PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For DragonSwap (Staker)

18 April 2024

paladinsec.co

info@paladinsec.co

Table of Contents

Table of Contents 2

Disclaimer 3

1 Overview 4

1.1 Summary 4

1.2 Contracts Assessed 4

1.3 Findings Summary 5

1.3.1 DragonswapStaker and DragonswapStakerBoosted 6

1.3.2 DragonswapStakerFactory 6

2 Findings 7

2.1 DragonswapStaker and DragonswapStakerBoosted 7

2.1.1 Privileged Functions 7

2.1.2 Issues & Recommendations 8

2.2 DragonswapStakerFactory 15

2.2.1 Privileged Functions 15

2.2.2 Issues & Recommendations 16

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for DragonSwap's Staker contracts on the SEI network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	DragonSwap
URL	https://dragonswap.app
Platform	SEI
Language	Solidity
Preliminary Contracts	https://github.com/dragonswap-app/dragonswap-farming/tree/91fec8f1df2a543e4a0cf3e93dd04ae90365a0a5
Resolution	https://github.com/dragonswap-app/dragonswap-farming/commit/cb9e0e425e0864235446004853a5894347f0f216

1.2 Contracts Assessed

Name	Contract	Live Code Match
`DragonswapStaker`	`0xF1d1598c2C48a70D708537693c89d6D699a902cb`	MATCH
`DragonswapStakerBoosted`	`0x69D40C24552A9e6884345e5c9966363178c708dF`	MATCH
`DragonswapStakerFactory`	`0x34975246B5a3c8e9E1C19BF9B6a6088b741d4E2D`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
Governance	0	-	-	-
High	2	2	-	-
Medium	3		1	2
Low	3	2	-	1
Informational	2	2	-	-
Total	**10**	**6**	**1**	**3**

Classification of Issues

Severity	Description
Governance	Issues under this category are where the governance or owners of the protocol have certain privileges that users need to be aware of, some of which can result in the loss of user funds if the governance's private keys are lost or if they turn malicious, for example.
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 DragonswapStaker and DragonswapStakerBoosted

ID	Severity	Summary	Status
01	HIGH	`totalAllocPoint` is incorrectly calculated in set	✓ RESOLVED
02	HIGH	First funder can set ratio to an unfavorable one to brick booster rewards	✓ RESOLVED
03	MEDIUM	Users who are blacklisted in `rewardToken` cannot claim rewards of `boosterToken`	ACKNOWLEDGED
04	MEDIUM	Users will be unable to withdraw and deposit if `totalAllocPoint == 0`	ACKNOWLEDGED
05	MEDIUM	Contracts do not work with tokens that have a fee on transfer or non-standard ERC20 tokens	PARTIAL
06	LOW	Reward leakage in staking contracts due to rounding down	✓ RESOLVED
07	LOW	DoS by adding multiple pools	ACKNOWLEDGED
08	INFO	Typographical issues	✓ RESOLVED

1.3.2 DragonswapStakerFactory

ID	Severity	Summary	Status
09	LOW	`DragonswapStakerFactory` deploys contracts with `msg.value` and initializes instances with `msg.value`, but the `deploy` function is not payable	✓ RESOLVED
10	INFO	Typographical issues	✓ RESOLVED

2 Findings

2.1 DragonswapStaker and DragonswapStakerBoosted

`DragonswapStaker` is a modified variant of Sushiswap's masterchef contract. Users can stake pool tokens and receive rewards in `rewardToken`. Notably, the key changes made are:

1. A new variable `totalDeposits` was introduced to the pool to track total user deposits. When `updatePool` is called, this new variable `totalDeposits` is used to track `lpsupply` instead of `balanceOf` of contract.
2. New function `fund()` is added. It is a permission-less function which allows anyone to fund reward tokens to the contract.
3. Addition of a new variable `rewardsPaidOut` to track total rewards paid to users.
4. Contracts are now upgradable.

`DragonswapStakerBooster` is largely similar, except the concept of Boosted rewards is also introduced. When users stake, they are not only rewarded in reward tokens, but a percentage of the reward tokens are also given to users in boosted tokens.

2.1.1 Privileged Functions

- `add`
- `set`

2.1.2 Issues & Recommendations

Issue #01	**`totalAllocPoint` is incorrectly calculated in `set`**
Severity	HIGH SEVERITY

Description	The issue lies in the incorrect subtraction during the recalculation of `totalAllocPoint`. The code is supposed to subtract the existing `allocPoint` of a pool and `_allocPoint` (the new allocation point) from the total allocation point. However, due to a coding error, both are subtracted. This can result in a lesser `totalAllocPoint` than anticipated.
	`totalAllocPoint -= poolInfo[_pid].allocPoint + _allocPoint;`
	
	The code translates into
	`totalAllocPoint = totalAllocPoint - poolInfo[_pid].allocPoint - _allocPoint;`

Recommendation	Consider changing the subtraction into `totalAllocPoint = totalAllocPoint - poolInfo[_pid].allocPoint + _allocPoint;`
Resolution	RESOLVED

Issue #02	First funder can set ratio to an unfavorable one to brick booster rewards
Severity	HIGH SEVERITY
Description	`fund()` is permissionless which means that anyone can fund the boosted staker contract. Every amount funded must follow the initial ratio set by the first funder. Any extra amount sent is refunded to the sender based on the initial ratio. The first funder can set the ratio. A malicious funder can brick the booster rewards by setting ratio to an unfavorable one. For instance, the first funder sends `1e18` `rewardAmount` and 1 `boosterAmount`. `inputRatio` would be `1e36`, meaning that the ratio we have now is `1e18 : 1`. Any future funders will have the `boosterAmount` reward they have sent be refunded due to the extremely small initial ratio for `boosterAmount` set. Thus, the booster rewards contract cannot be utilized correctly the way it was intended.
Recommendation	The protocol should explicitly set a reasonable initial ratio instead of letting it be set by the public in a permissionless manner.
Resolution	RESOLVED

Issue #03	**Users who are blacklisted in `rewardToken` cannot claim rewards of `boosterToken`**
Severity	MEDIUM SEVERITY
Description	Common ERC20s such as USDC and USDT have a blacklist function. When an address is blacklisted, the address cannot send or receive the respective token. Within the `withdraw()` function of `DragonswapStakerBoosted.sol`, reward tokens, booster tokens, and pool tokens are all calculated and then sent to the caller. If the user is blacklisted in `rewardToken`, `withdraw()` will be DOSed. Users can however use `emergencyWithdraw()` to reclaim their pool tokens. However, not only do they have to give up their `rewardToken` (rightfully so since they are blacklisted), they also have to give up their `boostedToken`.
Recommendation	Instead of using a push method of transferring tokens to users, use a pull method instead. When users accrue rewards, use a new variable to track them, and allow users to claim rewards on their own accord.
Resolution	ACKNOWLEDGED

Issue #04	**Users will be unable to withdraw and deposit if `totalAllocPoint == 0`**
Severity	MEDIUM SEVERITY
Description	The pending and `updatePool` functions calculate rewards based on the value of `totalAllocPoint`. More specifically, a division is done on `totalAllocPoint`. If `totalAllocPoint == 0`, then there will be a division by zero error and both these functions will revert. This will cause a DOS on two core functions of the Masterchef contract: `withdraw` and `deposit`.
Recommendation	Consider adding the check of `totalAllocPoint != 0` in both functions. Note that this issue is found in both `DragonswapStaker.sol` and `DragonswapStakerBoosted.sol`.
Resolution	ACKNOWLEDGED The team stated: "Owner might want to add a pool but determine the allocation point later, which would not let anyone deposit or withdraw at the time, even if alloc-point of an existing pool was suddenly changed to zero, users would still be able to make an emergency withdraw so every problem regarding this is evadable."

Issue #05	Contracts do not work with tokens that have a fee on transfer or non-standard ERC20 tokens
Severity	MEDIUM SEVERITY

Description	Both contracts do not work with tokens that have a fee on transfer or with tokens that do not respect ERC20 standard, like ERC677 or ERC777.
	
	For tokens with a fee on transfer, the `balanceBefore/balanceAfter` approach which calculates the real amount that is transferred in the contract should be used.
	
	For ERC677/ERC777, the checks-effects-interaction is not respected in several instances.
	
	For example within `emergencyWithdraw`, the transfer to the user is done before updating the state variables.
	
	`function emergencyWithdraw(uint256 _pid) external {`
	`    PoolInfo storage pool = poolInfo[_pid];`
	`    UserInfo storage user = userInfo[_pid][msg.sender];`
	
	`    pool.totalDeposits -= user.amount;`
	`    pool.pooledToken.safeTransfer(address(msg.sender), user.amount);`
	`    emit EmergencyWithdraw(msg.sender, _pid, user.amount);`
	
	`    user.amount = 0;`
	`    user.rewardDebt = 0;`
	`}`

Recommendation	Consider not using any tokens with a fee on transfer and ERC677/ERC777 tokens. Furthermore, we highly recommend to respecting the Checks-Effects-Interactions pattern to avoid unwanted reentrancy attack paths.
Resolution	PARTIALLY RESOLVED This issue has been fixed to support tokens with a fee on transfer or in cases where the received amount differs from what was originally sent.

Issue #06	Reward leakage in staking contracts due to rounding down
Severity	LOW SEVERITY
Description	When users call `fund()` on the contract, `endTimestamp` is calculated as: `endTimestamp += rewardAmount / rewardPerSecond;` `endTimestamp` is used to calculate the amount of rewards users are entitled to in each second. However, small amounts of rewards are lost here due to the rounding down effect for `endTimestamp`. For instance, if `rewardAmount == 5` and `rewardPerSecond == 2`, `endTimestamp = 2`. Only a maximum reward of 4 can be released to users and the additional one reward is stuck in the contract.
Recommendation	Check that the `rewardAmount` amount funded is a multiple of `rewardPerSecond`, or refund users the excess reward sent in.
Resolution	RESOLVED

Issue #07	DoS by adding multiple pools
Severity	LOW SEVERITY
Description	The owner can add as many pools as they wish, eventually rendering `massUpdatePools` unusable as it will reiterate through all the pools.
Recommendation	Consider either adding a tested `MAX_LIMIT` on the number of pools that can be added or consider closely monitoring this with fork-testing before adding a new pool. This means that before adding a new pool, a fork-test should be done which mimics adding a new pool and then call `massUpdatePools` and `set` to see if the gas limits are manageable.
Resolution	ACKNOWLEDGED

Issue #08	Typographical issues
Severity	INFORMATIONAL

Description	`if (_pooledToken == address(0)) revert();`
	
	The revert in the add function in `DragonswapStaker` has no revert error.
	
	—
	`rewardToken.safeTransfer(msg.sender, pendingRewards);`
	`emit Payout(msg.sender, pendingRewards);`
	
	`rewardsPaidOut += pendingRewards;`
	
	Within `withdraw`, the handling of `pendingRewards` should be transferred to the user only if `pendingRewards` is greater than 0 to save gas.

Recommendation	Consider fixing the typographical issues.
Resolution	RESOLVED

2.2 DragonswapStakerFactory

`DragonswapStakerFactory` handles the deployment of DragonswapStaker and DragonswapStakerBoosted. These contract instances are deployed as clones. Importantly, while clones are typically immutable, DragonswapStaker and DragonswapStakerBoosted are both upgradable contracts so the implementation contract can still be upgraded.

2.2.1 Privileged Functions

- `setImplementationClassic`
- `setImplementationBoosted`
- `deployClassic`
- `deployBoosted`

2.2.2 Issues & Recommendations

Issue #09	**`DragonswapStakerFactory` deploys contracts with `msg.value` and initializes instances with `msg.value`, but the `deploy` function is not `payable`**
Severity	LOW SEVERITY
Description	The `deploy` function attempts to send `msg.value` to the newly created clone instance. `if (data.length > 0) { (bool success, ) = instance.call{value: msg.value}(data); if (!success) revert(); }` However, `msg.value` is always `0` as both `deployClassic()` and `deployBoosted` are not `payable`. We believe that according to the use case of DragonStaker, native Ether does not need to be sent to the newly created instance, hence we can hardcode the value to 0.

Recommendation	Consider making the following change:
	`- (bool success, ) = instance.call{value: msg.value}(data);`
	`+ (bool success, ) = instance.call{value: 0}(data);`

Resolution	RESOLVED

Issue #10	Typographical issues
Severity	INFORMATIONAL
Description	Consider using `uint256` instead of `uint` for clarity and consistency throughout the contracts. — The `private` function naming variable should use `_deploy()` instead of `deploy()` to follow Solidity's naming convention. — Factory deploys contracts with `msg.value` but deployed contract instances do not have `receive()` function. Since we do not expect to send native ETH, consider cleaning up the code by changing `msg.value` -> 0.
Recommendation	Consider making the recommended changes mentioned above.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY