PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Trader Joe (Auto Pools)

25 July 2023

paladinsec.co info@paladinsec.co

Table of Contents

Disclaimer

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

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

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

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

Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Trader Joe's Auto Pools contracts on the Avalanche 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	Trader Joe
URL	http://traderjoexyz.com/
Platform	Avalanche
Language	Solidity
Preliminary contracts	https://github.com/traderjoe-xyz/auto-pool-token-farm/tree/8c48d92e9e9f743d9dc97204b402a4f836d5075f/src
Resolution #1	https://github.com/traderjoe-xyz/auto-pool-token-farm/tree/564f73c19a1aaed662862fbf95ee05669b6fcb35/src

1.2 Contracts Assessed

Name	Contract	Live Code Match
APTFarm	0x57FF9d1a7cf23fD1A9fd9DC07823F950a22a718C	✓ MATCH
SimpleRewarderPerSec	0xe7CE3fAC9CB2933aB96fE1dc6B9D58d2fb4303eF	✓ MATCH
RewarderFactory	0x501b8AFd35df20f531fF45F6f695793AC3316c85	✓ MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● High	2	1	-	1
● Medium	2	1	-	1
● Low	7	7	-	-
● Informational	5	3	-	2
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 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 Global Issues

ID	Severity	Summary	Status
01	HIGH	GeneralGlobal: Contracts without fallback functions might lock their funds forever governance privileges	✓ RESOLVED

1.3.2 APTFarm

ID	Severity	Summary	Status
02	HIGH	Withdrawals can potentially revert	ACKNOWLEDGED
03	MEDIUM	Users will lose additional rewards when rewarder is changed	ACKNOWLEDGED
04	MEDIUM	Joe can be the staking token	✓ RESOLVED
05	LOW	`joePerSec` can result in an overflow	✓ RESOLVED
06	LOW	Lack of valid PID check	✓ RESOLVED
07	LOW	`pendingTokens()` will revert for native tokens	✓ RESOLVED
08	INFO	Zero input values are not checked	✓ RESOLVED
09	INFO	Lack of validation for `_joe`	✓ RESOLVED

1.3.3 SimpleRewarderPerSec

ID	Severity	Summary	Status
10	LOW	`_tokenPerSec` can be above 1e30	✓ RESOLVED
11	LOW	Contract uses `balanceOf` for `apToken` amount	✓ RESOLVED
12	LOW	Not adhering to the CEI pattern introduces a risk of read-only re-entrancy	✓ RESOLVED
13	INFO	Typographical issues	✓ RESOLVED

1.3.4 RewarderFactory

ID	Severity	Summary	Status
14	LOW	Reward token contract size is checked even for native tokens	✓ RESOLVED
15	INFO	Using a as salt for deployment via `create2` makes the use of `create2` redundant	ACKNOWLEDGED
16	INFO	Changing rewarder implementation may lock funds in previous rewarders	ACKNOWLEDGED

2 Findings

2.1 General Issues

The issues in this section apply to the protocol as a whole.

2.1.1 Issues & Recommendations

Issue #01	Global: Contracts without fallback functions might lock their funds forever
Severity	● HIGH SEVERITY

Description	The `SimpleRewarderPerSec` allows for distributing ether or ERC20 tokens. However, if in fact ether is going to be distributed, this will revert if the depositor is a contract without a fallback function, resulting in all deposited funds locked:
	
	`function _transferNative(address to, uint256 amount) internal {`
	
	`    (bool success,) = to.call{value: amount}("");`
	
	`    if (!success) { revert SimpleRewarderPerSec__TransferFailed(); }`

Recommendation	One fix could be to not require the success of this call, however, that would eventually result in other undesired edge-cases. Another fix would be to only allow EOAs to deposit to the masterchef. We recommend thoroughly thinking about this issue and the potential fix but it all comes back to the first high issue within the APTFarm.
Resolution	✓ RESOLVED

2.2 APTFarm

`APTFarm` is a customized MasterChef which allows users to stake tokens and earn Joe rewards.

The contract owner can create different pools whereas every staking token can only have one assigned pool, each pool has its own reward allocation which is defined by joePerSec. Additionally, a rewarder contract can be attached to each pool which allows users to receive an additional reward token.

The contract relies on a sufficient joe token balance for reward payouts - if there are periods with an insufficient joe token balance, the accumulated amount will be stored in a unpaidRewards variable for each user, which can then be harvested once the contract again has enough funds.

The owner can change the reward rate and the rewarder address for each pool at any time.

2.2.1 Privileged Functions

- `add`
- `set`
- `skim`
- `transferOwnership`

2.2.2 Issues & Recommendations

Issue #02	**Withdrawals can potentially revert**
Severity	● HIGH SEVERITY
Description	As with most masterchefs using an additional rewarder, the call to the rewarder is forced to succeed. This was implemented to prevent an exploit where a user can force the call to run out of gas via gas-griefing to manipulate the staked balance in the rewarder contract. However, this can result in issues where the whole withdrawal functionality reverts, potentially locking user funds, especially in combination with the first high risk issue.
Recommendation	Consider executing a simple call to the rewarder contract, ensuring that the forwarded gas is always sufficient to execute the call. This method prevents exploiters from gas-griefing attacks while still results in a success of the withdrawal call whenever there are issues with the underlying rewarder contract.
Resolution	● ACKNOWLEDGED The team will ensure that the rewarder is always valid.

Issue #03	**Users will lose additional rewards when rewarder is changed**
Severity	● MEDIUM SEVERITY
Description	Whenever the rewarder is changed, users will lose their unclaimed rewards which have been accumulated since the last deposit/withdrawal/ew/harvest.
Recommendation	As a fix is non-trivial, consider communicating every change with the community so they can claim their rewards beforehand.
Resolution	● ACKNOWLEDGED Such changes will be communicated with the community upfront.

Issue #04	Joe can be the staking token
Severity	● MEDIUM SEVERITY
Description	There is no check to ensure that the Joe token cannot be the staking token. This might result in a loss of staker's Joe if Joe can unexpectedly be staked.
Recommendation	Consider implementing a check during the add function to ensure it is not possible to add Joe as a staking token.
Resolution	✓ RESOLVED

Issue #05	`joePerSec` can result in an overflow
Severity	● LOW SEVERITY
Description	`joePerSec` can be set to a huge amount which will result in an overflow during the reward calculation.
Recommendation	Consider setting an upper limit for this variable.
Resolution	✓ RESOLVED

Issue #06	Lack of valid PID check
Severity	● LOW SEVERITY
Description	Throughout the codebase, several functions can be called with non-existent PIDs. This might result in undesired edge-cases.
Recommendation	Consider adding the following check in `_updateFarm`: `if (farm.lastRewardTimestamp == 0) revert APTFarm__InvalidFarmIndex();`
Resolution	✓ RESOLVED

Issue #07	`pendingTokens()` will revert for native tokens
Severity	● LOW SEVERITY

Description	`pendingTokens()` is a `view` function used to retrieve information regarding the pending Joe rewards as well as the additional rewards (if any). It tries to fetch the ERC20 token symbol of the bonus rewards token:
	
	`IRewarder rewarder = farm.rewarder;`
	`if (address(rewarder) != address(0)) {`
	`    bonusTokenAddress = address(rewarder.rewardToken());`
	`    bonusTokenSymbol =`
	`IERC20Metadata(bonusTokenAddress).symbol();`
	`    pendingBonusToken = rewarder.pendingTokens(user);`
	`}`
	
	The issue is that `bonusTokenAddress` can be set to `address(0)` if it is the native token of the blockchain. Therefore, calling `pendingTokens()` for a farm which has additional rewards distributed in native tokens will likely revert.

Recommendation	Consider fetching the token symbol in a safe manner using a low-level statical call.
Resolution	✓ RESOLVED

Issue #08	Zero input values are not checked
Severity	● INFORMATIONAL
Description	It is possible for a user to deposit 0 `apTokens` as well as to pass an empty array as an input for `harvestRewards()` which may cause unexpected behavior.
Recommendation	Consider implementing zero value and zero array length checks.
Resolution	✓ RESOLVED

Issue #09	Lack of validation for _joe
Severity	● INFORMATIONAL
Description	Within the constructor, `_joe` is not validated against `address(0)`. This will result in the contract malfunctioning.
Recommendation	Consider validating `_joe` appropriately.
Resolution	✓ RESOLVED

2.3 SimpleRewarderPerSec

`SimpleRewarderPerSec` is a simple rewarder contract which is used on top of MasterChefs. The `onJoeReward` function is meant to be called during any harvest and balance changing event by the MasterChef and distributes rewards to the users based on their staked amount and staked time. It is important to mention that, unlike most rewarders which contain logic that allows them to be used for more than one pool, this rewarder is used for only one pool=.

This contract is deployed by the `RewarderFactory` using TraderJoe's `ImmutableClone` library.

2.3.1 Privileged Functions

- `setRewardRate`
- `emergencyWithdraw`
- `transferOwnership`

2.3.2 Issues & Recommendations

Issue #10	`_tokenPerSec` can be above 1e30
Severity	● LOW SEVERITY
Description	Even though the natspec specifically mentions the upper limit for the variable, it can still be set above the limit within the `setRewardRate` function. This can result in overflows for raw situations.
Recommendation	Consider validating it the same way as done in the `initialize` function.
Resolution	✓ RESOLVED

Issue #11	Contract uses `balanceOf` for apToken amount
Severity	● LOW SEVERITY
Description	The contract fetches the balance of the staked token within the masterchef, however, this can manipulate the reward share: `uint256 aptSupply = _apToken().balanceOf(address(_aptFarm()));`
Recommendation	Consider fetching the balance in the mapping for the corresponding PID.
Resolution	✓ RESOLVED The proper balance is being passed as an argument to `onJoeReward`. However, the `pendingTokens` function still uses the traditional balance.

Issue #12	Not adhering to the CEI pattern introduces a risk of read-only re-entrancy
Severity	● LOW SEVERITY
Description	The CEI pattern is not adhered to in `SimpleRewarderPerSec.onJoeReward` since all the user-related state changes are made after the rewards are sent to the user. Moreover, unlike the standard MasterChef contract, `SimpleRewarderPerSec` supports native tokens as a reward token. Therefore when rewards are sent in native tokens, the receive/fallback function of the receiver will be called. During this call, `SimpleRewarderPerSec.pendingTokens` and `APTFarm.pendingTokens` will still use the non-updated state and therefore will return a positive amount of pending tokens from the additional rewards even if they were just transferred.
Recommendation	Consider following the CEI pattern in `SimpleRewarderPerSec.onJoeReward`.
Resolution	✓ RESOLVED `onJoeReward` has been refactored — first the old variables are cached, then the new ones are set and the reward balance is calculated based on the old values.

Issue #13	Typographical issues
Severity	INFORMATIONAL
Description	L21 `SimpleRewarderPerSec`, a comment says that the contract owner should "set the block reward" whereas the reward is actually distributed per second. ——— `FarmInfo` memory farm can be taken as the return value from `_updateFarm()` in on`JoeReward` instead of being read again from storage to save gas. ——— L175 `_aptAmount` can be used instead of `user.amount`. ——— `emergencyWithdraw()` can be marked external.
Recommendation	Consider fixing the issues.
Resolution	✓ RESOLVED

2.4 RewarderFactory

RewarderFactory is a factory contract which allows any address with the REWARDER_CREATOR_ROLE to deploy proxy contracts that point to a specific implementation using TraderJoe's ImmutableClone library. This allows specific arguments to be stored directly in the proxy which then will be padded at the end of the calldata allowing the caller to access them via the getter functions in Clone. These arguments are the following:

1. rewardToken
2. apToken
3. aptFarm
4. isNative

Even though the implementation can be changed by the owner, it is meant to deploy SimpleRewarderPerSec contracts for the APTFarm.

2.4.1 Privileged Functions

- setSimpleRewarderImplementation
- grantCreatorRole
- transferOwnership

2.4.2 Issues & Recommendations

Issue #14	**Reward token contract size is checked even for native tokens**
Severity	● LOW SEVERITY

Description	`createRewarder` checks whether the passed `rewardToken` and `apToken` addresses correspond to deployed contracts.
	
	`if (!Address.isContract(address(rewardToken)) || !Address.isContract(address(apToken))) {`
	`    revert RewarderFactory__InvalidAddress();`
	`}`
	
	However, `rewardToken` can be set to be the native token for the corresponding chain via the `isNative` flag. Therefore, the expected `rewardToken` address is `address(0)` but it has no bytecode and the transaction will therefore revert.

Recommendation	Consider checking `Address.isContract(address(rewardToken))` only if `isNative == false`.
Resolution	✓ RESOLVED

Issue #15	Using a nonce as salt for deployment via `create2` makes the use of `create2` redundant
Severity	● INFORMATIONAL
Description	Using `create2` to deploy contracts is useful when one wants to compute deterministic addresses in advance. This is mostly used to send funds to the pre-computed address before the contract is deployed. The latter is probably expected to happen with the `SimpleRewarderPerSec` since a comment mentions the following: *It assumes no minting rights, so requires a set amount of `YOUR_TOKEN` to be transferred to this contract prior.* To compute `create2` addresses, a salt is used as a parameter in contrast to the nonce used with the normal deployment via `create`. However, the salt in the `RewarderFactory` is actually computed as the `keccak256` hash of a nonce which is incremented with each deployment. This makes using `create2` redundant and also sending funds to the pre-computed addresses risky since the contract addresses now again depend on the order in which the contracts are deployed (this is the problem that `create2` aims to solve).
Recommendation	Consider if it is in fact desired to send funds upfront to the deterministic address, and if yes, consider using the immutable arguments for computing the salt instead of a nonce.
Resolution	● ACKNOWLEDGED

Issue #16	Changing rewarder implementation may lock funds in previous rewarders
Severity	● INFORMATIONAL
Description	The `RewarderFactory` implements a function that allows the owner to change the implementation of the rewarders that are deployed. This is risky because as described in the previous issue, `create2` computed addresses are used to send funds before deployment. Therefore, if reward tokens have been sent in advance to a rewarder address computed via `create2` before it was deployed and then the implementation was changed, the rewarder contract cannot be deployed unless the implementation is changed back to the one used to compute the `create2` address. Combined with the previous issue of addresses being dependent on the nonce as well, if the implementation was changed and a rewarder with the new implementation was deployed, the potentially sent funds to the address computed with the previous nonce and implementation will be lost.
Recommendation	Consider if it is in fact desired to send funds upfront to the deterministic address, and if so, consider timelocking the function or remove it and use another factory contract for different implementations.
Resolution	● ACKNOWLEDGED

PALADIN

BLOCKCHAIN SECURITY