PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For DragonSwap (Competition)

13 October 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's Competition 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/comp-contracts/commit/2a5af205b58b574bad1a93fa33c40b78a3df71e6
Resolution	https://github.com/dragonswap-app/comp-contracts/commit/4f618ad9af0f5370da28f31b57fdb1408a5c5092

1.2 Contracts Assessed

Name	Contract	Live Code Match
`Competition`		
`Factory`		

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
Governance	2	-	-	2
High	1	1	-	-
Medium	0	-	-	-
Low	8	4	-	4
Informational	7	6	-	1
Total	**18**	**11**	-	**7**

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 Competition

ID	Severity	Summary	Status
01	GOV	Governance privileges	ACKNOWLEDGED
02	HIGH	Users can drain the competition by forging input parameters of the swaps	✓ RESOLVED
03	LOW	Rebase tokens or tokens with a fee on transfer do not work	ACKNOWLEDGED
04	LOW	Reentrancy risks	✓ RESOLVED
05	LOW	Users can trade outside the competition timeframe	✓ RESOLVED
06	LOW	Users cannot forfeit their stuck tokens to rejoin	ACKNOWLEDGED
07	LOW	ETH sent to the protocol can get stuck	✓ RESOLVED
08	LOW	Exiting could be problematic if `swapTokens` array gets too large	ACKNOWLEDGED
09	LOW	Exit on some tokens might always revert due to usage of `transfer` instead of `safeTransfer`	✓ RESOLVED
10	LOW	Leftover validation when exiting is insufficient	ACKNOWLEDGED
11	INFO	Incorrect check in the `_toAddress`	✓ RESOLVED
12	INFO	Peculiar address used as swap token placeholder	✓ RESOLVED
13	INFO	Typographical issues	✓ RESOLVED
14	INFO	Use `Ownable2Step` as a safer alternative to `Ownable`	✓ RESOLVED
15	INFO	No mechanism to remove swap tokens	ACKNOWLEDGED
16	INFO	Insufficient validation	✓ RESOLVED

1.3.2 Factory

ID	Severity	Summary	Status
17	GOV	Governance Privileges	ACKNOWLEDGED
18	INFO	Typographical issues	✓ RESOLVED

2 Findings

2.1 Competition

`Competition` implements a token swapping competition that lets the participants swap tokens using various paths defined by the owner in the swap tokens.

Users can deposit only if the competition is active but they can exit at any point in time which will make him ineligible for participation until they enter again.

2.1.1 Privileged Functions

- `addSwapTokens`
- `transferOwnership`
- `renounceOwnership`

2.1.2 Issues & Recommendations

Issue #01	Governance privileges
Severity	GOVERNANCE
Description	The contract owner has full control over several actions that can impact the outcome of a transaction. E.g. Add swap tokens.
Recommendation	Consider incorporating a Gnosis multi-signature contract as the owner and ensuring that the Gnosis participants are trusted entities.
Resolution	ACKNOWLEDGED The team will use a multi-signature wallet to manage to contract once Gnosis is available on SEI.

Issue #02	Users can drain the competition by forging input parameters of the swaps
Severity	HIGH SEVERITY
Description	During swaps competitions, users must swap assets deposited by them within the competition contract. With each swap the balances of `tokenIn` and `tokenOut` changes. `function _noteSwap(address _tokenIn, address _tokenOut, uint256 _amountIn, uint256 _amountOut, SwapType _swapType) private { // Decrease _tokenIn balance balances[msg.sender][_tokenIn] -= _amountIn; // Increase _tokenOut balance. balances[msg.sender][_tokenOut] += _amountOut; // Emit event. emit NewSwap(msg.sender, _tokenIn, _tokenOut, _amountIn, _amountOut, _swapType); }`  When the user wishes to exit the competition, they can call the `exit` function which will iterate through all the balances of the user and transfer them out.  The problem lies in the swap functions for V2—the inputs have a receiver parameter which is meant to transfer the `out` tokens but this parameter is not sanitized to always be `address(this)`, which means that a user can forge the input parameters and put as the receiver an address of their choice, getting the funds out of the contract. They can keep doing this until the contract is fully drained. Furthermore, they can also exit their positions if they do not wish to drain only via the swaps.
Recommendation	Consider always overwriting the input parameters and set the receiver as `address(this)`.
Resolution	RESOLVED The recipient has been overridden to `address(this)`.

Issue #03	Rebase tokens or tokens with a fee on transfer do not work
Severity	LOW SEVERITY
Description	Throughout the contract, the amounts are stored as received from the input and does not take into consideration that the token might rebase or have fee on transfers. This will cause imbalances in the amounts if such tokens are used.
Recommendation	Consider not using such tokens. If such tokens must be used, then consider caching the balance before/after of every transfer to get the actual amount that was transferred in.
Resolution	ACKNOWLEDGED The team stated: "We do not intend to support such types of tokens as the V2 protocol does not support them."

Issue #04	Reentrancy risks
Severity	LOW SEVERITY
Description	Throughout the contract, several external calls are performed before the state update, some of which are inevitable due to the nature of the logic, e.g. during swaps, the amount in/out may need to be retrieved from the external call so it can be updated correctly. However, others can be avoided, e.g. during exit. If by any chance an exploiter were able to call custom code in that external call, then the contract can be potentially drained.
Recommendation	Consider adding a reentrancy guard to all the public/external functions. Furthermore consider moving the call in exit.
Resolution	RESOLVED All the functions now have a reentrancy guard.

Issue #05	Users can trade outside the competition timeframe
Severity	LOW SEVERITY

Description	During swap competitions, users must swap assets deposited by them within the `Competition` contract. There are two checks that are done when a user wants to swap:
	- `onceOn`
	- `notOut`
	`onceOn` checks that the competition has started and `notOut` checks that the user did not leave the competition.
	`onceOn` performs the following check:
	`function _isOnCheck() private view {`
	`  if (block.timestamp < startTimestamp) revert NotOnYet();`
	`}`
	As seen above, it only checks that the has competition started but not if it has ended, which will allow users to trade outside the competition timeframe.
	We marked this as low because the DragonSwap team will snapshot and reward the users at the end of the competition so any trade outside of the competition time will be obsolete.

Recommendation	Consider checking that that the competition has not ended: `if (block.timestamp > endTimestamp) revert Finished();`
Resolution	RESOLVED

Issue #06	Users cannot forfeit their stuck tokens to rejoin
Severity	LOW SEVERITY
Description	When a user calls `exit()`, the function attempts to transfer all their token balances back to them. If any transfer succeeds, `madeWithdrawal` is set to `true`. If any transfer fails, `leftoverExists` is set to `true`. If `madeWithdrawal` is `true` and `isOut[msg.sender]` is `false`, the user is marked as `isOut[msg.sender] = true`. If there are no leftovers (`leftoverExists` is `false`), `isOut[msg.sender]` is reset to `false`, allowing the user to rejoin the competition. If any of the user's token transfers fail (e.g., due to a faulty token contract), `leftoverExists` remains `true`, and `isOut[msg.sender]` remains `true`. The `notOut` modifier on other functions like `deposit` and `swap` prevents the user from participating while `isOut[msg.sender]` is `true`. This means the user cannot deposit new funds or perform swaps even if they have successfully withdrawn other tokens and wish to continue participating.
Recommendation	Consider allowing the user to specify on the `exit` function that they can forfeit in case of failure. This will zero its balance also for the tokens that fail to withdraw. Furthermore, an `onlyOwner` function should be implemented to let the owner zero all the balances of the contract after a certain period of time after the competition ends to prevent funds getting stuck in the contract. E.g. An `onlyOwner` function that can withdraw any ERC20 token from the contract could be activated after 3 months of the competition ending, leaving the users enough time to exit their positions.
Resolution	ACKNOWLEDGED The team stated: "This is intentional in order not to make the exit flow too elaborate. Users can always join using a new wallet and the effect would be the same (without discarding their own funds)."

Issue #07	ETH sent to the protocol can get stuck
Severity	LOW SEVERITY
Description	The payable modifier is added to the following functions: - `swapExactTokensForTokens()` - `swapTokensForExactTokens()` - `exactInputSingle()` - `exactInput()` - `exactOutputSingle()` - `exactOutput()` However, the contract operates only with regular tokens and not with ETH directly. So if any ETH is sent to the contract, it will get stuck.
Recommendation	Consider removing the `payable` modifier.
Resolution	RESOLVED

Issue #08	Exiting could be problematic if swapTokens array gets too large
Severity	LOW SEVERITY
Description	The `exit()` function is used by depositors in the Competition contract to take out their balances. The function loops through the `swapTokens` array and transfers the balances. The array is modified by the owner and can only grow over time. If the array ever gets too large and gas prices jump too high, exiting might be DOSed or be too expensive to execute.
Recommendation	Consider providing start and end indexes to allow the caller to control which deposits to take out. If this change is not compatible with the expected behavior of the contract, make sure to closely monitor the size of the array so it does not get too large.
Resolution	ACKNOWLEDGED

Issue #09	**Exit on some tokens might always revert due to usage of transfer instead of `safeTransfer`**
Severity	LOW SEVERITY
Description	The `exit()` function is used by depositors in the Competition contract to take out their balances. It loops through every token and transfers the balance if the caller has any balances. The transfer is executed through a low-level call, which is checked for success. The issue is that success will always equal to `false` (even if transfer is successful) for some non-standard ERC implementation such as USDT. The case is especially relevant here since the protocol uses stablecoins (`stable0` and `stable1`) and if USDT is used it can be quite problematic, leading to double spending.
Recommendation	Use `safeTransfer` wrapped in try/catch block as reliable alternative to the current implementation
Resolution	RESOLVED

Issue #10	Leftover validation when exiting is insufficient
Severity	LOW SEVERITY

Description	When exiting, all of the balances of the caller are transferred out. In case some transfer fails, the `isOut` mapping is set to `true`, locking the depositor from rejoining the competition. This is the logic used:
	`if (madeWithdrawal && !isOut[msg.sender]) {`
	`  isOut[msg.sender] = true;`
	`  emit Exit(msg.sender);`
	`}`
	
	`if (!leftoverExists) {`
	`  isOut[msg.sender] = false;`
	`}`
	
	An edge case here is if all transfers fail, the first `if` branch will never activate and `isOut` will remain `false`, and the second check will also be skipped.
	
	The result is that even though no transfer succeeded, the depositor has not been blocked from participation.

Recommendation	Consider adding the following check to ensure the caller is blocked every time there is leftover:
	
	`if (leftoverExists) {`
	`  isOut[msg.sender] = true;`
	`}`

Resolution	ACKNOWLEDGED The team stated: "This is intentional behavior. If user is blocked from leaving then there is no benefit from disabling his participation too."

Issue #11	Incorrect check in the `_toAddress`
Severity	INFORMATIONAL
Description	The `_toAddress` function in the Utils library is used to extract an address from a bytes array. The first check is incorrect as it should check against the bytes array length. `if (_start > type(uint256).max - 20) revert ToAddressOverflow();`
Recommendation	Consider replacing it with `if (_start >_bytes.length - 20) revert ToAddressOverflow();`
Resolution	RESOLVED

Issue #12	Peculiar address used as swap token placeholder
Severity	INFORMATIONAL
Description	In order to avoid zero value being a swap token id, the `swapTokens` array is first initialized with the hex value of the string `firstslotplaceholder`. While this has a low chance of colliding with any real token that will be deployed in the future, the risk might still be there.
Recommendation	Consider using `0xded` instead.
Resolution	RESOLVED

Issue #13	Typographical issues
Severity	INFORMATIONAL
Description	Competition contract: Line 120 "existance" should be "existence". Line 216 "Perfrom" should be "Perform". — ICompetition interface: Line 17 "occuring" should be "occurring". Line 36 "alredy" should be "already". Line 38 "depsit" should be "deposit". — Utils library: Line 14 "substitues" should be "substitutes". — The `initialize` function should emit `SwapTokenAdded` event for `stable0` and `stable1` as well.
Recommendation	Consider resolving all the typographical issues
Resolution	RESOLVED

Issue #14	Use `Ownable2Step` as a safer alternative to `Ownable`
Severity	INFORMATIONAL
Description	The contract uses `OwnableUpgradeable` for access control. A safer alternative is to use `Ownable2StepUpgradeable`—instead of directly transferring to the new owner, the transfer only completes when the new owner accepts ownership.
Recommendation	Consider using `Ownable2StepUpgradeable` instead of `OwnableUpgradeable`. This makes ownership transfers much safer and prevents accidental transfers to invalid addresses. Note: If you decide to go with `Ownable2StepUpgradeable` make sure to call `__Ownable_init(newOwner)` in the initializer, because since v 5.0 `Ownable2StepUpgradeable` does not call it in `__Ownable2Step_init()` and would leave the contract without an owner.
Resolution	RESOLVED

Issue #15	No mechanism to remove swap tokens
Severity	INFORMATIONAL
Description	The contract uses the `swapTokens` array state variable to store each token can be used for swaps. The admin can add new tokens through the `addSwapTokens()` function. The issue is that tokens can only be added but cannot be removed. This might be needed for example when the token is problematic (malicious, not-working properly etc.) or if the array becomes too large and gas inefficient to loop through.
Recommendation	Consider adding an additional function that the owner can call to remove tokens. Also consider using OZ's `enumerableSet` instead of plain arrays which will make adding and removing elements faster and cheaper. Note: Adding this mechanism would also require changes to the exit function to ensure depositors' funds do not get stuck.
Resolution	ACKNOWLEDGED The team stated: "Removing swap tokens could cause a potential disruption in the competitor strategies. Introduction of such a feature would increase the complexity of the flow as well. For now we will go without it. (multi-sig wallet should add tokens so it should be done very carefully + we don't see any major risk coming from this for the moment.)"

Issue #16	Insufficient validation
Severity	INFORMATIONAL
Description	Within all swap functions, consider checking that `path.length > 1` and revert early if that is not the case. Within all swap functions, check if `amountIn` is not 0.
Recommendation	Consider implementing the above recommendations.
Resolution	RESOLVED

2.2 Factory

`Factory` handles the creation of competitions—only the owner of this contract can create competitions by specifying custom parameters. The competitions are stored then into storage to serve them for third parties.

2.2.1 Privileged Functions

- `setImplementation`
- `deploy`
- `transferOwnership`
- `renounceOwnership`

2.2.2 Issues & Recommendations

Issue #17	Governance Privileges
Severity	GOVERNANCE
Description	The contract owner has full control over several actions that can impact the outcome of a transaction. - set implementation for deployed contracts - deploy new competition contracts
Recommendation	Consider incorporating a Gnosis multi-signature contract as the owner and ensuring that the Gnosis participants are trusted entities.
Resolution	ACKNOWLEDGED The team will use a multi-signature wallet to manage to contract once Gnosis is available on SEI.

Issue #18	Typographical issues
Severity	INFORMATIONAL
Description	Line 71 `allSales` should be `deployments` — Within the `Competition` contract, the tokens are called `stable0` and `stable1`, but within the `Factory`, they are called `usdc` and `usdt`. Consider renaming the ones in `Factory` to match the ones in `Competition`.
Recommendation	Consider resolving all the typographical issues.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY