Blockchain Security Audit Framework

Essential Practices for Smart Contract Security

Published by Agochar Tech LLP
hello@agochar.com | agochar.com

Understanding Smart Contract Risks

Smart contracts are immutable code handling significant value, making security paramount. Once deployed, vulnerabilities cannot be easily patched.

Common Vulnerability Categories:

Reentrancy Attacks

External calls allowing attackers to re-enter functions before state updates complete. The DAO hack ($60M) exploited this vulnerability.

Integer Overflow/Underflow

Arithmetic operations exceeding variable bounds, causing unexpected behavior. Use SafeMath or Solidity 0.8+ built-in checks.

Access Control Issues

Missing or incorrect permission checks allowing unauthorized function calls. Critical for admin functions and upgradability.

Front-Running

Miners or bots observing pending transactions and inserting their own transactions ahead. Common in DEX and auction contracts.

Oracle Manipulation

Exploiting price feeds or external data sources to manipulate contract behavior. Flash loan attacks often leverage this.

Security Audit Process

A comprehensive audit follows structured phases:

1. Specification Review

Understand intended functionality, architecture, and threat model. Review documentation, requirements, and previous audits.

2. Automated Analysis

Run static analysis tools to identify common patterns:

- Slither: Solidity static analyzer
- Mythril: Security analysis for EVM bytecode
- Echidna: Property-based fuzzing

3. Manual Code Review

Line-by-line review by experienced auditors focusing on:

- Business logic flaws
- Access control correctness
- State management issues
- Gas optimization opportunities

4. Testing & Verification

Execute test suites and verify edge cases:

- Unit tests with high coverage
- Integration tests for contract interactions
- Formal verification for critical functions

5. Report & Remediation

Document findings with severity ratings, provide recommendations, and verify fixes.

Security Best Practices

Defensive coding patterns for smart contracts:

Checks-Effects-Interactions

Order operations: validate inputs, update state, then make external calls. Prevents reentrancy attacks.

Access Control Patterns

Use OpenZeppelin AccessControl or Ownable. Implement role-based permissions for sensitive functions.

Upgrade Patterns

If upgradability is required, use transparent or UUPS proxy patterns. Carefully manage upgrade authorization.

Circuit Breakers

Implement pause functionality for emergency situations. Enable quick response to discovered vulnerabilities.

Rate Limiting

Limit transaction frequency and amounts to reduce attack impact. Implement withdrawal delays for large amounts.

Monitoring & Alerts

Set up real-time monitoring for unusual activity:

- Large value transfers
- Unusual function call patterns
- Contract balance changes
- Failed transaction spikes