

Smart Contract Security Audit Report for Gala

AnChain.AI Inc.

Auditors: AnChain.AI Audit Team

Contact: Audit@AnChain.ai

Date: 02-17-2023

Index

1. Introduction
    1. Audit Info
    2. Methodology
    3. Environment
2. Executive Summary
    1. Key Findings
3. Smart Contract Scope
    1. Audited Source Code File
4. Vulnerability Audit
5. Business Logic Audit
6. Gas Consumption Analysis
7. Conclusion
8. Appendix
    1. Vulnerability Technical Explanation
    2. Severity Level
    3. Reference
9. Disclaimer

1. Introduction

1.1 Audit Info

This document includes the results of the security audit for the client's smart contract code GalaV2 as found in the section 3. The security audit was conducted by the AnChain.AI team from 02-14-2023 to 02-17-2023.

The purpose of this audit is to review source code, functionality on test net, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

1.2 Methodology

AnChain.AI team adopts a suite of tools and best practice from cybersecurity to audit smart contract source code for vulnerabilities.

3 key aspects in the AnChain.AI security audit methodology:

- Vulnerability audit. We use various tools to scan for known vulnerabilities, including our proprietary CAS sandbox.
- Statistical audit. Our sandbox will predict the risk score for the audited code with hundreds of thousands of mainnet smart contracts to understand security stance.
- Business logic audit. Our experts will design specific test cases to cover the key custom functions, to identify potential risk exposures.

1.3 Environment

Compiled Version: v0.8.9.commit.e5eed63a

Compiled System: MAC

Compiled Tool: Hardhat

2. Executive Summary

2.1 Key Findings

We **did not identify high severity vulnerabilities** that would compromise the integrity of the project. This upgradeable ERC20 smart contract **meets** the AnChain.AI smart contract auditing standards. However we did identify some low severity issues (see section 5 for details).

3. Smart Contract Scope

3.1 Audited Source Code File

File Name	Md5
GalaV2.sol	`d211b2a9b23ba1d4b276e205c65ecc1e`

4. Vulnerability Audit

4.1 Vulnerability Checklist

Type	Status
Numerical Overflow	PASS
Authorization Check	PASS
Transaction Ordering Dependency	PASS
Parity Multisig Bug	PASS
Random Number Practice	PASS
Re-entrancy	PASS
Function Visibility	PASS
Event Emitting	PASS
ERC Token Formatting	PASS

5. Business Logic Audit

This section is dedicated to the functionality test of the audited smart contract. The purpose is to cover the main business logic for completeness and correctness.

The smart contract has been deployed on Testnet, with the specified environment and parameters.

Compile Test

tool	Hardhat environment
version	solc Compiler v0.8.9.commit.e5eed63a
result	PASS

Code modification for compiling

file name	n/a
before	n/a
after	n/a

There is no code modification for this audit.

Deployment Test:

Local Testnet	PASS

5.1 grantRole

function grantRole (bytes32 role, address _account)

Description

This function assigns a role to an address

Specification

Caller should be admin	PASS
Contract should not be paused	PASS
Caller should not be blocklisted	PASS
Role assignee should not be address zero	PASS

Test Cases

Admin should be able to assign a new role	PASS
Should fail if caller is not admin	PASS
Should fail if contract is paused	PASS
Should fail if assignee is zero address	PASS
Should fail if assignee is blocklisted	PASS

Findings

Inconsistent events **Severity-** Low severity

There is an inconsistency and redundancy in emitted events depending on which function is used to assign a new role. This inconsistency can impact the monitoring of contract events and also introduces extra gas cost for emitting redundant simialr events.

grantRole function emits *RoleGranted* event if an admin, minter, upgrader or blocklister role is assigned whereas function *addMinter, addUpgrader,* and *addBlocklister* are emitting *RoleGranted* along with redundant custom events-

```
    event MinterAdded
    event UpgraderAdded
    event BlocklisterAdded
```

e.g.-
addMinter function emits *MinterAdded* and *RoleGranted* events
grantRole function emits *RoleGranted* if called directly

Recommendations

We recommend removing custom role assignment events *MinterAdded, UpgraderAdded, and BlocklisterAdded* etc since they are redundant to *RoleGranted* event and will save gas costs as well.

5.2 revokeRole

function revokeRole (bytes32 role, address _account)

Description

This function revokes role from an address

Specification

Caller should be admin	PASS
Contract should not be paused	PASS
Role assignee should not be address zero	PASS

Test Cases

Admin should be able to revoke a role	PASS
Should fail if caller is not admin	PASS
Should fail if contract is paused	PASS
Should fail if assignee is zero address	PASS

Findings

Inconsistent events **Severity-** Low severity

There is an inconsistency and redundancy in emitted events depending on which function is used to revoke a role. This inconsistency can impact the monitoring of contract events and also introduces extra gas cost for emitting redundant simialr events.

revokeRole function emits *RoleRevoked* event if an an admin, minter, upgrader or blocklister role is removed whereas function *removeMinter, removeUpgrader,* and *removeBlocklister* are emitting *RoleRevoked* along with custom events-

```
event MinterRemoved
event UpgraderRemoved
event BlocklisterRemoved
```

e.g.-
removeMinter function emits *MinterRemoved* and *RoleRevoked* events
revokeRole function only emits *RoleRevoked* if called directly

Recommendations

We recommend removing custom events *MinterRemoved, UpgraderRemoved, and BlocklisterRemoved* since they are redundant to *RoleRevoked* event and will save gas costs as well.

5.3 addMinter

function addMinter (address account)

Description

This function adds a new minter

Specification

Caller should be admin	PASS
Contract should not be paused	PASS
Caller should not be blocklisted	PASS
Role assignee should not be address zero	PASS

Test Cases

Admin should be able to add minter	PASS
Should fail if caller is not admin	PASS
Should fail if contract is paused	PASS
New Minter should be able to mint tokens	PASS

5.4 removeMinter

function removeMinter (address account)

Description

This function removes a minter

Specification

Caller should be admin	PASS
Contract should not be paused	PASS
Caller should not be blocklisted	PASS
Role assignee should not be address zero	PASS

Test Cases

Admin should be able to remove minter	PASS
Should fail if caller is not admin	PASS
Should fail if contract is paused	PASS
Removed minter should not be able to mint tokens	PASS

5.5 addBlocklister

function addBlocklister (address account)

Description

This function add a blocklister

Specification

Caller should be admin	PASS
Contract should not be paused	PASS
Caller should not be blocklisted	PASS
Role assignee should not be address zero	PASS

Test Cases

admin should be able to add blocklister	PASS
Should fail if caller is not admin	PASS
Should fail if contract is paused	PASS
New blocklister should be able to blocklist address	PASS

5.6 removeBlocklister

function removeBlocklister(address account)

Description

This function removes a blocklister

Specification

Caller should be admin	PASS
Contract should not be paused	PASS
Caller should not be blocklisted	PASS
Role assignee should not be address zero	PASS

Test Cases

Admin should be able to remove blocklister	PASS
Should fail if caller is not admin	PASS
Should fail if contract is paused	PASS
Removed blocklister should not be able to blocklist an address	PASS

5.7 addUpgrader

function addMinter (address account)

Description

This function adds a new upgrader

Specification

Caller should be admin	PASS
Contract should not be paused	PASS
Caller should not be blocklisted	PASS
Role assignee should not be address zero	PASS

Test Cases

admin should be able to add upgrader	PASS
Should fail if caller is not admin	PASS
Should fail if contract is paused	PASS
New upgrader should be able to upgrade implementation	PASS

5.8 removeUpgrader

function removeMinter (address account)

Description

This function removes an upgrader

Specification

Caller should be admin	PASS
Contract should not be paused	PASS
Caller should not be blocklisted	PASS
Role assignee should not be address zero	PASS

Test Cases

Admin should be able to remove upgrader	PASS
Should fail if caller is not admin	PASS
Should fail if contract is paused	PASS
Removed upgrader should not be able to upgrade implementation	PASS

5.9 transferAdmin

function addMinter (address account)

Description

This function transfer admin role to new address

Specification

Caller should be admin	PASS
Contract should not be paused	PASS
Caller should not be blocklisted	PASS
Role assignee should not be address zero	PASS

Test Cases

admin should be able to transfer admin role	PASS
Should fail if caller is not admin	PASS
Should fail if contract is paused	PASS

5.10 mintBulk

function mintBulk (address[] accounts, uint256[] amounts)

Description

This function allows to mint tokens to multiple addresses

Specification

Caller should have minter role	PASS
Contract should not be paused	PASS
Receiver should not be blocklisted	PASS

Test Cases

Minter admin should be able mint tokens	PASS
Should fail if caller is not Minter admin	PASS
Should fail if new amount is exceeding total Cap	PASS
Should fail if any receiving address is block listed	PASS

5.11 initialize

function initialize (string name, string symbol)

Description

Initialization of proxy contract

Specification

Should only be called once	PASS

Test Cases

Should not be able to re-initialize the contract	PASS

5.12 pause

function pause ()

Description

This function changes the contract to pause state

Specification

Caller should be admin	PASS

Test Cases

Admin should be able to pause contract	PASS
Should fail if caller is not admin	PASS

5.13 unpause

function unpause ()

Description

This function changes the contract to pause state

Specification

Caller should be admin	PASS

Test Cases

Admin should be able to unpause contract	PASS
Should fail if caller is not admin	PASS

5.14 upgradeTo

function upgradeTo (address newImplementation)

Description

This function upgrades the contract to use new implementation contract

Specification

Caller should be upgrader admin	PASS
Implementation Address should be a contract	PASS

Test Cases

Should upgrade the contract to new implementation	PASS
Upgrade should fail if caller is not admin	PASS

5.15 approve

function approve (address spender, uint256 amount)

Description

This function approves an allowance for a spender

Specification

Caller should be blocklisted	PASS
Contract should not be paused	PASS
Spender should not be block listed	PASS

Test Cases

Wallet should be able to approve allowance	PASS
Should fail if caller is block listed	PASS
Should fail if spender is block listed	PASS

5.16 increaseAllowance

function increaseAllowance (address spender, uint256 amount)

Description

This function increases allowance for a spender

Specification

Caller should be admin	PASS
Contract should not be paused	PASS
Spender should not be block listed	PASS

Test Cases

Wallet should be able to increase allowance	PASS
Should fail if caller is block listed	PASS
Should fail if spender is block listed	PASS

5.17 decreaseAllowance

function decreaseAllowance (address spender, uint256 amount)

Description

This function decreases allowance for a spender

Specification

Contract should not be paused	PASS
Caller should not be block listed	PASS
Spender should not be blocklisted	PASS

Test Cases

Wallet should be able decrease allowance	PASS
Should fail if caller is block listed	PASS
Should fail if spender is block listed	PASS

5.18 mint

function mint (address account, uint256 amount)

Description

This function allows to mint tokens

Specification

Caller should have minter role	PASS
Contract should not be paused	PASS
Receiver should not be blocklisted	PASS

Test Cases

Minter admin should be able mint tokens	PASS
Should fail if caller is not Minter admin	PASS
Should fail if new amount is exceeding total Cap	PASS
Should fail if receiving address is block listed	PASS

5.19 blocklist

function blocklist (address account)

Description

This function allows to blocklist an address

Specification

Caller should be blocklist admin	PASS

Test Cases

blocklist admin should be able to blocklist a wallet	PASS
Should fail if caller does not have BLOCKLISTER_ROLE	PASS

5.20 unBlocklist

function unBlocklist (address account)

Description

This function allows to un-blocklist an address

Specification

Caller should be blocklist admin	PASS

Test Cases

Blocklist admin should be able to unblock list a wallet	PASS
Should fail if caller does not have BLOCKLISTER_ROLE	PASS

5.21 burn

function burn (uint256 amount)

Description

This function allows to burn token

Specification

Contract should not be paused	PASS
Caller should not be block listed	PASS
Holder should not be block listed	PASS

Test Cases

Wallet should be able to burn tokens	PASS
Should fail if blocklisted	PASS

5.22 burnFrom

function burnFrom (address account, uint256 amount)

Description

This function allows to spender to burn token on behalf of an address

Specification

Contract should not be paused	PASS
Caller should not be block listed	PASS
Holder should not be block listed	PASS

Test Cases

Spender should be able to burn tokens for approved amount	PASS
Should fail if amount exceeds allowance	PASS
Should fail if caller is blocklisted	PASS
Should fail if token holder is blocklisted	PASS

5.23 _beforeTokenTransfer

function _beforeTokenTransfer *(address to, uint256 amount)*

Description

This is a helper function for transfer of tokens

Specification

Contract should not be paused	PASS
Caller should not be block listed	PASS
Holder should not be block listed	PASS

Test Cases

Should be able to transfer tokens	PASS
Should fail if sender is blocklisted	PASS
Should fail if caller is blocklisted	PASS
Should fail if receiver is blocklisted	PASS

6. Gas Consumption Analysis

We tested gas consumptions of main functions, and all results are within normal gas consumption range.

Function Name	Gas Consumption
approve	~58,000
decreaseAllowance	~50,000
increaseAllowance	~58,000
transfer	~58,000
transferFrom	~63,000
blocklist	~8,000
burn	~40,000
burnFrom	~48,000
unpause	~30,000
pause	~51,000
unBlocklist	~36,000
removeMinter	~35,000

7. Conclusion

The audited smart contract source code extends the functionality of an upgradeable ERC20 token using OpenZeppelin's open source libraries[1] **We have confirmed this contract is secure.**

8. Appendix

8.1 Vulnerability Technical Explanation

These vulnerability scannings are powered by the patented AnChain.AI CAS (Smart Contract Auditing Sandbox), including static analysis, dynamic analysis, and statistical analysis.

- **Numerical Overflow**

Severity Level: high

In ETH, failing to run a numerical check on arithmetic operations may cause the values to overflow or underflow, which may cause crypto asset loss. It is a good practice to use "SafeMath" library for all the numeric operations to take care of overflow and underflow issue of all the numeric operations.

The code provided pass the 'numerical overflow' vulnerability check.

- **Authorization**

Severity Level: high

Parameters passed into a function should be consistent with the actual caller.

The best practice is to use 'require()' to check if the user executing the action matches the provided parameter.

The code provided pass the 'Authorization' vulnerability check.

- **Transaction Ordering Dependency:**

Severity Level: high

In ETH, timestamp of a transaction getting approved can be manipulated by vicious minors. Transaction Ordering Dependency refer to critical logic fault if minors approve the later submitted transaction PRIOR than the earlier ones.

Transaction ordering dependency does not have critical impact on this contract.

- **Parity Multisig Bug/ Delegate Call:**

Severity Level: High

The code does not call any external contracts and thus there is no issue about parity multisig bug.

- **Random Number**

Severity Level: High

Random number generator algorithm should not use predictable seeds.

The code does not generate or use any random number, and thus there is no issue about the random number issue.

The code provided pass the "Random Number" vulnerability check.

- **Re-entrancy Attack:**

Severity Level: High

The code does not contain any function that is vulnerable to re-entrancy attack.

- **Function Visibility:**

Severity Level: Low

The code does not contain any function that is improperly set with visibility.

- **Event Emitting:**

Severity Level: Low

The ERC code has emit transfer & approval event properly, which meets a ERC token standard.

- **ERC Token Standard:**

Severity Level: Low

The token contract has the following variables: name, symbol, decimal, totalSupply, which meets the ERC token standard.

8.2 Severity Level

Level	Description
High	The issue poses an existential risk to the project, and the issue identified could lead to massive financial or reputational repercussions.
Medium	The potential risk is large, but there is some ambiguity surrounding whether or not the issue would practically manifest.
Low	The risk is small, unlikely, or not relevant to the project in a meaningful way.
Code Quality	The issue identified does not pose any obvious risk, but fixing it would improve overall code quality, conform to recommended best practices, and perhaps lead to fewer development issues in the future.

8.3 Reference

[1]. OpenZeppelin upgradeable contracts. Retrieved from https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable

9. Disclaimer

Disclaimer - Smart Contract Auditing - ETH

AnChain.ai makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and AnChain.ai specifically disclaims all implied warranties or merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

AnChain.ai will not be liable for any lost profits, business, contracts, revenues, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand against the company by any other party. If no event will AnChain.ai be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if AnChain.ai has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the client's team and only the source code AnChain.ai notes as being within the scope of AnChain.ai's review within this report. This report does not include an audit of the deployment scripts used to deploy the contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than AnChain.ai. Such hyperlinks are provided for your reference and convenience only and are the exclusive responsibility of such websites' owners. You agree that AnChain.ai is not responsible for the content or operation of such websites and that AnChain.ai shall have no liability to your or any other person or entity for the use of third party websites. AnChain.ai assumes no responsibility for the use of third-party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.