// HALBORN

DRAFT

Pangolin - RewarderViaMultiplier

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **January 10th, 2022 - January 11th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 3
CONTACTS 3
1 EXECUTIVE OVERVIEW 4
1.1 INTRODUCTION 5
1.2 AUDIT SUMMARY 5
1.3 TEST APPROACH & METHODOLOGY 5
RISK METHODOLOGY 6
1.4 SCOPE 8
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 9
3 FINDINGS & TECH DETAILS 10
3.1 (HAL-01) OUT OF BOUNDS ARRAY ACCESS - HIGH 12
Description 12
Code Location 12
Risk Level 13
Recommendation 14
Remediation Plan 17
3.2 (HAL-02) DOS WITH BLOCK GAS LIMIT - LOW 18
Description 18
Risk Level 18
Recommendation 18
Remediation Plan 18
3.3 (HAL-03) LACK OF OVERFLOW PROTECTION - LOW 19
Description 19
Code Location 19

Risk Level 19
Recommendation 19
Remediation Plan 20
3.4 (HAL-04) USING ++I CONSUMES LESS GAS THAN I++ IN LOOPS - INFORMATIONAL 21
Description 21
Code Location 21
Proof of Concept 21
Risk Level 22
Recommendation 22
Remediation Plan 22
4 AUTOMATED TESTING 23
4.1 STATIC ANALYSIS REPORT 24
Description 24
Slither results 24
4.2 AUTOMATED SECURITY SCAN 25
Description 25
MythX results 26

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	01/10/2022	Roberto Reigada
0.2	Document Updates	01/11/2022	Roberto Reigada
0.3	Draft Review	01/11/2022	Gabi Urrutia
1.0	Remediation Plan	01/14/2022	Roberto Reigada
1.1	Remediation Plan Review	01/16/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Roberto Reigada	Halborn	Roberto.Reigada@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Pangolin engaged Halborn to conduct a security audit on their fee collector smart contract beginning on January 10th, 2022 and ending on January 11th, 2022. The security assessment was scoped to the smart contract provided in the Github repository pangolindex/governance.

1.2 AUDIT SUMMARY

The team at Halborn was provided a week for the engagement and assigned a full time security engineer to audit 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 audit is to:

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

In summary, Halborn identified some security risks that were addressed by Pangolin team.

1.3 TEST APPROACH & METHODOLOGY

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

- 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 hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Brownie, Remix IDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.

3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

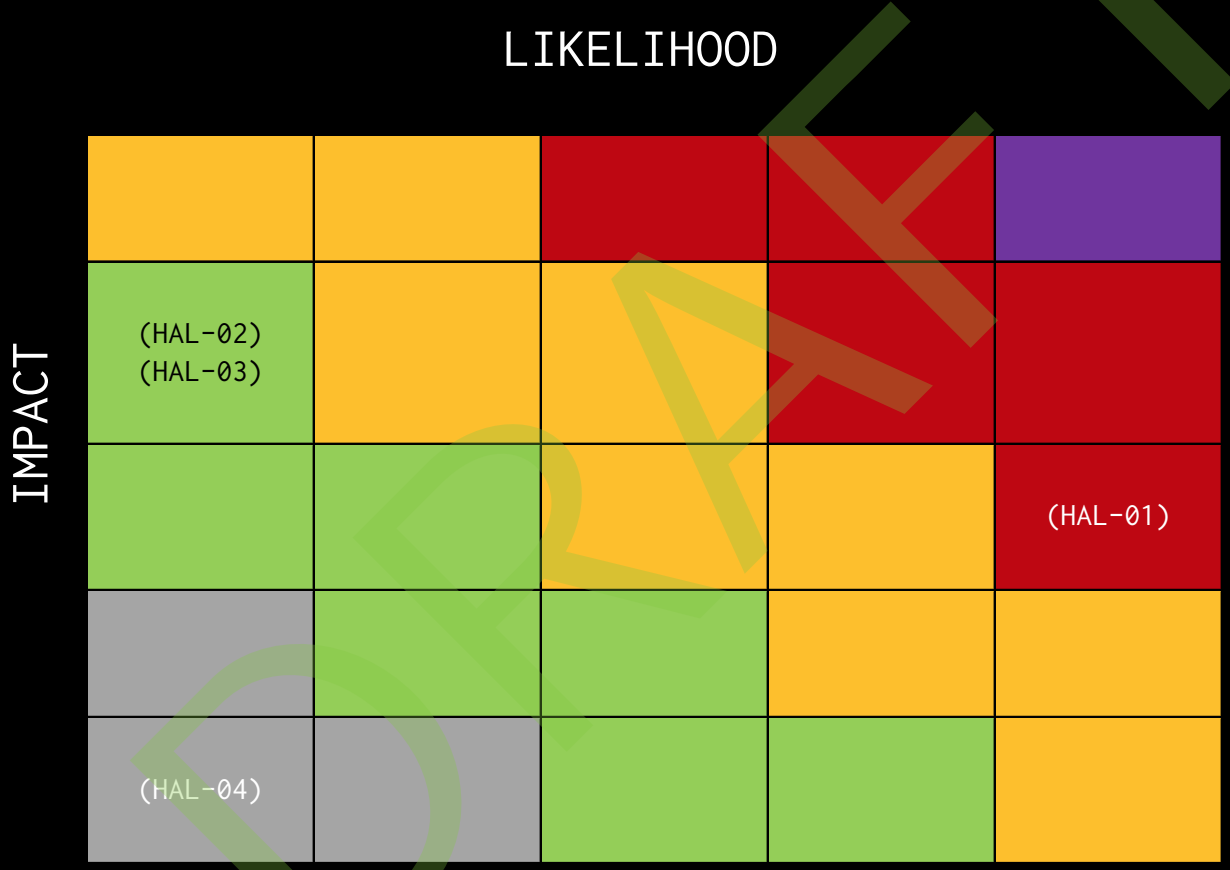
IN-SCOPE:
The security assessment was scoped to the following smart contract:

- RewarderViaMultiplier.sol

Commit ID: 4e2d59227b3198580c07ded7afa0c0367ed272aa
Fixed Commit ID: dfaa62d8a719a24a1688b909a4c7d397d619132b

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	0	2	1



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - OUT OF BOUNDS ARRAY ACCESS	High	SOLVED - 01/14/2022
HAL02 - DOS WITH BLOCK GAS LIMIT	Low	SOLVED - 01/14/2022
HAL03 - LACK OF OVERFLOW PROTECTION	Low	SOLVED - 01/14/2022
HAL04 - USING ++I CONSUMES LESS GAS THAN I++ IN LOOPS	Informational	SOLVED - 01/14/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) OUT OF BOUNDS ARRAY ACCESS - HIGH

Description:

The contract RewarderSimple will always revert during the deployment as the constructor tries to write into a position of an array that does not exist.

Code Location:

Listing 1: RewarderViaMultiplier.sol (Lines 15,48)

```
14 IERC20[] private rewardTokens;
15 uint256[] private rewardMultipliers;
16 address private immutable CHEF_V2;
17
18 /// @dev Should match the precision of the base reward token (PNG)
19 uint256 private constant BASE_REWARD_TOKEN_DIVISOR = 1e18;
20
21 /// @dev Additional reward quantities that might be owed to users
       trying to claim after funds have been exhausted
22 mapping(address => uint256[]) private rewardDebts;
23
24 /// @param _rewardTokens The address of each additional reward
       token
25 /// @param _rewardMultipliers The amount of each additional reward
        token to be claimable for every 1 base reward (PNG) being
       claimed
26 /// @param _chefV2 The address of the chef contract where the base
        reward (PNG) is being emitted
27 /// @notice Each reward multiplier should have a precision
       matching that individual token
28 constructor (
29     address[] memory _rewardTokens,
30     uint256[] memory _rewardMultipliers,
31     address _chefV2
32 ) public {
33     require(
34         _rewardTokens.length > 0
35          && _rewardTokens.length == _rewardMultipliers.length,
```

```
36          "RewarderSimple::Invalid input lengths"
37      );
38
39      require(
40          _chefV2 != address(0),
41          "RewarderSimple::Invalid chef address"
42      );
43
44      for (uint256 i; i < _rewardTokens.length; i++) {
45          require(_rewardTokens[i] != address(0), "RewarderSimple::
                Cannot reward zero address");
46          require(_rewardMultipliers[i] > 0, "RewarderSimple::
                Invalid multiplier");
47
48          rewardTokens[i] = IERC20(_rewardTokens[i]);
49      }
50
51      rewardMultipliers = _rewardMultipliers;
52      CHEF_V2 = _chefV2;
53  }
```

push should be used to write into a new position of an array:
- Line 48; rewardTokens.push(IERC20(_rewardTokens[i]));

A similar issue occurs in:
- Line 57: rewardDebts[user][i].
- Line 72: rewardDebts[user][i].
- Line 75: amounts[i] = rewardBal;
- Line 77: amounts[i] = pendingReward;
- Line 87: rewardDebts[user][i]
- Line 88: amounts[i] = pendingReward;

Risk Level:

Likelihood - 5
Impact - 3

Recommendation:

It is recommended to modify the smart contract code as shown below:

Listing 2: RewarderViaMultiplier.sol (Lines 22,48,71,87)

```
1  // SPDX-License-Identifier: MIT
2
3  pragma solidity 0.6.12;
4
5  import "@boringcrypto/boring-solidity/contracts/libraries/
       BoringERC20.sol";
6  import "@boringcrypto/boring-solidity/contracts/libraries/
       BoringMath.sol";
7
8  import "./interfaces/IRewarder.sol";
9
10 contract RewarderSimple is IRewarder {
11     using BoringMath for uint256;
12     using BoringERC20 for IERC20;
13
14     IERC20[] private rewardTokens;
15     uint256[] private rewardMultipliers;
16     address private immutable CHEF_V2;
17
18     /// @dev Should match the precision of the base reward token (
           PNG)
19     uint256 private constant BASE_REWARD_TOKEN_DIVISOR = 1e18;
20
21     /// @dev Additional reward quantities that might be owed to
           users trying to claim after funds have been exhausted
22     mapping(address => mapping(uint256 => uint256)) private
           rewardDebts;
23
24     /// @param _rewardTokens The address of each additional reward
            token
25     /// @param _rewardMultipliers The amount of each additional
           reward token to be claimable for every 1 base reward (PNG)
           being claimed
26     /// @param _chefV2 The address of the chef contract where the
           base reward (PNG) is being emitted
27     /// @notice Each reward multiplier should have a precision
           matching that individual token
28     constructor (
29         address[] memory _rewardTokens,
```

```
30          uint256[] memory _rewardMultipliers,
31          address _chefV2
32      ) public {
33          require(
34              _rewardTokens.length > 0
35               && _rewardTokens.length == _rewardMultipliers.length,
36              "RewarderSimple::Invalid input lengths"
37          );
38
39          require(
40              _chefV2 != address(0),
41              "RewarderSimple::Invalid chef address"
42          );
43
44          for (uint256 i; i < _rewardTokens.length; i++) {
45              require(_rewardTokens[i] != address(0), "
                    RewarderSimple::Cannot reward zero address");
46              require(_rewardMultipliers[i] > 0, "RewarderSimple::
                    Invalid multiplier");
47
48              rewardTokens.push(IERC20(_rewardTokens[i]));
49          }
50
51          rewardMultipliers = _rewardMultipliers;
52          CHEF_V2 = _chefV2;
53      }
54
55      function onReward(uint256, address user, address to, uint256
            rewardAmount, uint256) onlyMCV2 override external {
56          for (uint256 i; i < rewardTokens.length; i++) {
57              uint256 pendingReward = rewardDebts[user][i] +
                    rewardAmount.mul(rewardMultipliers[i]) /
                    BASE_REWARD_TOKEN_DIVISOR;
58              uint256 rewardBal = rewardTokens[i].balanceOf(address(
                    this));
59              if (pendingReward > rewardBal) {
60                  rewardDebts[user][i] = pendingReward - rewardBal;
61                  rewardTokens[i].safeTransfer(to, rewardBal);
62              } else {
63                  rewardDebts[user][i] = 0;
64                  rewardTokens[i].safeTransfer(to, pendingReward);
65              }
66          }
67      }
```

```
68
69      /// @notice Shows pending tokens that can be currently claimed
70      function pendingTokens(uint256, address user, uint256
            rewardAmount) override external view returns (IERC20[]
            memory tokens, uint256[] memory amounts) {
71          amounts = new uint256[](rewardTokens.length);
72          for (uint256 i; i < rewardTokens.length; i++) {
73              uint256 pendingReward = rewardDebts[user][i] +
                    rewardAmount.mul(rewardMultipliers[i]) /
                    BASE_REWARD_TOKEN_DIVISOR;
74              uint256 rewardBal = rewardTokens[i].balanceOf(address(
                    this));
75              if (pendingReward > rewardBal) {
76                  amounts[i] = rewardBal;
77              } else {
78                  amounts[i] = pendingReward;
79              }
80          }
81          return (rewardTokens, amounts);
82      }
83
84      /// @notice Shows pending tokens including rewards accrued
            after the funding has been exhausted
85      /// @notice these extra rewards could be claimed if more
            funding is added to the contract
86      function pendingTokensDebt(uint256, address user, uint256
            rewardAmount) external view returns (IERC20[] memory tokens
            , uint256[] memory amounts) {
87          amounts = new uint256[](rewardTokens.length);
88          for (uint256 i; i < rewardTokens.length; i++) {
89              uint256 pendingReward = rewardDebts[user][i] +
                    rewardAmount.mul(rewardMultipliers[i]) /
                    BASE_REWARD_TOKEN_DIVISOR;
90              amounts[i] = pendingReward;
91          }
92          return (rewardTokens, amounts);
93      }
94
95      modifier onlyMCV2 {
96          require(
97              msg.sender == CHEF_V2,
98              "Only MCV2 can call this function."
99          );
100         _;
```

```
101     }
102
103 }
```

Remediation Plan:

SOLVED: The Pangolin team addressed all the out of bound accesses in the smart contract, following our suggested code.

3.2 (HAL-02) DOS WITH BLOCK GAS LIMIT - LOW

Description:

In the constructor of the contract RewarderSimple, different rewardTokens are stored in an array. This array has no limitation and can contain infinite rewardTokens. This array is iterated later on by the onReward() function. If the array is big enough, the block gas limit will be reached and the transaction will never get processed. For that reason, it is recommended to add a require statement in the constructor that does not allow adding, for example, more than 100 different rewardTokens.

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

It is recommended to add a require statement in the constructor that does not allow adding, for example, more than 100 different rewardTokens.

Remediation Plan:

SOLVED: The Pangolin team added the suggested require statement in the constructor. It is not possible anymore to deploy the contract with more than 100 rewardTokens.

3.3 (HAL-03) LACK OF OVERFLOW PROTECTION - LOW

Description:

In the contract RewarderViaMultiplier.sol, in the calculation of the variable pendingReward the addition operation is vulnerable to overflow. The likelihood of overflowing this operation is very low, as that could happen only in the case that the pendingReward is higher than UINT256_MAX_VALUE = 2^256 - 1.

Code Location:

RewarderViaMultiplier.sol
- Line 57: uint256 pendingReward = rewardDebts[user][i] + rewardAmount.mul(rewardMultipliers[i])/ BASE_REWARD_TOKEN_DIVISOR;
- Line 72: uint256 pendingReward = rewardDebts[user][i] + rewardAmount.mul(rewardMultipliers[i])/ BASE_REWARD_TOKEN_DIVISOR;
- Line 87: uint256 pendingReward = rewardDebts[user][i] + rewardAmount.mul(rewardMultipliers[i])/ BASE_REWARD_TOKEN_DIVISOR;

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

It is recommended to use SafeMath.add() function in the addition operation as shown below:
uint256 pendingReward = rewardDebts[user][i].add(rewardAmount.mul(rewardMultipliers[i])/ BASE_REWARD_TOKEN_DIVISOR);

Remediation Plan:

SOLVED: The Pangolin team uses now SafeMath.add() to perform the mentioned operations.

3.4 (HAL-04) USING ++I CONSUMES LESS GAS THAN I++ IN LOOPS - INFORMATIONAL

Description:

In the loop, the variable `i` is incremented using `i++`. It is known that, in loops, using `++i` costs less gas per iteration than `i++`.

Code Location:

RewarderViaMultiplier.sol
- Line 44: `for (uint256 i; i < _rewardTokens.length; i++){`
- Line 56: `for (uint256 i; i < rewardTokens.length; i++){`
- Line 71: `for (uint256 i; i < rewardTokens.length; i++){`
- Line 86: `for (uint256 i; i < rewardTokens.length; i++){`

Proof of Concept:

For example, based in the following test contract:

Listing 3: Test.sol

```
1 //SPDX-License-Identifier: MIT
2 pragma solidity 0.8.9;
3
4 contract test {
5     function postiincrement(uint256 iterations) public {
6         for (uint256 i = 0; i < iterations; i++) {
7         }
8     }
9     function preiincrement(uint256 iterations) public {
10        for (uint256 i = 0; i < iterations; ++i) {
11        }
12    }
13 }
```

We can see the difference in the gas costs:

```
>>> test_contract.postiincrement(1)
Transaction sent: 0xlecede6b109b707786d3685bd71dd9f22dc389957653036ca04c4cd2e72c5e0b
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 44
  test.postiincrement confirmed   Block: 13622335   Gas used: 21620 (0.32%)

<Transaction '0xlecede6b109b707786d3685bd71dd9f22dc389957653036ca04c4cd2e72c5e0b'>
>>> test_contract.preiincrement(1)
Transaction sent: 0x205f09a4d2268de4c1a40f35bb2ec2847bf2ab8d584909b42c71a022b047614a
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 45
  test.preiincrement confirmed   Block: 13622336   Gas used: 21593 (0.32%)

<Transaction '0x205f09a4d2268de4c1a40f35bb2ec2847bf2ab8d584909b42c71a022b047614a'>
>>> test_contract.postiincrement(10)
Transaction sent: 0x98c04430526a59balcf947c114b62666a4417165947d31bf300cd6ae68328033
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 46
  test.postiincrement confirmed   Block: 13622337   Gas used: 22673 (0.34%)

<Transaction '0x98c04430526a59balcf947c114b62666a4417165947d31bf300cd6ae68328033'>
>>> test_contract.preiincrement(10)
Transaction sent: 0xf060d04714eff8482a828342414d5a20be9958c822d42860e7992aba20elde05
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 47
  test.preiincrement confirmed   Block: 13622338   Gas used: 22601 (0.34%)

<Transaction '0xf060d04714eff8482a828342414d5a20be9958c822d42860e7992aba20elde05'>
```

Risk Level:

Likelihood - 1

Impact - 1

Recommendation:

It is recommended to use `++i` instead of `i++` to increment the value of an uint variable inside a loop. This is not applicable outside of loops.

Remediation Plan:

SOLVED: The Pangolin team uses now `++i` to increment the `i` variable inside loops, saving some gas.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats, Slither was run on the all-scoped 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.

Slither results:

RewarderViaMultiplier.sol

```
RewarderSimple.onReward(uint256,address,address,uint256,uint256).i (contracts/RewarderViaMultiplier.sol#56) is a local variable never initialized
RewarderSimple.pendingTokensDebt(uint256,address,uint256).i (contracts/RewarderViaMultiplier.sol#86) is a local variable never initialized
RewarderSimple.pendingTokens(uint256,address,uint256).i (contracts/RewarderViaMultiplier.sol#71) is a local variable never initialized
RewarderSimple.constructor(address[],uint256[],address).i (contracts/RewarderViaMultiplier.sol#44) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

RewarderSimple.onReward(uint256,address,address,uint256,uint256) (contracts/RewarderViaMultiplier.sol#55-67) has external calls inside a loop: rewardBal = rewardTokens[i].balanceOf(address(this)) (contracts/RewarderViaMultiplier.sol#58)
RewarderSimple.pendingTokens(uint256,address,uint256) (contracts/RewarderViaMultiplier.sol#70-81) has external calls inside a loop: rewardBal = rewardTokens[i].balanceOf(address(this)) (contracts/RewarderViaMultiplier.sol#73)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

BoringERC20.safeDecimals(IERC20) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringERC20.sol#17-20) is never used and should be removed
BoringERC20.safeName(IERC20) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringERC20.sol#12-15) is never used and should be removed
BoringERC20.safeSymbol(IERC20) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringERC20.sol#7-10) is never used and should be removed
BoringERC20.safeTransferFrom(IERC20,address,address,uint256) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringERC20.sol#27-30) is never used and should be removed
BoringMath.add(uint256,uint256) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringMath.sol#5) is never used and should be removed
BoringMath.sub(uint256,uint256) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringMath.sol#6) is never used and should be removed
BoringMath.to128(uint256) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringMath.sol#10-11) is never used and should be removed
BoringMath.to32(uint256) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringMath.sol#16-19) is never used and should be removed
BoringMath.to64(uint256) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringMath.sol#12-15) is never used and should be removed
BoringMath128.add(uint128,uint128) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringMath.sol#23) is never used and should be removed
BoringMath128.sub(uint128,uint128) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringMath.sol#24) is never used and should be removed
BoringMath32.add(uint32,uint32) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringMath.sol#33) is never used and should be removed
BoringMath32.sub(uint32,uint32) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringMath.sol#34) is never used and should be removed
BoringMath64.add(uint64,uint64) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringMath.sol#28) is never used and should be removed
BoringMath64.sub(uint64,uint64) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringMath.sol#29) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Low level call in BoringERC20.safeSymbol(IERC20) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringERC20.sol#7-10):
        - (success,data) = address(token).staticcall(abi.encodeWithSelector(0x95d89b41)) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringERC20.sol#8)
Low level call in BoringERC20.safeName(IERC20) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringERC20.sol#12-15):
        - (success,data) = address(token).staticcall(abi.encodeWithSelector(0x06fdde03)) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringERC20.sol#13)
Low level call in BoringERC20.safeDecimals(IERC20) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringERC20.sol#17-20):
        - (success,data) = address(token).staticcall(abi.encodeWithSelector(0x313ce567)) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringERC20.sol#18)
Low level call in BoringERC20.safeTransfer(IERC20,address,uint256) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringERC20.sol#22-25):
        - (success,data) = address(token).call(abi.encodeWithSelector(0xa9059cbb,to,amount)) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringERC20.sol#23)
Low level call in BoringERC20.safeTransferFrom(IERC20,address,address,uint256) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringERC20.sol#27-30):
        - (success,data) = address(token).call(abi.encodeWithSelector(0x23b872dd,from,to,amount)) (node_modules/@boringcrypto/boring-solidity/contracts/libraries/BoringERC20.sol#28)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Variable RewarderSimple.CHEF_V2 (contracts/RewarderViaMultiplier.sol#16) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

- No major issues were found by Slither.

4.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on all the contracts and sent the compiled results to the analyzers to locate any vulnerabilities.

MythX results:

RewarderViaMultiplier.sol

Report for contracts/RewarderViaMultiplier.sol
https://dashboard.mythx.io/#/console/analyses/8ac25df1-f11e-4749-9f95-f8749356b029

Line	SWC Title	Severity	Short Description
44	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
45	(SWC-110) Assert Violation	Unknown	Out of bounds array access
46	(SWC-110) Assert Violation	Unknown	Out of bounds array access
48	(SWC-110) Assert Violation	Unknown	Out of bounds array access
56	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
57	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
57	(SWC-110) Assert Violation	Unknown	Out of bounds array access
57	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
58	(SWC-110) Assert Violation	Unknown	Out of bounds array access
60	(SWC-110) Assert Violation	Unknown	Out of bounds array access
60	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
61	(SWC-110) Assert Violation	Unknown	Out of bounds array access
63	(SWC-110) Assert Violation	Unknown	Out of bounds array access
64	(SWC-110) Assert Violation	Unknown	Out of bounds array access
71	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
72	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
72	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
72	(SWC-110) Assert Violation	Unknown	Out of bounds array access
73	(SWC-110) Assert Violation	Unknown	Out of bounds array access
75	(SWC-110) Assert Violation	Unknown	Out of bounds array access
77	(SWC-110) Assert Violation	Unknown	Out of bounds array access
86	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
87	(SWC-110) Assert Violation	Unknown	Out of bounds array access
87	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
87	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
88	(SWC-110) Assert Violation	Unknown	Out of bounds array access

- Integer Overflows and Underflows are correctly flagged by MythX, although they are very unlikely to occur.
- Some assert violations are correctly flagged by MythX as is described in one of the findings.

THANK YOU FOR CHOOSING

// HALBORN