QuillAudits

Audit Report
May, 2021

Dfyn

Contents

Introduction

This audit report highlights the overall security of the DFYN staking rewards and staking reward factory with commit hash f44a4. With this report, QuillHash Audit has tried to ensure the reliability of the smart contract by completing the assessment of their system's architecture and smart contract codebase.

Auditing Approach and Methodologies applied

In this audit, we consider the following crucial features of the code.

- Whether the implementation of ERC 20 standards.
- Whether the code is secure.
- Gas Optimization
- Whether the code meets the best coding practices.
- Whether the code meets the SWC Registry issue.

The audit has been performed according to the following procedure:

Manual Audit

- Inspecting the code line by line and revert the initial algorithms of the protocol and then compare them with the specification
- Manually analyzing the code for security vulnerabilities.
- Gas Consumption and optimisation
- Assessing the overall project structure, complexity & quality.
- Checking SWC Registry issues in the code.
- Unit testing by writing custom unit testing for each function.
- Checking whether all the libraries used in the code of the latest version.
- Analysis of security on-chain data.
- Analysis of the failure preparations to check how the smart contract performs in case of bugs and vulnerability.

Automated analysis

- Scanning the project's code base with Mythril, Slither, Echidna, Manticore, others.
- Manually verifying (reject or confirm) all the issues found by tools.
- Performing Unit testing.

- Manual Security Testing (SWC-Registry, Overflow)
- Running the tests and checking their coverage.

Report: All the gathered information is described in this report.

Audit Details

Project Name: DFYN

Contract commit hash:

https://github.com/dfyn/dual-farm/commit/f44a4dcbeb41f38a9c02cb877a8c95b92685f972

Contract files:

https://github.com/dfyn/dual-farm/blob/main/contracts/StakingRewardsFactory.sol

https://github.com/dfyn/dual-farm/blob/main/contracts/StakingRewards.sol

Language: Solidity
Platform and tools: HardHat, Remix, VScode, solhint and other tools mentioned in the automated analysis section.

Audit Goals

The focus of this audit was to verify whether the smart contract is secure, resilient, and working according to the standard specs. The audit activity can be grouped into three categories.

Security
Identifying security related issues within each contract and the system of contract.

Sound Architecture
Evaluating the architect of a system through the lens of established smart contract best practice and general software practice.

Code Correctness and Quality
A full review of the contract source code. The primary areas of focus include

- Correctness.
- Section of code with high complexity.
- Readability.
- Quantity and quality of test coverage.

Issue Categories

Every issue in this report was assigned a severity level from the following:

High severity issues

Issues on this level are critical to the smart contract's performance/ functionality and should be fixed before moving to a live environment.

Medium severity issues

Issues on this level could potentially bring problems and should eventually be fixed.

Low severity issues

Issues on this level are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Informational

These are severity four issues which indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

	High	Medium	Low	Informational
Open	0	0	3	1
Closed	0	1	1	0

Manual Audit

SWC Registry test

We have tested some known SWC registry issues. Out of all tests, only SWC 116, 102 and 103 got detected. All are the low priority ones and we have discussed them above already.

Serial No.	Description	Comments
SWC-132	Unexpected Ether balance	Pass: Avoided strict equality checks for the Ether balance in a contract
SWC-131	Presence of unused variables	Pass: No unused variables
SWC-128	DoS With Block Gas Limit	Pass
SWC-122	Lack of Proper Signature Verification	Pass
SWC-120	Weak Sources of Randomness from Chain Attributes	Pass
SWC-119	Shadowing State Variables	Pass: No ambiguous found.
SWC-118	Incorrect Constructor Name	Pass. No incorrect constructor name used
SWC-116	Timestamp Dependence	Found
SWC-115	Authorization through tx.origin	Pass: No tx.origin found
SWC-114	Transaction Order Dependence	Pass

Serial No.	Description	Comments
SWC-113	DoS with Failed Call	Pass: No failed call
SWC-112	Delegatecall to Untrusted Callee	Pass
SWC-111	Use of Deprecated Solidity Functions	Pass: No deprecated function used
SWC-108	State Variable Default Visibility	Pass: Explicitly defined visibility for all state variables
SWC-107	Reentrancy	Pass
SWC-106	Unprotected SELF-DESTRUCT Instruction	Pass: Not found any such vulnerability
SWC-104	Unchecked Call Return Value	Pass: Not found any such vulnerability
SWC-103	Floating Pragma	Found
SWC-102	Outdated Compiler Version	Found
SWC-101	Integer Overflow and Underflow	Pass

High level severity issues

No issues found

Medium level severity issues

There was 1 medium severity issue found.

1. **Costly Loop**
   The loop in the contract includes state variables like .length of a non-memory array, in the condition of the for loops.
   As a result, these state variables consume a lot more extra gas for every iteration of the 'for' loop.
   The below functions include such loops at the above-mentioned lines:
   - **notifyRewardAmounts () → StakingRewardsFactory.sol**
   - **notifyRewardAmount → StakingRewardsFactory.sol**
   - **Deploy → StakingRewardsFactory.sol**
   - **stakerBalances → StakingRewards.sol**
   - **getReward → StakingRewards.sol**
   - **rescueFunds → StakingRewards.sol**

```
106     function notifyRewardAmounts() public {
107         require(
108             stakingTokens.length > 0,
109             "StakingRewardsFactory::notifyRewardAmounts: called before any deploys"
110         );
111         for (uint256 i = 0; i < stakingTokens.length; i++) {
112             notifyRewardAmount(stakingTokens[i]);
113         }
114     }
```

Recommendation:
It's quite effective to use a local variable instead of a state variable like .length in a loop. For instance,

```
uint256 local_variable = _groupInfo.addresses.length;
 for (uint256 i = 0; i < local_variable; i++) {
            if (_groupInfo.addresses[i] == msg.sender) {
                _isAddressExistInGroup = true;
                _senderIndex = i;
                break;
            }
        }
```

Reading reference link
https://blog.b9lab.com/getting-loopy-with-solidity-1d51794622ad

Status: As informed by the Dfyn team, they are working on Layer 2. For Layer 2 this bug would be false positive. Hence marking the issue CLOSED.

Low level severity issues

There were 4 low severity issues found.

1. **Description** → **SWC 102**: Outdated Compiler Version

```
1  // SPDX-License-Identifier: MIT
2
3  pragma solidity >=0.6.11;
4
```

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version.

Remediation
It is recommended to use a recent version of the Solidity compiler which is Version 0.8.4.

2. **Description** → **SWC 103:** Floating Pragma

```
1  // SPDX-License-Identifier: MIT
2
3  pragma solidity >=0.6.11;
4
```

Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation
Lock the pragma version and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package. Otherwise, the developer would need to manually update the pragma in order to compile locally.

3. **Description:** Potential use of "block.timestamp" as source of randomness

   In StakingRewardsFactory.sol: Line no: 35 & 120

   In StakingRewards.sol: Line no: 85, 220, 222, 224, 235, 239

```
219         uint256 rewardRate = tokenRewardRate[rewardToken];
220         if (block.timestamp >= periodFinish) {
221             rewardRate = reward.div(rewardsDuration);
222             periodFinish = block.timestamp.add(rewardsDuration);
223         } else {
224             uint256 remaining = periodFinish.sub(block.timestamp);
225             uint256 leftover = remaining.mul(rewardRate);
226             rewardRate = reward.add(leftover).div(remaining);
227         }
```

   Contracts often need access to time values to perform certain types of functionality. Values such as block.timestamp, and block.number can give you a sense of the current time or a time delta, however, they are not safe to use for most purposes.
   In the case of block.timestamp, developers often attempt to use it to trigger time-dependent events. As Ethereum is decentralized, nodes can synchronize time only to some degree. Moreover, malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. However, miners can't set a timestamp smaller than the previous one (otherwise the block will be rejected), nor can they set the timestamp too far ahead in the future. Taking all of the above into consideration, developers can't rely on the preciseness of the provided timestamp.

   Remediation

   Developers should write smart contracts with the notion that block values are not precise, and the use of them can lead to unexpected effects. Alternatively, they may make use of oracles.

   References

   - Safety: Timestamp dependence
   - Ethereum Smart Contract Best Practices - Timestamp Dependence

- How do Ethereum mining nodes maintain a time consistent with the network?
- Solidity: Timestamp dependency, is it possible to do safely?

Status: As informed by the Dfyn team, they are working on Layer 2. For Layer 2 this bug would be false positive. Hence marking the issue CLOSED.

4. **Description:** Prefer external to public visibility level

   A function with a **public** visibility modifier that is not called internally. Changing the visibility level to **external** increases code readability. Moreover, in many cases, functions with **external** visibility modifiers spend less gas compared to functions with **public** visibility modifiers. The function definition in the file StakingRewards.sol are marked as public
   - lastTimeRewardApplicable
   - rewardPerToken
   - Earned
   - Withdraw
   - getReward

   And in the file StakingRewardsFactory.sol below function definition are also marked as public
   - rescueFunds
   - rescueFactoryFunds
   - notifyRewardAmounts
   - notifyRewardAmount
   - stakingRewardsInfo

   However, it is never directly called by another function in the same contract or in any of its descendants. Consider marking it as"external" instead

   Recommendations
   Use the external visibility modifier for functions never called from the contract via internal call. Reading Link.

   Note: Exact same issue was found while using automated testing by Mythx.

Informational

1. **Description: Missing reentrancy protection (StakingRewards.sol)**

   The **getReward** function did not make use of a modifier to protect against potential reentrancy attacks. If a token were to implement a callback (e.g. ERC-223 or ERC-777), the function could in theory be targeted through a reentrancy attack. However, as the checks-effects pattern was used the potential for exploitation was mitigated.

```
189
190     function getReward()
191         public
192         override
193         nonReentrant
194         updateReward(_msgSender())
195     {
196         for (uint256 i = 0; i < rewardTokens.length; i++) {
197             uint256 reward = rewards[_msgSender()][rewardTokens[i]];
198             if (reward > 0) {
199                 rewards[_msgSender()][rewardTokens[i]] = 0;
200                 IERC20(rewardTokens[i]).safeTransfer(_msgSender(), reward);
201                 emit RewardPaid(_msgSender(), reward);
202             }
203         }
204     }
205
```

   Recommendations
   A ReentrancyGuard could be used to protect against reentrancy attacks as a defence-in-depth measure.

Functional test

Function test has been done for multiple functions of both the files. Results are below

StakingRewardsFactory.sol

- **deploy** function was able to deploy a staking reward contract for the staking token and store the reward tokens addresses and amounts.
  -- > PASS

- **rescueFunds** function was able to rescue extra funds from StakingRewards.sol also, cannot able to access staking tokens.
  --> PASS

- **rescueFactoryFunds** function was able to rescue extra funds from StakingRewardsFactory.sol and, transfer them to the owners account.
  --> PASS

- **notifyRewardAmounts** function was able to transfer the reward amount of all the staking reward pools.
  --> PASS

- **notifyRewardAmount** function notify reward amount for an individual staking token.
  --> PASS

- **stakingRewardsInfo** returns a staking reward pool address, array of reward token addresses and array of reward amount.
  --> PASS

StakingReward.sol

- **totalSupply** returns total supply of staking token.
  --> PASS

- **stakerBalances** returns array of stakers and array of their balances.
  --> PASS

- **lastTimeRewardApplicable** returns the minimum between Period End and current time.
  --> PASS

- **rewardPerToken** returns reward per token
  --> PASS

- **earned returns** reward earned by caller
  --> PASS

- **getRewardForDuration** returns the duration in unix timestamp.
  --> PASS

- **stakeWithPermit** function works as stake for LP Tokens and emits the event.
  --> PASS

- **withdraw** function works for UnStaking of LP Tokens and emits the event.
  --> PASS

- **getReward** function calculates the rewards for a caller and transfers the reward.
  --> PASS

- **exit** Withdraw all LP tokens and rewards for a caller.
  --> PASS

- **notifyRewardAmount** will call this method to notify to start reward generation
  --> PASS

- **rescueFunds** rescue extra funds only called my factory also, cannot able to access staking tokens
  --> PASS

- **updateReward** acts as a modifier that updates pool information in every mutation calls.
  --> PASS

Automated Testing

We have used multiple automated testing frameworks. This makes code more secure and common attacks. The results are below.

Slither

Slither is a Solidity static analysis framework that runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to easily write custom analyses. Slither enables developers to find vulnerabilities, enhance their code comprehension, and quickly prototype custom analyses. After running Slither we got the results below.

```
Compilation warnings/errors on ./StakingRewards.sol:
Error: Source "@openzeppelin/contracts/math/Math.sol" not found: File not found.
 --> ./StakingRewards.sol:5:1:
  |
5 | import "@openzeppelin/contracts/math/Math.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/math/SafeMath.sol" not found: File not found.
 --> ./StakingRewards.sol:6:1:
  |
6 | import "@openzeppelin/contracts/math/SafeMath.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/token/ERC20/SafeERC20.sol" not found: File not found.
 --> ./StakingRewards.sol:7:1:
  |
7 | import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/utils/ReentrancyGuard.sol" not found: File not found.
 --> ./StakingRewards.sol:8:1:
  |
8 | import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/math/SafeMath.sol" not found: File not found.
 --> ./libraries/NativeMetaTransaction/NativeMetaTransaction.sol:5:1:
  |
```

```
crytic_compile.platform.exceptions.InvalidCompilation: Invalid solc compilation Error: Source "@openzeppe
lin/contracts/math/Math.sol" not found: File not found.
 --> ./StakingRewards.sol:5:1:
  |
5 | import "@openzeppelin/contracts/math/Math.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/math/SafeMath.sol" not found: File not found.
 --> ./StakingRewards.sol:6:1:
  |
6 | import "@openzeppelin/contracts/math/SafeMath.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/token/ERC20/SafeERC20.sol" not found: File not found.
 --> ./StakingRewards.sol:7:1:
  |
7 | import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/utils/ReentrancyGuard.sol" not found: File not found.
 --> ./StakingRewards.sol:8:1:
  |
8 | import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/math/SafeMath.sol" not found: File not found.
 --> ./libraries/NativeMetaTransaction/NativeMetaTransaction.sol:5:1:
  |
```

```
lin/contracts/math/Math.sol" not found: File not found.
 --> ./StakingRewards.sol:5:1:
  |
5 | import "@openzeppelin/contracts/math/Math.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/math/SafeMath.sol" not found: File not found.
 --> ./StakingRewards.sol:6:1:
  |
6 | import "@openzeppelin/contracts/math/SafeMath.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/token/ERC20/SafeERC20.sol" not found: File not found.
 --> ./StakingRewards.sol:7:1:
  |
7 | import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/utils/ReentrancyGuard.sol" not found: File not found.
 --> ./StakingRewards.sol:8:1:
  |
8 | import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/math/SafeMath.sol" not found: File not found.
 --> ./libraries/NativeMetaTransaction/NativeMetaTransaction.sol:5:1:
  |
5 | import '@openzeppelin/contracts/math/SafeMath.sol';
```

```
Compilation warnings/errors on StakingRewardsFactory.sol:
Error: Source "@openzeppelin/contracts/token/ERC20/IERC20.sol" not found: File not found.
 --> StakingRewardsFactory.sol:4:1:
  |
4 | import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/access/Ownable.sol" not found: File not found.
 --> StakingRewardsFactory.sol:5:1:
  |
5 | import "@openzeppelin/contracts/access/Ownable.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/math/Math.sol" not found: File not found.
 --> StakingRewards.sol:5:1:
  |
5 | import "@openzeppelin/contracts/math/Math.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/math/SafeMath.sol" not found: File not found.
 --> StakingRewards.sol:6:1:
  |
6 | import "@openzeppelin/contracts/math/SafeMath.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/token/ERC20/SafeERC20.sol" not found: File not found.
 --> StakingRewards.sol:7:1:
  |
```


```
4 | import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/access/Ownable.sol" not found: File not found.
 --> StakingRewardsFactory.sol:5:1:
  |
5 | import "@openzeppelin/contracts/access/Ownable.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/math/Math.sol" not found: File not found.
 --> StakingRewards.sol:5:1:
  |
5 | import "@openzeppelin/contracts/math/Math.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/math/SafeMath.sol" not found: File not found.
 --> StakingRewards.sol:6:1:
  |
6 | import "@openzeppelin/contracts/math/SafeMath.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/token/ERC20/SafeERC20.sol" not found: File not found.
 --> StakingRewards.sol:7:1:
  |
7 | import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
Error: Source "@openzeppelin/contracts/math/Math.sol" not found: File not found.
 --> StakingRewards.sol:5:1:
  |
5 | import "@openzeppelin/contracts/math/Math.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/math/SafeMath.sol" not found: File not found.
 --> StakingRewards.sol:6:1:
  |
6 | import "@openzeppelin/contracts/math/SafeMath.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/token/ERC20/SafeERC20.sol" not found: File not found.
 --> StakingRewards.sol:7:1:
  |
7 | import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/utils/ReentrancyGuard.sol" not found: File not found.
 --> StakingRewards.sol:8:1:
  |
8 | import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Error: Source "@openzeppelin/contracts/math/SafeMath.sol" not found: File not found.
 --> libraries/NativeMetaTransaction/NativeMetaTransaction.sol:5:1:
  |
5 | import '@openzeppelin/contracts/math/SafeMath.sol';
```

Description

Unable to locate the right file from Openzepplin

Recommendation

Use the specific version tag of each repo to get rid of the above error like done at Uniswap.

Manticore

Manticore is a symbolic execution tool for the analysis of smart contracts and binaries. It executes a program with symbolic inputs and explores all the possible states it can reach. It also detects crashes and other failure cases in binaries and smart contracts.
Manticore results throw the same warning which is similar to the Slither warning.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or endorsement of the **Dfyn platform**. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Dfyn Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Summary

The use of smart contracts is simple and the code is relatively small. Altogether the code is written and demonstrates effective use of abstraction, separation of concern, and modularity. But there are a few issues/vulnerabilities to be tackled at various security levels, it is recommended to fix them before deploying the contract on the main network. Given the subjective nature of some assessments, it will be up to the Dfyn team to decide whether any changes should be made.

Dfyn

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com