Cyberscope

Audit Report

BlaroThings

January 2025

SHA256	5403b7b236e450f843055bc9bfbbb2d944377d66a70aa00ba5f0cd6f4ec611a6
Audited by	© cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Audit Updates

Initial Audit	13 Jan 2025
Corrected Phase 2	23 Jan 2025

Source Files

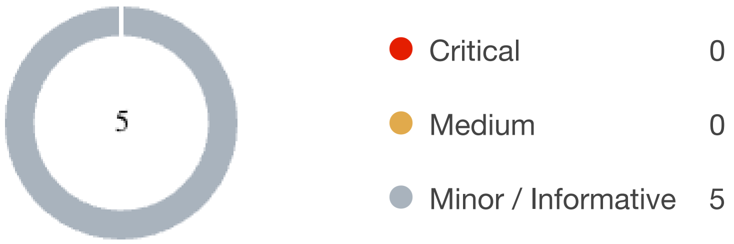
Filename	SHA256
DAO.sol	5403b7b236e450f843055bc9bfbbb2d944377d66a70aa00ba5f0cd6f4ec611a6

Overview

The DAOGovernance contract is a decentralized governance system designed to manage proposals and voting within a DAO (Decentralized Autonomous Organization). The contract allows the owner to create proposals that include details such as a description, an address for potential fund transfers, an Ether transfer amount, and an optional state-change action. Proposals can then be voted on by participants with assigned voting power. Each voter can contribute their voting weight, and proposals are executed if they receive a minimum number of votes (`minVotesRequired`), ensuring a degree of consensus before implementation.

Key features include the ability to transfer Ether to a specified recipient if a proposal dictates such an action. The contract ensures security through mechanisms like OpenZeppelin's ReentrancyGuard to prevent reentrancy attacks and Ownable to restrict sensitive operations to the owner. Voting power is dynamically assignable by the owner, and each participant's voting history is tracked to prevent duplicate votes on the same proposal. Additionally, the contract supports Ether deposits and uses SafeMath for safe arithmetic operations, enhancing its reliability.

Findings Breakdown



Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	5	0	0	0

Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
●	CCR	Contract Centralization Risk	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	L09	Dead Code Elimination	Unresolved

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	DAO.sol#L479,508,515,543
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

The contract allows the owner to assign voting power arbitrarily. This power could be exploited by assigning the owner high voting power, enabling them to unilaterally create a proposal, vote on it, and execute it with minimal or no participation from other voters. This behavior undermines the decentralized governance principles of a DAO, as decisions can be manipulated by the owner without requiring consensus from the broader community.

```
function createProposal(
    string memory description,
    address recipient,
    uint256 amount,
    bool changeState
) external onlyOwner {
    require(bytes(description).length > 0, "Proposal description cannot be empty");
    require(votingPower[msg.sender] >= minVotesRequired, "Insufficient voting power to create proposal");
    ...
}
...
proposal.voteCount = proposal.voteCount.add(votingPower[msg.sender]);
...
function executeProposal(uint256 proposalId) external nonReentrant onlyOwner {
    require(proposalId <= proposalCount, "Invalid proposalId");
    Proposal storage proposal = proposals[proposalId];
    require(!proposal.executed, "Proposal already executed");
    require(proposal.voteCount >= minVotesRequired, "Not enough votes to execute");
    ...
}
```

Additionally, the `assignVotingPower` function grants the owner unrestricted authority to modify the voting power of all users. This centralization of control introduces several risks, such as:

1. The owner has unilateral control over the governance mechanism, which undermines decentralization.
2. The owner could assign disproportionate voting power to specific addresses, enabling manipulation of proposals and decisions.
3. Users must trust that the owner will act in the best interest of the DAO, reducing trustless guarantees central to blockchain systems.
4. The dependency on the owner’s decisions conflicts with the principles of distributed and community-driven governance.

```
function assignVotingPower(address voter, uint256 power) external
onlyOwner {
    require(voter != address(0), "Invalid address");
    require(power > 0, "Voting power must be positive");

    totalVotingPower =
totalVotingPower.sub(votingPower[voter]).add(power); // Update total
voting power
    votingPower[voter] = power;
}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	DAO.sol#L475
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
minVotesRequired
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	DAO.sol#L548
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
votingPower[voter] = power;
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	DAO.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily in cases where the explanatory error message is not used.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/stable/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	DAO.sol#L29,211
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function _contextSuffixLength() internal view virtual returns (uint256) {
        return 0;
    }

function _reentrancyGuardEntered() internal view returns (bool) {
        return _status == _ENTERED;
    }
```

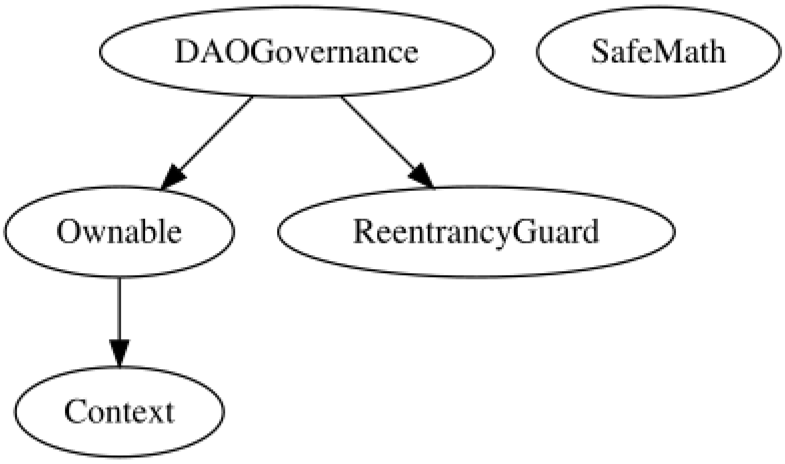
Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

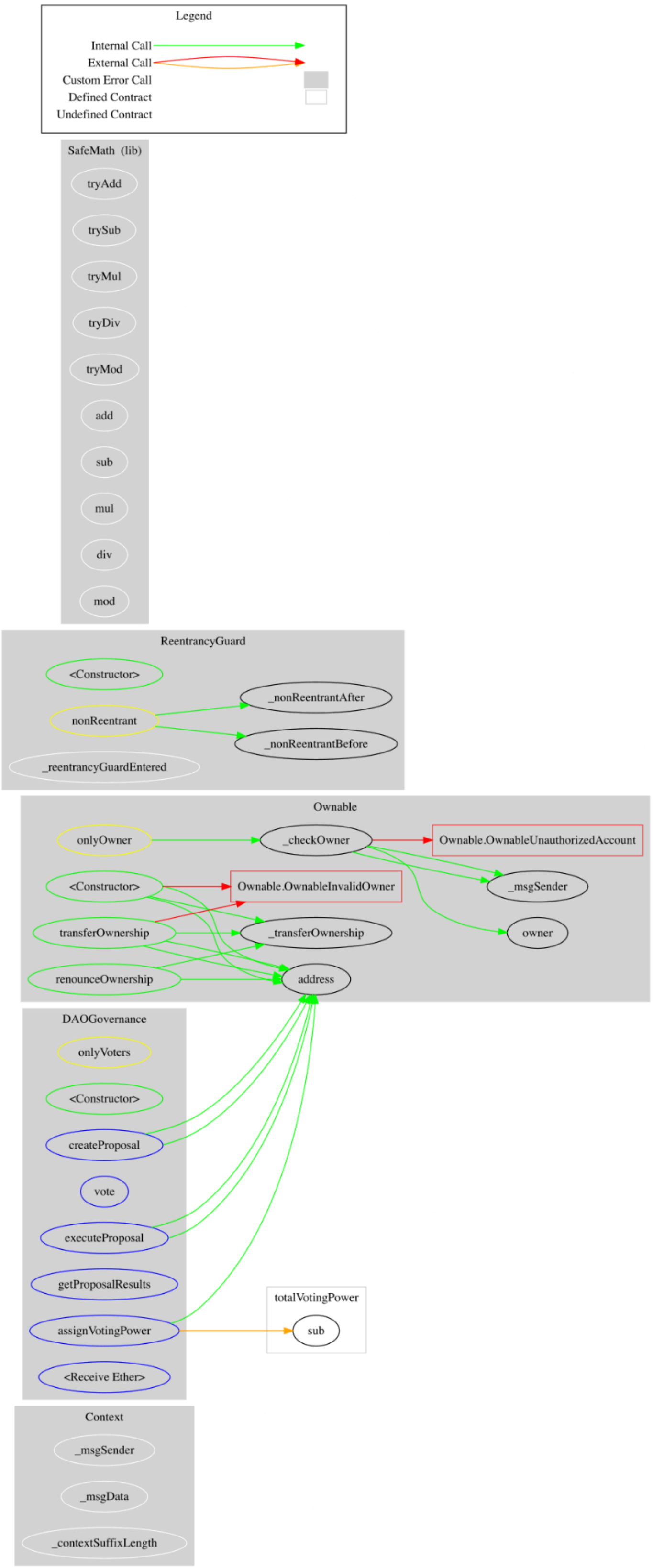
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
DAOGovernance	Implementation	Ownable, ReentrancyGuard		
		Public	✓	Ownable
	createProposal	External	✓	onlyOwner
	vote	External	✓	onlyVoters
	executeProposal	External	✓	nonReentrant onlyOwner
	getProposalResults	External		-
	assignVotingPower	External	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod
ReentrancyGuard
<Constructor>
nonReentrant
_nonReentrantAfter
_nonReentrantBefore
_reentrancyGuardEntered
Ownable
onlyOwner
_checkOwner
Ownable.OwnableUnauthorizedAccount
<Constructor>
Ownable.OwnableInvalidOwner
_msgSender
transferOwnership
_transferOwnership
owner
renounceOwnership
address
DAOGovernance
onlyVoters
<Constructor>
createProposal
vote
executeProposal
getProposalResults
assignVotingPower
<Receive Ether>
totalVotingPower
sub
Context
_msgSender
_msgData
_contextSuffixLength

Summary

BlaroThings contract implements a governance mechanism. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

cyberscope.io