EasyFi Farming

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **August 6th - August 11th 2021**

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) LOCKOUT OWNER ROLE - MEDIUM 12

Description 12

PoC Steps 12

Code Location 13

Risk Level 13

Recommendation 14

Remediation Plan 14

3.2 (HAL-02) MISSING ZERO-ADDRESS CHECK - LOW 15

Description 15

Code Location 15

Risk Level 16

Recommendation 16

Remediation Plan 16

3.3 (HAL-03) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL 17

Description 17

Affected Smart Contract Functions 17

Risk Level 17

Recommendation 17

Remediation Plan 18

3.4 (HAL-04) USE OF BLOCK.TIMESTAMP - INFORMATIONAL 19

Description 19

Code Location 19

Risk Level 21

Recommendation 21

Remediation Plan 22

3.5 (HAL-05) LACK OF REWARD DURATION SETTER FUNCTION - INFORMATIONAL 22

Description 22

Code Location 22

Risk Level 23

Recommendation 23

Remediation Plan 23

4 AUTOMATED TESTING 24

4.1 STATIC ANALYSIS REPORT 25

Description 25

Results 25

4.2 AUTOMATED SECURITY SCAN 27

Description 27

Results 27

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/06/2021	Gokberk Gulgun
0.9	Document Edits	08/10/2021	Gokberk Gulgun
1.0	Final Draft	08/11/2021	Gabi Urrutia
1.1	Remediation Plan	08/17/2021	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
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

EasyFi engaged Halborn to conduct a security audit on their **Farming Smart Contract** beginning on August 06th, 2021 and ending August 11th, 2021. The security assessment was scoped to the smart contract provided in the Github repository EasyFi Farming Smart Contracts.

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 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 to achieve the following:

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

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverable set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure smart-contract development.

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 the smart contract audit.While manual testing is recommended to uncover flaws in logic, process,and implementation; automated testing techniques help enhance coverage of smart contracts 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 read 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.
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions.(Slither)
- Testnet deployment (Truffle, Ganache)

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. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was 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: farming/contracts/farmingFactory.sol

REPOSITORY : EasyFi Farming Smart Contracts

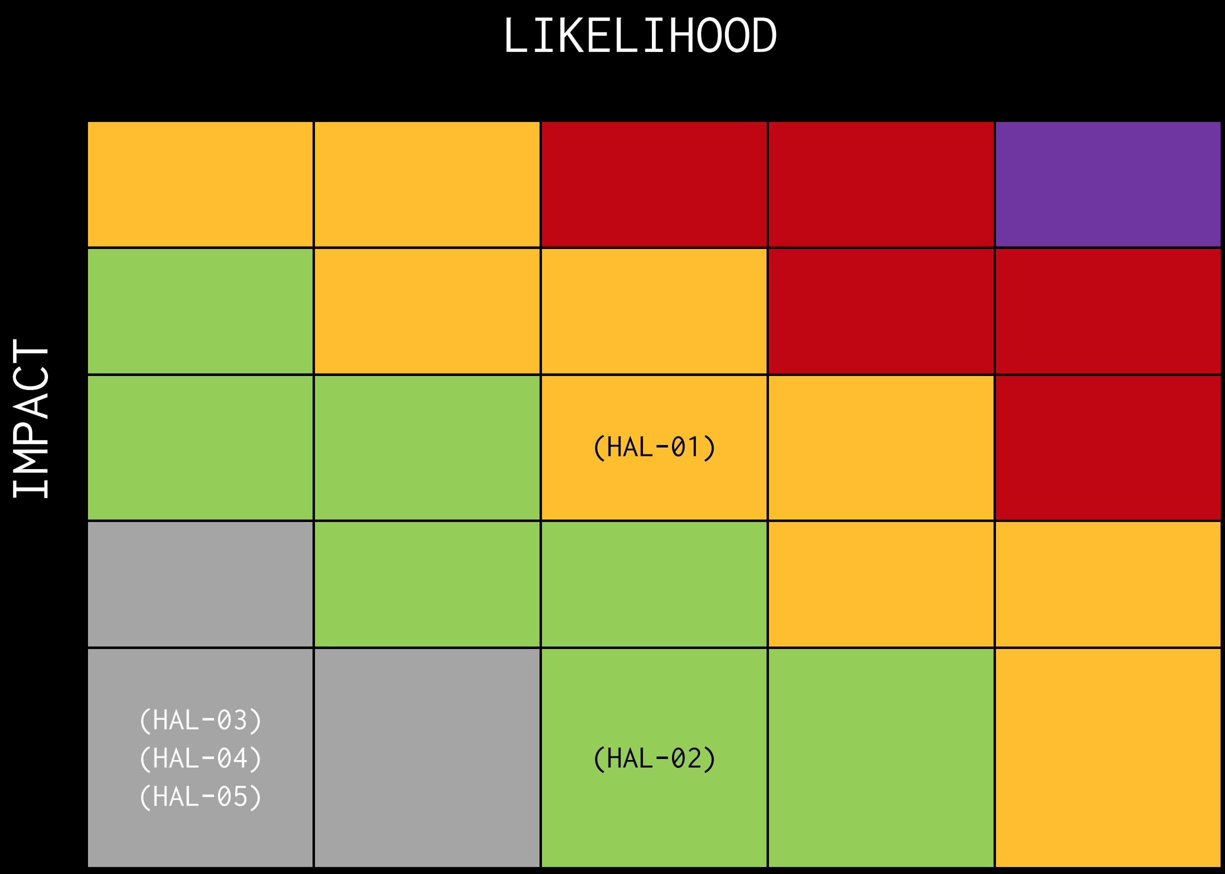
COMMIT ID : f468b5c14093d6b92d88c04f02567750a3284f10

FIXED COMMIT ID: 97764cad9d336e64a59fc381284ee6a5fb9d64d4

OUT-OF-SCOPE : External libraries and economics attacks

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	1	3



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) LOCKOUT OWNER ROLE	Medium	SOLVED - 08/13/2021
(HAL-02) MISSING ZERO-ADDRESS CHECK	Low	SOLVED - 08/13/2021
(HAL-03) POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	SOLVED - 08/13/2021
(HAL-04) USE OF BLOCK.TIMESTAMP	Informational	NOT APPLICABLE
(HAL-05) LACK OF REWARD DURATION SETTER FUNCTION	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

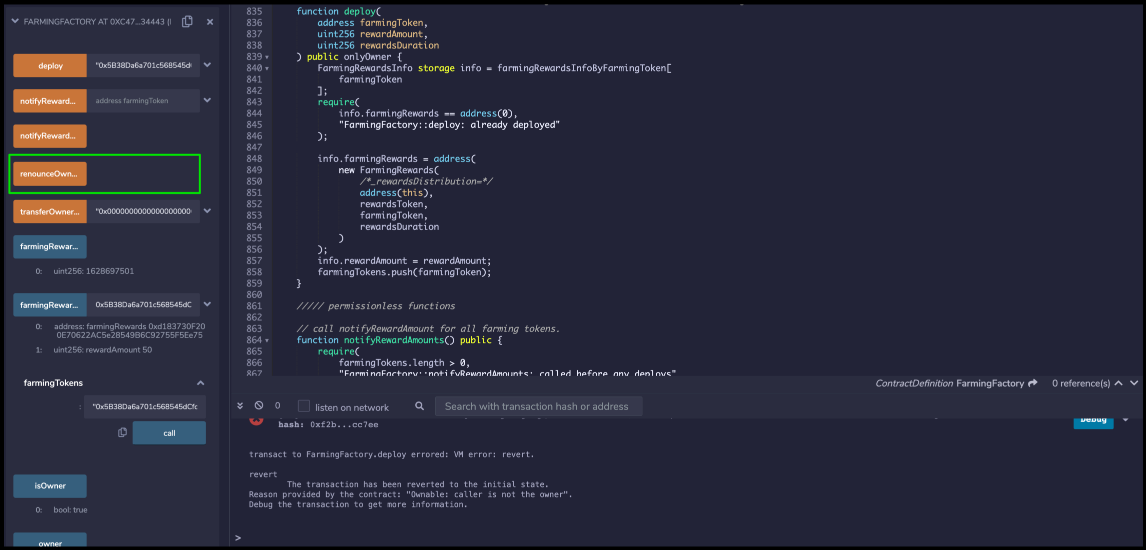
3.1 (HAL-01) LOCKOUT OWNER ROLE - MEDIUM

Description:

The Owner of the contract is usually the account which deploys the contract. As a result, the Owner is able to perform some privileged actions. In the FarmingFactory smart contract, the renounceOwnership function is used to renounce being an owner. The deploy() function in the FarmingFactory smart contract, utilizes new farming reward contract. If an owner is mistakenly renounced administrative access which ends up calling deploy() require msg.sender to be the incorrectly used owner address. In such a case, contracts would have to be redeployed.

PoC Steps:

- Deploy a FarmingFactory contract.
- Renounce an owner of the contract.
- Deploy function is not accessible with the current owner of the function.



FINDINGS & TECH DETAILS

Code Location:

farmingFactory.sol Line #840

Listing 1: farmingFactory.sol (Lines 840)

```
840     function deploy(
841         address farmingToken,
842         uint256 rewardAmount,
843         uint256 rewardsDuration
844     ) public onlyOwner {
845         FarmingRewardsInfo storage info =
            farmingRewardsInfoByFarmingToken[
846             farmingToken
847         ];
848         require(
849             info.farmingRewards == address(0),
850             "FarmingFactory::deploy: already deployed"
851         );
852
853         info.farmingRewards = address(
854             new FarmingRewards(
855                 /*_rewardsDistribution=*/
856                 address(this),
857                 rewardsToken,
858                 farmingToken,
859                 rewardsDuration
860             )
861         );
862         info.rewardAmount = rewardAmount;
863         farmingTokens.push(farmingToken);
864     }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

It's recommended that the Owner is not able to call renounceOwnership without transferring the Ownership to other address before. In addition, if a multi-signature wallet is used, calling renounceOwnership function should be confirmed for two or more users. As an other solution, Renounce Ownership functionality can be disabled with the following line of codes.

Listing 2: Disable Renounce Ownership (Lines 2)

```
function renounceOwnership () public override onlyOwner {
    revert ("can 't renounceOwnership here "); // not possible
        with this smart contract
}
```

Remediation Plan:

SOLVED: EasyFi team removed renounceOwnership function.

3.2 (HAL-02) MISSING ZERO-ADDRESS CHECK - LOW

Description:

The constructors from **FarmingRewards** and **FarmingFactory** contract should perform a zero-address check when receives an address as a user-supplied parameter.

Code Location:

FarmingFactory - Line #832

Listing 3: FarmingFactory.sol (Lines 832)

```
824     constructor(address _rewardsToken, uint256
            _farmingRewardsGenesis)
825         Ownable()
826     {
827         require(
828             _farmingRewardsGenesis >= block.timestamp,
829             "FarmingFactory::constructor: genesis too soon"
830         );
831
832         rewardsToken = _rewardsToken;
833         farmingRewardsGenesis = _farmingRewardsGenesis;
834     }
```

FarmingRewards - Line #618

Listing 4: FarmingRewards.sol (Lines 626,627)

```
618     constructor(
619         address _rewardsDistribution,
620         address _rewardsToken,
621         address _farmingToken,
622         uint256 _rewardsDuration
623     ) {
624         rewardsToken = IERC20(_rewardsToken);
```

```
        farmingToken = IERC20(_farmingToken);
        rewardsDistribution = _rewardsDistribution;
        rewardsDuration = _rewardsDuration;
    }
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendation:

Add proper address validation when assigning a value to a variable from user-supplied data.

For example:

Listing 5: Modifier.sol (Lines 2,3,4)

```
1 modifier validAddress(address addr) {
2     require(addr != address(0), "Address cannot be 0x0");
3     require(addr != address(this), "Address cannot be contract");
4     _;
5   }
```

Remediation Plan:

SOLVED: EasyFi team added the address validation.

3.3 (HAL-03) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In the public functions, array arguments are immediately copied to memory, while external functions can read directly from calldata. Reading calldata is cheaper than memory allocation. Public functions need to write the arguments to memory because public functions may be called internally. Internal calls are passed internally by pointers to memory. Thus, the function expects its arguments being located in memory when the compiler generates the code for an internal function.
Also, methods do not necessarily have to be public if they are only called within the contract-in such case they should be marked internal.

Affected Smart Contract Functions:

FarmingFactory:
deploy,notifyRewardAmounts,notifyRewardAmount

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider as much as possible declaring external variables instead of public variables. As for best practice, you should use external if you expect that the function will only be called externally and use public if you need to call the function internally. To sum up, all can access to public functions, external functions only can be accessed externally and internal functions can only be called within the contract.

Remediation Plan:

SOLVED: EasyFi team changed the visibility of deploy() and notifyRewardAmounts() function from public to external.

3.4 (HAL-04) USE OF BLOCK.TIMESTAMP - INFORMATIONAL

Description:

In the **Farming Contracts** repository, The contracts are using block.timestamp. The global variable block.timestamp does not necessarily hold the current time, and may not be accurate. Miners can influence the value of block.timestamp to perform Maximal Extractable Value (MEV) attacks. There is no guarantee that the value is correct, only that it is higher than the previous block's timestamp.

Code Location:

FarmingRewards.sol Line #645

Listing 6: FarmingRewards.sol (Lines)

```
645     function lastTimeRewardApplicable() public view override
            returns (uint256) {
646         return Math.min(block.timestamp, periodFinish);
647     }
```

FarmingRewards.sol Line #745

Listing 7: FarmingRewards.sol (Lines)

```
745     function notifyRewardAmount(uint256 reward)
746         external
747         override
748         onlyRewardsDistribution
749         updateReward(address(0))
750     {
751         if (block.timestamp >= periodFinish) {
752             rewardRate = reward.div(rewardsDuration);
753         } else {
754             uint256 remaining = periodFinish.sub(block.timestamp);
755             uint256 leftover = remaining.mul(rewardRate);
756             rewardRate = reward.add(leftover).div(rewardsDuration)
```

```
                 ;
        }

        uint256 balance = rewardsToken.balanceOf(address(this));
        require(
            rewardRate <= balance.div(rewardsDuration),
            "Provided reward too high"
        );

        lastUpdateTime = block.timestamp;
        periodFinish = block.timestamp.add(rewardsDuration);
        emit RewardAdded(reward);
    }
```

FarmingFactory.sol Line #824

Listing 8: FarmingFactory.sol (Lines)

```
824     constructor(address _rewardsToken, uint256
            _farmingRewardsGenesis)
825         Ownable()
826     {
827         require(
828             _farmingRewardsGenesis >= block.timestamp,
829             "FarmingFactory::constructor: genesis too soon"
830         );
831
832         rewardsToken = _rewardsToken;
833         farmingRewardsGenesis = _farmingRewardsGenesis;
834     }
```

FarmingFactory.sol Line #881

Listing 9: FarmingFactory.sol (Lines)

```
881     function notifyRewardAmount(address farmingToken) public {
882         require(
883             block.timestamp >= farmingRewardsGenesis,
884             "FarmingFactory::notifyRewardAmount: not ready"
885         );
886
887         FarmingRewardsInfo storage info =
```

```
            farmingRewardsInfoByFarmingToken[
                farmingToken
        ];
        require(
            info.farmingRewards != address(0),
            "FarmingFactory::notifyRewardAmount: not deployed"
        );

        if (info.rewardAmount > 0) {
            uint256 rewardAmount = info.rewardAmount;
            info.rewardAmount = 0;

            require(
                IERC20(rewardsToken).transfer(
                    info.farmingRewards,
                    rewardAmount
                ),
                "FarmingFactory::notifyRewardAmount: transfer
                    failed"
            );
            FarmingRewards(info.farmingRewards).notifyRewardAmount
                (
                rewardAmount
            );
        }
    }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Use block.number instead of block.timestamp or now to reduce the risk of MEV attacks. Check if the timescale of the project occurs across years, days and months rather than seconds.

Remediation Plan:

NOT APPLICABLE: EasyFi team claims that the use of block.timestamp is deliberated. In addition, the timescale in farming contract is higher than 900 seconds.

3.5 (HAL-05) LACK OF REWARD DURATION SETTER FUNCTION - INFORMATIONAL

Description:

In the **FarmingRewards** contract, rewards duration have been set at the constructor. However, rewards duration can not be changed after the deploy. The requirements should be reviewed by **EasyFi Team**. If they need change **rewardsDuration** after the period (**periodFinish**) finished they should implement functions via onlyRewardsDistribution role.

Code Location:

FarmingRewards.sol Line #645

Listing 10: FarmingRewards.sol (Lines 627)

```
618     constructor(
619         address _rewardsDistribution,
620         address _rewardsToken,
621         address _farmingToken,
622         uint256 _rewardsDuration
623     ) {
624         rewardsToken = IERC20(_rewardsToken);
625         farmingToken = IERC20(_farmingToken);
626         rewardsDistribution = _rewardsDistribution;
627         rewardsDuration = _rewardsDuration;
628     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Review the requirements of the farming contract and If the setter function is required, the function should be implemented.

Remediation Plan:

ACKNOWLEDGED: EasyFi team claims that the use is deliberated because the duration will not be changed after the deployment.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. 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. 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.

Results:

```
INFO:Detectors:
FarmingRewards.notifyRewardAmount(uint256) (contracts/farmingFactory.sol#745-772) performs a multiplication on the result of a division:
        -rewardRate = reward.div(rewardsDuration) (contracts/farmingFactory.sol#752)
        -leftover = remaining.mul(rewardRate) (contracts/farmingFactory.sol#755)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Reentrancy in FarmingRewards.exit() (contracts/farmingFactory.sol#738-741):
        External calls:
        - withdraw(_balances[msg.sender]) (contracts/farmingFactory.sol#739)
                - farmingToken.safeTransfer(msg.sender,amount) (contracts/farmingFactory.sol#725)
                - (success,returndata) = address(token).call(data) (contracts/farmingFactory.sol#498)
        - getReward() (contracts/farmingFactory.sol#740)
                - (success,returndata) = address(token).call(data) (contracts/farmingFactory.sol#498)
                - rewardsToken.safeTransfer(msg.sender,reward) (contracts/farmingFactory.sol#733)
        State variables written after the call(s):
        - getReward() (contracts/farmingFactory.sol#740)
                - _guardCounter += 1 (contracts/farmingFactory.sol#542)
        - getReward() (contracts/farmingFactory.sol#740)
                - lastUpdateTime = lastTimeRewardApplicable() (contracts/farmingFactory.sol#778)
        - getReward() (contracts/farmingFactory.sol#740)
                - rewardPerTokenStored = rewardPerToken() (contracts/farmingFactory.sol#777)
        - getReward() (contracts/farmingFactory.sol#740)
                - rewards[msg.sender] = 0 (contracts/farmingFactory.sol#732)
                - rewards[account] = earned(account) (contracts/farmingFactory.sol#780)
        - getReward() (contracts/farmingFactory.sol#740)
                - userRewardPerTokenPaid[account] = rewardPerTokenStored (contracts/farmingFactory.sol#781)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
FarmingRewards.constructor(address,address,address,uint256)._rewardsDistribution (contracts/farmingFactory.sol#619) lacks a zero-check on :
                - rewardsDistribution = _rewardsDistribution (contracts/farmingFactory.sol#626)
FarmingFactory.constructor(address,uint256)._rewardsToken (contracts/farmingFactory.sol#824) lacks a zero-check on :
                - rewardsToken = _rewardsToken (contracts/farmingFactory.sol#832)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

AUTOMATED TESTING

```
INFO:Detectors:
Reentrancy in FarmingRewards.exit() (contracts/farmingFactory.sol#738-741):
        External calls:
        - withdraw(_balances[msg.sender]) (contracts/farmingFactory.sol#739)
                - farmingToken.safeTransfer(msg.sender,amount) (contracts/farmingFactory.sol#725)
                - (success,returndata) = address(token).call(data) (contracts/farmingFactory.sol#498)
        - getReward() (contracts/farmingFactory.sol#740)
                - (success,returndata) = address(token).call(data) (contracts/farmingFactory.sol#498)
                - rewardsToken.safeTransfer(msg.sender,reward) (contracts/farmingFactory.sol#733)
        Event emitted after the call(s):
        - RewardPaid(msg.sender,reward) (contracts/farmingFactory.sol#734)
                - getReward() (contracts/farmingFactory.sol#740)
Reentrancy in FarmingRewards.farm(uint256) (contracts/farmingFactory.sol#703-714):
        External calls:
        - farmingToken.safeTransferFrom(msg.sender,address(this),amount) (contracts/farmingFactory.sol#712)
        Event emitted after the call(s):
        - Farmed(msg.sender,amount) (contracts/farmingFactory.sol#713)
Reentrancy in FarmingRewards.farmWithPermit(uint256,uint256,uint8,bytes32,bytes32) (contracts/farmingFactory.sol#677-701):
        External calls:
        - IUniswapV2ERC20(address(farmingToken)).permit(msg.sender,address(this),amount,deadline,v,r,s) (contracts/farmingFactory.sol#689-697)
        - farmingToken.safeTransferFrom(msg.sender,address(this),amount) (contracts/farmingFactory.sol#699)
        Event emitted after the call(s):
        - Farmed(msg.sender,amount) (contracts/farmingFactory.sol#700)
Reentrancy in FarmingRewards.getReward() (contracts/farmingFactory.sol#729-736):
        External calls:
        - rewardsToken.safeTransfer(msg.sender,reward) (contracts/farmingFactory.sol#733)
        Event emitted after the call(s):
        - RewardPaid(msg.sender,reward) (contracts/farmingFactory.sol#734)
Reentrancy in FarmingRewards.withdraw(uint256) (contracts/farmingFactory.sol#716-727):
        External calls:
        - farmingToken.safeTransfer(msg.sender,amount) (contracts/farmingFactory.sol#725)
        Event emitted after the call(s):
        - Withdrawn(msg.sender,amount) (contracts/farmingFactory.sol#726)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
FarmingRewards.getReward() (contracts/farmingFactory.sol#729-736) uses timestamp for comparisons
        Dangerous comparisons:
        - reward > 0 (contracts/farmingFactory.sol#731)
FarmingRewards.notifyRewardAmount(uint256) (contracts/farmingFactory.sol#745-772) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp >= periodFinish (contracts/farmingFactory.sol#751)
        - require(bool,string)(rewardRate <= balance.div(rewardsDuration),Provided reward too high) (contracts/farmingFactory.sol#764-767)
FarmingFactory.constructor(address,uint256) (contracts/farmingFactory.sol#824-834) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_farmingRewardsGenesis >= block.timestamp,FarmingFactory::constructor: genesis too soon) (contracts/farmingFactory.sol#827-830)
FarmingFactory.notifyRewardAmount(address) (contracts/farmingFactory.sol#881-910) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= farmingRewardsGenesis,FarmingFactory::notifyRewardAmount: not ready) (contracts/farmingFactory.sol#882-885)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (contracts/farmingFactory.sol#375-386) uses assembly
        - INLINE ASM (contracts/farmingFactory.sol#382-384)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

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 fruit 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 the testers machine and sent the compiled results to the analyzers to locate any vulnerabilities. Only security-related findings are shown below.

Results:

No issues were found by MythX.

THANK YOU FOR CHOOSING

HALBORN