RISK AUDIT

for

AITV.GG

on

Aug 07, 2025

FIDESIUM

Executive Summary

Report

7
/100
TOTAL
Low risk
Aug 08, 2025

15
/100
TOTAL
Low risk
Aug 07, 2025

Abstract

Fidesium's automated risk assessment service was requested to perform a risk posture audit on TriviTournament **contracts**

Repository Link:
https://github.com/agentcoinorg/dao-contracts

Initial Commit Hash:

`28aa42b4222f5734127585c26783f7ad1865aff8`

Included Contracts:

- AITVAirdropVesting.sol
- DeployVotingEscrow.s.sol

Issue Summary

Critical	High	Medium	Low	Info
0 Issues	1 Issues	~~5~~ 1 Issues	~~4~~ Issues	~~2~~ 3 Issues

Caveats

AITV's codebase is generally well written, but does incur a handful of flaws.

Test Approach

Fidesium performed both Whitebox and Blackbox testing, as per the scope of the engagement, and relied on automated security testing.

Methodology

The assessment methodology covered a range of phases and employed various tools, including but not limited to the following:

- Mapping Content and Functionality of API
- Application Logic Flaws
- Access Handling
- Authentication/Authorization Flaws
- Brute Force Attempt
- Input Handling
- Source Code Review
- Fuzzing of all input parameter
- Dependency Analysis

Severity Definitions

Critical	The issue can cause large economic losses, large-scale data disorder or loss of control of authority management.
High	The issue puts users' sensitive information at risk or is likely to lead to catastrophic financial implications.
Medium	The issue puts a subset of users' sensitive information at risk, reputation damage or moderate financial impact.
Low	The risk is relatively small and could not be exploited on a recurring basis, or is low-impact to the client's business.
Informational	The issue does not pose an immediate risk but is relevant to security best practices or defence in Depth.

Risk Issues

Vulnerability	Description	Risk	Probability	Status
Deployment Script: Implicitly trusted binary	The `DeployVotingEscrow.s.sol` script trusts the bytecode without verification	High	Unlikely	Resolved
Reliance on Block Timestamp	`AITVAirdropVesting.sol` contract relies on `block.timestamp`, which can be manipulated by miners.	Medium	Low	Acknowledged
Missing Pausability	The `AITVAirdropVesting.sol` contract does not implement pausability. This could limit the ability of the developer to respond in an emergency.	~~Medium~~Info	Medium	Remediated
One Step Ownership Transfer	The `AITVAirdropVesting.sol` contract apply the `Ownable` pattern. It relies on a one step `transferOwnership` strategy. This exposes these contracts to accidental ownership transfer to malicious or invalid wallets	Medium	Low	Resolved
Missing Contract Validation	The `AITVAirdropVesting` contract presumes `token` and `votingEscrow` to be set correctly	Medium	Medium	Resolved
Missing Deployment Contract Variable Validation	The `DeployVotingEscrow.s.sol` script presumes `AITV_TOKEN_ADDRESS` is a valid contract	Medium	Low	Resolved
Missing Deployment Success Validation	The `DeployVotingEscrow.s.sol` script presumes deploy success	Low	Medium	Resolved
Gas Inefficiency: Unnecessary computation	The `AITVAirdropVesting.claimAndForfeitRemaining` function computes over constants	Info	Info	Remediated
Gas Inefficiency: uint256	The `AITVAirdropVesting` contract uses `uint256`	Info	Info	Active

Risk Overview

Team Risk

Low risk: 1

No issues found in founding team

Doxxing Status	Team Experience	Risk Summary
Public	Highly relevant	Low

Smart Contract Risks

Risk summary: ~~21~~13

The contracts are well written, and secure with only a few minor issues..

Vulnerabilities Critical

Current scan criticals Clear

During this scan no critical security vulnerabilities were identified. The assessment covered all key components of the project, including smart contract logic, access controls, and potential attack vectors. While no critical issues were found, we recommend ongoing security monitoring and best practices to maintain the integrity and resilience of the system.

Vulnerabilities High

Deployment Script: Implicitly trusted binary

Vulnerability severity: **High**

Vulnerability probability: **Unlikely**

The `DeployVotingEscrow.s.sol` script trusts the bytecode without verification

This represents a supply chain risk. An attacker can inject malicious bytecode, through compromised build tools/compilers, CI/CD tooling, developer machines, or tampered repositories.

`bytes memory bytecode = vm.parseJsonBytes(json, bytecodePath);`

Recommendations:

Validate bytecode, and select functions:

```
bytes32 expectedBytecodeHash = 0x[KNOWN_GOOD_HASH];
bytes32 actualHash = keccak256(bytecode);
require(actualHash == expectedBytecodeHash, "Bytecode hash mismatch - possible tampering");
string memory methodPath = ".['lib/curve-dao-contracts/contracts/VotingEscrow.vy'].method_identifiers";
string memory methodsJson = vm.parseJsonString(json, methodPath);

string memory constructorPath = ".['__init__(address,string,string,string)']";
string memory actualSelector = vm.parseJsonString(methodsJson, constructorPath);

bytes4 expectedConstructor = bytes4(keccak256("__init__(address,string,string,string)"));
string memory expectedSelectorStr = vm.toString(expectedConstructor);

require(
    keccak256(abi.encodePacked(actualSelector)) == keccak256(abi.encodePacked(expectedSelectorStr)),
    "Constructor selector mismatch - artifact may be compromised"
);
```

Action Taken:

Resolved at commit `41d44bd885e70dac1dfb99a159e1b4da4958e665`

Vulnerabilities Medium

Reliance on Block Timestamp

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

`AITVAirdropVesting` contract relies on `block.timestamp`, which can be manipulated by miners.

- `claimAndForfeitRemaining`
- `claimAndDepositToLock`
- `rescueTokens`

Recommendations:

- Use block numbers in addition to timestamps.
- If timestamps are necessary, use trusted external oracles.

Missing Deployment Contract Variable Validation

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

The `DeployVotingEscrow.s.sol` script presumes `AITV_TOKEN_ADDRESS` is a valid contract

Due to error or malicious oversight, this could be set to an invalid or malicious value

Recommendations:

- Validate `AITV_TOKEN_ADDRESS` contains a contract
- Validate `AITV_TOKEN_ADDRESS` has the expected abi

Missing Pausability

Vulnerability severity: ~~Medium~~**Info**

Vulnerability probability: **Medium**

The `AITVAirdropVesting` contract does not implement pausability. This could limit the ability of the developer to respond in an emergency.

Recommendations:

Use `Pausable` from OpenZeppelin

Action Taken:

Remediated through the existence of `rescueTokens` allowing the Owner to withdraw all assets from the contract, pausing the contract in practice

This remains a low impact issue as manual token recovery requires slightly more thought (e.g addresses, token amounts), and could be a slower response loop than outirght pausing

Vulnerabilities Medium

One Step Ownership Transfer

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

The `AITVAirdropVesting` contract apply the `Ownable` pattern. It relies on a one step `transferOwnership` strategy. This exposes these contracts to accidental ownership transfer to malicious or invalid wallets.

Recommendations:

Implement `Ownable2Step` to drive a two step ownership transfer. This will require applying `Upgradeable` independently.

Missing Contract Validation

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

The `AITVAirdropVesting.sol` contract are presumes `token` and `votingEscrow` to be set correctly

These could be misconfigured, either through malice or accidental misconfiguration to point to incorrect contracts, ranging in severity from rendering the protocol non functional, to actively and maliciously draining customers

Recommendations:

Validate contract abis and codesize in constructor

```
uint256 codeSize;
assembly {
    tokenCodeSize := extcodesize(token)
}
require(tokenCodeSize > 0, "token is not a contract");
assembly {
    votingEscrowCodeSize := extcodesize(votingEscrow)
}
require(votingEscrowCodeSize > 0, "VotingEscrow is not a contract");
try IERC20(_token).totalSupply() returns (uint256) {
} catch {
    revert InvalidTokenAddress();
}
try IVotingEscrow(_votingEscrow).locked(address(this)) returns (int128, uint256) {
} catch {
    revert NoVotingEscrowConfigured();
}
```

Vulnerabilities Low

Missing Deployment Success Validation

Vulnerability severity: **Low**

Vulnerability probability: **Medium**

The `DeployVotingEscrow.s.sol` script presumes deploy success

```
assembly { deployedAddress := create(0, add(fullBytecode, 0x20), mload(fullBytecode)) } return deployedAddress;
```

Recommendations:

Validate deployment succeeded:

```
require(deployedAddress != address(0), "Deployment failed")
```

Vulnerabilities Info

Gas Inefficiency: Unnecessary computation

Vulnerability severity: **Info**

Vulnerability probability: **Info**

The `AITVAirdropVesting.claimAndForfeitRemaining` function computes over constants

```
uint256 unlockedPercent =
IMMEDIATE_UNLOCK_BASIS_POINTS + ((elapsed * (MAX_BASIS_POINTS - IMMEDIATE_UNLOCK_BASIS_POINTS)) / VESTING_DURATIO
```

Recommendations:

Precompute (or even hardcode) known constant math

```
uint256 public constant PRECOMPUTED_RESULT = (MAX_BASIS_POINTS - IMMEDIATE_UNLOCK_BASIS_POINTS)) / VESTING_DURATI
uint8 public constant PRECOMPUTED_RESULT = 10;
```

Action Taken:

No explicit action was taken, however this is remediated by deploying on Base where gas is cheap.

Gas Inefficiency: uint256

Vulnerability severity: **Info**

Vulnerability probability: **Info**

The `AITVAirdropVesting` contract uses `uint256`

This is gas inefficient on every operation

Recommendations:

Validate your business logic and ensure the smallest possible `uint` is used

Disclaimer

Disclaimer

This report is governed by the Fidesium terms and conditions.

This report does not constitute an endorsement or disapproval of any project or team, nor does it reflect the economic value or potential of any related product or asset. It is not investment advice and should not be used as the basis for investment decisions. Instead, this report provides an assessment intended to improve code quality and mitigate risks inherent in cryptographic tokens and blockchain technology.

Fidesium does not guarantee the absence of bugs or vulnerabilities in the technology assessed, nor does it comment on the business practices, models, or regulatory compliance of its creators. All services, reports, and materials are provided "as is" and "as available," without warranties of any kind, including but not limited to merchantability, fitness for a particular purpose, or non-infringement.

Cryptographic assets and blockchain technologies are novel and carry inherent technical risks, uncertainties, and the possibility of unpredictable outcomes. Assessment results may contain inaccuracies or depend on third-party systems, and reliance on them is solely at the Customer’s risk.

Fidesium assumes no liability for content inaccuracies, personal injuries, property damages, or losses related to the use of its services, reports, or materials. Third-party components are provided "as is," and any warranties are strictly between the Customer and the third-party provider.

These services and materials are intended solely for the Customer's use and benefit. No third party or their representatives may claim rights to or rely on these services, reports, or materials under any circumstances.