Smart Contract Audit Report

Voting

Generated on 8/9/2025

Contract Hash: 8733eabff340d15e...

Compiler Version: 0.8.0

Executive Summary

This audit report presents the findings from a comprehensive security analysis of the smart contract. A total of 9 potential issues were identified, including 4 critical or high-severity vulnerabilities that require immediate attention. Additionally, 1 gas optimization opportunities were identified that could reduce transaction costs. All findings include detailed descriptions, code locations, and remediation recommendations to help improve the contract's security posture.

9	1
TOTAL ISSUES	CRITICAL
3	**4**
HIGH	MEDIUM
1	**0**
LOW	INFORMATIONAL

Vulnerability Overview

Severity	Count	Percentage
CRITICAL	1	11.1%
HIGH	3	33.3%
MEDIUM	4	44.4%
LOW	1	11.1%
INFORMATIONAL	0	0.0%

Vulnerability Details

Critical Best Practice Violation

CRITICAL

Contract has syntax errors: missing semicolons, incorrect struct/mapping/array declarations, and malformed constructor/function signatures. The code will not compile.

Location: `Voting.sol:3:1`
Source: `ai`
Confidence: `100%`

Recommendation:

Follow Solidity best practices and coding standards to improve code quality and security.

High Best Practice Violation

HIGH

Candidates should be stored as an array (Candidate[]) but is declared as a single public variable. This prevents multiple candidates from being stored and breaks the voting logic.

Location: `Voting.sol:8:5`
Source: ai
Confidence: 95%

Recommendation:

Follow Solidity best practices and coding standards to improve code quality and security.

High Best Practice Violation

HIGH

Missing visibility specifier for 'owner' variable. Should be 'address public owner' or 'address private owner'.

Location: `Voting.sol:9:5`
Source: ai
Confidence: 95%

Recommendation:

Follow Solidity best practices and coding standards to improve code quality and security.

High Best Practice Violation

HIGH

Mapping syntax is incorrect. Should be 'mapping(address => bool) public voters' instead of 'mappingaddress > bool public voters'.

Location: `Voting.sol:10:5`
Source: ai
Confidence: 95%

Recommendation:

Follow Solidity best practices and coding standards to improve code quality and security.

Best Practice Violation

MEDIUM

Contract lacks proper documentation with NatSpec comments for functions and variables.

Location: `Voting.sol:1:1`
Source: ai
Confidence: 90%

Recommendation:

Follow Solidity best practices and coding standards to improve code quality and security.

Security Issue

MEDIUM

The vote() function increments vote count but doesn't check if voting period is active. Users can vote outside the designated voting period.

Location: `Voting.sol:32:5`
Source: `ai`
Confidence: `90%`

Recommendation:

Review and address the identified security concern.

Best Practice Violation

MEDIUM

No event emissions for critical actions like voting or adding candidates. Events are important for transparency and off-chain monitoring.

Location: `Voting.sol:32:5`
Source: ai
Confidence: 85%

Recommendation:

Follow Solidity best practices and coding standards to improve code quality and security.

Access Control Issue

MEDIUM

No mechanism to transfer ownership or renounce ownership. Consider using OpenZeppelin's Ownable pattern.

Location: `Voting.sol:9:5`
Source: ai
Confidence: 85%

Recommendation:

Implement proper access control mechanisms using modifiers and role-based permissions.

Gas Optimization Opportunity

LOW

getAllVotesOfCandiates() function has a typo in the name and returns entire array which could be gas expensive for large number of candidates.

Location: Voting.sol:42:5
Source: ai
Confidence: 80%

Recommendation:

Optimize gas usage by reviewing the identified code patterns and implementing suggested improvements.

Gas Optimizations

The following optimizations could reduce gas consumption:

~150 gas

gas_optimization

getAllVotesOfCandiates() function has a typo in the name and returns entire array which could be gas expensive for large number of candidates.

Location: `Voting.sol:42:5`

Total Estimated Gas Savings: ~150 gas per transaction

Security Recommendations

Security [HIGH]

Description: Fix all syntax errors and ensure the contract compiles successfully
Implementation: 1. Add missing semicolons after struct declarations 2. Fix struct syntax with proper braces 3. Correct mapping syntax from 'mappingaddress > bool' to 'mapping(address => bool)' 4. Fix function parameter syntax 5. Change 'candidates' from single instance to array: 'Candidate[] public candidates'

Security [HIGH]

Description: Add voting period validation in the vote() function
Implementation: 1. Add require statement: require(getVotingStatus(), 'Voting is not active') 2. This ensures votes can only be cast during the designated voting period

Best Practices [MEDIUM]

Description: Implement events for transparency
Implementation: 1. Add event: event VoteCast(address indexed voter, uint256 indexed candidateIndex) 2. Add event: event CandidateAdded(string name, uint256 index) 3. Emit these events in respective functions

Security [MEDIUM]

Description: Use OpenZeppelin's Ownable contract for access control
Implementation: 1. Import '@openzeppelin/contracts/access/Ownable.sol' 2. Inherit from Ownable: contract Voting is Ownable 3. Replace custom onlyOwner modifier with OpenZeppelin's implementation

Gas Optimization [LOW]

Description: Optimize getAllVotesOfCandiates function

Implementation: 1. Fix typo in function name to 'getAllVotesOfCandidates' 2. Consider adding pagination or individual candidate retrieval functions for better gas efficiency

Access Control [HIGH]

Description: Implement proper role-based access control
Implementation: Use OpenZeppelin's AccessControl or Ownable contracts for permission management

Appendix

Contract Information

Contract Name	Voting
File Hash	8733eabff340d15ece9e0f3472b63221931274e2c6bc51fa367db889cd0c6608
Compiler Version	0.8.0
Created	8/9/2025

Code Metrics

Lines of Code	1
Function Count	0
Cyclomatic Complexity	1

Methodology

This audit was conducted using a combination of static analysis tools and AI-powered security analysis:

- **Static Analysis:** Automated scanning using Slither and custom AST analysis
- **AI Analysis:** Machine learning models trained on security vulnerabilities
- **Manual Review:** Expert validation of findings and recommendations

This report was generated by Audit Wolf - Smart Contract Security Platform