SMART CONTRACT AUDIT

ZOKYO.

Apr 29, 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

TECHNICAL SUMMARY

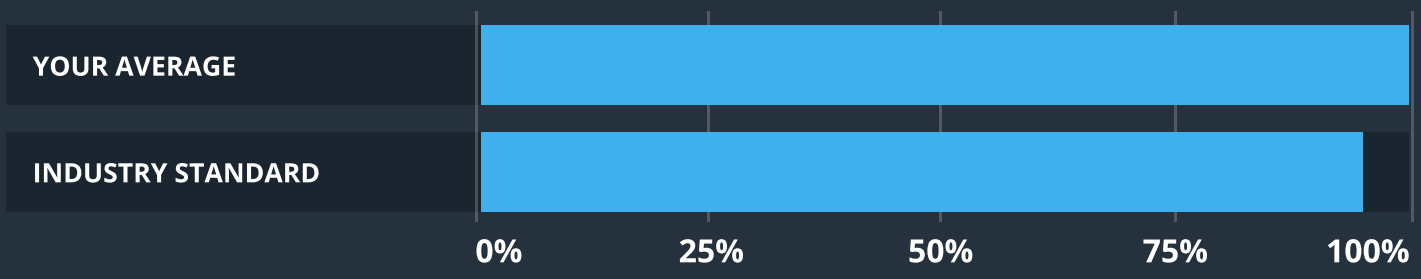
This document outlines the overall security of the ArGo smart contracts, evaluated by Zokyo's Blockchain Security team.

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

Contract Status

There were 2 critical issues found during the audit, which were successfully resolved by ArGo team.

Testable Code



The testable code is 100%, 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 ArGo 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 ArGo repository – https://github.com/argoapp-live/argo-contracts.
Last commit – 3d6cbf08bf6bf3a7bc6af88cb45e91dd9b341ee9.

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 ArGo 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.	3	Testing contract logic against common and uncommon attack vectors.
2	Cross-comparison with other, similar smart contracts by industry leaders.	4	Thorough, manual review of the codebase, line-by-line.

EXECUTIVE SUMMARY

There were 2 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.

Contracts are well written and structured. The findings during the audit have no impact on contract performance or security, so it is fully production-ready.

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

ARGO.sol

LOW | UNRESOLVED

Unused import (5th line).

ArgoTokenVesting.sol

CRITICAL | RESOLVED

Incorrect values for release times are set in the constructor since there is no binding to the current time, so the user can easily withdraw all funds at once.

```
61          _token = token_;
62          for (uint256 i = 0; i < releaseTime_.length; i++) {
63              vestPeriodInfoArray.push(
64                  VestPeriodInfo({
65                      percent: percent_[i],
66                      releaseTime: releaseTime_[i],
67                      released: false
68                  })
69              );
70          }
71          _beneficiary = beneficiary_;
```

Example Case:
Three values are transferred to the releaseTime_ array - 100, 1000, 10000 (in seconds). The current timestamp is 1,000,000 (there are even more on the mainnet). When creating the VestPeriodInfo structure, we must pass 1000000 + 100 in the releaseTime field, and so on. But in the code now just 100, 1000 ...

Example Case:
Use block.timestamp when creating an object of the VestPeriodInfo structure.

HIGH | RESOLVED

Low accuracy. After division, loss of accuracy is possible.

```
134                     amount =
135                         amount +
136                         vestPeriodInfoArray[i].percent.mul(totalBalance).div(
137                             100
138                         );
```

Recommendation:
Since there are no fractional numbers in solidity, one method is used. We can take any number as one hundred percent. For example, let's say we have 100% = 10 ^ 27, then 1% = 10 ^ 25.

Example:

```
175     uint256 constant PRECISION = 10 ** 25;
176     uint256 constant PERCENTAGE_100 = 100 * PRECISION;
177
178     amount = amount + vestPeriodInfoArray[i].percent.mul(PRECISION).mul(totalBalance).div(PERCENTAGE_100);
```

With this method, the accuracy of the calculations will be much more accurate.

MEDIUM | RESOLVED

In the constructor “require” is duplicated.

```
52          require(
53              beneficiary_ != address(0),
54              "ArgoTokenVesting: beneficiary address should not be zero address"
55          );
56          require(
57              beneficiary_ != address(0),
58              "ArgoTokenVesting: beneficiary address should not be zero address"
59          );
```

Recommendation:
Delete duplicates.

MEDIUM | RESOLVED

A lot of reading from the store. Inefficient use of gas.

```
125         for (uint256 i = 0; i < vestPeriodInfoArray.length; i++) {
126             console.log("In first loop");
127             console.log("releaseTime", vestPeriodInfoArray[i].releaseTime);
128             console.log("block.timestamp", block.timestamp);
129             if (vestPeriodInfoArray[i].releaseTime < block.timestamp) {
130                 console.log("First Condition passed");
131                 if (!vestPeriodInfoArray[i].released) {
132                     console.log("Second Condition passed");
133                     vestPeriodInfoArray[i].released = true;
134                     amount =
135                         amount +
136                         vestPeriodInfoArray[i].percent.mul(totalBalance).div(
137                             100
138                         );
139                     console.log(
140                         "New Amount added",
141                         vestPeriodInfoArray[i].percent.mul(totalBalance).div(
142                             100
143                         )
144                     );
145                 }
146             } else {
147                 break;
148             }
149         }
```

Recommendation:

Copy the structure to a local variable or copy only the most repetitive fields to variables.

MEDIUM | RESOLVED

Using + instead of add.

```
134                 amount =
135                     amount +
136                     vestPeriodInfoArray[i].percent.mul(totalBalance).div(
137                         100
138                     );
```

Recommendation:
Use the add method of the safeMath library, as in theory an overflow is possible.

LOW | RESOLVED

Variable totalBalance without an access modifier.

Recommendation:
You can make an access modifier public for convenience and to increase user trust.

LOW | RESOLVED

Now variable _setTotalCalled and function setTotalCalled are not used anywhere and are not needed.

LOW | RESOLVED

Since variable _totalBalance is now public, it doesn't need a getter. It is also customary to name public fields of contracts without an underscore.

LOW | RESOLVED

Since hardhat / console is no longer used, you need to remove the import.

LOW | RESOLVED

The logger is not related to the business logic. If possible, remove it as it increases the cost of the function itself.

Recommendation:
If it has no relation to the business logic, it's better to remove it in order to improve gas efficiency.

LOW | RESOLVED

Incorrect names for variables.

```
44        uint256[] memory releaseTime_,
45        uint256[] memory percent_
```

Recommendation:
Use plural names.

LOW | RESOLVED

Reading from the store at each iteration (Line 125). Inefficient use of gas.

```
119     /**
120      * @notice Transfers tokens held by timelock to beneficiary.
121      */
122     function release() public virtual {
123         // solhint-disable-next-line not-rely-on-time
124         uint256 amount;
125         for (uint256 i = 0; i < vestPeriodInfoArray.length; i++) {
126             console.log("In first loop");
127             console.log("releaseTime", vestPeriodInfoArray[i].releaseTime);
128             console.log("block.timestamp", block.timestamp);
```

Recommendation:
Move the length of the vestPeriodInfoArray array into a local variable.

LOW | RESOLVED

Functions are used instead of variables?

```
151
152         token().safeTransfer(beneficiary(), amount);
153     }
```

Recommendation:
Uuse variables directly.

ArgoVestingFactory.sol

CRITICAL | RESOLVED

EmergencyWithdraw must be with onlyOwner modifier.

HIGH | UNRESOLVED

There may be a problem that the sum of all percentages can be more than 100% or less than 100%. In the first case, an error will occur, since there will not be enough funds on the contract to pay. In the second case, tokens will remain on the contract, which then will not go anywhere.

```
39      constructor(
40          address _argoAddress,
41          address[] memory _addressList,
42          uint256[] memory _percentList,
43          uint256[] memory _epochToRelease,
44          uint256[] memory _amountList
45      ) {
```

Recommendation:
Create a function that will check that the amount of interest is equal to 100%.

MEDIUM | RESOLVED

The setTotalBalance function is superfluous.

```
116         );
117         IERC20(argoToken).transfer(
118             address(vesting),
119             whiteListedAddressMapping[msg.sender].amount
120         );
121
122         vesting.setTotalBalance();
123
124         emit AmountWithdrawn(
125             msg.sender,
126             address(vesting),
127             whiteListedAddressMapping[msg.sender].amount
128         );
129     }
```

Recommendation:
The amount can be passed directly to the ArgoTokenVesting constructor. Then ArgoTokenVesting does not need to be Ownable and has setTotalBalance method.

LOW | UNRESOLVED

There is an extra word 'Mapping' in the name of the map and it is better to make a plural.

```
33      //mapping of address of their vesting contract with their address
34      mapping(address => bool) public tokenVestingContractMappingStatus;
35
36      //mapping of whiteListed users
37      mapping(address => whiteListedAddressInfo) public whiteListedAddressMapping;
```

Recommendation:
For example, whiteListedAddressMapping => whiteListedAddresses.

LOW | UNRESOLVED

Lines 50 and 73 have an extra space after the word address.

LOW | RESOLVED

The name of the structure is whiteListedAddressInfo with a lowercase letter.

Recommendation:
Capitalize the letter.

LOW | RESOLVED

Inefficient use of gas. Many calls to the array object.

```
75        for (uint256 i = 0; i < _addressList.length; i++) {
76            if (!tokenVestingContractMappingStatus[_addressList[i]]) {
77                tokenVestingContractMappingStatus[_addressList[i]] = true;
78                whiteListedAddressMapping[_addressList[i]].amount = _amountList[
79                    i
80                ];
81            }
82
83            emit AddressWhitelisted(_addressList[i]);
84        }
```

Recommendation:
We can store _addressList [i] in a separate variable for optimization.

LOW | RESOLVED

SafeTransfer is not used here.

```
117       IERC20(argoToken).transfer(
118           address(vesting),
119           whiteListedAddressMapping[msg.sender].amount
120       );
```

Recommendation:
If safeTransfer is not needed here, please delete the library connection.

LOW | RESOLVED

Function emergencyWithdraw returns nothing.

Recommendation:
Remove returns from the function signature.

LOW | RESOLVED

The name of the withdraw function and the corresponding event is incorrect.

Recommendation:
Please use more appropriate names such as createVesting and vestingCreated.

LOW | RESOLVED

Incorrect name.

```
39      constructor(
40          address _argoAddress,
41          address[] memory _addressList,
42          uint256[] memory _percentList,
43          uint256[] memory _epochToRelease,
44          uint256[] memory _amountList
45      ) {
```

Recommendation:
Use the plural form.

LOW | RESOLVED

There is a lot of extra storage in the withdraw function.

```
92      function withdraw() public {
93          require(
94              tokenVestingContractMappingStatus[msg.sender],
95              "Address not whitelisted"
96          );
97          require(
98              !whiteListedAddressMapping[msg.sender].withdrawn,
99              "Amount already withdrawn by address"
100         );
101         require(
102             whiteListedAddressMapping[msg.sender].amount > 0,
103             "Withdraw amount is not set"
104         );
105         whiteListedAddressMapping[msg.sender].withdrawn = true;
106
107         ArgoTokenVesting vesting =
108             new ArgoTokenVesting(
109                 IERC20(argoToken),
110                 msg.sender,
111                 epochsToRelease,
112                 percentList
113             );
114         whiteListedAddressMapping[msg.sender].deployedVestingAddress = address(
115             vesting
116         );
117         IERC20(argoToken).transfer(
118             address(vesting),
119             whiteListedAddressMapping[msg.sender].amount
120         );
121
122         vesting.setTotalBalance();
123
124         emit AmountWithdrawn(
125             msg.sender,
126             address(vesting),
127             whiteListedAddressMapping[msg.sender].amount
128         );
129     }
130 }
```

Recommendation:
Save the most frequently used variables (or structure) to a local variable.

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Secured team

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

Contract: ARGO

ARGO Test Cases

- ✓ should deploy with correct name (133ms)
- ✓ should deploy with correct symbol (134ms)
- ✓ should deploy with correct decimals (121ms)
- ✓ should deploy with expected total supply (153ms)
- ✓ shouldn't deploy if cap exceeded (591ms)
- ✓ should burn tokens correct (322ms)
- ✓ shouldn't burn tokens if amount to burn exceeds balance (449ms)
- ✓ should mint tokens correct (372ms)
- ✓ shouldn't mint if cap exceeded (268ms)
- ✓ shouldn't mint to the zero address (219ms)
- ✓ should burn from recipient tokens correct (702ms)
- ✓ shouldn't burn from recipient tokens if amount exceeds allowance (650ms)

Contract: ARGOTokenVesting

ARGOTokenVesting Test Cases

- ✓ should deploy with correct beneficiary (278ms)
- ✓ shouldn't deploy if beneficiary is the zero address (408ms)
- ✓ shouldn't deploy if token is the zero address (417ms)
- ✓ shouldn't deploy if amount of release times values not equal to amount of percentages values (420ms)
- ✓ should deploy with correct releaseTime (669ms)
- ✓ should deploy with correct releasePercent (742ms)
- ✓ should deploy with correct total balance (326ms)
- ✓ should return deployed token address correct (288ms)
- ✓ shouldn't release if given release time not achieved (535ms)
- ✓ should release correct (1691ms)

Contract: ARGOVestingFactory

ARGOTokenVesting Test Cases

- ✓ shouldn't deploy if percentage list is empty (307ms)
- ✓ should't deploy if address List is empty (349ms)
- ✓ should't deploy if address List length not equal to percentage list (333ms)
- ✓ shouldn't deploy if percentage list length not equal to release epoch (385ms)
- ✓ should add addresses to white list correct (253ms)
- ✓ should't add addresses to white list if addresses length not equal to amount list length (252ms)
- ✓ should remove addresses from white list correct (919ms)
- ✓ shouldn't create vesting twice (1486ms)
- ✓ should't create vesting if amount not set (705ms)
- ✓ should create vesting correct (1713ms)
- ✓ should withdraw correct (576ms)

33 passing (18s)

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
contracts\	100.00	100.00	100.00	100.00	
ARGO.sol	100.00	100.00	100.00	100.00	
ArgoTokenVesting.sol	100.00	100.00	100.00	100.00	
ArgoVestingFactory.sol	100.00	100.00	100.00	100.00	
All files	**100.00**	**100.00**	**100.00**	**100.00**	

We are grateful to have been given the opportunity to work with the ArGo 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 ArGo team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.