AMM, DMM & Farm

Smart Contract Audit Report
Prepared for EvryNet

Date Issued:	Nov 9, 2021
Project ID:	AUDIT2021018-1
Version:	v2.0
Confidentiality Level:	Public

Report Information

Project ID	AUDIT2021018-1
Version	v2.0
Client	EvryNet
Project	AMM, DMM & Farm
Auditor(s)	Weerawat Pawanawiwat Suvicha Buakhom Patipon Suwanbol Peeraphut Punsuwan
Author	Peeraphut Punsuwan Patipon Suwanbol
Reviewer	Suvicha Buakhom
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
2.0	Nov 9, 2021	Update issue id	Peeraphut Punsuwan
1.0	Oct 29, 2021	Full report	Peeraphut Punsuwan Patipon Suwanbol

Contact Information

Company	Inspex
Phone	(+66) 90 888 7186
Telegram	t.me/inspexco
Email	audit@inspex.co

Table of Contents

1. Executive Summary

As requested by EvryNet, Inspex team conducted an audit to verify the security posture of the AMM,DMM & Farm smart contracts between Sep 20, 2021 and Sep 30, 2021. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of AMM,DMM & Farm smart contracts. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify smart contract vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem. Practical recommendations are provided according to each vulnerability found and should be followed to remediate the issue.

1.1. Audit Result

In the initial audit, Inspex found 4 high, 4 medium, 1 low, 4 very low, and 5 info-severity issues. With the mitigation solutions and fixes confirmed by the project team, while 1 low and 2 very low issues were acknowledged by the team. Therefore, Inspex trusts that AMM,DMM & Farm smart contracts have sufficient protections to be safe for public use. However, in the long run, Inspex suggests resolving all issues found in this report.

1.2. Disclaimer

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), Inspex suggests conducting multiple independent assessments to minimize the risks. Lastly, nothing contained in this audit report should be considered as investment advice.

2. Project Overview

2.1. Project Introduction

EvryNet is an intelligent financial services platform providing infrastructure that enables developers and businesses to build an unlimited number of Centralised/Decentralised Finance (CeDeFi) applications, interoperable with many of the world's leading blockchains for "evryone".

Evry Finance is made up of two components, Farm and Swap. Farm is the main feature responsible for distributing $EVRY reward on the platform. The users can deposit tokens to the pools added in the farm and earn $EVRY as a reward. While Swap is divided into two parts, Automated Market Maker (AMM) and Dynamic Market Maker (DMM). The users can perform ERC20 token swapping easily with the liquidity pool of the platform. Users can also provide liquidity to the pools and gain a part of the swapping fee as an incentive.

Scope Information:

Project Name	AMM,DMM & Farm
Website	https://evrynet.io/
Smart Contract Type	Ethereum Smart Contract
Chain	Ethereum, Binance Smart Chain
Programming Language	Solidity

Audit Information:

Audit Method	Whitebox
Audit Date	Sep 20, 2021 - Sep 30, 2021
Reassessment Date	Oct 14, 2021

The audit method can be categorized into two types depending on the assessment targets provided:

1. **Whitebox**: The complete source code of the smart contracts are provided for the assessment.
2. **Blackbox**: Only the bytecodes of the smart contracts are provided for the assessment.

2.2. Scope

The following smart contracts were audited and reassessed by Inspex in detail:

2.2.1. AMM Swap

AMM Swap Initial Audit: (Commit: 35ce66036b7f5c6a2f54d5266a00e2591394979c)

Contract	Location (URL)
EvryERC20	https://github.com/Evry-Finance/evry-finance-amm-swap/blob/35ce66036b/contracts/EvryERC20.sol
EvryFactory	https://github.com/Evry-Finance/evry-finance-amm-swap/blob/35ce66036b/contracts/EvryFactory.sol
EvryPair	https://github.com/Evry-Finance/evry-finance-amm-swap/blob/35ce66036b/contracts/EvryPair.sol
EvryRouter	https://github.com/Evry-Finance/evry-finance-amm-swap/blob/35ce66036b/contracts/EvryRouter.sol

AMM Swap Reassessment: (Commit: a4a2322a557783f5f3326984999c6ac619d19ebd)

Contract	Location (URL)
EvryERC20	https://github.com/Evry-Finance/evry-finance-amm-swap/blob/a4a2322a55/contracts/EvryERC20.sol
EvryFactory	https://github.com/Evry-Finance/evry-finance-amm-swap/blob/a4a2322a55/contracts/EvryFactory.sol
EvryPair	https://github.com/Evry-Finance/evry-finance-amm-swap/blob/a4a2322a55/contracts/EvryPair.sol
EvryRouter	https://github.com/Evry-Finance/evry-finance-amm-swap/blob/a4a2322a55/contracts/EvryRouter.sol

2.2.2. DMM Swap

DMM Swap Initial Audit: (Commit: 144843b7db62e6fc1cb764ade4ab02af08c8450d)

Contract	Location (URL)
DMMFactory	https://github.com/Evry-Finance/evry-finance-dmm-swap/blob/144843b7db/contracts/DMMFactory.sol
DMMFactoryDelegate	https://github.com/Evry-Finance/evry-finance-dmm-swap/blob/144843b7db/contracts/DMMFactoryDelegate.sol
DMMPool	https://github.com/Evry-Finance/evry-finance-dmm-swap/blob/144843b7d

	b/contracts/DMMPool.sol
DMMRouter02	https://github.com/Evry-Finance/evry-finance-dmm-swap/blob/144843b7db/contracts/periphery/DMMRouter02.sol
DMMRouter02DelegateCall	https://github.com/Evry-Finance/evry-finance-dmm-swap/blob/144843b7db/contracts/periphery/DMMRouter02DelegateCall.sol
DaoRegistry	https://github.com/Evry-Finance/evry-finance-dmm-swap/blob/144843b7db/contracts/periphery/DaoRegistry.sol
ManageUser	https://github.com/Evry-Finance/evry-finance-dmm-swap/blob/144843b7db/contracts/ManageUser.sol
ManageUserAddress	https://github.com/Evry-Finance/evry-finance-dmm-swap/blob/144843b7db/contracts/ManageUserAddress.sol
VolumeTrendRecorder	https://github.com/Evry-Finance/evry-finance-dmm-swap/blob/144843b7db/contracts/VolumeTrendRecorder.sol

DMM Swap Reassessment: (Commit: 144843b7db62e6fc1cb764ade4ab02af08c8450d)

Contract	Location (URL)
DMMFactory	https://github.com/Evry-Finance/evry-finance-dmm-swap/blob/144843b7db/contracts/DMMFactory.sol
DMMFactoryDelegate	https://github.com/Evry-Finance/evry-finance-dmm-swap/blob/144843b7db/contracts/DMMFactoryDelegate.sol
DMMPool	https://github.com/Evry-Finance/evry-finance-dmm-swap/blob/144843b7db/contracts/DMMPool.sol
DMMRouter02	https://github.com/Evry-Finance/evry-finance-dmm-swap/blob/144843b7db/contracts/periphery/DMMRouter02.sol
DMMRouter02DelegateCall	https://github.com/Evry-Finance/evry-finance-dmm-swap/blob/144843b7db/contracts/periphery/DMMRouter02DelegateCall.sol
DaoRegistry	https://github.com/Evry-Finance/evry-finance-dmm-swap/blob/144843b7db/contracts/periphery/DaoRegistry.sol
ManageUser	https://github.com/Evry-Finance/evry-finance-dmm-swap/blob/144843b7db/contracts/ManageUser.sol
ManageUserAddress	https://github.com/Evry-Finance/evry-finance-dmm-swap/blob/144843b7db/contracts/ManageUserAddress.sol
VolumeTrendRecorder	https://github.com/Evry-Finance/evry-finance-dmm-swap/blob/144843b7db/contracts/VolumeTrendRecorder.sol

2.2.3. Farm

Farm Initial Audit: (Commit: 2d194cdce2621ce08c863579bce8a51bc7bafcd)

Contract	Location (URL)
Farms	https://github.com/Evry-Finance/evry-finance-farm/blob/2d194cdbce/contracts/Farms.sol
EvryDistributor	https://github.com/Evry-Finance/evry-finance-farm/blob/2d194cdbce/contracts/EVRYDistributor.sol

Farm Reassessment: (Commit: 573a14e6eee92826aedfe440c3d94d0d9ed4a279)

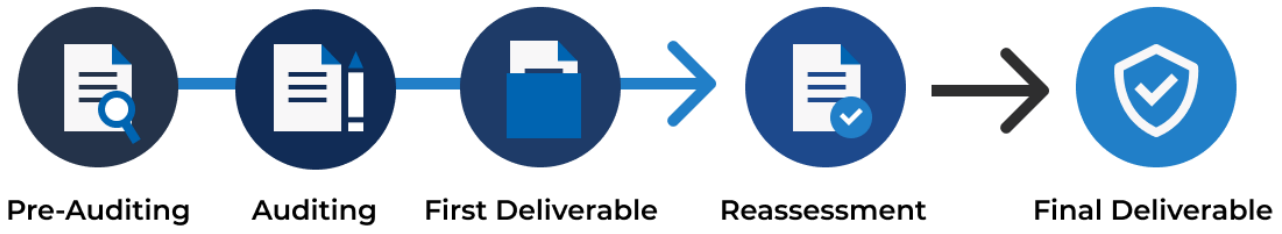
Contract	Location (URL)
Farms	https://github.com/Evry-Finance/evry-finance-farm/blob/573a14e6ee/contracts/Farms.sol
EvryDistributor	https://github.com/Evry-Finance/evry-finance-farm/blob/573a14e6ee/contracts/EVRYDistributor.sol

The assessment scope covers only the in-scope smart contracts and the smart contracts that they inherit from.

3. Methodology

Inspex conducts the following procedure to enhance the security level of our clients' smart contracts:

1. **Pre-Auditing**: Getting to understand the overall operations of the related smart contracts, checking for readiness, and preparing for the auditing
2. **Auditing**: Inspecting the smart contracts using automated analysis tools and manual analysis by a team of professionals
3. **First Deliverable and Consulting**: Delivering a preliminary report on the findings with suggestions on how to remediate those issues and providing consultation
4. **Reassessment**: Verifying the status of the issues and whether there are any other complications in the fixes applied
5. **Final Deliverable**: Providing a full report with the detailed status of each issue



3.1. Test Categories

Inspex smart contract auditing methodology consists of both automated testing with scanning tools and manual testing by experienced testers. We have categorized the tests into 3 categories as follows:

1. **General Smart Contract Vulnerability (General)** - Smart contracts are analyzed automatically using static code analysis tools for general smart contract coding bugs, which are then verified manually to remove all false positives generated.
2. **Advanced Smart Contract Vulnerability (Advanced)** - The workflow, logic, and the actual behavior of the smart contracts are manually analyzed in-depth to determine any flaws that can cause technical or business damage to the smart contracts or the users of the smart contracts.
3. **Smart Contract Best Practice (Best Practice)** - The code of smart contracts is then analyzed from the development perspective, providing suggestions to improve the overall code quality using standardized best practices.

3.2. Audit Items

The following audit items were checked during the auditing activity.

General
Reentrancy Attack
Integer Overflows and Underflows
Unchecked Return Values for Low-Level Calls
Bad Randomness
Transaction Ordering Dependence
Time Manipulation
Short Address Attack
Outdated Compiler Version
Use of Known Vulnerable Component
Deprecated Solidity Features
Use of Deprecated Component
Loop with High Gas Consumption
Unauthorized Self-destruct
Redundant Fallback Function
Insufficient Logging for Privileged Functions
Invoking of Unreliable Smart Contract
Advanced
Business Logic Flaw
Ownership Takeover
Broken Access Control
Broken Authentication
Use of Upgradable Contract Design
Improper Kill-Switch Mechanism

Improper Front-end Integration
Insecure Smart Contract Initiation
Denial of Service
Improper Oracle Usage
Memory Corruption
Best Practice
Use of Variadic Byte Array
Implicit Compiler Version
Implicit Visibility Level
Implicit Type Inference
Function Declaration Inconsistency
Token API Violation
Best Practices Violation

3.3. Risk Rating

OWASP Risk Rating Methodology[1] is used to determine the severity of each issue with the following criteria:

- **Likelihood**: a measure of how likely this vulnerability is to be uncovered and exploited by an attacker.
- **Impact**: a measure of the damage caused by a successful attack

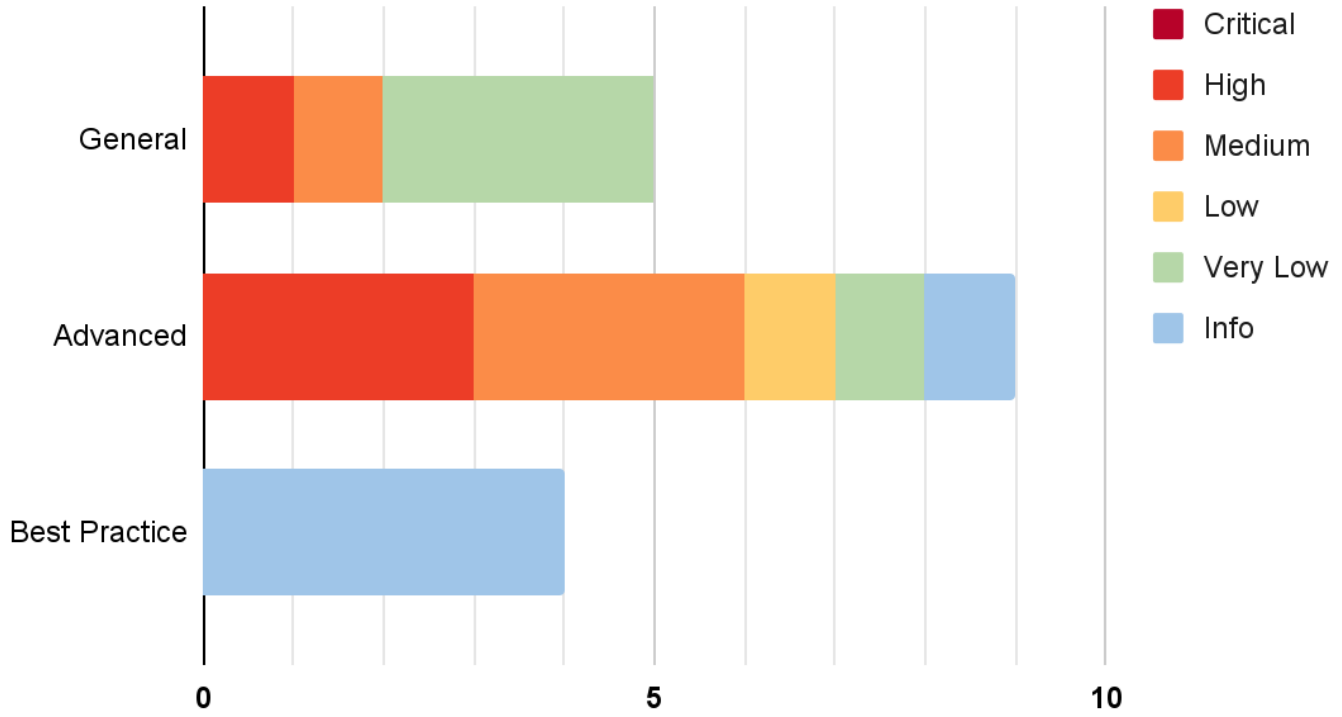
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would be categorized as **Info**.

Impact \ Likelihood	Low	Medium	High
Low	**Very Low**	**Low**	**Medium**
Medium	**Low**	**Medium**	**High**
High	**Medium**	**High**	**Critical**

4. Summary of Findings

From the assessments, Inspex has found 18 issues in three categories. The following chart shows the number of the issues categorized into three categories: **General**, **Advanced**, and **Best Practice**.



The statuses of the issues are defined as follows:

Status	Description
Resolved	The issue has been resolved and has no further complications.
Resolved *	The issue has been resolved with mitigations and clarifications. For the clarification or mitigation detail, please refer to Chapter 5.
Acknowledged	The issue's risk has been acknowledged and accepted.
No Security Impact	The best practice recommendation has been acknowledged.

The information and status of each issue can be found in the following table:

ID	Scope	Title	Category	Severity	Status
IDX-001	AMM Swap	Design Flaw in Fee Calculation	Advanced	High	Resolved
IDX-002	AMM Swap	Centralized Control of State Variable	General	Medium	Resolved *
IDX-003	AMM Swap	Unchecked Value for Fees Setting	Advanced	Low	Acknowledged
IDX-004	AMM Swap	Insufficient Logging for Privileged Functions	General	Very Low	Resolved
IDX-005	AMM Swap	Inexplicit Solidity Compiler Version	Best Practice	Info	No Security Impact
IDX-006	DMM Swap	Centralized Control of State Variable	General	Medium	Resolved *
IDX-007	DMM Swap	Insufficient Logging for Privileged Functions	General	Very Low	Acknowledged
IDX-008	DMM Swap	Improper Platform Fee Condition	Advanced	Very Low	Acknowledged
IDX-009	DMM Swap	Improper Function Visibility	Best Practice	Info	No Security Impact
IDX-010	Farm	Use of Upgradable Contract	Advanced	High	Resolved *
IDX-011	Farm	Denial of Service in Beneficiary Mechanism	Advanced	High	Resolved
IDX-012	Farm	Improper Token Release Mechanism	Advanced	High	Resolved
IDX-013	Farm	Reward Miscalculation	Advanced	Medium	Resolved *
IDX-014	Farm	Centralized Control of State Variable	General	Medium	Resolved *
IDX-015	Farm	Insufficient Logging for Privileged Functions	General	Very Low	Resolved
IDX-016	Farm	Unsupported Design for Deflationary Token	Advanced	Info	Resolved
IDX-017	Farm	Availability of Emergency Withdraw Function	Best Practice	Info	No Security Impact
IDX-018	Farm	Improper Function Visibility	Best Practice	Info	No Security Impact

* The mitigations or clarifications by EvryNet can be found in Chapter 5.

5. Detailed Findings Information

5.1. AMM Swap - Design Flaw in Fee Calculation

ID	IDX-001
Target	EvryPair
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** High **Impact:** Medium The fees of the platform can be bypassed by calling the `swap()` function through a custom contract. This causes loss of profit to the platform and reduces the reward of the liquidity providers, resulting in loss of reputation for the platform. **Likelihood:** High This flaw can be used by any attacker with the knowledge in writing custom smart contract to avoid fees, resulting in high motivation for the attack.
Status	Resolved EvryNet team has resolved this issue as suggested in commit 26f055cd853f363c356e084aca8e7350b394abec by removing the fee parameters from the `swap()` function.

5.1.1. Description

The `swap()` function allows the caller to swap between 2 tokens (`token0` and `token1`) in `EvryPair` contract.

According to the platform's business design, the `EvryRouter` contract is used to send tokens to the `EvryPair` contract and call the `swap()` function with fee parameters to collect the fees (the platform's fee and the liquidity's fee) when the users want to swap tokens.

However, since the `swap()` function's visibility is `external` without any other access control, anyone can execute this function and freely control the input parameters.

EvryPair.sol

```
139 function swap(
140         uint[2] memory amountOut,
141         address to,
142         address feeToPlatform,
143         uint feePlatformBasis,
144         uint feeLiquidityBasis,
145         bytes calldata data
146     )
```

```
147         external
148         lock
149         override
150     {
151
152     FeeConfiguration memory feeConfiguration = FeeConfiguration({
153         feeToPlatform: feeToPlatform,
154         feePlatformBasis: feePlatformBasis,
155         feeLiquidityBasis: feeLiquidityBasis,
156         amount0Out: amountOut[0],
157         amount1Out: amountOut[1]
158     });
159
160     require(feeConfiguration.amount0Out > 0 || feeConfiguration.amount1Out > 0,
    'Evry: INSUFFICIENT_OUTPUT_AMOUNT');
161
162     (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
163     require(feeConfiguration.amount0Out < _reserve0 &&
    feeConfiguration.amount1Out < _reserve1, 'Evry: INSUFFICIENT_LIQUIDITY');
164
165     uint balance0;
166     uint balance1;
167     { // scope for _token{0,1}, avoids stack too deep errors
168         address _token0 = token0;
169         address _token1 = token1;
170         require(to != _token0 && to != _token1, 'Evry: INVALID_TO');
171         if (feeConfiguration.amount0Out > 0) _safeTransfer(_token0, to,
    feeConfiguration.amount0Out); // optimistically transfer tokens
172         if (feeConfiguration.amount1Out > 0) _safeTransfer(_token1, to,
    feeConfiguration.amount1Out); // optimistically transfer tokens
173         if (data.length > 0) IEvryCallee(to).evryCall(msg.sender,
    feeConfiguration.amount0Out, feeConfiguration.amount1Out, data);
174         balance0 = IERC20(_token0).balanceOf(address(this));
175         balance1 = IERC20(_token1).balanceOf(address(this));
176     }
177     uint amount0In = balance0 > _reserve0 - feeConfiguration.amount0Out ?
    balance0 - (_reserve0 - feeConfiguration.amount0Out) : 0;
178     uint amount1In = balance1 > _reserve1 - feeConfiguration.amount1Out ?
    balance1 - (_reserve1 - feeConfiguration.amount1Out) : 0;
179     require(amount0In > 0 || amount1In > 0, 'Evry: INSUFFICIENT_INPUT_AMOUNT');
180
181     { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
182         uint totalFee = feeConfiguration.feePlatformBasis
    .add(feeConfiguration.feeLiquidityBasis);
183         uint balance0Adjusted = balance0.mul(10000)
    .sub(amount0In.mul(totalFee));
184         uint balance1Adjusted = balance1.mul(10000)
```

```
.sub(amount1In.mul(totalFee));
185         require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0)
.mul(_reserve1).mul(10000**2), 'Evry: K');
186     }
187
188     _update(balance0, balance1);
189     {
190     // emit Swap(msg.sender, amount0In, amount1In, amountOut, to);
191 }
192     if (amount0In > 0) {
193         sendFeeToPlatform(token0, amount0In, feeConfiguration.feePlatformBasis,
feeConfiguration.feeToPlatform);
194     } else {
195         sendFeeToPlatform(token1, amount1In, feeConfiguration.feePlatformBasis,
feeConfiguration.feeToPlatform);
196     }
197     _sync();
198
199 }
```

As a result, anyone can create a contract to bypass the swapping fee by transferring the input token to the `EvryPair` contract address and executing the `swap()` function with `feePlatformBasis` and `feeLiquidityBasis` set to 0, allowing the attacker to swap the token through the pair contract without paying any fee.

5.1.2. Remediation

Inspex suggests removing the input parameters for the fees and get the fees configurations from the `EvryFactory` contract, for example:

EvryPair.sol

```
139 function swap(
140         uint[2] memory amountOut,
141         address to,
142         bytes calldata data
143     )
144         external
145         lock
146         override
147     {
148     (uint256 feeToPlatform, uint256 feePlatformBasis, uint256
feeLiquidityBasis) = IEvryFactory(factory).getFeeConfiguration();
149     FeeConfiguration memory feeConfiguration = FeeConfiguration({
150         feeToPlatform: feeToPlatform,
151         feePlatformBasis: feePlatformBasis,
152         feeLiquidityBasis: feeLiquidityBasis,
```

```
153             amount0Out: amountOut[0],
154             amount1Out: amountOut[1]
155         });
156 
157         require(feeConfiguration.amount0Out > 0 || feeConfiguration.amount1Out > 0,
    'Evry: INSUFFICIENT_OUTPUT_AMOUNT');
158 
159         (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
160         require(feeConfiguration.amount0Out < _reserve0 &&
    feeConfiguration.amount1Out < _reserve1, 'Evry: INSUFFICIENT_LIQUIDITY');
161 
162         uint balance0;
163         uint balance1;
164         { // scope for _token{0,1}, avoids stack too deep errors
165             address _token0 = token0;
166             address _token1 = token1;
167             require(to != _token0 && to != _token1, 'Evry: INVALID_TO');
168             if (feeConfiguration.amount0Out > 0) _safeTransfer(_token0, to,
    feeConfiguration.amount0Out); // optimistically transfer tokens
169             if (feeConfiguration.amount1Out > 0) _safeTransfer(_token1, to,
    feeConfiguration.amount1Out); // optimistically transfer tokens
170             if (data.length > 0) IEvryCallee(to).evryCall(msg.sender,
    feeConfiguration.amount0Out, feeConfiguration.amount1Out, data);
171             balance0 = IERC20(_token0).balanceOf(address(this));
172             balance1 = IERC20(_token1).balanceOf(address(this));
173         }
174         uint amount0In = balance0 > _reserve0 - feeConfiguration.amount0Out ?
    balance0 - (_reserve0 - feeConfiguration.amount0Out) : 0;
175         uint amount1In = balance1 > _reserve1 - feeConfiguration.amount1Out ?
    balance1 - (_reserve1 - feeConfiguration.amount1Out) : 0;
176         require(amount0In > 0 || amount1In > 0, 'Evry: INSUFFICIENT_INPUT_AMOUNT');
177 
178         { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
179             uint totalFee =
    feeConfiguration.feePlatformBasis.add(feeConfiguration.feeLiquidityBasis);
180             uint balance0Adjusted = balance0.mul(10000)
    .sub(amount0In.mul(totalFee));
181             uint balance1Adjusted = balance1.mul(10000)
    .sub(amount1In.mul(totalFee));
182             require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0)
    .mul(_reserve1).mul(10000**2), 'Evry: K');
183         }
184 
185         _update(balance0, balance1);
186         {
187         // emit Swap(msg.sender, amount0In, amount1In, amountOut, to);
188         }
```

```
189     if (amount0In > 0) {
190         sendFeeToPlatform(token0, amount0In, feeConfiguration.feePlatformBasis,
    feeConfiguration.feeToPlatform);
191     } else {
192         sendFeeToPlatform(token1, amount1In, feeConfiguration.feePlatformBasis,
    feeConfiguration.feeToPlatform);
193     }
194     _sync();
195
196 }
```

Please note that all other contracts that call the `swap()` function should be adjusted accordingly.

5.2. AMM Swap - Centralized Control of State Variable

ID	IDX-002
Target	EvryFactory
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** Medium **Impact:** Medium The controlling authorities can change the critical state variables to gain additional profit. Thus, it is unfair to the other users. **Likelihood:** Medium There is nothing to restrict the changes from being done; however, these actions can only be performed by the contract owner.
Status	Resolved * EvryNet team has confirmed that timelock will be used to own the contracts. This will allow the users to monitor the changes to the smart contracts and act accordingly before the changes are applied. However, the timelock was not in use at the time of the reassessment. Therefore, Inspex suggests the platform users to confirm the usage of the timelock before using the platform.

5.2.1. Description

Critical state variables can be updated any time by the controlling authorities. Changes in these variables can cause impacts to the users, so the users should accept or be notified before these changes are effective.

However, there is currently no constraint to prevent the authorities from modifying these variables without notifying the users.

The controllable privileged state update functions are as follows:

File	Contract	Function	Modifier
EvryFactory.sol (L:64)	EvryFactory	setFeeToPlatform()	onlyOwner
EvryFactory.sol (L:68)	EvryFactory	setPlatformFee()	onlyOwner
EvryFactory.sol (L:72)	EvryFactory	setLiquidityFee()	onlyOwner
@openzeppelin/contracts/access/Ownable.sol (L:54)	Ownable	renounceOwnership()	onlyOwner

@openzeppelin/contracts/access/Ownable.sol (L:63)	Ownable	transferOwnership()	onlyOwner

Please note that the `Ownable` contract is inherited from the OpenZeppelin library.

5.2.2. Remediation

In the ideal case, the critical state variables should not be modifiable to keep the integrity of the smart contract. However, if modifications are needed, Inspex suggests limiting the use of these functions via the following options:

- Implementing a community-run governance to control the use of these functions
- Using a Timelock contract to delay the changes for a sufficient amount of time, e.g., 24 hours

5.3. AMM Swap - Unchecked Value for Fees Setting

ID	IDX-003
Target	EvryFactory
Category	Advanced Smart Contract Vulnerability
CWE	CWE-755: Improper Handling of Exceptional Conditions
Risk	**Severity:** Low **Impact:** Medium The swap transactions will be reverted, resulting in denial of service on the `EvryRouter` contract and loss of reputation for the platform. **Likelihood:** Low It is unlikely that the contract owner will set the total fee (platform fee and liquidity fee) to be over 10,000 basis points (100 %). This is because it is unprofitable for the platform, so there is low motivation in doing this.
Status	**Acknowledged** EvryNet team has acknowledged this issue and decided not to fix it in this release.

5.3.1. Description

The `EvryRouter` contract allows the users to swap tokens through the token pairs. For example, when the user swaps tokens through the `swapExactTokensForTokens()` function, the `EvryRouter` contract will calculate the token output amount (the amount of token that the user will receive after a successful swap) given the swap token input amount (`amountIn`) from the user as in line 237 by calling the `getAmountsOut()` function in the `EvryLibrary` library.

EvryRouter.sol

```
230 function swapExactTokensForTokens(
231     uint amountIn,
232     uint amountOutMin,
233     address[] calldata path,
234     address to,
235     uint deadline
236 ) external virtual override ensure(deadline) returns (uint[] memory amounts) {
237     amounts = EvryLibrary.getAmountsOut(factory, amountIn, path);
238     require(amounts[amounts.length - 1] >= amountOutMin, 'EvryRouter:
    INSUFFICIENT_OUTPUT_AMOUNT');
239     TransferHelper.safeTransferFrom(
240         path[0], msg.sender, EvryLibrary.pairFor(factory, path[0], path[1]),
    amounts[0]
241     );
```

```
242         _swap(amounts, path, to);
243     }
```

The `getAmountsOut()` function will calculate the token output amount, applying the fee of the platform in line number 108 and 111.

EvryLibrary.sol

```
103 // performs chained getAmountOut calculations on any number of pairs
104 function getAmountsOut(address factory, uint amountIn, address[] memory path)
    internal view returns (uint[] memory amounts) {
105     require(path.length >= 2, 'EvryLibrary: INVALID_PATH');
106     amounts = new uint[](path.length);
107     amounts[0] = amountIn;
108     uint feeAmount = getBasisTotalFee(factory);
109     for (uint i; i < path.length - 1; i++) {
110         (uint reserveIn, uint reserveOut) = getReserves(factory, path[i],
    path[i + 1]);
111         amounts[i + 1] = getAmountOut(amounts[i], reserveIn, reserveOut,
    feeAmount);
112     }
113 }
```

The `getBasisTotalFee()` function allows the `EvryLibrary` library to retrieve the total fee by summing up the platform fee and the liquidity fee.

EvryLibrary.sol

```
58  function getBasisTotalFee(address factory)
59      internal
60      view
61      returns (uint256 totalFee)
62  {
63      uint256 platformFee = IEvryFactory(factory).feePlatformBasis();
64      uint256 protocalFee = IEvryFactory(factory).feeLiquidityBasis();
65      totalFee = platformFee.add(protocalFee);
66  }
```

The contract owner of the `EvryFactory` contract can set the platform fee and the liquidity fee at any time with any value using the `setPlatformFee()` and `setLiquidityFee()` functions.

EvryFactory.sol

```
68  function setPlatformFee(uint256 feeBasis) external onlyOwner override {
69      feePlatformBasis = feeBasis;
70  }
71
72  function setLiquidityFee(uint256 feeBasis) external onlyOwner override {
```

```
73      feeLiquidityBasis = feeBasis;
74  }
```

During the calculation of token out amount, the `amountOut` value relies on the `feeMultiplier` which is affected from subtracting the `feeAmount` from 10,000.

```
83  function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut, uint
    feeAmount) internal pure returns (uint amountOut) {
84      require(amountIn > 0, 'EvryLibrary: INSUFFICIENT_INPUT_AMOUNT');
85      require(reserveIn > 0 && reserveOut > 0, 'EvryLibrary:
    INSUFFICIENT_LIQUIDITY');
86      uint256 feeMuiltiplier = 10000 - feeAmount;
87      uint amountInWithFee = amountIn.mul(feeMuiltiplier);
88      uint numerator = amountInWithFee.mul(reserveOut);
89      uint denominator = reserveIn.mul(10000).add(amountInWithFee);
90      amountOut = numerator / denominator;
91  }
```

Hence, if the total fee (the platform's fee and the liquidity's fee) exceeds 10,000 basis points, the integer subtraction overflow will happen to the value of `feeMuiltiplier`, resulting in the value being incorrect and causes the transaction to be reverted during the amount calculation within the `Pair` contract.

5.3.2. Remediation

Inspex suggests setting the upper bound limit for the total fee (the platform fee and the liquidity fee) to prevent the total fee from exceeding 10,000 basis points.

EvryFactory.sol

```
68  function setPlatformFee(uint256 feeBasis) external onlyOwner override {
69      require(feeLiquidity.add(feeBasis) <= 10000, 'EvryFactory: total fee cannot
    exceed 10000');
70      feePlatformBasis = feeBasis;
71  }
72
73  function setLiquidityFee(uint256 feeBasis) external onlyOwner override {
74      require(feePlatformBasis.add(feeBasis) <= 10000, 'EvryFactory: total fee
    cannot exceed 10000');
75      feeLiquidityBasis = feeBasis;
76  }
```

5.4. AMM Swap - Insufficient Logging for Privileged Functions

ID	IDX-004
Target	EvryFactory
Category	General Smart Contract Vulnerability
CWE	CWE-778: Insufficient Logging
Risk	**Severity:** Very Low **Impact:** Low Privileged functions' executions cannot be monitored easily by the users. **Likelihood:** Low It is not likely that the execution of the privileged functions will be a malicious action.
Status	Resolved EvryNet team has resolved this issue as suggested in commit 26f055cd853f363c356e084aca8e7350b394abec by emitting events in privileged functions.

5.4.1. Description

Privileged functions that are executable by the controlling parties are not logged properly by emitting events. Without events, it is not easy for the public to monitor the execution of those privileged functions, allowing the controlling parties to perform actions that cause big impacts on the platform.

For example, the owner can set the platform's fee by executing the `setFeeToPlatform()` function in the `EvryFactory` contract, and no events are emitted.

EvryFactory.sol

```
64 function setFeeToPlatform(address _feeToPlatform) external onlyOwner override {
65     feeToPlatform = _feeToPlatform;
66 }
```

The following table contains all the privileged functions that do not emit the event:

File	Contract	Function	Modifier
EvryFactory.sol (L:64)	EvryFactory	setFeeToPlatform()	onlyOwner
EvryFactory.sol (L:68)	EvryFactory	setPlatformFee()	onlyOwner
EvryFactory.sol (L:72)	EvryFactory	setLiquidityFee()	onlyOwner
EvryFactory.sol (L:76)	EvryFactory	transferAdmin()	onlyAdmin

5.4.2. Remediation

Inspex suggests emitting events for the execution of privileged functions, for example:

EvryFactory.sol

```
63  event SetFeeToPlatform(address _feeToPlatform);
64  function setFeeToPlatform(address _feeToPlatform) external onlyOwner override {
65      feeToPlatform = _feeToPlatform;
66      emit SetFeeToPlatform(_feeToPlatform);
67  }
```

5.5. AMM Swap - Inexplicit Solidity Compiler Version

ID	IDX-005
Target	EvryERC20 EvryFactory EvryPair EvryRouter
Category	Smart Contract Best Practice
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	No Security Impact EvryNet team has acknowledged this best practice recommendation.

5.5.1. Description

The Solidity compiler versions declared in the smart contracts were not explicit. Each compilation may be done using different compiler versions, which may potentially result in compatibility issues, for example:

EvryERC20.sol

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.7.6;
```

The following table contains all targets which the inexplicit compiler version is declared.

Contract	Version
EvryERC20	^0.7.6
EvryFactory	^0.7.6
EvryPair	^0.7.6
EvryRouter	^0.7.6

5.5.2. Remediation

Inspex suggests fixing the solidity compiler to the latest stable version. At the time of the audit, the latest stable version of Solidity compiler in major 0.7 is v0.7.6.

For example:

EvryERC20.sol

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.7.6;
```

5.6. DMM Swap - Centralized Control of State Variable

ID	IDX-006
Target	DaoRegistry DMMFactoryDelegate
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** Medium **Impact:** Medium The controlling authorities can change the critical state variables to gain additional profit. Thus, it is unfair to the other users. **Likelihood:** Medium The functions can only be called by the authorized parties; however, there is nothing to restrict the changes from being done by them.
Status	Resolved * EvryNet team has confirmed that timelock will be used to own the contracts to mitigate this issue. This will allow the users to monitor the changes to the smart contracts and act accordingly before the changes are applied. However, the timelock was not in use at the time of the reassessment. Therefore, Inspex suggests the platform users to confirm the usage of the timelock before using the platform.

5.6.1. Description

Critical state variables can be updated any time by the controlling authorities. Changes in these variables can cause impacts to the users, so the users should accept or be notified before these changes are effective.

However, there is no constraint to prevent the authorities from modifying these variables without notifying the users.

The controllable privileged state update functions are as follows:

File	Contract	Function	Modifier
DaoRegister.sol (L:21)	DaoRegistry	addPool()	onlyOwner
DMMRouter02DelegateCall.sol (L:124)	DMMRouter02DelegateCall	addLiquidityNewPool()	_admin
DMMRouter02DelegateCall.sol (L:209)	DMMRouter02DelegateCall	addLiquidityNewPoolETH()	_superAdmin

DMMFactoryDelegate.sol (L:40)	DMMFactoryDelegate	setFeeConfiguration()	-
DMMFactoryDelegate.sol (L:51)	DMMFactoryDelegate	setFeeSetter()	-
ManageUserAddress.sol (L:21)	ManageUserAddress	addAdmin()	-
ManageUserAddress.sol (L:28)	ManageUserAddress	removeAdmin()	-
ManageUserAddress.sol (L:28)	ManageUserAddress	transferSuperAdmin()	-
@openzeppelin/contracts/access/Ownable.sol (L:54)	DaoRegistry	renounceOwnership()	onlyOwner
@openzeppelin/contracts/access/Ownable.sol (L:63)	DaoRegistry	transferOwnership()	onlyOwner

Please note that the `Ownable` contract is inherited from the OpenZeppelin library.

5.6.2. Remediation

In the ideal case, the critical state variables should not be modifiable to keep the integrity of the smart contract. However, if modifications are needed, Inspex suggests limiting the use of these functions via the following options:

- Implementing a community-run governance to control the use of these functions
- Using a `Timelock` contract to delay the changes for a sufficient amount of time, e.g., 24 hours

5.7. DMM Swap - Insufficient Logging for Privileged Functions

ID	IDX-007
Target	DMMRouter02DelegateCall ManageUserAddress
Category	General Smart Contract Vulnerability
CWE	CWE-778: Insufficient Logging
Risk	**Severity:** Very Low **Impact:** Low Privileged functions' executions cannot be monitored easily by the users. **Likelihood:** Low It is not likely that the execution of the privileged functions will be a malicious action.
Status	**Acknowledged** EvryNet team has acknowledged this issue and decided not to fix it in this release.

5.7.1. Description

Privileged functions that are executable by the controlling parties are not logged properly by emitting events. Without events, it is not easy for the public to monitor the execution of those privileged functions, allowing the controlling parties to perform actions that cause big impacts on the platform.

For example, the owner can add a new admin by executing the `addAdmin()` function in the `ManageUserAddress` contract, and no event is emitted.

ManageUserAddress.sol

```
21 function addAdmin(address user) public {
22     require(listUsers[msg.sender] == SUPER_ADMIN_ROLE, "NO PERMISSION");
23     bytes32 _role = getRole(user);
24     require(_role != SUPER_ADMIN_ROLE && _role != ADMIN_ROLE, "CAN NOT SET USER
25 TO ADMIN");
26     listUsers[user] = ADMIN_ROLE;
   }
```

The following table contains all the privileged functions that do not emit the event:

File	Contract	Function	Modifier / Role
DMMRouter02DelegateCall.sol (L:124)	DMMRouter02DelegateCall	addLiquidityNewPool()	_admin
ManageUserAddress.sol	ManageUserAddress	addAdmin()	SUPER_ADMIN_ROLE

(L:21)			
ManageUserAddress.sol (L:28)	ManageUserAddress	removeAdmin()	SUPER_ADMIN_ROLE
ManageUserAddress.sol (L:28)	ManageUserAddress	transferSuperAdmin()	SUPER_ADMIN_ROLE

5.7.2. Remediation

Inspex suggests emitting events for the execution of privileged functions, for example:

ManageUserAddress.sol

```
21 event AddAdmin(address user)
22 function addAdmin(address user) public {
23     require(listUsers[msg.sender] == SUPER_ADMIN_ROLE, "NO PERMISSION");
24     bytes32 _role = getRole(user);
25     require(_role != SUPER_ADMIN_ROLE && _role != ADMIN_ROLE, "CAN NOT SET USER
26 TO ADMIN");
27     listUsers[user] = ADMIN_ROLE;
28     emit AddAdmin(user);
   }
```

5.8. DMM Swap - Improper Platform Fee Condition

ID	IDX-008
Target	DMMPool
Category	Advanced Smart Contract Vulnerability
CWE	CWE-755: Improper Handling of Exceptional Conditions
Risk	**Severity:** Very Low **Impact:** Low The platform fee can be lower than what the platform intended, and that difference will be added to the part owned by the liquidity providers instead. This can cause a small amount of business impact to the platform. **Likelihood:** Low This attack requires the use of a custom smart contract, and only one side of token swapping is affected in each pool. Furthermore, the amount of fee is not high, and the attacker still has to pay the same amount of fee, resulting in low motivation for the attack.
Status	**Acknowledged** EvryNet team has acknowledged this issue and decided not to fix it in this release.

5.8.1. Description

The `EvryRouter02` contract allows users to swap tokens through the token pairs. Most of the external functions, which are available for the users to swap tokens, do the swapping action by forwarding all necessary parameters to the internal `_swap()` function, for example, the `swapExactTokensForTokens()`.

DMMRouter02.sol

```
128 function swapExactTokensForTokens(
129     uint256 amountIn,
130     uint256 amountOutMin,
131     address[] memory poolsPath,
132     IERC20[] memory path,
133     address to,
134     uint256 deadline
135 ) public virtual override ensure(deadline) returns (uint256[] memory amounts) {
136     verifyPoolsPathSwap(poolsPath, path);
137     amounts = DMMLibrary.getAmountsOut(amountIn, poolsPath, path);
138     require(
139         amounts[amounts.length - 1] >= amountOutMin,
140         "DMMRouter: INSUFFICIENT_OUTPUT_AMOUNT"
141     );
142     IERC20(path[0]).safeTransferFrom(msg.sender, poolsPath[0], amounts[0]);
143     _swap(amounts, poolsPath, path, to);
```

```
144 }
```

The `_swap()` function calls the `swap()` function on the pool contract in line 124 in order to swap tokens.

DMMRouter02.sol

```
108 //  SWAP
109 // requires the initial amount to have already been sent to the first pool
110 function _swap(
111     uint256[] memory amounts,
112     address[] memory poolsPath,
113     IERC20[] memory path,
114     address _to
115 ) private {
116     for (uint256 i; i < path.length - 1; i++) {
117         (IERC20 input, IERC20 output) = (path[i], path[i + 1]);
118         (IERC20 token0, ) = DMMLibrary.sortTokens(input, output);
119         uint256 amountOut = amounts[i + 1];
120         (uint256 amount0Out, uint256 amount1Out) = input == token0
121             ? (uint256(0), amountOut)
122             : (amountOut, uint256(0));
123         address to = i < path.length - 2 ? poolsPath[i + 1] : _to;
124         IDMMPool(poolsPath[i]).swap(amount0Out, amount1Out, to, new bytes(0));
125     }
126 }
```

In the `DMMPool` contract, the `swap()` function is used to swap tokens between input and output tokens which are held in the pair contract (each `DMMPool` individually). The calculation process of swapping tokens includes the platform fee and the liquidity fee.

For the platform's fee, the `DMMPool` contract validates the `amount0In` value as in line 213 whether it is more than 0 or not. If it is greater than 0, the contract assumes that the swap input token is `_token0`.

Hence, the platform's fee of swapping tokens from `_token1` to `_token0` can be manipulated by making the value of `amount0In` to be more than 0 and as low as possible since the contract validates the amount of `_token0` (`amount0In`) in the first place.

DMMPool.sol

```
160 /// @dev this low-level function should be called from a contract
161 /// @dev which performs important safety checks
162 function swap(
163     uint256 amount0Out,
164     uint256 amount1Out,
165     address to,
166     bytes calldata callbackData
167 ) external override nonReentrant {
168     require(amount0Out > 0 || amount1Out > 0, "DMM:
```

```
INSUFFICIENT_OUTPUT_AMOUNT");
169     (bool isAmpPool, ReserveData memory data) = getReservesData(); // gas
savings
170     require(
171         amount0Out < data.reserve0 && amount1Out < data.reserve1,
        "DMM: INSUFFICIENT_LIQUIDITY"
172     );
173
174     ReserveData memory newData;
175     {
176         // scope for _token{0,1}, avoids stack too deep errors
177         IERC20 _token0 = token0;
178         IERC20 _token1 = token1;
179         require(to != address(_token0) && to != address(_token1), "DMM:
INVALID_TO");
180         if (amount0Out > 0) _token0.safeTransfer(to, amount0Out); //
optimistically transfer tokens
181         if (amount1Out > 0) _token1.safeTransfer(to, amount1Out); //
optimistically transfer tokens
182         if (callbackData.length > 0)
183             IDMMCallee(to).dmmSwapCall(msg.sender, amount0Out, amount1Out,
callbackData);
184         newData.reserve0 = _token0.balanceOf(address(this));
185         newData.reserve1 = _token1.balanceOf(address(this));
186         if (isAmpPool) {
187             newData.vReserve0 =
data.vReserve0.add(newData.reserve0).sub(data.reserve0);
188             newData.vReserve1 =
data.vReserve1.add(newData.reserve1).sub(data.reserve1);
189         }
190     }
191     uint256 amount0In = newData.reserve0 > data.reserve0 - amount0Out
192         ? newData.reserve0 - (data.reserve0 - amount0Out)
193         : 0;
194     uint256 amount1In = newData.reserve1 > data.reserve1 - amount1Out
195         ? newData.reserve1 - (data.reserve1 - amount1Out)
196         : 0;
197     require(amount0In > 0 || amount1In > 0, "DMM: INSUFFICIENT_INPUT_AMOUNT");
198     uint256 feeInPrecision = verifyBalanceAndUpdateEma(
199         amount0In,
200         amount1In,
201         isAmpPool ? data.vReserve0 : data.reserve0,
202         isAmpPool ? data.vReserve1 : data.reserve1,
203         isAmpPool ? newData.vReserve0 : newData.reserve0,
204         isAmpPool ? newData.vReserve1 : newData.reserve1
205     );
206
```

```
207     _update(isAmpPool, newData);
208     emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to,
209 feeInPrecision);
210     {
211         IERC20 tokenFee;
212         uint256 amountIn;
213         if (amount0In > 0) {
214             tokenFee = token0;
215             amountIn = amount0In;
216         } else {
217             tokenFee = token1;
218             amountIn = amount1In;
219         }
220         sendFeeToPlatform(feeInPrecision, tokenFee, amountIn);
221     }
222 }
```

This can be achieved by creating a contract and transfer `_token0` with a tiny amount to the `DMMPool` contract (e.g., 1 wei) to fulfill the condition in line 213 and then call the `swap()` function to swap from `_token1` to `_token0`.

DMMPool.sol

```
210 {
211     IERC20 tokenFee;
212     uint256 amountIn;
213     if (amount0In > 0) {
214         tokenFee = token0;
215         amountIn = amount0In;
216     } else {
217         tokenFee = token1;
218         amountIn = amount1In;
219     }
220     sendFeeToPlatform(feeInPrecision, tokenFee, amountIn);
221 }
```

This causes the platform fee to be calculated from the value of `amount0In` instead of `amount1In`.

5.8.2. Remediation

Inspex suggests modifying the fee collection condition in the `swap()` function to validate both `amount0In` and `amount1In` values, for example:

DMMPool.sol

```
160 /// @dev this low-level function should be called from a contract
161 /// @dev which performs important safety checks
162 function swap(
163     uint256 amount0Out,
164     uint256 amount1Out,
165     address to,
166     bytes calldata callbackData
167 ) external override nonReentrant {
168     require(amount0Out > 0 || amount1Out > 0, "DMM:
    INSUFFICIENT_OUTPUT_AMOUNT");
169     (bool isAmpPool, ReserveData memory data) = getReservesData(); // gas
    savings
170     require(
171         amount0Out < data.reserve0 && amount1Out < data.reserve1,
        "DMM: INSUFFICIENT_LIQUIDITY"
172     );
173
174     ReserveData memory newData;
175     {
176         // scope for _token{0,1}, avoids stack too deep errors
177         IERC20 _token0 = token0;
178         IERC20 _token1 = token1;
179         require(to != address(_token0) && to != address(_token1), "DMM:
    INVALID_TO");
180         if (amount0Out > 0) _token0.safeTransfer(to, amount0Out); //
    optimistically transfer tokens
181         if (amount1Out > 0) _token1.safeTransfer(to, amount1Out); //
    optimistically transfer tokens
182         if (callbackData.length > 0)
183             IDMMCallee(to).dmmSwapCall(msg.sender, amount0Out, amount1Out,
    callbackData);
184         newData.reserve0 = _token0.balanceOf(address(this));
185         newData.reserve1 = _token1.balanceOf(address(this));
186         if (isAmpPool) {
187             newData.vReserve0 =
    data.vReserve0.add(newData.reserve0).sub(data.reserve0);
188             newData.vReserve1 =
    data.vReserve1.add(newData.reserve1).sub(data.reserve1);
189         }
190     }
191     uint256 amount0In = newData.reserve0 > data.reserve0 - amount0Out
192         ? newData.reserve0 - (data.reserve0 - amount0Out)
```

```
193         : 0;
194     uint256 amount1In = newData.reserve1 > data.reserve1 - amount1Out
195         ? newData.reserve1 - (data.reserve1 - amount1Out)
196         : 0;
197     require(amount0In > 0 || amount1In > 0, "DMM: INSUFFICIENT_INPUT_AMOUNT");
198     uint256 feeInPrecision = verifyBalanceAndUpdateEma(
199         amount0In,
200         amount1In,
201         isAmpPool ? data.vReserve0 : data.reserve0,
202         isAmpPool ? data.vReserve1 : data.reserve1,
203         isAmpPool ? newData.vReserve0 : newData.reserve0,
204         isAmpPool ? newData.vReserve1 : newData.reserve1
205     );
206 
207     _update(isAmpPool, newData);
208     emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to,
209 feeInPrecision);
210     {
211         if (amount0In > 0) {
212             sendFeeToPlatform(feeInPrecision, token0, amount0In);
213         }
214         if (amount1In > 0) {
215             sendFeeToPlatform(feeInPrecision, token1, amount1In);
216         }
217     }
218 }
```

5.9. DMM Swap - Improper Function Visibility

ID	IDX-009
Target	ManageUserAddress
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	No Security Impact EvryNet team has acknowledged this best practice recommendation.

5.9.1. Description

Functions with public visibility copy calldata to memory when being executed, while external functions can read directly from calldata. Memory allocation uses more resources (gas) than reading directly from calldata.

The following source code shows that the `addAdmin()` function of the `ManageUserAddress` contract is set to public and it is never called from any internal function.

ManageUserAddress.sol

```
21 function addAdmin(address user) public {
22     require(listUsers[msg.sender] == SUPER_ADMIN_ROLE, "NO PERMISSION");
23     bytes32 _role = getRole(user);
24     require(_role != SUPER_ADMIN_ROLE && _role != ADMIN_ROLE, "CAN NOT SET USER
   TO ADMIN");
25     listUsers[user] = ADMIN_ROLE;
26 }
```

The following table contains all functions that have `public` visibility and are never called from any internal function.

File	Contract	Function
ManageUserAddress.sol (L:21)	ManageUserAddress	addAdmin()
ManageUserAddress.sol (L:28)	ManageUserAddress	removeAdmin()
ManageUserAddress.sol (L:35)	ManageUserAddress	transferSuperAdmin()

5.9.2. Remediation

Inspex suggests changing all functions' visibility to `external` if they are not called from any internal function as shown in the following example:

ManageUserAddress.sol

```
21 function addAdmin(address user) external {
22     require(listUsers[msg.sender] == SUPER_ADMIN_ROLE, "NO PERMISSION");
23     bytes32 _role = getRole(user);
24     require(_role != SUPER_ADMIN_ROLE && _role != ADMIN_ROLE, "CAN NOT SET USER
   TO ADMIN");
25     listUsers[user] = ADMIN_ROLE;
26 }
```

5.10. Farm - Use of Upgradable Contract

ID	IDX-010
Target	Farms
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: High** **Impact: High** The logic of the affected contract can be arbitrarily changed. This allows the proxy owner to perform malicious actions e.g., stealing the user funds anytime they want. **Likelihood: Medium** This action can be performed by the proxy owner without any restriction.
Status	Resolved * EvryNet team has confirmed that timelock will be used to own the contracts to mitigate this issue. This will allow the users to monitor the changes to the smart contracts and act accordingly before the changes are applied. However, the timelock was not in use at the time of the reassessment. Therefore, Inspex suggests the platform users to confirm the usage of the timelock before using the platform.

5.10.1. Description

Smart contracts are designed to be used as agreements that cannot be changed forever. When a smart contract is upgraded, the agreement can be changed from what was previously agreed upon.

As the `Farms` smart contract is upgradable, the logic of it could be modified by the owner anytime, making the smart contract untrustworthy.

5.10.2. Remediation

Inspex suggests deploying the contract without the proxy pattern or any solution that can make smart contract upgradable.

However, if the upgradability is needed, Inspex suggests mitigating this issue by implementing a timelock mechanism with a sufficient length of time to delay the changes. This allows the platform users to monitor the timelock and be notified of the potential changes being done on the smart contracts.

5.11. Farm - Denial of Service in Beneficiary Mechanism

ID	IDX-011
Target	Farms
Category	Advanced Smart Contract Vulnerability
CWE	CWE-755: Improper Handling of Exceptional Conditions
Risk	**Severity: High** **Impact: High** The victim won't be able to execute the `deposit()` function of the Farms contract, causing disruption of service and loss of reputation to the platform. **Likelihood: Medium** This attack can be done by anyone to any address without prior deposit; however, there is no direct benefit for the attacker, resulting in low motivation for the attack.
Status	**Resolved** EvryNet team has resolved this issue as suggested in commit 573a14e6eee92826aedfe440c3d94d0d9ed4a279 by allowing the withdrawal by the beneficiary to the funder.

5.11.1. Description

In the `Farms` contract, users can deposit tokens specified in each pool to gain $EVRY reward using the `deposit()` function. The `_for` variable in the function can be controlled by the users, allowing the deposit by one address for another beneficiary address to gain the reward. The first address that deposits for each `_for` address will be set in the `user.fundedBy` in line 140, preventing others from depositing or withdrawing for that beneficiary due to the condition in line 133.

Farms.sol

```
124 function deposit(
125     address _for,
126     uint256 pid,
127     uint256 amount
128 ) external nonReentrant {
129     PoolInfo memory pool = updatePool(pid);
130     UserInfo storage user = userInfo[pid][_for];
131
132     // Validation
133     if (user.fundedBy != address(0)) require(user.fundedBy == msg.sender,
    "Farms::deposit:: bad sof");
134
135     // Effects
```

```
136     _harvest(_for, pid);
137
138     user.amount = user.amount.add(amount);
139     user.rewardDebt = user.rewardDebt.add(amount.mul(pool.accEVRYPerShare) /
    ACC_EVRY_PRECISION);
140     if (user.fundedBy == address(0)) user.fundedBy = msg.sender;
141
142     // Interactions
143     stakeTokens[pid].safeTransferFrom(msg.sender, address(this), amount);
144     if (isEvryPool(pool.lpToken)) evrySupply = evrySupply.add(amount);
145
146     emit Deposit(msg.sender, pid, amount, _for);
147 }
```

This behavior can be abused by others to disrupt the use of the smart contract. Malicious actors can perform deposits with 0 `amount` for another `_for` address without prior any deposit, preventing that `_for` address from being used by the actual owner.

5.11.2. Remediation

Inspex suggests allowing the `_for` address to perform withdrawal to return the funds to the `fundedBy` address and set the `fundedBy` to `address(0)` when `user.amount` is 0, for example:

Farms.sol

```
153 function withdraw(
154     address _for,
155     uint256 pid,
156     uint256 amount
157 ) external nonReentrant {
158     PoolInfo memory pool = updatePool(pid);
159     UserInfo storage user = userInfo[pid][_for];
160 
161     require(user.fundedBy == msg.sender || msg.sender == _for,
    "Farms::withdraw:: only funder or beneficiary (_for address) ");
162     require(user.amount >= amount, "Farms::withdraw:: amount exceeds");
163 
164     // Effects
165     _harvest(_for, pid);
166 
167     user.rewardDebt = user.rewardDebt.sub(amount.mul(pool.accEVRYPerShare) /
    ACC_EVRY_PRECISION);
168     user.amount = user.amount.sub(amount);
169 
170     // Interactions
171     stakeTokens[pid].safeTransfer(user.fundedBy, amount); // return the funds
    to the funder
172     if (user.amount == 0) user.fundedBy = address(0);
173     if (isEvryPool(pool.lpToken)) evrySupply = evrySupply.sub(amount);
174 
175     emit Withdraw(user.fundedBy, pid, amount, _for);
176 }
```

5.12. Farm - Improper Token Release Mechanism

ID	IDX-012
Target	EVRYDistributor Farms
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity: High** **Impact: High** The smart contracts will be unusable when the cap is reached, preventing the users from depositing or withdrawing their funds. This can cause high damages to the users' funds and loss of reputation to the platform. **Likelihood: Medium** It is likely that $EVRY released from the EVRYDistributor will eventually reach the cap.
Status	**Resolved** EvryNet team has resolved this issue as suggested in commit 573a14e6eee92826aedfe440c3d94d0d9ed4a279 by checking the amount of token released from the `EVRYDistributor` contract.

5.12.1. Description

The `updatePool()` function in the `Farms` contract is used to calculate and distribute the reward to the users. The $EVRY reward is transferred from the `EVRYDistributor` contract using the `release()` function at line 238.

Farms.sol

```
225 function updatePool(uint256 pid) public returns (PoolInfo memory pool) {
226     pool = poolInfo[pid];
227     if (block.number > pool.lastRewardBlock) {
228         uint256 stakeTokenSupply;
229         if (isEvryPool(pool.lpToken)) {
230             stakeTokenSupply = evrySupply;
231         } else {
232             stakeTokenSupply = stakeTokens[pid].balanceOf(address(this));
233         }
234         if (stakeTokenSupply > 0 && totalAllocPoint > 0) {
235             uint256 blocks = block.number.sub(pool.lastRewardBlock);
236             uint256 evryReward =
(blocks.mul(evryPerBlock).mul(pool.allocPoint)).div(totalAllocPoint);
237
238             evryDistributor.release(evryReward);
```

```
239
240             pool.accEVRYPerShare =
pool.accEVRYPerShare.add((evryReward.mul(ACC_EVRY_PRECISION)).div(stakeTokenSup
ply));
241         }
242         pool.lastRewardBlock = block.number;
243         poolInfo[pid] = pool;
244         emit UpdatePool(pid, pool.lastRewardBlock, stakeTokenSupply,
pool.accEVRYPerShare);
245     }
246 }
```

The amount of reward to be released is limited by the `cap` parameter and the balance of $EVRY in the `EVRYDistributor` contract.

EVRYDistributor.sol

```
29 function release(uint256 amount) external nonReentrant onlyOwner {
30
31     require(released.add(amount) <= cap, "release: cap exceed");
32     require(amount <= evry.balanceOf(address(this)), "release: not enough
evry");
33
34     released = released.add(amount);
35     evry.safeTransfer(msg.sender, amount);
36 }
```

However, when the sum of the the reward to be released and the amount released is more than the `cap` amount, the `release()` function will be unusable, causing the transactions that call this function to be reverted, disrupting the availability of the platform.

5.12.2. Remediation

Inspex suggests modifying the `EVRYDistributor` contract to release only the available balance, and return the value of the $EVRY amount released. That value should be checked and used to update the reward of the pools in the `Farm` contract, for example:

EVRYDistributor.sol

```
29 function release(uint256 amount) external nonReentrant onlyOwner returns
30 (uint256 toReleaseAmount) {
31     if (released.add(amount) >= cap) {
32         toReleaseAmount = cap.sub(released);
33     }
34
35     uint256 evryBalance = evry.balanceOf(address(this));
36     if (toReleaseAmount > evryBalance) {
37         toReleaseAmount = evryBalance;
```

```
38      }
39
40      released = released.add(toReleaseAmount);
41      evry.safeTransfer(msg.sender, toReleaseAmount);
    }
```

Farms.sol

```
225 function updatePool(uint256 pid) public returns (PoolInfo memory pool) {
226     pool = poolInfo[pid];
227     if (block.number > pool.lastRewardBlock) {
228         uint256 stakeTokenSupply;
229         if (isEvryPool(pool.lpToken)) {
230             stakeTokenSupply = evrySupply;
231         } else {
232             stakeTokenSupply = stakeTokens[pid].balanceOf(address(this));
233         }
234         if (stakeTokenSupply > 0 && totalAllocPoint > 0) {
235             uint256 blocks = block.number.sub(pool.lastRewardBlock);
236             uint256 evryReward = (blocks.mul(evryPerBlock)
    .mul(pool.allocPoint)).div(totalAllocPoint);
237
238             uint256 evryReleased = evryDistributor.release(evryReward);
239
240             pool.accEVRYPerShare = pool.accEVRYPerShare
    .add((evryReleased.mul(ACC_EVRY_PRECISION)).div(stakeTokenSupply));
241         }
242         pool.lastRewardBlock = block.number;
243         poolInfo[pid] = pool;
244         emit UpdatePool(pid, pool.lastRewardBlock, stakeTokenSupply,
    pool.accEVRYPerShare);
245     }
246 }
```

5.13. Farm - Reward Miscalculation

ID	IDX-013
Target	Farms
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** Medium The reward miscalculation can lead to an unfair $EVRY distribution for the platform users, causing them to gain more or less $EVRY than they should. **Likelihood:** Medium These functions can only be called by the contract owner, but the reward miscalculation is very likely to happen whenever they are called.
Status	Resolved * EvryNet team has resolved this issue in commit 573a14e6eee92826aedfe440c3d94d0d9ed4a279 by adding the `massUpdatePools()` function. However, the remediation applied has a minor chance of causing denial of service due to excessive gas usage, as the loop round in `massUpdatePools()` function cannot be decreased even when a pool is disabled by having its `allocPoint` set to 0. Inspex suggests modifying the contract to make it possible to reduce the loop round. EvryNet team has confirmed that they will resolve this issue in the next release.

5.13.1. Description

When the owner updates the state variables used in $EVRY reward calculation without updating all pools with `updatePool()` function, the users' reward will be miscalculated because the `accEVRYPerShare` state for each pool is not updated accordingly in each pool.

The `accEVRYPerShare` is calculated using `evryPerBlock` and `totalAllocPoint` states. Therefore, if they are modified without updating every pool, the new value will be used to calculate the reward of the period between the last update and the changing of state, resulting in a miscalculation.

Farms.sol

```
222 /// @notice Update reward variables of the given pool.
223 /// @param pid The index of the pool. See poolInfo.
224 /// @return pool returns the Pool that was updated
225 function updatePool(uint256 pid) public returns (PoolInfo memory pool) {
226     pool = poolInfo[pid];
227     if (block.number > pool.lastRewardBlock) {
```

```
228         uint256 stakeTokenSupply;
229         if (isEvryPool(pool.lpToken)) {
230             stakeTokenSupply = evrySupply;
231         } else {
232             stakeTokenSupply = stakeTokens[pid].balanceOf(address(this));
233         }
234         if (stakeTokenSupply > 0 && totalAllocPoint > 0) {
235             uint256 blocks = block.number.sub(pool.lastRewardBlock);
236             uint256 evryReward = (blocks.mul(evryPerBlock)
    .mul(pool.allocPoint)).div(totalAllocPoint);
237
238             evryDistributor.release(evryReward);
239
240             pool.accEVRYPerShare = pool.accEVRYPerShare
    .add((evryReward.mul(ACC_EVRY_PRECISION)).div(stakeTokenSupply));
241         }
242         pool.lastRewardBlock = block.number;
243         poolInfo[pid] = pool;
244         emit UpdatePool(pid, pool.lastRewardBlock, stakeTokenSupply,
    pool.accEVRYPerShare);
245     }
246 }
```

The functions that modify `evryPerBlock` and `totalAllocPoint` state variables are as follows:

- `setEvryPerBlock()`
- `addPool()`
- `setPoolAllocation()`

To demonstrate the impact, please consider the following scenario:

Assuming the state at block 1000 to be as follows:

- `evryPerBlock` = 100
- pool 0 `allocPoint` = 100
- pool 0 `lastRewardBlock` = 1000
- `totalAllocPoint` = 1000

The formula for the reward calculation is as follows:

```
reward = blocks * evryPerBlock * allocPoint / totalAllocPoint
```

If the following sequence of actions is done:

Block	evryPerBlock	Action
1000	100	The update() function is called to calculate the reward of pool 0.
1100	200	The setEvryPerBlock() function is used to set evryPerBlock to 200.
1200	200	The update() function is called to calculate the reward of pool 0.

The reward will be calculated as follows:

- Block 1000 -> Block 1200 reward:

```
reward = (1200 - 1000) * 200 * 100 / 1000 = 4000 $EVRY
```

However, the reward should be calculated using the `evryPerBlock` of each period as follows:

- Block 1000 -> Block 1100 reward:

```
reward = (1100 - 1000) * 100 * 100 / 1000 = 1000 $EVRY
```

- Block 1100 -> Block 1200 reward:

```
reward = (1200 - 1000) * 200 * 100 / 1000 = 2000 $EVRY
```

Therefore, in this scenario, the reward for pool 0 should be 3000 $EVRY instead of 4000 $EVRY.

5.13.2. Remediation

Inspex suggests adding the `massUpdatePool()` function and calling it before the state variables related to the reward calculation are updated. For example:

Farms.sol

```
1  function massUpdatePools() public {
2      uint256 length = poolInfo.length;
3      for (uint256 pid = 0; pid < length; ++pid) {
4          if (poolInfo[pid].allocPoint > 0) {
5              updatePool(pid);
6          }
7      }
8  }
```

Farms.sol

```
79 function setEvryPerBlock(uint256 _evryPerBlock) external onlyOwner {
80     massUpdatePools();
81     evryPerBlock = _evryPerBlock;
82 }
```

Farms.sol

```
87  function addPool(
88      uint256 allocPoint,
89      IERC20 _stakeToken,
90      uint256 _startBlock
91  ) external onlyOwner {
92      require(!isDuplicatedPool(_stakeToken), "Farms::addPool:: stakeToken dup");
93
94      massUpdatePools();
95
96      uint256 lastRewardBlock = block.number > _startBlock ? block.number :
    _startBlock;
97      totalAllocPoint = totalAllocPoint.add(allocPoint);
98
99      stakeTokens.push(_stakeToken);
100
101     poolInfo.push(
102         PoolInfo({ lpToken: _stakeToken, allocPoint: allocPoint,
    lastRewardBlock: lastRewardBlock, accEVRYPerShare: 0 })
103     );
104     emit AddPool(stakeTokens.length.sub(1), allocPoint, _stakeToken);
105 }
```

Farms.sol

```
108 function setPoolAllocation(uint256 _pid, uint256 _allocPoint) external
    onlyOwner {
109     massUpdatePools();
110
111     // Remove current AP value of pool _pid from total AP, then add new one.
112     totalAllocPoint =
    totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
113
114     // Replace old AP value with new one.
115     poolInfo[_pid].allocPoint = _allocPoint;
116
117     emit SetPoolAllocation(_pid, _allocPoint);
118 }
```

5.14. Farm - Centralized Control of State Variable

ID	IDX-014
Target	Farms
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** Medium **Impact:** Medium The controlling authorities can change the critical state variables to gain additional profit. Thus, it is unfair to the other users. **Likelihood:** Medium There is nothing to restrict the changes from being done; however, these actions can only be performed by the contract owner.
Status	Resolved * EvryNet team has confirmed that timelock will be used to own the contracts. This will allow the users to monitor the changes to the smart contracts and act accordingly before the changes are applied. However, the timelock was not in use at the time of the reassessment. Therefore, Inspex suggests the platform users to confirm the usage of the timelock before using the platform.

5.14.1. Description

Critical state variables can be updated any time by the controlling authorities. Changes in these variables can cause impacts to the users, so the users should accept or be notified before these changes are effective.

However, there is no constraint to prevent the authorities from modifying these variables without notifying the users.

The controllable privileged state update functions are as follows:

File	Contract	Function	Modifier
Farms.sol (L:79)	Farms	setEvryPerBlock()	onlyOwner
Farms.sol (L:87)	Farms	addPool()	onlyOwner
Farms.sol (L:108)	Farms	setPoolAllocation()	onlyOwner
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol (L:60)	Farms	renounceOwnership()	onlyOwner

@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol (L:69)	Farms	transferOwnership()	onlyOwner

Please note that the `OwnableUpgradeable` contract is inherited from the OpenZeppelin library.

5.14.2. Remediation

In the ideal case, the critical state variables should not be modifiable to keep the integrity of the smart contract. However, if modifications are needed, Inspex suggests limiting the use of these functions via the following options:

- Implementing a community-run governance to control the use of these functions
- Using a `Timelock` contract to delay the changes for a sufficient amount of time, e.g., 24 hours

5.15. Farm - Insufficient Logging for Privileged Functions

ID	IDX-015
Target	Farms
Category	General Smart Contract Vulnerability
CWE	CWE-778: Insufficient Logging
Risk	**Severity:** Very Low **Impact:** Low Privileged functions' executions cannot be monitored easily by the users. **Likelihood:** Low It is not likely that the execution of the privileged functions will be a malicious action.
Status	Resolved EvryNet team has resolved this issue as suggested in commit 573a14e6eee92826aedfe440c3d94d0d9ed4a279 by emitting an event for the `setEvryPerBlock()` function.

5.15.1. Description

Privileged functions that are executable by the controlling parties are not logged properly by emitting events. Without events, it is not easy for the public to monitor the execution of those privileged functions, allowing the controlling parties to perform actions that cause big impacts to the platform.

The owner can modify the `evryPerBlock` by executing `setEvryPerBlock()` function in the `Farms` contract, and no event is emitted.

Farms.sol

```
79 function setEvryPerBlock(uint256 _evryPerBlock) external onlyOwner {
80     evryPerBlock = _evryPerBlock;
81 }
```

5.15.2. Remediation

Inspex suggests emitting events for the execution of all privileged functions, the `setEvryPerBlock()` function in this case, for example:

Farms.sol

```
79  event SetEvryPerBlock(uint256 _oldEvryPerBlock, uint256 _newEvryPerBlock);
80  function setEvryPerBlock(uint256 _evryPerBlock) external onlyOwner {
81      emit SetEvryPerBlock(evryPerBlock, _evryPerBlock);
82      evryPerBlock = _evryPerBlock;
83  }
```

5.16. Farm - Unsupported Design for Deflationary Token

ID	IDX-016
Target	Farms
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved EvryNet team has resolved this issue as suggested in commit 573a14e6eee92826aedfe440c3d94d0d9ed4a279 by checking the token balance before and after the transfer.

5.16.1. Description

In `Farms` contract, the users can deposit their tokens to acquire rewards ($EVRY). The deposited tokens can be a normal token or LP token depending on the pools added by the contract owner.

However, in the `deposit()` function, an issue could arise when the pool uses a deflationary token (the token that reduces the circulating supply itself when it is transferred).

This means the amount of token that users deposit will be reduced due to the deflationary mechanism, but the contract recognizes it as the full amount as in line 138.

Farms.sol

```
124 function deposit(
125     address _for,
126     uint256 pid,
127     uint256 amount
128 ) external nonReentrant {
129     PoolInfo memory pool = updatePool(pid);
130     UserInfo storage user = userInfo[pid][_for];
131
132     // Validation
133     if (user.fundedBy != address(0)) require(user.fundedBy == msg.sender,
    "Farms::deposit:: bad sof");
134
135     // Effects
136     _harvest(_for, pid);
137
```

```
138     user.amount = user.amount.add(amount);
139     user.rewardDebt = user.rewardDebt.add(amount.mul(pool.accEVRYPerShare) /
    ACC_EVRY_PRECISION);
140     if (user.fundedBy == address(0)) user.fundedBy = msg.sender;
141
142     // Interactions
143     stakeTokens[pid].safeTransferFrom(msg.sender, address(this), amount);
144     if (isEvryPool(pool.lpToken)) evrySupply = evrySupply.add(amount);
145
146     emit Deposit(msg.sender, pid, amount, _for);
147 }
```

The failure of recognizing the token amount could lead to the following scenarios:

Scenario 1: Unable to withdraw staking tokens

Assuming that there is a pool in the `Farms` contract which receives a deflationary token ($TOKEN) with 10% burn rate when the token is transferred.

Currently, there is only User A who stakes $TOKEN to the $TOKEN pool in the `Farms` contract.

Holder	Balance
User A	100

Total $TOKEN in the `Farms` contract: 90

User B deposits 100 $TOKEN to the $TOKEN pool in the `Farms` contract. The `Farms` contract will receive 90 $TOKEN since $TOKEN has 10% deduction from the deflationary mechanism, in this case 10 $TOKEN.

Holder	Balance
User A	100
User B	100

Total $TOKEN in the `Farms` contract: 180

User B then withdraws 100 $TOKEN from the `Farms` contract. The `Farms` contract will validate whether the withdrawn `amount` exceeds the `user.amount`.

Farms.sol

```
153 function withdraw(
154     address _for,
155     uint256 pid,
156     uint256 amount
```

```
157 ) external nonReentrant {
158     PoolInfo memory pool = updatePool(pid);
159     UserInfo storage user = userInfo[pid][_for];
160 
161     require(user.fundedBy == msg.sender, "Farms::withdraw:: only funder");
162     require(user.amount >= amount, "Farms::withdraw:: amount exceeds");
163 
164     // Effects
165     _harvest(_for, pid);
166 
167     user.rewardDebt = user.rewardDebt.sub(amount.mul(pool.accEVRYPerShare) /
ACC_EVRY_PRECISION);
168     user.amount = user.amount.sub(amount);
169     if (user.amount == 0) user.fundedBy = address(0);
170 
171     // Interactions
172     stakeTokens[pid].safeTransfer(msg.sender, amount);
173     if (isEvryPool(pool.lpToken)) evrySupply = evrySupply.sub(amount);
174 
175     emit Withdraw(msg.sender, pid, amount, _for);
176 }
```

Since User B deposited 100 $TOKEN and the balance of $TOKEN in the contract is greater than 100, User B is allowed to withdraw 100 $TOKEN.

Holder	Balance
User A	100
User B	0

Total $TOKEN in the `Farms` contract: 80

As a result, if User A decides to withdraw 100 $TOKEN, or even 90 $TOKEN, this transaction will be reverted since the balance in the contract is insufficient.

Scenario 2: Reward Calculation Exploit

Assuming that there is a pool in the `Farms` contract which receives a deflationary token ($TOKEN) with 10% burn rate when the token is transferred.

Currently, there are several users who stake $TOKEN to the $TOKEN pool in the `Farms` contract with a total supply of 100 $TOKEN.

User A deposits 100 $TOKEN to the contract, and the contract receives 90 $TOKEN due to the deflationary mechanism, resulting in a total supply of 190 $TOKEN.

After that, User A withdraws 100 $TOKEN from staking, the `Farms` contract will then calculate the reward in line 240. During the calculation, the reward is affected by the total amount of $TOKEN (`stakeTokenSupply`).

Farms.sol

```
225 function updatePool(uint256 pid) public returns (PoolInfo memory pool) {
226     pool = poolInfo[pid];
227     if (block.number > pool.lastRewardBlock) {
228         uint256 stakeTokenSupply;
229         if (isEvryPool(pool.lpToken)) {
230             stakeTokenSupply = evrySupply;
231         } else {
232             stakeTokenSupply = stakeTokens[pid].balanceOf(address(this));
233         }
234         if (stakeTokenSupply > 0 && totalAllocPoint > 0) {
235             uint256 blocks = block.number.sub(pool.lastRewardBlock);
236             uint256 evryReward = (blocks.mul(evryPerBlock)
    .mul(pool.allocPoint)).div(totalAllocPoint);
237
238             evryDistributor.release(evryReward);
239
240             pool.accEVRYPerShare = pool.accEVRYPerShare
    .add((evryReward.mul(ACC_EVRY_PRECISION)).div(stakeTokenSupply));
241         }
242         pool.lastRewardBlock = block.number;
243         poolInfo[pid] = pool;
244         emit UpdatePool(pid, pool.lastRewardBlock, stakeTokenSupply,
    pool.accEVRYPerShare);
245     }
246 }
```

Since the `Farms` contract registers the `user.amount` of User A as 100 $TOKEN, the withdrawn $TOKEN amount will be 100, resulting in reducing the total amount of $TOKEN in the contract to 90 $TOKEN.

Hence, the value of `pool.accEVRYPerShare` can be increased dramatically by manipulating the total amount of $TOKEN (`stakeTokenSupply`) to be as low as possible.

User A can repeatedly execute `withdraw()` and `deposit()` functions to drain the $TOKEN in the contract until it is as low as possible, for example, 1 `wei`, causing `accEVRYPerShare` state to be overly inflated, so the users can claim an exceedingly large amount of reward ($EVRY) from the contract.

However, since only LP tokens are planned to be used in `Farms` pools, there is no direct impact for this issue.

5.16.2. Remediation

Inspex suggests modifying the logic of the `deposit()` function to validate the amount of the token received from the user instead of using the value of `amount` parameter directly, for example:

Farms.sol

```
124 function deposit(
125     address _for,
126     uint256 pid,
127     uint256 amount
128 ) external nonReentrant {
129     PoolInfo memory pool = updatePool(pid);
130     UserInfo storage user = userInfo[pid][_for];
131 
132     // Validation
133     if (user.fundedBy != address(0)) require(user.fundedBy == msg.sender,
    "Farms::deposit:: bad sof");
134 
135     // Effects
136     _harvest(_for, pid);
137     if (user.fundedBy == address(0)) user.fundedBy = msg.sender;
138 
139     uint256 tokenBalanceBefore = stakeTokens[pid].balanceOf(address(this));
140     stakeTokens[pid].safeTransferFrom(msg.sender, address(this), amount);
141     uint256 tokenAmountReceived = stakeTokens[pid].balanceOf(address(this))
    .sub(tokenBalanceBefore);
142 
143     user.amount = user.amount.add(tokenAmountReceived);
144     user.rewardDebt =
    user.rewardDebt.add(tokenAmountReceived.mul(pool.accEVRYPerShare) /
    ACC_EVRY_PRECISION);
145 
146     if (isEvryPool(pool.lpToken)) evrySupply =
    evrySupply.add(tokenAmountReceived);
147 
148     emit Deposit(msg.sender, pid, amount, _for);
149 }
```

Please note that remediations for other issues are not yet applied to the example above.

5.17. Farm - Availability of Emergency Withdraw Function

ID	IDX-017
Target	Farms
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	No Security Impact EvryNet team has acknowledged this best practice recommendation.

5.17.1. Description

In most `MasterChef`-like yield farming contracts, the `emergencyWithdraw()` function is available for the users to withdraw their funds in an emergency case without caring about the reward. However, there is no such function for the users to use in the `Farms` contract.

This function can be useful in the cases that unforeseen circumstances cause impact to the state variables of the contract, causing the `withdraw()` function to be unusable due to unfulfilled conditions, so the users may not be able to withdraw their funds in a normal way.

5.17.2. Remediation

Inspex suggests implementing an `emergencyWithdraw()` function for the users to use in an emergency case, for example:

Farms.sol

```
1  // Withdraw without caring about rewards. EMERGENCY ONLY.
2  function emergencyWithdraw(address _for, uint256 pid) external nonReentrant {
3      PoolInfo memory pool = poolInfo[pid];
4      UserInfo storage user = userInfo[pid][_for];
5
6      require(user.fundedBy == msg.sender || msg.sender == _for,
   "Farms::emergencyWithdraw:: only funder or beneficiary");
7
8      stakeTokens[pid].safeTransfer(user.fundedBy, amount);
9
10     emit EmergencyWithdraw(user.fundedBy, pid, user.amount, _for);
11
12     if (isEvryPool(pool.lpToken)) evrySupply = evrySupply.sub(user.amount);
```

```
13      user.amount = 0;
14      user.rewardDebt = 0;
15      user.fundedBy = address(0);
16  }
```

5.18. Farm - Improper Function Visibility

ID	IDX-018
Target	Farms
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	No Security Impact EvryNet team has acknowledged this best practice recommendation.

5.18.1. Description

Functions with public visibility copy calldata to memory when being executed, while external functions can read directly from calldata. Memory allocation uses more resources (gas) than reading directly from calldata.

For example, the following source code shows that the `initialize()` function of the `Farms` contract is set to public and it is never called from any internal function.

Farms.sol

```
67  function initialize(
68      EVRYDistributor _evryDistributor,
69      uint256 _evryPerBlock
70  ) public initializer {
71      __Ownable_init();
72      __ReentrancyGuard_init();
73
74      evryDistributor = _evryDistributor;
75      evry = evryDistributor.evry();
76      evryPerBlock = _evryPerBlock;
77  }
```

5.18.2. Remediation

Inspex suggests changing the `initialize()` function's visibility to `external` if it is not called from any internal function as shown in the following example:

Farms.sol

```
67  function initialize(
68      EVRYDistributor _evryDistributor,
```

```
69      uint256 _evryPerBlock
70  ) external initializer {
71      __Ownable_init();
72      __ReentrancyGuard_init();
73
74      evryDistributor = _evryDistributor;
75      evry = evryDistributor.evry();
76      evryPerBlock = _evryPerBlock;
77  }
```

6. Appendix

6.1. About Inspex

Inspex is formed by a team of cybersecurity experts highly experienced in various fields of cybersecurity. We provide blockchain and smart contract professional services at the highest quality to enhance the security of our clients and the overall blockchain ecosystem.

Follow Us On:

Website	https://inspex.co
Twitter	@InspexCo
Facebook	https://www.facebook.com/InspexCo
Telegram	@inspex_announcement

6.2. References

[1] “OWASP Risk Rating Methodology.” [Online]. Available: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology. [Accessed: 08-May-2021]

inspeX

CYBERSECURITY PROFESSIONAL SERVICE