

PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For D2

20 March 2023



paladinsec.co



info@paladinsec.co

Table of Contents

Table of Contents 2

Disclaimer 4

1 Overview 5

1.1 Summary 5

1.2 Contracts Assessed 5

1.3 Findings Summary 6

1.3.1 DSQStaking 7

1.3.2 DSQToken 7

1.3.3 esDSQToken 7

1.3.4 NFTStaking 8

1.3.5 Router 8

1.3.6 StakingRewards 9

1.3.7 esDSQStaking 9

2 Findings 10

2.1 DSQStaking 10

2.1.1 Privileged Functions 11

2.1.2 Issues & Recommendations 12

2.2 DSQToken 22

2.2.1 Token Overview 22

2.2.2 Privileged Functions 22

2.2.3 Issues & Recommendations 23

2.3 esDSQToken 24

2.3.1 Token Overview 24

2.3.2 Privileged Functions 25

2.3.3 Issues & Recommendations 26

2.4 NFTStaking 28

2.4.1 Privileged Functions 29

2.4.2 Issues & Recommendations 30
2.5 Router 37
2.5.1 Privileged Functions 37
2.5.2 Issues & Recommendations 38
2.6 StakingRewards 39
2.6.1 Privileged Functions 39
2.6.2 Issues & Recommendations 40
2.7 esDSQStaking 47
2.7.1 Privileged Functions 47
2.7.2 Issues & Recommendations 48

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 D2 on the Arbitrum 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	D2
URL	https://d2.finance/
Platform	Arbitrum
Language	Solidity
Preliminary Contracts	https://github.com/D-Squared-Tech/solidity/tree/9ec491b41d9cd069d0adde53a96fc3ab22a30de4/contracts

1.2 Contracts Assessed

Name	Contract	Live Code Match
DSQStaking	0x866d31bf2e3b6bf152B2f8a51807219581324A7C	MATCH
DSQToken	0xdb0c6fc9e01cd95eb1d3bbae6689962de489cd7b	MATCH
esDSQToken	0xf76d53de08c53b891e291d870e543bf8a0f6d314	MATCH
NFTStaking	Not deployed	UNDEPLOYED
Router	0x0FEcAcd06304CDe4b3b94073f71A52fAFF8D5410	MATCH
StakingRewards	Not deployed	UNDEPLOYED
esDSQStaking	0x646974510e5443491bEE2E577d33e979C8A7E983	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	4	4	-	-
Medium	3	2	-	1
Low	5	4	-	1
Informational	27	15	1	11
Total	**39**	**25**	**1**	**13**

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 with 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 DSQStaking

ID	Severity	Summary	Status
01	HIGH	`rewardsToken` and `stakingToken` can be the same	✓ RESOLVED
02	LOW	If set to zero, `rewardsDuration` reverts due to division by zero	✓ RESOLVED
03	INFO	`recoverERC20` can be used to recover `stakingToken` if it has multiple contract entry points	ACKNOWLEDGED
04	INFO	`exit` cannot be called for an account that has no pending rewards	✓ RESOLVED
05	INFO	Staking tokens with a fee on transfer are not supported	ACKNOWLEDGED
06	INFO	`setRewardsDistribution` lacks event emission	✓ RESOLVED
07	INFO	General lack of validation	✓ RESOLVED
08	INFO	`notifyRewardAmount` can dilute existing rewards if the reward parameter is small and there are few remaining rewards	ACKNOWLEDGED
09	INFO	Arithmetic operations can be simplified	✓ RESOLVED
10	INFO	Gas optimizations	✓ RESOLVED

1.3.2 DSQToken

ID	Severity	Summary	Status
11	INFO	DSQToken can be burned by anyone	ACKNOWLEDGED

1.3.3 esDSQToken

ID	Severity	Summary	Status
12	LOW	`router` and `esDSQStaking` must be added to `permanentWhitelist`	ACKNOWLEDGED
13	INFO	No functionality to batch add/remove addresses	✓ RESOLVED
14	INFO	Users cannot easily access addresses which are permanently whitelisted	✓ RESOLVED
15	INFO	Gas optimization	✓ RESOLVED

1.3.4 NFTStaking

ID	Severity	Summary	Status
16	HIGH	withdraw uses real time weighted amount instead of weighted amount	✓ RESOLVED
17	MEDIUM	Contract is not pausable	✓ RESOLVED
18	LOW	`setTokenWeights` does not ensure that a higher class will always have a greater weight than a lower class	✓ RESOLVED
19	LOW	If `rewardsDuration` is set to zero, it will cause a divide by zero revert	✓ RESOLVED
20	INFO	`getWeightedAmount` will return a `weightedAmount` of 0 for some token ids	ACKNOWLEDGED
21	INFO	For NFT transfers, `transferFrom` instead of `safeTransferFrom` can be used to reduce gas costs	ACKNOWLEDGED
22	INFO	`whenNotPaused` modifier is unnecessary for `stakeMultiple`	✓ RESOLVED
23	INFO	`setRewardsDistribution` lacks event emission	✓ RESOLVED
24	INFO	`notifyRewardAmount` can dilute existing rewards if the reward parameter is small and there are few remaining rewards	ACKNOWLEDGED
25	INFO	Lack of validation	✓ RESOLVED
26	INFO	Typographical errors	ACKNOWLEDGED

1.3.5 Router

ID	Severity	Summary	Status
27	INFO	Typographical errors	✓ RESOLVED
28	INFO	Rewards can be claimed even when router is paused	ACKNOWLEDGED

1.3.6 StakingRewards

ID	Severity	Summary	Status
29	HIGH	`rewardsToken` and `stakingToken` can be the same	✓ RESOLVED
30	MEDIUM	Contract is not pausable	✓ RESOLVED
31	MEDIUM	Staking tokens with a fee on transfer are not supported	ACKNOWLEDGED
32	LOW	If set to zero, rewardsDuration reverts due to division by zero	✓ RESOLVED
33	INFO	`recoverERC20` can be used to recover `stakingToken` if it has multiple contract entry points	ACKNOWLEDGED
34	INFO	`setRewardsDistribution` lacks event emission	✓ RESOLVED
35	INFO	`notifyRewardAmount` can dilute existing rewards if the reward parameter is small and there are few remaining rewards	ACKNOWLEDGED
36	INFO	Arithmetic operations can be simplified	✓ RESOLVED

1.3.7 esDSQStaking

ID	Severity	Summary	Status
37	HIGH	Admin stake will most likely not work as expected	✓ RESOLVED
38	INFO	Lack of validation	PARTIAL
39	INFO	Lack of `safeTransfer` usage	✓ RESOLVED

2 Findings

2.1 DSQStaking

`DSQStaking` is a simple staking contract forked from the known Synthetix `stakingRewards` contract (https://github.com/Synthetixio/synthetix/blob/develop/contracts/StakingRewards.sol).

Users can stake the `stakingToken` and earn rewards in form of the `rewardsToken`. Rewards are distributed amongst all stakers based on their staked balance and can be configured only by the `rewardsDistribution` contract via `notifyRewardAmount`, which simply allocates the desired reward amount over the `rewardsDuration`.

Unlike most traditional staking contracts, the `stake`, `withdraw`, `getRewards` and `exit` functions can only be called via the `router` contract and not directly by any user.

2.1.1 Privileged Functions

Owner

- setRouter (only once)
- setRewardsDistribution
- setRewardsDuration
- recoverERC20
- transferOwnership
- renounceOwnership

Router

- stake
- withdraw
- getReward
- exit

RewardsDistribution

- notifyRewardAmount

2.1.2 Issues & Recommendations

Issue #01	`rewardsToken` and `stakingToken` can be the same
Severity	HIGH SEVERITY
Description	In the case where the `rewardsToken` is the same as the `stakingToken`, `rewardsToken.balanceOf(address(this))` would return the sum of `stakingToken` and `rewardsToken`. This allows the `RewardsDistribution` to set a much higher reward rate using `notifyRewardAmount` based on a much higher perceived `rewardsToken` balance. Over time, the higher rate would exceed the `rewardsToken` balance and eat into the `stakingToken` balance, resulting in insufficient token balances for all users to withdraw. Additionally, another side effect of both tokens being the same contract address would mean that `recoverERC20` cannot recover the reward token.
Recommendation	Consider adding a check in the constructor to prevent both tokens from being the same address. `require(_rewardsToken != _stakingToken, "Same token");`
Resolution	RESOLVED

Issue #02	If set to zero, `rewardsDuration` reverts due to division by zero
Severity	LOW SEVERITY

Location	L119
	`rewardRate = reward / rewardsDuration;`
	
	L123
	`rewardRate = (reward + leftover) / rewardsDuration;`
	
	L131
	`require(rewardRate <= balance / rewardsDuration, "DSQStaking: Provided reward too high");`

Description	Within `notifyRewardAmount`, `rewardsDuration` is used as the denominator for some calculations as shown in the above examples. If this variable is set to zero, it would cause a divide by zero revert, thus not allowing the new reward emission state to be updated.
Recommendation	Consider adding a non-zero check in `setRewardsDuration` to ensure that `rewardsDuration` cannot be set to zero. `require(_rewardsDuration != 0, “No zero duration”);`
Resolution	RESOLVED

Issue #03	**`recoverERC20` can be used to recover `stakingToken` if it has multiple contract entry points**
Severity	INFORMATIONAL
Description	For a token such as ERC20 TUSD (on ETH mainnet) which has more than one entry point to transfer tokens, the secondary entry point's contract address can be used as the recovery token, thus allowing the owner to withdraw the staking tokens, which should not be allowed. https://medium.com/chainsecurity/trueusd-compound-vulnerability-bc5b696d29e2 This issue has been categorized as informational, as the intended staking token would be DSQ token which does not have multiple entry points.

Recommendation	Ensure that there is no change in the staking contract's staking token when `recoverERC20` is called.
	
	`uint256 before = stakingToken.balanceOf(address(this));`
	`IERC20(tokenAddress).safeTransfer(owner(), tokenAmount);`
	`uint256 after = stakingToken.balanceOf(address(this)) - before;`
	`require( after == before, "stakingToken changed");`

Resolution	ACKNOWLEDGED

Issue #04	`exit` cannot be called for an account that has no pending rewards
Severity	INFORMATIONAL
Description	Due to this `require` statement in `getReward`, the function would revert if the account has no rewards. `require(reward > 0, "DSQStaking: No reward");` This means a user who has 0 pending rewards (e.g. staking after a reward period has ended) will be unable to call the `exit` function. The user will still be able to get the staked balance back by using withdraw, so there are no funds at risk of being stuck.

Recommendation	We recommend following the original logic in Staking Rewards to only send the `rewardsToken` out and emit the event if the user has a non-zero pending reward.
	`if (reward > 0) {`
	`  rewards[account] = 0;`
	`  rewardsToken.safeTransfer(recipient, reward);`
	`  emit RewardPaid(account, recipient, reward);`
	`}`

Resolution	RESOLVED

Issue #05	Staking tokens with a fee on transfer are not supported
Severity	INFORMATIONAL

Location	L87~
	`_totalSupply = _totalSupply + amount;`
	`_balances[account] = _balances[account] + amount;`
	
	`stakingToken.safeTransferFrom(router, address(this), amount);`

Description	`amount` is used to increase the `totalSupply` and user's staked balance. If the staking token has some form of fee on transfer which results in the actual token amount received by the contract being less than the `amount` parameter, it would result in an irregularity between the internal account and actual token balances. This issue has been categorized as informational, as the intended staking token would be DSQ token, which does not have any fee on transfer functionality.
Recommendation	In this specific case, since `stake` can only be called by `router`, and `router` does not support fee on transfer tokens too, the `safeTransferFrom` would revert due to the router having insufficient balance. This issue can be acknowledged, but if in future any token with a fee on transfer is used, both the router and staking contract will need to use the actual amount received by calculating after - before.
Resolution	ACKNOWLEDGED

Issue #06	`setRewardsDistribution` lacks event emission
Severity	INFORMATIONAL
Description	`setRewardsDistribution` changes the `rewardsDistribution` address, but does not emit an event.
Recommendation	Consider adding an event to emit the old and new `rewardsDistribution` address for easier querying on when `rewardsDistribution` has been changed.
Resolution	RESOLVED

Issue #07	General lack of validation
Severity	INFORMATIONAL

Description	The contract has multiple sections where a validation is missing, especially sections that will break the contract if the address is set to the zero address:
	
	Line 46
	`constructor(address _owner, address _rewardsDistribution, address _rewardsToken, address _stakingToken)`
	
	`_rewardsToken` and `_stakingToken` should have an `address(0)` check.
	
	Line 145
	`function setRewardsDistribution(address _rewardsDistribution)`
	
	The `_rewardsDistribution` address should have an `address(0)` check.
	
	Line 158
	`function setRouter(address _router)`
	
	The `_router` should have an `address(0)` check.

Recommendation	Consider validating the sections accordingly.
Resolution	RESOLVED

Issue #08	**`notifyRewardAmount` can dilute existing rewards if the reward parameter is small and there are few remaining rewards**
Severity	INFORMATIONAL

Description	If the current reward period has not finished and there are few remaining rewards, calling `notifyRewardAmount` with a small reward parameter can cause a dilution of the new `rewardRate` as `rewardsDuration` is used as the denominator.
	`uint256 remaining = periodFinish - block.timestamp;`
	`uint256 leftover = remaining * rewardRate;`
	`rewardRate = (reward + leftover) / rewardsDuration;`

Recommendation	The `rewardsDistribution` role should take care in calling the `notifyRewardAmount` with a suitable amount.
Resolution	ACKNOWLEDGED

Issue #09	Arithmetic operations can be simplified
Severity	INFORMATIONAL

Description	There are several arithmetic operations that can be simplified in order to keep the contract more readable.
	
	Line 87
	`_totalSupply = _totalSupply + amount;`
	`Simply use _totalSupply += amount.`
	
	Line 88
	`_balances[account] = _balances[account] + amount;`
	
	Simply use `_balances[account] += amount`.
	
	The same applies for the `withdraw` function — consider using the `-=` pattern.

Recommendation	Consider implementing the suggested changes.
Resolution	RESOLVED

Issue #10	Gas optimizations
Severity	INFORMATIONAL

Description	The contract contains a few sections which can be optimized in order to save gas.
	
	Line 151
	`emit RewardsDurationUpdated(rewardsDuration);`
	
	Consider emitting the parameter variable instead of fetching the variable from storage.
	
	Line 160
	`emit NewRouter(router, _router);`
	
	Since the `router` will always be `address(0)`, consider emitting `address(0)` or simply remove the `router` parameter completely.

Recommendation	Consider following the recommendations in order to save gas.
Resolution	RESOLVED

2.2 DSQToken

`DSQToken` is a simple ERC20 implementation which is mintable by the owner with a `MAX_SUPPLY` of `500_000` tokens. It is extended by the ERC20 `burnable` extension which allows anyone to burn their tokens or burn tokens if the allowance allows it.

2.2.1 Token Overview

Address	TBD
Token Supply	500_000
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	None
Pre-mints	TBD

2.2.2 Privileged Functions

- `mint`
- `transferOwnership`
- `renounceOwnership`

2.2.3 Issues & Recommendations

Issue #11	DSQToken can be burned by anyone
Severity	INFORMATIONAL

Description	As `DSQToken` inherits `ERC20Burnable` as is without overriding any of its functions, anyone can burn their own DSQ tokens or another address's tokens if granted the allowance to do so.
	
	`function burn(uint256 amount) public virtual {`
	`  _burn(_msgSender(), amount);`
	`}`
	
	`function burnFrom(address account, uint256 amount) public virtual {`
	`  _spendAllowance(account, _msgSender(), amount);`
	`  _burn(account, amount);`
	`}`

Recommendation	There appears to be no usage or references of the `burn` functionality of the DSQToken within the codebase. Confirm if it is indeed required. Otherwise, consider removing the `burn` functionality.
Resolution	ACKNOWLEDGED

2.3 esDSQToken

esDSQToken is a simple ERC20 token with a whitelist feature and access control.

The DEFAULT_ADMIN_ROLE can restrict transfers, which means that only whitelisted addresses can transfer tokens. Addresses can be permanently or temporarily whitelisted by the admin, which then will be stored in whitelist. Addresses which have been permanently added can never be removed again, whereas other addresses can be removed via removeWhitelistAddress.

The token also implements the ERC20 burnable extension and allows the MINTER_ROLE to mint tokens up to a MAX_SUPPLY of 250_000 tokens. The DEFAULT_ADMIN_ROLE can add as many minters as desired.

2.3.1 Token Overview

Address	TBD
Token Supply	500_000
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	None
Pre-mints	TBD

2.3.2 Privileged Functions

- `mint`
- `setRestrictedTransfer`
- `addWhitelistAddress`
- `removeWhitelistAddress`
- `transferOwnership`
- `renounceOwnership`

2.3.3 Issues & Recommendations

Issue #12	`router` and `esDSQStaking` must be added to `permanentWhitelist`
Severity	LOW SEVERITY
Description	If `router` and `esDSQStaking` are not added to `permanentWhitelist`, the `DEFAULT_ADMIN_ROLE` can call `removeWhitelistAddress` to remove them, thus preventing the staking or withdrawal of any staked `esDSQ`. This also applies to any LP and the swap router as well in order to ensure correct swap functionality.
Recommendation	Ensure that `router` and `esDSQStaking` and any LP pair contract addresses are added with `_permanent` as `true`.
Resolution	ACKNOWLEDGED

Issue #13	No functionality to batch add/remove addresses
Severity	INFORMATIONAL
Description	Currently, the contract has no functionality to add and remove addresses in batches. This can become inefficient and gas-expensive if the admin wants to add/remove a lot of addresses
Recommendation	Consider implementing batch add/remove functionality in order to save gas and increase comfortability.
Resolution	RESOLVED

Issue #14	Users cannot easily access addresses which are permanently whitelisted
Severity	INFORMATIONAL
Description	Currently, users can not easily view which addresses are permanently whitelisted. As this might be valuable for users to inspect, it makes sense to add logic for that.
Recommendation	Consider adding all addresses which are permanently whitelisted to an `enumerableSet`.
Resolution	RESOLVED

Issue #15	Gas optimization
Severity	INFORMATIONAL
Description	Within `setRestrictedTransfer`, the event emits two parameters; however, due to the function logic, it is only necessary to emit the latter parameter.
Recommendation	Consider removing the first parameter in order to circumvent the storage access and save gas.
Resolution	RESOLVED

2.4 NFTStaking

`NFTStaking` allows users to stake NFTs in order to receive rewards. Users can stake two different NFTs which are determined by `stakingToken1` and `stakingToken2`. Each ID has their own weight, which results in more rewards for rarer IDs compared to common IDs.

For `stakingToken1`, the following IDs will have the corresponding weights:

- 0-2: `GENESIS_LEGENDARY_WEIGHT`
- 3-12: `GENESIS_RARE_WEIGHT`
- 13-32: `GENESIS_UNCOMMON_WEIGHT`

For `stakingToken2`, the following IDs will have the corresponding weights:

- 0-10: `NORMAL_RARE_WEIGHT`
- 11-190: `NORMAL_UNCOMMON_WEIGHT`
- 191-2190: `NORMAL_COMMON_WEIGHT`

The contract owner can determine the `rewardRate` and `rewardDuration` freely while the contract balance must cover the desired reward for the whole duration, except when the owner calls `withdrawERC20` with the `rewardToken` as parameter.

Additionally, the contract owner retains the privilege to change the aforementioned token weights without any limitation when calling `setTokenWeights`. By default, the weights range from 6e18 to 1e18. For IDs outside of the aforementioned ranges, the weights will always be zero. The contract is also pausable, which means that staking can be temporarily or permanently disabled while withdrawals will always work.

2.4.1 Privileged Functions

- `transferOwnership`
- `renounceOwnership`
- `notifyRewardAmount`
- `recoverERC20`
- `recoverERC721`
- `setRewardsDuration`
- `setRewardsDistribution`
- `setTokenWeights`

2.4.2 Issues & Recommendations

Issue #16	**`withdraw` uses real time weighted amount instead of weighted amount**
Severity	HIGH SEVERITY
Description	`setTokenWeights` allows the contract owner to change the weights of the various classes. These weight amounts are then returned based on the token and token id in `getWeightedAmount`. In the case where a user stakes a token with a weight amount of X and `setTokenWeights` is called later to increase the weight amount of that token and token id to 2X, 2X instead of X will be deducted from `_totalSupply` and the user's value in the `_balances` mapping if the user withdraws that staked token. This can cause a revert due to integer underflow (X - 2X), and the user will be unable to withdraw the staked token. Alternatively, if the weight amount is changed to X/2, while the user will be able to withdraw the token, the user will still be left with X/2 `balance`, and `_totalSupply` would have been wrongly reduced by X/2 instead of X. This causes a dilution of rewards for all staking users.
Recommendation	Consider recording the weight amount at the point of staking, and that weight amount should be used during the subtraction of total and user balance during withdrawal.
Resolution	RESOLVED

Issue #17	Contract is not pausable
Severity	MEDIUM SEVERITY
Description	The contract inherits the `Pausable` library from OpenZeppelin and expects `stake` and `stakeMultiple` to only be called if the contract is not paused. However, the contract does not expose any external pause and unpause functions which effectively prevents the pause feature from being used.
Recommendation	Consider adding external pause and unpause functions.
Resolution	RESOLVED

Issue #18	`setTokenWeights` does not ensure that a higher class will always have a greater weight than a lower class
Severity	LOW SEVERITY

Description	The following are the default weights for the various token classes
	`uint256 public GENESIS_LEGENDARY_WEIGHT = 6e18;`
	`uint256 public GENESIS_RARE_WEIGHT = 5e18;`
	`uint256 public GENESIS_UNCOMMON_WEIGHT = 4e18;`
	`uint256 public NORMAL_RARE_WEIGHT = 3e18;`
	`uint256 public NORMAL_UNCOMMON_WEIGHT = 2e18;`
	`uint256 public NORMAL_COMMON_WEIGHT = 1e18;`
	However, within `setTokenWeights`, the owner can pass any values in any order. This means that a `GENESIS_LEGENDARY_WEIGHT` can have a lower weight than `GENESIS_RARE_WEIGHT`.
	This can skew an imbalance in the token types and their weights.

Recommendation	Consider adding the following check to ensure that a weight with a lower index is always greater than a higher index.
	`for (uint256 i; i < 5; i++) {`
	`  require(weights[i] > weights[i+1], "invalid weight");`
	`}`

Resolution	RESOLVED

Issue #19	**If `rewardsDuration` is set to zero, it will cause a divide by zero revert**
Severity	LOW SEVERITY

Location	L119
	`rewardRate = reward / rewardsDuration;`
	
	L123
	`rewardRate = (reward + leftover) / rewardsDuration;`
	
	L131
	`require(rewardRate <= balance / rewardsDuration, "DSQStaking: Provided reward too high");`

Description	Within `notifyRewardAmount`, `rewardsDuration` is used as the denominator for some calculations as above. If set to zero, it would cause a divide by zero revert, thus not allowing the new reward emission state to be updated.
Recommendation	Consider adding a non-zero check in `setRewardsDuration` to ensure that `rewardsDuration` cannot be set to zero. `require(_rewardsDuration != 0, “No zero duration”);`
Resolution	RESOLVED

Issue #20	**getWeightedAmount will return a weightedAmount of 0 for some token ids**
Severity	INFORMATIONAL

Location	L93~
	`function getWeightedAmount(uint256 id, bool genesis) public view returns (uint256) {`
	`uint256 weightedAmount;`
	
	`if (genesis) {`
	`if (id <= 2) {`
	`weightedAmount = GENESIS_LEGENDARY_WEIGHT;`
	`} else if (id <= 12) {`
	`weightedAmount = GENESIS_RARE_WEIGHT;`
	`} else if (id <= 32) {`
	`weightedAmount = GENESIS_UNCOMMON_WEIGHT;`
	`}`
	`} else {`
	`if (id <= 10) {`
	`weightedAmount = NORMAL_RARE_WEIGHT;`
	`} else if (id <= 190) {`
	`weightedAmount = NORMAL_UNCOMMON_WEIGHT;`
	`} else if (id <= 2190) {`
	`weightedAmount = NORMAL_COMMON_WEIGHT;`
	`}`
	`}`
	
	`return weightedAmount;`
	`}`

Description	For genesis token ids above 32, and non-genesis token ids above 2190, `weightedAmount` would be returned as zero as it is never set to any amount, and is defined as the initial default value of 0.
Recommendation	Confirm if there will be any token ids above 32 for the genesis token contract, and above 2190 for the non-genesis token contract.
Resolution	ACKNOWLEDGED

Issue #21	**For NFT transfers, `transferFrom` instead of `safeTransferFrom` can be used to reduce gas costs**
Severity	INFORMATIONAL
Description	`safeTransferFrom` is used mainly in the following 2 user facing functions: - `stake` - `withdraw` Instead of using `safeTransferFrom`, which calls `IERC721Receiver(to).onERC721Received(_msgSender(), from, tokenId, data)` if the `to` address is a contract, `transferFrom` can be used to skip that behavior. For `stake`, the `from` is the user address which already has the NFT, and the `to` address is the NFTStaking contract which we know already can transfer out the NFTs using the `withdraw` function. For `withdraw`, the `to` address is the user who staked the NFT, and we already know that it has to previously support ERC721 tokens, otherwise it could not have received the NFT or staked.
Recommendation	Consider using `transferFrom` to transfer the NFTs.
Resolution	ACKNOWLEDGED

Issue #22	**`whenNotPaused` modifier is unnecessary for `stakeMultiple`**
Severity	INFORMATIONAL
Description	As `stakeMultiple` calls `stake`, which already has the `whenNotPaused` modifier, it is unnecessary for `stakeMultiple` to have it as well.
Recommendation	Remove `whenNotPaused` from `stakeMultiple`.
Resolution	RESOLVED

Issue #23	**`setRewardsDistribution` lacks event emission**
Severity	INFORMATIONAL
Description	`setRewardsDistribution` changes the `rewardsDistribution` address, but does not emit an event.
Recommendation	Consider adding an event to emit the old and new `rewardsDistribution` address, for easier querying when `rewardsDistribution` has been changed.
Resolution	RESOLVED

Issue #24	**`notifyRewardAmount` can dilute existing rewards if the reward parameter is small and there are few remaining rewards**
Severity	INFORMATIONAL

Description	If the current reward period has not finished, and there are few remaining rewards, calling `notifyRewardAmount` with a small `reward` parameter can cause a dilution of the new `rewardRate` as `rewardsDuration` is used as the denominator.
	`uint256 remaining = periodFinish - block.timestamp;`
	`uint256 leftover = remaining * rewardRate;`
	`rewardRate = (reward + leftover) / rewardsDuration;`

Recommendation	The `rewardsDistribution` role should take care in calling the `notifyRewardAmount` with a suitable amount.
Resolution	ACKNOWLEDGED

Issue #25	Lack of validation
Severity	INFORMATIONAL
Location	Line 225 `function setRewardsDistribution(address _rewardsDistribution)`
Description	The function above will break if the `address` is set to the zero address.
Recommendation	Consider validating that `address` cannot be set to the zero address.
Resolution	RESOLVED

Issue #26	Typographical errors
Severity	INFORMATIONAL

Description	Line 208
	`function recoverERC20(address tokenAddress, uint256 tokenAmount)`
	
	Line 213
	`function recoverERC721(address tokenAddress, uint256 tokenId)`
	
	Both `tokenAddress` parameters can be directly casted as type `IERC20` and `IERC721`.

Recommendation	Consider fixing all typographical issues.
Resolution	ACKNOWLEDGED

2.5 Router

`Router` is the entrypoint for users to interact with the `esDSQStaking` and `DSQStaking` contracts.

Users can call all desired functions within the `esDSQStaking` and `DSQStaking` contracts only via the `Router`. Additionally, the admin can stake via `adminStakeESDSQStaking` in the `esDSQStaking` contract for any desired address, which removes the necessity of having a matching stake in `DSQStaking`. The contract is also pausable by the admin which prevents all staking but not withdrawal/claiming operations.

2.5.1 Privileged Functions

- `transferOwnership`
- `renounceOwnership`
- `pause`
- `unpause`

2.5.2 Issues & Recommendations

Issue #27	Typographical errors
Severity	INFORMATIONAL
Description	Line 44 `constructor(address _owner, address _dsq, address _esdsq, address _dsqStaking, address _esdsqStaking)` `_dsq` and `_esdsq` can be directly cached as `IERC20`.
Recommendation	Consider fixing the above issue(s).
Resolution	RESOLVED

Issue #28	Rewards can be claimed even when router is paused
Severity	INFORMATIONAL
Description	`harvestDSQStakingAndStakeESDSQStaking` and `claimESDSQStakingAndStakeDSQStaking` both have the `whenNotPaused` modifier, preventing execution when the router contract is paused. However, `harvestDSQStaking` and `claimESDSQStaking` do not have this modifier. In the case where there are some issues with rewards calculation (e.g. math issues resulting in excessive rewards), it would be ideal to only allow staked funds, but not allow claiming of rewards.
Recommendation	Clarify if rewards should be claimable if the router is paused.
Resolution	ACKNOWLEDGED

2.6 StakingRewards

`StakingRewards` is a simple staking contract inspired by the Synthetix `stakingRewards` contract (https://github.com/Synthetixio/synthetix/blob/develop/contracts/StakingRewards.sol).

Users can simply stake the `stakingToken` in order to receive `rewardTokens`. The contract owner can determine the reward rate via `notifyRewardAmount`.

2.6.1 Privileged Functions

- `transferOwnership`
- `renounceOwnership`
- `recoverERC20`
- `setRewardsDuration`
- `setRewardsDistribution`

2.6.2 Issues & Recommendations

Issue #29	**rewardsToken and stakingToken can be the same**
Severity	HIGH SEVERITY
Description	In the case where the `rewardsToken` is the same token as the `stakingToken`, `rewardsToken.balanceOf(address(this))` would return the sum of `stakingToken` and `rewardsToken`. This allows the `RewardsDistribution` to set a much higher reward rate using `notifyRewardAmount` based on a much higher perceived `rewardsToken` balance. Over time, the higher rate would exceed the `rewardsToken` balance and eat into the `stakingToken` balances, resulting in insufficient token balances for all users to withdraw. Additionally, another side effect of both tokens being the same contract address would mean that `recoverERC20` will not be able to recover the reward token.
Recommendation	Add a check in the constructor to prevent both tokens from being the same address. `require(_rewardsToken != _stakingToken, "Same token");`
Resolution	RESOLVED

Issue #30	Contract is not pausable
Severity	MEDIUM SEVERITY
Description	The contract inherits the Pausable library from OpenZeppelin and expects `stake` to only be called if the contract is not paused. However, the contract does not expose any external pause and unpause functions, which effectively prevents the pause feature from being used.
Recommendation	Consider adding external pause and unpause functions.
Resolution	RESOLVED

Issue #31	Staking tokens with a fee on transfer are not supported
Severity	MEDIUM SEVERITY

Location	L84~
	`_totalSupply = _totalSupply + amount;`
	`_balances[account] = _balances[account] + amount;`
	
	`stakingToken.safeTransferFrom(router, address(this), amount);`

Description	`amount` is used to increase the `totalSupply` and user's staked balance. If the staking token has some form of fee on transfer which results in the actual token amount received by the contract being less than the amount parameter, it would result in an irregularity between the internal account and actual token balances.

Recommendation	Get the actual amount by doing after - before of the token balances of the staking contract to get the actual received amount.
	
	`uint256 before = stakingToken.balanceOf(address(this));`
	`stakingToken.safeTransferFrom(router, address(this), amount);`
	`amount = stakingToken.balanceOf(address(this)) - before;`
	
	`_totalSupply = _totalSupply + amount;`
	`_balances[account] = _balances[account] + amount;`

Resolution	ACKNOWLEDGED

Issue #32	If set to zero, `rewardsDuration` reverts due to division by zero
Severity	LOW SEVERITY

Location	L116
	`rewardRate = reward / rewardsDuration;`
	
	L120
	`rewardRate = (reward + leftover) / rewardsDuration;`
	
	L128
	`require(rewardRate <= balance / rewardsDuration, "DSQStaking: Provided reward too high");`

Description	In `notifyRewardAmount`, `rewardsDuration` is used as the denominator for some calculations such as the above. If it is set to zero, it would cause a division by zero revert, thus not allowing the new reward emission state to be updated.
Recommendation	Consider adding a non-zero check in `setRewardsDuration` to ensure that `rewardsDuration` cannot be set to zero. `require(_rewardsDuration != 0, “No zero duration”);`
Resolution	RESOLVED

Issue #33	**`recoverERC20` can be used to recover `stakingToken` if it has multiple contract entry points**
Severity	INFORMATIONAL
Description	For a token such as ERC20 TUSD (on ETH mainnet) which has more than one entry point to transfer tokens, the secondary entry point's contract address can be used as the recovery token, allowing the owner to withdraw the staking tokens, which should not be allowed. https://medium.com/chainsecurity/trueusd-compound-vulnerability-bc5b696d29e2 This issue has been categorized as informational, as the intended staking token would be the DSQ token which does not have multiple entry points.

Recommendation	Ensure that there is no change in the staking contract's staking Token when recoverERC20 is called.
	
	`uint256 before = stakingToken.balanceOf(address(this));`
	`IERC20(tokenAddress).safeTransfer(owner(), tokenAmount);`
	`uint256 after = stakingToken.balanceOf(address(this)) - before;`
	`require( after == before, "stakingToken changed");`

Resolution	ACKNOWLEDGED

Issue #34	**`setRewardsDistribution` lacks event emission**
Severity	INFORMATIONAL
Description	`setRewardsDistribution` changes the `rewardsDistribution` address, but does not emit an event.
Recommendation	Consider adding an event to emit the old and new `rewardsDistribution` address, for easier querying on when `rewardsDistribution` has been changed.
Resolution	RESOLVED

Issue #35	**`notifyRewardAmount` can dilute existing rewards if the reward parameter is small and there are few remaining rewards**
Severity	INFORMATIONAL

Description	If the current reward period has not finished, and there are few remaining rewards, calling `notifyRewardAmount` with a small reward parameter can cause a dilution of the new `rewardRate` as `rewardsDistribution` is used as the denominator.
	
	`uint256 remaining = periodFinish - block.timestamp;`
	`uint256 leftover = remaining * rewardRate;`
	`rewardRate = (reward + leftover) / rewardsDuration;`

Recommendation	The `rewardsDistribution` role should take care in calling the `notifyRewardAmount` with a suitable amount.
Resolution	ACKNOWLEDGED

Issue #36	Arithmetic operations can be simplified
Severity	INFORMATIONAL

Description	There are several arithmetic operations that can be simplified in order to keep the contract more readable.
	
	Line 85
	`_totalSupply = _totalSupply + amount;`
	
	Simply use `_totalSupply += amount`.
	
	Line 86
	`_balances[account] = _balances[account] + amount;`
	
	Simply use `_balances[account] += amount`.
	
	The same applies for the `withdraw` function — consider using the `-=` pattern.

Recommendation	Consider implementing the suggested changes.
Resolution	RESOLVED

2.7 esDSQStaking

`esDSQStaking` allows users to stake the `esDSQToken` and receive a `rewardToken` as reward, which is the `DSQToken`. Unlike traditional staking contracts, the reward can only be claimed once a vesting period has passed. Additionally, the user must have a stake in `DSQStaking` that is equal or higher in order to be able to stake in this contract. If the user then decides to withdraw from the `DSQStaking` contract, the vesting period gets exceeded by the time until the user stakes the appropriate balance in `DSQStaking` again. `emergencyWithdraw` allows users to withdraw their stake anytime, however, this will also reset the vesting position.

The admin has the privilege to stake for users without fulfilling the staking requirement in `DSQStaking`.

2.7.1 Privileged Functions

- `transferOwnership`
- `renounceOwnership`
- `setRewardRate`
- `setVestingDuration`
- `setRouter`

2.7.2 Issues & Recommendations

Issue #37	Admin stake will most likely not work as expected
Severity	HIGH SEVERITY
Description	Within the stake function with `_adminStake = true`, the admin can stake for any address without needing to have a deposit in the `DSQStaking` contract. While that works as intended, the flaw is within the `notify` function: whenever the corresponding address deposits or withdraws from `DSQStaking`, which is highly likely, it will always call `notify` which then sets the vesting position on `pause` and requires a subsequent corresponding deposit within DSQStaking in order to unpause the vesting again which makes the whole admin logic void.
Recommendation	Consider if this an issue, and if so, additional logic needs to be implemented that marks the position as `_adminStake` and excludes this from any `notify` call.
Resolution	RESOLVED There are now two possibilities: regular stake and admin stake. Due to the change, a user can now have both staking positions simultaneously. The `notify` function will now only check the regular stake for the corresponding balance. The `claim` and `emergencyWithdraw` function will take care of both stakes if the user has both.

Issue #38	Lack of validation
Severity	INFORMATIONAL

Description	The contract has multiple sections where a validation is not implemented, especially sections that will break the contract if the address is set to the zero address.
	
	Line 80
	`constructor(address _owner, address _dsq, address _esdsq, address _dsqStaking)`
	
	Consider ensuring that `_dsq`, `_esdsq` and `_dsqStaking` cannot be `address(0)`.
	
	Line 229
	`function setRouter(address _router)`
	
	Consider ensuring that `_router` can never be `address(0)`.

Recommendation	Consider validating the issues above.
Resolution	PARTIALLY RESOLVED

Issue #39	Lack of `safeTransfer` usage
Severity	INFORMATIONAL
Description	Within `claim` and `emergencyWithdraw`, the `transfer` method is used to transfer tokens from the contract to the recipient / `_account`. This will not work for tokens that will return `false` on transfer (or malformed tokens that do not have a return value). This is marked as informational as both tokens used, `esDSQ` and `DSQ` both have `transfer` and `transferFrom` functions that follow the ERC20 standards.
Recommendation	Consider using `safeTransfer` instead of `transfer` as is done throughout most of this contract.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY