PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Orca DAO

17 October 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 Orca DAO Finance 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	Orca DAO
URL	https://avai.finance
Platform	Avalanche
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
AVAI	0x346A59146b9b4a77100D369a3d18E8007A9F46a6	MATCH
Bank	0x599D8908f783df097E390E0F53d5F93B68F1126a 0xc029713e92383426c9b387b124c0bf6271d08b80 (WAVAX) 0x4805d6563b36a02c5012c11d6e15552f50066d58 (ETH) 0x1ea60d781376c06693dfb21d7e5951caec13f7e4 (BTC)	MATCH
WAVAXGateway	0x4FFFa5602112fd0C7B327A503F67f229F6D0828A	MATCH
Orca	0x8B1d98A91F853218ddbb066F20b8c63E782e2430	MATCH
PodLeader	0x111E1E97435b57467E79d4930acc4B7EB3d478ad	MATCH
TeamPayment	0x4422fB9aFb547E8ed7A61AC9dE0255C760Ea55C1	MATCH
USDCExchange	0x7352Ef87724E1725595add3CA2a9820775FB41d2	MATCH
VestingWallet	0x318DFBE56155F9999FA913cddcaA5764A2B52134	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	3	3	-	-
Medium	4	2	-	2
Low	12	8	1	3
Informational	39	10	1	28
Total	**58**	**23**	**2**	**33**

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 AVAI

ID	Severity	Summary	Status
01	HIGH	Gov Privilege: Governance can change crucial aspects of the protocol to potentially drain the contracts of all supplied tokens	RESOLVED
02	INFO	Gas optimization: roles can be cached	ACKNOWLEDGED
03	INFO	Redundant `hasRole` requirements	RESOLVED
04	INFO	Lack of events for `changeTreasury`, `setGainRatio`, `setDebtRatio`, `setDebtCeiling`, `setPriceSource`, `setTokenPeg`, `setStabilityPool`, `setGateway`, `setClosingFee`, `setOpeningFee` and `setTreasury`	RESOLVED
05	INFO	There are a few typographical errors within the contract	RESOLVED

1.3.2 Bank

ID	Severity	Summary	Status
06	HIGH	The protocol will likely malfunction for collateral tokens with decimals different than 18 tokens and ChainLink feeds different than 8 decimals	RESOLVED
07	MEDIUM	Once a stability pool is added, only the stability pool can liquidate	RESOLVED
08	LOW	Deflationary tokens are not supported as collateral	RESOLVED
09	LOW	No minimum debt amount	RESOLVED
10	LOW	Gov privilege: The gateway address is very privileged throughout the system	RESOLVED
11	LOW	`getPriceSource` underflows if ChainLink ever returns a negative price by error	RESOLVED
12	LOW	Lack of parameter validation	RESOLVED
13	LOW	`changeTreasury` allows changing the treasury to the zero address	RESOLVED
14	LOW	`destroyVaults` does not adhere to checks-effects-interactions which could allow an exploiter to abuse third-party contracts if a bank an ERC-777 (or similar) token is ever added	RESOLVED
15	LOW	Bad debt might accumulate in the system, especially when less trusted coins are added as collateral	PARTIAL
16	INFO	Gas optimization: roles can be cached	ACKNOWLEDGED

17	INFO	`openingFee` is unused	ACKNOWLEDGED
18	INFO	Event parameters are not indexed	ACKNOWLEDGED
19	INFO	Gas optimization: The code contains many redundancies	PARTIAL
20	INFO	Typographical errors	RESOLVED
21	INFO	Lack of events for `setGateway`, `setStabilityPool`, `setPriceSource` and `setTreasury`	RESOLVED
22	INFO	Tip: Tracking the debt ceiling within the AVAI contract might be safer	ACKNOWLEDGED
23	INFO	`checkLiquidation` reverts instead of returning a boolean	RESOLVED
24	INFO	`supportsInterface` does not return true on all supported interfaces	ACKNOWLEDGED
25	INFO	Gas optimization: AVAI can be burned directly during liquidations	RESOLVED
26	INFO	`debtRatio` and `gainRatio` have ambiguous names	ACKNOWLEDGED
27	INFO	`halfDebt` is a misnomer	ACKNOWLEDGED
28	INFO	`debtRatio` can only be set to very distinct values without much flexibility	ACKNOWLEDGED
29	INFO	Ambiguous naming of treasury vault	ACKNOWLEDGED

1.3.3 WAVAXGateway

ID	Severity	Summary	Status
30	INFO	Using the term `vault` for the bank address is a misnomer	ACKNOWLEDGED
31	INFO	Wrongful usage of `assert`	ACKNOWLEDGED
32	INFO	Lack of explicit check to validate that the vault is in fact a WAVAX vault	ACKNOWLEDGED
33	INFO	Typographical error in the contract	ACKNOWLEDGED

1.3.4 Orca

ID	Severity	Summary	Status
34	INFO	`permit` and `delegateBySig` can be frontrun causing denial of service	ACKNOWLEDGED

1.3.5 PodLeader

ID	Severity	Summary	Status
35	MEDIUM	Governance privilege: No restrictions on deposit fees	ACKNOWLEDGED
36	MEDIUM	`endTimestamp` logic is wrong causing the farm to stop emitting rewards to late	ACKNOWLEDGED
37	LOW	Treasury can be set to the zero address potentially blocking both deposits and withdrawals	ACKNOWLEDGED
38	LOW	Adding a EOA or non-token contract as a pool will break `updatePool` and `massUpdatePools`	ACKNOWLEDGED
39	LOW	Deflationary tokens are not supported	ACKNOWLEDGED
40	INFO	Precision vulnerability causes insufficient rewards for certain tokens	ACKNOWLEDGED
41	INFO	`setRewardsPerSecond` updates historical rewards since the last pool activity as well	ACKNOWLEDGED
42	INFO	`pendingRewards` and `updatePool` reverts while `totalAllocPoint` is zero	ACKNOWLEDGED
43	INFO	Orca and `startTimestamp` can be made immutable	ACKNOWLEDGED
44	INFO	The receive function is unused	ACKNOWLEDGED
45	INFO	`setTreasury` and `rewardsActive` can be made external	ACKNOWLEDGED
46	INFO	The `changedAddress` event is unused	ACKNOWLEDGED
47	INFO	Typographical errors in the contract	ACKNOWLEDGED

1.3.6 TeamPayment

ID	Severity	Summary	Status
48	INFO	The contract malfunctions when teammates are added while payments have already been released	ACKNOWLEDGED
49	INFO	`totalShares`, `totalReleased`, `shares`, `released`, `payee`, `addPayee` and `release` can be made `external`	ACKNOWLEDGED

1.3.7 USDCExchange

ID	Severity	Summary	Status
50	HIGH	Governance privilege: The USDCExchange can be upgraded or configured to potentially take all USDC and minted AVAI	RESOLVED
51	LOW	Lack of maximum mint could be disastrous if USDC loses peg	RESOLVED
52	INFO	The treasury that receives fees can be set to the zero address preventing minting and redeeming of AVAI	RESOLVED
53	INFO	Lack of events for `changeTreasury`, `setUSDCRate`, `setAVAIRate` and `setTreasury`	RESOLVED
54	INFO	Gas optimization: AVAI can be burned directly during redemption	RESOLVED

1.4.8 VestingWallet

ID	Severity	Summary	Status
55	MEDIUM	Owner of the VestingWallet can withdraw all unvested tokens	RESOLVED
56	INFO	Anyone can call a release which might be annoying for the recipient their accounting	ACKNOWLEDGED
57	INFO	Calling release before the `cliff` results in an ambiguous underflow reversion	ACKNOWLEDGED
58	INFO	`_orca`, `_revocable` and revoked should be marked as `public`	ACKNOWLEDGED

2 Findings

2.1 AVAI

Avai is a stable coin token that can be minted by users who deposit collateral into a Bank — they can then use this collateral to take out a Collateralized Debt Position (CDP) of AVAI tokens. The AVAI token is supposed to be kept at a $1 peg naturally by the system. The contract furthermore contains governance functions to adjust crucial parameters of the banks and there is also a governance function to change the actual proxy implementation of all banks.

⚠️ Warning ⚠️

The token is upgradeable and thus considered centralised. The protocol governance should be trusted as it can potentially drain all funds within the system.

2.1.1 Privileged Roles and Actions

Banks

- `Mint [ MINTER_ROLE ]`
- `Burn [ BURNER_ROLE ]`

PAUSER_ROLE governance wallets

- `pause [ high risk ]`
- `unpause`
- `renounceRole`

DEFAULT_ADMIN_ROLE governance wallets

- `addBank [ high risk ]`
- `changeTreasury [ high risk ]`
- `setGainRatio [ high risk ]`
- `setDebtRatio [ high risk ]`
- `setDebtCeiling`
- `setPriceSource [ high risk ]`
- `setTokenPeg [ high risk ]`
- `setStabilityPool [ high risk ]`
- `setGateway [ high risk ]`
- `setClosingFee [ high risk ]`
- `setMintingPaused`
- `setOpeningFee`
- `setTreasury`
- `grantRole [ high risk ]`
- `revokeRole`
- `renounceRole`
- `upgradeToNewBank [ high risk ]`

2.1.2 Issues & Recommendations

Issue #01	**Gov Privilege: Governance can change crucial aspects of the protocol to potentially drain the contracts of all supplied tokens**
Severity	HIGH SEVERITY
Description	The AVAI token is the central controlling interface for most crucial aspects of the Orca System. It allows any governance address with the `DEFAULT_ADMIN_ROLE` set to make crucial modifications to the protocol, including but not limited to upgrading all the important aspects of the protocol since the contracts are upgradeable proxies. Since a role-based mechanism is used, it is difficult for users to validate which wallets have actually been privileged. Third parties who wish to validate this would unfortunately have to go through all events created by the AccessControl dependency to figure out which wallets are currently privileged as there can be many. The main roles are the `DEFAULT_ADMIN_ROLE` which can change all crucial aspects of the system, while wallets with the `PAUSER_ROLE` can freeze all transfers of the AVAI token. Due to the anonymous nature of decentralised finance (DeFi), users have become quite wary of protocols with large privileges and it will likely boost investor confidence if this is addressed seriously. ! It should be noted that through AVAI, other treasurer wallets can be promoted to manage banks. Therefore, important bank variable changes might be made directly without these transactions going through the AVAI contract. All contracts should be carefully inspected for governance wallets.

Recommendation	First, consider making it easier for third parties to discover which wallets have been privileged by including an array of the accounts that have roles — this way, a third-party reviewer can easily iterate over the privileged accounts without resorting to web3 usage. Next, consider designing a strong governance structure where it is unlikely and ideally impossible for the governance to abuse these privileges. A decent short-term solution is doxx-ing or KYC'ing oneself to parties trusted by the community as one will be less inclined to steal funds when their identities are known.
Resolution	RESOLVED Although this risk is still present, the client has undergone an internal KYC session with Paladin. The client has furthermore upgraded the code to allow for inspectable `getRoleMember` and `getRoleMemberCount` functions. Even though already assigned roles will no longer be included in these, if new roles are assigned they will appear in these mappings. We recommend larger investors to go through the role granting events to figure out which wallets have received roles before this mapping was created. The multisig requires two of the two owners to approve transactions before they can commence. Both owner addresses have been verified and KYC'd by Paladin through a video call including a demo transaction and third-party KYC service.

KYC Details	**Signature 1** Nickname: SeaFi Brad Address: 0x6ab5a513a2aade2f5c834403e033c8eb7e594b04 KYC: Yes Address ownership verification: Yes **Signature 2** Nickname: SeaFi Kyle Address: 0x9f8a5b35f5508071cf2304a670eab0803f3737aa KYC: Yes Address ownership verification: Yes It should be noted that collusion between the two signers or the compromise of both of their wallets is not impossible. However, given the previous steps, the risk of this should be significantly less compared to similar protocols.

Issue #02	Gas optimization: roles can be cached
Severity	INFORMATIONAL

Description	Throughout the contract, functions are managed using roles. However, these roles are encoded as constants using the keccak256 hash of the role description.
	
	Because of the way constants work within Solidity (they become pure functions returning the right-hand side), this hash will be recomputed each time the role is fetched. This means that every time a role is used, a keccak256 hash is computed.
	
	We used the following contract to demonstrate this inefficiency:
	
	`contract test {`
	`  bytes32 public constant BURNER_ROLE_EXPENSIVE = keccak256("BURNER_ROLE");`
	`  bytes32 public constant BURNER_ROLE_CHEAP = 0x3c11d16cbaffd01df69ce1c404f6340ee057498f5f00246190ea54220576a848;`
	
	`  function expensive() external view returns (bytes32){`
	`  return BURNER_ROLE_EXPENSIVE;`
	`  }`
	
	`  function cheap() external view returns (bytes32){`
	`  return BURNER_ROLE_CHEAP;`
	`  }`
	`}`
	
	Without optimization, the `cheap()` function was 16 gas less expensive while with 200 rounds of optimization, a reduction of 44 gas was achieved.

Recommendation	Consider hardcoding the roles within the contract.
Resolution	ACKNOWLEDGED The client has indicated that they'd like to keep the code simple and readable for now.

Issue #03	Redundant hasRole requirements
Severity	INFORMATIONAL

Location	Lines 126-128
	`function pause() public onlyRole(PAUSER_ROLE) {`
	`  require(`
	`  hasRole(PAUSER_ROLE, _msgSender()),`
	
	Lines 143-145
	`function unpause() public onlyRole(PAUSER_ROLE) {`
	`  require(`
	`  hasRole(PAUSER_ROLE, _msgSender()),`

Description	The pause and unpause function contain redundant role verification. Since the `onlyRole` modifier already reverts unauthorised users, the `hasRole` requirement will always pass when it is reached.
Recommendation	Consider validating that this code is redundant. If so, consider removing the `hasRole` requirement in favor of only having the modifier.
Resolution	RESOLVED The redundant modifiers have been removed.

Issue #04	**Lack of events for `changeTreasury`, `setGainRatio`, `setDebtRatio`, `setDebtCeiling`, `setPriceSource`, `setTokenPeg`, `setStabilityPool`, `setGateway`, `setClosingFee`, `setOpeningFee` and `setTreasury`**
Severity	INFORMATIONAL
Description	Important functions should emit events to keep a track record of when and how they have been called.
Recommendation	Consider adding events to the above functions.
Resolution	RESOLVED The client has indicated that users can listen to the banks as events are emitted at this level.

Issue #05	**There are a few typographical errors within the contract**
Severity	INFORMATIONAL

Description	The contract contains a few typographic errors:
	`Line 164`
	`changes the Treasury. Can only every be one treasury!`
	This should be "There can only ever be one treasury"
	`Line 214`
	`Set the price source for this vault`
	This comment is wrongly copied as the relevant function sets the "token peg".

Recommendation	Consider resolving the above typographical errors.
Resolution	RESOLVED

2.2 Bank

Within the Orca System, a bank represents a single token which can be used as collateral to mint AVAI against. Users can thus store this token inside vaults in the bank contract. Once collateral is stored in a Bank, this user can then mint AVAI up to a certain percentage of the value stored. Each bank can have a maximum `debtCeiling` which indicates the total amount of AVAI that can be borrowed within this bank, to limit tail risks of the collateral becoming worthless.

There is a `closingFee` which is taken from the collateral when debt is paid off manually or through liquidations. This fee can be freely set by governance. In addition, there is also a penalty for being liquidated when your vault goes under the minimum collateralization ratio. This penalty can also be freely set in percentage terms.

The contract contains the possibility of incorporating a stability pool which has not yet been developed at this point and is thus not audited. Stability pools are a way to automatically manage liquidations in a CDP protocol like the Orca System. Currently, liquidations are done by anyone who wants to profit off the liquidation penalty which is sent to them as renumeration for taking care of the liquidation. As of this report (8 October 2021), half of the position can be liquidated once a vault becomes under-collateralized, and this value can be freely adjusted by governance.

⚠ Warning ⚠

The banks are upgradeable and thus considered centralised. The protocol governance should be trusted as it can potentially drain all funds within the system.

2.2.1 Privileged Roles and Actions

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

- `setStabilityPool [ high risk ]`
- `setGateway [ high risk ]`
- `setClosingFee [ high risk ]`
- `setOpeningFee`
- `setTreasury [ high risk ]`
- `changeTreasury`
- `setGainRatio [ high risk ]`
- `setDebtRatio [ high risk ]`
- `setDebtCeiling`
- `setPriceSource [ high risk ]`
- `setTokenPeg [ high risk ]`
- `grantRole [ high risk ]`
- `revokeRole`
- `renounceRole`

2.2.2 Issues & Recommendations

Issue #06	**The protocol will likely malfunction for collateral tokens with decimals different than 18 tokens and ChainLink feeds different than 8 decimals**
Severity	HIGH SEVERITY
Description	Throughout the contract, the assumption is made that the collateral token has the same number of decimals (18) as AVAI and that the ChainLink feed returns a USD price with 8 decimals. If these decimals are different for a new bank either no borrows can be made or people could borrow freely because of the orders of magnitude difference in what the protocol thinks the user has in collateral vs what they really have.
Recommendation	Consider upgrading the protocol when such tokens are ever added to dynamically fetch the feed's and token's `.decimals()`. It should be noted that this update needs to be made in multiple locations.
Resolution	RESOLVED The client has indicated that they will upgrade the contract once they add a token with a different number of decimals or a ChainLink feed with a different number to 8 decimals.

Issue #07	Once a stability pool is added, only the stability pool can liquidate
Severity	MEDIUM SEVERITY
Description	The protocol can easily be upgraded to include a stability pool. Although such a pool contract was not present in the files we received and the client has explained that there is currently no such contract, it generally has a pool of AVAI tokens which users receive interest on. Then automatically, when vaults are eligible for liquidation, the stability pool is used to liquidate them. Oftentimes, stability pools are not implemented well in the sense that wallets can frontrun and dodge unprofitable liquidations by quickly exiting and re-entering the pool before and after said liquidation. More importantly to the core functionality of the Orca System is that once a stability pool is added, no manual liquidations can occur anymore. This means that all liquidations must go through the stability contract. It is because of this reason this issue was created since a mistake in the stability contract might lead to positions not being able to liquidate. An example of this is when the stability pool runs out of AVAI and there is no fallback mechanism in place.
Recommendation	Although this is not necessarily bad, there should always be guaranteed options for third-parties to trigger liquidations even if the stability pool has run out of liquidity. This issue will be marked as resolved on the note that the client will carefully let their stability pool be audited and/or peer-reviewed when they do come up with an implementation for it. Furthermore, the client needs to understand that their implementation should not stop functioning if the stability pool is "dry" of AVAI. Finally, the client should understand that their stability pool implementation might need some notion of "activation" and "deactivation" time to ensure wallets can't dodge unprofitable liquidations.
Resolution	RESOLVED The client has committed to getting the stability pool audited once they do add one. It is therefore important that users validate from time to time that no stability pool has been secretly created and once one is publicly created, that it is either audited or properly peer-reviewed.

Issue #08	Deflationary tokens are not supported as collateral
Severity	LOW SEVERITY
Description	Many less reputable tokens include a transfer-tax on token transfer. This means when 1 token is sent, the receiver might only receive 0.98 tokens. These tokens are currently not supported by the system as the user would still receive the whole token as withdrawable collateral.
Recommendation	Consider not adding deflationary tokens as they are usually less reputable collateral. If this ever is desired, consider using a before-after balance checking pattern combined with a reentrancy guard to discover the amount of tokens that was actually received. This pattern is common in Masterchefs and the Uniswap router.
Resolution	RESOLVED The client has indicated that they have no plans to add such tokens.

Issue #09	No minimum debt amount
Severity	LOW SEVERITY
Description	The Bank allows users to create a very small debt position — this position could be so small that under times of high fees it becomes unprofitable for third-parties to liquidate the position increasing the chances of bad debt accumulating in the system.
Recommendation	Consider adding a minimum debt amount.
Resolution	RESOLVED The client has added a minimum debt amount. It should be noted that this method is still insufficient to truly prevent the issue since this minimum can be immediately partially paid off again. We recommended the client to upgrade to a locked minimum debt model where the minimum debt can only be unlocked during liquidations or `destroyVault`.

Issue #10	Gov privilege: The gateway address is very privileged throughout the system
Severity	LOW SEVERITY
Description	Each bank can have an associated gateway contract. This contract can then interact with the bank as a proxy for the user. The main use-case is for wrapping and unwrapping native AVAX into an ERC-20 token which the protocol understands. However, since the gateway has privileges to withdraw collateral from any user, it has a large privilege throughout the system and should always be carefully reviewed and audited. This issue is marked as low as Issue #01 already sets the bar for the governance privileges throughout the contract. However, if a proper governance mechanism is ever implemented, this issue will help remind third-parties that the modification of the gateway to for example an EOA is a very, very bad sign.
Recommendation	First, consider making it easier for third-parties to discover which wallets have been privileged by including an array of the accounts that have roles; this way, a third-party reviewer can easily iterate over the privileged accounts without resorting to web3 usage. Next, consider designing a strong governance structure where it is unlikely and ideally impossible for the governance to abuse these privileges. A decent short-term solution is doxx-ing or KYC'ing oneself to parties trusted by the community as one will be less inclined to steal funds when their identities are known. Finally, we recommend that all gateways are audited by an auditor and/or thoroughly peer-reviewed.
Resolution	RESOLVED The client has committed to undergoing KYC with Paladin and has described that all users can view that only the WAVAX bank has such a gateway, as only this bank has any use for such a gateway. Users can furthermore then compare the addresses of these with our audit. The client has committed to undergoing further audits or peer-reviews if they ever plan to add new gateways.

Issue #11	**`getPriceSource` underflows if ChainLink ever returns a negative price by error**
Severity	LOW SEVERITY

Location	Lines 282-286
	`function getPriceSource() public view returns (uint256) {`
	`    // And get the latest round data`
	`    (, int256 price, , , ) = priceSource.latestRoundData();`
	`    return uint256(price);`
	`}`

Description	The ChainLink price is converted to an unsigned integer without validating that it's greater than zero. If it's less than zero, due to underflow, this will result in the system thinking the price is enormous. For example, a price of -1 returned by ChainLink would make the protocol think the asset is worth 1.158e+69 USD. This issue is marked as low severity as the odds of this issue presenting itself might be deemed low.
Recommendation	Consider adding a greater than zero requirement. `require(price > 0, "price <= 0");`
Resolution	RESOLVED A requirement of `price >= 0` has been added.

Issue #12	Lack of parameter validation
Severity	LOW SEVERITY
Description	The contract does not validate important governance variables. Under certain parameterizations, liquidations will become impossible.

Recommendation	Consider adding parameter validation to the initializer:
	`require(minimumCollateralPercentage_ >= 100);`
	`// test that price source works (also in setPriceSource)`
	`(, int256 price, , ,) = AggregatorV3Interface(priceSource_).latestRoundData();`
	`require(price > 0, "malfunctioning price source");`
	And consider adding the following requirements to the relevant set functions (some require the address import by OpenZeppelin).
	`require(gateway.isContract(), "no contract");`
	`// check that liquidations will likely remain possible under the parameterization`
	`require((1000 * gainRatio / debtRatio) * (10000 + closingFee) < 10000**2);`

Resolution	RESOLVED

Issue #13	changeTreasury allows changing the treasury to the zero address
Severity	LOW SEVERITY
Description	Calling `changeTreasury` with `address(0)` as parameter can cause the `TREASURY_ROLE` to be assigned to the zero address. However, this should be done through `renounceRole` instead.
Recommendation	Consider adding `require(to != address(0), "Can not set the role to address 0")`.
Resolution	RESOLVED `changeTreasury` has been removed.

Issue #14	**destroyVaults does not adhere to checks-effects-interactions which could allow an exploiter to abuse third-party contracts if a bank an ERC-777 (or similar) token is ever added**
Severity	LOW SEVERITY
Location	Lines 377-396

```
function destroyVault(uint256 vaultID)
   external
   virtual
   onlyVaultOwner(vaultID)
   nonReentrant
 {
   require(vaultDebt[vaultID] == 0, 'Vault as outstanding debt');

   if (vaultCollateral[vaultID] != 0) {
     token.safeTransfer(msg.sender, vaultCollateral[vaultID]);
   }

   _burn(vaultID);

   delete vaultExistence[vaultID];
   delete vaultCollateral[vaultID];
   delete vaultDebt[vaultID];

   emit DestroyVault(vaultID);
 }
```

Description

The `destroyVault` function first transfers out the collateral before it actually destroys the vault. If this token transfer would allow the `msg.sender` to inject code, like is possible with all ERC-777 tokens (these are not very common though), they could potentially execute arbitrary code in this location including transfering the vault to another address as this functionality is not locked with a reentrancy modifier.

In case derivative protocols would exist or come into existence that make use of the transferability of vaults, this code mistake could be abused to make these protocols believe that they are receiving a valuable vault, by transferring the vault to the third-party protocol before the vault is actually destroyed.

This issue is marked as low risk since the fundamental protocol seems to be secure against potential unsafe external call risks like reentrancy. Furthermore we currently know of no protocols that make use of the vault NFT that could be abused. However, not adhering to the checks-effects-interactions is in general considered bad practice and therefore this issue is still included.

Recommendation	Consider adhering to checks-effects-interactions by moving the collateral transfer to the bottom of the function.
Resolution	RESOLVED The collateral is now transferred at the end of this function.

Issue #15	**Bad debt might accumulate in the system, especially when less trusted coins are added as collateral**
Severity	LOW SEVERITY
Description	The system relies on the assumption that liquidators can liquidate positions that are about to go below 100% collateralization ratio. However, if the collateral would crash to $0 or a very low value in a matter of seconds, it is unlikely that the system can react in time and once it realizes the price is too low, there is no incentive to liquidate the positions causing bad/unhealthy/phantom debt (these are all terms used in the industry for the same issue). It should be noted that this issue is present in almost all lending protocols.
Recommendation	Consider building a treasury that can be used to alleviate bad debt. Less reputable tokens should furthermore have a low `debtCeiling` or be not added as debt at all. In general the system should avoid tokens with higher tail risk — risk of a large sudden drop in value.
Resolution	PARTIALLY RESOLVED The client has indicated they will adjust the debt ceiling according to the risk of the collateral.

Issue #16	Gas optimization: roles can be cached
Severity	INFORMATIONAL

Description	Throughout the contract, functions are managed using roles. However, these roles are encoded as constants using the keccak256 hash of the role description.
	
	Because of the way constants work within solidity (they become pure functions returning the right-hand side), this hash will be recomputed each time the role is fetched. This means that every time a role is used, a keccak256 hash is computed.
	
	We used the following contract to demonstrate this inefficiency:
	
	`contract test {`
	`  bytes32 public constant BURNER_ROLE_EXPENSIVE = keccak256("BURNER_ROLE");`
	`  bytes32 public constant BURNER_ROLE_CHEAP = 0x3c11d16cbaffd01df69ce1c404f6340ee057498f5f00246190ea54220576a848;`
	
	`  function expensive() external view returns (bytes32){`
	`  return BURNER_ROLE_EXPENSIVE;`
	`  }`
	
	`  function cheap() external view returns (bytes32){`
	`  return BURNER_ROLE_CHEAP;`
	`  }`
	`}`
	
	Without optimization, the `cheap()` function was 16 gas less expensive while with 200 rounds of optimization, a reduction of 44 gas happened.

Recommendation	Consider hardcoding the roles within the contract.
Resolution	ACKNOWLEDGED The client has indicated that they would like to keep the code simple and readable for now.

Issue #17	`openingFee` is unused
Severity	INFORMATIONAL
Description	The `openingFee` and the related governance function are unused.
Recommendation	Consider removing variables which are not actively used in the contract to improve the ease of third-parties reviewing these contracts.
Resolution	ACKNOWLEDGED

Issue #18	Event parameters are not indexed
Severity	INFORMATIONAL
Description	None of the events have indexed parameters. Having a parameter indexed will let you search through the events based on individual values of this parameter like an address.
Recommendation	Add indexed on important parameters like addresses and IDs which third-parties or the project UI might want to query on.
Resolution	ACKNOWLEDGED

Issue #19	Gas optimization: The code contains many redundancies
Severity	INFORMATIONAL
Description	The protocol often refetches the price from the ChainLink aggregator multiple times within a single function, it furthermore asserts several math operations as if they could overflow. Finally, several requirements are checked multiple times while flowing through the various functions. As Avalanche is a network with a significant gas cost on operations, this might be undesirable. We decided against pointing out every single redundancy as they might have been added as a precaution against compiler issues for example. ❗ Sometimes `assert` is used while `require` would be more appropriate, for example when `getPriceSource` returns a value of zero, which could occur.
Recommendation	Consider addressing the redundancies if gas optimization is a concern.
Resolution	PARTIALLY RESOLVED Many of these redundancies have been removed.

Issue #20	Typographical errors
Severity	INFORMATIONAL

Description	The contract contains a few typographical errors:
	
	Line 113
	`// 10 dollas`
	
	This should be dollars.
	
	Line 175
	`* @dev changes the Treasury. Can only every be one treasury!`
	
	This should be `* @dev Changes the Treasury. There can only ever be one treasury!`.
	
	Line 289
	`* @dev returns the base token's address`
	
	This comment is wrongly copied. This returns the price of AVAI as if it was returned at the chainlink decimals.
	
	Line 416
	`* @dev ALlows vault owner to deposit ERC20 collateral`
	
	This should be `* @dev Allows vault owner to deposit ERC20 collateral`.

Recommendation	Consider resolving the above typographical errors.
Resolution	RESOLVED

Issue #21	**Lack of events for `setGateway`, `setStabilityPool`, `setPriceSource` and `setTreasury`**
Severity	INFORMATIONAL
Description	Important functions should emit events to keep a track record of when and how they have been called.
Recommendation	Consider adding events to the above functions.
Resolution	RESOLVED

Issue #22	**Tip: Tracking the debt ceiling within the AVAI contract might be safer**
Severity	INFORMATIONAL
Description	The Banks keep track of how much outstanding debt (AVAI) they have minted and will prevent more borrowing once the `debtCeiling` has been reached. It might however make more sense to keep track of these ceilings within the AVAI contract as it will undoubtedly be upgraded less often than the banks and can therefore keep the ceilings intact even through a bad bank upgrade.
Recommendation	Consider moving the `debtCeiling` to AVAI.
Resolution	ACKNOWLEDGED The client prefers to keep this within the banks.

Issue #23	`checkLiquidation` reverts instead of returning a boolean
Severity	INFORMATIONAL
Description	The public `checkLiquidation` function reverts if the position cannot be liquidated, this might not be desirable as third-parties might expect a function that returns a boolean on whether the position can be liquidated when they would query this function, as reversion can be ambiguous (eg. out of gas). Furthermore it (and other locations of the code) contain division by zero's which are not asserted. This is fine since the codebase is using solidity 0.8 which means this code will revert but it's inconsistent with the other math safety checks.
Recommendation	Consider updating the function to return a boolean. Also consider adding an explicit reversion message if the denominator `debtValue` is equal to zero in this and other locations of the code where a division by `debtValue` is made. Currently many of the functions will return ambiguous errors in this case, without a clear error message. Finally, this function can be marked as `external` as it is not used within the contract.
Resolution	RESOLVED

Issue #24	`supportsInterface` does not return `true` on all supported interfaces
Severity	INFORMATIONAL
Description	The `supportsInterface` function does not return true on all supported interfaces, this might lead third-party contracts into thinking this contract does not support certain interfaces which it in fact does.
Recommendation	Consider adding the necessary interfaceIds to a `supportedInterfaces` mapping.
Resolution	ACKNOWLEDGED

Issue #25	Gas optimization: AVAI can be burned directly during liquidations
Severity	INFORMATIONAL

Location	Lines 651-662
	`stablecoin.safeTransferFrom(msg.sender, address(this), halfDebt);`
	
	`vaultDebt[vaultID_] -= halfDebt;`
	
	`uint256 _closingFee = (halfDebt * closingFee * getPricePeg()) /`
	`(getPriceSource() * 10000);`
	
	`vaultCollateral[vaultID_] -= (_closingFee + tokenExtract);`
	`vaultCollateral[treasury] += _closingFee;`
	
	`tokenDebt[msg.sender] += tokenExtract;`
	`stablecoin.burn(address(this), halfDebt);`

Description	The `liquidateVault` function first transfers AVAI to the vault to then burns it — this requires an approval by the user which might be seen as cumbersome by some users, and this also uses slightly more gas.
Recommendation	Consider burning stablecoin directly from `msg.sender`.
Resolution	RESOLVED The client has upgraded to burn the stablecoin directly from `msg.sender`.

Issue #26	debtRatio and gainRatio have ambiguous names
Severity	INFORMATIONAL
Description	Since the debtRatio and gainRatio have different meanings in terms of being a 'ratio', their naming should also be different. The gain ratio is divided by zero within the protocol while the debtRatio isn't for example.
Recommendation	Consider renaming the two variables to indicate that they are divided by different terms.
Resolution	ACKNOWLEDGED

Issue #27	`halfDebt` is a misnomer
Severity	INFORMATIONAL
Description	The contract calls variables halfDebt as half of the debt position is initially liquidated during liquidations, however, as debtRatio is adjustable, so is the liquidation portion causing this variable to not represent half of the vault's debt under different parameterizations of debtRatio.
Recommendation	Consider renaming the aforementioned variable.
Resolution	ACKNOWLEDGED

Issue #28	`debtRatio` can only be set to very distinct values without much flexibility
Severity	INFORMATIONAL
Description	The `debtRatio` value is not very precise: the governance can decide to set it in a way that liquidations can only liquidate 0%, 50%, 66.66%, 75%... of the outstanding debt position. A value of say 25% is not achievable due to the lack of precision.
Recommendation	Consider adding a larger precision to the `debtRatio` term.
Resolution	ACKNOWLEDGED

Issue #29	Ambiguous naming of treasury vault
Severity	INFORMATIONAL
Description	The protocol sends the closing fee collateral to collateral for a 'treasury' vault, however, in the contract treasury is also used for the governance address which causes this naming to be ambiguous and confusing for third-parties that might need to review the code.
Recommendation	Consider renaming the variables to avoid ambiguity.
Resolution	ACKNOWLEDGED

2.3 WAVAXGateway

The WAVAXGateway is used to wrap the WAVAX Bank to allow users to deposit and withdraw AVAX directly.

It should be noted that any WAVAX that is accidentally sent to the contract can be withdrawn by both the governance and unprivileged third parties. However, since this cannot be abused we've not included this exploit as an issue.

2.3.1 Issues & Recommendations

Issue #30	Using the term vault for the bank address is a misnomer
Severity	INFORMATIONAL
Description	The WAVAXGateway uses the term `vault` to signify a bank, this is a misnomer and might confuse third-party reviewers.
Recommendation	Consider renaming `vault` to `bank`.
Resolution	ACKNOWLEDGED

Issue #31	Wrongful usage of assert
Severity	INFORMATIONAL
Location	Line 12 `assert(wavax != address(0));`
Description	The contract asserts a statement which could be failed by a test-case. This is bad practice and not recommended as assert should be reserved for cases which cannot fail.
Recommendation	Consider using `require` instead.
Resolution	ACKNOWLEDGED

Issue #32	Lack of explicit check to validate that the vault is in fact a WAVAX vault
Severity	INFORMATIONAL
Description	The WAVAXGateway does not explicitly check that the provided bank is in fact a WAVAX bank, which might lead to people accidentally using the gateway for non-WAVAX banks. This issue is marked as informational since the governance would have to set the gateway on the non-WAVAX bank which would be very odd.
Recommendation	Consider validating that the bank is WAVAX using an `onlyWAVAXBank` modifier.
Resolution	ACKNOWLEDGED

Issue #33	Typographical error in the contract
Severity	INFORMATIONAL
Description	The contract contains the following typographical error: Line 75 `@dev withdraws avax to the user upon destroying vault` This comment has been wrongfully copied to the getPaid function which in fact receives the AVAX for liquidating other people's vaults.
Recommendation	Consider fixing the typographical error.
Resolution	ACKNOWLEDGED

2.4 Orca

The Orca token is the governance token for Orca DAO. It is a simple ERC-20 token which implements the draft ERC20Permit contract. ERC20Permit is a mechanism which allows you to create a signature that allows another contract to generate an approval for transferring your token without you having to actually do an approve transaction. In addition, it implements the OpenZeppelin ERC20Votes contract which is a clean governance voting implementation of the Compound voting mechanism. It has a total supply of 150 million tokens and no further minting is possible.

2.4.1 Issues & Recommendations

Issue #34	`permit` and `delegateBySig` can be frontrun causing denial of service
Severity	INFORMATIONAL
Description	Currently, if `delegateBySig` or `permit` is executed twice, the second execution will be reverted. It is thus in theory possible for a bot to pick up `delegateBySig` or `permit` 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 since the actual contract execution will revert since both `permit` and `delegateBySig` are not idempotent due to the incremental nonce. This issue is marked as informational as it is even present even in Uniswap v2 pairs and we have yet to see it be abused.
Recommendation	Carefully consider this in derivative contracts, instead of blindly executing `permit` or `delegateBySig` – this could be wrapped in a try-catch for example.
Resolution	ACKNOWLEDGED

2.5 PodLeader

The PodLeader is a simple staking contract based on the MasterChef by SushiSwap. It adds deposit fees and logic to make it compatible with a chain that does not work well with block number-based time tracking. Deposit fees can be set up to 100%.

The contract does not mint tokens directly thus the team must ensure to provide it with sufficient native tokens.

2.5.1 Privileged Roles and Actions

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

- `setTreasury`
- `addRewardsBalance`
- `add`
- `Set`
- `updateDepositFee`
- `setRewardsPerSecond`

2.5.2 Issues & Recommendations

Issue #35	Governance privilege: No restrictions on deposit fees
Severity	MEDIUM SEVERITY
Description	The contract contains no restrictions on deposit fees – this has historically been used to trick people into depositing while the fees are set to 100% to steal their tokens. The review site RugDoc has also started marking these staking contracts as Medium Risk as a result.
Recommendation	Consider adding caps to the deposit fees to boost investor confidence and make the farm safer.
Resolution	ACKNOWLEDGED The PodLeader is not upgradeable and therefore this cannot easily be changed. The client has indicated that this contract is behind a Timelock with Multisig. The client has furthermore committed to undergoing KYC with Paladin.

Issue #36	**endTimestamp logic is wrong causing the farm to stop emitting rewards to late**
Severity	MEDIUM SEVERITY
Location	Lines 520-534

```
function _setRewardsEndTimestamp() internal {
    if (rewardsPerSecond > 0) {
        uint256 rewardFromTimestamp = block.timestamp >=
startTimestamp
        ? block.timestamp
        : startTimestamp;
        uint256 newEndTimestamp = rewardFromTimestamp +
         (orca.balanceOf(address(this)) / rewardsPerSecond);
        if (
          newEndTimestamp > rewardFromTimestamp && newEndTimestamp
!= endTimestamp
        ) {
        emit ChangedRewardsEndTimestamp(endTimestamp,
newEndTimestamp);
        endTimestamp = newEndTimestamp;
        }
    }
}
```

Description	The contract contains logic to automatically determine when the farm should stop since rewards are manually transferred to the PodLeader and will eventually run out. This logic checks the Orca balance of the PodLeader. However, this balance might be allocated to users already who haven't harvested yet, and it could also be stakes from the native pool. It should be noted that the protocol has explained that they will not add a native pool into this contract however.
Recommendation	Consider rewriting the `endTimestamp` logic to account only for tokens sent by `addRewardsBalance` and to subtract any `accumulatedRewards`. Because this contract is already deployed and not upgradeable, the client should stop emissions before the PodLeader runs out of supply. This way, users will still be able to do their final harvest.
Resolution	ACKNOWLEDGED

Issue #37	**Treasury can be set to the zero address potentially blocking both deposits and withdrawals**
Severity	LOW SEVERITY
Description	The governance can call `setTreasury` with the zero address by accident which will block most deposits and withdrawals since transfering the deposit fees to the zero address will revert for most tokens. This issue is marked as low severity since users can still use the `emergencyWithdraw` function to withdraw their positions.
Recommendation	Consider adding a non-zero check. `require(_treasury != address(0));`
Resolution	ACKNOWLEDGED

Issue #38	**Adding a EOA or non-token contract as a pool will break `updatePool` and `massUpdatePools`**
Severity	LOW SEVERITY
Description	`updatePool` will always call `pool.totalStaked;` on the token of this pool, and will fail to do so if it is not a token contract.
Recommendation	Consider simply adding a test line in the add function. If the token does not exist, this will make sure the add function fails. `_lpToken.balanceOf(address(this));`
Resolution	ACKNOWLEDGED

Issue #39	Deflationary tokens are not supported
Severity	LOW SEVERITY
Description	When deflationary tokens are added to the pools, this can result in these tokens becoming unwithdrawable since people can withdraw all tokens they have sent to the PodLeader, while the PodLeader only received a part of these tokens. This issue is marked as low severity as we doubt that the client has a need to add such tokens. We believe the PodLeader mainly exists to stake LP tokens, which do not have such taxes.

Recommendation	Consider using the current standard of handling deposits, which is based on how Uniswap handles transfer fees:
	
	`uint256 balanceBefore = pool.lpToken.balanceOf(address(this));`
	`pool.lpToken.transferFrom(msg.sender, address(this), _amount);`
	`_amount =`
	`pool.lpToken.balanceOf(address(this)).sub(balanceBefore);`
	
	Note that by using this method, you can also add the specific transfer tax logic for the native token.

Resolution	ACKNOWLEDGED

Issue #40	Precision vulnerability causes insufficient rewards for certain tokens
Severity	INFORMATIONAL
Description	Token precision is 1e12 while the preferred precision is 1e18 due to token pools with large supply. In tokens with large supply, precision may be vulnerable due to .div(pool.totalStaked) and rewards may be compromised in these pools.
Recommendation	To prevent this from ever happening consider changing precision of tokens to 1e18. We've also received feedback from clients that for certain "doge coins" an even higher precision is desired (eg. 1e20) since they have such a large supply.
Resolution	ACKNOWLEDGED

Issue #41	setRewardsPerSecond updates historical rewards since the last pool activity as well
Severity	INFORMATIONAL
Description	The setRewardsPerSecond method does not call massUpdatePools causing it to update rewards for pools which have not recently been updated, since the last time they've been updated.
Recommendation	Consider adding a massUpdatePools call within this function to ensure that the function does not increase or decrease any past rewards.
Resolution	ACKNOWLEDGED

Issue #42	`pendingRewards` and `updatePool` reverts while `totalAllocPoint` is zero
Severity	INFORMATIONAL
Description	The functions `pendingRewards` and `updatePool` revert if the `totalAllocPoint` variable is zero, therefore deposit and withdrawals would revert if all `allocPoints` are ever set to zero. Furthermore, the frontend might glitch due to `pendingRewards` reverting.
Recommendation	Consider adding `&& totalAllocPoint != 0` to the `if` statement in pendingRewards. Within `updatePool`, this check can be added to the if statement which updates the last reward timestamp and returns.
Resolution	ACKNOWLEDGED

Issue #43	`Orca` and `startTimestamp` can be made `immutable`
Severity	INFORMATIONAL
Description	Variables set in the constructor but not changed throughout the contract can be marked as `immutable` to communicate this to third-party reviewers and also save on gas cost.
Recommendation	Consider marking the above variables as immutable.
Resolution	ACKNOWLEDGED

Issue #44	The `receive` function is unused
Severity	INFORMATIONAL
Description	The PodLeader still contains a receive function that allows accepting of AVAX tokens, however since the current implementation uses ERC-20 tokens exclusively, this function is obsolete and might lead to people accidentally sending the contract AVAX.
Recommendation	Consider removing the `receive` fallback function.
Resolution	ACKNOWLEDGED

Issue #45	`setTreasury` and `rewardsActive` can be made `external`
Severity	INFORMATIONAL
Description	Functions that are not used within the contract can be marked as `external` to indicate this behavior to third-party reviewers and to save gas on certain occasions.
Recommendation	Consider marking the above functions as `external`.
Resolution	ACKNOWLEDGED

Issue #46	The `changedAddress` event is unused
Severity	INFORMATIONAL
Description	The contract contains events that are unused.
Recommendation	Consider removing the above events.
Resolution	ACKNOWLEDGED

Issue #47	Typographical errors in the contract
Severity	INFORMATIONAL

Description	The contract contains the following typographical errors:
	
	Line 184
	`DO NOT add the same token more than once. Rewards will be messed up if you do.`
	
	This comment seems incorrect since the `totalStaked` variable is now used for token supplies. We believe multiple pools with the same token can be added without messing up the rewards because of this. Traditional Masterchefs would use the Masterchef balance to discover the token supply, which would of course cause double-counting when multiple pools were present.
	
	`* @param _depositFeeBp If true, users get voting power for deposits`
	
	This comment does not explain the purpose of deposit fees and might therefore mislead third-party reviews.

Recommendation	Consider fixing the above typographical errors.
Resolution	ACKNOWLEDGED

2.6 TeamPayment

The TeamPayment is a simple utility contract for the Orca team to split Orca tokens they receive over their team members. It allows configuring the contract with team member wallets and their respective shares of the total pot. Orca sent to the TeamPayment contract can then be withdrawn by the team members pro-rata their shares.

2.6.1 Privileged Roles and Actions

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

- `addPayee`

2.6.2 Issues & Recommendations

Issue #48	**The contract malfunctions when teammates are added while payments have already been released**
Severity	INFORMATIONAL
Description	When teammates are added in hindsight after payments have already been released, these new teammates receive a claim on all previous payments pro-rata their current share. This is of course not possible since at this point there will always be insufficient Orca in the contract to pay out everyone.
Recommendation	Consider only allowing `addPayee` to be called while no funds have been released: `require(totalReceived == 0, "already started");`
Resolution	ACKNOWLEDGED

Issue #49	**`totalShares`, `totalReleased`, `shares`, `released`, `payee`, `addPayee` and `release` can be made `external`**
Severity	INFORMATIONAL
Description	Functions that are not used within the contract can be marked as `external` to indicate this behavior to third-party reviewers and to save gas on certain occasions.
Recommendation	Consider marking the above functions as `external`.
Resolution	ACKNOWLEDGED

2.7 USDCExchange

The USDCExchange is a contract that can mint AVAI to users at 1.0075 USDC and therefore puts an upper limit on the price of AVAI. The small fee compared to the 1:1 price is sent to the treasury address. Furthermore, the USDCExchange provides a soft lower limit on the price as well: while it has sufficient USDC, users can repurchase USDC from the contract at a quote of 0.9925 USDC per AVAI. Thus, while the contract has USDC in the reserves, the lower-limit of the AVAI price is 0.9925 USDC. However, once the reserves run out, the buying pressure is determined similar to DAI, based on the idea that borrowers will want to repurchase the cheaper AVAI to pay off their debts at a discount.

Warning: The USDCExchange is upgradeable and thus considered centralised. The protocol governance should be trusted as it can potentially drain all USDC within the system.

2.7.1 Privileged Roles and Actions

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

- `changeTreasury`
- `setUSDCRate`
- `setAVAIRate`
- `renounceOwnership`
- `transferOwnership`

2.7.2 Issues & Recommendations

Issue #50	**Governance privilege: The USDCExchange can be upgraded or configured to potentially take all USDC and minted AVAI**
Severity	HIGH SEVERITY
Description	The governance proxy admin can upgrade the USDCExchange to a malicious contract to potentially take out all USDC and mint an enormous amount of AVAI to dump. Furthermore, the USDCExchangeOwner can adjust the default exchange rates freely which could be abused to give themselves very favorable exchange rates to also take out all USDC and mint a large amount of AVAI to dump. Due to the anonymous nature of DeFi, users have become quite wary of protocols with large privileges and it will likely boost investor confidence to address this seriously.
Recommendation	Consider designing a strong governance structure where it is unlikely and ideally impossible for the governance to abuse these privileges. A decent short-term solution is doxx-ing or KYC'ing oneself to parties trusted by the community as one will be less inclined to steal funds when their identities are known.
Resolution	RESOLVED Although this risk is still present, the client has gone through an internal KYC session with Paladin. Furthermore the client has indicated that these privileges are behind a timelocked multisig contract and they have implemented basic governance structures to reduce the likelihood of this issue happening.

Issue #51	Lack of maximum mint could be disastrous if USDC loses peg
Severity	LOW SEVERITY
Description	Although USDC is one of the safest stablecoins out there, there is a very small non-zero chance that it does lose peg and become worthless due to some edge-case event. In this case, AVAI would go down together with USDC as USDC can be freely exchanged to AVAI without limit at a price of 1.0075 USDC per AVAI. AVAI will just keep on being minted until it reaches the low price of USDC.

Recommendation	Consider adding a buffer to the amount of AVAI that can be minted within a period. For example, expanding the supply by 5% every hour might be appropriate.
	`mapping(uint256 => uint256) accumulatedAVAI;`
	`…`
	`uint256 period = block.timestamp / (1 hour);`
	`require(accumulatedAVAI[period] + amountToSend <= avai.totalSupply() * hourlyLimit / 10000, “too much avai minted this hour”);`
	`accumulatedAVAI[period] += amountToSend;`

Resolution	RESOLVED An hourly limit has been added once the total supply of AVAI exceeds 1,000,000 AVAI. The hourly limit is set in basis points of the total supply.

Issue #52	The treasury that receives fees can be set to the zero address preventing minting and redeeming of AVAI
Severity	INFORMATIONAL
Description	By calling `changeTreasury` with the zero address as a parameter all minting and redeeming will be prevented because usdc does not allow transfers (of the fees) to the zero address.
Recommendation	Consider adding a requirement to prevent this. `require(newTreasury != address(0), “zero address”);`
Resolution	RESOLVED

Issue #53	Lack of events for `changeTreasury`, `setUSDCRate`, `setAVAIRate` and `setTreasury`
Severity	INFORMATIONAL
Description	Important functions should emit events to keep a track record of when and how they have been called.
Recommendation	Consider adding events to the above functions.
Resolution	RESOLVED

Issue #54	Gas optimization: AVAI can be burned directly during redemption
Severity	INFORMATIONAL

Location	Lines 104-108
	`avai.safeTransferFrom(msg.sender, address(this), amount);`
	`// Burn excess, keep fee`
	`avai.burn(address(this), amount);`
	`// Transfer amount minus fees to sender`
	`usdc.safeTransfer(msg.sender, amountToSend);`

Description	The `redeem` function first transfers AVAI to the USDCExchange contract to then burn it. This requires an approval by the user which might be seen as cumbersome by some users, and it also uses slightly more gas.
Recommendation	Consider burning stablecoin directly from `msg.sender`.
Resolution	RESOLVED

2.8 VestingWallet

The VestingWallet is a simple linear vesting schedule contract with a start timestamp (`cliff`). If the contract is configured to be revocable, the contract owner can withdraw all tokens that are still being vested to the recipient.

2.8.1 Privileged Roles and Actions

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

- `receiveToken`
- `revoke`
- `ransferOwnership`
- `renounceOwnership`

2.8.2 Issues & Recommendations

Issue #55	**Owner of the VestingWallet can withdraw all unvested tokens**
Severity	MEDIUM SEVERITY
Description	For VestingWallets that are revocable, the owner can withdraw all tokens which have not vested yet. This might be misleading for third-party investors who think these tokens are locked up.
Recommendation	This issue will be marked as resolved if the client shows that all active VestingWallets either have `_revocable` set to `false` or have their ownership renounced.
Resolution	RESOLVED Ownership has been renounced.

Issue #56	**Anyone can call a `release` which might be annoying for the recipient their accounting**
Severity	INFORMATIONAL
Description	The `release` method can be called by anyone. If the recipient is in a jurisdiction where they have to account for each important transaction this might bring them a lot of work.
Recommendation	Consider making release only callable by owner and beneficiary.
Resolution	ACKNOWLEDGED

Issue #57	Calling release before the `cliff` results in an ambiguous underflow reversion
Severity	INFORMATIONAL
Description	Before the `cliff` timestamp has been reached (`_start`), calling `release` will result in an ambiguous error.
Recommendation	Consider adding a nicer error: `require(block.timestamp >= _start, "Not started");`
Resolution	ACKNOWLEDGED

Issue #58	`_orca`, `_revocable` and `revoked` should be marked as public
Severity	INFORMATIONAL
Description	Important variables that might be of value to third-parties including the beneficiary should be marked as public.
Recommendation	Consider marking the above variables as `public` so that the beneficiary can for example inspect that their vesting wallet is non-revocable.
Resolution	ACKNOWLEDGED


PALADIN
BLOCKCHAIN SECURITY