Flow thực hiện tấn công front-running lên password attack contract*

## **2. Cách dựng private EVM blockchain**

Nhóm sử dụng thư viện “hardhat” [9] để chạy private EVM node. Hardhat là một thư viện chạy trên môi trường Node.js, cung cấp cho các lập trình viên smart contract một private EVM node để thực hiện kiểm thử với các cấu hình có thể tùy chỉnh theo nhu cầu của lập trình viên. Để có thể cung cấp được một demo trực quan nhất, nhóm đã quyết định cấu hình private EVM node này với các tiêu chí như sau:

- Cơ chế chọn transaction của miner ở trong mạng private EVM blockchain này phải hoàn toàn giống với cơ chế ở trên mạng blockchain Ethereum. Với yêu cầu này, nhóm đã nghiên cứu thư viện “hardhat” nói trên, để tìm cách thiết lập cơ chế chọn transaction của miner sao cho:

+ Các transactions có gas price cao hơn sẽ được thêm vào trước.

+ Nếu như hai transaction có cùng gas price, thì transaction nào được thêm vào trước, thì sẽ được đưa vào block trước.

+ Nếu như transaction nào không hợp lệ, thì sẽ bị hủy bỏ bởi node khi người dùng gửi transaction đó lên node đó. Ví dụ: nếu transaction có số nonce nhỏ hơn số nonce của người tạo transaction, thì transaction này là một transaction không hợp lệ.

Trong quá trình nghiên cứu nhóm đã nhận thấy thư viện “hardhat” đã thiết lập đủ yêu cầu cho cơ chế trên khi chạy node.

A white text with black text

Description automatically generated

*Link ảnh:* [*https://hardhat.org/hardhat-network/docs/explanation/mining-modes*](https://hardhat.org/hardhat-network/docs/explanation/mining-modes)

- Người kiểm thử phải có toàn quyền quyết định khi nào block mới sẽ được đào. Nếu như để cho private EVM node được đào block mới tự động, thì sẽ rất khó để demo quá trình transaction được đưa vào mempool do thời gian tạo block mới ở private EVM node sẽ rất nhanh, khoảng 1-2 giây. Với yêu cầu này, nhóm đã cấu hình thêm vào file cấu hình hardhat ở dòng 45-47 để đè vào cơ chế gốc của hardhat. Ở dòng 46, nhóm đã đặt “auto” sang giá trị “false” để tắt cơ chế tự động đào block mới. Ở dòng 48, nhóm đã đặt “allowUnlimitedContractSize” sang giá trị “true” để bỏ giới hạn kích thước của contract.

A black background with colorful text

Description automatically generated

Toàn bộ cấu hình của file hardhat của nhóm sẽ có code như sau:

*/\*\**

*\* Import các thư viện nên có để tạo nên một môi trường với đầy đủ chức năng*

*\*/*

require("dotenv").config();

require("@nomiclabs/hardhat-etherscan");

require("ethereum-waffle");

require("@nomiclabs/hardhat-waffle");

require("hardhat-gas-reporter");

require("solidity-coverage");

require("@nomiclabs/hardhat-ethers");

require("hardhat-deploy-ethers");

require("@nomiclabs/hardhat-solhint");

require("@nomiclabs/hardhat-web3");

require("dotenv/config");

require("hardhat-deploy");

require("hardhat-preprocessor");

require("@openzeppelin/hardhat-upgrades");

require("hardhat-gas-reporter");

const fs = require("fs");

*/\*\**

*\* Đọc các file thư viện import từ file remappings.txt*

*\* nếu như chúng ta không dùng các thư viện được tải sẵn*

*\* ở trong node\_modules*

*\*/*

function getRemappings() {

  return fs

    .readFileSync("remappings.txt", "utf8")

    .split("\n")

    .filter(Boolean) *// remove empty lines*

    .map((line) => line.trim().split("="));

}

module.exports = {

  solidity: {

    version: "0.8.13",

    settings: {

      optimizer: {

        enabled: true,

        runs: 200,

      },

    },

  },

  networks: {

    localhost: {

*// Thiết lập để chạy node với url: 127.0.0.1:8545*

*// và web socket: 127.0.0.1:8545/websocket*

      url: "http://127.0.0.1:8545",

    },

    hardhat: {

*// Thiết lập chainId cho môi trường private EVM blockchain*

      chainId: 1337,

      mining: {

*// tắt cơ chế tự động mine block*

        auto: false,

      },

*// bỏ giới hạn kích thước của contract*

      allowUnlimitedContractSize: true,

    },

  },

  etherscan: {

*// (optional), dùng để thực hiện quá trình xác thực code contract trên mainnet*

    apiKey: "",

    customChains: [],

  },

  preprocess: {

*/\*\**

*\* Tiền xử lí đọc các file thư viện import từ file remappings.txt*

*\* nếu như chúng ta không dùng các thư viện được tải sẵn*

*\* ở trong node\_modules*

*\*/*

    eachLine: () => ({

      transform: (line) => {

        if (line.match(/ from "/i)) {

          getRemappings().forEach(([find, replace]) => {

            if (line.match(find)) {

              line = line.replace(find, replace);

            }

          });

        }

        return line;

      },

    }),

  },

  paths: {

*// Thiết lập thư mục để đọc contracts, compile contracts*

    sources: "src/contracts",

    tests: "src/test",

    cache: "./cache",

    artifacts: "src/artifacts",

  },

};

Dưới đây là kết quả sau khi nhóm thực hiện chạy private node này. Node đã cung cấp đủ các yêu cầu trên và node cũng đã cung cấp các tài khoản có sẵn số dư để thực hiện việc kiểm thử trên mạng này. Ngoài ra, đoạn log từ lệnh chạy cũng cho thấy rằng node đã được chạy với url http là http://127.0.0.1:8545 và url websocket là ws://127.0.0.1:8545/websocket, giống với cấu hình ta đã thiết lập ở “hardhat”.

A screenshot of a computer program

Description automatically generated

## **3. Viết Password Attack contract**

Để demo mô hình tấn công front-running, nhóm đã viết một contract là Password Attack contract để minh họa thiệt hại do front-running attack gây ra nếu lập trình viên không viết code smart contract một cách an toàn. Dưới đây là đoạn code Password Attack contract:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract *PasswordAttack* {

*bytes32* public hashPassword =

       0x8283cdfe5070f6aae4bb830a6567261f6e54d1ad13c42cd0cba8df5d3370c678;

    constructor() {}

    receive() external payable {}

    function withdraw(*string* memory *password*) external {

        require(

            keccak256(*abi*.encodePacked(password)) == hashPassword,

            "Err: Not correct password"

        );

        (*bool* success, ) = payable(*msg.sender*).call{

            value: *address*(*this*).balance

        }("");

        require(success, "Err: Withdraw failed");

    }

}

Trong contract trên, có một biến là hashPassword có kiểu bytes32 trong ngôn ngữ Solidity (là một kiểu dữ liệu dùng để lưu trữ một chuỗi bytes có độ dài cố định là 32 bytes) dùng để lưu mã băm của password sau khi thực hiện băm password với hàm băm keccak256, chính là mã băm của đoạn “huydangkhoi” sau khi băm với hàm băm keccak256. Hàm “receive() external payable {}” giúp contract có khả năng nhận được native token như một tài khoản bình thường trên mạng, phục vụ cho việc chuyển tiền vào contract.

Hàm “withdraw” trong contract được dùng để người chơi thực hiện đoán mật khẩu contract và nhận thưởng là số dư của contract. Hàm này có truyền vào một tham số là “password” với kiểu “string”, cho phép người chơi đoán mật khẩu theo dạng “plain text”. Đoạn code:

require(

     keccak256(*abi*.encodePacked(password)) == hashPassword,

     "Err: Not correct password"

);

thực hiện việc băm mật khẩu mà người dùng nhập dưới dạng “plain text” với hàm băm keccak256. Sau khi băm, sẽ thực hiện kiểm tra mã băm này với biến “hashPassword” của contract. Nếu hai mã băm là giống nhau, contract sẽ thực hiện chuyển tiền thưởng là số dư của contract cho người đoán đúng. Nếu hai mã băm là khác nhau, contract sẽ thông báo lỗi cho người chơi là “Err: Not correct password” và người chơi sẽ không được nhận thưởng. Nếu người chơi đoán đúng mật khẩu, contract sẽ thực thi đoạn code:

(*bool* success, ) = payable(*msg.sender*).call{

            value: *address*(*this*).balance

        }("");

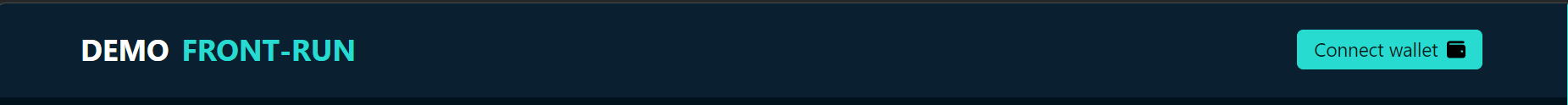
        require(success, "Err: Withdraw failed");

để chuyển tiền thưởng cho người chơi. Hàm “payable(msg.sender).call …” trả về một biến “success” có kiểu là “bool”. Nếu biến “success” bằng “true” thì việc chuyển tiền sẽ thành công. Ngược lại, nếu biến “success” bằng “false” thì việc chuyển tiền sẽ thất bại và có thông báo lỗi “Err: Withdraw failed”.

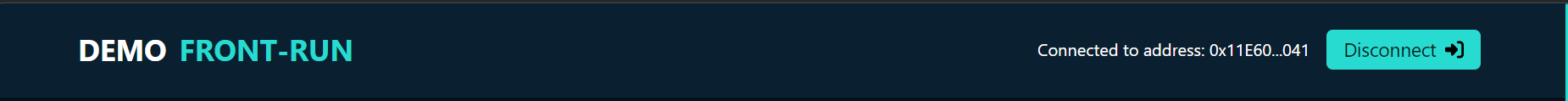
## **4. Cách dựng web và giao diện ứng dụng**

Để có thể demo trực quan việc áp dụng mô hình tấn công front-running lên Password Attack contract, nhóm đã thiết kế và dựng lên một ứng dụng web có thể tương tác được với ví Metamask, Password Attack contract và lắng nghe được các transaction trong mempool cũng như trong một block mới được đào. Công nghệ nhóm dùng để dựng web là NextJs, kết hợp với sử dụng thư viện Wagmi cho phép kết nối với ví Metamask và thư viện Ether.js để tương tác với contract, lắng nghe transaction trong mempool và trong block mới. [10] [11] [12]

Dưới đây là phần header của web khi người dùng chưa kết nối với ví metamask:



Để kết nối với ví Metamask, người dùng sẽ click vào button “Connect wallet”. Sau khi người dùng đã kết nối với ví, giao diện sẽ thay đổi như sau:



Phần thông tin về Password Attack contract sẽ bao gồm tên contract, hash password của contract, số dư hiện tại của contract và cả hàm deposit để thực hiện chuyển tiền lên contract. Để thực hiện deposit, người dùng sẽ nhập số tiền (native token) vào ô input “Deposit amount”. Sau đó, người dùng sẽ click vào button “deposit” để thực hiện giao dịch chuyển tiền lên contract.

A screenshot of a computer screen

Description automatically generated

Phần thực hiện front-running, nhóm viết một UI riêng là “Front Run Demo”. Người dùng có thể nhập password dưới dạng “plain text” để đoán mật khẩu contract và gas price tùy theo nhu cầu họ muốn. Sau khi nhập xong, người dùng sẽ click vào button “withdraw” để thực hiện đoán mật khẩu. Đồng thời, kẻ tấn công cũng có thể thực hiện đoán mật khẩu tương tự như người dùng bình thường nhưng hắn sẽ đặt gas price cao hơn và click vào button “withdraw”. Ngoài ra, button “mine” dùng để đào block mới trong mạng private EVM.

A screenshot of a login form

Description automatically generated

Để thể hiện được sự thay đổi số dư của người chơi cũng như contract, nhóm cũng dựng UI riêng có tên là “Account Balance Updated”. Trong phần này, nhóm thiết kế để khi có sự thay đổi số dư của người đang kết nối với ví và contract sẽ được hiển thị ngay trên giao diện. Để tương quan hơn, số dư của người dùng và contract được hiển thị ở trạng thái trước khi thay đổi và sau khi thay đổi.

A screenshot of a computer

Description automatically generated

Để hiển thị những transaction nào hiện đang có trong mempool, nhóm thiết kế UI riêng có tên là “Pending Mempool”. Trong phần này, sẽ hiển thị những transaction nằm trong mempool và chưa được thực thi.

A screenshot of a computer

Description automatically generated

Khi một transaction được thêm vào mempool, nhóm đã thực hiện để lắng nghe transaction này ở thời gian thực thông qua cơ chế websocket sử dụng thư viện ether.js. Dưới đây là đoạn code lắng nghe các transaction:

A computer screen shot of a program code

Description automatically generated

Ở dòng 17, nhóm sử dụng phương thức “on” của “provider” và truyền vào với tham số là “pending” để lắng nghe các transaction mới được thêm vào mempool. Lập trình viên hoàn toàn có thể truyền một hàm xử lí dữ liệu transaction thông qua việc gọi hàm “listen” và truyền vào hàm đó.

Để hiển thị những transaction nào đã được thực thi, nhóm thiết kế UI riêng có tên là “Executed Transactions”.

A screenshot of a computer

Description automatically generated

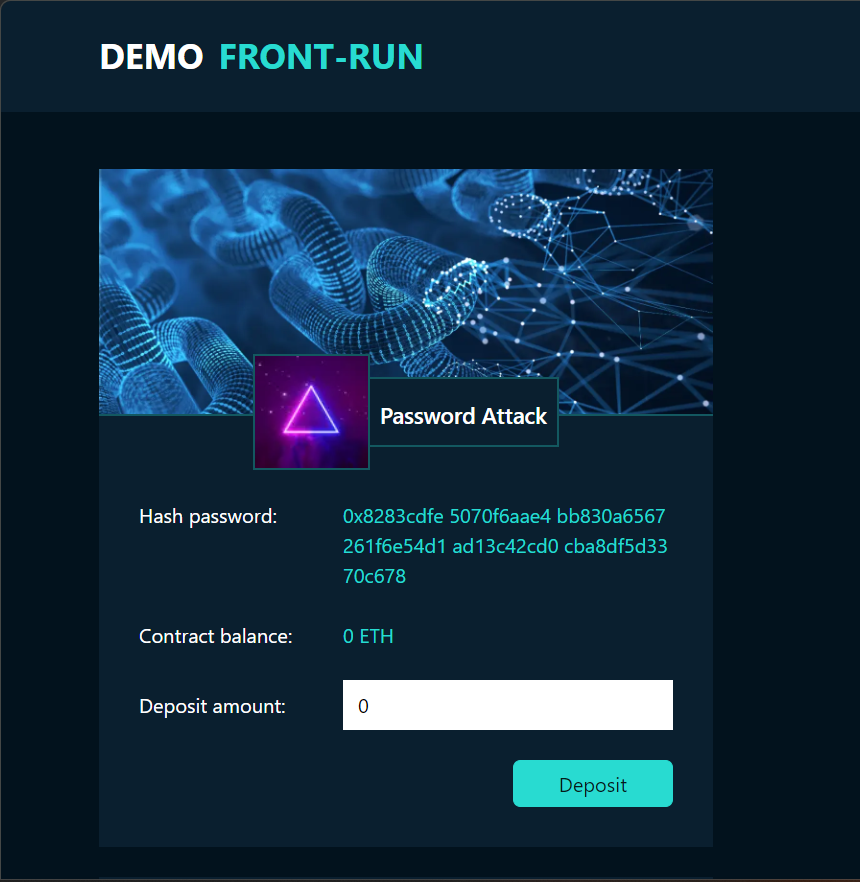
Dưới đây là đoạn code lắng nghe những transaction nào đã được thực thi:



Ở dòng 20, nhóm sử dụng phương thức “on” của “provider” và truyền vào với tham số là “block” để lắng nghe block nào vừa được thêm vào mạng. Sau đấy, nhóm sẽ thực hiện lấy các transaction có trong block đó và tạo thành một mảng chứa các transaction đã được thực thi. Mảng này sẽ trở thành tham số truyền vào hàm “callback”. Nhóm sẽ truyền một hàm xử lí dữ liệu các transaction đã được thực thi thông qua gọi hàm listen có tham số truyền vào là hàm “callback”.

## **5. Các bước thực hiện demo**

Đầu tiên, người tạo ra contract sẽ deploy contract. Sau đó, giao diện sẽ hiển thị giá trị của mật khẩu đã được băm thông qua thuật toán keccak256 trên contract, số dư hiện tại của contract. Ngoài ra, hàm deposit được sử dụng để chuyển native token vào contract.



Tiếp theo, người tạo ra contract sẽ thực hiện deposit 200 ETH vào contract Password Attack. Sau khi transaction được thực hiện, số dư của contract sẽ tăng từ 0 ETH lên 200 ETH.

A screenshot of a computer screen

Description automatically generated

Tiếp theo, một người chơi là Alice tham gia vào việc đoán mật khẩu contract để có thể nhận được phần thưởng là số dư contract (200 ETH). Sau đấy, Alice đoán mật khẩu của contract là “huydangkhoi” và may mắn mật khẩu đấy là mật khẩu chính xác, cô đặt gas price cho lần withdraw này là 9 Gwei. Sau khi click nút withdraw, transaction của Alice sẽ được thêm vào mempool. Hiện tại số dư tài khoản của Alice là 10000 ETH và số dư của contract vẫn là 200 ETH.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Sau khi transaction của Alice được thêm vào mempool thì Eve (attacker) xem các transactions nằm trong mempool và thấy transaction của Alice. Sau đó, Eve decode trường data và thấy rằng Alice đã đoán đúng được password của contract. Từ đó, Eve sẽ thực hiện một transaction mới với trường data giống với trường data trong transaction của Alice nhưng có gas price cao hơn là 10 Gwei nhằm đẩy độ ưu tiên transaction của mình lên cao hơn transaction của Alice khi các transaction này được thêm vào block mới. Sau khi bấm nút withdraw, transaction của Eve cũng được thêm vào mempool. Hiện tại số dư của Eve là 10000 ETH và số dư contract vẫn là 200 ETH.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Sau khi các transactions này được thêm vào block mới và block mới được đào thành công thì transaction của Eve sẽ được thực thi trước transaction của Alice. Do đó, Eve sẽ là người nhận được số dư của contract chứ không phải là Alice. Hình ảnh bên dưới minh họa kết quả sau khi block mới được đào:

- Các transactions ở trong “Pending Mempool”, giờ đã được chuyển sang phần “Executed Transactions”:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

- Số dư của Eve tăng thêm 200 ETH từ 10000 lên 10200 ETH và số dư của contract giảm từ 200 ETH về 0 ETH. Trong khi đó, số dư của ví Alice không đổi.

A screenshot of a video game account balance

Description automatically generated

A screenshot of a video game account balance update

Description automatically generated

## **6. Giải pháp:**

Một giải pháp mà nhóm đề ra để ngăn chặn cuộc tấn công front-running trên Password Attack contract đó chính là thay vì để người dùng gọi hàm withdraw lên contract, nhóm sẽ dùng một server backend có một API Post “withdraw” để người dùng đoán mật khẩu thông qua API đó. Sau khi một người dùng đoán đúng mật khẩu, phía backend sẽ gọi hàm “withdraw” để trả thưởng cho ví người dùng. Tuy nhiên, do điều kiện môn học không cho phép, bởi vì nhóm không có server và kinh phí để deploy server nên giải pháp trên chỉ là một giải pháp do nhóm đề xuất dựa trên kinh nghiệm đã phát triển các ứng dụng Web3 ngoài thực tế.

Về phía smart contract, đoạn code Password Attack contract sẽ được sửa đổi như sau:

- Ngay khi contract Password Attack được đưa lên mạng blockchain, địa chỉ owner của contract sẽ được thiết lập thông qua constructor. Như ở dòng 10, nhóm đã lưu địa chỉ deploy contract vào biến là owner.

- Hàm withdraw lúc này sẽ giới hạn chỉ có người deploy contract mới được gọi. Ngoài ra, hàm “withdraw” sẽ phải truyền thêm một tham số nữa là rewarder (địa chỉ người nhận số dư contract). Dòng số 29, nhóm đã sửa đổi lại đoạn code để chuyển thành cơ chế chuyển tiền cho địa chỉ rewarder.

A screenshot of a computer program

Description automatically generated

Về phía backend, do lúc này backend là người thực thi transaction một cách tự động khi điều kiện của API Post được kích hoạt. Do đó, việc private key được lưu trên server là một điều bắt buộc. Lúc này, ta phải có cách để bảo mật server backend để tránh một cuộc tấn công khác là SSH Bruth-force Attack. Với loại tấn công này, nhóm đề xuất cách tốt nhất đó chính là:

- Ở phía server, ta sẽ nên sử dụng Key-based SSH để login vào server thay vì dùng Password Authentication SSH. Dưới đây là cách tắt password authentication:

A screenshot of a computer program

Description automatically generated

*Link ảnh:* [*https://docs.bitnami.com/virtual-machine/faq/get-started/enable-ssh-keys/*](https://docs.bitnami.com/virtual-machine/faq/get-started/enable-ssh-keys/)

- Flow code backend sẽ như dưới hình. Ở bước người dùng đoán password thông qua API Post, ta sẽ gửi lên 2 trường chính là user\_address và guess\_password. Ở mỗi lần API Post này được gọi, nó sẽ thực hiện các bước sau: Đầu tiên, server gọi lên contract để lấy giá trị hash password ở trên contract, so sánh với giá trị guess\_password sau khi được băm bởi thuật toán keccak256. Nếu như bằng nhau thì server sẽ gọi hàm “withdraw” và đặt trường rewarder là địa chỉ user\_address của người dùng. Nếu không bằng nhau thì sẽ thông báo lỗi cho người dùng.

A diagram of a server

Description automatically generated

*Flow ứng dụng giải pháp mới đã được đề ra*

# **V. Tài liệu tham khảo**

|  |  |
| --- | --- |
| [1] | E. Mikalauskas, "Cybernews - $280 million stolen per month from crypto transactions," [Online]. Available: https://cybernews.com/crypto/flash-boys-2-0-front-runners-draining-280-million-per-month-from-crypto-transactions/. |
| [2] | A. M. Antonopoulos, "Github Bitcoin Book - Chapter 9," [Online]. Available: https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch09.asciidoc. |
| [3] | L. Team, "LCX," [Online]. Available: https://www.lcx.com/how-does-a-blockchain-transaction-work/. |
| [4] | B. Becher, "Bulitin - What Is a Consensus Mechanism?," [Online]. Available: https://builtin.com/blockchain/consensus-mechanism. |
| [5] | L. Conway, "Blockworks - Proof-of-Work vs. Proof-of-Stake: Which Is Better?," [Online]. Available: https://blockworks.co/news/proof-of-work-vs-proof-of-stake-whats-the-difference. |
| [6] | PhD. Đào Thành Chung, PhD. Đỗ Bá Lâm, "Oraichain Hackathon - Tài liệu Khóa học "Blockchain Anyone"," [Online]. Available: https://hackathon.orai.io/library/6421406c4148201092fa4d9a/. |
| [7] | Immunebytes, "Immunebytes - Front-Running Attacks in Blockchain: The Complete Guide," [Online]. Available: https://www.immunebytes.com/blog/front-running-attacks-in-blockchain-the-complete-guide/. |
| [8] | A. Allen, "Matcha - What is a sandwich attack?," [Online]. Available: https://blog.matcha.xyz/article/what-is-a-sandwich-attack. |
| [9] | Hardhat, "Harhat - Mining Modes," [Online]. Available: https://hardhat.org/hardhat-network/docs/explanation/mining-modes. |
| [10] | NextJs, "NextJs - Documentations," [Online]. Available: https://nextjs.org/docs. |
| [11] | EtherJs, "EtherJs - Documentations," [Online]. Available: https://docs.ethers.org/v5/. |
| [12] | Wagmi, "Wagmi - Documentations," [Online]. Available: https://wagmi.sh/react/getting-started. |