OpenZeppelin Security Audit

Ganjes DAO Smart Contracts

Project	Ganjes DAO Smart Contracts
Audit Date	August 20, 2025
Tools Used	Slither Static Analysis, OpenZeppelin Defender SDK
Total Findings	80 issues identified
Risk Level	HIGH (Critical reentrancy vulnerabilities)
Status	Requires immediate attention before deployment

Ganjes DAO Smart Contracts

# Executive Summary **Project**: Ganjes DAO Smart Contracts **Audit Date**: August 20, 2025 **Tools Used**: Slither Static Analysis, OpenZeppelin Defender SDK **Contracts Audited**: - GanjesDAOOptimized.sol - GanjesDAOSimplified.sol - ProposalManagement.sol - SimpleToken.sol

# Audit Overview This audit identified **80 findings** across the smart contract system, ranging from critical security vulnerabilities to code quality improvements. The analysis covers reentrancy attacks, access control issues, and best practice violations.

Critical Findings (High Risk)

# ■ RE-1: Reentrancy Vulnerabilities in Proposal Creation **Severity**: Critical **Contract**: GanjesDAOSimplified.sol:95-138, ProposalManagement.sol:135-228 **Description**: State variables are written after external calls in `createProposal()` functions, creating reentrancy attack vectors. **Impact**: Attackers could manipulate proposal creation limits and bypass cooldown periods. **Affected Code**: ```solidity // External call followed by state changes !governanceToken.transferFrom(msg.sender,address(this),PROPOSAL_DEPOSIT_AMOUNT) lastProposalTime[msg.sender] = block.timestamp; // Vulnerable proposalCountByUser[msg.sender] ++; //

Vulnerable ``` **Recommendation**: Implement Checks-Effects-Interactions pattern or use OpenZeppelin's ReentrancyGuard.

# ■ RE-2: Reentrancy in Voting Functions **Severity**: Critical **Contract**: GanjesDAOSimplified.sol:140-171 **Description**: Multiple state variables updated after external token transfers in `vote()` function. **Impact**: Vote manipulation and potential double-spending attacks. **Recommendation**: Apply reentrancy protection and reorder operations.

High Risk Findings

# ■ AC-1: Missing Access Control on Critical Functions **Severity**: High **Contract**: Multiple contracts **Description**: Several administrative functions lack proper access control mechanisms. **Recommendation**: Implement OpenZeppelin's AccessControl or Ownable patterns.

# ■ TX-1: Transaction Order Dependence **Severity**: High **Contract**: GanjesDAOSimplified.sol, ProposalManagement.sol **Description**: Functions vulnerable to MEV attacks and front-running. **Recommendation**: Implement commit-reveal schemes or use timestamp-based ordering.

Medium Risk Findings

# ■ EQ-1: Dangerous Strict Equality Check **Severity**: Medium **Contract**: ProposalManagement.sol:418 **Description**: Using `==` for timestamp comparison can be unreliable. **Affected Code**: ```solidity cooldownPassed = timeUntilNextProposal == 0 ``` **Recommendation**: Use `<=` or range checks instead of strict equality.

# ■ US-1: Unused State Variables **Severity**: Medium **Description**: Multiple state variables declared but never used, increasing gas costs. **Recommendation**: Remove unused variables or mark as private if needed for inheritance.

# ■ UF-1: Unused Functions **Severity**: Medium **Description**: 15+ functions defined but never called, bloating contract size. **Recommendation**: Remove dead code or document if intended for future use.

Low Risk & Informational Findings

# ■ NC-1: Naming Convention Violations **Severity**: Low **Description**: 25+ parameters not following mixedCase convention. **Examples**: - `_projectName` → `projectName` - `_fundingGoal` → `fundingGoal` - `_proposalId` → `proposalId`

# ■ GS-1: Gas Optimization Opportunities **Severity**: Informational **Description**: Variables that should be declared as constant or immutable. **Affected Variables**: - `SimpleToken.decimals` → should be constant - `SimpleToken.name` → should be constant - `SimpleToken.symbol` → should be constant - `GanjesDAO.admin` → should be immutable - `votingDuration` → should be immutable

# ■ LD-1: Literals with Too Many Digits **Severity**: Informational **Description**: Large number literals reduce readability. **Examples**: ```solidity MAX_FUNDING_GOAL = 1000000 * 10 ** 18; // Use 1e6 * 1e18 or constants ```

Compilation Issues

# ■■ COMP-1: Stack Too Deep Error **Severity**: High **Contract**: GanjesDAOOptimized.sol **Description**: Contract fails to compile due to stack depth limitations. **Error**: `Stack too deep. Try compiling with --via-ir` **Recommendation**: 1. Enable IR-based code generation in Hardhat config 2. Reduce local variable usage in functions 3. Split complex functions into smaller ones

OpenZeppelin Integration Recommendations

# 1. Security Modules ```solidity import "@openzeppelin/contracts/security/ReentrancyGuard.sol"; import "@openzeppelin/contracts/security/Pausable.sol"; import "@openzeppelin/contracts/access/Ownable.sol"; ```

# 2. Safe Math & Token Standards ```solidity import "@openzeppelin/contracts/token/ERC20/IERC20.sol"; import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol"; ```

# 3. Governance Framework Consider migrating to OpenZeppelin's Governor contracts for standardized DAO functionality.

Risk Summary | Severity | Count | Examples | |----------|--------|----------| | Critical | 2 | Reentrancy vulnerabilities | | High | 8 | Access control, MEV risks | | Medium | 15 | Logic issues, gas inefficiencies | | Low | 30+ | Naming conventions, optimizations | | Informational | 25+ | Code quality improvements |

Prioritized Remediation Plan

# Phase 1 (Immediate - Critical/High Risk) 1. ■ Implement reentrancy guards on all external calls 2. ■ Add proper access control to admin functions 3. ■ Fix compilation

issues in GanjesDAOOptimized.sol 4. ■ Review and secure token transfer operations

# Phase 2 (Short-term - Medium Risk) 1. ■ Replace strict equality checks with range checks 2. ■ Remove unused state variables and functions 3. ■ Implement proper error handling 4. ■ Add input validation on all external functions

# Phase 3 (Long-term - Low Risk/Optimization) 1. ■ Fix naming convention violations 2. ■ Declare appropriate variables as constant/immutable 3. ■ Optimize gas usage patterns 4. ■ Improve code documentation

Tools & Methodologies - **Slither Static Analysis**: Automated vulnerability detection - **OpenZeppelin Defender SDK**: Security monitoring and alerts - **Manual Code Review**: Logic and business rule validation - **Compilation Testing**: Solidity compiler optimization flags

Conclusion The Ganjes DAO contracts show a solid foundation but require immediate attention to critical security vulnerabilities, particularly reentrancy attacks. The extensive use of OpenZeppelin's battle-tested security primitives is recommended to mitigate identified risks. **Overall Risk Rating**: ■■ **HIGH** (due to reentrancy vulnerabilities) **Recommended Action**: Address critical findings before deployment --- **Audit conducted using OpenZeppelin security standards and

methodologies** **Next Review**: After implementing Phase 1 fixes

Generated on August 20, 2025 at 08:51 PM
Audit conducted using OpenZeppelin security standards