



PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Vesq (Presale)

18 November 2021



paladinsec.co



info@paladinsec.co

Table of Contents

Table of Contents	2
Disclaimer	3
1 Overview	4
1.1 Summary	4
1.2 Contracts Assessed	4
1.3 Findings Summary	5
1.3.1 VSQPresale	6
1.3.2 VSQTokenRedeem	6
1.3.3 Locker	6
2 Findings	7
2.1 VSQPresale	7
2.1.1 Issues & Recommendations	8
2.2 VSQTokenRedeem	12
2.2.1 Issues & Recommendations	13
2.3 Locker	14
2.3.1 Issues & Recommendations	15

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's presale contracts 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 (Presale)
URL	https://www.vesq.io/
Platform	Polygon
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
VSQPresale	VSQPresale.sol	
VSQTokenRedeem	VSQTokenRedeem.sol	
Source	https://github.com/VESQHQ/vesq-contracts/tree/7f09297c85060aa8b16e4cbf78bee4a0c33aa091	

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● High	0	-	-	-
● Medium	1	1	-	-
● Low	2	2	-	-
● Informational	8	8	-	-
Total	11	11	-	-

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 VSQPresale

ID	Severity	Summary	Status
01	LOW	Current per wallet limit scheme incentivizes users to wait until the very end of the presale	RESOLVED
02	INFO	Lack of constructor validation	RESOLVED
03	INFO	Contract only works for specific token configurations	RESOLVED
04	INFO	Unnecessary precision multiplier	RESOLVED
05	INFO	sendUnclaimedToTreasuryAddress does not adhere to checks-effects-interactions	RESOLVED
06	INFO	Erroneous usage of assert	RESOLVED
07	INFO	Error message in sendUnclaimedToTreasuryAddress states that it is going to burn the tokens	RESOLVED

1.3.2 VSQTokenRedeem

ID	Severity	Summary	Status
08	MEDIUM	1e9 multiplier is inconsistent with VSQPresale	RESOLVED
09	INFO	Lack of constructor validation	RESOLVED

1.3.3 Locker

ID	Severity	Summary	Status
10	LOW	Lack of constructor validation	RESOLVED
11	INFO	Odd contract layout	RESOLVED

2 Findings



2.1 VSQPresale



The VSQPresale contract allows whitelisted users to purchase VSQ tokens with FRAX tokens. The initial price is set at 0.04 VSQ per FRAX supplied and can be changed between 0.004 VSQ and 0.4 VSQ until 4 hours before the start of the presale. After this time, the price becomes fixed within the contract and its value is readable in the `salePriceE35` public variable, which needs to be divided by 10^{35} to retrieve the conversion rate (this variable is therefore presently $4 \cdot 10^{33}$).



The presale size is fixed at 50,000 VSQ tokens and will take four days. Whitelisted users can only make a single purchase and can purchase up to the remaining presale size divided by the amount of remaining whitelisted users that have not purchased yet. If the remaining presale size is 1000 VSQ and 10 whitelisted members remain, this would set the limit at 100 VSQ per user.



Any unsold VSQ tokens are sent to the treasury at the end of the presale.



2.1.1 Issues & Recommendations



Issue #01	Current per wallet limit scheme incentivizes users to wait until the very end of the presale
Severity	 LOW SEVERITY
Location	<u>Line 62</u> <code>uint256 maxVSQPurchase = VSQRemaining / remainingBuyers;</code>
Description	<p>Currently, the maximum amount of VSQ a single wallet can purchase is based on the amount of remaining VSQ divided by the amount of remaining buyers.</p> <p>As users can only ever purchase at most up to this amount, this value is non-decreasing with purchases. This means that over time, the maxVSQPurchase can only go up which might incentivize whitelisted users to wait until the very last second of the presale to have the maximum possible maxVSQPurchase value.</p>
Recommendation	Consider whether this is desired behavior, if not, consider redesigning the behavior to allow for less time-critical incentives.
Resolution	 RESOLVED <p>The client has indicated this is desired behavior. Users can therefore wait for potential gain at the risk of being too late.</p>



Issue #02 Lack of constructor validation	
Severity	 INFORMATIONAL
Description	Currently there is no validation that the parameters with which the contract has been created make sense, this could lead to accidental mistakes if the deployment isn't done carefully.
Recommendation	Consider validating that startBlock is in the future and that the addresses are non-zero.
Resolution	 RESOLVED The recommended validation has been added.

Issue #03 Contract only works for specific token configurations	
Severity	 INFORMATIONAL
Description	<p>Currently the VSQPresale requires the VSQ token to be 9 decimals, the FRAX token to be 18 decimals, and neither of the tokens to require SafeERC20.</p> <p>This issue is marked as informational since these requirements do in fact match the expected deployment of the client. We have still included this as an issue in case the client wishes to deploy to a new network for example.</p> <p>It should finally be noted that a check within the purchase function is out of bounds under the current configuration:</p> <pre>require(fraxToSpend <= 1.25 * 1e25, "too much frax to spend");</pre>
Recommendation	Consider using SafeERC20, consider calling .decimals() on the tokens to generalize the logic to any token decimals.
Resolution	 RESOLVED The client has generalized the contract to work for any decimals. The out of bounds check has also been removed. SafeERC20 is now used consistently.

Issue #04 Unnecessary precision multiplier	
Severity	 INFORMATIONAL
Location	<u>Line 91</u> <code>fraxSpent = ((VSQPurchaseAmount * fraxToSpend * 1e24) / originalVSQAmount) / 1e24;</code>
Description	Line 91 multiplies and divides by 1e24. However, since these two operations are done within the same line of code, they cancel each other out. Simply adhering to multiplication before division would result in the exact same level of precision.
Recommendation	Consider validating that removing this precision multiplier results in no precision loss and then consider removing it.
Resolution	 RESOLVED

Issue #05 sendUnclaimedToTreasuryAddress does not adhere to checks-effects-interactions	
Severity	 INFORMATIONAL
Description	<p>Currently the governance function <code>sendUnclaimedToTreasuryAddress</code> does not adhere to checks-effects-interactions.</p> <p>This issue has been marked as informational since Paladin could not find severe consequences of this.</p>
Recommendation	<p>Consider moving the following effect to the top of the function, right under the checks (require calls).</p> <pre>hasRetrievedUnsoldPresale = true;</pre>
Resolution	 RESOLVED



Issue #06 Erroneous usage of assert	
Severity	 INFORMATIONAL
Location	<u>Line 82</u> <pre>assert(VSQPurchaseAmount <= IERC20(VSQAddress).balanceOf(address(this)));</pre>
Description	Solidity contracts should only use <code>assert</code> for checks that can never fail, but this is not the case for the check on line 82 as there is no guarantee that there is enough VSQ in the contract to cover this amount.
Recommendation	Consider using <code>require</code> for the check on line 82.
Resolution	 RESOLVED



Issue #07 Error message in <code>sendUnclaimedToTreasuryAddress</code> states that it is going to burn the tokens	
Severity	 INFORMATIONAL
Location	<u>Line 107</u> <pre>require(!hasRetrievedUnsoldPresale, "can only burn unsold presale once!");</pre>
Description	The <code>sendUnclaimedToTreasuryAddress</code> function returns an error that the tokens would be burned. However, the unsold tokens are in fact sent to the treasury.
Recommendation	Consider correcting the error message.
Resolution	 RESOLVED

2.2 VSQTokenRedeem

The VSQTokenRedeem allows redeeming the Presale VSQ token for a VSQ token variant.

2.2.1 Issues & Recommendations

Issue #08	1e9 multiplier is inconsistent with VSQPresale
Severity	 MEDIUM SEVERITY
Location	<u>Lines 35-36</u> IERC20(preVSQ).transferFrom(msg.sender, BURN_ADDRESS, VSQSwapAmount); IERC20(VSQAddress).transfer(msg.sender, VSQSwapAmount * 1e9);
Description	The swap grants 10 ⁹ times more VSQ tokens than preVSQ that needs to be sent. Since the VSQPresale contract already seems to be dealing with a 9 decimal token, we do not presently understand why this multiplication is being made. If this is by error, this would result in the first swapper likely withdrawing all VSQ tokens.
Recommendation	Consider explaining why this 1e9 multiplication is done. If it is there by error, consider removing it.
Resolution	 RESOLVED The code has been generalized for any decimal count. This issue was never present within the current deployment as the client their presale VSQ token, which was outside of the audit scope, apparently had 0 decimals.



Issue #09	Lack of constructor validation
Severity	 INFORMATIONAL
Description	Although there is constructor validation already, it is not complete.
Recommendation	Consider requiring that _VSQAddress is nonzero.
Resolution	 RESOLVED

2.3 Locker

The Locker contract is a simple ERC20 token locker which allows the owner to define an unlock block number. Any tokens which have been sent to the Locker only become withdrawable once this block number, publicly inspectable as `UNLOCK_BLOCKNUMBER`, has been reached.



2.3.1 Issues & Recommendations

Issue #10	Lack of constructor validation
Severity	 LOW SEVERITY
Description	Currently the constructor is not validated – this might cause accidental error or cause users to not notice the fact that the UNLOCK_BLOCKNUMBER has been set to a value in the past.
Recommendation	Consider requiring the UNLOCK_BLOCKNUMBER to be at least a reasonable amount in the future within the constructor. Furthermore, if infinite locks are not a requirement, consider adding a maximum to avoid user error.
Resolution	 RESOLVED

Issue #11**Odd contract layout****Severity** INFORMATIONAL**Description**

Although the contract is perfect from a code design perspective, the contract is formatted strangely. The contract definition has been placed all the way at the top where the license identifier would go. Furthermore, NatSpec has been used to document the constructor but was not used to document `claimToken`. If ever a documentation where to be generated using tools that can interpret NatSpec, it might be valuable to be consistent with using it.

Recommendation

Consider using NatSpec consistently throughout the contract and consider moving the contract definition comment to right above line 7. Line 1 should be reserved for the license identifier.

Consider also including the Solidity pragma version.

The constructor visibility (`public`) is unnecessary if the contract were to be used on Solidity 0.8.0 or higher.

Finally the constructor has a wrongly copied comment which needs to be removed:

Constructs the PlushToken contract.

Resolution RESOLVED



PALADIN
BLOCKCHAIN SECURITY