



PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Vesq

10 December 2021



paladinsec.co



info@paladinsec.co

Table of Contents

Table of Contents	2
Disclaimer	5
1 Overview	6
1.1 Summary	6
1.2 Contracts Assessed	7
1.3 Findings Summary	8
1.3.1 Global Issues	9
1.3.2 VSQERC20	9
1.3.3 sVSQERC20	10
1.3.4 wsVSQ	10
1.3.5 BondDepository	10
1.3.6 EthBondDepository	11
1.3.7 Staking	11
1.3.8 StakingDistributor	11
1.3.9 StakingHelper	12
1.3.10 StakingWarmup	12
1.3.11 StandardBondingCalculator	12
1.3.12 Treasury	13
1.3.13 VSQZapIn	13
1.3.14 Code style-related Issues	14
2 Findings	15
2.1 Global Issues	15
2.1.1 Issues & Recommendations	16
2.2 VSQERC20	27
2.2.1 Token Overview	27
2.2.2 Privileged Roles	27
2.2.3 Issues & Recommendations	28

2.3 sVSQERC20	30
2.3.1 Token Overview	30
2.3.2 Privileged Roles	31
2.3.3 Issues & Recommendations	32
2.4 wsVSQ	35
2.4.1 Token Overview	35
2.4.2 Issues & Recommendations	36
2.5 BondDepository	37
2.5.1 Privileged Roles	37
2.5.2 Issues & Recommendations	38
2.6 EthBondDepository	47
2.6.1 Privileged Roles	47
2.6.2 Issues & Recommendations	48
2.7 Staking	51
2.7.1 Privileged Roles	52
2.7.2 Issues & Recommendations	53
2.8 StakingDistributor	55
2.8.1 Privileged Roles	55
2.8.2 Issues & Recommendations	56
2.9 StakingHelper	59
2.9.1 Issues & Recommendations	60
2.10 StakingWarmup	61
2.10.1 Privileged Roles	61
2.10.2 Issues & Recommendations	61
2.11 StandardBondingCalculator	62
2.11.1 Issues & Recommendations	63
2.12 Treasury	66
2.12.1 Privileged Roles	67
2.12.2 Issues & Recommendations	68
2.13 VSQZapIn	74

2.13.1 Privileged Roles	74
2.13.2 Issues & Recommendations	75
2.14 Code style-related Issues	76
2.14.1 Issues & Recommendations	77



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 advocacy 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 Vesq on the Polygon 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	Vesq
URL	https://vesq.io
Platform	Polygon
Language	Solidity



1.2 Contracts Assessed

Name	Contract	Live Code Match
VSQERC20	0x29F1e986FCa02B7E54138c04C4F503DdDD250558	✓ MATCH
sVSQERC20	0xbbb0FDc2Fe6D3cEB6Bf76Cc955173a07EDbF57494	✓ MATCH
wsVSQ	0x05B33f816d2C0C2D20F0777a75ad549df05bF24D	✓ MATCH
BondDepository	Frax-VSQ 0x8Ab125D6D6ea1743e53Ca79595046f2a0c76A551	✓ MATCH
	Mai-VSQ 0x2d7c40Cd0228264AE5a73F01bC54FA13C4476Da1	
	Dai-VSQ 0xF4acBe9de1Fae931C5A67115184069474d4fAdad	
	Frax Bond 0x5AeC30AFe641EBE8789D8B21223F6D2C74f6fE2C	
	Mai Bond 0x9fD78920cdbE6f365A557cec282cc88152e35670	
	Dai Bond 0x6Fd68930eC828ec5906B0FDEC686F3f459C08d1A	
EthBondDepository	EthBondDepository.sol	✓ MATCH
Staking	0x2F3E9e54bD4513D1B49A6d915F9a83310638CFC2	✓ MATCH
StakingDistributor	0xabE372DCFB8800B3cDE30f1d6666401C765f2F3B	✓ MATCH
StakingHelper	0x493Fdb9ddFd51873a878494B0E4d858D6DEc57E9	✓ MATCH
StakingWarmup	0xE33e7247BdF5FDeB6705E820F3f26823Ea294F13	✓ MATCH
StandardBondingCalculator	0xFEEaDb0798EF580b1394eb38659Cf85cC25D43e4	✓ MATCH
Treasury	0x8c7290399cECbBBf31E471951Cc4C2ce91F5073c	✓ MATCH
VSQZapIn	VSQZapIn.sol (Deployed contracts were not provided)	✓ MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● High	3	3	-	-
● Medium	2	2	-	-
● Low	19	16	1	2
● Informational	32	25	2	5
Total	56	46	3	7

Classification of Issues

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

1.3.1 Global Issues

ID	Severity	Summary	Status
01	HIGH	Gov Privilege: Governance can change crucial aspects of the protocol to potentially drain the contracts of all supplied tokens	RESOLVED
02	MEDIUM	The last owner can be reclaimed across multiple contracts	RESOLVED
03	LOW	New owner variable is private	RESOLVED
04	LOW	Usage of transferFrom/transfer instead of safeTransferFrom/safeTransfer	RESOLVED
05	INFO	permit can be frontrun and cause denial of service	RESOLVED
06	INFO	The contracts become unusable in the year 2106	RESOLVED
07	INFO	The contracts do not work with reflective tokens	RESOLVED

1.3.2 VSQERC20

ID	Severity	Summary	Status
08	LOW	_burnFrom is marked as public	RESOLVED
09	INFO	Transactionless approval permit mechanism still references zero swap	RESOLVED
10	INFO	Vault can be set to the zero address	RESOLVED

1.3.3 sVSQERC20

ID	Severity	Summary	Status
11	LOW	Infrequent rebases incentivize malicious parties to strategically (re)order transactions for arbitrage and steal all rebased tokens off the LP pairs	RESOLVED
12	INFO	DOMAIN_SEPARATOR can be made immutable	RESOLVED
13	INFO	Under a constant and small circulating supply, the non-circulating supply starts increasing more rapidly with every rebase	ACKNOWLEDGED

1.3.4 wsVSQ

ID	Severity	Summary	Status
14	INFO	Gas optimization: Usage of decimals as a precision multiplier	RESOLVED

1.3.5 BondDepository

ID	Severity	Summary	Status
15	HIGH	Vested amount is relocked on deposit, even if the deposit is made by third-parties allowing for targeted Denial of Service	RESOLVED
16	LOW	Adjustment target is never reached	RESOLVED
17	LOW	Deposit is vulnerable to reentrancy if the principle has a reentrancy vector	RESOLVED
18	LOW	The maximum debt can be exceeded by at most maxPayout	RESOLVED
19	INFO	bondPriceInUSD is denominated in the decimals of the other token in the LP and might not be correct for non stablecoin LPs	RESOLVED
20	INFO	initializeBondTerms has no validation	RESOLVED
21	INFO	Contract does not work with a zero vestingTerm	RESOLVED
22	INFO	Contract could theoretically run out of VSQ	RESOLVED

1.3.6 EthBondDepository

ID	Severity	Summary	Status
23	LOW	priceFeed is internal	RESOLVED
24	INFO	Phishing: Frontend could trick users into sending too much MATIC for deposits	RESOLVED
25	INFO	bondPrice calculation is inconsistent with BondDepository	ACKNOWLEDGED
26	INFO	The protocol will likely malfunction if ChainLink feeds for MATIC will be different than 8 decimals	RESOLVED

1.3.7 Staking

ID	Severity	Summary	Status
27	HIGH	Staked amount is relocked on subsequent stakes, even if the stake is made by third-parties allowing for targeted Denial of Service	RESOLVED
28	LOW	Lock logic might be wrongly defined	RESOLVED
29	INFO	Rebases can be arbitrated/frontran	RESOLVED

1.3.8 StakingDistributor

ID	Severity	Summary	Status
30	LOW	Adjustment target is never reached	RESOLVED
31	LOW	Unbounded gas usage due to extensive for-loop usage	RESOLVED
32	LOW	Lack of validation	RESOLVED
33	INFO	Adjustments are not reset when recipient is removed	RESOLVED

1.3.9 StakingHelper

ID	Severity	Summary	Status
34	LOW	Phishing: stake function allows to stake towards a different address	ACKNOWLEDGED

1.3.10 StakingWarmup

No issues found.

1.3.11 StandardBondingCalculator

ID	Severity	Summary	Status
35	MEDIUM	Does not support LP pairs where the second currency has less than 9 decimals	RESOLVED
36	LOW	StandardBondingCalculator can only value pairs in which the two tokens have equal "value"	RESOLVED
37	LOW	Markdown will misbehave if an LP with two non-VSQ tokens is provided	RESOLVED
38	INFO	markdown function is vulnerable to price manipulation	ACKNOWLEDGED

1.3.12 Treasury

ID	Severity	Summary	Status
39	LOW	Lack of component safeguards in a system that plans to increase in number of components over time is considered brittle	ACKNOWLEDGED
40	LOW	Adding a token as both a liquidity and reserve token would cause it to be double counted in the treasury value	RESOLVED
41	LOW	repayDebtWithVSQ has inconsistent privilege requirements which allows for slight privilege escalation	RESOLVED
42	INFO	Unnecessary comparison to true on withdraw function	RESOLVED
43	INFO	Reserve value mechanism could cause withdrawals and other operations to temporarily fail	RESOLVED
44	INFO	Lack of safeTransfer usage within incurDebt	RESOLVED
45	INFO	Manage will always do an excessReserve check even if the token is not within the liquidity or reserves tokens	RESOLVED

1.3.13 VSQZapIn

ID	Severity	Summary	Status
46	LOW	Phishing risk: Users could be mislead into undesirable swaps	PARTIAL
47	INFO	swapData can be denoted as calldata throughout the contract	RESOLVED

1.3.14 Code style-related Issues

ID	Severity	Summary	Status
48	INFO	Inconsistency: Unused mint function emits an event from address(this) while the mint logic during initialization emits an event from the zero address	RESOLVED
49	INFO	Various functions can be made external	RESOLVED
50	INFO	Lack of events for various functions	RESOLVED
51	INFO	Typographical errors	RESOLVED
52	INFO	Unused variables/dependencies throughout the contracts	RESOLVED
53	INFO	Gas optimization: Contract uses hardcoded strings in SafeMath functions	PARTIAL
54	INFO	Uncast addresses make the code more verbose than it needs to be	ACKNOWLEDGED
55	INFO	Ambiguous errors	PARTIAL
56	INFO	Gas optimization: storage variables are frequently unnecessarily reread	ACKNOWLEDGED

2 Findings

2.1 Global Issues

The issues in this section are applicable to the entire protocol.



2.1.1 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

VESQ is a protocol that is responsible for the issuance and management of an algorithmic, free-floating stable asset, VSQ, which is backed by a treasury. As the system has many components which need to be governed, like how the treasury is potentially used, which assets could be used as bonds and the very important parameters of the bond issuance protocol, there is by nature an extreme amount of governance privilege. Essentially, if governance cannot be controlled, both all VSQ and all funds in the treasury can be considered compromised. It is therefore of utmost importance that the team addresses this concern seriously.

Some of the most important governance privileges are that the treasury manager can add new contracts that can mint any amount of VSQ (up to the maximum allowed by the reserve value), the manager can furthermore add contracts that can potentially withdraw all funds stored in the treasury. Finally, within the VSQ token, "vault" ownership could be moved by the VSQ token owner to a new address which can then again mint as many VSQ tokens as they want, in this case without limit. Other potential risk vectors include depositing bad tokens into the treasury which allow privileged contracts to take out valuable assets in return and sVSQ tokens in the Staking contract can be taken out by governance through the lock bonus mechanism.

Due to the anonymous nature of decentralized finance, 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 the team to parties trusted by the community as one will be less inclined to steal funds when their identities are known.

Paladin also strongly recommends transferring the ownership of the contracts to a VESQ Multisig. Furthermore, if VESQ Multisig is comprised of many well known community parties, Paladin can also mark this issue as resolved by going through an pseudonymous identity verification round with all multisig participants that includes validating that the community member in fact owns their respective multisig key.

Resolution



Although this risk is still present, the client has undergone an internal KYC session with Paladin. Generally speaking, even though it does not eliminate the risk, it does reduce it.

Paladin has confirmed that the two parties that underwent KYC own the Gnosis MultiSig at 0x4F64c22FB06ab877Bf63f7064fA21C5c51cc85bf. This MultiSig has also been confirmed to be a valid Gnosis deployment. 2 out of 2 signatures are required for any actions from this address.

Signature 1

Nickname: Syed

Address: 0xe06a1e612c4D5D07784D6e2e528A23Fe2bA75D97

KYC: Yes

Address ownership verification: Yes

Signature 2

Nickname: Zackary

Address: 0x1fA58d1361f0344E367761dDE020Df9417aA3565

KYC: Yes

Address ownership verification: Yes

Note that as the issue is still present, if the clients are hacked, a third party might abuse it.

As the contracts are not yet deployed at the writing of this issue, users will have to validate that the ownership positions of all contracts point to this address. Larger investors who like to be careful furthermore need to be diligent in going over all privileged addresses in the treasury.



Location

sVSQERC20Lines 988-991

```
function renounceManagement() public virtual override
onlyManager {
    emit OwnershipPushed(_owner, address(0));
    _owner = address(0);
}
```

Lines 999-1003

```
function pullManagement() public virtual override {
    require(msg.sender == _newOwner, "Ownable: must be new
owner to pull");
    emit OwnershipPulled(_owner, _newOwner);
    _owner = _newOwner;
}
```

BondDepositoryLines 36-39

```
function renounceManagement() public virtual override
onlyManager {
    emit OwnershipPushed(_owner, address(0));
    _owner = address(0);
}
```

Line 47-51

```
function pullManagement() public virtual override {
    require(msg.sender == _newOwner, "Ownable: must be new
owner to pull");
    emit OwnershipPulled(_owner, _newOwner);
    _owner = _newOwner;
}
```

Staking

Lines 534-537

```
function renounceManagement() public virtual override
onlyManager {
    emit OwnershipPushed(_owner, address(0));
    _owner = address(0);
}
```

Lines 545-549

```
function pullManagement() public virtual override {
    require(msg.sender == _newOwner, "Ownable: must be new
owner to pull");
    emit OwnershipPulled(_owner, _newOwner);
    _owner = _newOwner;
}
```

StakingDistributor

Lines 344-347

```
function renouncePolicy() public virtual override
onlyPolicy() {
    emit OwnershipTransferred( _policy, address(0) );
    _policy = address(0);
}
```

Lines 354-358

```
function pullPolicy() public virtual override {
    require( msg.sender == _newPolicy );
    emit OwnershipTransferred( _policy, _newPolicy );
    _policy = _newPolicy;
}
```

Treasury

Lines 163-166

```
function renounceManagement() public virtual override
onlyManager {
    emit OwnershipPushed(_owner, address(0));
    _owner = address(0);
}
```

Lines 168-172

```
function pullManagement() public virtual override {
    require(msg.sender == _newOwner, "Ownable: must be new
owner to pull");
    emit OwnershipPulled(_owner, _newOwner);
    _owner = _newOwner;
}
```

Description

Within the ownership implementation of the different contracts, ownership can be renounced, however, the last owner can reclaim this at any moment as the new owner variable was never reset.

It should furthermore be noted that before the first ownership transfer is made, the zero address can claim ownership over the contract. This is hardly problematic as the zero contract is not known to be owned by anyone and probabilistically speaking, under the current address scheme, the chances of anyone ever owning it are negligible.

Note: On StakingDistributor, the Ownership implementation is implemented as Policy.

Recommendation



Consider using BoringOwnable instead.



Resolution



BoringOwnable is now used throughout the codebase.

<https://github.com/boringcrypto/BoringSolidity/blob/f05de5f250056730c3fd3e5a5d1e572c2d113023/contracts/BoringOwnable.sol>

Issue #03	New owner variable is private
Severity	 LOW SEVERITY
Location	<p>sVSQERC20: _newOwner variable</p> <p>BondDepository: _newOwner variable</p> <p>Staking: _newOwner variable</p> <p>StakingDistributor: _newPolicy variable</p> <p>Treasury: _newOwner variable</p>
Description	Throughout the contracts that implement the Ownership pattern, the variable that denotes the new owner is private. 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 BoringOwnable is now used throughout the codebase, making this issue obsolete.

Issue #04	Usage of transferFrom/transfer instead of safeTransferFrom/safeTransfer
Severity	 LOW SEVERITY
Description	<p>Throughout the contracts, the transfer tokens functionality does not work with tokens that return false.</p> <p><u>BondDepository</u></p> <p>VSQ tokens which return false are not supported; tokens which do not return a boolean are not supported. Consider using safeTransferFrom. It should be noted that safeTransferFrom is consistently used throughout the rest of the protocol except in this contract and a few others making this an inconsistency issue as well.</p> <p><u>EthBondDepository</u></p> <p>weth tokens which return false are not supported; tokens which do not return a boolean are not supported. Consider using safeTransferFrom. It should be noted that safeTransferFrom is consistently used throughout the rest of the protocol except in this contract and a few others making this an inconsistency issue as well.</p> <p><u>StakingHelper</u></p> <p>VSQ tokens which return false are not supported; tokens which do not return a boolean are not supported. Consider using safeTransferFrom. It should be noted that safeTransferFrom is consistently used throughout the rest of the protocol except in this contract and a few others making this an inconsistency issue as well.</p> <p><u>StakingWarmup</u></p> <p>sVSQERC20 tokens which return false are not supported, tokens which do not return a boolean are not supported. Consider using safeTransfer. It should be noted that safeTransfer is consistently used throughout the rest of the protocol except in this contract and a few others making this an inconsistency issue as well.</p>
Recommendation	Consider using safeTransferFrom or safeTransfer.
Resolution	 RESOLVED

Issue #05**permit can be frontrun and cause denial of service****Severity** INFORMATIONAL**Description**

Many of the tokens contain a transactionless approval scheme based on [EIP-2612](#). This mechanism is most well-known by users when they break up Uniswap LP tokens without having to explicitly send an approval transaction, instead they just have to make a signature.



Just like with Uniswap permits, if `permit` is executed twice, the second execution will be reverted. It is thus in theory possible for a bot to pick up `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. This could allow for denial of service.

Recommendation



Within derivative protocols, one can consider using try-catch for `permit` and validating the approval afterwards.

Resolution RESOLVED

The client has indicated that they will use try-catch logic whenever attempting such interactions. We remind the client to be careful of gas-griefing where the try section fails because gas runs out but the rest of the code can still complete on the remainder of gas

Issue #06	The contracts become unusable in the year 2106
Severity	 INFORMATIONAL
Location	Staking::Line 710 (example) epoch.endTime = epoch.endTime.add32(epoch.length);
Description	<p>Throughout the contract, timestamps are supposed to fit in 32 bit integers. However, once the year 2106 is reached, this is no longer possible and most functionality will fail and revert.</p> <p>For example, once the timestamp of the next epoch does not fit into an unsigned 32 bit integer, the rebase function will revert. Furthermore, as soon as this timestamp is reached, even if the epoch hasn't caught up yet, rebase() will no longer be callable on previous epochs since uint32(block.timestamp) overflows to a low number.</p>
Recommendation	Consider whether the contract will survive for this long, if so, consider using a larger integer for the timestamp or adding an overflow mechanism (Uniswap for example designed their timestamps to still work with overflows).
Resolution	 RESOLVED All small data types have been moved to uint256.



Issue #07	The contracts do not work with reflective tokens
Severity	 INFORMATIONAL
Description	The whole VESQ system is completely incompatible with any transfer-tax tokens. Whether as principle tokens, or forked versions of VSQ or sVSQ, transfer taxes are not supported.
Recommendation	Consider avoiding any tokens with transfer taxes, rebase mechanisms or other special logic going on. These can be wrapped in a simple wrapped equivalent that has no auxiliary transfer logic going on.
Resolution	 RESOLVED The client has confirmed reflective tokens are not going to be used at all.



2.2 VSQERC20

The VSQ token is the main token within the VESQ ecosystem. It is a simple ERC-20 token which is extended with [EIP-2612](#) permit capabilities. Users may know of such permit capabilities from when they break up Uniswap LP tokens. In this instance, instead of explicitly needing to transmit an approve transaction, they can simply sign it without any gas cost or transaction. The token can be minted freely by the vault address, which is settable at any time by the contract owner.

2.2.1 Token Overview



Address	TBC
Token Supply	Unlimited
Decimal Places	9
Transfer Max Size	No maximum
Transfer Min Size	No minimum
Transfer Fees	None
Pre-mints	None



2.2.2 Privileged Roles



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

- `mint`
- `setVault`
- `transferOwnership`
- `claimOwnership`

2.2.3 Issues & Recommendations

Issue #08 _burnFrom is marked as public	
Severity	 LOW SEVERITY
Description	<p>The contract contains a function, burnFrom, to allow burning VSQ from another account given that this account has given you approval. As is common with ERC-20 implementations, the public burnFrom function calls the internal function _burnFrom. Within this implementation however, _burnFrom is marked as public by accident.</p> <p>This does not have side effects and therefore does not affect investors in any way. However, it might signal to third-party reviewers that the code was not carefully reviewed before deployment.</p>
Recommendation	Consider marking the _burnFrom function as internal.
Resolution	 RESOLVED

Issue #09 Transactionless approval permit mechanism still references zero swap	
Severity	 INFORMATIONAL
Location	Line 803 <pre>require(signer != address(0) && signer == owner, "ZeroSwapPermit: Invalid signature");</pre>
Description	<p>For single transaction approvals, EIP-2612-based permits are used. These are known by users when they break up Uniswap LPs and only need to sign the approval instead of actually creating a transaction.</p> <p>However, within the implementation used by VESQ, an error message still references ZeroSwap. This might signal that the codebase wasn't reviewed before deployment to third-party reviewers and is therefore best adjusted.</p>
Recommendation	Consider adjusting the error message.
Resolution	 RESOLVED

Issue #10 Vault can be set to the zero address	
Severity	 INFORMATIONAL
Description	The setVault function allows for the vault, which is the only account that can mint VSQ, to be set to zero. It is common practice to add a non-zero function to such set functions to prevent potential errors going undetected.
Recommendation	Consider adding non-zero checks to setVault.
Resolution	 RESOLVED

2.3 sVSQERC20

The sVSQERC20 (sVSQ) token is a rebasing token which increases the sVSQ supply and therefore the user balances whenever the staking contract calls rebase on it. It is kept somewhat backed by FRAX. It has a different approach than a normal stablecoin that is usually pegged to a certain asset.

sVSQ follows a similar implementation to Ampleforth and is likely inspired by this protocol.

2.3.1 Token Overview



Address	TBC
Token Supply	Unlimited
Decimal Places	9
Transfer Max Size	No maximum
Transfer Min Size	No minimum
Transfer Fees	None
Pre-mints	5,000,000 [to Staking contract]

2.3.2 Privileged Roles

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

- `Initialize` [callable once]
- `setIndex` [callable once]
- `rebase`
- `transferOwnership`
- `claimOwnership`

2.3.3 Issues & Recommendations

Issue #11	Infrequent rebases incentivize malicious parties to strategically (re)order transactions for arbitrage and steal all rebased tokens off the LP pairs
Severity	 LOW SEVERITY
Description	<p>The contract periodically increases the user balances as part of the rebases. If these rebases were to occur sufficiently infrequently, say every week, they might be an incentive for either miners or advanced users to strategically order their transactions in a way that they temporarily hold a balance right before the rebase to receive rewards on it.</p> <p>Furthermore, even if rebases were to be made frequently, the balance of the LP pairs will increase each time a rebase occurs. If <code>skim()</code> is called on the LP pairs right after this occurs, the skimmer will receive all tokens of that rebase. As there are many bots that do this as soon as such an opportunity arises, going as far as using the mempool to be sufficiently fast, it is almost certain that all rebases on the LP pair tokens have been skimmed and dumped.</p>
Recommendation	<p>Consider frequently rebasing and ensuring that no unprivileged user can rebase from a contract which would allow them to flashloan sVSQ temporarily. Furthermore, consider manually calling <code>sync()</code> or <code>skim()</code> on the LP pairs through a contract that calls the rebase. This way the tokens can either be incorporated in the reserves or taken out of the pairs again to prevent unnecessary selling pressure. It is important that this last step is done within a single transaction by a contract as to not have someone frontrun the governance attempt to take the tokens out again.</p>
Resolution	 RESOLVED
	<p>The client has indicated that rebases will occur frequently. This however does not prevent the skim issue which has been indicated to not be a problem as no liquidity will be added to such pairs because of this issue.</p>

Issue #12**DOMAIN_SEPARATOR 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.

It should be noted that within the EIP-712 draft implementation by OpenZeppelin, the `DOMAIN_SEPARATOR` is somewhat mutable as to allow the `chainId` to evolve. This logic has however not been implemented within this implementation of EIP-712.

It should finally be noted that the `Permit` typehash uses `value` as a parameter, while the function uses `amount`. It could be better to use `value` as the `permit` parameter as well.

Recommendation

Consider making this variable explicitly `immutable`.

Resolution RESOLVED

Issue #13

Under a constant and small circulating supply, the non-circulating supply starts increasing more rapidly with every rebase

Severity

 INFORMATIONAL

Location

```
rebaseAmount =  
profit_.mul( _totalSupply ).div( circulatingSupply_ );
```

Description

The rebase amount is based upon the profit to rebase multiplied by the total supply divided by the circulating supply. This is because the sVSQ contract is unable to discriminate against sVSQ within the staking contract during rebases. Contrary to implementations like SafeMoon, there is no way to exclude accounts from rebases. If no adjustment would be made, a portion of the profit would be lost to the sVSQ that is sitting in the Staking contract. To account for this, the rebase amount is increased to ensure that the circulating supply exactly gets that profit.


If then for some reason the `circulatingSupply_` is kept very low, let's say at a nominal 1, the `_totalSupply` increases more rapidly with every rebase. If profit is also 1, and `_totalSupply` is 10, `_totalSupply` would increase to about 20, during the next rebase of 1 profit, `_totalSupply` would increase to 40. In this situation the `MAX_SUPPLY` could be reached rather quickly.

This could be a potential denial of service attack during the bootstrapping of an Ohm protocol fork, while there are no stakers yet.

Recommendation

Consider this situation carefully. Consider the rate of (exponential) growth of `_totalSupply` under the current setup. This issue will be resolved on the notice that the client has inspected this rate of growth and that `MAX_SUPPLY` is not to be reached in an extremely long time, even if a majority of the stakers decides to unstake.

Resolution

 ACKNOWLEDGED

This issue can be marked as resolved once the vesq deployment has stabilized and Paladin recognizes it is unlikely that this cycle is entered into.



2.4 wsVSQ

The wsVSQ token wraps the sVSQERC20 token in a fixed balance alternative. Instead of increasing in balance with every rebase, wsVSQ can be liquidated for more and more sVSQ over time.

2.4.1 Token Overview

Address	TBC
Token Supply	Unlimited
Decimal Places	9
Transfer Max Size	No maximum
Transfer Min Size	No minimum
Transfer Fees	None
Pre-mints	None

2.4.2 Issues & Recommendations

Issue #14 Gas optimization: Usage of decimals as a precision multiplier	
Severity	 INFORMATIONAL
Description	The contract uses the wsVSQ decimals as a precision multiplier, however, as this variable is not immutable this creates a small ~200 gas overhead every time it is read from memory.
Recommendation	Consider either using a constant precision multiplier, caching the decimals in an immutable variable or making decimals immutable.
Resolution	 RESOLVED Decimals have been made immutable.

2.5 BondDepository

The Bond Depository is one of the main contracts within VESQ. It allows users to sell their LP tokens for VSQ futures which vest linearly over the next period. Periodically, the rate at which VSQ is given for LP tokens adjusts upwards or downwards which can be freely configurable by the governance. No more bonds can be issued than a certain maximum. The contribution to this maximum decays over time allowing for more bonds to be issued. Vested VSQ can be instantly staked if desired by the user. The DAO receives a percentage of the minted VSQ according to the `terms.fee` parameter.

2.5.1 Privileged Roles

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

- `initializeBondTerms`
- `setBondTerms`
- `setAdjustment`
- `setStaking`
- `transferOwnership`
- `claimOwnership`

2.5.2 Issues & Recommendations

Issue #15	Vested amount is relocked on deposit, even if the deposit is made by third-parties allowing for targeted Denial of Service
------------------	---

Severity

 HIGH SEVERITY

Description

Currently the `deposit` function, which is used to deposit LP tokens into a bond, will reset the vesting term of any previous deposits. The vested duration since the last redemption would therefore be lost if the user deposits again. This can be used by malicious parties to create griefing for different wallets. The griefing goes as follow:

1. Listen to the `BondRedeemed` method in the mempool. This way you know when a user is about to claim their vested portion.
2. As soon as you detect it, you send a deposit to them with a tiny amount. This resets their timer.
3. They now need to wait a whole bond duration again.

Repeat this whenever you detect `BondRedeemed` in the mempool and you have effectively locked in all VSQ.

Recommendation

Consider either removing the functionality to `deposit` to another account or making this a whitelisted operation. The same could be considered for the `redeem` function to reduce the attack vector.

We also recommend removing this functionality from the `redeem` method.

Resolution

 RESOLVED

A whitelist has been added to both deposits and redemption to other accounts. This means only whitelisted contracts and wallet can still undertake this action, preventing any unprivileged party from abusing this.

Location

Lines 976-994

```
function adjust() internal {
    uint timeCanAdjust =
adjustment.lastTime.add( adjustment.buffer );
    if( adjustment.rate != 0 && block.timestamp >=
timeCanAdjust ) {
        uint initial = terms.controlVariable;
        if ( adjustment.add ) {
            terms.controlVariable =
terms.controlVariable.add( adjustment.rate );
            if ( terms.controlVariable >=
adjustment.target ) {
                adjustment.rate = 0;
            }
        } else {
            terms.controlVariable =
terms.controlVariable.sub( adjustment.rate );
            if ( terms.controlVariable <=
adjustment.target ) {
                adjustment.rate = 0;
            }
        }
        adjustment.lastTime = uint32(block.timestamp);
        emit ControlVariableAdjustment( initial,
terms.controlVariable, adjustment.rate, adjustment.add );
    }
}
```

Description

The code contains an adjust function which allows adjusting the control variable with a fixed increment or decrement after a fixed period. It furthermore contains a target after which the adjustment stops once it is reached.

However, due to the code implementation, the target might be slightly missed, as the adjustment will only stop after it is passed due to the increments being rather large.

Furthermore, if the target would be set close to zero, the subtraction might cause this to revert.

Recommendation Consider setting the info rate to the target once the target has been reached. Consider also resetting the target as to have a cleaner state.



It should be noted that this adjustment method is also slightly wasteful in gas as it often re-reads `terms.controlVariable` from storage. If gas-usage is a concern, consider caching some of these variables.



Resolution






The client has implemented the recommended behavior and added logic to prevent the subtraction underflow.





Issue #17	Deposit is vulnerable to reentrancy if the principle has a reentrancy vector
Severity	 LOW SEVERITY
Description	<p>The deposit function currently does adjustments of the <code>totalDebt</code> and value calculations after the principle has been transferred. This allows an external party to inject code to avoid the <code>maxDebt</code> calculation and furthermore manipulate (the current calculator only allows expensive increment-only manipulation by sending tokens to the pair and calling <code>sync</code>) the value in deposit compared to the local value if a token which allows reentrancy is added.</p> <p>With such a token, <code>maxDebt</code> could be completely circumvented in the current design.</p> <p>This issue is marked as low severity as we expect principle tokens to be LP pairs mostly, however, we did notice that these can be single-asset on existing Ohm forks as well.</p>
Recommendation	Consider reorganizing the deposit function to adhere to checks-effects-interactions.
Resolution	 RESOLVED <p>The deposit function now has a reentrancy guard: The client should however remember that this only prevents reentrancy in this specific function and not in other parts of the system like the LP pair.</p>

Issue #18	The maximum debt can be exceeded by at most maxPayout
Severity	 LOW SEVERITY
Location	<u>Line 866</u> <pre>require(totalDebt <= terms.maxDebt, "Max capacity reached");</pre>
Description	Currently the check that the maximum amount of debt is not exceeded does not include the newly created debt, this allows for the maximum debt to be exceeded by at most maxDebt.
Recommendation	Consider including value, which is the new debt, in this requirement. <pre>require(totalDebt.add(value) <= terms.maxDebt, "Max capacity reached");</pre>
Resolution	 RESOLVED The client has updated the codebase to properly validate the maxDebt: <pre>uint value = ITreasury(treasury).valueOf(principle, _amount); require(totalDebt.add(value) <= terms.maxDebt, "Max capacity reached");</pre>

Issue #19	bondPriceInUSD is denominated in the decimals of the other token in the LP and might not be correct for non stablecoin LPs
Severity	 INFORMATIONAL
Description	<p>bondPriceInUSD is denominated in the decimals of the other token in the LP and might not be correct for non stablecoin LPs. As this function is primarily used on the frontend this issue has been marked as informational.</p> <p> standardizedDebtRatio has similar behavior.</p>
Recommendation	Consider handling this correctly on the frontend.
Resolution	 RESOLVED
	The client has indicated this is handled correctly.



Issue #20	initializeBondTerms has no validation
Severity	 INFORMATIONAL
Description	<p>The <code>initializeBondTerms</code> function has no validation of the parameters that are used for initialization of the bond terms when the contract is first deployed.</p> <p>! In addition, <code>controlVariable</code> can go to zero with the adjustments, making the <code>initializeBondTerms</code> available to be called again. We are unsure why there should be an <code>initialDebt</code> on initialization function.</p> <p>! The <code>setAdjustment</code> function on the other hand becomes completely locked out if <code>controlVariable</code> ever reaches zero, which is strange behavior to have defined so implicitly.</p>
Recommendation	Consider adding proper validation for this function and remove the <code>initialDebt</code> parameter if there is no need for an initial debt.
Resolution	 RESOLVED
	Validation has been added to many of the parameters. Re-initialization is prevented with a check on the <code>lastDecay</code> parameter. Note that <code>maxPayout</code> still lacks validation.

Issue #21**Contract does not work with a zero vestingTerm****Severity** INFORMATIONAL**Description**

The `debtDecay` function reverts due to a division by zero if `terms.vestingTerm` is set to zero. Furthermore, the `percentVestedFor` function will always return a zero vested percentage if the remaining vesting duration is zero (eg. with a zero `vestingTerm`). This should more accurately return 10,000 (100%) as at this point the bonds instantly vests. The contract would therefore become unusable if the `vestingTerm` is zero.

Recommendation

Consider making the requirement of a non-zero vesting term explicit when the term is set.

Resolution RESOLVED

Within `initializeBondTerms` and `setBondTerms`, the non-zero `vestingTerm` requirement has been made explicit.



Severity

 INFORMATIONAL

Description

There is currently no guarantee that the number of VSQ that the depository receives from the treasury is sufficient to cover the payouts. This is because a profit is withheld by the treasury and a fee is sent to the DAO.

Recommendation

Consider making the requirements within the parameters more explicit as to prevent the situation where more VSQ can be allocated to payouts than is maintained in the depository. A crude check is to simply reduce the payout to at most the amount received during the deposit function.

Resolution

 RESOLVED

The client has explained how an underflow within the bond depository significantly reduces the chances of this situation occurring. This is because within the bond, it calculates the amount of profit the treasury can keep for itself and this profit would become negative if the treasury wouldn't give enough tokens for the payout.

The only situation where this issue can remain relevant is if the value calculator within the treasury returns a different value within the bond and treasury.

Under the current bond calculator this is not possible without reentrancy vectors between the two calls in a single atomic transaction.

This issue has therefore been marked as resolved since VESQ expects to not use any other calculator.

2.6 EthBondDepository

The EthBondDepository is similar to the BondDepository but differs in that it allows for MATIC and WMATIC to be deposited. It furthermore uses ChainLink to calculate the UI price.

As the EthBondDepository contract is extremely similar to BondDepository, any recurring issues have been omitted from this section of the audit. Users can assume that most if not all of the issues within BondDepository are also present within EthBondDepository.



2.6.1 Privileged Roles



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

- `initializeBondTerms`
- `setBondTerms`
- `setAdjustment`
- `setStaking`
- `transferOwnership`
- `claimOwnership`



2.6.2 Issues & Recommendations

Issue #23	priceFeed is internal
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 this variable as public.
Resolution	 RESOLVED

Issue #24	Phishing: Frontend could trick users into sending too much MATIC for deposits
Severity	 INFORMATIONAL
Description	<p>Currently the deposit requires that at least sufficient MATIC is sent to cover the provided amount the user wants; however, there is no check that it must be equal. If the frontend is ever compromised, this could be abused to send excessive MATIC to the contract, without the users earning VSQ futures in return.</p> <p>The comment on refundETH furthermore indicates that the excess MATIC goes to the user, while it ironically goes to the DAO.</p>
Recommendation	Consider fixing these errors
Resolution	 RESOLVED A proper refund mechanism has been introduced.

Severity

 INFORMATIONAL

Description	BondDepository	EthBondDepository
	<pre>price_ = terms.controlVariable.mul(d ebtRatio()).add(10000000000).div(1e7);</pre>	<pre>price_ = terms.controlVariable.mul(d ebtRatio()).div(1e5);</pre>

The two bondPrice calculations differ in the fact that the BondDepository still adds 1 to the price while the EThBondDepository does not.



Note that the difference in division is not a problem since both contracts correctly account for the different precision rate.

Recommendation Consider explaining this inconsistency and if it is not necessary, consider making both contracts more consistent.

Resolution

 ACKNOWLEDGED

The client has indicated that they are okay with this small inconsistency, as the end result is the same.

Issue #26	The protocol will likely malfunction if ChainLink feeds for MATIC will be different than 8 decimals
Severity	 INFORMATIONAL
Description	Throughout the contract, the assumption is made that ChainLink feed returns a MATIC price with 8 decimals. If Chainlink were to change this in the future, it will cause multiple issues.
Recommendation	<p>Consider upgrading the protocol to dynamically fetch the MATIC token's <code>.decimals()</code>.</p> <p>It should be noted that this update needs to be made in multiple locations.</p>
Resolution	 RESOLVED <p>The client has indicated that they will exclusively use feeds with 8 decimals. They've also made this requirement explicit in the constructor of the contract.</p>



2.7 Staking

Staking is a contract that lets investors stake their VSQ into an identical amount of sVSQ. sVSQ, which is essentially staked VSQ, increases in quantity over time through rebases. When funds are deposited, they are locked into the StakingWarmup for a number of epochs, after this period, the sVSQ (including potentially rebased amounts) becomes claimable using the claim contract. If VESQ chooses a zero-length locking period, they can a StakingHelper contract that stakes and immediately calls claim within a single transaction. This avoids users from having to make these two transactions themselves. If the lock were to be present, users could call `forfeit()` to retrieve their initial VSQ and forgo any increase in the locked sVSQ amount.

Users can call the `unstake` function to trade in an identical amount of sVSQ for VSQ. It should be noted that sufficient VSQ needs to be present in the Staking contract, but this is governed by the other components of the systems.

It should be noted that after the zero length warmup period has expired, anyone can call `claim` for you to move the now unlocked sVSQ to your wallet. Users should be mindful of this behavior in case they interact with any protocols that blindly take their whole sVSQ balance.



Finally, the contract can send a portion of its sVSQ balance to the locker contract. The locker contract is out of scope but the basics of this functionality is that the staking contract will see this as an outstanding loan (an asset) and a debt (new sVSQ in circulation), therefore there is no direct effect on the rebasing. However, the sVSQ which is rebased onto the outstanding debt cannot be withdrawn into the staking contract anymore, we therefore assume that this rebased sVSQ would be used for other means.



2.7.1 Privileged Roles



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

- `setContract`
- `transferOwnership`
- `claimOwnership`

2.7.2 Issues & Recommendations

Issue #27	Staked amount is relocked on subsequent stakes, even if the stake is made by third-parties allowing for targeted Denial of Service
Severity	 HIGH SEVERITY
Description	<p>Currently, the stake method which is used to stake VSQ to receive sVSQ has a capability to use a WarmUp strategy on staking. Everytime someone stakes, their stake is locked by a duration called the warmupPeriod. After this period is finished, the staker can claim their rewards.</p> <p>However, since users can stake for others, this can be used to create griefing for different wallets. The griefing goes as follow:</p> <ol style="list-style-type: none">1. Listen to the unstake() method being called in the mempool. This way you know when a user is about to claim their staking rewards.2. As soon as you detect it, call stake() with a small amount. This resets their warmup timer.3. They now need to wait a whole new warmup period again. <p>Repeat this whenever you detect a stake call in the mempool and you've effectively locked in all the rewards in the staking contract.</p>
Recommendation	Consider removing the functionality to stake to another account or making this a whitelisted operation or never use a warmup approach on staking by keeping the warmupPeriod to 0.
Resolution	 RESOLVED
	<p>The client has forced the warmupPeriod to always be zero by making it constant. The warmup functionality is therefore disabled for all practical reasons.</p>

Issue #28	Lock logic might be wrongly defined
Severity	 LOW SEVERITY
Description	<p>The contract contains two functions: giveLockBonus and returnLockBonus which allow to essentially loan out some of the sVSQ tokens to the lock contract. However, when the loan is repaid, the lock retains all rebased rewards. This is not necessarily wrong but as no lock contract was included within the scope of this audit, we are unsure about whether this is desired.</p> <p>From a technical perspective, the current implementation does make sense as if the "interest" would be sent back, returnLockBonus would add credit to the staking contract balance which might have unintended side-effects.</p>
Recommendation	Consider whether this is desired behavior.
Resolution	 RESOLVED The lock logic has been removed.

Issue #29	Rebases can be arbitrated/frontran
Severity	 INFORMATIONAL
Description	<p>All through the contract protects against flash loaning VSQ to use it to capture rebases, an advanced party could still purchase VSQ the block before a rebase occurs to then sell it afterwards.</p> <p>If this party has some control over their timing or some control to prevent other users from arbitrating their purchase, this could be profitable and result in less sVSQ for the other stakers.</p>
Recommendation	Consider rebasing very frequently or using a staking method where VSQ staked is directly incorporated. Locking stakes as is done in Ohm is also possible.
Resolution	 RESOLVED The client has indicated they will rebase sufficiently often to make such arbitrages unprofitable.

2.8 StakingDistributor

The distributor is a contract that mints VSQ to the governance configured recipients every time an epoch ends. The amount of VSQ to mint is a percentage set by the governance of the total VSQ supply. The distributor therefore has the ability to trigger a minting from the Treasury to all the recipients added by the governance at every epoch. Therefore, every time a rebase is done at the end of an epoch, the VSQ total supply increases. After each distribution the rate of the distribution is adjusted based on an adjustment variable that is set by the governance.


Users should carefully keep an eye on this contract as it has the power of distributing the whole VSQ supply (and more!) to recipients at every epoch.

2.8.1 Privileged Roles

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

- `addRecipient [! high risk]`
- `removeRecipient`
- `setAdjustment [! high risk]`
- `transferOwnership`
- `claimOwnership`

2.8.2 Issues & Recommendations

Issue #30	Adjustment target is never reached
Severity	 LOW SEVERITY
Location	<pre>function adjust(uint _index) internal { Adjust memory adjustment = adjustments[_index]; if (adjustment.rate != 0) { if (adjustment.add) { // if rate should increase info[_index].rate = info[_index].rate.add(adjustment.rate); // raise rate if (info[_index].rate >= adjustment.target) { // if target met adjustments[_index].rate = 0; // turn off adjustment } } else { // if rate should decrease info[_index].rate = info[_index].rate.sub(adjustment.rate); // lower rate if (info[_index].rate <= adjustment.target) { // if target met adjustments[_index].rate = 0; // turn off adjustment } } } }</pre>
Description	<p>The code contains an adjust function which allows adjusting the emission rate towards a recipient with a fixed increment or decrement after every distribution. It furthermore contains a target after which the adjustment stops once it is reached.</p> <p>However, due to the code implementation, the target might be slightly missed, as the adjustment will only stop after it is passed due to the increments being rather large.</p> <p>Furthermore, if the target would be set close to zero, the subtraction might cause this to revert.</p>

Recommendation Consider setting the info rate to the target once the target has been reached. Consider furthermore resetting the target as to have a cleaner state.

```
info[ _index ].rate = adjustment.target;  
delete adjustments[ _index ];
```

It should be noted that this adjustment method is also slightly wasteful in gas as it often re-reads `info[index].rate` from storage. If gas-usage is a concern, consider caching some of these variables.

Resolution

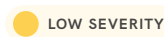


The recommended code section has been implemented together with subtraction overflow protection.

Issue #31

Unbounded gas usage due to extensive for-loop usage

Severity



Description

Many for loops are used to iterate over the recipients. If there are many recipients, this causes high gas cost and could increase in gas cost to the point where the distribute function would become uncallable. Since `removeRecipient` does not reduce the loop size, there could be a point in time where a new Distributor would have to be deployed as gas cost has risen so much.



Recommendation



Consider enforcing a limit of recipients within `addRecipient`, as a reminder that this is not unbounded. Consider also reusing indices if recipients are removed by using the traditional array index deletion pattern where the last index is moved into the deleted index, and the array is shortened by one. This pattern requires a re-linking of the adjustments mapping to the new index.

Resolution



Up to 5 recipients can be added.

Issue #32	Lack of validation
Severity	 LOW SEVERITY
Description	Currently there is no upper limit to the rate which a recipient can receive newly minted VSQ at. If the governance of this contract was ever compromised, or user error occurred, the whole supply and more could be minted to a recipient by accident.
Recommendation	Consider requiring the rates and target to be within reasonable limits, eg. 5% of the total supply at most. Consider limiting the number of recipients to a reasonable number, eg. 5 recipients at most.
Resolution	 RESOLVED A maximum rate of 5e4 has been installed (5%).



Issue #33	Adjustments are not reset when recipient is removed
Severity	 INFORMATIONAL
Description	<p>Within the removeRecipient function, the relevant adjustment struct is not deleted. Deleting this struct might be considered cleaner and could furthermore reduce the gas cost of removeRecipient.</p> <p>If the code is ever updated to reuse the empty index on deletion (array index deletion pattern), deleting the adjustment would also be a defensive move if the new codebase forgets to also move the adjustment into the empty index.</p>
Recommendation	Consider resetting the adjustment on removeRecipient.
Resolution	 RESOLVED Adjustments are now reset.

2.9 StakingHelper

The StakingHelper is a simple utility contract that has just 1 method that does a stake and a claim within one transaction for the staking contract. As VESQ does not use the locking functionality introduced in Ohm, they want to avoid the requirement that people have to call "claim" themselves manually after each staking.



2.9.1 Issues & Recommendations

Issue #34	Phishing: stake function allows to stake towards a different address
Severity	 LOW SEVERITY
Description	<p>The stake function withdraws VSQ tokens from the transaction sender; however, these tokens are granted to the recipient which is a parameter of the stake function. If the frontend is ever compromised, it could be expected that the hacker might simply set the recipient parameter to their wallet to steal all newly staked sVSQ.</p> <p>Most advanced users have become adept enough to check the contract which they are interacting with, but not yet the parameters, as these are displayed in bytecode by Metamask.</p> <p>This issue applies to the Staking contract as well.</p>
Recommendation	<p>Consider only allowing staking to one's own account. Consider also making this fix in Staking if it is a user-facing contract.</p> <p>! If this recommendation is not implemented, we recommend non-zero validation in the staking contract to provide an explicit error message if the recipient is set to the zero address to prevent user errors. It should be noted that due to the overrides within the sVSQ token, these tokens no longer revert on transfers to the zero address, VSQ tokens still do.</p>
Resolution	<p> ACKNOWLEDGED</p> <p>This is still possible, however, a non-zero check was added to prevent user error.</p>

2.10 StakingWarmup

The StakingWarmup contract is a very simple helper contract that is used to store the staked VSQ balances of users. It is used exclusively by the Staking contract though everyone can of course transfer both sVSQ and VSQ to it manually.

2.10.1 Privileged Roles

The following functions can be called by the Staking contract:

- `retrieve`

2.10.2 Issues & Recommendations

No issues found.






2.11 StandardBondingCalculator

The StakingBondingCalculator was designed by Ohm as an LP valuing contract that would use $1 \text{ OHM} = 1 \text{ DAI}$ as the values of the individual components of the LP pair. It uses the correct approach of valuing LP pairs by rebalancing the pair as to have equally valued reserves. It was however only designed to work for OHM+stable pairs and assumes that OHM is worth \$1 to derive the value of the pair.



2.11.1 Issues & Recommendations

Issue #35	Does not support LP pairs where the second currency has less than 9 decimals
Severity	 MEDIUM SEVERITY
Location	<u>Line 278</u> <pre>uint decimals = token0.add(token1).sub(IERC20(_pair).decimals());</pre>
Description	<p>The StandardBondingCalculator does a decimal adjustment to make sure that whatever the decimals of the two LP tokens, the resulting number of decimals is 18. This calculation is:</p> $\text{decimals(token0)} + \text{decimals(token1)} - \text{decimals(pair)}$ <p>As VSQ has 9 decimals and the pair 18 decimals, the paired token must have at least 9 decimals or this calculation will revert due to underflow. This is notoriously not the case for most stablecoins on avalanche making this contract unusable for these.</p>
Recommendation	Consider adjusting the logic to start multiplying instead of dividing if the decimals would be negative. The client could consider an if-else branch for if the pair decimals are smaller than the sum of the token decimals and invert the logic for the new branch.
Resolution	 RESOLVED <p>The client has added logic to adjust the decimals in either direction supporting the previously unsupported cases.</p>

Issue #36**StandardBondingCalculator can only value pairs in which the two tokens have equal "value"****Severity** LOW SEVERITY**Description**



The StakingBondingCalculator was designed by Ohm as an LP valuing oracle that would use 1 OHM = 1 DAI as the oracle value to value the number of OHMs (or DAI) the LP is worth. It was only designed to work for OHM+stable pairs. The bonding calculator is therefore insufficiently equipped for tokens with unequal 'value' (within parentheses as the value of VSQ is not equal to \$1, however the system uses this to calculate the value).



Recommendation

Consider this carefully and consider using different oracles if other LP pairs need to be priced, or if pricing needs to occur at the current VSQ value. The client should remember that pricing LPs is notoriously difficult and that an approach involving K and oracle prices would still be required. Furthermore the client should remember that within the Treasury system, the LPs are not valued at their present value, instead they are valued at their eventual \$1 value.

Resolution RESOLVED

The client has indicated they will not be using non-stablecoin pairs.

Issue #37 Markdown will misbehave if an LP with two non-VSQ tokens is provided	
Severity	 LOW SEVERITY
Location	<u>Lines 299-303</u> <pre> if (IUniswapV2Pair(_pair).token0() == Time) { reserve = reserve1; } else { reserve = reserve0; } </pre>
Description	Currently the markdown function assumes it is being provided a VSQ/OTHER LP, however, if it were to be provided an OTHER/OTHER LP, it would wrongly assume that the second token is in fact VSQ.
Recommendation	Consider this carefully in all locations where markdown is used, or consider making this function explicitly revert in this circumstance.
Resolution	 RESOLVED A requirement for one of the tokens to be VSQ has been included.

Issue #38 markdown function is vulnerable to price manipulation	
Severity	 INFORMATIONAL
Description	The markdown function can be manipulated at a relatively low cost by wrapping the call in a buy and sell (or vice-versa) to adjust the reserves. This leads to the markdown function, which is used to calculate the relative value of the pair (compared to the long-term value), being unuseable for any oracle functionality as it can be manipulated.
Recommendation	The client should take not to never use this function as an oracle or a trusted source.
Resolution	 ACKNOWLEDGED

2.12 Treasury

The treasury is one of the central components within VESQ. It keeps all the underlying assets that are deposited through the bonds and keeps track of debt if any other components borrow these treasuries. It is using a queue approach to change the governance addresses for different actions inside the treasury which is very similar to a timelock. It also gives the possibility for certain addresses to borrow from the Treasury (against sVSQ) and repay the borrowed amount. It furthermore allows the possibility to repay the debt with VSQ. Finally and most importantly, it allows any reward manager (eg. the staking distributor) to mint VSQ. It should be noted that no more VSQ can be minted than the total number of reserves in \$. If VSQ would be freely exchangeable for the reserves, this puts a lower limit of \$1 on the value of VSQ as long as no reserves are lost.

It should be noted that the treasury code is currently written extremely verbose and implementing a dependency like AccessControl (RBAC) might simplify the contract greatly.





2.12.1 Privileged Roles

The following functions can be called by the Staking contract:

- `auditReserves`
- `toggle`
- `incurDebt`
- `withdraw`
- `deposit`
- `repayDebtWithReserve`
- `repayDebtWithOhm`
- `manage`
- `mintRewards`
- `queue`
- `transferOwnership`
- `claimOwnership`




2.12.2 Issues & Recommendations

Issue #39	Lack of component safeguards in a system that plans to increase in number of components over time is considered brittle
Severity	 LOW SEVERITY
Description	<p>The treasury is responsible for minting new VSQ. Any account with the reward manager role can do this. This however also means that if any single of these accounts or contracts would be compromised, the whole system would fail.</p> <p>Such a practice is not bad in itself, but it is a setup we call 'brittle'. In general, when the security of a system is based upon all components acting correctly, and this set of components is planned to increase over time, odds are that one day a component will misbehave and the whole system goes under. This has been witnessed with Cream recently on Ethereum and more traditionally with PancakeBunny (and many of their forks) on BSC and other chains.</p>
Recommendation	Consider incorporating hourly limits to all functions within the treasury, each account can only mint/borrow/... up to their hourly limit every hour. Permissions should be pausable instantly by the DAO. With such a setup, if a new component ever turns out to have a vulnerability, only a few hours of mints might be stolen.
Resolution	 ACKNOWLEDGED
	<p>The client has acknowledged the implications of not having hourly limits and they will take full responsibility for making sure the components will behave as expected.</p>

Issue #40

Adding a token as both a liquidity and reserve token would cause it to be double counted in the treasury value

Severity

 LOW SEVERITY

Description

Currently the function that calculates the value of the reserves does not validate that a token has already been counted. This means that if the same token is added twice to the reserves lists, this token would be double counted.

The client has considered this possibility by not allowing a token to be added twice to either the reserve tokens list or the liquidity tokens list. However, the possibility remains open that the token is added to both the reserve and liquidity tokens list once, which would cause double counting.

Recommendation


Consider either not double counting in the reserve value calculating function, or consider not allowing a token to be added to either of these lists.

Resolution

 RESOLVED

Checks have been added to prevent this double addition.



Issue #41**repayDebtWithVSQ has inconsistent privilege requirements which allows for slight privilege escalation****Severity** LOW SEVERITY**Description**

Currently the repayDebtWithVSQ function requires the sender to have the "debtor" role, which essentially means they can borrow from the reserve. However, this operation also does a withdrawal from the reserve which normally requires the "reserve spender" role. This role verification is not made however.

To clarify on this: repayDebtWithVSQ is essentially a combination of withdraw, which allows you to withdraw reserves if you burn an equivalent amount of VSQ and repayDebtWithReserve, which allows you to repay your debt by transferring tokens to the reserve. repayDebtWithVSQ combines these by having you repay your debt by burning VSQ.



As the roles already have very large privileges within the system, this issue is only marked as low risk since it hardly increases the risk profile.

Recommendation

Consider also requiring the reserveSpender role for the repayDebtWithVSQ function, as this behavior seems inconsistent with what the roles should be allowed to do.

Resolution RESOLVED

repayDebtWithVSQ now requires the reserve spender role as well.

Issue #42	Unnecessary comparison to true on withdraw function
Severity	 INFORMATIONAL
Location	<u>Line 364</u> require(isReserveSpender[msg.sender] == true, "Not approved");
Description	In the withdraw function, the require checks to see if the msg.sender is in the isReserveSpender mapping, which is a mapping of address=>bool. Therefore comparison to true is redundant.
Recommendation	Consider removing the comparison to true. require(isReserveSpender[msg.sender], "Not approved");
Resolution	 RESOLVED



Issue #43**Reserve value mechanism could cause withdrawals and other operations to temporarily fail****Severity** INFORMATIONAL**Description**

Whenever tokens are added or deleted to the Treasury, the present (USD) value of them is added or subtracted to the `totalReserves`. However, if their value increases over time for some reason, withdrawals might revert because the functions try to reduce `totalReserves`. This could furthermore be abused if ever an oracle (`bondCalculator`) is used that turns out to be manipulatable, in this case a malicious party can always increase the value of the currency before it is withdrawn to potentially cause that withdrawal to revert.



Recommendation



As this is a purely informational issue no specific steps need to be taken unless the client considers this prohibitive. The client should simply make sure to call `auditReserves` if withdrawals start failing and build no functionality that relies on withdrawals to work 100% of the time (otherwise this functionality should call `auditReserves`).

If withdrawals start failing often due to DoS, the client should remember this issue and double check if some oracle is malfunctioning.

Resolution RESOLVED

The client has indicated they will call `auditReserves` if this issue is ever detected.

Issue #44 Lack of safeTransfer usage within incurDebt	
Severity	 INFORMATIONAL
Location	<u>Line 398</u> IERC20(_token).transfer(msg.sender, _amount);
Description	In the incurDebt function, the transfer method is used to transfer tokens from Treasury to the msg.sender. This will not work for tokens that will return false on transfer (or malformed tokens that do not have a return value).
Recommendation	Consider using safeTransfer instead of transfer as is done throughout most of this contract.
Resolution	 RESOLVED

Issue #45 Manage will always do an excessReserve check even if the token is not within the liquidity or reserves tokens	
Severity	 INFORMATIONAL
Location	<u>Lines 451-452</u> uint value = valueOf(_token, _amount); require(value <= excessReserves(), "Insufficient reserves");
Description	Currently, the reserves are only comprised of "liquidity" and "reserve" tokens. Therefore, if an asset which is not within these two categories is withdrawn from the Treasury, it should not affect the excess reserves and the excess reserves validation is therefore redundant.
Recommendation	Consider wrapping the excess reserve check in an if statement that only executes if the token that is being withdrawn is part of the liquidity or reserves tokens. Otherwise consider requiring the token being withdrawn to be within either of these categories.
Resolution	 RESOLVED The check has been wrapped in an if-statement.

2.13 VSQZapIn



The VSQZapIn contract is a zapping contract that will allow users to deposit into the BondDepository within a single transaction. It currently does not support transfer-tax tokens at all. It can use any type of exchange contract to fulfill the quotes and is not limited to Uniswap-v2 compatible exchanges. It furthermore uses an advanced calculation method similar to Zapper to almost perfectly balance the pair when adding liquidity, this reduces potential tokenomical waste.



2.13.1 Privileged Roles

The following functions can be called by the Staking contract:

- `toggleContractActive`
- `setApprovedTargets`
- `addPairAddress`
- `removePairAddress`
- `addReserveAddress`
- `removeReserveAddress`
- `renounceOwnership`
- `transferOwnership`

2.13.2 Issues & Recommendations

Issue #46	
Phishing risk: Users could be mislead into undesirable swaps	
Severity	 LOW SEVERITY
Description	The contract allows the governance to provide different swap exchanges and allows the inputs to be set in a way that potentially tokens could be lost by the user if they do not pay careful attention. Furthermore the "to" address which would receive the VSQ features can be changed to a malicious address.
Recommendation	Consider carefully protecting the frontend against any malicious take-overs (as have happened in the past with both Cream and PancakeSwap). Consider furthermore educating the users on checking their transactions carefully.
Resolution	 PARTIALLY RESOLVED Although this is possible, the client has indicated that their frontend has undergone proper security steps and that they will document this within their docs.



Issue #47	
swapData can be denoted as calldata throughout the contract	
Severity	 INFORMATIONAL
Description	As all internal functions that use the swapData bytes are exclusively called by external functions that provide these bytes, the swapData parameters can always be denoted as calldata to save significantly on gas cost.
Recommendation	Consider moving all swapData parameters to calldata.
Resolution	 RESOLVED



2.14 Code style-related Issues

The following are coding style issues that Paladin spotted throughout the contracts of the VESQ protocol. Paladin has aggregated the ones that occurred frequently into this section to shorten the report.





2.14.1 Issues & Recommendations

Issue #48	Inconsistency: Unused mint function emits an event from address(this) while the mint logic during initialization emits an event from the zero address
Severity	 INFORMATIONAL
Description	<p>The <code>_mint</code> function is inconsistent with the way tokens are actually minted in that it sets the token contract itself as the transfer origin. This would likely mislead explorers and third-party tools into thinking that tokens were taken out of the token contract itself.</p> <p><u>Functions</u></p> <ul style="list-style-type: none">- VSQERC20: <code>_mint</code>- sVSQERC20: <code>_mint</code> <p>This issue has been marked as informational as <code>_mint</code> is presently unused, making this a purely informational concern.</p>
Recommendation	Consider making <code>_mint</code> consistent with the recommended practice of using address zero as the origin for the mint transfer.
Resolution	 RESOLVED

Issue #49	Various functions can be made external
Severity	 INFORMATIONAL
Description	<p>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.</p> <p><u>Functions</u></p> <ul style="list-style-type: none">- VSQERC20: burn and burnFrom- Staking: claim and index- StakingDistributor: nextRewardFor
Recommendation	Consider marking the above variables as external.
Resolution	 RESOLVED



Issue #50	Lack of events for various functions
Severity	<div data-bbox="459 208 635 235">INFORMATIONAL</div>
Description	<p>Functions that affect the status of sensitive variables should emit events as notifications.</p> <ul style="list-style-type: none"> - sVSQERC20: <code>setIndex</code> ! The return value is furthermore unnecessary unless this is some highly standard interface like ERC-20. - wsVSQ: <code>wrap</code> and <code>unwrap</code> - Staking: <code>stake</code>, <code>claim</code>, <code>forfeit</code>, <code>toggleDepositLock</code>, <code>unstake</code>, <code>rebase</code>, <code>giveLockBonus</code>, <code>returnLockBonus</code>, <code>setContract</code> and <code>setWarmup</code> - BondDepository: <code>initializeBondTerms</code>, <code>setBondTerms</code>, <code>setAdjustment</code>, <code>setStaking</code> and <code>recoverLostToken</code> - StakingDistributor: <code>distribute</code>, <code>adjust</code>, <code>addRecipient</code>, <code>removeRecipient</code> and <code>setAdjustment</code> - StakingHelper: <code>stake</code> - VSQZapIn: <code>setApprovedTargets</code>, <code>toggleContractActive</code>, <code>addPairAddress</code>, <code>removePairAddress</code>, <code>addReserveAddress</code> and <code>removeReserveAddress</code>
Recommendation	<p>Add events for the above functions. Consider removing the return variable.</p>
Resolution	<div data-bbox="459 1473 593 1505">RESOLVED</div>

Issue #51	Typographical errors
Severity	 INFORMATIONAL
Description	<p><u>Contracts</u></p> <ul style="list-style-type: none"> - sVSQERC20: Token comments mention ERC-777 which is not relevant to sVSQ <p><u>Line 537 (example)</u> <i>// Present in ERC777</i></p> <p>Throughout the ERC20 dependency of the tokens, references to EIP-777 are made, this improved token standard is known to cause many exploits as it allows for reentrancy on any transfer, it could therefore scare less adept third-party reviewers into thinking this is an ERC-777 token while it in fact is not.</p> <p>Line 629 furthermore contains a typographical error: <i>// Overrideen in ERC777</i></p> <ul style="list-style-type: none"> - Staking: setWarmup is an ambiguous function name and should be renamed to setWarmupPeriod. - BondDepository: unnecessary convert of treasury to address <p><u>Line 889</u> address(treasury)</p> <ul style="list-style-type: none"> - EthBondDepository Throughout the contract blocks are referred in the comments even though VESQ deployment is using a timestamp approach. IWETH9 is used as an interface name for Wrapped MATIC which may be misleading for third-parties who might want to inspect the contract. - VSQZapIn: The contract still mentions Quickswap throughout the comments.
Recommendation	Consider fixing the above typographical errors.
Resolution	 RESOLVED

Severity

 INFORMATIONAL

Description

The contract includes unused variables. These unnecessarily increase the contract source code size and gas consumption, also it can make third-party reviewing more cumbersome.

Contracts

- **VSQERC20**: ERC20TOKEN_ERC1820_INTERFACE_ID and unused dependency EnumerableSet

Line 606

```
bytes32 constant private ERC20TOKEN_ERC1820_INTERFACE_ID =  
keccak256( "ERC20Token" );
```

The custom token dependency contains a constant variable containing a hashed interface identifier. However, this variable is not used throughout the contract.

There are furthermore many references to ERC-777 which could mislead less adept code reviewers into believing this is an ERC-777 token. As ERC-777 tokens are traditionally associated with exploit vulnerability, it is best to avoid such confusion.

- **wsVSQ**: Address and SafeERC20

Lines 749-750

```
using SafeERC20 for ERC20;  
using Address for address;
```

- **BondDepository**: ERC20, ERC20Permit, IERC2612Permit and Counters
- **StakingDistributor**: SafeERC20
- **VSQZapIn**: _getBalance

Recommendation

Consider removing the above variables.

Resolution

 RESOLVED

Issue #53**Gas optimization: Contract uses hardcoded strings in SafeMath functions****Severity** INFORMATIONAL**Location**

VSQERC20::Line 895-899 (example)

```
uint256 decreasedAllowance_ =  
    allowance(account_, msg.sender).sub(  
        amount_,  
        "ERC20: burn amount exceeds allowance"  
    );
```

Description



The contract injects the error message into SafeMath. This is known to cost extra gas, even on the happy path, as it causes memory allocation.

Recommendation

Consider checking the identity explicitly using a require statement and then using non-safe math to do the subtractions and additions instead. SafeMath has also created the trySub and tryAdd functions in more recent versions to address this gas usage concern.

Resolution PARTIALLY RESOLVED

The example location has been resolved but the gas inefficiency remains present in multiple parts of the codebase. We want to note that this is no issue for users at all as it is only a gas optimization issue and the client will launch on an extremely cheap network from a gas-cost perspective.

Issue #54	Uncast addresses make the code more verbose than it needs to be
Severity	 INFORMATIONAL
Location	<u>StakingHelper::Line 95</u> IERC20(VSQ).transferFrom(msg.sender, address(this), _amount);
Description	<p>Throughout the contract, addresses are stored using the address type instead of the interface type. This requires the code to cast them to the correct interface every time these addresses are used and makes them prone to typing errors. It should be noted that address typing is purely syntactic and does not have any runtime benefits (either through performance security).</p>
Recommendation	<p>Consider casting all addresses to the correct types within the storage portions of the contract. This is done in a large number of locations so the client will have to simply go over all contracts to do this.</p> <p>Sometimes an address has multiple types (eg. IsVSQ + IERC20), in this case we recommend making an aggregate interface that inherits both of them or making IsVSQ inherit IERC20 in this example.</p>
Resolution	 ACKNOWLEDGED



Issue #55**Ambiguous errors****Severity** INFORMATIONAL**Location**

(Examples)

sVSQERC20::Line 1011

```
require(msg.sender == stakingContract);
```

sVSQERC20::Line 1070

```
require(INDEX == 0);
```

Description

The contract contains locations of code which do not revert with an error message, instead they revert ambiguously leaving users to potentially wonder what happened with their transaction. It makes writing coverage tests furthermore difficult as these cannot explicitly check for the reversion method.

Within the sVSQ token, often transfers also revert with ambiguous errors if the user does not have enough allowance or tokens. This is likely the most severe location of this issue as these events might occur frequently.



Recommendation

Consider adding explicit reversion messages to the aforementioned locations and any other reversion locations which could cause a worse user experience.

Resolution PARTIALLY RESOLVED

Error messages have been added in many locations of the code except the following.

- BondDepository/EthBondDepository missed recoverTokenLost, constructor, setStacking errors
- Staking: constructor
- StakingDistributor: addRecipient, constructor
- StakingHelper: constructor
- StakingWarmup: retrieve
- StandardBondingCalculator: constructor
- Treasury: constructor, queue
- wsVSQ: constructor

Issue #56	Gas optimization: storage variables are frequently unnecessarily reread
Severity	 INFORMATIONAL
Location	(Examples) <u>sVSQERC20::Line 1191-1192</u> <pre>_allowedValue[msg.sender][spender] = _allowedValue[msg.sender][spender].add(addedValue); emit Approval(msg.sender, spender, _allowedValue[msg.sender][spender]);</pre>
Description	<p>The contract often unnecessarily re-reads variables from storage, while they could be derived from variables stored in memory. This causes gas to be wasted unnecessarily (about 200 gas per read).</p> <p>This issue is aggregated into a single issue as we wish to not unnecessarily clutter the report with a high issue count given that the client is unlikely to want to make many changes to the codebase for micro-optimization. Upon request by VESQ, our internal documentation with all locations of code that can be optimized can be provided either in the report or privately.</p>
Recommendation	Within derivative protocols, one can consider using try-catch for permit and validating the approval afterwards.
Resolution	 ACKNOWLEDGED



PALADIN
BLOCKCHAIN SECURITY