PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For PolyWhale Vault

21 September 2021

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.

1 Overview

This report has been prepared for PolyWhale V2. 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	PolyWhale Vaults
URL	https://polywhale.finance
Platform	Polygon
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
KrillVault	KrillVault.sol	
KrillRewardPool	KrillRewardPool.sol	
StratManager	StratManager.sol	
FeeManager	FeeManager.sol	
StrategySushiswapLP	StrategySushiswapLP.sol	
StrategyQuickswapLP	StrategyQuickswapLP.sol	
StrategyJetswapLP	StrategyJetswapLP.sol	
StrategyDynfLP	StrategyDynfLP.sol	
StrategyWaultLP	StrategyWaultLP.sol	
StrategyApeswapLP	StrategyApeswapLP.sol	
StakingRewards	StakingRewards.sol	
StakingRewardsFactory	StakingRewardsFactory.sol	
KrillFeeBatch	KrillFeeBatch.sol	
KrillTreasury	KrillTreasury.sol	

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged or Unresolved
● High	9	6	-	3
● Medium	3	3	-	-
● Low	26	16	-	10
● Informational	28	24	-	4
Total	**66**	**49**	**-**	**17**

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 KrillVault

ID	Severity	Summary	Status
01	HIGH	Upgrades to a malicious strategy allow the dev to withdraw all staked funds after the timelock delay of proposing this malicious upgrade has expired	✓ RESOLVED
02	HIGH	Lack of validation for old and new `want` token when migrating strategies	✓ RESOLVED
03	HIGH	In case the underlying Masterchef has deposit fees, governance could burn all funds by emergency withdrawing and calling `earn()` over and over again	✓ RESOLVED
04	MEDIUM	Lack of lower limit validation for strategy's `approvalDelay`	✓ RESOLVED
05	LOW	In case there are deposit fees or transfer taxes, deposits can be prevented through an expensive attack by sending tokens to the vault	✓ RESOLVED
06	LOW	Lack of validation that the investor received sufficient shares	ACKNOWLEDGED
07	INFO	Lack of validation that the `earn` function actually increases the vault value	✓ RESOLVED
08	INFO	Tokenomics: Deposits are inefficient for tokens with transfer taxes	ACKNOWLEDGED
09	INFO	Lack of events for `deposit`, `withdraw` and `inCaseTokensGetStuck`	✓ RESOLVED
10	INFO	Lack of check for receipt token if destination is vault address	✓ RESOLVED
11	INFO	Gas optimization: `withdraw` function could be simplified	✓ RESOLVED

1.3.2 KrillRewardPool

ID	Severity	Summary	Status
12	HIGH	Lack of `emergencyWithdraw` function could lead to funds being stuck	✓ RESOLVED
13	LOW	`LPTokenWrapper` only works with tokens that have no transfer taxes	✓ RESOLVED
14	LOW	Lack of constructor safety guards	✓ RESOLVED
15	LOW	Calling `notifyRewardAmount` with an excessive amount could potentially block deposits and withdrawals	UNRESOLVED
16	LOW	`notifyRewardAmount` can be called by the owner without actually transferring in reward funds, potentially blocking deposits and withdrawals	✓ RESOLVED
17	LOW	`stakedToken` and `rewardToken` can be made `immutable` and `public`	✓ RESOLVED
18	INFO	`getReward` will fail if there are insufficient tokens in the pool, potentially blocking exit as well	✓ RESOLVED
19	INFO	Lack of event for `inCaseTokensGetStuck`	✓ RESOLVED

1.3.3 StratManager

ID	Severity	Summary	Status
20	HIGH	Governance privilege: Governance can change the vault linked to the strategy	✓ RESOLVED
21	MEDIUM	Governance privilege: Swap router can be changed to steal rewards and dust LP tokens	✓ RESOLVED
22	LOW	Setting `krillFeeRecipient` or `strategist` to the zero address will break `harvest` functionality	✓ RESOLVED
23	INFO	Lack of events for `setKeeper`, `setStrategist`, `setUnirouter`, `setVault` and `setFeeRecipient`	✓ RESOLVED

1.3.4 FeeManager

ID	Severity	Summary	Status
24	INFO	Lack of events for `setCallFee` and `setWithdrawalFee`	✓ RESOLVED

1.3.5 StrategyLP Contracts (General Issues)

The issues below apply to all StrategyLP contracts included in this audit: StrategySushiswapLP, StrategyQuickswapLP, StrategyJetswapLP, StrategyDynfLP, StrategyWaultLP and StrategyApeswapLP.

ID	Severity	Summary	Status
25	LOW	Dust tokens will accumulate over time	✓ RESOLVED
26	INFO	`retireStrat` can be removed if upgradeability is removed	✓ RESOLVED
27	INFO	Lack of events for `panic` and `retireStrat`	✓ RESOLVED
28	INFO	`panic` can be made `external`	✓ RESOLVED

1.3.6 StrategySushiswapLP

ID	Severity	Summary	Status
29	LOW	Lack of constructor parameter validation could lead to loss of funds if misconfigured	✓ RESOLVED
30	INFO	`want`, `chef` and `poolId` can be made `immutable`	✓ RESOLVED

1.3.7 StrategyQuickswapLP

ID	Severity	Summary	Status
31	LOW	Lack of constructor parameter validation could lead to loss of funds if misconfigured	✓ RESOLVED
32	INFO	`want` and `rewardPool` can be made `immutable`	✓ RESOLVED

1.3.8 StrategyJetswapLP

ID	Severity	Summary	Status
33	LOW	Lack of constructor parameter validation could lead to loss of funds if misconfigured	✓ RESOLVED
34	INFO	`want`, `chef` and `poolId` can be made `immutable`	✓ RESOLVED

1.3.9 StrategyDynfLP

ID	Severity	Summary	Status
35	HIGH	Currently strategyDynfLP is configured to use Polycat farms which no longer has rewards	ACKNOWLEDGED
36	HIGH	`deposit` function does not validate deposit fees	ACKNOWLEDGED
37	HIGH	Governance privilege: Calling `emergencyWithdraw` and `deposit` iteratively will result in all funds being sent to the Masterchef deposit `feeAddress`	ACKNOWLEDGED
38	LOW	Lack of constructor parameter validation could lead to loss of funds if misconfigured	ACKNOWLEDGED
39	LOW	Usage of wMaticDfyn as a middle token might cause high slippage due to illiquid pairs	ACKNOWLEDGED
40	LOW	Typo: Dynf should be Dfyn	ACKNOWLEDGED
41	INFO	`want`, `chef` and `poolId` can be made `immutable`	ACKNOWLEDGED

1.3.10 StrategyWaultLP

ID	Severity	Summary	Status
42	LOW	Lack of constructor parameter validation could lead to loss of funds if misconfigured	✓ RESOLVED
43	INFO	`want` and `rewardPool` can be made `immutable`	✓ RESOLVED

1.3.11 StrategyApeswapLP

ID	Severity	Summary	Status
44	LOW	Lack of constructor parameter validation could lead to loss of funds if misconfigured	✓ RESOLVED
45	INFO	`want`, `chef` and `poolId` can be made `immutable`	✓ RESOLVED

1.3.12 StrategyAave

ID	Severity	Summary	Status
46	LOW	Inefficient withdrawal method could lead to funds getting stuck temporarily if the vault has a large holding and all funds are loaned out on Aave	ACKNOWLEDGED
47	LOW	Inefficient withdrawal method could lead to funds getting lost if Aave ever incorporates deposit fees	ACKNOWLEDGED
48	LOW	Lack of constructor parameter validation could lead to loss of funds if misconfigured	✓ RESOLVED
49	LOW	Fees are unnecessarily routed through WETH causing unnecessary slippage	ACKNOWLEDGED
50	INFO	`want`, `aToken`, `varDebtToken`, `borrowRateMax` and `minLeverage` can be made `immutable`	✓ RESOLVED
51	INFO	`panic` and `userAccountData` can be made external	✓ RESOLVED

1.3.13 StakingRewards

ID	Severity	Summary	Status
52	HIGH	Lack of `emergencyWithdraw` function could lead to funds being stuck	✓ RESOLVED
53	LOW	`stake` only works with non transfer-tax tokens	✓ RESOLVED
54	LOW	Lack of constructor safety guards	✓ RESOLVED
55	INFO	Calling `notifyRewardAmount` with an excessive amount could potentially block deposits and withdrawals	ACKNOWLEDGED
56	INFO	`stakedToken` and `rewardToken` can be made `immutable`	✓ RESOLVED
57	INFO	`getReward` will fail if there are insufficient tokens in the pool, potentially blocking exit as well	✓ RESOLVED
58	INFO	`permit` can be frontrun and cause denial of service	ACKNOWLEDGED
59	INFO	Lack of event for `inCaseTokensGetStuck`	✓ RESOLVED

1.3.14 StakingRewardsFactory

ID	Severity	Summary	Status
60	LOW	`update` can be frontrun to distribute a reward twice	ACKNOWLEDGED
61	INFO	Wrong amount notified if the `rewardsToken` has a transfer tax	✓ RESOLVED

1.3.15 KrillFeeBatch

ID	Severity	Summary	Status
62	MEDIUM	Governance privilege: Contract is highly reconfigurable by the owner, allowing the owner to adjust the parameter so the owner receives all WMATIC fees in the contract	✓ RESOLVED
63	LOW	`wNativeToKrillRoute` is not initialized in the constructor	✓ RESOLVED
64	LOW	`harvest` is vulnerable to frontrunning	ACKNOWLEDGED

1.3.16 KrillTreasury

ID	Severity	Summary	Status
65	INFO	call is preferred over transfer	✓ RESOLVED
66	INFO	The owner can withdraw tokens at any time	✓ RESOLVED

2 Findings

2.1 KrillVault

2.1.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `proposeStrat`
- `upgradeStrat`
- `inCaseTokensGetStuck`

2.1.2 Issues & Recommendations

Issue #01	**Upgrades to a malicious strategy allow the dev to withdraw all staked funds after the timelock delay of proposing this malicious upgrade has expired**
Severity	● HIGH SEVERITY
Location	Line 1043 `function upgradeStrat() public onlyOwner {`
Description	As the PolyWhale vaults are forked from Beefy Finance, it contains the same upgradeability code as Beefy has. The idea is that when the vault strategy changes over time, users do not need to restake; instead the developers can simply upgrade the code to a new strategy. If the governance chooses a malicious contract to upgrade to, the governance can steal all staked funds. For some vaults, however, this might be seen as an excessive privilege. For one, it is very unlikely that the QuickSwap staking contract is going to change or break for example. For these simple strategies, we recommend removing this governance privilege in favour of promoting decentralisation and investor confidence that PolyWhale cannot steal their investors' funds. Furthermore, for simple strategies like PancakeSwap compounding, third-party reviewers like RugDoc may view this as an excessive privilege and either mark the vault as *High Risk* or *Not Eligible*. The main risk in our experience with simple staking contracts is that the withdraw function could break due to the reward mechanism. However, this issue is already taken care of in the `panic()` method in the strategies, which emergency withdraws all funds without interacting with the reward mechanism. It should be noted that this is exactly the same upgradeability code as Beefy has and Beefy has similar governance privileges. It should also be noted that larger investors can protect themselves by actively listening to `NewStratCandidate` events emitted by the vault. These events announce that an upgrade can happen after the approval delay expires. Investors can then review the new strategy and decide to unstake if it is malicious.

Recommendation	Consider whether upgradeability is a necessary requirement. The client could consider conducting a poll with their users to see which option they prefer. If the client is comfortable with asking investors to restake in a new vault when a strategy adjustment has to be made, they could consider removing the `upgradeStrat`, `proposeStrat` and `StratCandidate` functions.
Resolution	✓ RESOLVED `upgradeStrat`, `proposeStrat` and `StratCandidate` have been removed.

Issue #02	Lack of validation for old and new want token when migrating strategies
Severity	● HIGH SEVERITY

Location	Line 1027
	`function proposeStrat(address _implementation`
	`) public onlyOwner {`

Description	While there is a validation of the new strategy in `proposeStrat` to ensure that the vault address of it is the same as the vault itself, there is no guarantee that the old and new strategy's `want` token is the same. If the `want` is different (e.g Token A for old strategy, B for new strategy), and a migration goes through, the old `want` tokens (A) would be transferred from the old strategy into the vault, and the new `want` token (B) would be deposited to the new strategy. In such a case, users will be unable to withdraw their original deposited tokens as the `want` token has changed. Also, since the `want` token has now changed, the original `want` tokens can be withdrawn by the owner using `inCaseTokensGetStuck`.

Recommendation	Consider adding a check in `proposeStrategy` that ensures that the `want` token hasn't changed compared to the current strategy.
	
	Lines 1027-1031
	`function proposeStrat(address _implementation) public onlyOwner {`
	`    require(address(this) == IStrategy(_implementation).vault(),`
	`"Proposal not valid for this Vault");`
	`    require(want() == IStrategy(_implementation).want(),`
	`“Different want”);`
	`    stratCandidate = StratCandidate({`
	`        implementation: _implementation,`
	`        proposedTime: block.timestamp`
	`     });`
	
	`    emit NewStratCandidate(_implementation);`
	`}`

Resolution	✓ RESOLVED `upgradeStrat`, `proposeStrat` and `StratCandidate` have been removed.

Issue #03	In case the underlying Masterchef has deposit fees, governance could burn all funds by emergency withdrawing and calling `earn()` over and over again
Severity	● HIGH SEVERITY
Description	A lot of the common Masterchefs have or allow for deposit fees on their pools. If the governance of this vault ever turns truly malicious, they could repeatedly call the `panic` and `earn` methods over and over again until all funds are lost to the deposit fees (or transfer taxes).
Recommendation	Consider allowing emergency withdrawal functions (`panic` and `retireStrat`) to only be called once on all underlying strategies, closing the strategy permanently.
Resolution	✓ RESOLVED `retireStrat()` has been removed and `panic()` can be only called once.

Issue #04	Lack of lower limit validation for strategy's `approvalDelay`
Severity	● MEDIUM SEVERITY
Description	Although the `approvalDelay` cannot be modified after initialization in the constructor, it is possible to set a value of 0, or a low value. This can allow instant or almost instant changes to the underlying strategy. The severity for this is adjusted as it is set in the constructor, so users can verify the state variable which cannot be modified before depositing into the vault.
Recommendation	Consider adding a lower limit check for `approvalDelay` in the constructor. For example, if the lower limit is 7 days, the value should be >= 7 days. `require(_approvalDelay >= 7 days, "Insufficient approval delay");`
Resolution	✓ RESOLVED `_approvalDelay` has been removed.

Issue #05	In case there are deposit fees or transfer taxes, deposits can be prevented through an expensive attack by sending tokens to the vault
Severity	● LOW SEVERITY

Location	Lines 966-981
	`function deposit(uint _amount) public nonReentrant {`
	`    strategy.beforeDeposit();`
	`    uint256 _pool = balance();`
	`…`
	`    earn();`
	`    uint256 after = balance();`
	`    _amount = _after.sub(_pool); // Additional check for deflationary tokens`
	`…`
	`}`

Description	To allow deposits to be made in Masterchefs with deposit fees, the vault checks the balance of the vault before and after the deposit and adds the incrementation to the user. However, a malicious actor could transfer in tokens in the vaults which would then be staked as well in the `earn()` call. Since these tokens are already added to `balance()`, the only impact they have on the final _amount is a negative one, since the vault value decreases as they are staked. Exploit specification: 1. A user wishes to deposit 1 BUSD in a vault that deposits this in a 4% deposit fee masterchef 2. Before the user does this, a malicious attacker transfers in 30 BUSD to the vault (using a simple ERC20 transfer) 3. When the user deposits, the vault actually deposits 31 BUSD in the Masterchef resulting in a 1.24 BUSD fee 4. Since the vault value after the deposit is larger than before, the subtraction is negative and reverts the deposit.

Recommendation	In case the underlying strategies do not have any deposit fees, no changes need to be made, except for the case that transfer tax tokens will ever be added. In case strategies with deposit fees are added, this potential vector needs to be considered.
	
	Should the client wish to resolve this issue, an `amount` parameter could be added to the internalized `earn()` function to only deposit the deposited amount.
	
	`function earn(uint256 amount) internal {`
	`    want().safeTransfer(address(strategy), amount);`
	`    strategy.deposit();`
	`}`

Resolution	✓ RESOLVED

Issue #06	**Lack of validation that the investor received sufficient shares**
Severity	● LOW SEVERITY
Description	There is currently no validation in the `deposit` function to verify that the user received sufficient shares. An example where this could be problematic is if the underlying strategy deposits in Masterchef with deposit fees and the fees are set to 100%. In this case, the user receives zero shares but their funds will still be deposited.
Recommendation	Consider adding a check in the `deposit` or `earn` function to ensure that a fraction (eg. 90%) of the balance/deposit amount has been added to the `balance()`.
Resolution	● ACKNOWLEDGED

Issue #07	Lack of validation that the earn function actually increases the vault value
Severity	● INFORMATIONAL
Description	The earn function is used to compound the vault; however, in case this compound actually leads to a loss in the underlying strategy (for example due to transfer-tax tokens after an emergencyWithdraw is called), this could lead to all depositors having a reduced value per share.
Recommendation	Consider reverting harvest calls that reduce the share value. However, since earn() is used to deposit transfer-tax and deposit in Masterchefs with a deposit-fee, this will prevent all deposits. Thus the client should consider making the current earn() implementation internal and only adding the requirement to a new external harvest() function.

```
function harvest() external {
   uint256 _prevBal = balance();
   earn();
   require(balance() >= _prevBal, "not profitable");
}

function earn() internal {
    uint256 _bal = available();
    want().safeTransfer(address(strategy), _bal);
    strategy.deposit();
}
```

A similar check could be added to the deposit function to ensure that the value of shares is not reduced there either, but in that case, the balance/share should be verified since both numbers will usually increase.

Note that this harvest() functionality now suffers from a similar denial of service vector as the deposit() function. This could be prevented by adding an explicit amount parameter to harvest.

Resolution	✓ RESOLVED

Issue #08	Tokenomics: Deposits are inefficient for tokens with transfer taxes
Severity	● INFORMATIONAL
Description	Since deposits first transfer the funds into the vault, then to the strategy and finally into the actual underlying protocol, this might cause significant loss of funds compared to direct staking.
Recommendation	Consider the implications of this if the vault is ever considered for tokens with transfer taxes.
Resolution	● ACKNOWLEDGED The client has stated that the vault does not use tokens with transfer taxes.

Issue #09	Lack of events for `deposit`, `withdraw` and `inCaseTokensGetStuck`
Severity	● INFORMATIONAL
Description	`deposit`, `withdraw` and `inCaseTokensGetStuck` functions do not emit any events, even though such functions change the state of the contract.
Recommendation	Add events for the above functions.
Resolution	✓ RESOLVED

Issue #10	Lack of check for receipt token if destination is vault address
Severity	INFORMATIONAL
Description	Users might mistakenly send the receipt token to the vault thinking that it would allow them to redeem their underlying funds and lose funds in the process. In such a case, the vault owner would be able to recover those tokens using the `inCaseTokensGetStuck` function, but there is no use case where the receipt token is required to be sent to the vault. Note that although this strictly speaking is not compliant with the ERC-20 specification, many users have sent Moo tokens to Beefy vaults directly, losing them permanently.

Recommendation	Consider modifying the `transfer` function to revert if the destination address is the vault contract address.
	
	`function transfer(address recipient, uint256 amount) public override returns (bool) {`
	`    require(recipient != address(this), "!Use deposit function");`
	`    return super.transfer();`
	`}`

Resolution	✓ RESOLVED

Issue #11	Gas optimization: `withdraw` function could be simplified
Severity	● INFORMATIONAL
Description	In the `withdraw` function, a before-after pattern is done to calculate how much tokens are withdrawn from the strategy. Then `r`, the amount to be sent to the user, is set to `b.add(_diff)`. This could be simplified to save gas.
Recommendation	The following code in withdraw can be simplified as such:
Resolution	✓ RESOLVED

As is:

```
if (b < r) {
    uint _withdraw = r.sub(b);
    strategy.withdraw(_withdraw);
    uint _after = want().balanceOf(address(this));
    uint _diff = _after.sub(b);

    if (_diff < _withdraw) {
        r = b.add(_diff);
    }
}
```

Recommended:

```
if (b < r) {
    uint _withdraw = r.sub(b);
    strategy.withdraw(_withdraw);
    uint _after = want().balanceOf(address(this));
    r = _after;
}
```

2.2 KrillRewardPool

The KrillRewardPool is a staking pool based on the Synthetix reward pool. It allows users to stake a staking token and receive reward tokens over time.

2.2.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `notifyRewardAmount`
- `inCaseTokensGetStuck`

2.2.2 Issues & Recommendations

Issue #12	Lack of `emergencyWithdraw` function could lead to funds being stuck
Severity	● HIGH SEVERITY
Description	During certain misconfigurations like sending insufficient tokens to the reward pool or setting an extremely high `rewardRate`, the reward mechanism will malfunction and revert. Currently there's no `emergencyWithdraw` function that bypasses the reward mechanism for this case.

Recommendation	Consider adding an emergency withdraw function like the following one:
	`function emergencyWithdraw() external {`
	`    userRewardPerTokenPaid[msg.sender] = 0`
	`    rewards[msg.sender] = 0;`
	`    super.withdraw(balanceOf(msg.sender));`
	`}`

Resolution	✓ RESOLVED

Issue #13	`LPTokenWrapper` only works with tokens that have no transfer taxes
Severity	● LOW SEVERITY

Location	Lines 545-546
	`_balances[msg.sender] = _balances[msg.sender].add(amount);`
	`stakedToken.safeTransferFrom(msg.sender, address(this), amount);`

Description	The `LPTokenWrapper` and by extension the KrillRewardPool will only work for standard tokens without a transfer tax. This is because the stake function adds the transferred amount to the balance instead of the potentially lower received amount. This information is marked as low severity since the name `LPTokenWrapper` indicates these pools will only ever be used for LP tokens.
Recommendation	Consider updating the logic in the stake function to do a before-after pattern deposit like that of Uniswap. This however requires reentrancy guards to ensure this is not abused.
Resolution	✓ RESOLVED

Issue #14	Lack of constructor safety guards
Severity	● LOW SEVERITY
Description	The constructor currently lacks validation checks. This could lead to the code being deployed with the `stakedToken` address being equal to the `rewardToken` address. This specific setup could result in loss of stakes since the stakes could be given out as rewards to users.
Recommendation	Consider validating that the constructor tokens are not equal to each other. `require(_stakedToken != _rewardToken, “same tokens”);`
Resolution	✓ RESOLVED

Issue #15	**Calling `notifyRewardAmount` with an excessive amount could potentially block deposits and withdrawals**
Severity	● LOW SEVERITY
Description	If this function is called with too high a reward amount (such as one that exceeds the reward token balance in this contract), this may cause `notifyRewardAmount` to revert.
Recommendation	Consider adding a maximum amount to the reward which can be notified: `require(reward <= MAX_REWARD_INCREMENT);`
Resolution	● UNRESOLVED

Issue #16	**`notifyRewardAmount` can be called by the owner without actually transferring in reward funds, potentially blocking deposits and withdrawals**
Severity	● LOW SEVERITY
Description	The `notifyRewardAmount` function currently does not validate that funds are actually transferred in. This could lead to the contract having insufficient funds.

Recommendation	Consider validating that the balance is sufficiently high at the end of the `notifyRewardAmount` function:
	`uint balance = rewardsToken.balanceOf(address(this));`
	`require(rewardRate <= balance.div(DURATION), "Provided reward too high");`
	Source: Synthetix PR #617

Resolution	✓ RESOLVED

Issue #17	**`stakedToken` and `rewardToken` can be made `immutable` and `public`**
Severity	● LOW SEVERITY
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. Furthermore, important state variables should be marked as `public` in light of making the code more accessible to third-party reviewers. The `stakedToken` variable is currently `private`.
Recommendation	Consider making `stakedToken` and `rewardToken` explicitly `immutable` and `public`.
Resolution	✓ RESOLVED

Issue #18	**`getReward` will fail if there are insufficient tokens in the pool, potentially blocking exit as well**
Severity	● INFORMATIONAL
Description	Currently the `getReward` function tries to transfer out the reward and will simply revert if there are insufficient reward tokens. Within the Masterchef, this issue is avoided through a `safeRewardTransfer` function that transfers the contract balance out if the amount exceeds the balance. This way the function can never revert due to there being insufficient reward tokens.
Recommendation	Consider whether its desirable to have a fallback like the Masterchef has or to actually continue to revert these transactions so users do not lose their accumulated reward balance in this edge case scenario.
Resolution	✓ RESOLVED

Issue #19	Lack of event for `inCaseTokensGetStuck`
Severity	● INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add event for `inCaseTokensGetStuck`.
Resolution	✓ RESOLVED The function has been removed.

2.3 StratManager

The StratManager is a dependency implemented by the various strategies. It stores some important governance-related variables.

2.3.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `setKeeper`
- `setStrategist`
- `setUnirouter`
- `setVault`
- `setKrillFeeRecipient`

2.3.2 Issues & Recommendations

Issue #20	**Governance privilege: Governance can change the vault linked to the strategy**
Severity	● HIGH SEVERITY
Description	The Owner is able to call `setVault` which can change the vault contract that calls key privileged functions in the strategy contract such as `withdraw` and `retireStrat`.
Recommendation	Consider removing `setVault` since there is no obvious use case for it.
Resolution	✓ RESOLVED Resolved in all strategies except StrategyDyfnLP, but the client has mentioned that they would not be using StrategyDyfnLP.

Issue #21	**Governance privilege: Swap router can be changed to steal rewards and dust LP tokens**
Severity	● MEDIUM SEVERITY
Location	Line 1327 `function setUnirouter(address _unirouter) external onlyOwner {`
Description	The owner can change the router which is used to convert rewards into LP tokens, and setting this router to a malicious one can be abused to leach the rewards.
Recommendation	Consider whether the `uniRouter` ever needs to be changed and whether it would not be easier to simply redeploy. If there is a need for the `uniRouter` to be changed, consider putting the strategies behind a sufficiently long timelock.
Resolution	✓ RESOLVED Resolved in all strategies except StrategyDyfnLP, but the client has mentioned that they would not be using StrategyDyfnLP.

Issue #22	**Setting `krillFeeRecipient` or `strategist` to the zero address will break harvest functionality**
Severity	LOW SEVERITY
Location	Transferring tokens to the zero address will revert transactions. Thus, it is good practice to hardcode non-zero checks of addresses involved with token transfers.

Description	To prevent this from ever happening by accident and to limit governance risks, consider adding a requirement like the following :
	`require(_krillFeeRecipient != address(0), "!nonzero");`
	`require(_strategist != address(0), "!nonzero");`
	to the `setKrillFeeRecipient` and `setStrategist` function. It is desireable to add non-zero checks to the other setters as well.

Recommendation	Setting `krillFeeRecipient` or `strategist` to the zero address will break harvest functionality.
Resolution	✓ RESOLVED Resolved in all strategies except StrategyDyfnLP, but the client has mentioned that they would not be using StrategyDyfnLP.

Issue #23	Lack of events for `setKeeper`, `setStrategist`, `setUnirouter`, `setVault` and `setFeeRecipient`
Severity	● INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	✓ RESOLVED Resolved in all strategies except StrategyDyfnLP, but the client has mentioned that they would not be using StrategyDyfnLP.

2.4 FeeManager

The FeeManager is a dependency implemented by the various strategies. It allows the manager (owner or keeper) to update the fee distribution between:

- Call fee (up to 11.1% of the total fee) used to incentivize the harvester
- Strategist fee (fixed 11.2% of the total fee), presumably a governance fee
- Krill fee (up to 77.7% of the total fee), presumably used for burning krill

Performance fees are usually hardcoded to 4.5% of the harvest rewards and are converted to Krill, creating buying pressure. A withdrawal fee of up to 0.5% can also be set.

2.4.1 Privileged Roles

- `setCallFee`
- `setWithdrawalFee`

2.4.2 Issues & Recommendations

Issue #24	Lack of events for `setCallFee` and `setWithdrawalFee`
Severity	● INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	✓ RESOLVED Resolved in all strategies except StrategyDyfnLP, but the client has mentioned that they would not be using StrategyDyfnLP.

2.5 StrategyLP Contracts (General)

The issues below apply to all StrategyLP contracts included in this audit: StrategySushiswapLP, StrategyQuickswapLP, StrategyJetswapLP, StrategyDynfLP, StrategyWaultLP, StrategyApeswapLP and StrategyAave.

2.5.1 Privileged Roles

The following privileges apply to the owner of the StrategySushiswapLP, StrategyQuickswapLP, StrategyJetswapLP, StrategyDynfLP, StrategyWaultLP, StrategyApeswapLP and StrategyAave contracts:

- `panic`
- `pause`
- `unpause`
- `setKeeper`
- `setStrategist`
- `setUnirouter`
- `setVault`
- `setKrillFeeRecipient`
- `setCallFee`
- `setWithdrawalFee`

The vault has the following privilege in the StrategySushiswapLP, StrategyQuickswapLP, StrategyJetswapLP, StrategyDynfLP, StrategyWaultLP, StrategyApeswapLP and StrategyAave contracts:

- `withdraw`
- `retireStrat`

2.5.1 Issues & Recommendations

Issue #25	Dust tokens will accumulate over time
Severity	● LOW SEVERITY
Description	Over time, due to imbalances in the swaps, either `lpToken0` or `lpToken1` will slowly accumulate in the strategy. This will always only be a fraction of the value so is not that severe.
Recommendation	Consider adding a function that converts `lpToken0` and `lpToken1` back to the `want` token.
Resolution	✓ RESOLVED Resolved in all strategies except StrategyDyfnLP, but the client has mentioned that they would not be using StrategyDyfnLP.

Issue #26	`retireStrat` can be removed if upgradeability is removed
Severity	● INFORMATIONAL
Description	In case the client decides to remove upgradeability, `retireStrat` can be removed since it can only be called by the vault.
Recommendation	Consider removing `retireStrat` in case upgradeability is removed.
Resolution	✓ RESOLVED Resolved in all strategies except StrategyDyfnLP, but the client has mentioned that they would not be using StrategyDyfnLP.

Issue #27	Lack of events for `panic` and `retireStrat`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for `panic` and `retireStrat`.
Resolution	✓ RESOLVED Resolved in all strategies except StrategyDyfnLP, but the client has mentioned that they would not be using StrategyDyfnLP.

Issue #28	`panic` can be made `external`
Severity	INFORMATIONAL
Description	The `panic` function can be changed from `public` to `external`. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider making these functions `external`.
Resolution	✓ RESOLVED Resolved in all strategies except StrategyDyfnLP, but the client has mentioned that they would not be using StrategyDyfnLP.

2.6 StrategySushiswapLP

The StrategySushiswapLP stakes LP tokens in the Sushiswap Masterchef. Earned rewards are sold and compounded for more LP tokens.

The strategy has a withdrawal fee of up to 0.5% which is disabled while the strategy is paused. The owner is also excluded from the withdrawal fee. A fixed fee of 4.5% is taken from harvests, converted to Krill and distributed according to the distribution specified through the FeeManager functions. This distribution can involve sending the tokens to the `krillFeeRecipient`, caller or `strategist` (governance).

2.6.1 Issues & Recommendations

Issue #29	**Lack of constructor parameter validation could lead to loss of funds if misconfigured**
Severity	LOW SEVERITY
Description	Under certain circumstances, the Uniswap operations might remove `want` tokens since there is an overlap between the configured tokens.
Recommendation	Consider adding validation to the constructor to ensure that the `want` token is not equal to the native token or `output`. Also consider verifying that `krillFeeRecipient` and `strategist` are non-zero. The project team should also take care to validate that all parameters, especially the underlying `chef` parameter, are set correctly.
Resolution	✓ RESOLVED

Issue #30	**`want`, `chef` and `poolId` 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 the above parameters explicitly `immutable`.
Resolution	✓ RESOLVED

2.7 StrategyQuickswapLP

The StrategyQuickswapLP stakes LP tokens in the Quickswap staking addresses. Earned rewards are sold and compounded for more LP tokens.

The strategy has a withdrawal fee of up to 0.5% which is disabled while the strategy is paused. The owner is also excluded from the withdrawal fee. A fixed fee of 4.5% is taken from harvests, converted to Krill and distributed according to the distribution specified through the FeeManager functions. This distribution can involve sending the tokens to the `krillFeeRecipient`, caller or `strategist` (governance).

2.7.1 Issues & Recommendations

Issue #31	Lack of constructor parameter validation could lead to loss of funds if misconfigured
Severity	● LOW SEVERITY
Description	Under certain circumstances, the Uniswap operations might remove `want` tokens since there is an overlap between the configured tokens.
Recommendation	Consider adding validation to the constructor to ensure that the `want` token is not equal to `output`. Also consider verifying that `krillFeeRecipient` and `strategist` are non-zero. The project team should also take care to validate that all parameters, especially the underlying `chef` parameter, are set correctly. Consider also including the constructor validation from the more rich StrategySushiswapLP constructor.
Resolution	✓ RESOLVED

Issue #32	`want` and `rewardPool` 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 the above parameters explicitly `immutable`.
Resolution	✓ RESOLVED

2.8 StrategyJetswapLP

The StrategyJetswapLP stakes LP tokens in the Jetswap Masterchef. Earned rewards are sold and compounded for more LP tokens.

The strategy has a withdrawal fee of up to 0.5% which is disabled while the strategy is paused. The owner is also excluded from the withdrawal fee. A fixed fee of 4.5% is taken from harvests, converted to Krill and distributed according to the distribution specified through the FeeManager functions. This distribution can involve sending the tokens to the `krillFeeRecipient`, caller or `strategist` (governance).

2.8.1 Issues & Recommendations

Issue #33	**Lack of constructor parameter validation could lead to loss of funds if misconfigured**
Severity	● LOW SEVERITY
Description	Under certain circumstances, the Uniswap operations might remove `want` tokens since there is an overlap between the configured tokens.
Recommendation	Consider adding validation to the constructor to ensure that the `want` token is not equal to `output`. Also consider verifying that `krillFeeRecipient` and `strategist` are non-zero. The project team should also take care to validate that all parameters, especially the underlying `chef` parameter, are set correctly. Consider also including the constructor validation from the more rich StrategySushiswapLP constructor.
Resolution	✓ RESOLVED

Issue #34	**`want`, `chef` and `poolID` 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 the above parameters explicitly `immutable`.
Resolution	✓ RESOLVED

2.9 StrategyDynfLP

The StrategyDynfLP stakes LP tokens in the Polycat Masterchef. Earned rewards are sold and compounded for more LP tokens.

The strategy has a withdrawal fee of up to 0.5% which is disabled while the strategy is paused. The owner is also excluded from the withdrawal fee. A fixed fee of 4.5% is taken from harvests, converted to Krill and distributed according to the distribution specified through the FeeManager functions. This distribution can involve sending the tokens to the `krillFeeRecipient`, caller or `strategist` (governance).

2.9.1 Issues & Recommendations

Issue #35	**Currently `strategyDynfLP` is configured to use Polycat farms which no longer has rewards**
Severity	🔴 HIGH SEVERITY
Description	The StrategyDynfLP contract seems to be configured to stake funds in the Polycat Masterchef.
Recommendation	Consider explaining whether this contract is supposed to stake in Dfyn or in generic deposit-fee Masterchefs.
Resolution	● ACKNOWLEDGED The client will not be using this contract.

Issue #36	**`deposit` function does not validate deposit fees**
Severity	🔴 HIGH SEVERITY
Description	If the underlying Masterchef ever decides to set deposit fees to 100%, this is not caught in the `deposit` function and the deposited funds are lost permanently.
Recommendation	Consider checking how much funds are actually deposited in the masterchef and requiring that the fee taken is not excessive. The following code requires the fee to never exceed 10% for example.
Resolution	● ACKNOWLEDGED The client will not be using this contract.

```
function deposit() public whenNotPaused {
    uint256 wantBal = IERC20(want).balanceOf(address(this));
    uint256 balBefore = balanceOfPool();
    if (wantBal > 0) {
        IMasterChef(masterchef).deposit(poolId, wantBal,
referrer);
    }
   uint256 balAfter = balanceOfPool();
   require(balAfter.sub(balBefore) >= wantBal.mul(9).div(10));
}
```

Issue #37	Governance privilege: Calling `emergencyWithdraw` and `deposit` iteratively will result in all funds being sent to the Masterchef deposit `feeAddress`
Severity	● HIGH SEVERITY
Description	If there is a deposit fee, iteratively unstaking and staking in the vault will result in the vault value declining over time. A malicious governance could use this to drain the vault to the Masterchef `feeAddress`.
Recommendation	Consider removing `retireStrat` (or making it non-upgradeable) and only making `panic` callable once through a `panicked` boolean state variable.
Resolution	● ACKNOWLEDGED The client will not be using this contract.

Issue #38	Lack of constructor parameter validation could lead to loss of funds if misconfigured
Severity	● LOW SEVERITY
Description	Under certain circumstances, the Uniswap operations might remove `want` tokens since there is an overlap between the configured tokens. It is extremely important that this strategy is not used with ETH or WMATIC as the `want` token.
Recommendation	Consider adding validation to the constructor to ensure that the `want` token is not equal to ETH, WMATIC or `output`. Consider also verifying that `krillFeeRecipient` and `strategist` are non-zero.
Resolution	● ACKNOWLEDGED The client will not be using this contract.

Issue #39	Usage of wMaticDfyn as a middle token might cause high slippage due to illiquid pairs
Severity	● LOW SEVERITY
Description	The StrategyDynfLP contract automatically uses wrapped MATIC by Dfyn as the middle token in the swap route – this could cause high slippage since these pairs are likely illiquid on Quickswap.
Recommendation	Consider not using this route or explaining why this route is desirable.
Resolution	● ACKNOWLEDGED The client will not be using this contract.

Issue #40	Typo: Dynf should be Dfyn
Severity	● LOW SEVERITY
Description	The contract is called StrategyDynfLP while the underlying protocol is actually called Dfyn.
Recommendation	Consider renaming the contract.
Resolution	● ACKNOWLEDGED The client will not be using this contract.

Issue #41	`want`, `chef` and `poolID` 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 the above parameters explicitly `immutable`.
Resolution	● ACKNOWLEDGED The client will not be using this contract.

2.10 StrategyWaultLP

The StrategyWaultLP stakes LP tokens in the Wault Finance Masterchef. Earned rewards are sold and compounded for more LP tokens.

The strategy has a withdrawal fee of up to 0.5% which is disabled while the strategy is paused. The owner is also excluded from the withdrawal fee. A fixed fee of 4.5% is taken from harvests, converted to Krill and distributed according to the distribution specified through the FeeManager functions. This distribution can involve sending the tokens to the `krillFeeRecipient`, caller or `strategist` (governance).

2.10.1 Issues & Recommendations

Issue #42	**Lack of constructor parameter validation could lead to loss of funds if misconfigured**
Severity	● LOW SEVERITY
Description	Under certain circumstances, the Uniswap operations might remove `want` tokens since there is an overlap between the configured tokens.
Recommendation	Consider adding validation to the constructor to ensure that the `want` token is not equal to `output`. Also consider verifying that `krillFeeRecipient` and `strategist` are non-zero. The project team should also take care to validate that all parameters, especially the underlying `chef` parameter, are set correctly. Consider also including the constructor validation from the more rich StrategySushiswapLP constructor.
Resolution	✓ RESOLVED

Issue #43	**`want` and `rewardPool` 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 the above parameters explicitly `immutable`.
Resolution	✓ RESOLVED

2.11 StrategyApeswapLP

The StrategyApeswapLP stakes LP tokens in the ApeSwap Masterchef. Earned rewards are sold and compounded for more LP tokens.

The strategy has a withdrawal fee of up to 0.5% which is disabled while the strategy is paused. The owner is also excluded from the withdrawal fee. A fixed fee of 4.5% is taken from harvests, converted to Krill and distributed according to the distribution specified through the FeeManager functions. This distribution can involve sending the tokens to the `krillFeeRecipient`, caller or `strategist` (governance).

2.11.1 Issues & Recommendations

Issue #44	**Lack of constructor parameter validation could lead to loss of funds if misconfigured**
Severity	● LOW SEVERITY
Description	Under certain circumstances, the Uniswap operations might remove `want` tokens since there is an overlap between the configured tokens.
Recommendation	Consider adding validation to the constructor to ensure that the `want` token is not equal to `output`. Also consider verifying that `krillFeeRecipient` and `strategist` are non-zero. The project team should also take care to validate that all parameters, especially the underlying `chef` parameter, are set correctly.
Resolution	✓ RESOLVED

Issue #45	**`want`, `chef` and `poolID` 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 the above parameters explicitly `immutable`.
Resolution	✓ RESOLVED

2.12 StrategyAave

The StrategyAaveLP stakes tokens in the Aave lending protocol. Earned rewards are sold and compounded for more LP tokens.

The strategy has a withdrawal fee of up to 0.5% which is disabled while the strategy is paused. The owner is also excluded from the withdrawal fee. A fixed fee of 4.5% is taken from harvests, converted to Krill and distributed according to the distribution specified through the FeeManager functions. This distribution can involve sending the tokens to the `krillFeeRecipient`, caller or `strategist` (governance).

2.12.1 Issues & Recommendations

Issue #46	**Inefficient withdrawal method could lead to funds getting stuck temporarily if the vault has a large holding and all funds are loaned out on Aave**
Severity	● LOW SEVERITY
Description	During the withdrawal, all funds are temporarily unstaked from Aave. If there are not enough funds in the Aave pool to meet the pool TVL, this will block withdrawals until funds become available again. This issue has famously presented itself on Autofarm with their Venus vaults. Since the Autofarm stablecoin pools had tens of millions of dollars staked, they became non-withdrawable for days when Venus reached their full lending capacity.
Recommendation	Consider the likelihood of this scenario on the more liquid Aave protocol. If the likelihood is high enough, consider rethinking the withdrawal method to only deleverage what is necessary.
Resolution	● ACKNOWLEDGED The client has mentioned that their TVL will likely not be high enough for this to be happen.

Issue #47	Inefficient withdrawal method could lead to funds getting lost if Aave ever incorporates deposit fees
Severity	● LOW SEVERITY
Description	During the withdrawal, all funds are temporarily unstaked from Aave. If Aave ever upgrades their protocol to introduce deposit fees (they have proxy contracts), this could lead to significant inefficiencies. This issue has famously presented itself on Autofarm with their Venus vaults. At some point after a governance vote, Venus introduced a tiny deposit fee. Since the whole pool is unstaked and restaked on every withdrawal, this led to significant losses for the pool participants after this upgrade was done. Venus pools were permanently closed afterwards.
Recommendation	Consider monitoring the Aave governance closely for upgrades to react to these sorts of changes proactively. In case the likelihood of this scenario is deemed high enough, consider a more efficient withdrawal method as discussed in the previous issue.
Resolution	● ACKNOWLEDGED The client stated that they will follow Aave's developments closely to ensure they do not adopt new policies which might add fees.

Issue #48	Lack of constructor parameter validation could lead to loss of funds if misconfigured
Severity	● LOW SEVERITY
Description	Under certain circumstances, the Uniswap operations might remove want tokens since there is an overlap between the configured tokens.
Recommendation	Consider adding validation to the constructor parameters: 1. `want` should never equal WMATIC (otherwise its taken out). 2. `krillFeeRecipient` and `strategist` should be non-zero. 3. `borrowRate` and `borrowRateMax` should be set under 100 and `borrowRate` should be smaller than `borrowRateMax`. 4. `borrowDepth` should be smaller than `BORROW_DEPTH_MAX`
Resolution	✓ RESOLVED

Issue #49	Fees are unnecessarily routed through WETH causing unnecessary slippage
Severity	● INFORMATIONAL
Description	The StrategyAave rewards are routed from WMATIC to WETH to the staking token. This extra step through WETH could result in higher slippage and fees since most of these pairs are less liquid (and there is the extra base fee of using an extra step).
Recommendation	Consider removing WETH from the swap routes and instead using a 2-asset based route [wmatic, want].
Resolution	● ACKNOWLEDGED

Issue #50	`want`, `aToken`, `varDebtToken`, `borrowRateMax` and `minLeverage` 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 these variables explicitly `immutable`.
Resolution	✓ RESOLVED

Issue #51	`panic` and `userAccountData` can be made `external`
Severity	● INFORMATIONAL
Description	The `panic` and `userAccountData` functions can be changed from `public` to `external`. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider making these functions `external`.
Resolution	✓ RESOLVED

2.13 StakingRewards

The StakingRewards is a staking pool based on the Synthetix reward pool. It allows users to stake a staking token and receive reward tokens over time.

2.13.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `notifyRewardAmount`

2.13.2 Issues & Recommendations

Issue #52	Lack of `emergencyWithdraw` function could lead to funds being stuck
Severity	HIGH SEVERITY
Description	During certain misconfigurations like sending insufficient tokens to the reward pool or setting an extremely high `rewardRate`, the reward mechanism will malfunction and revert. Currently there's no `emergencyWithdraw` function that bypasses the reward mechanism for this case.

Recommendation	Consider adding an emergency withdraw function like the following one:
	`function emergencyWithdraw() external {`
	`    userRewardPerTokenPaid[msg.sender] = 0`
	`    rewards[msg.sender] = 0;`
	`    super.withdraw(balanceOf(msg.sender));`
	`}`

Resolution	✓ RESOLVED

Issue #53	stake only works with non transfer-tax tokens
Severity	● LOW SEVERITY

Location	Lines 506-512
	`function stake(uint256 amount) external nonReentrant updateReward(msg.sender) {`
	`    require(amount > 0, "Cannot stake 0");`
	`    _totalSupply = _totalSupply.add(amount);`
	`    _balances[msg.sender] = _balances[msg.sender].add(amount);`
	`    stakingToken.safeTransferFrom(msg.sender, address(this), amount);`
	`    emit Staked(msg.sender, amount);`
	`}`

Description	The StakingRewards contract will only work for standard tokens without a transfer tax. This is because the `stake` function adds the transferred amount to the balance instead of the potentially lower amount that may be received. This information is marked as low severity since the name LPTokenWrapper indicates these pools will only ever be used for LP tokens.
Recommendation	Consider updating the logic in the `stake` function to do a before-after pattern deposit like how it is done in Uniswap. This solution however requires reentrancy guards to ensure this is not abused.
Resolution	✓ RESOLVED

Issue #54	Lack of constructor safety guards
Severity	● LOW SEVERITY
Description	The constructor currently lacks validation checks. This could lead to the code being deployed with the `stakedToken` address being equal to the `rewardToken` address. This specific setup could result in loss of stakes since the stakes could be given out as rewards to users.
Recommendation	Consider validating that the constructor tokens are not equal to each other. `require(_stakingToken != _rewardToken, “same token”);`
Resolution	✓ RESOLVED

Issue #55	Calling `notifyRewardAmount` with an excessive amount could potentially block deposits and withdrawals
Severity	● INFORMATIONAL
Description	If this function is called with too high a reward amount (such as one that exceeds the reward token balance in this contract), `notifyRewardAmount` may revert.
Recommendation	Consider adding a maximum amount to the reward which can be notified: `require(reward <= MAX_REWARD_INCREMENT);`
Resolution	● ACKNOWLEDGED

Issue #56	`stakedToken` and `rewardToken` 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 `stakedToken` and `rewardToken` explicitly `immutable`.
Resolution	✓ RESOLVED

Issue #57	`getReward` will fail if there are insufficient tokens in the pool, potentially blocking exit as well
Severity	INFORMATIONAL
Description	Currently the `getReward` function tries to transfer out the reward and will simply revert if there are insufficient reward tokens. Within the Masterchef, this issue is avoided through a `safeRewardTransfer` function that transfers the contract balance out if the amount exceeds the balance. This way, the function can never revert due to there being insufficient reward tokens.
Recommendation	Consider whether its desirable to have a fallback like the Masterchef has or to actually continue to revert these transactions so users do not lose their accumulated reward balance in this edge case scenario.
Resolution	✓ RESOLVED The balance of the pool is given if the amount is less that the reward.

Issue #58	`permit` can be frontrun and cause denial of service
Severity	INFORMATIONAL
Description	Currently if `permit` is executed twice, the second execution will be reverted. It is thus in theory possible for a bot to pick up `stakeWithPermit` transactions in the mempool and execute the permit call before `stakeWithPermit` is executed. Due to the mechanics of `permit`, the second call reverts and thus `stakeWithPermit` reverts as well.
Recommendation	Consider wrapping the `stakeWithPermit` in a try-catch clause. This way, the rest of the code is still attempted even if the permit does not work out.
Resolution	ACKNOWLEDGED

Issue #59	Lack of event for `inCaseTokensGetStuck`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add event for `inCaseTokensGetStuck`.
Resolution	✓ RESOLVED The function has been removed.

2.14 StakingRewardsFactory

The StakingRewardsFactory is a utility contract that can be used by the governance to create a StakingRewards pool and manage administrative tasks like sending batches of rewards to the pools.

It should be noted that we've audited this contract with less focus on governance risks since we believe it is mainly used for administrative tasks. Of course, any rewards in the contract can be taken out again through the `pullExtraTokens` method (or by adding a special pool) and investors should not see this contract as a locked rewards contract.

2.14.1 Privileged Roles

The following functions can be called by the owner of the project:

- `deploy`
- `update`
- `pullExtraTokens`

2.14.2 Issues & Recommendations

Issue #60	**`update` can be frontrun to distribute a reward twice**
Severity	● LOW SEVERITY
Description	The `update` function schedules an update with the `rewardAmount` and duration of a pool. Similar to the `approve` method, it is vulnerable to frontrunning if the variables are already set. This can be explained this through an example. 1. The owner accidentally calls `update` with `rewardAmount` set to 10 tokens when they wanted to set it to15 tokens. 2. The owner calls `update` again and sets the `rewardAmount` to 15 tokens. 3. A malicious party has seen the second update and calls `notifyReward` amount before and after it, giving the pool 25 reward tokens.
Recommendation	We are unsure why `notifyRewardAmount` should have the ability to transfer `rewardsToken`. Each StakingRewards contract already has the ability to transfer those reward tokens via the `getReward` function. Consider removing the following line in this contract, or letting us know the purpose of this transfer function. `IERC20(rewardsToken).transfer(info.stakingRewards, rewardAmount), 'StakingRewardsFactory::notifyRewardAmount: transfer failed');`
Resolution	● ACKNOWLEDGED

Issue #61	Wrong amount notified if the `rewardsToken` has a transfer tax
Severity	INFORMATIONAL
Description	In case the `rewardsToken` has a transfer tax, the factory will notify the StakingRewards contract with the tokens sent, and not with the tokens the StakingContract received. This could result in a notification of a higher value when a transfer-tax token is used. This issue is marked as informational since we believe the StakingRewards contracts are not meant to be used with transfer-tax tokens.
Recommendation	In case this factory will be used for transfer-tax reward tokens, consider adding the recommended before-after pattern (this should always be combined with a reentrancy guard):

```
uint256 balBefore = IERC20(rewardsToken).balanceOf(info.stakingRewards);
IERC20(rewardsToken).transfer(info.stakingRewards, rewardAmount), '...')
rewardAmount = IERC20(rewardsToken).balanceOf(info.stakingRewards).sub(balBefore);
```

Resolution	✓ RESOLVED

2.15 KrillFeeBatch

The KrillFeeBatch contract manages the WMATIC tokens sent to it. Whenever anyone calls `harvest`, it will send half of the tokens to the configured `rewardPool` and the other half is split in two: half of this (or one-fourth of the total rewards) is converted to Krill and sent to the treasury while the other half is sent directly to the treasury.

2.15.1 Privileged Roles

The following functions can be called by the owner of the project:

- `setRewardPool`
- `setTreasury`
- `setUnirouter`
- `setNativeToKrillRoute`
- `inCaseTokensGetStuck`

2.15.2 Issues & Recommendations

Issue #62	**Governance privilege: Contract is highly reconfigurable by the owner, allowing the owner to adjust the parameter so the owner receives all WMATIC fees in the contract**
Severity	● MEDIUM SEVERITY
Description	Currently the contract has many configuration functions: `setRewardPool`, `setTreasury`, `setUniroute` and `setNativeToKrillRoute`. Each of these functions could be called by a malicious owner to redirect fees from their desired use-case to a new malicious use-case (e.g. transferring tokens to the owner's wallet). It would increase investor confidence to address and potentially reduce this privilege.
Recommendation	Consider whether any of these variables need to be upgraded and if so, consider putting the `KrillFeeBatch` contract behind a sufficiently long timelock so all users are informed well in advance about any potential updates to the parameters.
Resolution	✓ RESOLVED

Issue #63	`wNativeToKrillRoute` is not initialized in the constructor
Severity	● LOW SEVERITY
Location	Line 1027 `address[] public wNativeToKrillRoute = [wNative, krill];`
Description	The `wNativeToKrillRoute` is currently initialized with the zero variables `wNative` and `krill`. This requires another `setNativeToKrillRoute` call after contract deployment which could be accidentally forgotten. Until this call is made, no harvests are possible

Recommendation	Consider updating the `wNativeToKrillRoute` within the constructor itself. For example:
	`wNativeToKrillRoute[0] = _wNative;`
	`wNativeToKrillRoute[1] = _krill;`

Resolution	✓ RESOLVED

Issue #64	`harvest` is vulnerable to frontrunning
Severity	● LOW SEVERITY
Description	The `harvest` function swaps a fourth of the WMATIC in the contract to Krill. An arbitrage bot will sandwich this call in a buy and sell transaction around it to profit slightly from the price impact created. This results in a higher purchase price for the contract and a tokenomical loss. If the price impact is minimal, this loss will be minimal as well.
Recommendation	Consider calling `harvest` sufficiently frequently to limit the impact. Since frontrun attacks are less profitable the smaller the purchase amount, frequent harvests will significantly reduce the value lost to them.
Resolution	● ACKNOWLEDGED

2.16 KrillTreasury

The KrillTreasury is a simple token custodian contract. Tokens and matic can be sent to it and the contract owner can withdraw them at any time.

2.16.1 Privileged Roles

The following functions can be called by the owner of the project:

- `withdrawTokens`
- `withdrawMatic`

2.16.2 Issues & Recommendations

Issue #65	**`call` is preferred over `transfer`**
Severity	● INFORMATIONAL
Description	Solidity best practices recommend using `call` over `transfer`. This is because `transfer` has a small gas limit which could (but highly unlikely) revert if operation gas prices are ever adjusted.
Recommendation	Consider using `call` instead of `transfer`. Note that when `call` is used, the success result needs to be handled.
Resolution	✓ RESOLVED

Issue #66	**The owner can withdraw tokens at any time**
Severity	● INFORMATIONAL
Description	The KrillTreasury contract does not contain any inherent locking functionality. In case the use-case of the treasury is for example token-vesting, this could mislead users into thinking that the treasury itself takes care of this. Instead, it should be put behind a timelock to achieve vesting purposes.
Recommendation	Consider whether the treasury will be used for vesting/locking purposes and if so, consider placing it behind a sufficiently long timelock.
Resolution	✓ RESOLVED

PALADIN
BLOCKCHAIN SECURITY