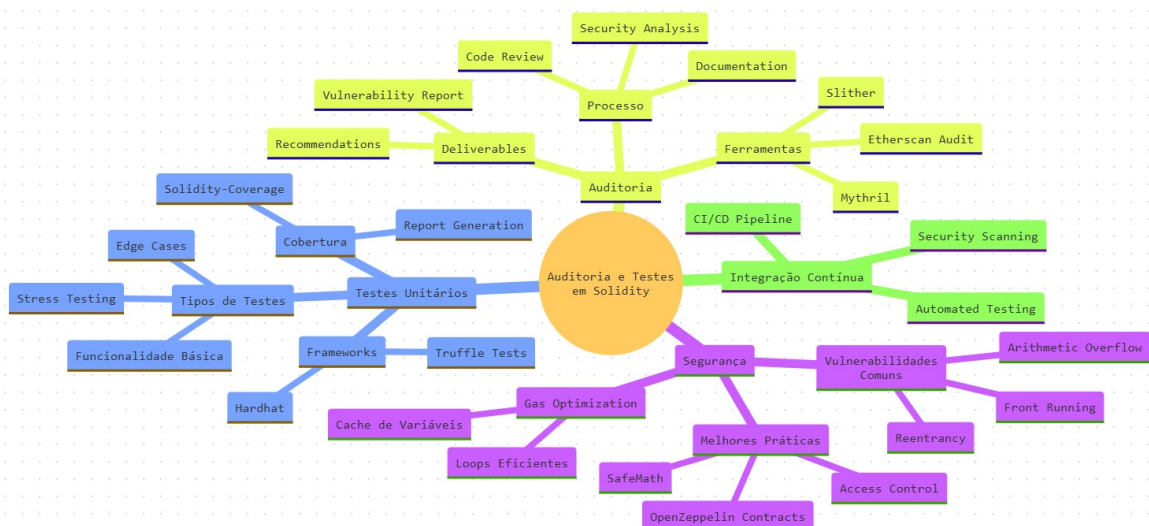




Testes e Auditoria

Identificar as ferramentas necessárias para a realização de testes e auditorias em contratos criados em Solidity

Prof. Jose Emiliano





Introdução



- Programar em uma blockchain é mais parecido a criar hardware do que programar software.
 - Ao implantar um contrato, não é possível corrigir erros, e qualquer erro pode resultar em perdas financeiras significativas e problemas de segurança.
 - Devem ser realizados testes para assegurar a segurança, funcionalidade, qualidade e eficiência dos contratos inteligentes.

Prof. Jose Emiliano



Introdução



- Os profissionais de blockchain e os desenvolvedores de contratos inteligentes dedicam uma quantidade significativa de tempo aos testes.
 - De acordo com vários estudos e entrevistas, cerca de 40-60% do tempo de desenvolvimento pode ser dedicado a testes e auditorias de segurança.

Prof. Jose Emiliano



Introdução



- Para testar contratos inteligentes, é vital ter objetivos claros, como a correção do sistema, a eficiência do gás ou fluxos de usuário específicos.
 - Definir casos de uso : As suítes de teste são baseadas nesses casos, que simulam interações usuário-sistema para descobrir possíveis erros.
 - Desenvolver um plano de testes e uma documentação detalhada: Esse plano descreve o escopo, a abordagem, os recursos e o cronograma dos testes, e ajuda a validar a qualidade do software.

Prof. Jose Emiliano



Porquê testar?



- **Segurança:**
 - Os contratos inteligentes lidam com ativos digitais, e qualquer vulnerabilidade pode ser explorada para roubar fundos. Testes rigorosos ajudam a identificar e mitigar esses riscos antes da implantação.
- **Funcionalidade:**
 - Garantir que o contrato inteligente se comporte conforme o esperado sob diferentes condições é vital para seu sucesso. Os testes verificam se todas as funções operam corretamente, se as mudanças de estado ocorrem como previsto e se os eventos são emitidos quando apropriado.

Prof. Jose Emiliano



Porquê testar?



- **Garantia da qualidade do código:**
 - Escrever testes pode ajudar a identificar e eliminar erros precocemente no ciclo de desenvolvimento, tornando o código mais robusto.
- **Aumento da velocidade de desenvolvimento**
- **Documentação:**
 - Os testes funcionam como uma forma de documentação, especificando claramente o comportamento esperado dos contratos inteligentes.
- **Eficiência:**
 - Os testes permitem otimizar o consumo de gás, o que pode reduzir os custos de transação.

Prof. Jose Emiliano



Tipos de testes



- **Automatizados**
- **Manuais**

Prof. Jose Emiliano



Tipos de testes



- Automatizados
 - Usam ferramentas que verificam o código (scripts) e simulam repetidamente uma interação humana
 - Tipos
 - Testes unitários – funções e componentes individuais, abrangendo cenários válidos (o que deve fazer) quanto inválidos.
 - Uso de Coverage (cobertura).
 - Testes de integração – verifica interações entre os componentes internos ou externos, como serviços de terceiros.
 - Testes baseados em propriedades – cumprimento de alguma propriedade definida.

Prof. Jose Emiliano



Tipos de testes



- Automatizados
 - Baseados em propriedades
 - Análise Estática
 - Verifica se um contrato satisfaz ou não uma propriedade baseado apenas na análise do código
 - Análise Dinâmica
 - Verifica se um contrato satisfaz ou não uma propriedade baseado na execução do contrato em si, injetando (fuzzing) entradas inválidas ou inesperadas em um sistema.

Prof. Jose Emiliano



Tipos de testes



- Manuais
 - Assistidos por humanos e envolvem a execução de cada cenário de teste .
 - Os testes manuais podem ser:
 - Em blockchain local
 - Ferramentas : Ganache, Truffle, Hardhat
 - Em redes de teste
 - Uso de Testnets : Arbitrum Sepolia

Prof. Jose Emiliano



Testes Unitários



- [Hardhat Tests](#) - Um dos principais ambientes de teste, baseado em ethers.js, Mocha e Chai. Ideal para desenvolvimento moderno em Solidity.
- [Foundry Tests](#) - Foundry inclui o Forge, um ambiente de testes extremamente rápido e versátil, capaz de realizar testes unitários, otimização de gás e *fuzzing* contratos.
- [Remix Test](#) - Muito usado em projetos pequenos e protótipos, conta com o plugin Solidity Unit Testing no Remix IDE para escrever e executar casos de teste.

Prof. Jose Emiliano



Testes Unitários



- [Truffle Tests](#) - Framework automatizado de testes de contratos. Apesar de estar sendo descontinuado, ainda pode ser usado via Hardhat.
- [Waffle](#) - Framework avançado de testes e desenvolvimento, baseado em ethers.js. Foco em testes limpos e de fácil leitura.
- [solidity-coverage](#) - Ferramenta de cobertura de código para Solidity. Embora seja independente, pode ser integrada com Hardhat.

Prof. Jose Emiliano



Análise Estática



- [Slither](#) - Ela ajuda a encontrar vulnerabilidades, melhora a compreensão do código e dá suporte à criação de análises personalizadas para contratos inteligentes.
- [Ethlint](#) - Linter para Solidity, usado para impor boas práticas de estilo e segurança.

Prof. Jose Emiliano



Análise Dinâmica



- [Echidna](#) - *Fuzzer* da Trail of Bits que utiliza propriedades definidas para detectar vulnerabilidades. Ideal para testes de robustez.
- [Diligence Fuzzing](#) - Ferramenta de *fuzzing* automatizada da ConsenSys, usada para encontrar violações de propriedades em contratos inteligentes.
- [Manticore](#) - Ambiente de execução simbólica dinâmica para análise de bytecode EVM. Pode ser usado tanto para *white-box fuzzing* quanto para testes de segurança.

Prof. Jose Emiliano



Análise Dinâmica



- [Mythril](#) - Avalia bytecode EVM usando execução simbólica para detectar vulnerabilidades, especialmente útil na fase de auditoria.
- [Diligence Scribble](#) - Linguagem de especificação e ferramenta de verificação em tempo de execução. Permite anotar contratos com propriedades e integrá-los a ferramentas como Diligence Fuzzing ou MythX.

Prof. Jose Emiliano



Segurança

Prof. Jose Emiliano



Segurança



- Problemas em dApps
 - Problemas de frontend
 - Erros em validação de campos
 - Erros em comunicação com JSON-API ou Ethers.js
 - Erros em chamada de métodos em contratos
 - Problemas de backend
 - Problemas de integração

Prof. Jose Emiliano



Segurança



- Problemas em dApps
 - Problemas de frontend
 - Problemas de backend
 - Erro no contrato (vulnerabilidades)
 - Comunicação com o contrato (endereço errado)
 - Erro na configuração do proxy
 - Erro na configuração das calls (inter-contratos)
 - Problemas de integração

Prof. Jose Emiliano



Segurança



- Problemas em dApps
 - Problemas de frontend
 - Problemas de backend
 - Problemas de integração
 - Uso de bibliotecas incorretas
 - Problemas nas chamadas de métodos dos contratos
 - Contratos proxy devem implementar interfaces

Prof. Jose Emiliano



Segurança



- **Vulnerabilidades Comuns:**
 - São pontos críticos que devem ser verificados constantemente
 - Reentrancy: ataques de múltiplas chamadas
 - Front Running: manipulação de ordem de transações
 - Arithmetic Overflow: Valide operações matemáticas

Prof. Jose Emiliano



Segurança



- **Melhores Práticas: Implemente padrões seguros**
 - OpenZeppelin: Use contratos testados e auditados
 - SafeMath: Evite problemas aritméticos
 - Access Control: Implemente controle de permissões adequado

Prof. Jose Emiliano



Segurança



- **Testes Unitários**
 - **Frameworks:** Escolha ferramentas adequadas
 - Truffle Tests: Para desenvolvimento com Truffle Suite
 - Hardhat: Ambiente completo de desenvolvimento
 - **Tipos de Testes:** Cubra diferentes cenários
 - Funcionalidade Básica: Verifique operações principais
 - Edge Cases: Teste condições extremas
 - Stress Testing: Valide performance sob carga

Prof. Jose Emiliano



Segurança



- **Auditoria**
 - **Ferramentas:** Utilize ferramentas especializadas
 - Mythril: Análise estática avançada
 - Slither: Detecção automática de vulnerabilidades
 - Etherscan Audit: Verificação na blockchain principal
 - **Processo:** Siga metodologia estruturada
 - Code Review: Revisão manual do código
 - Security Analysis: Análise de vulnerabilidades
 - Documentation: Documentação detalhada

Prof. Jose Emiliano



Segurança



- Integração Contínua
 - Implemente pipeline automatizado
 - CI/CD Pipeline: Automatize testes e deploy
 - Automated Testing: Execute testes automaticamente
 - Security Scanning: Monitoramento contínuo de segurança

Prof. Jose Emiliano



Front running



- O ataque de front-running é uma vulnerabilidade crítica nas blockchain que permite que atores maliciosos interceptem e explorem transações antes que estas sejam confirmadas na rede.
 - Monitoramento do mempool
 - Disputa da transação na frente dos validadores
 - Realização da transação

Prof. Jose Emiliano



Front Running

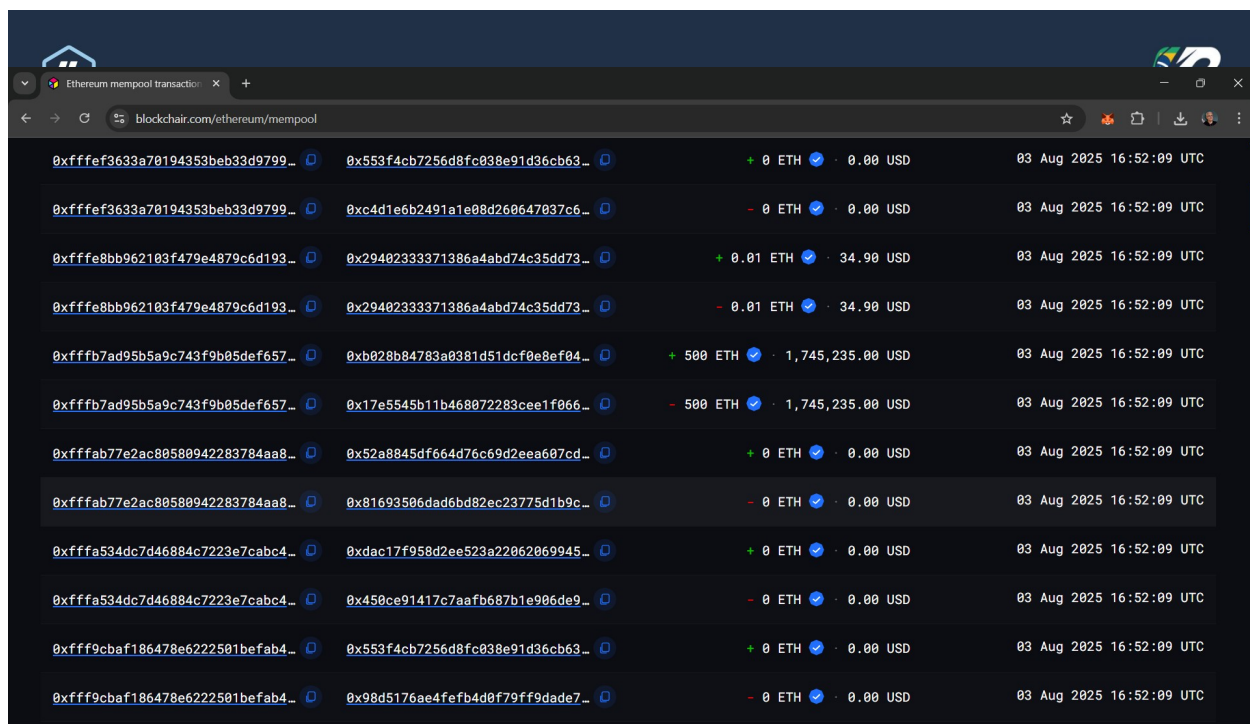


- Monitoramento do Mempool
 - Os bots maliciosos constantemente monitoram o mempool da blockchain
 - Eles procuram por transações que possam gerar lucro
 - Utilizam APIs especializadas para escutar transações não confirmadas

Prof. Jose Emiliano

Blockchair website showing the Ethereum mempool. The page displays a list of transactions with columns for Transaction id, Address, Amount, and Time. The transactions are sorted by time, showing a sequence of transactions occurring at 16:52:09 UTC on 03 Aug 2025. The amounts are in ETH and USD.

Transaction id	Address	Amount	Time
0xfffff8659f8b3de9827f32455361d...	0xc02aaa39b223fe8d0a0e5c4f27ea...	+ 0 ETH · 0.00 USD	03 Aug 2025 16:52:09 UTC
0xfffff8659f8b3de9827f32455361d...	0x39aab98d7293d38e052ecc3275de...	- 0 ETH · 0.00 USD	03 Aug 2025 16:52:09 UTC
0xfffff3633a70194353beb33d9799...	0x553f4cb7256d8fc038e91d36cb63...	+ 0 ETH · 0.00 USD	03 Aug 2025 16:52:09 UTC
0xfffff3633a70194353beb33d9799...	0xc4d1e6b2491a1e08d260647037c6...	- 0 ETH · 0.00 USD	03 Aug 2025 16:52:09 UTC
0xfffe8bb962103f479e4879c6d193...	0x2940233371386a4abd74c35dd73...	+ 0.01 ETH · 34.90 USD	03 Aug 2025 16:52:09 UTC
0xfffe8bb962103f479e4879c6d193...	0x2940233371386a4abd74c35dd73...	- 0.01 ETH · 34.90 USD	03 Aug 2025 16:52:09 UTC



Transaction Hash	Amount (ETH)	Amount (USD)	Timestamp
0x553f4cb7256d8fc038e91d36cb63...	+ 0 ETH	0.00 USD	03 Aug 2025 16:52:09 UTC
0xc4d1e6b2491a1e08d260647037c6...	- 0 ETH	0.00 USD	03 Aug 2025 16:52:09 UTC
0x2940233371386a4abd74c35dd73...	+ 0.01 ETH	34.90 USD	03 Aug 2025 16:52:09 UTC
0x2940233371386a4abd74c35dd73...	- 0.01 ETH	34.90 USD	03 Aug 2025 16:52:09 UTC
0xb028b84783a0381d51dcf0e8ef04...	+ 500 ETH	1,745,235.00 USD	03 Aug 2025 16:52:09 UTC
0x17e5545b11b468072283cee1f066...	- 500 ETH	1,745,235.00 USD	03 Aug 2025 16:52:09 UTC
0x52a8845df664d76c69d2eea607cd...	+ 0 ETH	0.00 USD	03 Aug 2025 16:52:09 UTC
0x81693506dad6bd82ec23775d1b9c...	- 0 ETH	0.00 USD	03 Aug 2025 16:52:09 UTC
0xdac17f958d2ee523a22062069945...	+ 0 ETH	0.00 USD	03 Aug 2025 16:52:09 UTC
0x450ce91417c7aafb687b1e906de9...	- 0 ETH	0.00 USD	03 Aug 2025 16:52:09 UTC
0x553f4cb7256d8fc038e91d36cb63...	+ 0 ETH	0.00 USD	03 Aug 2025 16:52:09 UTC
0x98d5176ae4feb4d0f79ff9dade7...	- 0 ETH	0.00 USD	03 Aug 2025 16:52:09 UTC



Front Running



- Detecção de Oportunidades
 - Identificam transações que podem mover o mercado
 - Procuram padrões de compras/vendas significativas
 - Detectam transações com valores altos ou potencial de impacto

Prof. Jose Emiliano

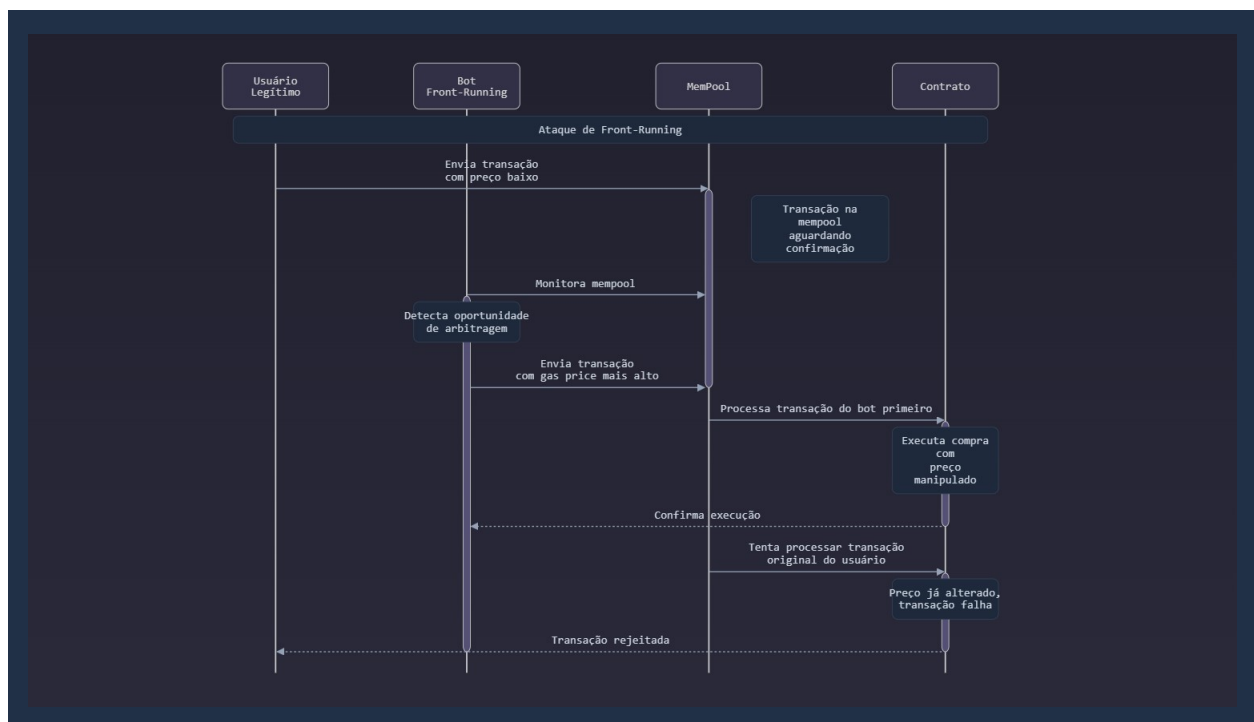


Front Running



- Execução do Ataque
 - Criam suas próprias transações aproveitando a oportunidade
 - Configuram um preço de gas mais alto para priorização
 - Garantem que sua transação seja processada primeiro

Prof. Jose Emiliano





Exemplo Prático



- Imagine uma situação comum:
 - Um investidor quer comprar 1000 tokens de um novo projeto
 - Ele envia sua transação com um preço de gas normal
 - Um bot front-runner detecta esta intenção de compra
 - O bot rapidamente compra 500 tokens do mesmo projeto
 - O preço sobe devido à demanda artificial
 - A transação original do investidor é executada a um preço mais alto

Prof. Jose Emiliano



Mitigação



- Soluções Técnicas
 - Usar batch orders em DEXs
 - Implementar time locks
 - Utilizar camadas L2
 - Adicionar proteções contra front-running nos contratos
- Boas Práticas
 - Dividir transações grandes em menores
 - Evitar transações durante períodos de alta volatilidade
 - Usar serviços de execução privada
 - Configurar alertas para movimentações suspeitas

Prof. Jose Emiliano



Contrato seguro



```
contract ProtegidoFrontRun {
    mapping(address => uint256) private ultimasTransacoes;

    modifier evitaFrontRun() {
        require(
            block.timestamp > ultimasTransacoes[msg.sender],
            "Suspeita de front-running"
        );
        _;
        ultimasTransacoes[msg.sender] = block.timestamp;
    }

    function transferir() public evitaFrontRun {
        // lógica da função
    }
}
```

Prof. Jose Emiliano

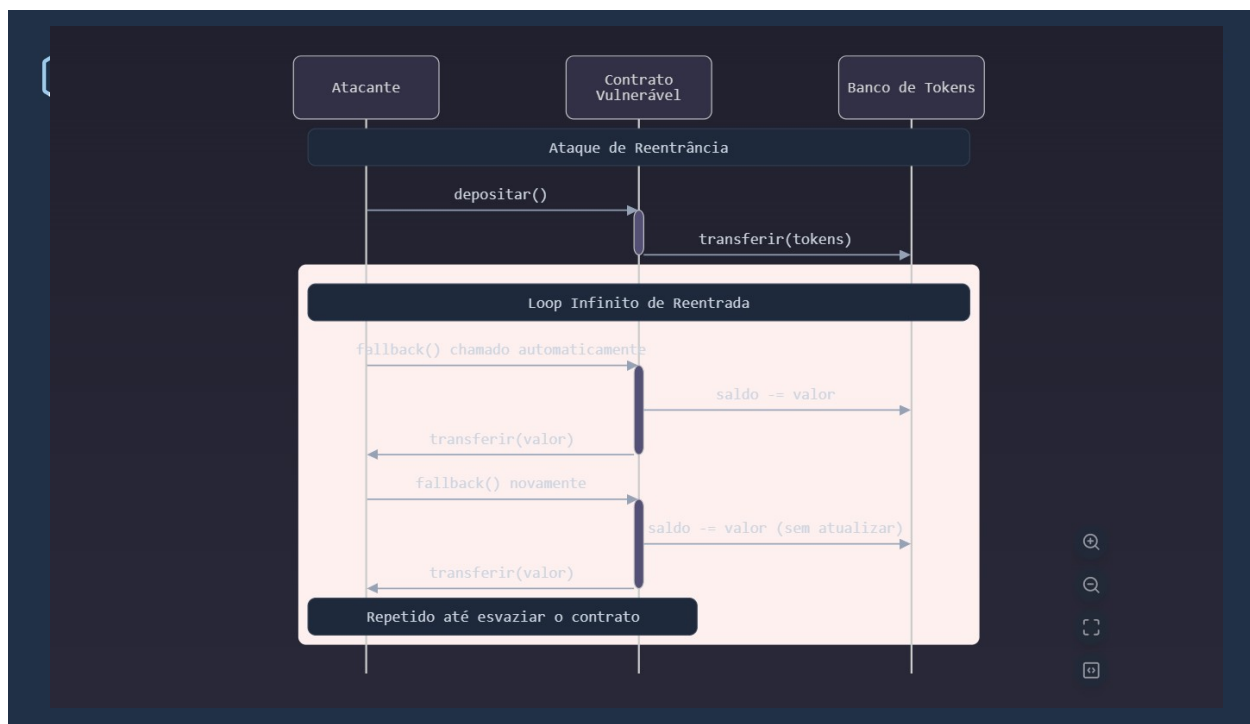


Reentrancy



- Explora a vulnerabilidade de contratos que transferem valores antes da verificação
 - Componentes Principais
 - Atacante: Contrato malicioso que explora a vulnerabilidade
 - Contrato Vulnerável: Implementação que segue o padrão perigoso de "transferir primeiro, verificar depois"
 - Banco de Tokens: Sistema que gerencia os saldos dos usuários

Prof. Jose Emiliano





Implementação segura

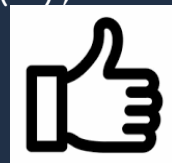


```
contract BancoSeguro {
    mapping(address => uint256) public saldos;

    function saque(uint256 valor) public {
        require(saldos[msg.sender] >= valor);

        // Primeiro atualiza o estado interno
        saldos[msg.sender] -= valor;

        // Depois realiza a transferência
        (bool success, ) = msg.sender.call{value: valor}("");
        require(success);
    }
}
```



Prof. Jose Emiliano



Arithmetic Overflow



- Explora limitações matemáticas nos tipos numéricos dos contratos inteligentes
 - Permite que um atacante realize operações impossíveis devido a problemas de representação de números.
 - Exemplo: variável uint2 não pode representar um número maior que 65535.
 - Foco no estouro de pilha por alocação de variáveis
 - Foco na obtenção de valores negativos para regras excessivamente flexíveis

Prof. Jose Emiliano



Exemplo



- Um uint2 pode representar valores até 65,535
 - Representação binária: 11111111111111
 - Valores interessantes:
 - Valor 1: 0
 - Valor 2: 16,383
 - Valor 3: 32,767
 - Valor 4: 49,151
 - Valor 5: 65,535

Prof. Jose Emiliano



Arithmetic Overflow



```
contract BancoVulneravel {
    mapping(address => uint256) public saldos;

    function depositar(uint256 valor) public {
        // Primeiro soma o valor novo ao saldo existente
        saldos[msg.sender] += valor;

        // Emite evento
        emit Depósito(msg.sender, valor);
    }

    function sacar(uint256 valor) public {
        // Apenas verifica se tem saldo suficiente
        require(saldos[msg.sender] >= valor);

        // Transfere os fundos
        (bool success, ) = msg.sender.call{value: valor}("");
        saldos[msg.sender] -= valor;
        require(success);
    }
}
```

Prof. Jose Emiliano



Arithmetic Overflow



```
import "@openzeppelin/contracts/utils/math/SafeMath.sol";

contract BancoSeguro {
    using SafeMath for uint256;

    mapping(address => uint256) public saldos;

    function depositar(uint256 valor) public {
        // Usa SafeMath para evitar overflow
        saldos[msg.sender] = saldos[msg.sender].add(valor);

        emit Depósito(msg.sender, valor);
    }

    function sacar(uint256 valor) public {
        require(saldos[msg.sender] >= valor, "Saldo insuficiente");

        // Subtração segura usando SafeMath
        saldos[msg.sender] = saldos[msg.sender].sub(valor);

        (bool success, ) = msg.sender.call{value: valor}("");
        require(success);
    }
}
```

add(): Soma com verificação de overflow
sub(): Subtração com verificação de underflow
mul(): Multiplicação com verificação de overflow
div(): Divisão com verificação de divisão por zero
mod(): Módulo com verificação de divisão por zero



Prof. Jose Emiliano



Testes de segurança

Prof. Jose Emiliano



Testes de segurança



- Um contrato é composto por variáveis de estado, variáveis globais e locais
 - Devidamente agrupados para atender os requisitos do sistema:
 - Deverá ser capaz de ...
 - A transferência se dará por...
 - Somente o proprietário deverá...
 - O contrato enviará dados para...

Prof. Jose Emiliano



Como gerar testes



- Analisar as funções públicas no seu contrato
 - É possível extrair e analisar funções públicas automaticamente com scripts de ou usar plugins como o Solidity Parser:

```
npm install @solidity-parser/parser
```

Prof. Jose Emiliano



```
1. const fs = require("fs");
2. const parser = require("@solidity-parser/parser");

3. const caminhoContrato = "contracts/contractDonation.sol";
4. const codigo = fs.readFileSync(caminhoContrato, "utf8");

5. try {
6.   const ast = parser.parse(codigo, { loc: true });
7.   parser.visit(ast, {
8.     ContractDefinition(node) {
9.       console.log(`Contrato encontrado: ${node.name}`);
10.    },

11.    FunctionDefinition(node) {
12.      const nomeFuncao = node.name || "(constructor)";
13.      const visibilidade = node.visibility || "sem visibilidade";
14.      const isPayable = node.stateMutability === "payable";

15.      console.log(`Função: ${nomeFuncao}`);
16.      console.log(`  - Visibilidade: ${visibilidade}`);
17.      console.log(`  - Payable: ${isPayable}`);
18.      console.log("---");
19.    },
```

Prof. Jose Emiliano



//continuação

```
1. StateVariableDeclaration(node) {
2.   node.variables.forEach((variavel) => {
3.     console.log(`Variável: ${variavel.name}`);
4.   });
5. }
6. });

7. } catch (e) {
8.   if (e instanceof parser.ParserError) {
9.     console.error("Erros de sintaxe no contrato:");
10.    e.errors.forEach(err => {
11.      console.error(`Linha ${err.line}: ${err.message}`);
12.    });
13.   } else {
14.     console.error("Erro inesperado:", e.message);
15.   }
16. }
```

Prof. Jose Emiliano





Outro exemplo

Prof. Jose Emiliano



Segurança



```
pragma solidity ^0.8.0; //versao inicial

contract SecureContract {
    mapping(address => uint256) private balances;

    constructor () {}

    function transfer(address recipient, uint256 amount) external {
        require(amount > 0, "Amount must be positive");
        require(balances[msg.sender] >= amount, "Insufficient balance");

        balances[msg.sender] -= amount;
        balances[recipient] += amount;
    }
}
```

Prof. Jose Emiliano



```
pragma solidity ^0.8.0; //versão ajustada para segurança
```

```
contract SecureContract {
    mapping(address => uint256) private balances;

    // Evento para registrar falhas de transferência. Indexed para pesquisar no log
    event TransferFailed(address indexed sender, address indexed recipient, uint256 amount,
        string reason);

    function transfer(address recipient, uint256 amount) external {
        if (amount <= 0) {
            emit TransferFailed(msg.sender, recipient, amount, "Amount must be positive");
            revert("Amount must be positive");
        }

        if (balances[msg.sender] < amount) {
            emit TransferFailed(msg.sender, recipient, amount, "Insufficient balance");
            revert("Insufficient balance");
        }

        balances[msg.sender] -= amount;
        balances[recipient] += amount;
    }
}
```

Prof. Jose Emiliano



Testes

Identificar etapas para a construção de testes

Prof. Jose Emiliano



Testes



- O que são
 - Facilidades disponibilizadas aos desenvolvedores por meio de plataformas que permitem
 - Interagir com o contrato
 - Forjar manipulações indevidas
 - Inserir valores indevidos mas possíveis

Prof. Jose Emiliano



Fuzzing



- É uma técnica de teste de software que insere dados aleatórios, inválidos ou inesperados em um programa para descobrir falhas, vulnerabilidades ou comportamentos inesperados.
 - A idéia é simples: bombardear o sistema com entradas imprevisíveis e observar como ele reage.
 - Como funciona na prática
 - Um fuzzer gera milhares de entradas malformadas automaticamente.
 - Essas entradas são enviadas para o programa em teste.
 - O sistema é monitorado para detectar crashes, exceções, vazamentos de memória ou comportamentos incorretos.
 - Se algo falhar, o fuzzer registra a entrada que causou o problema para análise posterior.

Prof. Jose Emiliano



Fuzzing



- Utilidade:
 - Detecta vulnerabilidades antes que sejam exploradas por atacantes.
 - Simula ataques reais de forma segura.
 - É usado em pentests, auditorias de segurança e validação de software crítico.

Prof. Jose Emiliano



Importância



- Segurança
- Funcionalidade
- Garantia de qualidade
- Aumento da velocidade
- Documentação
- Eficiência

Prof. Jose Emiliano



Métodos



- Testes automatizados
 - Os testes automatizados utilizam ferramentas que verificam automaticamente o código de um contrato inteligente em busca de erros de execução.
- Testes manuais
 - Os testes manuais são assistidos por humanos e envolvem a execução de cada caso de teste na suíte de testes, um após o outro.
 - Eles são diferentes dos testes automatizados, nos quais é possível executar simultaneamente múltiplos testes distintos em um contrato e obter um relatório que mostre todos os testes falhos e bem-sucedidos.

Prof. Jose Emiliano



Testes Automatizados



- Testes unitários
 - Foco em funções e componentes individuais
 - Baseia-se em suposições de parâmetros e o estado do sistema
 - Deve abranger cenários válidos quanto inválidos
- Testes de integração
- Testes baseados em propriedades

Prof. Jose Emiliano



Testes Automatizados



- Testes unitários
- Testes de integração
 - Rodam sobre testes unitários
 - Verifica interação entre componentes (internos ou externos)
- Testes baseados em propriedades

Prof. Jose Emiliano



Testes Automatizados

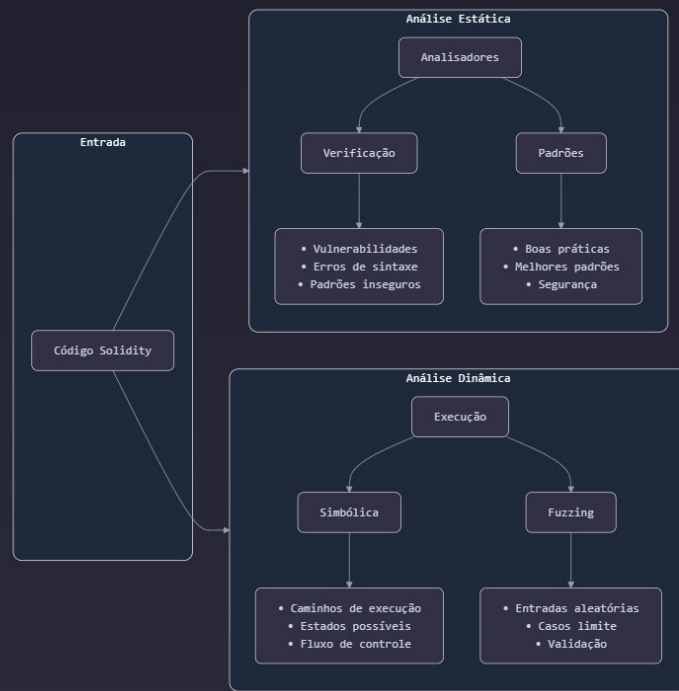


- Testes unitários
- Testes de integração
- Testes baseados em propriedades
 - Os testes baseados em propriedades são o processo de verificar se um contrato inteligente cumpre com alguma propriedade definida.
 - Analise estática
 - Análise dinâmica (fuzzing)

Prof. Jose Emiliano



• Resumo



Análise dinâmica



```
// Contrato A: contrato testado
contract ContratoA {
    bool public statusValido = false;
    int256 public codigoAtualizacao = 0;
    address public owner;
    constructor() {
        owner = msg.sender;
    }
    function atualizarStatus(bool novoStatus) public {
        statusValido = novoStatus;
    }
    function atualizarCodigo(int256 cod) public {
        codigoAtualizacao = cod;
    }
    function verificarStatus() public view returns (bool) {
        return statusValido;
    }
    function consultarCodigo() public view returns (int256) {
        return codigoAtualizacao;
    }
}
```

Prof. Jose Emiliano



Análise dinâmica



```
// Contrato B: contrato que interage/testa o A
interface IContratoA{
    function verificarStatus() external view returns (bool);
}
contract ContratoB {
    address contratoAAddress;
    constructor(address _contratoAAddress) {
        contratoAAddress = _contratoAAddress;
    }

    function testarContratoA() public view returns (string memory) {
        IContratoA contratoA = IContratoA(contratoAAddress);
        bool status = contratoA.verificarStatus();
        if (status) {
            return "Contrato A está válido!";
        } else {
            return "Contrato A NÃO está válido.";
        }
    }
}
```

Prof. Jose Emiliano



Prática



- Elabore o contrato B de forma a permitir testar todas as funcionalidades dinâmicas do contrato A
- O contrato A não está na sua máquina
 - Esta deployed na rede sepolia
 - A ABI do contrato está disponível no github

Prof. Jose Emiliano



Exemplo 2



```
contract ContratoBanco {
    mapping(address => uint256) private saldos;

    // Depositar dinheiro
    function depositar() external payable {
        saldos[msg.sender] += msg.value;
    }

    // Sacar dinheiro
    function sacar(uint256 valor) external {
        require(saldos[msg.sender] >= valor, "Saldo insuficiente");
        saldos[msg.sender] -= valor;
        payable(msg.sender).transfer(valor);
    }

    // Verificar saldo
    function consultarSaldo(address conta) external view returns (uint256) {
        return saldos[conta];
    }
}
```

Prof. Jose Emiliano



Prática



- Elabore um contrato auditor que interaja com o contrato banco:
 - Checar a função de saldo mínimo
 - Checar a existência de saldo correto de um cliente específico

Prof. Jose Emiliano

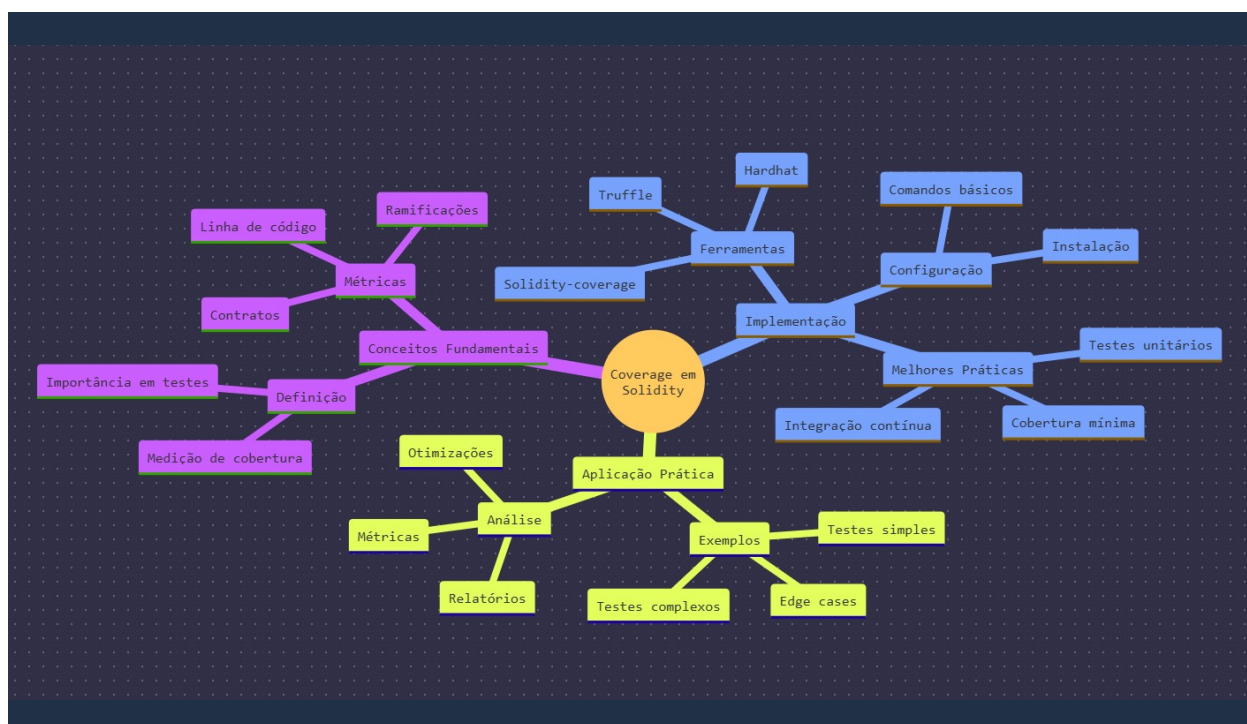


Coverage

Identificar o conceito e emprego de ferramentas de análise de cobertura

Prof. Jose Emiliano





Testes de Cobertura



- Coverage Tests
 - Coverage (ou cobertura de código) é uma métrica que mostra quais partes do seu contrato foram realmente testadas. Ela ajuda a responder:
 - “Será que meus testes cobrem todos os fluxos possíveis do contrato?”
 - Essa análise é essencial para garantir que seu contrato está seguro, funcional e sem brechas.



Testes de Cobertura



Métrica	Significado
% Stmts	Percentual de comandos (statements) executados pelos testes (ex. chamadas de função)
% Branch	Percentual de fluxos condicionais cobertos (ex: if, else, require, revert)
% Funcs	Percentual de funções chamadas ao menos uma vez nos testes
% Lines	Percentual de linhas de código executadas durante os testes
Uncovered Lines	Linhas que não foram testadas ou parcialmente testadas

Prof. Jose Emiliano



Exemplos



- **Statements:**
 - Usando `require(x > 0)` e `x = x + 1`, são dois statements. Se só o segundo for testado, o coverage será 50%.
- **Branches:**
 - Um `if (x > 0)` tem dois ramos: verdadeiro e falso. É necessário testar ambos para 100% de branches.
- **Functions:**
 - Se você tem 5 funções e só testa 3, o coverage será 60%.

Prof. Jose Emiliano



Exemplos



- **Lines:**
 - É a métrica mais direta — quantas linhas foram executadas nos testes.
- **Uncovered Lines:**
 - Mostra quais partes do código ainda não foram testadas, ajudando a identificar lacunas.

Prof. Jose Emiliano



Relatório de Cobertura



- É um resumo das atividades de cobertura de código gerado pela plataforma.
 - Acessível pelo arquivo `coverage/index.html` gerado após rodar `npx hardhat coverage`
 - é possível visualizar um relatório interativo com cores (verde, amarelo, vermelho) que indicam o nível de cobertura por linha e função.

Prof. Jose Emiliano



Testes de Cobertura



```
contract Bank {  
    mapping (address => uint256) public balances;  
  
    function deposit() public payable {  
        require(msg.value > 0, "Deposit amount must be greater than 0");  
        address sender = msg.sender;  
        balances[sender] += msg.value;  
    }  
  
    function withdraw(uint256 amount) public {  
        require(balances[msg.sender] >= amount, "Insufficient balance");  
        balances[msg.sender] -= amount;  
        payable(msg.sender).transfer(amount);  
    }  
}
```

Prof. Jose Emiliano

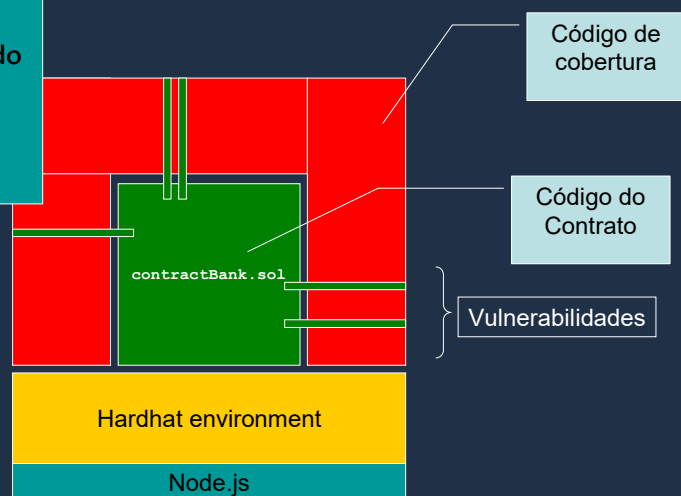


Testes de Cobertura



•Padrões de teste

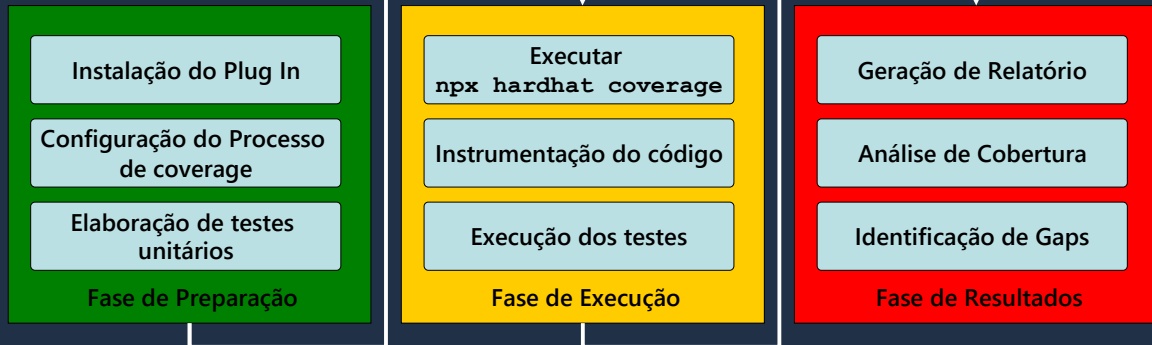
- Teste de comportamento esperado
- Teste de casos de erro
- Teste de limites
- Teste de múltiplos usuários



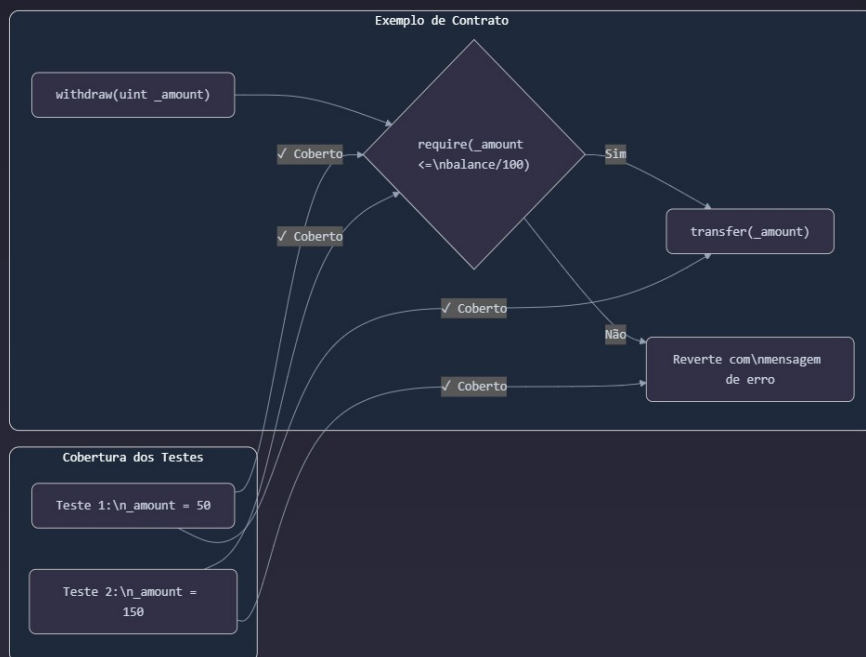
<https://docs.openzeppelin.com/test-environment/0.1/>



Etapas



Prof. Jose Emiliano





Testes de Cobertura



- Instalação
 - Instale o plugin:
 - `npm install --save-dev solidity-coverage`
 - Adicione no `hardhat.config.js`:
 - `require("solidity-coverage");`
 - Execute:
 - `npx hardhat coverage`

Prof. Jose Emiliano



Atividade prática



- Elabore o contrato de exemplo (`simpleDEx.sol`)
- Elabore uma rotina de testes para o contrato de exemplo (arquivo: `simpleDEx.test.js`)
- Instale os módulos de `solidity-coverage`
- Execute o `hardhat test`
- Execute o `hardhat coverage`

Prof. Jose Emiliano


```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.21;

contract SimpleDEX {
    struct Order {
        address seller;
        uint256 amount;
        uint256 price; // preço por unidade
    }
    mapping(uint256 => Order) public orders;
    uint256 public nextOrderId;

    function createOrder(uint256 amount, uint256 price) external {
        require(amount > 0 && price > 0, "Valores inválidos");
        orders[nextOrderId] = Order(msg.sender, amount, price);
        nextOrderId++;
    }

    function cancelOrder(uint256 orderId) external {
        require(orders[orderId].seller == msg.sender, "Não é o dono da ordem");
        delete orders[orderId];
    }

    function getOrder(uint256 orderId) external view returns (address, uint256, uint256) {
        Order memory ord = orders[orderId];
        return (ord.seller, ord.amount, ord.price);
    }
}
```



Script de teste



```
const { expect } = require("chai");
const { ethers } = require("hardhat");

describe("SimpleDEX", function () {
    let dex, owner, addr1;

    beforeEach(async () => {
        [owner, addr1] = await ethers.getSigners();
        const DEX = await ethers.getContractFactory("SimpleDEX");
        dex = await DEX.deploy();
        await dex.waitForDeployment();
    });

    it("Deve criar uma ordem válida", async () => { //CREATE ORDER
        await dex.createOrder(10, 5); // amount = 10, price = 5
        const ordem = await dex.getOrder(0);
        expect(ordem[0]).to.equal(owner.address);
        expect(ordem[1]).to.equal(10);
        expect(ordem[2]).to.equal(5);
    });
});
```



Script de teste



```
it("Deve aumentar o ID da próxima ordem", async () => { //TESTA ORDER ID
  await dex.createOrder(1, 1);
  await dex.createOrder(2, 2);
  expect(await dex.nextOrderId()).to.equal(2);
});
it("Deve cancelar uma ordem do próprio usuário", async () => { //TESTA CANCEL ORDER
  await dex.createOrder(3, 3);
  await dex.cancelOrder(0);
  const ordem = await dex.getOrder(0);
  expect(ordem[1]).to.equal(0); // amount deve ser zero (apagado)
});
it("Não deve permitir cancelar ordem de outro usuário", async () => {
  await dex.connect(addr1).createOrder(4, 4);
  await expect(dex.connect(owner).cancelOrder(0)).to.be.revertedWith("Não é o dono da ordem");
});
it("Não deve criar ordem com valor zero", async () => { // TESTA CREATE ORDER
  await expect(dex.createOrder(0, 0)).to.be.revertedWith("Valores inválidos");
});
```



Certificação de Contratos

Apresentar um roteiro para a certificação
de contratos solidity

Prof. Jose Emiliano



Introdução



- A elaboração de um smart contract é uma atividade de diversas etapas
 - Cada etapa é desenvolvida por uma equipe
 - Possui riscos inerentes
 - Ferramentas de testes não resolvem todos os problemas

Prof. Jose Emiliano



Ciclo de vida



- Escrita do contrato
 - Conformidade com regras de negócio
- Auditoria e testes de segurança
 - Verificação da conformidade com boas práticas de desenvolvimento e segurança
 - Testes estáticos e dinâmicos (Hardhat tests e Coverage)
- Implantação na rede Ethererum
 - Verificação e publicação (Etherscan)

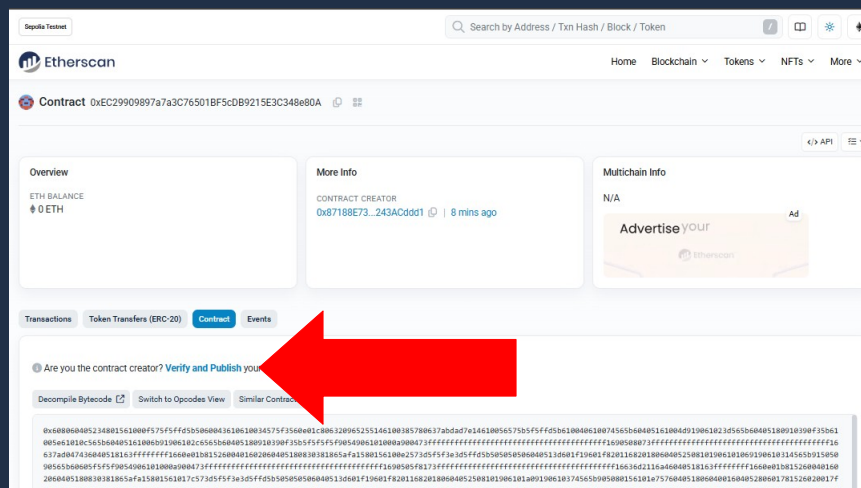
Prof. Jose Emiliano



Publicação



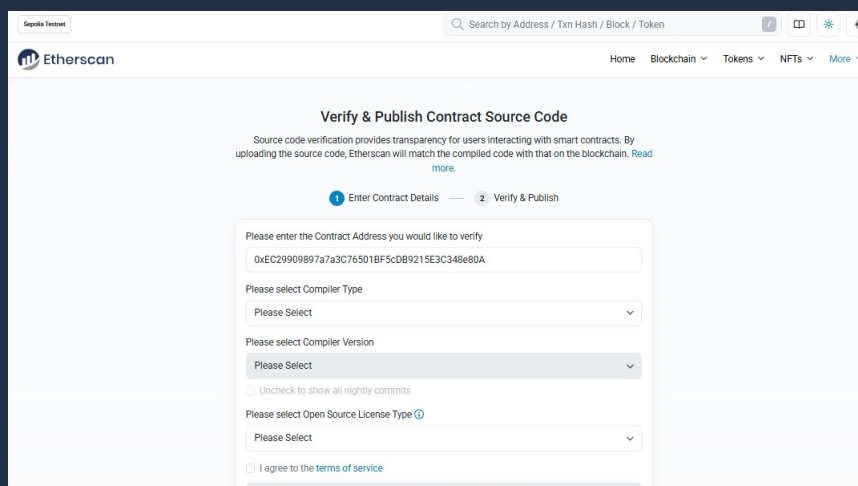
- Acesse Etherscan com o endereço do seu contrato



Publicação



- Preencha os parâmetros





Publicação



- Conclua a validação

Upload Contract Source Code

1. If the contract compiles correctly at [REMX](#), it should also compile correctly here.

2. We have limited support for verifying contracts created by another contract and there is a timeout of up to 45 seconds for each contract compiled.

3. For programmatic contract verification, check out the [Contract API Endpoint](#).

Contract Address: `0xEC29909897a7a3C76501BF5cDB9215E3C348e80A`

Compiler Type: `SINGLE FILE / CONCATENATED METHOD`

Compiler Version: `v0.8.0+commit.c7d478e`

Enter the Solidity Contract Code below *

[Fetch from Gist](#)

Prof. Jose Emiliano



Auditorias



- Auditoria de segurança pode ser manual ou automatizada:
 - MythX / Diligence Fuzzing
 - Slither (Python)
 - OpenZeppelin Defender

Prof. Jose Emiliano



Benefícios



- **Segurança reforçada**
 - Auditorias detectam vulnerabilidades como ataques de reentrância, overflow, ou problemas de autorização que podem ser explorados por hackers.
 - Credibilidade perante investidores e usuários.
 - Um contrato certificado transmite confiança para quem interage com ele — seja em uma aplicação DeFi, NFT ou DAO.
 - **Maior conformidade legal e regulatória**
 - Certificações ajudam projetos a se alinharem com práticas do setor e exigências legais, especialmente se forem listados em exchanges ou lidarem com ativos financeiros.

Prof. Jose Emiliano



Benefícios



- **Segurança reforçada**
 - **Transparência total**
 - Um contrato verificado e certificado deixa claro que o código está aberto ao público e que passou por escrutínio técnico.
 - **Prevenção de prejuízos**
 - Evitar bugs significa evitar perdas de tokens, reputação ou até falência de projetos.

Prof. Jose Emiliano



Empresas certificadoras



Empresa	Especialidade	Website
CertiK	Auditoria automatizada e certificação de segurança para contratos inteligentes	Certik
ConsenSys Diligence	Auditorias manuais detalhadas e ferramentas como MythX	ConsenSys Diligence
Trail of Bits	Segurança cibernética e revisão de código Solidity	Trail of Bits
OpenZeppelin	Ferramentas de segurança e contratos auditados	OpenZeppelin

Prof. Jose Emiliano



Ferramentas



- [Slither](#)
 - Uma ferramenta da Trail of Bits para análise estática.
- [Echidna](#)
 - Ferramenta da Trail of Bits para fuzzing.
- [Manticore](#)
 - Ferramenta da Trail of Bits para execução simbólica.
- [MythX](#)
 - Ferramenta de segurança para contratos inteligentes (paga).

Prof. Jose Emiliano



Rekt (wrecked)



- Este teste, adaptado para o mundo Web3 pela Trail of Bits, é utilizado para avaliar o comprometimento da equipe de desenvolvimento com a segurança.
1. Você documentou todos os atores, papéis e privilégios?
 2. Mantém documentação de todos os serviços externos, contratos e oráculos dos quais depende?
 3. Possui um plano de resposta a incidentes por escrito e que já tenha sido testado?
 4. Documenta as melhores formas de atacar seu próprio sistema?
 5. Realiza verificação de identidade e antecedentes de todos os colaboradores?

Prof. Jose Emiliano



Rekt (wrecked)



6. Tem alguém na equipe com foco específico em segurança como parte do seu papel?
7. Requer chaves de segurança físicas para acessar os sistemas de produção?
8. Seu sistema de gestão de chaves exige múltiplas pessoas e etapas físicas?
9. Define invariantes essenciais para o seu sistema e os testa a cada commit?
10. Utiliza as melhores ferramentas automatizadas para detectar problemas de segurança no seu código?
11. Submete o projeto a auditorias externas e mantém um programa de divulgação de vulnerabilidades ou recompensas por bugs?
12. Considerou e mitigou as formas de abuso contra os usuários do seu sistema?

Prof. Jose Emiliano



Ferramentas



- [Mythrill](#)
 - Versão gratuita do MythX.
- [ETH Security Toolbox](#)
 - Script para criar containers Docker configurados com as ferramentas de segurança da Trail of Bits.
- [ethersplay](#)
 - Disassembler para Ethereum.
- [Consensys Security Tools](#)
 - Uma lista de ferramentas da Consensys.

Prof. Jose Emiliano



Remix IDE Plug-Ins



- Contract Verification
 - Submete o contrato para avaliação em diversos provedores de serviços:
 - Sourcify
 - Etherscan
 - Blockscout
 - Routerscan

Prof. Jose Emiliano



Forge / Foundry

Identificar a alternativa ao hardhat

Prof. Jose Emiliano





Forge



- É uma ferramenta (Rust) para desenvolvimento de contratos inteligentes em Solidity, parte do ecossistema Foundry
 - Permite compilar, testar, depurar e implantar contratos inteligentes Ethereum escritos em Solidity.
 - É parte do conjunto de ferramentas Foundry, que também inclui Cast (interação com contratos), Anvil (nó local Ethereum) e Chisel (REPL para Solidity).

Prof. Jose Emiliano



Forge



- Principais recursos do Forge
 - Compilação rápida: Detecta automaticamente a versão do compilador Solidity necessária e recompila apenas os arquivos modificados.
 - Testes em Solidity
 - Fuzzing e testes de invariantes: Identifica casos extremos e garante propriedades do sistema em diferentes entradas.
 - Depuração interativa: Permite rastrear a execução dos contratos e visualizar alterações no armazenamento.
 - Suporte a múltiplos backends EVM: Ideal para simulações e testes em diferentes ambientes.

Prof. Jose Emiliano



Forge



- **Comparação com outras ferramentas**
 - Forge é construído para utilizar Solidity
 - É considerado uma alternativa mais rápida e leve ao Hardhat e Truffle, especialmente por não depender de JavaScript e por usar submódulos Git para gerenciar dependências

Prof. Jose Emiliano



Comparativo



Característica	Hardhat	Forge
Linguagem	JS/TS	Solidity
Velocidade	Boa	Muito rápida
Ferramentas extras	Plugins variados	Tools embutidas no Foundry
Experiência dev	Mais flexível	Mais enxuta e eficiente

Prof. Jose Emiliano



Ecosistema



Elemento (comando)	Descrição
Compilador (forge build)	Compila os contratos Solidity com extrema velocidade usando LLVM.
Testes (forge test)	Executa testes em Solidity com suporte a fuzzing e cobertura de código.
Deploy (forge create)	Faz o deploy de contratos para redes locais, testnet ou mainnet.
Interação (forge call, forge send)	Permite chamar funções e enviar transações para contratos.
Snapshot (forge snapshot)	Gera e compara snapshots de testes para garantir consistência.
Script (forge script)	Executa scripts automatizados para deploy, interações e mais.
Trace (forge trace)	Diagnóstico avançado de transações, mostrando chamadas internas.
Coverage (forge coverage)	Mede cobertura de testes para contratos Solidity.
Doc (forge doc)	Gera documentação dos contratos em Markdown.
Configuração (foundry.toml)	Arquivo central que gerencia settings do projeto como paths, flags etc.

Prof. Jose Emiliano



Forge



- **Instalação:**

- **Use:**

```
# Download script do instalador foundry `foundryup`  
curl -L https://foundry.paradigm.xyz | bash  
# Instale forge, cast, anvil, chisel  
foundryup  
# Instale a versão mais recente  
foundryup -i nightly  
# Crie o projeto  
forge init <nome>
```

Prof. Jose Emiliano



Forge



- Se você quiser saber onde o Foundry em si está instalado (não o projeto), ele geralmente fica em:
 - Linux/macOS: ~/.foundry
 - Windows: C:/Users/<SeuUsuário>/.foundry

Prof. Jose Emiliano



Forge



- Estrutura de projeto com Forge
 - src/: Código dos contratos inteligentes
 - test/: Testes escritos em Solidity
 - script/: Scripts para implantação e interação
 - lib/: Dependências externas (como OpenZeppelin)
 - foundry.toml: Arquivo de configuração do projeto

Prof. Jose Emiliano



Forge



- Principais comandos:
 - `forge build`
 - `forge build -- contracts src/<nome contrato>`
 - `forge clean`
 - `forge build -v`
 - `forge test -vvvv` (verbose logs)
 - `forge test --gas-report`

Prof. Jose Emiliano



Forge



- `vm` é uma interface especial que permite manipular o estado da blockchain durante testes e scripts.
 - Fornecida automaticamente quando você herda de `Test.sol` ou `Script.sol` da biblioteca `forge-std`.
 - Manipular remetente (`msg.sender`)
 - `vm.prank(address)` — simula uma chamada como se fosse feita por outro endereço
 - `vm.startPrank(address)` — simula múltiplas chamadas seguidas com outro remetente
 - Simular tempo e blocos
 - `vm.warp(timestamp)` — altera o tempo do bloco
 - `vm.roll(blockNumber)` — altera o número do bloco

Prof. Jose Emiliano



Forge



- Manipular saldo e storage
 - `vm.deal(address, amount)` — define o saldo de um endereço
 - `vm.store(address, slot, value)` — escreve diretamente em um slot de armazenamento
- Esperar erros e eventos
 - `vm.expectRevert()` — espera que uma chamada reverta
 - `vm.expectEmit()` — espera que um evento específico seja emitido
- Logar valores
 - `console.log(...)` — imprime valores no terminal (estilo Hardhat)

Prof. Jose Emiliano



Exemplo



```
// src/LazyDemo.sol
pragma solidity ^0.8.0;

contract LazyDemo {
    function multiply(uint256 a, uint256 b) public pure returns
        (uint256) {
        require(a < 1e18 && b < 1e18, "Valores muito grandes");
        return a * b;
    }
}
/*
Condições de teste:
1. O contrato deve multiplicar corretamente
2. O contrato deve falhar se valores forem muito grandes
*/
```

Prof. Jose Emiliano


```

const { expect } = require("chai");
const { ethers } = require("hardhat");

describe("LazyDemo", function () {
  let lazy;

  beforeEach(async () => {
    const LazyDemo = await ethers.getContractFactory("LazyDemo");
    lazy = await LazyDemo.deploy();
    await lazy.deployed();
  });

  it("deve multiplicar corretamente", async function () {
    expect(await lazy.multiply(2, 3)).to.equal(6);
    expect(await lazy.multiply(0, 5)).to.equal(0);
  });

  it("deve falhar se valores forem muito grandes", async function () {
    await expect(lazy.multiply(1e18, 2)).to.be.revertedWith("Valores muito grandes");
    await expect(lazy.multiply(2, 1e18)).to.be.revertedWith("Valores muito grandes");
  });
});

```

```

pragma solidity ^0.8.0
    forge-std/Test.sol
import "../src/LazyDemo.sol";

contract FuzzDemoTest is Test
    FuzzDemo demo;

    function setUp() public {
        demo = new FuzzDemo();
    }

    function testMultiplyFuzz(uint256 a, uint256 b) public {
        // Limita os valores para evitar overflow
        vm.assume(a < 1e18);
        vm.assume(b < 1e18);

        uint256 result = demo.multiply(a, b);

        // Verifica se o resultado bate com o esperado
        assertEq(result, a * b);
    }
}

```



Setup

Prof. Jose Emiliano



Setup



- Na pasta `c:/projetos_node`, crie uma pasta "foundry"
- Inicie o WSL
- Copie e cole o script "install.bash"
- Execute "foundryup"

Prof. Jose Emiliano

```
user@MyLenovo: /mnt/c/proj x + v
FOUNDRY Portable and modular toolkit
          for Ethereum Application Development
          written in Rust.

.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x

Repo      : https://github.com/foundry-rs/foundry
Book      : https://book.getfoundry.sh/
Chat      : https://t.me/foundry_rs/
Support   : https://t.me/foundry_support/
Contribute: https://github.com/foundry-rs/foundry/blob/master/CONTRIBUTING.md

.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x

foundryup: checking if foundryup is up to date...
foundryup: foundryup is up to date.
foundryup: installing foundry (version stable, tag stable)
foundryup: checking if forge, cast, anvil, and chisel for stable version are already installed
##### 100.0%
foundryup: no attestation found for this release, skipping SHA verification
foundryup: binaries not found or do not match expected hashes, downloading new binaries
foundryup: downloading forge, cast, anvil, and chisel for stable version
##### 100.0%
forge
cast
anvil
chisel
foundryup: downloading manpages
##### 100.0%
```



Setup



- Crie um projeto chamado "forge-teste" :
 - `forge init forge-teste`
- Verifique a estrutura de pastas



Setup



- Estrutura de pastas

```
forge-teste/  
├── src/  
│   └── Counter.sol  
├── test/  
│   └── Counter.t.sol  
├── out/  
│   ├── Counter.sol  
│   └── Counter.t.sol  
├── foundry.toml (configurações do ambiente)  
└── .env
```

Arquivos compilados
(.json) : ABI + bytecode

Prof. Jose Emiliano



Executando projeto

Prof. Jose Emiliano



Contrato de Teste



```
contract Bank {
    mapping (address => uint256) public balances
    function deposit() public payable {
        require(msg.value > 0, "Deposite mais que 0");
        address sender = msg.sender;
        balances[sender] += msg.value;
    }
    function withdraw(uint256 amount) public {
        require(balances[msg.sender] >= amount, "Saldo insuficiente");
        balances[msg.sender] -= amount;
        payable(msg.sender).transfer(amount);
    }
}
```

Prof. Jose Emiliano

```
pragma solidity ^0.8.0;
import "forge-std/Test.sol";
import "../src/Bank.sol";
contract BankTest is Test {
    Bank bank;
    address user = address(0xABCD);
    function setUp() public {
        bank = new Bank();
        vm.deal(user, 10 ether); // Dá saldo inicial ao usuário
    }
    function testFuzzDepositWithdraw(uint256 depositAmount, uint256 withdrawAmount) public {
        // Garante que os valores sejam razoáveis
        vm.assume(depositAmount > 0 && depositAmount <= 5 ether);
        vm.assume(withdrawAmount <= depositAmount);
        // Simula o usuário fazendo depósito
        vm.prank(user);
        bank.deposit{value: depositAmount}();
        // Verifica se o saldo foi atualizado
        assertEq(bank.balances(user), depositAmount);
        // Simula o saque
        vm.prank(user);
        bank.withdraw(withdrawAmount);
        // Verifica se o saldo foi reduzido corretamente
        assertEq(bank.balances(user), depositAmount - withdrawAmount);
    }
}
```

function deposit() public payable
{}

function withdraw(uint256 amount)
public
{}



Atividade prática



- Execute os testes do contrato Bank
 - Compile e execute os testes
 - Verifique o consumo de gas do contrato

Prof. Jose Emiliano



Deploy



- A funcionalidade chamada `forge create` permite fazer o deploy diretamente para redes como Ethereum Mainnet, Sepolia, Goerli, etc.
 - No arquivo `.env`, deve conter:
 - `PRIVATE_KEY="sua_chave_privada"`
 - `RPC_URL="https://sepolia.infura.io/v3/seu_projeto"`

Prof. Jose Emiliano



Deploy



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

import "forge-std/Script.sol";
import "../src/MeuContrato.sol";

contract DeployScript is Script {
    function run() external {
        vm.startBroadcast();
        MeuContrato contrato = new MeuContrato();
        console.log("Contrato implantado em:", address(contrato));
        vm.stopBroadcast();
    }
}
```

Prof. Jose Emiliano



Deploy



- Executar script

```
forge script script/Deploy.s.sol
--rpc-url SEU_RPC
--private-key SUA_CHAVE
--broadcast
```

Prof. Jose Emiliano



Prática

Prof. Jose Emiliano



• Solidity Donation

- O contrato inteligente Solidity Donation implementado na tem como objetivo permitir o recebimento de doações
- Funcionalidades Principais:
- Arrecadação de Fundos (doar):
 - Permite que qualquer usuário envie uma quantia em Ether ao contrato, desde que seja maior que zero.
 - Registra o endereço do doador em um array público doadores.
 - Atualiza a variável totalDonate, que representa o total de Ether recebido.
- Gestão de Beneficiário (dono):
 - O endereço que criou o contrato é definido como o dono, sendo o único autorizado a resgatar os valores arrecadados.
- Resgate de Fundos (resgatar):
 - Disponível apenas para o dono, permite a retirada total do saldo Ether do contrato.
 - Inclui verificações de segurança para garantir que apenas o dono execute essa ação e que o contrato possua saldo disponível.



Donation
contractDonation

```
Private:  
totalDonate: uint256  
Public:  
dono: address  
saldos: mapping(address=>uint256)  
doadores: address[]  
  
Public:  
<<navable>> doar()
```




Tarefas



- 1 – Escreva o contrato (contratoDonation.sol)
- 2 - Compile o contrato usando o Hardhat
- 3 – Elabore o script de teste para Hardhat (.js)
- 4 – **IMPORTANTE:** Deve concluir a tarefa no tempo de aula.

Tarefa vale: 30 pontos.

Prof. Jose Emiliano



Exploradores



- Exploradores de Blockchain
 - Essas plataformas mostram informações detalhadas de contratos, transações e endereços.
 - Etherscan: O mais conhecido. Se o contrato foi verificado, você pode visualizar o código fonte diretamente na aba "Contract". Também permite interagir com o contrato via interface.
 - Ethplorer: Focado em tokens, oferece uma visão simplificada dos contratos associados.
 - Blockchair: Combina visualizações de várias redes e oferece busca avançada por parâmetros técnicos.

Prof. Jose Emiliano

