// Smart Contract Security Assessment 11.05.2024 - 11.05.2024

Keys Contracts & Token

AgentCoin TV

HALBORN

Keys Contracts & Token - AgentCoin TV

Prepared by: HALBORN

Last Updated 11/20/2024

Date of Engagement by: November 5th, 2024 - November 5th, 2024

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
4	0	0	0	3	1

TABLE OF CONTENTS

1. Introduction

AgentCoin TV engaged Halborn to conduct a security assessment on their smart contracts revisions on November 7th, 2024. The security assessment was scoped to the smart contracts provided to the Halborn team.

Commit hashes and further details can be found in the Scope section of this report.

2. Assessment Summary

The team at Halborn was provided 1 day for the engagement and assigned a full-time security engineer to evaluate the security of the smart contract.

The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were partially addressed by the AgentCoin TV team:

- Standardize all contracts to use the same Solidity version.
- Upgrade to at least Solidity version 0.8.19 which has fixed these vulnerabilities.
- Implement a time-delayed, multi-signature governance mechanism for emergency action.

3. Test Approach And Methodology

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance code coverage and quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions. (`solgraph`)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs. (`MythX`)
- Static Analysis of security for scoped contract, and imported functions. (`Slither`)
- Testnet deployment. (`Brownie`, `Anvil`, `Foundry`)

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: agent-keys-contracts

(b) Assessed Commit ID: 5fee2e2

(c) Items in scope:

- AgentKey.sol
- AgentKeyWhitelist.sol
- IAgentKey.sol

Out-of-Scope: External libraries and financial-related attacks.

FILES AND REPOSITORY

(a) Repository: agent-token

(b) Assessed Commit ID: 28ce8be

(c) Items in scope:

- AgentcoinToken.sol

Out-of-Scope: External libraries and financial-related attacks.

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	3

INFORMATIONAL
1

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
CENTRALIZATION RISK: SINGLE ACCOUNT CONTROLS CONTRACT STOP AND RESERVE WITHDRAWAL	LOW	SOLVED - 11/11/2024
USE OF DEPRECATED AND VULNERABLE SOLIDITY VERSION 0.5.17	LOW	RISK ACCEPTED - 11/10/2024
SOLIDITY VERSION MISMATCH BETWEEN CORE CONTRACT AND DEPENDENCIES	LOW	RISK ACCEPTED - 11/11/2024
INITIALIZATION PATTERN INCONSISTENCY IN PROXY CONTEXT	INFORMATIONAL	RISK ACCEPTED - 11/11/2024

7. FINDINGS & TECH DETAILS

7.1 CENTRALIZATION RISK: SINGLE ACCOUNT CONTROLS CONTRACT STOP AND RESERVE WITHDRAWAL

// LOW

Description

The `stopAndTransferReserve` function in **AgentKey.sol** allows the beneficiary address to unilaterally stop the contract and drain all reserves without any time delay, multi-signature requirement, or governance approval:

```
function stopAndTransferReserve(address payable _recipient) external {
    require(msg.sender == beneficiary, "BENEFICIARY_ONLY");
    isStopped = true;
    Address.sendValue(_recipient, address(this).balance);
}
```

The beneficiary has the power to:
1. Stop all contract operations by setting `isStopped = true`
2. Immediately withdraw 100% of the contract's ETH balance
3. Execute both actions in a single atomic transaction
4. Send funds to any arbitrary address

This centralized control creates a systemic risk to the protocol as a single compromised key or malicious action leads to complete fund loss and permanent contract shutdown

BVSS

AO:S/AC:L/AX:L/C:N/I:C/A:N/D:N/Y:N/R:N/S:C (2.5)

Recommendation

Implement a time-delayed, multi-signature governance mechanism for emergency action.

Remediation

SOLVED: The owner of the contract will be the DAO's address, which mitigates the risk of a single EOA having this level of control.

7.2 USE OF DEPRECATED AND VULNERABLE SOLIDITY VERSION 0.5.17

// LOW

Description

The **AgentKey.sol** contract is compiled using Solidity version `0.5.17`:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.5.17;
```

This version contains multiple known vulnerabilities including:
1. `AbiReencodingHeadOverflowWithStaticArrayCleanup` (SOL-2022-6): When ABI-encoding tuples with statically-sized calldata arrays, it corrupts 32 leading bytes of dynamically encoded components.
2. `NestedCalldataArrayAbiReencodingSizeValidation` (SOL-2022-2): ABI-reencoding of nested dynamic calldata arrays does not perform proper size checks against calldata size and reads beyond calldatasize().
3. `ABIDecodeTwoDimensionalArrayMemory` (SOL-2021-2): The ABI decoder does not properly validate pointers for dynamically-sized data when decoding from memory, leading to incorrect results.
4. `DynamicArrayCleanup` (SOL-2020-10): Storage slots are not properly zeroed out when assigning dynamically-sized arrays with types <= 16 bytes.

Reference

The vulnerabilities in Solidity 0.5.17 affect core language features like ABI encoding/decoding and array handling. This exposes the contract to memory corruption, invalid data reading, and improper storage cleanup - compromising the reliability and security of core contract functionality.

BVSS

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (2.5)

Recommendation

Upgrade to at least Solidity version `0.8.19` which has fixed these vulnerabilities and includes additional safety features like:

- Built-in overflow checks
- More restrictive type casting
- Improved error handling
- Explicit mutability requirements

Remediation

RISK ACCEPTED: Due to the fact that not major risk from 0.5.17 are present in the code, `AgentCoin` team decided to not upgrade to a newer solidity version.

7.3 SOLIDITY VERSION MISMATCH BETWEEN CORE CONTRACT AND DEPENDENCIES

// LOW

Description

The **AgentKey** contract and its dependencies use different Solidity compiler versions which breaks version consistency across the protocol. The core contract uses Solidity `0.5.17` while its whitelist interface uses `0.8.13`:

```
solidity
Copy
// AgentKey.sol
pragma solidity 0.5.17;

import {DecentralizedAutonomousTrust} from "@fairmint/contracts/Decent
import "@openzeppelin/contracts-ethereum-package/contracts/utils/Addre

// AgentKeyWhitelist.sol
pragma solidity ^0.8.13;

contract AgentKeyWhitelist {
    function authorizeTransfer(address _from, address _to, uint256, bo
        require(_from == address(0) || _to == address(0), "TRANSFERS_D
    }
}
```

Impact :

1. Inconsistencies in type handling between compiler versions
2. Breaking changes in internal compiler behaviors between `0.5.x` and `0.8.x`
3. Different security checks and optimizations applied across contract components

BVSS

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (2.5)

Recommendation

Standardize all contracts to use the same Solidity version.

Remediation

RISK ACCEPTED: Due to the fact that not major risk from `0.5.17` are present in the code, `AgentCoin` team decided to not upgrade to a newer solidity version.

7.4 INITIALIZATION PATTERN INCONSISTENCY IN PROXY CONTEXT

// INFORMATIONAL

Description

The **AgentKey** contract uses a constructor for `initialization` while inheriting from an `initializable` contract (**DecentralizedAutonomousTrust**). This creates inconsistency in the initialization pattern when used with proxy contracts.

```
// AgentKey.sol
contract AgentKey is DecentralizedAutonomousTrust {
    constructor(
        uint256 _initReserve,
        address _currencyAddress,
        uint256 _initGoal,
        uint256 _buySlopeNum,
        uint256 _buySlopeDen,
        uint256 _investmentReserveBasisPoints,
        uint256 _setupFee,
        address payable _setupFeeRecipient,
        string memory _name,
        string memory _symbol
    ) public {
        initialize(
            _initReserve,
            _currencyAddress,
            _initGoal,
            _buySlopeNum,
            _buySlopeDen,
            _investmentReserveBasisPoints,
            _setupFee,
            _setupFeeRecipient,
            _name,
            _symbol
        );
    }
}

// DecentralizedAutonomousTrust.sol
function initialize(
    uint _initReserve,
    address _currencyAddress,
    ...
) public {
```

```
// Initialize logic
}
```

Impact :
1. Incorrect initialization sequence when deployed behind a proxy
2. Constructor code not included in the deployed proxy implementation
3. Broken initialization pattern inheritance chain

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

Convert the constructor to an initializer function:

```
contract AgentKey is DecentralizedAutonomousTrust {
/// @custom:oz-upgrades-unsafe-allow constructor
    constructor() initializer {}

    function initialize(
        uint256 _initReserve,
        address _currencyAddress,
        uint256 _initGoal,
        uint256 _buySlopeNum,
        uint256 _buySlopeDen,
        uint256 _investmentReserveBasisPoints,
        uint256 _setupFee,
        address payable _setupFeeRecipient,
        string memory _name,
        string memory _symbol
    ) public initializer {
        super.initialize(
            _initReserve,
            _currencyAddress,
            _initGoal,
            _buySlopeNum,
            _buySlopeDen,
            _investmentReserveBasisPoints,
            _setupFee,
            _setupFeeRecipient,
            _name,
            _symbol
        );
```

```
    }
}
```

Remediation

RISK ACCEPTED: No upgradability will be used because `AgentCoin` team decided not to change the current code.

8. AUTOMATED TESTING

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their abis and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

```
Reentrancy in ContinuousOffering._buy(address,address,uint256,uint256) (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#602-667):
        External calls:
        - _collectInvestment(_from,_currencyValue,msg.value,false) (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#618)
                - (success) = recipient.call.value(amount)() (node_modules/@openzeppelin/contracts-ethereum-package/contracts/utils/Address.sol#67)
                - (success,returndata) = address(token).call(data) (node_modules/@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/SafeERC20.sol#67)
                - Address.sendValue(msg.sender,refund) (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#298)
                - currency.safeTransferFrom(_from,address(this),_quantityToInvest) (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#311)
        External calls sending eth:
        - _collectInvestment(_from,_currencyValue,msg.value,false) (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#618)
                - (success) = recipient.call.value(amount)() (node_modules/@openzeppelin/contracts-ethereum-package/contracts/utils/Address.sol#67)
        State variables written after the call(s):
        - state = STATE_RUN (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#630)
        ContinuousOffering.state (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#155) can be used in cross function reentrancies:
        - ContinuousOffering._burn(address,uint256,bool) (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#245-262)
        - ContinuousOffering._buy(address,address,uint256,uint256) (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#602-667)
        - ContinuousOffering._sell(address,address,uint256,uint256) (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#789-816)
        - ContinuousOffering._transfer(address,address,uint256) (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#233-242)
        - ContinuousOffering.estimateBuyValue(uint256) (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#495-600)
        - ContinuousOffering.estimateSellValue(uint256) (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#712-787)
        - ContinuousOffering.state (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#155)
Reentrancy in ContinuousOffering._buy(address,address,uint256,uint256) (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#602-667):
        External calls:
        - _collectInvestment(_from,_currencyValue,msg.value,false) (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#618)
                - (success) = recipient.call.value(amount)() (node_modules/@openzeppelin/contracts-ethereum-package/contracts/utils/Address.sol#67)
                - (success,returndata) = address(token).call(data) (node_modules/@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/SafeERC20.sol#67)
                - Address.sendValue(msg.sender,refund) (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#298)
                - currency.safeTransferFrom(_from,address(this),_quantityToInvest) (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#311)
        - _transferCurrency(setupFeeRecipient,setupFee) (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#644)
                - Address.sendValue(_to,_amount) (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#325)
                - (success) = recipient.call.value(amount)() (node_modules/@openzeppelin/contracts-ethereum-package/contracts/utils/Address.sol#67)
                - (success,returndata) = address(token).call(data) (node_modules/@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/SafeERC20.sol#67)
                - currency.safeTransfer(_to,_amount) (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#329)
        - _mint(_to,tokenValue) (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#666)
                - whitelist.authorizeTransfer(_from,_to,_value,_isSell) (node_modules/@fairmint/c-org-contracts/contracts/ContinuousOffering.sol#201)
        External calls sending eth:
```

```
INFO:Detectors:
DecentralizedAutonomousTrust._distributeInvestment(uint256) (node_modules/@fairmint/c-org-contracts/contracts/DecentralizedAutonomousTrust.sol#235-252) performs a multiplication
on the result of a division:
        - reserve /= BASIS_POINTS_DEN (node_modules/@fairmint/c-org-contracts/contracts/DecentralizedAutonomousTrust.sol#244)
        - fee = reserve.mul(feeBasisPoints) (node_modules/@fairmint/c-org-contracts/contracts/DecentralizedAutonomousTrust.sol#246)
BigDiv.bigDiv2x1(uint256,uint256,uint256) (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#36-103) performs a multiplication on the result of a division:
        - value = numMin / factor (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#96)
        - temp = numMax / temp (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#100)
        - value = value.mul(temp) (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#101)
BigDiv.bigDiv2x1(uint256,uint256,uint256) (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#36-103) performs a multiplication on the result of a division:
        - factor *= temp (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#79)
        - factor /= MAX_BEFORE_SQUARE (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#94)
        - temp = numMax / temp (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#100)
BigDiv.bigDiv2x1(uint256,uint256,uint256) (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#36-103) performs a multiplication on the result of a division:
        - value = numMax / factor (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#80)
        - value = value.mul(numMin) (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#82)
BigDiv.bigDiv2x1(uint256,uint256,uint256) (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#36-103) performs a multiplication on the result of a division:
        - value = value.mul(numMin) (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#66)
        - value /= temp (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#86)
BigDiv.bigDiv2x2(uint256,uint256,uint256,uint256) (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#151-233) performs a multiplication on the result of a division
:
        - factor /= MAX_BEFORE_SQUARE (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#227)
        - temp /= MAX_BEFORE_SQUARE + 1 (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#230)
        - factor *= temp (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#231)
BigDiv.bigDiv2x2(uint256,uint256,uint256,uint256) (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#151-233) performs a multiplication on the result of a division
:
        - factor *= temp (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#210)
        - factor /= MAX_BEFORE_SQUARE (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#227)
        - temp /= MAX_BEFORE_SQUARE + 1 (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#230)
BigDiv.bigDiv2x2(uint256,uint256,uint256,uint256) (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#151-233) performs a multiplication on the result of a division
:
        - value = numMax / factor (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#212)
        - value = value.mul(numMin) (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#214)
BigDiv.bigDiv2x2(uint256,uint256,uint256,uint256) (node_modules/@fairmint/c-org-contracts/contracts/math/BigDiv.sol#151-233) performs a multiplication on the result of a division
:
```

All issues identified by `Slither` were proved to be false positives or have been added to the issue list in this report.

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.