PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For DragonSwap

09 February 2024

paladinsec.co

info@paladinsec.co

Table of Contents

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 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.

The portion of DragonSwap within scope of this audit is a decentralized exchange (DEX) project. Users will be able to use the platform to trade cryptocurrencies in a non-custodial manner without an intermediary.

General risks on DEX projects include:

- Contract bugs that can lead to exploitation of miscalculated swap fees or balance calculation.
- Not much liquidity supplied causing cost and pricing instability.
- Front-running risks by trading bots.

This audit covers issues found within the DragonSwap DEX protocol and provides recommendations for mitigating them.

1.1 Summary

Project Name	DragonSwap
URL	https://dragonswap.app
Platform	SEI
Language	Solidity
Preliminary Contracts	https://github.com/dragonswap-app/dragonswap-core/commit/4856f6b4209e39ca18bf87aaff17a9877fab1346
Resolution 1	https://github.com/dragonswap-app/dragonswap-core/commit/0645fca253b22fd3cf5bd312e4d2a7064715392b

1.2 Contracts Assessed

Name	Contract	Live Code Match
DragonswapERC20		
DragonswapPair		
DragonswapFactory		
DragonswapRouter		

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● High	0	-	-	-
● Medium	0	-	-	-
● Low	0	-	-	-
● Informational	11	5	-	6
Total	**11**	**5**	**-**	**6**

Classification of Issues

Severity	Description
● 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 DragonswapERC20

ID	Severity	Summary	Status
01	INFO	Approval event is not emitted if allowance is changed in transferFrom as suggested in the ERC-20 Token Standard (also present in Uniswap)	✓ RESOLVED
02	INFO	`permit` can be frontrun to prevent someone from calling `removeLiquidityWithPermit` (also present in Uniswap)	ACKNOWLEDGED

1.3.2 DragonswapPair

ID	Severity	Summary	Status
03	INFO	`factory` can be made `immutable`	ACKNOWLEDGED
04	INFO	Only 10% of the swap fee goes to governance even though the documentation says it is 30%	✓ RESOLVED
05	INFO	`uint` is used within the whole contract	ACKNOWLEDGED

1.3.3 DragonswapFactory

ID	Severity	Summary	Status
06	INFO	Typographical issue	✓ RESOLVED
07	INFO	`feeToSetter` lacks a non-zero requirement in the constructor	✓ RESOLVED
08	INFO	`uint` is used within the whole contract	ACKNOWLEDGED
09	INFO	Lack of events for the governance functions: `setFeeTo` and `setFeeToSetter`	✓ RESOLVED

1.3.4 DragonswapRouter

ID	Severity	Summary	Status
10	INFO	The liquidity addition functions are slightly inefficient for tokens with a fee on transfer (also present in Uniswap)	ACKNOWLEDGED
11	INFO	Phishing is possible by a malicious frontend by adjusting routes, tokens or from parameters (also present in Uniswap)	ACKNOWLEDGED

1.3.5 Interfaces & Libraries

No issues found.

2 Findings

2.1 DragonswapERC20

`DragonswapERC20` is an implementation of the ERC20 token standard which represents a share value of a user's assets that is supplied in the liquidity pools. This contract is inherited by `DragonswapPair.sol` and it is a near perfect fork of UniswapV2's `UniswapV2ERC20` core contract.

2.1.1 Issues & Recommendations

Issue #01	**Approval event is not emitted if allowance is changed in `transferFrom` as suggested in the ERC-20 Token Standard (also present in Uniswap)**
Severity	● INFORMATIONAL
Description	The ERC-20 standard specifies that an approval event should be emitted when the allowance of a user changes. However, within the ERC20 implementation of both Uniswap and DragonSwap, this is not done. You can read more about this improvement in Pull Request #65 of uniswap-core.
Recommendation	Consider adding `emit Approval(from, msg.sender, remaining)` in `transferFrom` when allowance is modified
Resolution	✓ RESOLVED

Issue #02	`permit` can be frontrun to prevent someone from calling `removeLiquidityWithPermit` (also present in Uniswap)
Severity	● INFORMATIONAL
Description	If permit is executed twice, the second execution will revert. It is possible, in theory, for a bot to pick up `permit` transactions in the mempool and execute them before a contract can. The implications of this issue is that a bad actor could prevent a user from removing liquidity with a permit through the router. It is a denial of service attack which is present in all AMMs but which we have yet to witness being used since there is no profit from it.
Recommendation	Consider this issue if there are ever complaints by users that their `removeLiquidityWithPermit` transactions are failing. It could be the case that someone is using this vector against them. We do not recommend changing this behavior since it would cause a lot of extra work modifying the frontend to account for the new permit behavior. This issue is also present in Uniswap after all.
Resolution	● ACKNOWLEDGED

2.2 DragonswapPair

`DragonswapPair` keeps track of pool token balances by issuing ERC20 receipt tokens to users when they supply liquidity to the protocol or burns the tokens when withdrawing liquidity. The contract also works hand-in-hand with the periphery contracts of the DEX when supplying or withdrawing liquidity in the protocol, or when trading assets.

It is a near perfect fork of UniswapV2's `UniswapV2Pair` core contract but with a slight modification to the LP minting fee and adjusted balance calculation. The LP minting fee changed from ⅙ of the liquidity to 1/10th.

These changes need to have a corresponding update in the periphery contracts as well or several attack vectors could be opened up and used to drain liquidity from the protocol.

2.2.1 Issues & Recommendations

Issue #03	`factory` can be made `immutable`
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the `immutable` keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making `factory` `immutable`. Note that this might not work on the current version and should therefore likely be acknowledged as remaining consistent with Uniswap V2 might be more desirable than perfectly gas-optimized code.
Resolution	ACKNOWLEDGED

Issue #04	Only 10% of the swap fee goes to governance even though the documentation says it is 30%
Severity	INFORMATIONAL

Description	Line 88
	`// if fee is on, mint liquidity equivalent to 9/30th (0.3) of the growth in sqrt(k)`
	
	Even though the above documentation indicates that 30% of the swap fees go to governance, the code actually only grants 10% of those fees to governance due to a mismatch in the `mul` parameter:
	
	Line 99
	`uint denominator = rootK.mul(9).add(rootKLast);`

Recommendation	Consider adjusting the code to, for example: `rootK.mul(2).add(rootKLast);` This would adjust the fee to 1/3.
Resolution	✓ RESOLVED The factor has been adjusted so that the fee is actually 30%.

Issue #05	`uint` is used within the whole contract
Severity	● INFORMATIONAL
Description	We recommend remaining consistent and only use `uint256`. Being consistent shows to third-party validators that the code has been carefully thought through.
Recommendation	Consider using `uint256` throughout the contract.
Resolution	● ACKNOWLEDGED

2.3 DragonswapFactory

`DragonswapFactory` is responsible for keeping track of all existing liquidity pairs of the DEX and allows users to create new ones. Furthermore, the contract also stores the necessary addresses where the mint fee is sent to, and the governance address. This contract is a near perfect fork of UniswapV2's UniswapV2Factory core contract.

2.3.1 Privileged Functions

- `setFeeTo`
- `setFeeToSetter`

2.3.2 Issues & Recommendations

Issue #06	Typographical issue
Severity	INFORMATIONAL
Description	Line 16 `event PairCreated(address indexed token0, address indexed token1, address pair, uint);` The last parameter can be `length`.
Recommendation	Consider fixing the typographical issue.
Resolution	✓ RESOLVED

Issue #07	`feeToSetter` lacks a non-zero requirement in the constructor
Severity	INFORMATIONAL
Description	During the deployment of the contract, the deployer is able to add any address as the `feeToSetter` including a zero address. Having a zero address as the `feeToSetter` would practically disable the two governance functions: `setFeeTo` and `feeToSetter`.
Recommendation	To prevent this from happening by accident, consider adding a non-zero address requirement to the relevant function.
Resolution	✓ RESOLVED

Issue #08	`uint` is used within the whole contract
Severity	INFORMATIONAL
Description	We recommend remaining consistent and only use `uint256`. Being consistent shows to third-party validators that the code has been carefully thought through.
Recommendation	Consider using `uint256` throughout the contract.
Resolution	ACKNOWLEDGED

Issue #09	Lack of events for the governance functions: `setFeeTo` and `setFeeToSetter`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for these functions.
Resolution	✓ RESOLVED

2.4 DragonswapRouter

`DragonswapRouter` represents the user interface contract into the DragonSwap DEX core. It provides user-facing functions to add and remove liquidity, and execute swaps.

It is nearly identical to the Uniswap V2 router with the sole change being the renaming of ETH to SEI.

2.4.1 Issues & Recommendations

Issue #10	**The liquidity addition functions are slightly inefficient for tokens with a fee on transfer (also present in Uniswap)**
Severity	● INFORMATIONAL
Description	The DragonswapRouter supports adding liquidity to LPs where one or both of the tokens have a fee on trasfer. However, due to the way these functions are implemented in the original Uniswap codebase, this results in too much of the token with no fees being sent to the pair. This effectively causes the pair to incorporate those accidental extra tokens into the reserves.
Recommendation	Consider whether this is an issue. We can point to a GitHub PR with a proposed alternative function if desired. This proposed function wastes significant extra gas and does not exist within Uniswap so it might not be worth implementing.
Resolution	● ACKNOWLEDGED

Issue #11	**Phishing is possible by a malicious frontend by adjusting routes, tokens or `from` parameters (also present in Uniswap)**
Severity	● INFORMATIONAL
Description	A malicious (for example hacked) frontend can easily mislead users in approving malicious transactions, even if the router matches the address described in this report. An obvious example of how this can be done is by changing the `to` parameter which indicates to whom tokens or liquidity has to be sent. Other ways to phish could include using malicious routes or tokens.
Recommendation	Consider carefully protecting the frontend and ideally having an unchangeable IPFS fallback implementation for it.
Resolution	● ACKNOWLEDGED

2.5 Interfaces & Libraries

The interfaces and libraries included in the project are identical to the `UniswapV2` implementation. There are no significant differences between the two aside from the rebranding and the replacement of the `uint` variable with `uint256`.

2.5.1 Issues & Recommendations

No issues found.

PALADIN
BLOCKCHAIN SECURITY