PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Timeleap Finance (Vaults)

16 August 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 the vaults contracts of Timeleap Finance. 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	Timeleap Finance (Vaults)
URL	https://timeleap.finance/
Platform	Polygon
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`BaseStrategy`		
`BaseStrategyLP`	`0x4436Ff0a96C77f38E66de5Dc1225Cfb51Bf88709`	MATCH
`BaseStrategyLPSingle`	`The contracts for all Quickswap LP strategies are the same and based off these contract parts.`	
`StrategyQuickSwap`		
`VaultChef`	`0x62902cc933e5d32717135a05a52c8a46f9b5a323`	MATCH
`StrategyVaultBuyBack`	`The team has decided not to use this contract`	

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	3	3	-	-
Medium	2	2	-	-
Low	9	8	1	-
Informational	17	16	-	1
Total	**31**	**29**	**1**	**1**

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 BaseStrategy

ID	Severity	Summary	Status
01	HIGH	If `panic()` and `unpause()` is ever called in sequence by the governance, an exploiter or governance can abuse the deposit function to steal half of all funds (or even all funds if repeated many times)	RESOLVED
02	MEDIUM	Router can be swapped to siphon transfer fees to any EOA	RESOLVED
03	LOW	Swap fees are sent to the `feeAddress` and `treasuryAddress`, which may be siphoned	RESOLVED
04	LOW	No non-zero validation before swaps can result in the transaction failing	RESOLVED
05	LOW	buyback function sends purchased native tokens to the `treasuryAddress`, which could dump them	RESOLVED
06	LOW	controllerFee, rewardRate and buyBackRate can be modified such that fees only go to the feeAddress	RESOLVED
07	INFORMATIONAL	Comment is outdated for `withdrawFeeFactorLL`	RESOLVED
08	INFORMATIONAL	Lack of events for `resetAllowances`, `pause`, `unpause`, `panic`, `unpanic`, and `setGov`	RESOLVED
09	INFORMATIONAL	The panic function does not revoke approvals to the staking contracts and router	RESOLVED
10	INFORMATIONAL	`_wantAmt > wantLockedTotal()` check seems to be unnecessary in the `_withdraw` function	RESOLVED
11	INFORMATIONAL	Gas optimization: buyback does two swaps while the same behavior can be achieved with a single swap	RESOLVED
12	INFORMATIONAL	Unnecessary addition in `_safeSwap` timestamp	RESOLVED
13	INFORMATIONAL	Calling the funds staked in the underlying staking contract shares is a misnomer	RESOLVED

1.3.2 BaseStrategyLP

No issues found.

1.3.3 BaseStrategyLPSingle

No issues found.

1.3.4 StrategyQuickSwap

ID	Severity	Summary	Status
14	HIGH	The StrategyQuickSwap could be deployed with malicious parameters	RESOLVED
15	MEDIUM	Lack of sanity checks in the constructor for _wantAddress and _earnedAddress	RESOLVED

1.3.5 VaultChef

ID	Severity	Summary	Status
16	LOW	Phising: Malicious strategies could be added as a pool (related to the issue in StrategyQuickSwap)	PARTIALLY RESOLVED
17	LOW	VaultChef does not work with transfer tax tokens	RESOLVED
18	LOW	Strategy can be drained through `withdrawAll` in case the underlying `withdraw` function is vulnerable to reentrancy	RESOLVED
19	LOW	No token validation is done on strategy addition, potentially causing `resetAllowances` to irreversibly fail	RESOLVED
20	INFORMATIONAL	`resetAllowances` and `resetSingleAllowance` should emit events	RESOLVED
21	INFORMATIONAL	`EmergencyWithdraw` event is unused	RESOLVED
22	INFORMATIONAL	`AddPool` event does not contain a `pid` parameter	ACKNOWLEDGED

1.3.6 StrategyVaultBuyBack

ID	Severity	Summary	Status
23	HIGH	If `panic()` and `unpause()` is ever called in sequence by the governance, an exploiter or governance can abuse the deposit function to steal half of all funds (or even all funds if repeated many times)	RESOLVED
24	LOW	`buyback` function sends purchased native tokens to the `treasuryAddress`, which could dump them	RESOLVED
25	INFORMATIONAL	The StrategyVaultBuyBack contract could be deployed with malicious parameters	RESOLVED
26	INFORMATIONAL	`earn`, `resetAllowances`, `panic` and `setGov` should emit events	RESOLVED
27	INFORMATIONAL	The `panic` function does not revoke approvals to the staking contracts	RESOLVED
28	INFORMATIONAL	`_wantAmt > wantLockedTotal()` check seems to be unnecessary in the `_farm` function	RESOLVED
29	INFORMATIONAL	Gas optimization: `buyback` does two swaps while the same behavior can be achieved with a single swap	RESOLVED
30	INFORMATIONAL	Unnecessary addition in `_safeSwap` timestamp	RESOLVED
31	INFORMATIONAL	Calling the funds staked in the underlying staking contract shares is a misnomer	RESOLVED

2 Findings

2.1 BaseStrategy

The BaseStrategy contract is owned by VaultChef. In the `deposit` function, `want` tokens are transferred in and the internal `_farm` function is called, where the before-after method is used in calculating how many shares were received in exchange for staking the `want` tokens. Initially, a withdrawal fee of 0.1% is set. This withdrawal fee can be changed to a maximum value of 0.2%.

2.1.1 Privileged Roles

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

- `deposit`
- `withdraw`
- `resetAllowances`
- `pause`
- `unpause`
- `panic`
- `unpanic`
- `setGov`
- `setSettings`

2.1.2 Issues & Recommendations

Issue #01	**If `panic()` and `unpause()` is ever called in sequence by the governance, an exploiter or governance can abuse the `deposit` function to steal half of all funds (or even all funds if repeated many times)**
Severity	HIGH SEVERITY
Description	After `panic()` is called by governance to withdraw all funds from the underlying staking contract, the funds remain idle in the strategy. However, when a deposit is made, the `_farm` function assumes that all funds added to the underlying staking contract were added by the user. An exploiter will simply call `deposit()` after `panic()` and `unpause()` is called to double all shares in the vault and they will be able to withdraw half of the vault's balance.
Recommendation	Consider removing the `unpause` function and renaming the `unpanic` function to `unpause` since it will correctly stake the idle balance again. In general, consider only calling `panic()` once to permanently disable the strategy. The reasoning is because calling `panic()` and `deposit()`/`harvest()` repeatedly could cause loss of funds by users and possibly gain by governance (for example through performance fees in a strategy where the `earn` token address is equal to the staking token address).
Resolution	RESOLVED The recommendation has been implemented.

Issue #02	Router can be swapped to siphon transfer fees to any EOA
Severity	MEDIUM SEVERITY
Description	The Gov address can update the router that is used for LP pair generation to any contract they desire. This could very well be a malicious contract that takes all transfer fees and sends them to the operator. A malicious router could also allow reentrancy and manipulation through code injection.
Recommendation	Consider removing this ability to change routers, or setting the Gov privilege behind a sufficiently long Timelock of for example at least 31 day to give users sufficient time to react appropriately if required.
Resolution	RESOLVED The router can no longer be swapped.

Issue #03	**Swap fees are sent to the `feeAddress` and `treasuryAddress`, which may be siphoned**
Severity	LOW SEVERITY
Description	The comments state that the fees generated will be used to pay for the compounding `earn` function in BaseStrategyLPSingle, though if the fees generated are large enough there may be an incentive for those fees to be siphoned away to a personal wallet.
Recommendation	Consider diverting funds to a designated contract that allows for fees to be spent on calling the `earn` function but limits the withdrawal abilities by privileged parties. Alternatively, a simpler solution is to consider being forthright with your community and regularly disclosing the amounts and usage of those accumulated fees in the `feeAddress` and `treasuryAddress`.
Resolution	RESOLVED Timeleap has said they will carefully explain these addresses and their purposes in the documentation of their project.

Issue #04	**No non-zero validation before swaps can result in the transaction failing**
Severity	LOW SEVERITY
Description	Uniswap will revert swaps with a zero amount.
Recommendation	Consider adding an `if (amountIn > 0)` check before all Uniswap router interactions.
Resolution	RESOLVED

Issue #05	**buyBack function sends purchased native tokens to the `treasuryAddress`, which could dump them**
Severity	LOW SEVERITY
Description	Half of the repurchased native tokens are sent to the Treasury address, and the other half to the Masterchef. The Treasury would have the ability to sell those accumulated native tokens, causing the token price to drop. The proceeds may then be withdrawn to a private address. Note also that sending native tokens to the Masterchef may cause dilution of rewards, as the total number of tokens in the Masterchef contract is used to determine the total number of deposits. Sending these native tokens to the Masterchef, despite not staking them, may thus cause rewards to be lowered for other users.
Recommendation	Consider sending the purchased tokens from the buyBack function directly to the burn address, so that users may be confident that the native tokens were truly taken out of circulation so as to have a net positive effect with the buy back and burn policy.
Resolution	RESOLVED The client has explained that there is no buyback policy and that the treasury plays an essential role in the growth and development of the protocol. The client plans to use these funds to develop more exciting features like a lending protocol. In light of transparency and the treasury being essential to the project's development, the client will display the treasury balance on their website. Secondly, the project plans to set the native pool on the Masterchef to zero `allocPoints`, resolving the dilution issue. We would still recommend the project to directly burn these tokens in the unlikely case of a native token Masterchef exploit.

Issue #06	**`controllerFee`, `rewardRate` and `buyBackRate` can be modified such that fees only go to the `feeAddress`**
Severity	LOW SEVERITY
Description	The only requirement in the setSettings function is that the sum of those three fees is 10%. There is no stopping controllerFee to be set to 10%, and the remaining two to be set to 0%, such that all fees are paid to the feeAddress only.
Recommendation	Consider setting minimum and maximum safeguards on each fee variable, such that there will always be fees allocated for the adequate payment of the `earn` and buyBack functions. Alternatively, perhaps it may be desirable to not allow for these fees to be modifiable, and users would be assured in knowing that there are immutable and clearly defined fee allocations.
Resolution	RESOLVED These can no longer be modified.

Issue #07	**Comment is outdated for `withdrawFeeFactorLL`**
Severity	INFORMATIONAL
Description	The comment says 0.2% withdrawal fee but `withdrawFeeFactorLL` is actually 2%.
Recommendation	Consider correcting the comment so that readers are aware that the maximum withdrawal fees is 2% rather than 0.2%.
Resolution	RESOLVED The maximum fee has been adjusted to 0.2%, which matches the comment.

Issue #08	**Lack of events for `resetAllowances`, `pause`, `unpause`, `panic`, `unpanic`, and `setGov`**
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for these functions.
Resolution	RESOLVED

Issue #09	**The `panic` function does not revoke approvals to the staking contracts and router**
Severity	INFORMATIONAL
Description	Currently, `panic` does not revoke approvals to the staking contracts and router; this is often done just in case it ever turns out that the staking contracts could transfer tokens from wallets without the wallet's consent.
Recommendation	Consider revoking the approvals in the `panic` function.
Resolution	RESOLVED The client has implemented the following which sets allowance to zero. `IERC20(wantAddress).safeApprove(uniRouterAddress, 0); // Revoke approval`

Issue #10	`_wantAmt > wantLockedTotal()` check seems to be unnecessary in the `_withdraw` function
Severity	INFORMATIONAL
Description	The check to see if the `_wantAmt` variable exceeds the `wantLockedTotal` sees unnecessary as there is already a prior check in `(_wantAmt > wantAmt)`.
Recommendation	Considering removing this check.
Resolution	RESOLVED The client has stated that this is an intended safeguard in case of `panic()`.

Issue #11	Gas optimization: buyback does two swaps while the same behavior can be achieved with a single swap
Severity	INFORMATIONAL
Description	Within the buyback function, two identical token swaps are made. It would be more gas efficient however to do this in a single swap and split the resulting time balance to the two addresses.
Recommendation	Consider using a single swap call and transferring out the resulting time balance to both addresses afterwards.
Resolution	RESOLVED

Issue #12	Unnecessary addition in `_safeSwap` timestamp
Severity	INFORMATIONAL
Description	Within the `_safeSwap` function, an unecessary addition of 600 seconds is added to the timestamp. This is not necessary and wastes gas for no reason since the timestamp check in a Uniswap router is greater or equal then the present timestamp (and timestamps do not change within a single block).
Recommendation	Replace `now.add(600)` with `now`.
Resolution	RESOLVED `add.(600)` has been removed.

Issue #13	Calling the funds staked in the underlying staking contract shares is a misnomer
Severity	INFORMATIONAL
Description	The contract uses the term "shares" for funds staked in the QuickSwap staking contract. This is misleading for third party auditors since these tokens are in fact not shares but instead simply staked tokens. The function `vaultSharesTotal()` is misleading as well.
Recommendation	Consider renaming the `vaultSharesTotal()` function to for example `totalInUnderlying()`.
Resolution	RESOLVED The client has implemented `totalInUnderlying()`.

2.2 BaseStrategyLP

This contract functions to swap accumulated tokens for earned tokens to be reinvested.

2.2.1 Privileged Roles

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

- `earn`

2.2.2 Issues & Recommendations

No issues found.

2.3 BaseStrategyLPSingle

This contract functions to harvest earned tokens to add to liquidity.

2.3.1 Privileged Roles

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

- `earn`

2.3.2 Issues & Recommendations

No issues found.

2.4 StrategyQuickSwap

StrategyQuickSwap is the strategy for QuickSwap's Dragon Lair.

2.4.1 Issues & Recommendations

Issue #14	The StrategyQuickSwap could be deployed with malicious parameters
Severity	HIGH SEVERITY
Description	In case a buyback vault is instantiated with malicious parameters, a simple code change check with the code in the audited contracts is insufficient. This is because the `_quickSwapAddress` could be set to a malicious contract that steals all staked funds.
Recommendation	Consider only using audited, transparent and safe strategy contracts and publishing the addresses of such verified contracts for each vault pool so that users may be able to ascertain that the strategy for which their funds will be deposited into is safe. There is no alternative resolution to guarantee that only safe strategy contracts are used, unfortunately, as any contract can be specified in the constructor, and thus the onus is on investors to ensure that the strategy contract that is in use is non-malicious. Also consider providing Paladin the full list of initial vaults so these can be verified and added to this audit through our code-verification program.
Resolution	RESOLVED The client has indicated that they will add a page to their documentation indicating the list of strategies and how to verify their deployment parameters.

Issue #15	**Lack of sanity checks in the constructor for `_wantAddress` and `_earnedAddress`**
Severity	MEDIUM SEVERITY
Description	There are no safeguards to ensure that the `_wantAddress` and `_earnedAddress` addresses are not set to the same token address, which may result in unintended negative consequences in the harvesting and withdrawal logics.
Recommendation	Consider setting the requirement in the constructor that prevents these address from overlapping, as such: `require(address(_wantAddress) != address(_earnedAddress), "wantAddress and earnedAddress cannot be the same");`
Resolution	RESOLVED The check has been added.

2.5 VaultChef

The VaultChef contract manages all strategies, deposits and withdrawal interactions across all strategies.

2.5.1 Privileged Roles

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

- `addPool`
- `resetAllowances`
- `resetSingleAllowance`

2.5.2 Issues & Recommendations

Issue #16	**Phising: Malicious strategies could be added as a pool (related to the issue in StrategyQuickSwap)**
Severity	LOW SEVERITY
Description	There are no safeguards in place for investors to know if the underlying strategy contract in the pool which they stake their funds in is malicious, which may result in the loss of funds if they do not carefully inspect the `poolId` parameter in their staking function (which is a difficult thing to do on metamask).
Recommendation	Consider only using audited, transparent and safe strategy contracts and publishing the addresses of such verified contracts for each vault pool so that users may be able to ascertain that the strategy for which their funds will be deposited into is safe. There is no alternative resolution to guarantee that only safe strategy contracts are used, unfortunately, as any contract can be specified in the `addPool` function, and thus the onus is on investors to ensure that the strategy contract that is in use is non-malicious. A partial resolution could be a sufficiently long timelock (eg. 24 hours) so the addition of new pools is announced beforehand.
Resolution	PARTIALLY RESOLVED The project has indicated that they will put the VaultChef behind a 24 hour timelock.

Issue #17	VaultChef does not work with transfer tax tokens
Severity	LOW SEVERITY
Description	During the deposit function, the `_wantAmt` (amount of tokens to deposit) is forwarded to the strategy without validating if all these tokens actually arrived in the VaultChef. This will cause the implementation of any strategy with a transfer-tax token to fail and prevent deposits from happening.
Recommendation	Consider using the before-after pattern of handling deposits to find out how many tokens are actually transferred in through the `safeTransferFrom` function.
Resolution	RESOLVED The before-after pattern has been implemented to allow for transfer-tax tokens.

Issue #18	Strategy can be drained through `withdrawAll` in case the underlying `withdraw` function is vulnerable to reentrancy
Severity	LOW SEVERITY
Description	Although the project correctly contains a `nonReentrant` modifier in `withdraw`, this modifier is forgotten in the similar `withdrawAll` utility function. This allows an exploiter to potentially drain a strategy through reentrancy since the edge-case `sharesRemoved > user.shares` does not follow the check-effects pattern. This issue is marked as low-risk since the odds of a strategy allowing a non-privileged user to reenter are low, furthermore the odds of the edge-case presenting itself are low as well.
Recommendation	Consider implementing the checks-effects-interactions pattern to prevent reentrancy, or introducing reentrancy guards to the function.
Resolution	RESOLVED `withdrawAll` now has a reentrancy guard.

Issue #19	**No token validation is done on strategy addition, potentially causing `resetAllowances` to irreversibly fail**
Severity	LOW SEVERITY
Description	When an EOA or non-ERC-20 contract is used as the `wantAddress` of a pool by accident within `addPool`, `resetAllowances` will fail forever since the `safeApprove` method will fail due to the allowance check in it, which would be called on a non-existent contract.
Recommendation	Consider adding in a test line to the `addPool` function, which will fail if the `want` token of the strategy does not contain a functioning allowance function. This should catch most of the cases where `resetAllowances` might fail. `IERC20(IStrategy(_strat).wantAddress()).allowance(address(this), address(_strat));`
Resolution	RESOLVED The client has implemented the test line.

Issue #20	**`resetAllowances` and `resetSingleAllowance` should emit events**
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

Issue #21	`EmergencyWithdraw` event is unused
Severity	INFORMATIONAL
Description	The VaultChef contains an unused `EmergencyWithdraw` event. This could mislead third-parties into thinking an emergency withdraw function is present.
Recommendation	Consider removing the unused event.
Resolution	RESOLVED

Issue #22	`AddPool` event does not contain a `pid` parameter
Severity	INFORMATIONAL
Description	The `AddPool` event does not contain a parameter indicating the pool id.
Recommendation	Consider adding a `pid` parameter to the event and setting it to `poolInfo.length.sub(1)`.
Resolution	ACKNOWLEDGED The client has stated that there isn't a need for indexed `pid` in `addPool` as they emit the pid in the deposit and withdraw events.

2.6 StrategyVaultBuyBack

This contract is the strategy for the allocation of fees earned to conduct Time token buybacks, and also deposits and withdrawals into strategies.

This strategy was audited with the assumption that no tokens with a transfer tax will be added to it. This assumption is reasonable since the underlying staking contract is QuickSwap, which we assume only allows for LP tokens. If tokens with a transfer tax are added, the `sharesRemoved` variable in the `withdraw` function will malfunction.

2.6.1 Privileged Roles

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

- `deposit`
- `withdraw`
- `earn`
- `pause`
- `unpause`
- `resetAllowances`
- `panic`
- `unpanic`
- `setSettings`
- `setGov`

2.6.2 Issues & Recommendations

Issue #23	**If `panic()` and `unpause()` is ever called in sequence by the governance, an exploiter or governance can abuse the `deposit` function to steal half of all funds (or even all funds if repeated many times)**
Severity	HIGH SEVERITY
Description	After `panic()` is called by governance to withdraw all funds from the underlying staking contract, the funds remain idle in the strategy. However, when a deposit is made, the `_farm` function assumes that all funds added to the underlying staking contract were added by the user. An exploiter will simply call `deposit()` after a `panic()` and `unpause()` call is done to double all shares in the vault and be able to withdraw half of the vaults balance.
Recommendation	Consider removing the `unpause` function and renaming the `unpanic` function to `unpause` since it will correctly stake the idle balance again.
Resolution	RESOLVED `panic` has been removed and `unpanic` now restakes the funds correctly.

Issue #24	**buyBack function sends purchased native tokens to the `treasuryAddress`, which could dump them**
Severity	LOW SEVERITY
Description	Half of the repurchased native tokens are sent to the Treasury address, and the other half to the Masterchef. The Treasury would have the ability to sell those accumulated native tokens, causing the token price to just drop again. The proceeds may then be withdrawn to a private address. Note also that sending native tokens to the Masterchef may cause dilution of rewards, as the total number of tokens in the Masterchef contract is used to determine the total number of deposits. Sending these native tokens to the Masterchef, despite not staking them, may thus cause rewards to be lowered for other users.
Recommendation	Consider sending the purchased tokens from the buyBack function directly to the burn address, so that users may be confident that the native tokens were truly taken out of circulation so as to have a net positive effect with the buy back and burn policy.
Resolution	RESOLVED The client has explained that there is no buyback policy and that the treasury plays an essential role in the growth and development of the protocol. The client plans to use these funds to develop more exciting features like a lending protocol. In light of transparency and the treasury being essential to the project's development, the client will display the treasury balance on their website. Secondly, the project plans to set the native pool on the Masterchef to zero `allocPoints`, resolving the dilution issue. We would still recommend the project to directly burn these tokens in the unlikely case of a native token Masterchef exploit.

Issue #25	**The StrategyVaultBuyBack contract could be deployed with malicious parameters**
Severity	INFORMATIONAL
Description	In case a buyback vault is instantiated with malicious parameters, a simple code change check with the code of the audited contract is insufficient. This is because the `_quickSwapAddress` could be set to a malicious contract that steals all staked funds.
Recommendation	Consider only using audited, transparent and safe strategy contracts, and publishing the addresses of such verified contracts for each vault pool, so that users may be able to ascertain that the strategy for which their funds will be deposited into is safe. There is no alternative resolution to guarantee that only safe strategy contracts are used, unfortunately, as any contract can be specified in the constructor, and thus the onus is on investors to ensure that the strategy contract that is in use is non-malicious. Also consider providing Paladin the full list of initial vaults so these can be verified and added to this audit through our code-verification program.
Resolution	RESOLVED The client has indicated that they will add a page to their documentation indicating the list of strategies and how to verify their deployment parameters.

Issue #26	`earn`, `resetAllowances`, `panic` and `setGov` should emit events
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

Issue #27	The `panic` function does not revoke approvals to the staking contracts
Severity	INFORMATIONAL
Description	Currently `panic` does not revoke approvals to the staking contracts and router, this is often done just in case it ever turns out that these staking contracts could transfer tokens from wallets without the wallets consent.
Recommendation	Consider revoking the approvals in the `panic` function.
Resolution	RESOLVED The client has implemented the following which sets allowance to zero. `IERC20(wantAddress).safeApprove(uniRouterAddress, 0); // Revoke approval`

Issue #28	**`_wantAmt > wantLockedTotal()` check seems to be unnecessary in the `_farm` function**
Severity	INFORMATIONAL
Description	The check to see if the `_wantAmt` variable exceeds the `wantLockedTotal` seems unnecessary as there is already a prior check in `(_wantAmt > wantAmt)`.
Recommendation	Consider removing this check.
Resolution	RESOLVED The client has stated that this is an intended safeguard in case of `panic()`.

Issue #29	**Gas optimization: buyback does two swaps while the same behavior can be achieved with a single swap**
Severity	INFORMATIONAL
Description	Within the buyback function, two identical token swaps are made. It would be more gas efficient however to do this in a single swap and split the resulting time balance to the two addresses.
Recommendation	Consider using a single swap call and transferring out the resulting time balance to both addresses afterwards.
Resolution	RESOLVED

Issue #30	Unnecessary addition in `_safeSwap` timestamp
Severity	INFORMATIONAL
Description	Within the `_safeSwap` function, an unecessary addition of 600 seconds is added to the timestamp. This is not necessary and wastes gas for no reason since the timestamp check in a Uniswap router is greater or equal then the present timestamp (and timestamps don't change within a single block).
Recommendation	Replace `now.add(600)` with `now`.
Resolution	RESOLVED This is not used any more.

Issue #31	Calling the funds staked in the underlying staking contract shares is a misnomer
Severity	INFORMATIONAL
Description	The contract uses the term "shares" for funds staked in the QuickSwap staking contract. This is misleading for third party auditors since these tokens are in fact not shares but instead simply staked tokens. The function `vaultSharesTotal()` is misleading as well.
Recommendation	Consider renaming the `vaultSharesTotal()` function to for example `totalInUnderlying()`.
Resolution	RESOLVED The client has implemented `totalInUnderlying()`.


PALADIN
BLOCKCHAIN SECURITY