POLARS

SMART CONTRACT AUDIT

ZOKYO.

July 23th, 2021 | v. 1.0

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges

SCORE

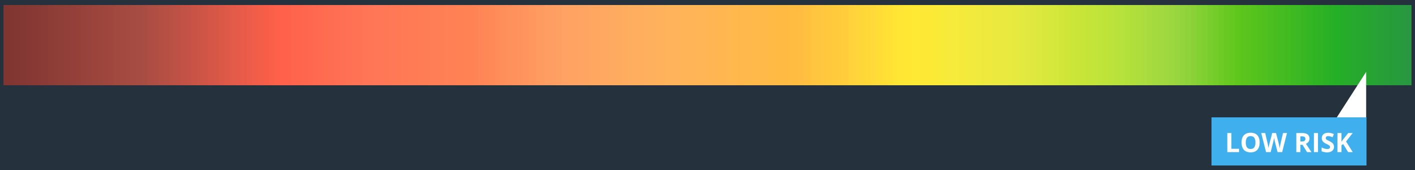
97

TECHNICAL SUMMARY

This document outlines the overall security of the Polars smart contracts, evaluated by Zokyo's Blockchain Security team.

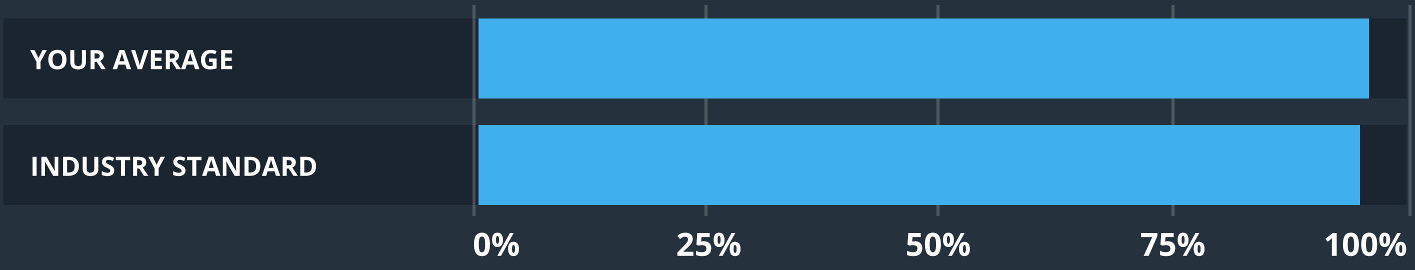
The scope of this audit was to analyze and document the Polars smart contract codebase for quality, security, and correctness.

Contract Status



There were no critical issues found during the audit.

Testable Code



The testable code is 96%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the Polars team put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the Polars repository – https://github.com/yurivin/Polars_Zokyo/commit/2731b277d50eabe69cf0aacc2d496b362572d22

Last commit – 00134fb15fc0a57d1d1b7b9b63b3d6f695560721

Throughout the review process, care was taken to ensure that the token contract:

- Implements and adheres to existing Token standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of gas, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of Polars smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1. Due diligence in assessing the overall code quality of the codebase.
2. Cross-comparison with other, similar smart contracts by industry leaders.
3. Testing contract logic against common and uncommon attack vectors.
4. Thorough, manual review of the codebase, line-by-line.

SUMMARY

Generally, the contracts provided for an audit are well written and structured. All the findings within the auditing process are presented in this document.

There were no critical issues found during the audit. All the mentioned findings may have an effect only in case of specific conditions performed by the contract owner. The findings during the audit have no impact on contract performance or security, so it is fully production-ready.

Taking into account the fact that mostly all of the issues were resolved and evaluating the contracts from the operational and security standpoints, we can give the score of 97%.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

● Critical

The issue affects the ability of the contract to compile or operate in a significant way.

● High

The issue affects the ability of the contract to compile or operate in a significant way.

● Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

● Low

The issue has minimal impact on the contract’s ability to operate.

● Informational

The issue has no impact on the contract’s ability to operate.

MANUAL REVIEW

Pool may not be able to withdraw bwTokens

MEDIUM | RESOLVED

In the SecondaryCollateralizationBWT.sol file there is a delegate function that moves bw and collateral tokens. But black and white tokens are left in the first pool.

So, in the first pool function withdraw will revert due to missing _bwTokens, in second pool due to _blackTokens or_whiteTokens.

Additional note: this function doesn't seem to be used at all.

Recommendation:
If this is expected behaviour - add a comment with description, If no, there are several possible solutions, depending on requirements, example:

- Don't allow to delegate if balance of white or black token > 0;
- Keep required amount of bwTokens to burn all black / white tokens;
- Check if newCollateralization supports black / white tokens, if so - move them along with other tokens;
- Remove this function.

Misleading event

LOW | RESOLVED

In SecondaryPoolBWT.sol file there are two misleading events:

```
emit BlackPriceCase1(eend.whitePrice); //line 429
emit WhitePriceCase1(eend.blackPrice); //line 470
```

Recommendation:
Depending on requirements, switch tokens or events. For example in case tokens: set black on line 429 and white on line 470.

Misleading error message [1]

LOW | RESOLVED

In SecondaryCollateralizationBWT.sol file in constructor there is a require:

```
require (bwtAddress != address(0), "WHITE TOKEN ADDRESS SHOULD NOT BE NULL");
```

Recommendation:
Replace "WHITE TOKEN" with "BWT TOKEN".

Misleading error message [2]

LOW | RESOLVED

In SecondaryCollateralizationBWT.sol there is a modifier:

```
modifier onlyGovernance () {
      require (_governanceAddress == msg.sender, "Caller should be pool");
      _;
}
```

Recommendation:
Replace "should be pool" with "should be governance".

Potentially misleading event

LOW | RESOLVED

In SecondaryCollateralizationBWT.sol file there are 2 functions (withdrawCollateral and withdraw). Despite different token, share the same “WithdrawLiquidity” event.

Recommendation:
There are several possible solutions, examples:

- Add additional event;
- Add variable to highlight token in current event;
- Remove event in withdrawCollateral.

Unexpected words order

INFORMATIONAL | UNRESOLVED

In SecondaryPoolBWT.sol file there are 11 occurrences of “...should be not...” expression, example: "WHITE token address should be not null".

I believe the correct word order is “...should not be...”.

Recommendation:
Replace "...should be not null" with "...should not be null" (and 10 other variations of this message).

Misleading comment

INFORMATIONAL | RESOLVED

In SecondaryPoolBWT.sol file there is:

```
// liquidity provider fee: initial 40% of total FEE
uint256 public _lpFee = 0.5 * 1e18;
// team fee: initial – 20% of total FEE
uint256 public _controllerFee = 0.1 * 1e18;
```

Recommendation:
Replace 40% with 50% and 20% with 10%.

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security team

As part of our work assisting Polars in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

Tests were based on the functionality of the code, as well as a review of the Polars contract requirements for details about issuance amounts and how the system handles these.

Code Coverage

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
SecondaryPoolBWT.sol	94.00	84.00	100.00	94.00	... 466, 469, 470
SecondaryCollateralizationBWT.sol	98.00	100.00	100.00	98.00	... 113, 130, 134
All files	**96.00**	**92.00**	**100.00**	**96.00**	

Test Results

```
Contract: SecondaryPool
  Init function
    ✓ Trying to Init function by passing controllerWalletAddress as zero (476ms)
    ✓ Trying to Init function by passing lpWalletAddress as zero (281ms)
    ✓ Trying to Init function by passing governanceWalletAddress as zero (265ms)
    ✓ Triggering the init function (1005ms)
    ✓ Trying to triggering Init function again which is started already (280ms)
  changeGovernanceAddress function
    ✓ Trying to trigger the changeGovernanceAddress function from Non-governance address (218ms)
    ✓ Trying to trigger the changeGovernanceAddress function with new governance
      address as zero (299ms)
    ✓ Changing the governance address (731ms)
  changePrimaryPoolAddress function
```

```
    ✓ Trying to trigger the changePrimaryPoolAddress function with new primary pool
      address as zero (252ms)
    ✓ Changing the PrimaryPool address (752ms)
  changeEventContractAddress function
    ✓ Trying to trigger the changeEventContractAddress function with new
      eventContractAddress as zero (206ms)
    ✓ Changing the event contract address (624ms)
  changeCollateralizationContractAddress function
    ✓ Trying to trigger the changePrimaryPoolAddress function with new primary pool
      address as zero (291ms)
    ✓ Changing the Collateralization Contract address (721ms)
  changeGovernanceWalletAddress function
    ✓ Trying to trigger the changeGovernanceWalletAddress function with new primary pool
      address as zero (266ms)
    ✓ Changing the GovernanceWallet address (749ms)
  addLIquidity
    ✓ addLiquidity by not having enough delegated BWT (508ms)
    ✓ addLiquidity with 300 tokens (761ms)
  buyBlack
    ✓ Trying to buy black token with destination address as zero (174ms)
    ✓ Trying to buy black token with NOT ENOUGH DELEGATED TOKENS (210ms)
    ✓ Trying to buy black token with maxprice greater than the token price (267ms)
    ✓ Trying to buy black token with not having enough delgated tokens (217ms)
    ✓ buy black tokens (927ms)
  sellBlack
    ✓ Trying to buy black token with destination address as zero (177ms)
    ✓ Trying to buy black token with NOT ENOUGH DELEGATED TOKENS (195ms)
    ✓ Trying to buyback black tokens more than sold in the pool (259ms)
    ✓ Trying to buyback black token with not having enough delgated tokens (306ms)
    ✓ Trying to buyback black token with minimum price greater than the price of the token (640ms)
    ✓ buyback black tokens (626ms)
  buyWhite
    ✓ Trying to buy white token with maxprice greater than the token price (299ms)
    ✓ buy white tokens (1028ms)
  sellWhite
    ✓ Trying to buyback white tokens more than sold in the pool (416ms)
    ✓ Trying to buyback white token with not having enough delgated tokens (248ms)
    ✓ Trying to buyback white token with minimum price greater than the price of the token (553ms)
    ✓ buyback white tokens (766ms)
  withdrawLIquidity
    ✓ Trying to withdraw liquidity without having enough delegated pool tokens
      on user balance (273ms)
    ✓ Trying to withdraw liquidity without having enough BLACK or WHITE tokens
      to withdraw (486ms)
    ✓ Trying to withdraw liquidity without having enough WHITE tokens on
      Collateralization contract balance (144ms)
    ✓ withdrawLIquidity (525ms)
  distributeProjectIncentives
    ✓ distributing project incentives (862ms)
  getBWBalances
    ✓ get the black and white tokens balances (182ms)
  submitEventStarted function
    ✓ Trying to trigger the submitEventStarted function from Non-contract address (235ms)
    ✓ Trying to trigger the submitEventStarted function with high event price
      change percent (338ms)
    ✓ Trying to trigger the submitEventStarted function with lower event price
      change percent (323ms)
    ✓ trigger the submitEventStarted function with appropriate price change (380ms)
  submitEventResult function
    ✓ Trying to submit event result with an inapropriate result value (291ms)
    ✓ Submit event result with result value as 0
    ✓ Submit event result with result value as white token won i.e 1
    ✓ Submit event result with result value as black token won i.e -1

Contract: SecondaryCollateralizationBWT
  addLiquidity
    ✓ addLiquidity by passing the destination address as 0x0 (338ms)
  withdraw
    ✓ Trying to withdraw by passing the destination address as 0x0 (301ms)
  withdrawCollateral
    ✓ Trying to withdrawCollateral by passing the destination address as 0x (234ms)
    ✓ Trying to withdrawCollateral by Not enough Collateral tokens on Collateralization
      contract balance (284ms)
  changePoolAddress function
    ✓ Trying to trigger the changePoolAddress function from Non-governance address (313ms)
    ✓ Trying to trigger the changePoolAddress function with new  pool address as zero (315ms)
    ✓ Changing the pool address (812ms)
  changeGovernanceAddress function
    ✓ Trying to trigger the changeGovernanceAddress function from Non-governance
      address (341ms)
    ✓ Trying to trigger the changeGovernanceAddress function with new governance
      address as zero (267ms)
    ✓ Changing the governance address (867ms)

59 passing (34s)
```

We are grateful to have been given the opportunity to work with the Polars team.

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.

Zokyo's Security Team recommends that the Polars team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.