Security Testing Documentation

Overview

Audityzer provides comprehensive security testing capabilities for Web3 applications, smart contracts, and DeFi protocols. This documentation covers security testing methodologies, tools, and best practices.

Security Testing Framework

Core Components

- **Vulnerability Scanner**: Automated detection of common Web3 vulnerabilities
- **AI-Powered Analysis**: Machine learning-based threat detection
- **Fuzzing Engine**: Automated input testing and edge case discovery
- **Static Analysis**: Code analysis without execution
- **Dynamic Analysis**: Runtime behavior analysis
- **Penetration Testing**: Simulated attack scenarios

Supported Vulnerability Types

- Reentrancy attacks
- Integer overflow/underflow
- Unauthorized access
- Price manipulation
- Flash loan attacks
- MEV exploitation
- Bridge vulnerabilities
- Governance attacks

Quick Start

Basic Security Scan

```
# Scan a smart contract
audityzer scan --contract 0x1234567890123456789012345678901234567890

# Scan with specific tests
audityzer scan --contract 0x123... --tests reentrancy,overflow,access

# Comprehensive security audit
audityzer audit --target 0x123... --deep-scan --ai-analysis
```

Configuration

```
// security.config.js
module.exports = {
  scanning: {
    enabled: true,
    interval: 300000, // 5 minutes
    depth: 'comprehensive',
    aiAnalysis: true
  },

  vulnerabilities: {
    reentrancy: { enabled: true, severity: 'high' },
    overflow: { enabled: true, severity: 'medium' },
    access: { enabled: true, severity: 'critical' },
    priceManipulation: { enabled: true, severity: 'high' }
  },

  reporting: {
    format: 'json',
    includeRemediation: true,
    confidenceThreshold: 0.8
  }
};
```

Vulnerability Detection

Reentrancy Detection

```
const { ReentrancyDetector } = require('audityzer');

describe('Reentrancy Tests', () => {
  let detector;

  beforeEach(() => {
    detector = new ReentrancyDetector({
      network: 'ethereum',
      gasLimit: 8000000
    });
  });

  test('should detect classic reentrancy vulnerability', async () => {
    const contractCode = `
      contract VulnerableContract {
        mapping(address => uint) public balances;

        function withdraw() public {
          uint amount = balances[msg.sender];
          (bool success,) = msg.sender.call{value: amount}("");
          require(success);
          balances[msg.sender] = 0; // State change after external call
        }
      }
    `;

    const result = await detector.analyze(contractCode);

    expect(result.vulnerabilities).toContainEqual(
      expect.objectContaining({
        type: 'reentrancy',
        severity: 'high',
        line: expect.any(Number),
        description: expect.stringContaining('external call before state change')
      })
    );
  });

  test('should detect cross-function reentrancy', async () => {
    const contractCode = `
      contract CrossFunctionReentrancy {
        mapping(address => uint) public balances;

        function withdraw() public {
          uint amount = balances[msg.sender];
          (bool success,) = msg.sender.call{value: amount}("");
          require(success);
          balances[msg.sender] = 0;
        }

        function transfer(address to, uint amount) public {
          require(balances[msg.sender] >= amount);
          balances[msg.sender] -= amount;
          balances[to] += amount;
        }
      }
    `;
```

```
    const result = await detector.analyzeCrossFunctionReentrancy(contractCode);
    expect(result.crossFunctionVulnerabilities).toHaveLength(1);
  });
});
```

Integer Overflow Detection

```
const { OverflowDetector } = require('auditzer');

describe('Overflow Tests', () => {
  test('should detect integer overflow vulnerability', async () => {
    const detector = new OverflowDetector();

    const contractCode = `
      contract OverflowVulnerable {
        uint8 public counter = 255;

        function increment() public {
          counter++; // Potential overflow
        }

        function add(uint8 a, uint8 b) public pure returns (uint8) {
          return a + b; // Potential overflow
        }
      }
    `;

    const result = await detector.analyze(contractCode);

    expect(result.vulnerabilities).toContainEqual(
      expect.objectContaining({
        type: 'integer_overflow',
        severity: 'medium',
        operation: 'addition'
      })
    );
  });

  test('should verify SafeMath usage', async () => {
    const detector = new OverflowDetector();

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

      contract SafeContract {
        using SafeMath for uint256;
        uint256 public value;

        function add(uint256 a) public {
          value = value.add(a);
        }
      }
    `;

    const result = await detector.analyze(safeCode);
    expect(result.vulnerabilities).toHaveLength(0);
  });
});
```

Access Control Testing

```
const { AccessControlTester } = require('adityzer');

describe('Access Control Tests', () => {
  test('should detect missing access controls', async () => {
    const tester = new AccessControlTester();

    const contractCode = `
      contract UnsafeContract {
        address public owner;
        uint256 public balance;

        constructor() {
          owner = msg.sender;
        }

        function withdraw() public {
          // Missing onlyOwner modifier
          payable(msg.sender).transfer(balance);
        }

        function setOwner(address newOwner) public {
          // Missing access control
          owner = newOwner;
        }
      }
    `;

    const result = await tester.analyze(contractCode);

    expect(result.vulnerabilities).toContainEqual(
      expect.objectContaining({
        type: 'missing_access_control',
        function: 'withdraw',
        severity: 'critical'
      })
    );
  });

  test('should verify role-based access control', async () => {
    const tester = new AccessControlTester();

    const result = await tester.testRoleBasedAccess({
      contract: '0x123...',
      roles: ['admin', 'user', 'moderator'],
      functions: ['mint', 'burn', 'pause']
    });

    expect(result.accessMatrix).toBeDefined();
    expect(result.violations).toHaveLength(0);
  });
});
```

DeFi Protocol Testing

AMM Security Testing

```
const { AMMSecurityTester } = require('adityzer');

describe('AMM Security Tests', () => {
  let tester;

  beforeEach(async () => {
    tester = new AMMSecurityTester({
      protocol: 'uniswap-v2',
      network: 'ethereum'
    });
    await tester.initialize();
  });

  test('should test for price manipulation attacks', async () => {
    const result = await tester.testPriceManipulation({
      pair: 'ETH/USDC',
      manipulationAmount: ethers.utils.parseEther('1000'),
      targetPriceChange: 0.1 // 10%
    });

    expect(result.vulnerable).toBe(false);
    expect(result.maxPriceImpact).toBeLessThan(0.05); // 5%
  });

  test('should test for flash loan attacks', async () => {
    const result = await tester.testFlashLoanAttack({
      pair: 'ETH/USDC',
      flashLoanAmount: ethers.utils.parseEther('10000'),
      strategy: 'price_manipulation'
    });

    expect(result.profitable).toBe(false);
    expect(result.protections).toContain('slippage_protection');
  });

  test('should test liquidity provision security', async () => {
    const result = await tester.testLiquidityProvision({
      token0: 'ETH',
      token1: 'USDC',
      amount0: ethers.utils.parseEther('10'),
      amount1: ethers.utils.parseUnits('20000', 6)
    });

    expect(result.impermanentLossRisk).toBeLessThan(0.1);
    expect(result.slippageProtected).toBe(true);
  });
});
```

Lending Protocol Testing

```
const { LendingSecurityTester } = require('audityzer');

describe('Lending Protocol Tests', () => {
  test('should test liquidation mechanisms', async () => {
    const tester = new LendingSecurityTester({
      protocol: 'compound',
      network: 'ethereum'
    });

    const result = await tester.testLiquidation({
      borrower: '0x123...',
      collateral: 'ETH',
      debt: 'USDC',
      priceDropPercentage: 0.3 // 30% price drop
    });

    expect(result.liquidationTriggered).toBe(true);
    expect(result.healthFactor).toBeLessThan(1);
    expect(result.liquidationBonus).toBeGreaterThan(0);
  });

  test('should test oracle manipulation resistance', async () => {
    const tester = new LendingSecurityTester();

    const result = await tester.testOracleManipulation({
      asset: 'ETH',
      manipulationStrategy: 'flash_loan',
      targetPriceChange: 0.2
    });

    expect(result.resistant).toBe(true);
    expect(result.protections).toContain('time_weighted_average');
  });
});
```

AI-Powered Security Analysis

Machine Learning Detection

```
const { AISecurityAnalyzer } = require('audityzer');

describe('AI Security Analysis', () => {
  test('should detect anomalous patterns', async () => {
    const analyzer = new AISecurityAnalyzer({
      model: 'vulnerability-detection-v2',
      confidence: 0.8
    });

    const contractCode = `
      // Complex contract with potential vulnerabilities
    `;

    const result = await analyzer.analyze(contractCode);

    expect(result.confidence).toBeGreaterThan(0.8);
    expect(result.patterns).toBeDefined();
    expect(result.recommendations).toHaveLength(result.vulnerabilities.length);
  });

  test('should provide remediation suggestions', async () => {
    const analyzer = new AISecurityAnalyzer();

    const result = await analyzer.generateRemediation({
      vulnerability: {
        type: 'reentrancy',
        location: 'line 45',
        severity: 'high'
      },
      contractCode: '...'
    });

    expect(result.remediation).toContain('ReentrancyGuard');
    expect(result.codeExample).toBeDefined();
    expect(result.explanation).toBeDefined();
  });
});
```

Pattern Recognition

```
const { PatternRecognizer } = require('adityzer');

describe('Pattern Recognition', () => {
  test('should identify suspicious transaction patterns', async () => {
    const recognizer = new PatternRecognizer({
      timeWindow: 3600, // 1 hour
      minConfidence: 0.7
    });

    const transactions = [
      // Array of transaction data
    ];

    const result = await recognizer.analyzeTransactions(transactions);

    expect(result.suspiciousPatterns).toBeDefined();
    expect(result.riskScore).toBeGreaterThan(0);
  });
});
```

Fuzzing and Property Testing

Smart Contract Fuzzing

```
const { ContractFuzzer } = require('audityzer');

describe('Contract Fuzzing', () => {
  test('should fuzz contract functions', async () => {
    const fuzzer = new ContractFuzzer({
      contract: '0x123...',
      iterations: 10000,
      strategy: 'random'
    });

    const result = await fuzzer.fuzzAllFunctions({
      gasLimit: 8000000,
      timeout: 300000 // 5 minutes
    });

    expect(result.crashes).toHaveLength(0);
    expect(result.coverage).toBeGreaterThan(0.9);
    expect(result.uniqueInputs).toBeGreaterThan(1000);
  });

  test('should test invariant properties', async () => {
    const fuzzer = new ContractFuzzer();

    const result = await fuzzer.testInvariants({
      contract: '0x123...',
      invariants: [
        'totalSupply >= sum(balances)',
        'balance[user] >= 0',
        'allowance[owner][spender] >= 0'
      ]
    });

    expect(result.violations).toHaveLength(0);
  });
});
```

Transaction Fuzzing

```
const { TransactionFuzzer } = require('adityzer');

describe('Transaction Fuzzing', () => {
  test('should fuzz transaction parameters', async () => {
    const fuzzer = new TransactionFuzzer({
      network: 'ethereum',
      gasPrice: 'auto'
    });

    const result = await fuzzer.fuzzTransaction({
      to: '0x123...',
      value: 'random',
      gasLimit: 'random',
      data: 'random'
    });

    expect(result.successfulTransactions).toBeGreaterThan(0);
    expect(result.failedTransactions).toBeDefined();
    expect(result.gasUsageStats).toBeDefined();
  });
});
```

Bridge Security Testing

Cross-Chain Bridge Testing

```
const { BridgeSecurityTester } = require('auditzer');

describe('Bridge Security Tests', () => {
  test('should test message validation', async () => {
    const tester = new BridgeSecurityTester({
      protocol: 'layerzero',
      sourceChain: 'ethereum',
      targetChain: 'polygon'
    });

    const result = await tester.testMessageValidation({
      payload: '0x1234567890abcdef',
      tamperAttempts: 100
    });

    expect(result.validationPassed).toBe(true);
    expect(result.tamperedMessagesRejected).toBe(100);
  });

  test('should test replay attack protection', async () => {
    const tester = new BridgeSecurityTester();

    const result = await tester.testReplayProtection({
      message: {
        nonce: 1,
        payload: '0x123...',
        signature: '0x456...'
      },
      replayAttempts: 10
    });

    expect(result.replayAttacksPrevented).toBe(10);
  });
});
```

Performance and Load Testing

Load Testing

```
const { SecurityLoadTester } = require('auditzer');

describe('Security Load Tests', () => {
  test('should handle high scan volume', async () => {
    const tester = new SecurityLoadTester({
      concurrency: 50,
      duration: 60000 // 1 minute
    });

    const result = await tester.loadTestScanning({
      contractsPerSecond: 10,
      scanDepth: 'medium'
    });

    expect(result.successRate).toBeGreaterThan(0.95);
    expect(result.averageResponseTime).toBeLessThan(5000);
    expect(result.errorRate).toBeLessThan(0.05);
  });
});
```

Reporting and Documentation

Security Report Generation

```
const { SecurityReporter } = require('auditzer');

describe('Security Reporting', () => {
  test('should generate comprehensive security report', async () => {
    const reporter = new SecurityReporter({
      format: 'pdf',
      includeRemediation: true,
      includeCodeExamples: true
    });

    const report = await reporter.generateReport({
      target: '0x123...',
      scanResults: scanResults,
      aiAnalysis: aiResults
    });

    expect(report.summary).toBeDefined();
    expect(report.vulnerabilities).toHaveLength(scanResults.vulnerabilities.length);
    expect(report.recommendations).toBeDefined();
    expect(report.riskScore).toBeGreaterThan(0);
  });
});
```

Custom Report Templates

```
const customTemplate = {
  sections: [
    'executive_summary',
    'vulnerability_details',
    'remediation_steps',
    'code_examples',
    'risk_assessment'
  ],

  styling: {
    theme: 'professional',
    includeCharts: true,
    includeCodeHighlighting: true
  }
};

const report = await reporter.generateReport(scanResults, customTemplate);
```

Best Practices

Security Testing Checklist

- [ ] Test all public and external functions
- [ ] Verify access control mechanisms
- [ ] Test for reentrancy vulnerabilities
- [ ] Check integer overflow/underflow protection
- [ ] Validate input sanitization
- [ ] Test oracle manipulation resistance
- [ ] Verify upgrade mechanisms security
- [ ] Test emergency pause functionality
- [ ] Check for front-running vulnerabilities
- [ ] Validate gas optimization

Continuous Security Testing

```
// CI/CD Integration
const { ContinuousSecurityTester } = require('audityzer');

const tester = new ContinuousSecurityTester({
  triggers: ['commit', 'pull_request', 'deployment'],
  notifications: ['slack', 'email'],
  failOnCritical: true
});

// GitHub Actions integration
// .github/workflows/security.yml
```

Security Monitoring

```
const { SecurityMonitor } = require('adityzer');

const monitor = new SecurityMonitor({
  contracts: ['0x123...', '0x456...'],
  alerting: {
    webhook: 'https://your-webhook.com',
    email: 'security@yourcompany.com'
  },
  checks: [
    'new_vulnerabilities',
    'suspicious_transactions',
    'unusual_patterns'
  ]
});

monitor.start();
```

Integration Examples

Web3 Framework Integration

```
// Hardhat integration
const { task } = require('hardhat/config');

task('security-scan', 'Run security scan on contracts')
  .setAction(async (taskArgs, hre) => {
    const { SecurityScanner } = require('adityzer');

    const scanner = new SecurityScanner({
      network: hre.network.name
    });

    const contracts = await hre.artifacts.getAllFullyQualifiedNames();

    for (const contract of contracts) {
      const result = await scanner.scan(contract);
      console.log(`Security scan results for ${contract}:`, result);
    }
  });
```

Foundry Integration

```
# forge test with security scanning
forge test --match-contract SecurityTest -vvv
```

Troubleshooting

Common Issues

- **High false positive rate**: Adjust confidence thresholds
- **Slow scanning**: Optimize scan depth and parallelization

- **Memory issues**: Increase Node.js memory limit
- **Network timeouts**: Configure appropriate timeouts

Performance Optimization

```
const optimizedConfig = {
  scanning: {
    parallel: true,
    maxConcurrency: 10,
    timeout: 30000,
    caching: true
  },

  ai: {
    batchSize: 50,
    modelOptimization: true,
    gpuAcceleration: true
  }
};
```

For more information, visit our documentation site (https://docs.auditzyer.com) or join our Discord community (https://discord.gg/auditzyer).