PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For DogePup Vaults

18 September 2022

paladinsec.co

info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Paladin is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. Paladin may, at its discretion, claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for DogePup's Vault contracts on the DogeChain network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	DogePup Vault
URL	https://dogepup.dog/
Network	DogeChain
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`PoolVault`	`0xA366FBCa467cAC3F34bCb46366fBa9e2a0Bb070B`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	0	-	-	-
Medium	0	-	-	-
Low	10	6	-	4
Informational	6	6	-	-
Total	**16**	**12**	**-**	**4**

Classification of Issues

Severity	Description
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 PoolVault

ID	Severity	Summary	Status
01	LOW	Rounding vulnerability: `currentShares` could be zero	✓ RESOLVED
02	LOW	Flash deposit allows for theft of yield	✓ RESOLVED
03	LOW	Configurational governance risk: Owner can steal rewards and deployer can deploy the contract with a malicious Masterchef	✓ RESOLVED
04	LOW	Configurational governance risk: Compounding only works if the pair of `token0` and `token1` is the `stakingToken` and the routes and pids are configured correctly	✓ RESOLVED
05	LOW	`calculateHarvestRewards` and `calculateTotalPendingRewards` are returning incorrect values	✓ RESOLVED
06	LOW	Tokens with a changeable fee on transfer or deposit fee could drain the previous depositors of their share value	ACKNOWLEDGED
07	LOW	Withdrawers may receive a disproportionate amount in case of tokens with a fee on transfer or withdrawal fees	ACKNOWLEDGED
08	LOW	`harvest()` exposes sandwich-attack vulnerability	ACKNOWLEDGED
09	LOW	Accumulated dust amounts cannot be withdrawn	ACKNOWLEDGED
10	LOW	`pendingRewards()` is significantly flawed	✓ RESOLVED
11	INFO	Checks-effects-interactions pattern is not adhered to	✓ RESOLVED
12	INFO	Gas optimizations	✓ RESOLVED
13	INFO	Typographical errors	✓ RESOLVED
14	INFO	Lack of events for `emergencyWithdraw` and `inCaseTokensGetStuck`	✓ RESOLVED
15	INFO	`weth`, `burnadd`, `burnToken` and `burnRouter` can be made constant	✓ RESOLVED
16	INFO	`stakingToken` and `pid` can be made immutable	✓ RESOLVED

2 Findings

2.1 PoolVault

`PoolVault` is a straightforward auto-compounding vault that automatically reinvests all earned rewards.

Users can stake tokens into the vault and receive a share, which is calculated based on the number of tokens staked. These tokens will then be staked in the underlying Masterchef.

The feature that distinguishes this vault from others is the unique harvest mechanism which does the following:

1. Harvests all rewards from the underlying Masterchef
2. Transfers a `performanceFee` to the treasury (maximum 5%)
3. Transfers a `callFee` to the caller (maximum 1%)
4. Does a buyback to the `burnToken` which then transfers the `burnToken` to the `burnAddress` (maximum 15%)
5. Calculates the remaining rewards, creates the `lpPair` (`stakingToken`) and deposits it back into the underlying Masterchef

This system ensures a steady increase in the value of the shares which the depositors receive during their deposit, as well as steadily increases the value of the `burnToken` which we assume is a governance token of the project.

2.1.1 Privileged Functions

- `transferOwnership`
- `renounceOwnership`
- `setBurnFee`
- `setPerformanceFee`
- `setSlippage`
- `setCallFee`
- `emergencyWithdraw`
- `inCaseTokensGetStuck`
- `pause`
- `unpause`

2.1.2 Issues & Recommendations

Issue #01	Rounding vulnerability: `currentShares` could be zero
Severity	LOW SEVERITY
Description	`currentShares` is calculated within the deposit function as follows: `currentShares = (_amount * totalShares) / pool;` Since the vault accumulates value over time, the `totalShares / pool` ratio will adapt accordingly, meaning the pool will be larger than `totalShares`. In case the share/pool ratio changes severely, `currentShares` could round down, if `_amount` is very small. Proof of concept: 1. `_amount = 100, totalShares = 10000000000000000000000, pool = 100000000000000000000000` 2. `(100*10000000000000000000000) / 100000000000000000000000` 3. `currentShares = 0` Of course, these numbers are very specific and it is very unlikely a similar issue would occur in production, but this proof of concept shows that it is possible. This results in a loss of user funds without receiving any shares. This can specifically pose a risk if the first depositor has malicious intent. Such a malicious depositor might deposit 1 share and then donate a lot of underlying tokens to the vault, instantly setting the share ratio to be incredibly high. As the shares of new users will round to zero, this malicious user effectively steals all deposits of that vault.
Recommendation	Consider adding a `currentShares > 0` requirement after the calculation.
Resolution	RESOLVED

Issue #02	Flash deposit allows for theft of yield
Severity	LOW SEVERITY
Description	Currently, the `deposit` function does not execute a `harvest` before each deposit. The only way to trigger a harvest is by manually calling `harvest()`. This exposes the following attack-vector: 1. Malicious user deposits 2. Malicious user calls `harvest()` 3. Malicious user withdraws With these steps, the user can essentially steal a part of the yield from the accumulated rewards since the last harvest without staking for that time. However, since the `harvest()` function is expected to be called quite often, the impact of this attack is not huge.
Recommendation	Consider either acknowledging this issue or simply execute a `harvest` at the beginning of the `deposit` function.
Resolution	RESOLVED A harvest is made on each deposit.

Issue #03	**Configurational governance risk: Owner can steal rewards and deployer can deploy the contract with a malicious Masterchef**
Severity	LOW SEVERITY
Description	During contract deployment, the deployer can choose which staking contract the contract uses. A malicious deployer can thus use a malicious contract and then steal all tokens. Within the `harvest()` function, the contract does some logic with the reward tokens and then later creates a `lpPair` which should be the staking token. However, if the deployer decides to set `token0`, `token1`, `token0Route` and `token1Route` as something which results in a different `lpToken` than the staking token, the owner can simply withdraw this `lpToken` via `inCaseTokensGetStuck`.
Recommendation	Since the developer is well-known, we do not expect such behavior. However, it is always essential to check which Masterchef is used before staking in the vault. If desired, the owner can implement a check that the pair of `token0` and `token1` indeed results in the desired `stakingToken`.
Resolution	RESOLVED `token0` and `token1` are configured via the underlying tokens of the `stakingToken`.

Issue #04	**Configurational governance risk: Compounding only works if the pair of `token0` and `token1` is the `stakingToken` and the routes and pids are configured correctly**
Severity	LOW SEVERITY
Description	As explained in a later issue, the owner has the ability to steal a part of the rewards in a special case. Even if the owner does not have a malicious intention, this edge-case where the pair of `token0` and `token1` does not result in the `stakingToken` will cause a malfunction of the compounding feature. The created `lpToken` will then simply sit in the vault contract. The contract interacts a lot with the `UniswapV2Router` for swaps and liquidity adding. These swaps and liquidity adds only work for certain routes which are determined during contract creation. Even though there is no unlimited approval or way to abuse an approval to drain the `stakingToken`, these routes should be validated carefully. One malicious example is listed in the issue below. Additionally, swaps do not work for tokens with a fee on transfer — this should be kept in mind especially for the `earnToken`. Another thing to keep in mind is that the underlying token of `pid` should equal the staking token. This could be fetched from the Masterchef.

Recommendation	Consider initializing `token0` and `token1` using the token parameters within the `lpPair`:
	
	`token0 = IUniswapV2Pair(stakingToken).token0();`
	`token1 = IUniswapV2Pair(stakingToken).token1();`
	
	Consider also validating the beginning and ending of all routes and loading the `stakingToken` from the provided `pid`.

Resolution	RESOLVED

Issue #05	**`calculateHarvestRewards` and `calculateTotalPendingRewards` are returning incorrect values**
Severity	LOW SEVERITY
Description	Within both functions, the returned amount is aggregated from the `pendingAmount` within the `masterchef + available()`. However, `available()` is the balance of the `stakingToken` within the vault.
Recommendation	Consider changing `available()` to `rewardAvailable()` to display the correct return values.
Resolution	RESOLVED

Issue #06	**Tokens with a changeable fee on transfer or deposit fee could drain the previous depositors of their share value**
Severity	LOW SEVERITY
Description	Even though the contract is written in a logic that supports tokens with a fee on transfer, there is an important edge case to be aware of. The `deposit` function calculates `currentShares` based on `_amount`, this will hurt previous depositors in the following scenario: 1. Depositor A deposits 100 tokens when the fee on transfer is 10%: they receive 100 shares for their 100 tokens while the contract receives only 90 tokens. 2. Depositor B deposits 100 tokens when the fee on transfer is 20%: they also receive 100 shares for their 100 tokens while the contract receives only 80 tokens. 3. When a withdrawal is done, the `currentAmount` is based on the balance of the staking token — this means that Depositor B receives the same amount as Depositor A even if his deposit resulted in a smaller amount.
Recommendation	Consider either acknowledging this issue and simply not add tokens with a fee on transfer, or tokens with variable/changeable fees on transfer or changing the logic to fix this special edge-case.
Resolution	ACKNOWLEDGED

Issue #07	**Withdrawers may receive a disproportionate amount in case of tokens with a fee on transfer or withdrawal fees**
Severity	LOW SEVERITY
Description	Within the `withdraw` function, there are two possible scenarios: 1. Withdraw the required amount from the Masterchef if the vault does not have enough tokens and send the received amount to the caller. 2. Send the desired amount directly to the caller in case the vault has enough tokens. Due to this logic, the contract exposes a very nasty edge-case if a token with a transfer tax is used as a staking token: In case #1, the user will receive the `received` amount as payout. In case #2, the user will receive the originate `currentAmount` as payout. For case #1, if the token has a fee on transfer, the user will receive a significantly smaller amount compared to case #2 because the fee on transfer is deducted in the transfer from the Masterchef to the vault, compared to case #2 where no fees on transfer is deducted because there is no first transfer from the Masterchef to the vault.
Recommendation	Since we could not see this edge-case practically happening it will only be marked as low severity and it might simply be enough to acknowledge this issue. However, edge-cases like this should be kept in mind.
Resolution	ACKNOWLEDGED

Issue #08	`harvest()` exposes sandwich-attack vulnerability
Severity	LOW SEVERITY
Description	Within the `harvest` function, the reward token gets swapped various times. The probably most vulnerable swap is the buyback to the `burnToken`. Since we expect that the `burnToken` has a significant lower liquidity than `token0` and `token1`, `_buyback` could be used to execute a sandwich-attack, where the attacker first buys the `burnToken`, then calls `harvest()` and then again sells the `burnToken` for a profit.
Recommendation	Since we expect that `harvest()` is called often enough to limit the significance of the value change from `burnToken()` this issue is only marked as low severity and can therefore just be acknowledged. However, it is important to be aware of these forms of attacks.
Resolution	ACKNOWLEDGED

Issue #09	Accumulated dust amounts cannot be withdrawn
Severity	LOW SEVERITY
Description	Within `harvest()`, after all fees have been deducted, the contract calculates the leftover amount of `earnToken` which is swapped to `token0` and to `token1`. `uint256 lpSwapAmount = ((bal - currentPerformanceFee - currentCallFee - burnSwapAmount) / 2)` Afterwards, the swapped amounts of `token0` and `token1` will be used to add liquidity. However, due to slippage from the swaps, there will always be a small amount of one token left within the contract (it will have a small imbalance between `token0` and `token1`).
Recommendation	Consider adding logic which supports the created dust amounts or simply calling `inCaseTokensGetStuck`, swapping the tokens to the `earnToken` and sending it manually to the contract (the `harvest()` function will always fetch the balance of `earnToken` and work with that).
Resolution	ACKNOWLEDGED The admin will simply call `inCaseTokensGetStuck`, swap it to the `earnToken` and send it to the contract.

Issue #10	`pendingRewards()` is significantly flawed
Severity	LOW SEVERITY
Description	We are unsure about the intention of this function, but the logic is as follows: it fetches the `pendingReward` from the underlying Masterchef for the user address, but since the user should have staked via the vault, this value will probably be zero. Furthermore, it aggregates the return value with the aforementioned value and the shares of the users. We do not exactly understand what the original intention behind this function is.
Recommendation	Consider either removing or fixing the logic of this function.
Resolution	RESOLVED The function has been removed.

Issue #11	Checks-effects-interactions pattern is not adhered to
Severity	INFORMATIONAL
Description	Within the `deposit` function, the CEI pattern is not respected. The transfer of the `stakingToken` from the `msg.sender` to the vault is done at the beginning of the function before the effects occur. Paladin generally recommends following best-practices and this includes the CEI-pattern. However, since the function is guarded by the `notContract` modifier and the contract intends to only use `lpTokens` which generally do not expose a reenter functionality, this recommendation is more of a cosmetic issue.
Recommendation	Consider either acknowledging this issue or move the transfer before line 167.
Resolution	RESOLVED

Issue #12	Gas optimizations
Severity	INFORMATIONAL

Description	We have consolidated the sections which can be further optimized for gas usage below.
	
	Line 178
	`function withdrawAll() external notContract`
	
	The `notContract` modifier is redundant as `withdraw` is already guarded.
	
	Line 410
	`function pause() external onlyAdmin whenNotPaused`
	
	The internal `_pause` function is already guarded with the `whenNotPaused` modifier.
	
	Line 418
	`function unpause() external onlyAdmin whenPaused`
	
	The internal `_unpause` function is already guarded with the `whenPaused` modifier.

Recommendation	Consider implementing the above gas optimizations.
Resolution	RESOLVED

Issue #13	Typographical errors
Severity	INFORMATIONAL

Description	We have consolidated the typographical errors into a single issue to keep the report brief and readable.
	
	Line 16
	`uint256 amount; // number of amount for a user`
	
	The comment should say "*amount of shares* for a user"
	
	Line 17
	`uint256 lastDepositedTime; // keeps track of deposited time for potential penalty`
	
	There is no penalty.
	
	Line 118-120
	`uint256( 115792089237316195423570985008687907853269984665640564039457584007913129639935 )`
	
	Using `uint256(-1)` is sufficient.
	
	Line 153
	`uint256 currentShares = 0;`
	
	Simply writing `uint256 currentShares` is enough.
	
	Line 162
	`user.amount = user.amount + currentShares;`
	
	Line 165
	`totalShares = totalShares + currentShares;`
	
	Using the += pattern makes the code more readable.
	
	Line 350
	`* @dev Swap tokenA for tokenB using factory and router.`
	
	Only the router is used, not the factory.

	Line 378
	`block.timestamp + 600`
	
	It is not necessary to extend the deadline as the function will get executed in the same block.
	
	Line 464
	`user.amount = user.amount - _shares;`
	
	Line 465
	`totalShares = totalShares - _shares;`
	
	Using the `-=` pattern makes the code more readable.

Recommendation	Consider fixing the typographical errors.
Resolution	RESOLVED

Issue #14	**Lack of events for `emergencyWithdraw` and `inCaseTokensGetStuck`**
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 #15	`weth`, `burnadd`, `burnToken` and `burnRouter` can be made constant
Severity	INFORMATIONAL
Description	Variables that are never modified can be indicated as such with the constant keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the variables explicitly constant.
Resolution	RESOLVED

Issue #16	`stakingToken` and `pid` 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 variables explicitly immutable.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY