PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For AvaOne

22 December 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 AvaOne 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	AvaOne
URL	https://avaone.finance/
Platform	Avalanche
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
AvaOne	0xCF7101f34DBA0f3d5fFafD3D3Aa2b3Fc20C08775	MATCH
StakingPool	0xE2Ba504FfEf6E2264aCA41Bb2DDd3f943124467E 0x6D08Ac2E4a7eF708Da34699a460ee45F92e00bEA [slightly modified]	MATCH
MasterChefAvaoV2	0x8e1535Ef636E9DB2D31F932E7FCD802347aA42d7	MATCH
TraderJoeProxy	0x735FaD61B594656352Dd52BEEff03a832C44495d 0x9111187F9C5Be2af455c7A9621F3d91149ED22b2 0x7e52358cbA96241d964013F1c7A7C4A9F6ea4AF2 0xc57f4A0D0A5ea33b9d51AF5C0C273053C166666F 0xd7Aeb18C48A582BF325E262061591914707FF587 0xe16d9c5c3a0c5492bf9145befa1038c25b83b4be [for Lydia] 0x5d9b17f3e2ed1cf5a220507991e01bdc0c36a87c [for Lydia] 0x234d5be47d9258f225832998bb4d8fc4b22731c7 [for Lydia with small adjustments to allow for staking of LYD itself]	MATCH
PangolinProxy	0xa53d238d4509695c165C24EE29F580505F7A8f46 0x7939C10B4E9bF23318f1c3FD54d0C6dAD15528Ec 0xddD10AEFdA096223bBFD0B8EFA8201578c02Ad83	MATCH
Owned	Dependency	MATCH
Pausable	Dependency	MATCH
SimpleSplitter	0x47b90C25448dfF621b651FF5C94E6EcC4539b67d	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	3	3	-	-
Medium	3	3	-	-
Low	9	9	-	-
Informational	26	24	-	2
Total	**41**	**39**	**-**	**2**

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 AvaOne

ID	Severity	Summary	Status
01	LOW	`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	RESOLVED
02	INFO	Governance functionality is broken	ACKNOWLEDGED
03	INFO	`delegateBySig` can be frontrun and cause denial of service	ACKNOWLEDGED

1.3.2 StakingPool

ID	Severity	Summary	Status
04	HIGH	Reward logic is severely flawed which will eventually cause all deposits to be allocated as rewards	RESOLVED
05	LOW	Reward logic only makes sense if `rewardToken` is equal to `stakingToken`	RESOLVED
06	INFO	`rewardsToken` and `stakingToken` can be made `immutable`	RESOLVED
07	INFO	`msg.sender` is unnecessarily cast to `address(msg.sender)`	RESOLVED
08	INFO	`addToWhitelist` and `removeFromWhitelist` can be made external	RESOLVED
09	INFO	Lack of events for `addToWhitelist` and `removeFromWhitelist`	RESOLVED

1.3.3 MasterChefAvaoV2

ID	Severity	Summary	Status
10	HIGH	`emergencyWithdraw` function that moves underlying tokens back into the Masterchef allows for all staked tokens to be 'rugged' (stolen by governance)	RESOLVED
11	HIGH	`proxyRewardDebt` and `lpSupply` are not correctly set on `emergencyWithdraw`, causing severe malfunction over time	RESOLVED
12	MEDIUM	Phishing risk: Governance could add malicious strategies	RESOLVED
13	MEDIUM	`deposit`, `withdraw` and `emergencyWithdraw` could be vulnerable to reentrancy	RESOLVED
14	MEDIUM	Setting dev to the zero address will break deposit and withdrawal functionality potentially locking all staking tokens if governance is malicious	RESOLVED
15	LOW	`updateEmissionRate` has no maximum safeguard	RESOLVED
16	LOW	Governance privilege: Dev address can take up to 100% of newly minted reward tokens	RESOLVED
17	LOW	`emergencyWithdraw` does not work if the vault did not enable emergency yet which could cause users to lose access to staked funds	RESOLVED
18	LOW	The `pendingTokens` function will revert if `totalAllocPoint` is zero	RESOLVED
19	INFO	Contract does not work for reflective tokens	RESOLVED
20	INFO	`msg.sender` is unnecessarily cast to `address(msg.sender)`	RESOLVED
21	INFO	Duplicate requirement	RESOLVED
22	INFO	`add`, `set`, `rewarderBonusTokenInfo`, `deposit`, `withdraw`, `emergencyWithdraw`, `dev`, `setDevPercent`, `addToWhitelist`, `removeFromWhitelist` and `updateEmissionRate` can be made external	RESOLVED
23	INFO	Lack of events for `setDevPercent`, `addToWhitelist` and `removeFromWhitelist`	RESOLVED
24	INFO	`avao` can be made immutable	RESOLVED
25	INFO	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions	RESOLVED
26	INFO	Division before multiplication causes loss of precision	RESOLVED

1.3.4 TraderJoeProxy and PangolinProxy

ID	Severity	Summary	Status
27	LOW	`setRouting` could allow a malicious governance to take all bought back tokens	RESOLVED
28	LOW	`targetPool`, `singleAvaoPool` and emergencies are private	RESOLVED
29	INFO	Buyback could be abused for arbitrage	RESOLVED
30	INFO	`setRouting`, `setBuybackAndBurn`, `buyback` and `enableEmergency` can be made external	RESOLVED
31	INFO	`depositToken`, `rewardToken`, `controller`, `avalon`, `targetPoolId`, `targetPool`, `singleAvaoPool` and `uniswapRouter` can be made immutable	RESOLVED
32	INFO	Inconsistent allowance checks	RESOLVED
33	INFO	Return values of deposit and withdraw are unused	RESOLVED
34	INFO	Contract does not support a staking token equal to the reward token (eg. JOE)	RESOLVED
35	INFO	2**256 -1 simplifies to -1	RESOLVED
36	INFO	`get24HRewardForPool` reverts if `poolLpAmount` or `totalAllocPoint` is zero	RESOLVED

1.3.5 Owned

ID	Severity	Summary	Status
37	INFO	Zero address can claim ownership	RESOLVED

1.3.6 Pausable

No issues found.

1.3.7 SimpleSplitter

ID	Severity	Summary	Status
38	LOW	`avaone`, `okinaPrime`, `satoku`, `hashercat` and `saito` are private	RESOLVED
39	INFO	`avaone`, `okinaPrime`, `satoku`, `hashercat` and `saito` can be made immutable	RESOLVED
40	INFO	Lack of events for `splitBalanceBetweenAddress`	RESOLVED
41	INFO	`splitBalanceBetweenAddress` can be made external	RESOLVED

2 Findings

2.1 AvaOne

The AvaOne token is an ERC-20 token which will be used as the main reward token for the Masterchef. It allows for AVAO tokens to be minted when the `mint` function is called by the owner of the contract, which at the time of deployment would be the AvaOne team. Ownership is generally transferred to the Masterchef after deployment.

2.1.1 Token Overview

Address	0xCF7101f34DBA0f3d5fFafD3D3Aa2b3Fc20C08775
Token Supply	None
Decimal Places	18
Transfer Max Size	No maximum
Transfer Min Size	No minimum
Transfer Fees	None

2.1.2 Privileged Roles

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

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

2.1.2 Issues & Recommendations

Issue #01	**`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef**
Severity	LOW SEVERITY
Description	The `mint` function could be used to pre-mint tokens for legitimate uses including, but not limited to, the injection of initial liquidity, token presale, or airdrops; however, this function may also be used to pre-mint and dump tokens when the token contract has been deployed but before ownership is set to the Masterchef contract. This risk is prevalent amongst less-reputable projects, and any pre-mints can be prominently seen on the Blockchain.
Recommendation	Consider being forthright if this `mint` function has been used by letting your community know how much was minted, where they are currently stored, if a vesting contract was used for token unlocking, and finally the purpose of the mints.
Resolution	RESOLVED Ownership has been transferred to the Masterchef.

Issue #02	Governance functionality is broken
Severity	INFORMATIONAL
Description	Although there is YAM-related delegation code in the token contract which is usually used for governance and voting, the delegation code can be abused as the delegates are not moved during transfers and burns. This allows for double spending attacks on the voting mechanism. It should be noted that this issue is present in pretty much every single farm out there including PancakeSwap and even SushiSwap.
Recommendation	The broken delegation-related code can be removed to reduce the size of the contract. If voting is ever desired, it can still be done through snapshot.org, used by many of the larger projects.
Resolution	ACKNOWLEDGED

Issue #03	`delegateBySig` can be frontrun and cause denial of service
Severity	INFORMATIONAL
Description	Currently if delegateBySig is executed twice, the second execution will be reverted. It is thus in theory possible for a bot to pick up delegateBySig transactions in the mempool and execute them before a contract can. The issue with this is that the rest of said contract functionality would be lost as well. This could be a problem in case it would have been executed by a contract that would have rewarded you for your delegation for example.
Recommendation	Similar to the broken governance functionality issue, this can just be removed.
Resolution	ACKNOWLEDGED

2.2 StakingPool

The StakingPool contract is a simple staking contract based on Synthetix. It will allow users to stake AVAO tokens and earn more AVAO over time. It can only be used by EOAs or whitelisted contracts. The staking pool does not support a reflective deposit token.

2.2.1 Privileged Roles

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

- `nominateNewOwner`
- `acceptOwnership`
- `recoverERC20`
- `addToWhitelist`
- `removeFromWhitelist`

2.2.2 Issues & Recommendations

Issue #04	**Reward logic is severely flawed which will eventually cause all deposits to be allocated as rewards**
Severity	HIGH SEVERITY
Description	The `addRewardToPool` function bases the new number of rewards based on the number of reward tokens in the contract minus the amount of AVAO staked. Through this they attempt to only distribute tokens which are not allocated yet as they subtract user stakes. Any tokens which are not a part of the user stake are however distributed. This logic is severely incorrect because the non-staked tokens can also already be allocated as rewards. This essentially causes a single token to be allocated as a reward multiple times. When people eventually harvest their tokens, this will cause the AVAO balance to potentially deplete below the total staked supply and eventually possibly to zero.
Recommendation	Consider using the traditional Synthetix logic combined with a `transferFrom` to update the `rewardRate`.
Resolution	RESOLVED The traditional Synthetix logic is now employed.

Issue #05	**Reward logic only makes sense if `rewardToken` is equal to `stakingToken`**
Severity	LOW SEVERITY
Description	The contract is clearly only meant to function if the `rewardToken` is equal to the staking token. Even though the `addRewardToPool` logic is already wrongly defined even when these two variables are equal, it clearly indicates an intent for them to be equal and would definitely not function if these two variables were to be different. Although we are sure the developer already intended these to be the same, it would be valuable to readers and potential forkers if this requirement was made explicit.
Recommendation	Consider removing the two token variables and replacing them with a single immutable AVAO variable.
Resolution	RESOLVED The reward logic has been redesigned allowing for a different staking token.

Issue #06	**`rewardsToken` and `stakingToken` 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`. Given that this contract is meant to use AVAO as both the `rewardsToken` and `stakingToken`, we also recommend replacing these with a single variable to clarify this aspect.
Resolution	RESOLVED The client has indicated they cannot make variables `immutable` at this compiler version.

Issue #07	`msg.sender` is unnecessarily cast to `address(msg.sender)`
Severity	INFORMATIONAL
Description	`msg.sender` is cast to `address(msg.sender)` in multiple locations. This is unnecessary. This also applies to `tx.origin` which is often cast as well.
Recommendation	Consider replacing all occurrences of `address(msg.sender)` with `msg.sender`.
Resolution	RESOLVED

Issue #08	`addToWhitelist` and `removeFromWhitelist` can be made external
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword. 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 marking the above variables as external.
Resolution	RESOLVED

Issue #09	Lack of events for `addToWhitelist` and `removeFromWhitelist`
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

2.3 MasterChefAvaoV2

The MasterChefAvaoV2 is a hybrid contract between a traditional Masterchef and a vault project. It allows individual pools to be backed by a custom "strategy" contract which can do arbitrary things with the funds. Within this report, two such strategies (called proxies) were audited: TraderJoeProxy and PangolinProxy. Even though these contracts are called proxies, they are not upgradeable.

Throughout the lifecycle of the project, the Masterchef will mint AVAO tokens as a reward to the stakers in relation to the amount of tokens they have staked. There is also a second token which can be rewarded by the strategies (proxies).

2.3.1 Privileged Roles

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

- `transferOwnership`
- `renounceOwnership`
- `add`
- `set`
- `dev`

2.3.2 Issues & Recommendations

Issue #10	**`emergencyWithdraw` function that moves underlying tokens back into the Masterchef allows for all staked tokens to be 'rugged' (stolen by governance)**
Severity	HIGH SEVERITY
Description	The contract allows the governance to panic certain proxies (strategies), which unstakes the funds from the underlying protocol and moves them back into the Masterchef. This by itself is a great functionality, but there are presently many ways these could be stolen by the governance by having tokens in the Masterchef. First, the governance can add a malicious proxy that does not send any tokens on withdrawal to the Masterchef. The governance could then call `withdraw` on a pool with this strategy (proxy) and it would then withdraw the tokens which are still present in the Masterchef. Second, there is a second reward token mechanism present which allows a strategy to withdraw any token from the Masterchef — this as well allows for the theft of all funds by a malicious governance. As governance risk is an important factor within the evaluation of third-party reviewers, funds and private investors, we recommend addressing this issue seriously. Presently any token within the masterchef is insecure.
Recommendation	We recommend to simply never have tokens in the Masterchef, not even for strategies which would not stake anything in (eg. their proxy is not set). This way the changes can be kept minimal. Pools without an underlying strategy would use a dummy proxy that simply stores the tokens inside its address and gives them back on withdrawals.
Resolution	RESOLVED Tokens are now no longer moved back into the Masterchef — instead, they remain within the proxy.

Issue #11	**`proxyRewardDebt` and `lpSupply` are not correctly set on `emergencyWithdraw`, causing severe malfunction over time**
Severity	HIGH SEVERITY
Description	The client has introduced a new user variable, `proxyRewardDebt`, but this variable is not reset within `emergencyWithdraw`. It is a common mistake to implement new features but not account for them in the `emergencyWithdraw` function. In this case, the impact seems to be rather minimal in that certain functions would temporarily not work until the user makes a deposit. Secondly the `lpSupply` is not decreased on `emergencyWithdraw`, while it is on normal withdrawals. This causes rewards to potentially be severely incorrect over time.
Recommendation	Consider resetting the `proxyRewardDebt` on `emergencyWithdraw`, similar to how the `rewardDebt` is reset. Consider decreasing the `lpSupply` properly.
Resolution	RESOLVED The recommendation has been implemented.

Issue #12	Phishing risk: Governance could add malicious strategies
Severity	MEDIUM SEVERITY
Description	Although the strategies are immutable, the governance can add contracts with potential malicious code as strategies for new pools. Although existing pools and deposits would be safe, this could mislead users into depositing in potentially unsafe pools, even if they are diligent enough to check the contract address.
Recommendation	Consider setting up a proper governance structure. Short-term solutions could include undergoing KYC with a trusted party or including well-known community members on the multisig that governs these functions. Another great solution is to clearly document this within the docs of the project and explain how users can validate their pool strategy themselves. Finally, having a third-party assess which pools have been validated and are considered great is a decent solution. This issue will be marked as resolved when either of the previously mentioned solutions are implemented, or when the client shows they have implemented another reasonable solution themselves.
Resolution	RESOLVED The owner of the Controller contract will be set to a multi-signature contract using gnosis fork on Pangolin, located at https://multisig.pangolin.exchange.

Issue #13	**`deposit`, `withdraw` and `emergencyWithdraw` could be vulnerable to reentrancy**
Severity	MEDIUM SEVERITY
Description	The `deposit`, `withdraw` and `emergencyWithdraw` functions lack reentrancy guards. If a token is ever added that allows for reentrancy exploit, this pool could be drained through the `emergencyWithdraw` function. In addition, the native token could be drained with a `rewardToken` that allows for reentrancy (this vector is present in `deposit` and `withdraw`).
Recommendation	Consider adding reentrancy guards to the above functions.
Resolution	RESOLVED

Issue #14	**Setting dev to the zero address will break deposit and withdrawal functionality potentially locking all staking tokens if governance is malicious**
Severity	MEDIUM SEVERITY
Description	Within most token contracts, minting or transferring tokens to the zero address will revert the transaction. This could cause deposits and withdrawals to revert because of the mint, which would lock up all staking tokens until governance changes this.
Recommendation	To prevent this from ever happening by accident and to limit governance risks, consider adding a requirement like the following `require(_dev != address(0), "!nonzero");` to the relevant configuration function. There are also other cases which require to be dealt with to prevent this scenario from possibly occuring: 1. `allocPoints` need to be small enough so that it can not cause overflows (enforcement needed in add and set) 2. Emission rate needs to be small enough so that it cannot cause overflows (enforcement needed in `updateEmissionRate`)
Resolution	RESOLVED

Issue #15	**`updateEmissionRate` has no maximum safeguard**
Severity	LOW SEVERITY
Description	The function to update rewards currently has no safeguard on its maximum value. Projects sometimes accidentally update their emission rate to a severely high number either by accident or with malicious intent. By having a maximum value, the code itself enforces the reward rate to always be within a reasonable range, preventing mistakes which cannot be reverted once they are made. This might also boost investor confidence as they know that the governance cannot suddenly set the emission rate to a very high value.
Recommendation	Consider adding a `MAX_EMISSION_RATE` variable and setting it to a reasonable value. `require(_avaoPerSec <= MAX_EMISSION_RATE,"Too high");`
Resolution	RESOLVED

Issue #16	**Governance privilege: Dev address can take up to 100% of newly minted reward tokens**
Severity	LOW SEVERITY
Description	The contract allows for the dev address to take up to 100% of the newly minted reward tokens — this might be seen as excessive by many users.
Recommendation	Consider capping this variable to a reasonable limit or explaining why it might someday need to be set to 100%.
Resolution	RESOLVED This cap is now limited to 50%, which is arguably still high but at least half of what it was initially.

Issue #17	**`emergencyWithdraw` does not work if the vault did not enable emergency yet which could cause users to lose access to staked funds**
Severity	LOW SEVERITY
Description	For many investors, having a reliable emergencyWithdraw function is extremely important as it has a higher degree of reliability compared to the traditional withdrawal method which often has complex reward logic going on that could break. Presently, `emergencyWithdraw` can only be called once governance enables this on the underlying strategy. This might be prohibitive for investors.
Recommendation	Consider replacing the requirement with an `if` statement similar to the following within `emergencyWithdraw`: `if (!pool.proxy.emergencied()) pool.proxy.withdraw(_amount);`
Resolution	RESOLVED

Issue #18	**The `pendingTokens` function will revert if `totalAllocPoint` is zero**
Severity	LOW SEVERITY
Description	In the `pendingTokens` function, at some point a division is made by the `totalAllocPoint` variable. If all pools have their rewards set to zero, this variable will be zero as well. The requests will then revert with a division by zero error.
Recommendation	Consider only calculating the accumulated rewards since the `lastRewardTimestamp` if the `totalAllocPoint` variable is greater than zero. This check can simply be added to the existing check that verifies the `block.timestamp` and `pool.lpSupply`, like so: `if (block.timestamp > pool.lastRewardTimestamp && pool.lpSupply != 0 && totalAllocPoint > 0) {`
Resolution	RESOLVED

Issue #19	Contract does not work for reflective tokens
Severity	INFORMATIONAL
Description	The contract presently does not work for reflective tokens. As we understand that the logic is designed in a way that supporting them would be difficult and inefficient, we have marked this issue as informational.
Recommendation	Consider whether these ever need to be added, if not, consider making this lack of support explicit with a comment above the add function for your team and potential forkers to remember that the system will severely malfunction with such tokens.
Resolution	RESOLVED The client has clearly documented that these are not supported.

Issue #20	`msg.sender` is unnecessarily cast to `address(msg.sender)`
Severity	INFORMATIONAL
Description	msg.sender is cast to `address(msg.sender)` throughout the contract when used with `pool.lpToken.safeTransfer()`. This is unnecessary.
Recommendation	Consider replacing all occurrences of `address(msg.sender)` with `msg.sender`.
Resolution	RESOLVED

Issue #21	Duplicate requirement
Severity	INFORMATIONAL
Description	Within the `setDevPercent`, the requirement that the percentage can be at most 100% of the minted tokens is made twice.
Recommendation	Consider only having this requirement once, and setting it to a more reasonable limit than 100%.
Resolution	RESOLVED

Issue #22	**`add`, `set`, `rewarderBonusTokenInfo`, `deposit`, `withdraw`, `emergencyWithdraw`, `dev`, `setDevPercent`, `addToWhitelist`, `removeFromWhitelist` and `updateEmissionRate` can be made `external`**
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword. 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 marking the above variables as external.
Resolution	RESOLVED

Issue #23	**Lack of events for `setDevPercent`, `addToWhitelist` and `removeFromWhitelist`**
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 #24	**avao 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 variable explicitly immutable.
Resolution	RESOLVED

Issue #25	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions
Severity	INFORMATIONAL
Description	Within `updatePool`, `deposit`, `withdraw` and the pending rewards function, `accAvaoPerShare` is based upon the `lpSupply` variable. `pool.accAvaoPerShare = pool.accAvaoPerShare.add(avaoReward.mul(1e12).div(pool.lpSupply).mul(lpPercent).div(1000));` However, if this `lpSupply` becomes a severely large value, there will be precision errors due to rounding. This is famously seen when pools decide to add meme-tokens which usually have huge supplies and no decimals.
Recommendation	Consider increasing precision to `1e18` across the entire contract. It should be noted that even a precision of `1e18` has been considered small when tokens like PolyDoge were added to masterchefs of our client. In case the client thinks it's realistic that such tokens will be added we recommend testing which precision variable is most appropriate to support them without potentially reverting due to overflows.
Resolution	RESOLVED

Issue #26	Division before multiplication causes loss of precision
Severity	INFORMATIONAL
Description	Throughout the rewarder logic for lpPercent, division occurs before multiplication which causes slight loss of precision.
Recommendation	Whenever chaining multiplication and division calls, always do multiplication before division if overflow reversions are not a concern. This way maximum precision is maintained.
Resolution	RESOLVED

2.4 TraderJoeProxy and PangolinProxy

The TraderJoeProxy is an underlying strategy for the Masterchef — it allows for the staking of LP tokens into the Trader Joe Masterchef and uses those rewards partially as a reward for the Masterchef to distribute and partially to buyback AVAO. Part of the AVAO that is bought back is burned and part of it is distributed to the AVAO staking pool as reward.

The PangolinProxy contract is extremely similar but stakes into Pangolin instead.

2.4.1 Privileged Roles

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

- `enableEmergency`
- `setBuybackAndBurn`
- `setRouting`

2.4.2 Issues & Recommendations

Issue #27	**`setRouting` could allow a malicious governance to take all bought back tokens**
Severity	LOW SEVERITY
Description	The contract allows for the governance to change the exchange route over time to buy back the AVAO tokens. However, if a malicious route was chosen buyback could stop working or equally bad, all bought back tokens could be siphoned to governance.
Recommendation	Consider whether the route needs to be updated over time, if not consider making it set solely during construct creation.
Resolution	RESOLVED `setRouting` has been removed.

Issue #28	**`targetPool`, `singleAvaoPool` and `emergencies` are private**
Severity	LOW SEVERITY
Description	Important variables that third-parties might want to inspect should be marked as public so that these third-parties can easily inspect them through the explorer, web3 and derivative contracts.
Recommendation	Consider marking the above variables as public.
Resolution	RESOLVED

Issue #29	Buyback could be abused for arbitrage
Severity	INFORMATIONAL
Description	If buyback is not called for a long period, it could allow a user to sandwich the call within a buy and sale to take "sandwich" profit. This issue is marked as informational as the client has already added incentive for buyback to be called frequently.
Recommendation	This issue will be automatically marked as resolved as it is purely informational.
Resolution	RESOLVED The client is aware of this theoretical implication.

Issue #30	`setRouting`, `setBuybackAndBurn`, `buyback` and `enableEmergency` can be made external
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword. 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 marking the above variables as external.
Resolution	RESOLVED

Issue #31	**`depositToken`, `rewardToken`, `controller`, `avalon`, `targetPoolId`, `targetPool`, `singleAvaoPool` and `uniswapRouter` 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

Issue #32	**Inconsistent allowance checks**
Severity	INFORMATIONAL
Description	The allowance checks throughout the contract sometimes use >= and sometimes >.
Recommendation	Consider always using greater than to check the allowance.
Resolution	RESOLVED

Issue #33	**Return values of `deposit` and `withdraw` are unused**
Severity	INFORMATIONAL
Description	The return values of `deposit` and `withdraw` are unused.
Recommendation	Consider removing these values and updating the interface in the Masterchef.
Resolution	RESOLVED

Issue #34	**Contract does not support a staking token equal to the reward token (eg. JOE)**
Severity	INFORMATIONAL
Description	The contract does not allow for staking the reward token of the underlying Masterchef because the `deposit` function will deposit the whole balance.
Recommendation	Consider making this requirement explicit in the constructor.
Resolution	RESOLVED The client has created a second contract that allows for these tokens to be equal. It will be used exclusively for Lydia Finance.

Issue #35	**2**256 -1 simplifies to -1**
Severity	INFORMATIONAL
Description	The client uses 2**256 -1 to denote the maximum value of a uint256, however, this code in fact overflows to zero and then subtracts one.
Recommendation	Consider simply using `type(uint256).max`, as is considered best practice. Otherwise `uint256(-1)` is also common.
Resolution	RESOLVED `type(uint256).max` is now used consistently.

Issue #36	**`get24HRewardForPool` reverts if `poolLpAmount` or `totalAllocPoint` is zero**
Severity	INFORMATIONAL
Description	The `get24HRewardForPool` function divides by `poolLpAmount` and `totalAllocPoint` without checking that it can be zero — this causes it to revert while there is no stake yet.
Recommendation	Consider adding in early returns for these cases.
Resolution	RESOLVED

2.5 Owned

The Owned contract is a simple ownership management dependency which allows the assignment of an owner that can access privileged functions. This contract improves upon the traditional Ownable contract with the fact that the new owner needs to accept the ownership.

It should be noted that some contracts within the system still use the traditional Ownable.

2.5.1 Privileged Roles

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

- `nominateNewOwner`
- `acceptOwnership`

2.5.2 Issues & Recommendations

Issue #37	Zero address can claim ownership
Severity	INFORMATIONAL
Description	The contract will always have the pending owner set to the zero address while no owner is pending. If this address were to be claimed by anyone (which is extremely improbable under the common signature scheme), this person could then gain ownership of all contracts using this dependency.
Recommendation	Consider requiring the `nominatedOwner` to be non-zero on `acceptOwnership`.
Resolution	RESOLVED Non-zero checks have been added.

2.6 Pausable

The Pausable contract is a dependency which allows the owner to pause functionality of the contracts which use it.

2.6.1 Privileged Roles

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

- `nominateNewOwner`
- `acceptOwnership`

2.6.2 Issues & Recommendations

No issues found.

2.7 SimpleSplitter

The SimpleSplitter is a governance utility contract to split the AvaOne balance in the contract among the team members following the following split:

- okinaPrime: 30%
- satoku: 30%
- hashercat: 30%
- Treasury: 6%
- saito: 4%

2.7.1 Issues & Recommendations

Issue #38	`avaone`, `okinaPrime`, `satoku`, `hashercat` and `saito` are private
Severity	LOW SEVERITY
Description	Important variables that third-parties might want to inspect should be marked as public so that these third-parties can easily inspect them through the explorer, web3 and derivative contracts.
Recommendation	Consider marking the above variables as public.
Resolution	RESOLVED

Issue #39	`avaone`, `okinaPrime`, `satoku`, `hashercat` and `saito` 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

Issue #40	Lack of events for `splitBalanceBetweenAddress`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the function.
Resolution	RESOLVED

Issue #41	`splitBalanceBetweenAddress` can be made external
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword. 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 marking the variable as external.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY