MatrixSwap - Staking

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **January 28th, 2022 - February 6th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	01/28/2022	Roberto Reigada
0.2	Document Updates	02/06/2022	Roberto Reigada
0.3	Document Review	02/06/2022	Gabi Urrutia
1.0	Remediation Plan	02/22/2022	Roberto Reigada
1.1	Remediation Plan Review	02/22/2022	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

MatrixSwap engaged Halborn to conduct a security audit on their staking smart contract beginning on January 28th, 2022 and ending on February 6th, 2022. The security assessment was scoped to the smart contract provided in the GitHub repository Matrixswap/matrix-staking - master branch

1.2 AUDIT SUMMARY

The team at Halborn was provided one 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 mostly addressed by the MatrixSwap 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.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

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:

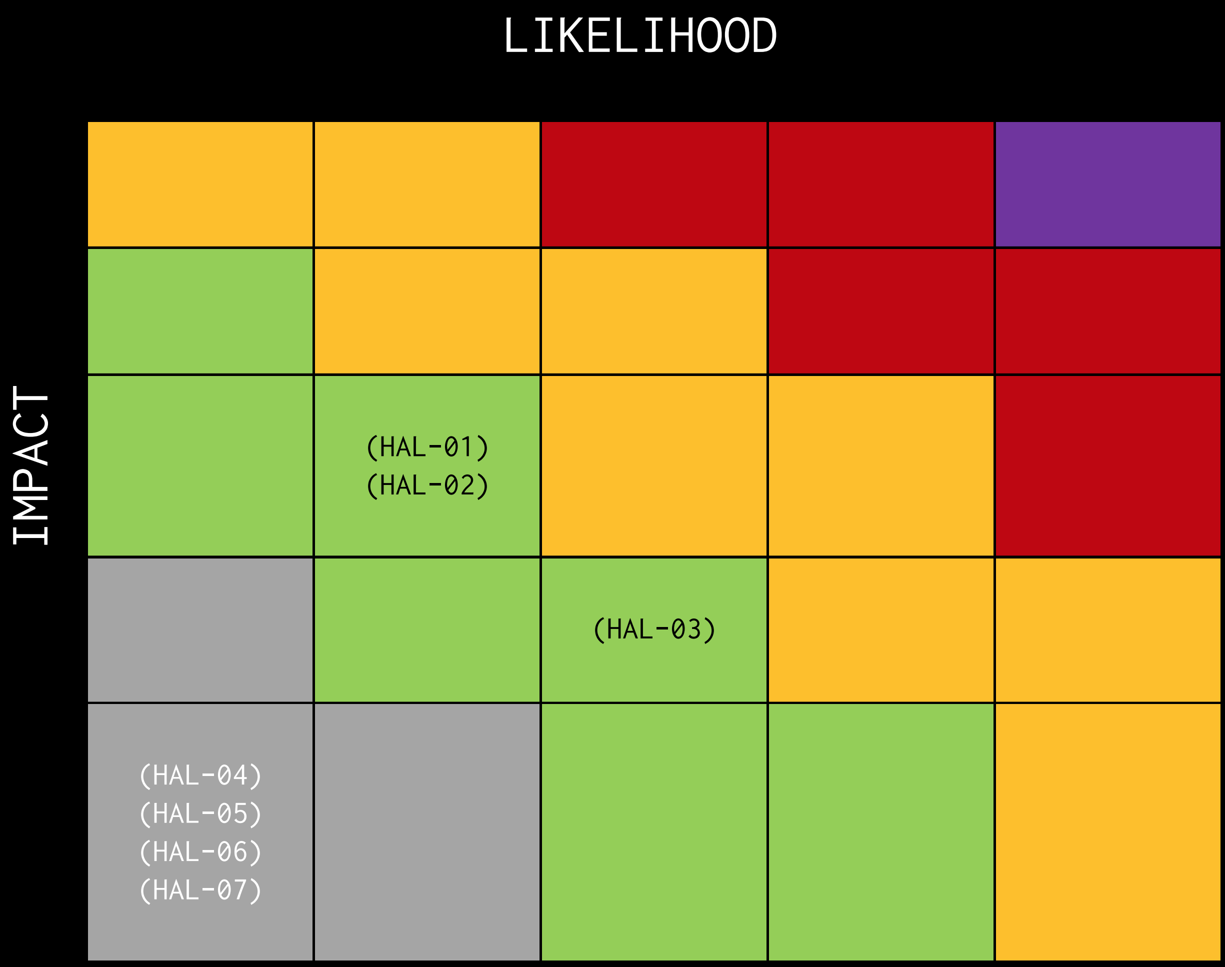
MatrixStaking.sol

Commit ID: c7d8d67b80cd9cfc87c064af1ad2eb2aab082b52

Fixed Commit ID: 9cbc0afde82da74c02b2c9704b4bdb8b40bf2971

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	3	4



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - UNCHECKED TRANSFERS	Low	SOLVED - 02/22/2022
HAL02 - UNUSED RETURNS	Low	RISK ACCEPTED
HAL03 - MISSING ZERO ADDRESS CHECKS	Low	RISK ACCEPTED
HAL04 - STATE VARIABLES MISSING CONSTANT MODIFIER	Informational	SOLVED - 02/22/2022
HAL05 - STATE VARIABLE MISSING IMMUTABLE MODIFIER	Informational	SOLVED - 02/22/2022
HAL06 - POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	SOLVED - 02/22/2022
HAL07 - INCORRECT ERC20 TOKEN NAME	Informational	SOLVED - 02/22/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) UNCHECKED TRANSFERS - LOW

Description:

In the contract MatrixStaking the return value of some external transfer/transferFrom calls are not checked. Several tokens do not revert in case of failure and return false. Checking the return value is also considered a best practice.

Code Location:

- Line 45: `matrix.transferFrom(msg.sender, address(this), _amount);`
- Line 56: `matrix.transfer(msg.sender, what);`

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

It is recommended to use SafeERC20, or ensure that the transfer/transferFrom return value is checked.

Remediation Plan:

SOLVED: The MatrixSwap team now uses the SafeERC20 library to perform all token transfers.

3.2 (HAL-02) UNUSED RETURNS - LOW

Description:

The return value of some external calls are not stored in a local or state variable. In the contract MatrixStaking there is an instance where an external method is being called, and the return values are ignored.

Code Location:

- Line 25:
  `IERC20(usdc).approve(address(uniswapV2Router), uint256(-1));`

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Ensure that all the return values of the function calls are used. Add a return value check to avoid an unexpected crash of the contract.

Remediation Plan:

RISK ACCEPTED: The MatrixSwap team accepted this risk of this finding.

3.3 (HAL-03) MISSING ZERO ADDRESS CHECKS - LOW

Description:

The constructor of the MatrixStaking contract is missing address validation. Every address should be validated and checked that is different from zero. This is also considered a best practice.

Code location:

Listing 1: MatrixStaking.sol (Line 23)

```
22 constructor(IERC20 _matrix) public {
23     matrix = _matrix;
24     minSwapAmount = 1000000000; // 1000 usdc
25     IERC20(usdc).approve(address(uniswapV2Router), uint256(-1));
26 }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

It is recommended to validate that every address input is different from zero.

Remediation Plan:

RISK ACCEPTED: The MatrixSwap team accepted this risk of this finding.

3.4 (HAL-04) STATE VARIABLES MISSING CONSTANT MODIFIER - INFORMATIONAL

Description:

State variables can be declared as constant or immutable. In both cases, the variables cannot be modified after the contract has been constructed. For constant variables, the value has to be fixed at compile-time, while for immutable, it can still be assigned at construction time. The following state variables are missing the constant modifier:

Listing 2: MatrixStaking.sol

```
IUniswapV2Router02 public uniswapV2Router  = IUniswapV2Router02(
    0xA102072A4C07F06EC3B4900FDC4C7B80b6c57429
);
address public usdc = 0x2791Bca1f2de4661ED88A30C99A7a9449Aa84174;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to add the constant modifier to the state variable mentioned.

Remediation Plan:

SOLVED: The MatrixSwap team declared the state variable mentioned as constant

3.5 (HAL-05) STATE VARIABLE MISSING IMMUTABLE MODIFIER - INFORMATIONAL

Description:

In the contract MatrixStaking, the state variable matrix can be declared as immutable to reduce the gas costs.

The immutable keyword was added to Solidity in 0.6.5. State variables can be marked immutable which causes them to be read-only, but only assignable in the constructor.

Code Location:

Listing 3: MatrixStaking.sol

```
IERC20 public matrix;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to add the immutable modifier to the matrix state variable.

Remediation Plan:

SOLVED: The MatrixSwap team declared the state variable mentioned as immutable

3.6 (HAL-06) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In the following contracts there are functions marked as public but they are never directly called within the same contract or in any of their descendants:

MatrixStaking.sol
- enter() (MatrixStaking.sol#30-46)
- leave() (MatrixStaking.sol#50-57)

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

If the functions are not intended to be called internally or by their descendants, it is better to mark all of these functions as external to reduce the gas costs.

Remediation Plan:

SOLVED: The MatrixSwap team declared the mentioned functions as external.

3.7 (HAL-07) INCORRECT ERC20 TOKEN NAME - INFORMATIONAL

Description:

The contract MatrixStaking is an ERC20 contract, which its ERC20.name() is Matrix and its ERC20.symbol() is xMatrix. Since this contract will be minting xMatrix tokens, it is recommended to set its ERC20.name() to xMatrix instead.

Code location:

Listing 4: MatrixStaking.sol

```
contract MatrixStaking is ERC20("Matrix", "xMatrix"), Ownable {
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to set the contract ERC20.name() to xMatrix instead of Matrix.

Remediation Plan:

SOLVED: The MatrixSwap team set the contract ERC20.name() to xMatrix instead of Matrix.

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:

MatrixStaking.sol

```
MatrixStaking.enter(uint256) (contracts/MatrixStaking.sol#30-46) ignores return value by matrix.transferFrom(msg.sender,address(this),_amount) (contracts/MatrixStaking.sol#45)
MatrixStaking.leave(uint256) (contracts/MatrixStaking.sol#50-57) ignores return value by matrix.transfer(msg.sender,what) (contracts/MatrixStaking.sol#56)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

MatrixStaking.enter(uint256) (contracts/MatrixStaking.sol#30-46) uses a dangerous strict equality:
        - totalShares == 0 || totalMatrix == 0 (contracts/MatrixStaking.sol#36)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

MatrixStaking.constructor(IERC20) (contracts/MatrixStaking.sol#22-26) ignores return value by IERC20(usdc).approve(address(uniswapV2Router),uint256(- 1)) (contracts/MatrixStaking.sol#25)
MatrixStaking.convert() (contracts/MatrixStaking.sol#63-79) ignores return value by uniswapV2Router.swapExactTokensForTokens(tokenAmount,0,path,address(this),block.timestamp) (contracts/MatrixStaking.sol#72-78)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

MatrixStaking.setMinSwapAmount(uint256) (contracts/MatrixStaking.sol#59-61) should emit an event for:
        - minSwapAmount = _amount (contracts/MatrixStaking.sol#60)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

Different versions of Solidity is used:
        - Version used: ['0.6.12', '>=0.6.0<0.8.0', '>=0.6.2']
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/math/SafeMath.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#3)
        - >=0.6.2 (node_modules/@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router01.sol#1)
        - >=0.6.2 (node_modules/@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router02.sol#1)
        - 0.6.12 (contracts/MatrixStaking.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.sol#20-23) is never used and should be removed
ERC20._setupDecimals(uint8) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#287-289) is never used and should be removed
SafeMath.div(uint256,uint256,string) (node_modules/@openzeppelin/contracts/math/SafeMath.sol#190-193) is never used and should be removed
SafeMath.mod(uint256,uint256) (node_modules/@openzeppelin/contracts/math/SafeMath.sol#152-155) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (node_modules/@openzeppelin/contracts/math/SafeMath.sol#210-213) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (node_modules/@openzeppelin/contracts/math/SafeMath.sol#24-28) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (node_modules/@openzeppelin/contracts/math/SafeMath.sol#60-63) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (node_modules/@openzeppelin/contracts/math/SafeMath.sol#70-73) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (node_modules/@openzeppelin/contracts/math/SafeMath.sol#45-53) is never used and should be removed
SafeMath.trySub(uint256,uint256) (node_modules/@openzeppelin/contracts/math/SafeMath.sol#35-38) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/math/SafeMath.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#3) is too complex
Pragma version>=0.6.2 (node_modules/@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router01.sol#1) allows old versions
Pragma version>=0.6.2 (node_modules/@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router02.sol#1) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Function IUniswapV2Router01.WETH() (node_modules/@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router01.sol#5) is not in mixedCase
Parameter MatrixStaking.enter(uint256)._amount (contracts/MatrixStaking.sol#30) is not in mixedCase
Parameter MatrixStaking.leave(uint256)._share (contracts/MatrixStaking.sol#50) is not in mixedCase
Parameter MatrixStaking.setMinSwapAmount(uint256)._amount (contracts/MatrixStaking.sol#59) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (node_modules/@openzeppelin/contracts/utils/Context.sol#21)" inContext (node_modules/@openzeppelin/contracts/utils/Context.sol#15-24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (node_modules/@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router01.sol#10) is too similar to IUniswapV2Rout
er01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (node_modules/@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router01.sol#11)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar

MatrixStaking.constructor(IERC20) (contracts/MatrixStaking.sol#22-26) uses literals with too many digits:
        - minSwapAmount = 1000000000 (contracts/MatrixStaking.sol#24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

MatrixStaking.usdc (contracts/MatrixStaking.sol#18) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/contracts/access/Ownable.sol#54-57)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/contracts/access/Ownable.sol#63-67)
name() should be declared external:
        - ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#64-66)
symbol() should be declared external:
        - ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#72-74)
decimals() should be declared external:
        - ERC20.decimals() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#89-91)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#103-105)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#115-118)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#123-125)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#134-137)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#152-156)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#170-173)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#189-192)
enter(uint256) should be declared external:
        - MatrixStaking.enter(uint256) (contracts/MatrixStaking.sol#30-46)
leave(uint256) should be declared external:
        - MatrixStaking.leave(uint256) (contracts/MatrixStaking.sol#50-57)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

AUTOMATED TESTING

- No major issues found by Slither, although checking the return value of the transfers and approve calls are considered a good practice.

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:

MatrixStaking.sol

Report for contracts/MatrixStaking.sol
https://dashboard.mythx.io/#/console/analyses/85999702-2afc-4ddb-aba1-bbbabfdb9366

Line	SWC Title	Severity	Short Description
67	(SWC-110) Assert Violation	Unknown	Out of bounds array access
68	(SWC-110) Assert Violation	Unknown	Out of bounds array access
69	(SWC-110) Assert Violation	Unknown	Out of bounds array access

- No issues were found by MythX. The Assert Violations are false positives.

THANK YOU FOR CHOOSING

HALBORN