HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: AugmentLabs
Date: February 28, 2023

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for AugmentLabs
Approved By	Marcin Ugarenko | Lead Solidity SC Auditor at Hacken OU
Type	ERC20 token; Staking
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://augmentlabs.io/
Changelog	20.01.2023 - Initial Review 16.02.2023 - Second Review 28.02.2023 - Third Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by AugmentLabs (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is review and security analysis of smart contracts in the repository:

Initial review scope

Repository	https://github.com/augmentlabs-io/contracts
Commit	87bbcbe571601c52cc6e9823558fc2f7115d0666
Whitepaper	https://docs.google.com/document/d/1pavVjJp_lrMJAarWTSNr44wCFA30qq07EW20OjzuUEo/
Functional Requirements	Business & Technical requirements for the USC stable coin.pdf
Technical Requirements	Business & Technical requirements for the USC stable coin.pdf
Contracts	File: ./contracts/AGC.sol SHA3: 65a7139b129bba7e7f2bddd4590e1c32c83d0495c12321cdb541338441f058d1 File: ./contracts/Controller.sol SHA3: 93c239987006d0709bf8394944340b10e479b6ed27cb79d5ba3c742fc09b7dc4 File: ./contracts/MasterChef.sol SHA3: 6d791a2fb147460b4834433c7b82cbcba8b64aebcc3a790af74d7cab2ac261b4 File: ./contracts/USC.sol SHA3: 14d8ffbd3049cbc3861138d038aac96bd591ac39b7471b3a9e9ca84e64de2623

Second review scope

Repository	https://github.com/augmentlabs-io/contracts
Commit	da96194625daae26f54b0b5ca057314ad8ad4038
Whitepaper	https://docs.google.com/document/d/1pavVjJp_lrMJAarWTSNr44wCFA30qq07EW20OjzuUEo/
Functional Requirements	https://docs.augmentlabs.io/smart-contracts/ Business & Technical requirements for the USC stable coin.pdf
Technical Requirements	https://docs.augmentlabs.io/smart-contracts/ Business & Technical requirements for the USC stable coin.pdf
Contracts	File: ./contracts/AGC.sol

	SHA3: df6d0135cd047b8bcbe46fae15840127ae2b33f1a97c5c612d2087f85878fbdb File: ./contracts/Controller.sol SHA3: 6faf980461e5fb57c67d32f2719369005593b01b94fa960802231b1a20a1dd9a File: ./contracts/MasterChef.sol SHA3: 2da1911975ee9ae463545104bb100fa2ac53f55b692bd38c806f0befac8d8734 File: ./contracts/USC.sol SHA3: 7d6a8ac0a59db90fc7956929433fa98498fb28254e2cf8edf899f418e681d9af

Third review scope

Repository	https://github.com/augmentlabs-io/contracts
Commit	03cbb1160c9d6681db6883adfb007245b2600799
Whitepaper	https://docs.google.com/document/d/1pavVjJp_lrMJAarWTSNr44wCFA30qq07EW20OjzuUEo/
Functional Requirements	https://docs.augmentlabs.io/smart-contracts/ Business & Technical requirements for the USC stable coin.pdf
Technical Requirements	https://docs.augmentlabs.io/smart-contracts/ Business & Technical requirements for the USC stable coin.pdf
Contracts	File: ./contracts/AGC.sol SHA3: d8f91e7ace14027d888c58e9d6912806f6861ec2a4e67b2595a91cc35ac4dcfe File: ./contracts/Controller.sol SHA3: 10178e61671f4b19e3cb70be7f362322df0ef7da44f63bb8fc8312ea0d269140 File: ./contracts/MasterChef.sol SHA3: b865503cb8b1844af196147882935412e07f21329ede94d45f6c9b0b469b7581 File: ./contracts/USC.sol SHA3: 7d6a8ac0a59db90fc7956929433fa98498fb28254e2cf8edf899f418e681d9af

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor gas optimization. These issues won't have a significant impact on code execution but affect code quality

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- A White Paper is provided.
- NatSpec is included in the code.
- Technical description and functional requirements are provided.

Code quality

The total Code Quality score is **9** out of **10**.

- The development environment is configured.
- 5 out of 103 tests are failing.

Test coverage

Code coverage of the project is **84.94%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is present.

Security score

As a result of the audit, the code contains no issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.3**.

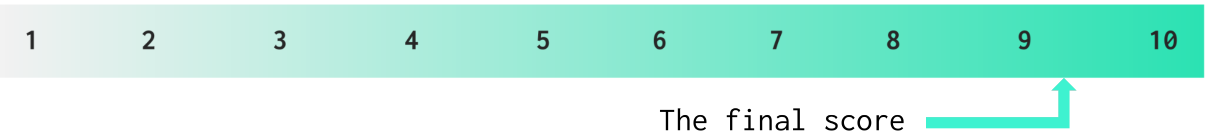

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
20 January 2023	13	7	5	1
16 February 2023	4	0	0	1
28 February 2023	0	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Passed
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Not Relevant
Presence of Unused Variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP Standards Violation	EIP	EIP standards should not be violated.	Passed
Assets Integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Not Relevant
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant

Token Supply Manipulation	Custom	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Passed
Gas Limit and Loops	Custom	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Not Relevant
Style Guide Violation	Custom	Style guides and best practices should be followed.	Passed
Requirements Compliance	Custom	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	Custom	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	Custom	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	Custom	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	Custom	The code should not reference draft contracts, which may be changed in the future.	Passed

System Overview

AugmentLabs is a mixed-purpose ERC20/Stacking system with the following contracts:

- *AGC* — upgradable ERC-20 token that mints all initial supply to a deployer. Minting and burning is allowed by the Minter on demand.
  It has the following attributes:
  - Name: AGC
  - Symbol: AGC
  - Differences from the ERC20 standard: the token is not transferable.
- *USC* — upgradable ERC-20 token that mints all initial supply to a deployer. Minting is allowed by the Minter on demand.
  It has the following attributes:
  - Name: USC
  - Symbol: USC
  - Other parameters are default for the ERC20 standard
- *TokenController* — an upgradable smart contract that will have MINTER_ROLE of AGC & USC to perform the redeem functionality between AGC/USC.
  - Has a safety switch.
  - Role-based authentication.
  - Can burn USC and mint AGC to a specific user address.
  - Can burn AGC and mint USC to a specific user address.
- *MasterChef* — an upgradable smart contract that allows users to stake USDT and get USC rewards at a specified ROI/year.
  - Has a safety switch.
  - Can mint USC to yield rewards to pool stakers.

Privileged roles

- The MasterChef contact has a single owner role.
- The TokenController contract has PAUSER_ROLE, UPGRADER_ROLE and REDEEMER_ROLE roles and must take the MINTER_ROLE of USC and AGC.
- The MINTER_ROLE of AGC and USC can mint tokens on demand without any restrictions. The burn functions in USC are unprotected.

Risks

- No substantial risks were identified.

Recommendations

- The system relies on the security of the privileged roles' private keys, which can impact the execution flow and security of the funds. We recommend those accounts to be at least ⅗ multi-sig.

Findings

🟥🟥🟥🟥 Critical

C01. Token Supply Manipulation

There is no check/update of the user's `lastRewardBlock`, and there is no mechanism to passively store user-earned yield when the user performs the deposit() function call.

An attacker can deposit small amounts into the pool from multiple addresses and wait to accumulate the time delta. Afterwards, they can use, for example, a FlashLoan to multiply the attack and make a large second deposit (for example, $1 million).

As the "lastRewardBlock" variable was not updated and rewards were not passively stored for the user, rewards will be calculated from the current large deposit and the accumulated time. The attacker can then withdraw the deposit and rewards (20% APR from $1 million) without having to actively stake such an amount of funds.

Path:
./contracts/MasterChef.sol : deposit()

Recommendation: Consider updating the reward system with mechanisms that prevent such manipulations.

Reference: https://solidity-by-example.org/defi/staking-rewards/

Status: Fixed
(revised commit: 03cbb1160c9d6681db6883adfb007245b2600799)

🟧🟧🟧 High

H01. Requirements Violation

The business requirements documents state that “only AFG multisig address or whitelisted addresses” are allowed to burn USC tokens, however the `burn()` function is inherited from ERC20BurnableUpgradeable.sol by default and is unprotected. Also users can burn their own token at will in both USC and AGC tokens.

This may lead to data inconsistency since the sum of user balances will stop being equal to the company’s balance.

Path:
./contracts/AGC.sol

Recommendation: The `burn()` function should be overridden and always revert. `burnFrom()` should be protected by access control.

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

H02. Requirements Violation

The business requirements documents state that “Users can’t transfer AGC to any other users”, but the `transfer()` and `transferFrom()`

functions are inherited from ERC20BurnableUpgradeable.sol by default. This may lead to data inconsistency since the sum of user balances will stop being equal to the company's balance.

Path:
./contracts/AGC.sol

Recommendation: Replace content of the `_beforeTokenTransfer()` function with `revert()`.

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

H03. Requirements Violation

An incorrect event is emitted at the end of the `redeemUSC()` function.

Path:
./contracts/Controller.sol : redeemUSC()

Recommendation: replace emitting the `AGCRedeemed` event with `USCRedeemed` in the function.

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

H04. Undocumented Behavior

The burning of USC tokens from the company's address instead of the user's when calling the `redeemUSC()` is undocumented and may be an error.

Path:
./contracts/Controller.sol : redeemUSC()

Recommendation: Check if the behavior of the function is correct and fix it or add the supporting documentation.

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

H05. Data Inconsistency

The crucial invariant that the balance of the company should be equal to the sum of user balances can be broken if the company address calls the `mint()` function. As the company is not implied to be one of the users, then the equation balanceOf(companyAddress) == _userBalance may stop being true.

Path:
./contracts/AGC.sol : mint()

Recommendation: Check that `require(userAddress != companyAddress, "Company cannot update its own userBalance")`.

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

Medium

M01. Dangerous Strict Zero Equality

The value `rewardAmount` is derived based on complex calculations which can lead to non-strict zero values due to binary number representation inside the EVM. This can potentially lead to a value which should be logically equal to zero actually being higher than zero.

Path:
./contracts/MasterChef.sol : tryPayUSC()

Recommendation: Implement a check against “epsilon” (maximum allowed fault) instead of a strict zero check.

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

M02. Owner Privilege Actions

The owner of the `MasterChef` contract can change `ROIPerYear` on demand. This can adversely affect the users’ trust, as the owner can lower their expected gain at any time or block withdrawing altogether by setting setROIPerYear to 0.

Path:
./contracts/MasterChef.sol : setROIPerYear()

Recommendation: Implement a `rewardPerTokenStored` system instead. Reference: https://solidity-by-example.org/defi/staking-rewards/

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

M03. Undocumented Behavior

The owner of the `MasterChef` contract can change `pool.multiplier` on demand. This can lead to a trust issue with users.

Path:
./contracts/MasterChef.sol : set()

Recommendation: Remove the possibility of modifying the `pool.multiplier` after pool creation or mention this possibility in the “stacking” paragraph of the documentation.

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

M04. Undocumented Functionality

The owner of the `MasterChef` contract can change `pool.rewardLockupBlock` on demand. This can lead to an infinite lock in the stacking contract.

Path:
./contracts/MasterChef.sol : set()

Recommendation: Remove the `_lockupBlock` parameter in the function and line `poolInfo[_pid].rewardLockupBlock = _lockupBlock;`.

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

M05. Missing Functionality

There is an option to change the `companyAddress` in the `Controller` contract, but there is no option to change the `companyAddress` in `AGC`.

Path:
./contracts/AGC.sol

Recommendation: If this is intended then the documentation should be updated. If not then proper migration functionality should be implemented which will involve moving the tokens to the new address.

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

M06. Using Block Number Attribute as a Proxy of Time

The contract takes a constructor parameter `blocksPerYear` and calculates rewards based on this value. Blocks per year is not a constant value and can lead to a serious fault if used as a time reference.

Path:
./contracts/MasterChef.sol

Recommendation: Drop block number usage for business logic as a time reference in favor of `block.timestamp`. Assume that a year is 365 days instead (and mention that in the document to avoid ambiguity).

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

M07. Best Practice Violation - Unchecked Transfer

It is considered following best practices to avoid unchecked transfer functions, which can potentially lead to DoS vulnerabilities.

Path:
./contracts/MasterChef.sol : withdraw(), safeUSCTransfer()

Recommendation: Follow common best practices: use the OpenZeppelin's `SafeERC20.sol` library and replace `transfer()` calls with `safeTransfer()` and `transferFrom()` with `safeTransferFrom()`.

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

■ Low

L01. Redundant Modifiers

The `virtual` modifier is superfluous in the functions of the top level contracts.

Paths:
./contracts/AGC.sol
./contracts/Controller.sol
./contracts/MasterChef.sol
./contracts/USC.sol

Recommendation: Remove the `virtual` modifier from functions of top level contracts.

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

L02. Floating Pragma

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Paths:
./contracts/AGC.sol
./contracts/Controller.sol
./contracts/MasterChef.sol
./contracts/USC.sol

Recommendation: Use strict pragma settings.

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

L03. Style Guide Violation - Implicit State Visibility

State variables visibility should be explicit.

Paths:
./contracts/AGC.sol : companyAddress, _userBalance
./contracts/Controller.sol : AGCToken, USCToken

Recommendation: Use explicit visibility modifiers.

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

L04. Style Guide Violation - Visibility Modifiers

Visibility modifiers should be first in the list of function modifiers.

Paths:
./contracts/AGC.sol : initialize()
./contracts/Controller.sol : initialize()
./contracts/MasterChef.sol : initialize()
./contracts/USC.sol: initialize()

Recommendation: It is best practice to put visibility modifiers first in the list of function modifiers.

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

L05. Style Guide Violation - Incorrect Function Order

Public functions should not be declared after public view functions.

Paths:
./contracts/AGC.sol
./contracts/Controller.sol
./contracts/MasterChef.sol

Recommendation: Rearrange functions to comply with official Solidity style guidelines.

Reference:
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#order-of-functions

Status: Fixed
(revised commit: 03cbb1160c9d6681db6883adfb007245b2600799)

L06. Style Guide Violation - Unused Function Parameters

The code contains an overridden function with unused, yet named parameters.

Paths:
./contracts/AGC.sol : _authorizeUpgrade()
./contracts/Controller.sol : _authorizeUpgrade()
./contracts/MasterChef.sol : _authorizeUpgrade()

Recommendation: Remove the names of the parameters to show explicitly that the parameters are not going to be used and are only intended to override an inherited function. For reference:

https://docs.soliditylang.org/en/v0.8.17/contracts.html#function-parameters

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

L07. Unfinished NatSpec

NatSpec is not complete - some Smart Contract members are undocumented.

Paths:
./contracts/AGC.sol
./contracts/Controller.sol
./contracts/MasterChef.sol
./contracts/USC.sol

Recommendation: Add NatSpec to undocumented members of the Smart Contracts.

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

L08. Missing Zero Address Validation

Address parameters are used without checking against the possibility of 0x0. This can lead to unwanted external calls to 0x0.

Paths:
./contracts/AGC.sol : initialize();
./contracts/Controller.sol : initialize(), setCompanyAddress()
./contracts/MasterChef.sol : initialize()

Recommendation: Implement zero address checks.

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

L09. Unindexed Events

Having indexed parameters in the events makes it easier to search for these events using indexed parameters as filters.

Path:
./contracts/Controller.sol : CompanyAddressUpdated

Recommendation: Use the “indexed” keyword for the event parameters.

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

L10. Unnecessary Code Complication

The functions contain checks against a boolean literal. Functions which return a boolean variable are recommended to be checked directly on their return value for the sake of code simplicity.

Path:
./contracts/MasterChef.sol : getPoolIdForLpToken(),
getPoolIdForLpToken()

Recommendation: Replace `poolExistence[_lpToken] != false` to `poolExistence[_lpToken]` and `poolExistence[_lpToken] == false` to `!poolExistence[_lpToken]`.

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

L11. Missing Zero Check - Uint

The functions can receive a zero `_amount` parameter which will still trigger further code execution and Gas usage without any practical effects. It is advisable to immediately halt the execution in case of invalid or non-effective parameters.

Path:
./contracts/MasterChef.sol : withdraw()

Recommendation: Implement a zero check at the beginning of function execution e.g. `require(_amount > 0, "amount can not be equal to zero")`.

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

L12. Functions That Can Be Declared External

In order to save Gas, public functions that are never called in the contract should be declared as external.

Paths:
./contracts/AGC.sol : mint(), pause(), unpause()
./contracts/Controller.sol : initialize(), pause(), unpause(), redeemAGC(), redeemUSC()
contracts/MasterChef.sol : initialize(), pause(), unpause(), setROIPerYear(), add(), set(), withdraw(), deactivatePool(), activatePool(), deposit(), getReward()
./contracts/USC.sol : initialize(), pause(), unpause(), mint()

Recommendation: Use the external attribute for functions never called from the contract.

Status: Fixed
(revised commit: 03cbb1160c9d6681db6883adfb007245b2600799)

L13. Redundant Code

The function `safeUSCTransfer()` is redundant. There is no apparent option for a rounding error. Tokens can be transferred directly from `address(this)` to `msg.sender` by minting first and then transferring the amount.

Path:
./contracts/MasterChef.sol : safeUSCTransfer()

Recommendation: Remove the redundant function and instead implement direct transfer inside the `withdraw()` function.

Status: Fixed
(revised commit: 921bd8ff6bfa2d09e0a8063f7583e4b5e19804a4)

L14. Redundant Use of SafeMath

Since Solidity v0.8.0, the overflow/underflow check is implemented via ABIEncoderV2 on the language level - it adds the validation to the bytecode during compilation.
There is no need to use the SafeMath library.

Path:
./contracts/MasterChef.sol

Recommendation: Remove the SafeMath library.

Status: Fixed
(revised commit: da96194625daae26f54b0b5ca057314ad8ad4038)

L15. Redundant Imports

The use of unnecessary imports will increase the Gas consumption of the code. Thus, they should be removed from the code.

Redundant imports decrease code readability.

Paths:
./contracts/MasterChef.sol : IERC20Upgradeable, USC

Recommendation: Consider removing redundant code.

Status: Fixed
(revised commit: 03cbb1160c9d6681db6883adfb007245b2600799)

L16. Style Guide Violation

State variables and local variables should never begin with a capital letter (except constants, which are written in all-capital letters). State variables such as `ROIPerYear`, `USCToken`, `USDTToken`, `AGCToken`, `USCToken` violate this convention. This can lead to confusion whether the developer is dealing with a variable or a type.

Paths:
./contracts/MasterChef.sol
./contracts/Controller.sol

Recommendation: Follow the official Solidity Style Guide: https://docs.soliditylang.org/en/v0.8.17/style-guide.html#local-and-state-variable-names. Consider renaming variables by pattern `USCToken` -> `uscToken` etc.

Status: Fixed
(revised commit: 03cbb1160c9d6681db6883adfb007245b2600799)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.