QuillAudits

Audit Report
July, 2021

PolyWhirl

Contents

Audit Details and Target

1. Contract Link

Link	Commit hash	UpdateVersion
Here	0306f7d3ef393b0b486d1cbe38897589403e1be3	Version 1
	3ab8d8da2ff11ab4e19224a24332fd494a3791f4	Version 2
	f81382d4bcfb16ce1463db832da5fa0a41ccf6df	Version 3

2. Audit Target

- To find the security bugs and issues regarding security, potential risks, and critical bugs.
- Check gas optimisation and check gas consumption.
- Check function reusability and code optimisation.
- Test the limit for token transfer and check the accuracy in decimals.
- Check the functions and naming conventions.
- Check the code for proper checks before every function call.
- Event trigger checks for security and logs.
- Checks for constant and function visibility and dependencies.
- Validate the standard functions and checks.
- Check the business logic and correct implementation.
- Automated script testing for multiple use cases including transfers, including values and multi transfer check.
- Automated script testing to check the validations and limit tests before every function call.
- Check the use of data type and storage optimisation.
- Calculation checks for different use cases based on the transaction, calculations and reflected values.

Functions list and audit details

Major Functions

- removeOtherERC20Tokens()
- withdrawUnsoldTokens()
- withdrawFunds()
- markRaiseInComplete()
- markRaiseComplete()
- redeemTokens()
- purchaseVC()
- purchaseIndividual()
- purchaseInfluencer()
- removeWhitelistVC()
- removeWhitelistIndividual()
- removeWhitelistInfluencer()
- whitelistVC()
- whitelistIndividual()
- whitelistInfluencer()

Overview of the contract

Polywhirl is a token and private sale contract based on the logic of investing and redeeming the tokens based on the calculations. Code is written with all possible checks and standard library functions for managing the token transfer, owner functions, user checks, claim, and redeem the tokens.
Code uses the latest stable version of solidity and ensures the correctness and transparency in the contract for every user.
Private sale is divided into three types of users: Influencers, Individuals, and VC.
Approve functions need to be used before many purchases and claims which need to be managed by the frontend, or if users are iterating directly to the contract then need to be approved at their end for the contract address.
Private sale contract only supports USDT coin so stable coin should be this or a similar one as per the contract logic.

Tokenomics

As per the information provided, the tokens generated will be initially transferred to the contract owner, and then further division will be done based on the business logic of the application. Tokens cannot be directly purchased from the smart contract, so there will be an additional or third-party platform that will help users to purchase the tokens. Tokens can be held, transferred, and delegated freely. Tokens are generated based on the supply, which can be checked by total supply.

Scope of Audit

The scope of this audit was to analyse Polywhirl smart contracts codebase for quality, security, and correctness. Checks for code optimisation, tokens security, potential risks, compatibility with multiple platforms and wallets.

Checked Vulnerabilities

The smart contract is scanned and checked for multiple types of possible bugs and issues. This mainly focuses on issues regarding security, attacks, mathematical errors, logical and business logic issues. Here are some of the commonly known vulnerabilities that are considered:

- TimeStamp dependencies.
- Variable and overflow
- Calculations and checks
- SHA values checks
- Vulnerabilities check for use case
- Standard function checks
- Checks for functions and required checks
- Gas optimisations and utilisation
- Check for token values after transfer
- Proper value updates for decimals
- Array checks
- Safemath checks
- Variable visibility and checks
- Error handling and crash issues
- Code length and function failure check
- Check for negative cases
- D-DOS attacks
- Comments and relevance
- Address hardcoded
- Modifiers check
- Library function use check
- Throw and inline assembly functions
- Locking and unlocking (if any)
- Ownable functions and transfer ownership
- checksArray and integer overflow possibility checks
- Revert or Rollback transactions check

Techniques and Methods

- Manual testing for each and every test cases for all functions.
- Running the functions, getting the outputs and verifying manually for multiple test cases.
- Automated script to check the values and functions based on automated test cases written in JS frameworks.
- Checking standard function and check compatibility with multiple wallets and platforms
- Checks with negative and positive test cases.
- Checks for multiple transactions at the same time and checks d-dos attacks.
- Validating multiple addresses for transactions and validating the results in managed excel.
- Get the details of failed and success cases and compare them with the expected output.
- Verifying gas usage and consumption and comparing with other standard token platforms and optimizing the results.
- Validate the transactions before sending and test the possibilities of attacks.

Structural Analysis
In this step we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.
SmartCheck.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of automated analysis were manually verified.

Static Analysis
Static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step a series of automated tools are used to test security of smart contracts.

Gas Consumption
In this step we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

High severity issues

Issues that must be fixed before deployment else they can create major issues.

Medium level severity issues

These issues will not create major issues in working but affect the performance of the smart contract.

Low level severity issues

These issues are more suggestions that should be implemented to refine the code in terms of gas, fees, speed and code accuracy

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

	High	Medium	Low	Informational
Open	0	0	0	0
Closed	0	0	3	3

Issues Found – Code Review / Manual Testing

High severity issues

No issues found under this category.

Medium severity issues

No issues found under this category.

Low level severity issues

1. **Contract:** PrivateSale.sol
   Line: 161, 213, 265 - All three functions have the same issue.
   Issues: if approved stable coin is less, it shows an incorrect error.
   Reasons: No check for approval before transfer.
   Suggested Fixes: Check for approved stable coin is needed before transaction.
   Status: Resolved
   The issue was resolved in version 2.

```
transact to PrivateSale.purchaseVc errored: VM error: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "Invalid values".
Debug the transaction to get more information.
```

2. **Contract:** PrivateSale.sol
   Line: redeemTokens Line: 311
   Issues: Incorrect error message or no check for less balance of the contract.
   Reasons: Need a check for contract balance before purchasing of tokens so that users will not be left with failing transactions at the time of token redeem.
   Suggested Fixes: Add appropriate checks for the same.
   Status: Resolved
   The issue was resolved in version 2.

```
transact to PrivateSale.redeemTokens errored: VM error: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "ERC20: transfer amount exceeds balance".
Debug the transaction to get more information.



call to WHIRL.balanceOf
```

3. **Contract:** PrivateSale.sol
   Line: 98,109,120
   Issues: Failing with large number of array values/ No check for 0x0 address, which can be a security concern.
   Reasons: check for max value for the array and check for 0x0 address.
   Suggested Fixes: Add appropriate checks for the same.
   Status: Resolved
   The issue was resolved in version 3.

```
function whitelistInfluencer(address[] memory _influencer_addresses)
    external
    onlyOwner
{
    for (uint256 i = 0; i < _influencer_addresses.length; i++) {
        require(influencer_whitelist[_influencer_addresses[i]] != true);
        influencer_whitelist[_influencer_addresses[i]] = true;
    }
    emit addedToInfluencerWhitelist(_influencer_addresses);
}
```

Informational

4. **Contract:** PrivateSale.sol
   Line: 174, 227, 273
   Issues: Extra gas usage
   Reasons: Extra gas usage
   Suggested Fixes: As the value is constant, so it can be declared constant or declared outside the function to save runtime gas cost. Add comment to display the user, the formula for calculation.
   Status: Resolved
   The issue was resolved in version 2.

```
173             );
174             uint256 stable_coin_amount = influencer_tokens
175             .mul(exchange_rate_cents)
176             .div(100) // cents to USD
177             .mul(
178                 1e6 // USDT or USDC both have 6 decimals
179             );
```

5. **Contract:** PrivateSale.sol
   Issues: No comment added in the code for any approve function, formula display or any condition for functions.
   Status: Resolved
   The issue was resolved in version 3.

6. **Contract:** PrivateSale.sol
   Line: 288, 190, 242
   Issues: Assignment operator missing so raised_funds variable value will never change or used.
   Status: Resolved
   The issue was resolved in version 2.

Functional Test Table

Function Names	Technical results	Logical results	Overall
removeOtherERC20Tokens()	Pass	Pass	Pass
withdrawUnsoldTokens()	Pass	Pass	Pass
withdrawFunds()	Pass	Pass	Pass
markRaiseInComplete()	Pass	Pass	Pass
markRaiseComplete()	Pass	Pass	Pass
redeemTokens()	Pass	Pass	Pass
purchaseVC()	Pass	Resolved	Resolved
purchaseIndividual()	Pass	Resolved	Resolved
purchaseInfluencer()	Pass	Resolved	Resolved
removeWhitelistVC()	Pass	Pass	Pass
removeWhitelistIndividual()	Pass	Pass	Pass
removeWhitelistInfluencer()	Pass	Pass	Pass
whitelistVC()	Pass	Resolved	Resolved
whitelistIndividual()	Pass	Resolved	Resolved
whitelistInfluencer()	Pass	Resolved	Resolved

Closing Summary

All the issues mentioned above are resolved and re-checked for correctness. This contract can be used for private sale purpose with the given logic.
There is no security issue or bugs for the tokens, so this can be used for the logic as mentioned.

Disclaimer

QuillHash audit is not a security warranty, investment advice, or an endorsement of the Polywhirl platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Polywhirl Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

PolyWhirl

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com