QuillAudits

Audit Report
March, 2021

Contents

Introduction

During the period of February 25th, 2021 to March 9th, 2021 – QuillHash Team performed security audit for AVS Staking smart contract. The code for audit was taken from the following link:
https://etherscan.io/address/0x090e69e7F48AFC059e480F297042DEA396B971e7#code

Updated contract:
https://github.com/Rock-n-Block/AVS-staking-contract (commit - 72bb1d9)

Overview of AlgoVest

AlgoVest is a multi DeFi-utility and deflationary cryptocurrency that uses the Protect, Reward and Burn mechanism as Proof of Capital Protection (PoCP) and derives its value from an underlying community treasury fund powered by a disruptive artificial intelligence trading system that protects and grows investments

The features & utility of the AlgoVest growing ecosystem

- AVS is an ERC20 token built on Ethereum
- AI capital protection & consistent earnings
- Fixed limited supply, non-mintable
- Strategic periodic token buyback every other month
- Deflationary multi-DeFi-utility token
- AVS will be the token for the growing AlgoVest ecosystem
- Store on ERC20 compatible wallets
- AVS will be listed and tradable on DEX and CEX exchanges
- Staked AVS will receive up to 20% APY rewards, claimable monthly

AVS Staking Features:
The staking platform allows to stake as many times as possible on different days and APY levels.

- To activate your AVS staking user needs to approve and confirm the transaction.
- No minimum AVS staking requirement.

- No staking fees.
- Un-staking fee - 2% of your staking rewards when user un-stake at the end of days staked.
- Locking period is between 15–180 days.

Details: https://algovest.fi/

Scope of Audit

The scope of this audit was to analyse AlgoVest Staking smart contract codebase for quality, security, and correctness.

Check Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level
- Address hardcoded
- Transaction-Ordering
- Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Using delete for arrays
- Integer overflow/underflow
- Locked money
- Private modifier
- Revert/require functions
- Using var
- Visibility
- Using blockhash
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contracts care was taken to ensure:

- Overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per intended behavior mentioned in whitepaper.
- Implementation of BEP20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

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.

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.

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.

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

Every issue in this report has been assigned with a severity level. There are four levels of severity and each of them has been explained below.

High severity issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality and we recommend these issues to be fixed before moving to a live environment.

Medium level severity issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems and they should still be fixed.

Low level severity issues

Low level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity four issues which 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	2
Closed	0	1	9	0

Issues Found – Code Review / Manual Testing

High severity issues

None.

Medium severity issues

1. **Reentrancy**
   One of the major dangers of calling external contracts is that they can take over the control flow, and make changes to your data that the calling function wasn't expecting. This class of bug can take many forms, and both of the major bugs that led to the DAO's collapse were bugs of this sort. It's recommended that the calls to external functions/events should happen after any changes to state variables in your contract so your contract is not vulnerable to a re-entrancy exploit.
   When control is transferred to recipient, care must be taken to not create reentrancy vulnerabilities. Consider using ReentrancyGuard or the checks-effects-interactions pattern.

   In the following functions state variables are written or events are emitted after an external function call:
   a.AVSOwnerWithdraw(uint256) [#86-104]:

```
External calls:
- require(bool, string)(avsAddress.transfer(sender,amount),StakingAVS: Could not send AVS
tokens) [#97-100]

State variables written or Events emitted after the call(s):
- allAVSTokens = allAVSTokens.sub(amount) [#101]
- unfreezedAVSTokens = unfreezedAVSTokens.sub(amount) [#102]
- AVSTokenOutcome(sender,amount,_currentDay()) [#103]
```

Auditor Remarks: Fixed

b.stakeEnd(uint256, uint256) [#157-262]:

```
External calls:
- avsAddress.transfer(sender,avsTokensToReturn) [#183]

State variables written or events emitted after the call(s):
- _removeStake(stakeIndex,stakeId) [#196]
- stakeList[sender][stakeIndex] = stakeList[sender][stakeListLength - 1] [#416-418]
- stakeList[sender].pop() [#419]
- totalStakedAVS = totalStakedAVS.sub(st.stakedAVS) [#200]
- --totalStakers [#198]
- AVSTokenOutcome(sender,avsTokensToReturn - st.stakedAVS,currDay) [#184-188]
- StakeEnd(sender,st.stakeId,avsTokensToReturn - st.stakedAVS,servedNumOfDays,currDay)
[#189-195]
```

Auditor Remarks: Fixed

c.AVSTokenDonation(uint256) [#75-84]:

```
External calls:
- require(bool,string)(avsAddress.transferFrom(sender,address(this),amount), StakingAVS: Could
not get AVS tokens) [#77-80]

State variables written or events emitted after the call(s):
- allAVSTokens = allAVSTokens.add(amount) [#81]
- unfreezedAVSTokens = unfreezedAVSTokens.add(amount) [#82]
- AVSTokenIncome(sender,amount,_currentDay()) [#83]
```

Auditor Remarks: Fixed

d.stakeStart(uint256,uint256) [#106-155]:

```
External calls:
- require(bool,string)(avsAddress.transferFrom(sender,address(this),amount), StakingAVS: AVS
token transfer failed) [#117-120]

State variables written after the call(s):
- allStakes.push(st) [#142]
- freezedAVSTokens = freezedAVSTokens.add(avsEarnings - amount) [#130]
- st = StakeInfo(++ stakeIdLast,currDay,numDaysStake,amount,avsEarnings - amount) [#133-140
- stakeList[sender].push(st) [#141]
```

```
- totalStakedAVS = totalStakedAVS.add(amount) [#154]
- ++ totalStakers [#152]
- unfreezedAVSTokens = unfreezedAVSTokens.sub(avsEarnings - amount) [#129]
- AVSTokenIncome(sender,amount,currDay) [#122]
- StakeStart(sender,amount,avsEarnings - amount,numDaysStake,currDay,stakeIdLast) [#143-150
- TokenFreezed(sender,avsEarnings - amount,currDay) [#131]
```

Auditor Remarks: Fixed

Following link explains more about Reentrancy and ReentrancyGuard
https://docs.openzeppelin.com/contracts/2.x/api/utils#ReentrancyGuard
https://blog.openzeppelin.com/reentrancy-after-istanbul/

Low level severity issues

1. **Compiler version should be fixed**
   Solidity source files indicate the versions of the compiler they can be compiled with. It's recommended to lock the compiler version in code, as future compiler versions may handle certain language constructions in a way the developer did not foresee. Also, it's recommended to use latest compiler version.

   Auditor Remarks: fixed

2. **Coding Style Issues**
   Coding style issues influence code readability and in some cases may lead to bugs in future. Smart Contracts have naming convention, indentation and code layout issues. It's recommended to use Solidity Style Guide to fix all the issues. Consider following the Solidity guidelines on formatting the code and commenting for all the files. It can improve the overall code quality and readability.

```
stakin contract.sol
  398:2  error  Line length must be no more than 120 but current length is 131  max-line-length

✖ 1 problem (1 error, 0 warnings)
```

Auditor Remarks: Fixed

3. **Order of layout**

   The order of functions as well as the rest of the code layout does not follow solidity style guide.

   Layout contract elements in the following order:

   a.Pragma statements
   b.Import statements
   c.Interfaces
   d.Libraries
   e.Contracts

   Inside each contract, library or interface, use the following order:

   a.Type declarations
   b.State variables
   c.Events
   d.Functions

   Please read following documentation links to understand the correct order:

   - https://solidity.readthedocs.io/en/v0.5.3/style-guide.html#order-of-layout
   - https://solidity.readthedocs.io/en/v0.5.3/style-guide.html#order-of-functions

   Auditor Remarks: Fixed

4. **Function state mutability should be restricted to pure**

   The functions _getAVSEarnings(), _getAVSEarnings_pen() and min() are suppose to return same values no matter how many times they are called. If the function is declared as view it read from the state that may have changed since the last time it was called.

   Lines: #349, 379, 422.

   Auditor Remarks: Fixed

5. **Function names is not in mixedCase**

   Function names in solidity other than constructors should use mixedCase. Following function names do not follow naming convention:

   -AVSTokenDonation(uint256) **[#75-84]**
   -AVSOwnerWithdraw(uint256) **[#86-104)]**
   -_getAVSEarnings_pen(uint256, uint256) **[#379-400]**
   -length_stakes() **[#432-434]**
   -seven_days() **[#436-470]**

Auditor Remarks: Fixed

6. **Constant name is not in UPPER_CASE_WITH_UNDERSCORES**
   Constants should be named with all capital letters with underscores separating words. The constant maxNumDays **[#38]** is not in UPPER_CASE_WITH_UNDERSCORES

   Auditor Remarks: Fixed

7. **Complex calculations can be simplified by storing values in variables**
   To make code less error prone and maintain readability of code make use of variables to store calculations.
   Calculations like following should be simplified [#372, 394, 399]:

   -avsAmount + avsAmount.mul(perc).div(10000).mul(uint256(numOfDays)).div(uint256(365))
   -rew = avsAmount.mul(perc).div(10000).mul(uint256(numOfDays)).div(uint256(365))
   -return avsAmount + (rew * uint256(80)) / uint256(100);

   Auditor Remarks: Fixed

NEW ISSUES:

8. **Implicit Visibility**
   It's a good practice to explicitly define visibility of state variables, functions, interface functions and fallback functions. The default visibility of state variables – internal; function – public; interface function - external.
   Line 24: Visibility is missing
   mapping(address => bool) whitelist;

   Auditor Remarks: Fixed

9. **Use external function modifier instead of public**
   The public functions that are never called by contract should be declared external to save gas.
   In the previous version of staking contract function not used were declared – external. In the current version following should be declared external:
   addInWhitelist() and removeFromWhiteList

   Auditor Remarks: Fixed

Informational

1. **Approve function of ERC-20 is vulnerable**
   A security issue called “Multiple Withdrawal Attack” - originates from two methods in the ERC20 standard for approving and transferring tokens. The use of these functions in an adverse environment (e.g., front-running) could result
   in more tokens being spent than what was intended. This issue is still open on the GitHub and several solutions have been made to mitigate it.

   For more details on how to resolve the multiple withdrawal attack check the following document for details:

   You can visit this link.

2. **Naming convention issues**
   It is recommended to follow all solidity naming conventions to maintain readability of code.
   Details: https://docs.soliditylang.org/en/v0.4.25/style-guide.html#naming-conventions

Automated Testing

Slither

Slither is an open-source Solidity static analysis framework. This tool provides rich information about Binance Smart Chain smart contracts and has the critical properties. It runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to easily write custom analyses. Slither enables developers to find vulnerabilities, enhance their code comprehension, and quickly prototype custom analyses.

- Detects vulnerable Solidity code with low false positives
- Identifies where the error condition occurs in the source code
- Easily integrates into continuous integration and Truffle builds
- Built-in 'printers' quickly report crucial contract information
- Detector API to write custom analyses in Python
- Ability to analyze contracts written with Solidity >= 0.4
- Intermediate representation (SlithIR) enables simple, high-precision analyses
- Correctly parses 99.9% of all public Solidity code
- Average execution time of less than 1 second per contract

```
INFO:Printers:
Compiled with solc
Number of lines: 856 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 5 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 2
Number of informational issues: 14
Number of low issues: 16
Number of medium issues: 14
Number of high issues: 1
ERCs: ERC20

+-----------------+-------------+-------+--------------------+-------------+--------------------+
|       Name      | # functions | ERCS  |     ERC20 info     | Complex code |      Features      |
+-----------------+-------------+-------+--------------------+-------------+--------------------+
|      IERC20     |      6      | ERC20 |     No Minting     |      No      |                    |
|                 |             |       | Approve Race Cond. |              |                    |
|                 |             |       |                    |              |                    |
|     SafeMath    |      13     |       |                    |      No      |                    |
| AlgoVestStaking |      25     |       |                    |     Yes      | Tokens interaction |
+-----------------+-------------+-------+--------------------+-------------+--------------------+
INFO:Slither:stakin_contract.sol analyzed (5 contracts)
```

```
INFO:Detectors:
Pragma version>=0.6.0<0.8.0 (Context.sol#3) is too complex
solc-0.6.2 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Redundant expression "this (Context.sol#21)" inContext (Context.sol#15-25)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Pragma version>=0.6.0<0.8.0 (IERC20.sol#3) is too complex
solc-0.6.2 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Pragma version>=0.6.0<0.8.0 (Context.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (Ownable.sol#3) is too complex
solc-0.6.2 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Redundant expression "this (Context.sol#21)" inContext (Context.sol#15-25)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (Ownable.sol#56-59)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (Ownable.sol#65-69)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Detectors:
Pragma version>=0.6.0<0.8.0 (SafeMath.sol#3) is too complex
solc-0.6.2 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
AlgoVestStaking (stakin_contract.sol#13-472) contract sets array length with a user-controlled value:
        - stakeList[sender].push(st) (stakin_contract.sol#141)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#array-length-assignment
```

```
INFO:Detectors:
Reentrancy in AlgoVestStaking.AVSTokenDonation(uint256) (stakin_contract.sol#75-84):
        External calls:
        - require(bool,string)(avsAddress.transferFrom(sender,address(this),amount),StakingAVS: Could not get AVS tokens) (stakin_contract.sol#77-80)
        State variables written after the call(s):
        - allAVSTokens = allAVSTokens.add(amount) (stakin_contract.sol#81)
        - unfreezedAVSTokens = unfreezedAVSTokens.add(amount) (stakin_contract.sol#82)
Reentrancy in AlgoVestStaking.stakeEnd(uint256,uint256) (stakin_contract.sol#157-262):
        External calls:
        - avsAddress.transfer(sender,avsTokensToReturn) (stakin_contract.sol#183)
        State variables written after the call(s):
        - totalStakedAVS = totalStakedAVS.sub(st.stakedAVS) (stakin_contract.sol#200)
        - -- totalStakers (stakin_contract.sol#198)
Reentrancy in AlgoVestStaking.stakeEnd(uint256,uint256) (stakin_contract.sol#157-262):
        External calls:
        - avsAddress.transfer(sender,st.stakedAVS.add((avsTokensToReturn_scope_0.sub(st.stakedAVS)).mul(98).div(100))) (stakin_contract.sol#215-220)
        State variables written after the call(s):
        - totalStakedAVS = totalStakedAVS.sub(st.stakedAVS) (stakin_contract.sol#238)
        - -- totalStakers (stakin_contract.sol#236)
Reentrancy in AlgoVestStaking.stakeStart(uint256,uint256) (stakin_contract.sol#106-155):
        External calls:
        - require(bool,string)(avsAddress.transferFrom(sender,address(this),amount),StakingAVS: AVS token transfer failed) (stakin_contract.sol#117-120)
        State variables written after the call(s):
        - allStakes.push(st) (stakin_contract.sol#142)
        - freezedAVSTokens = freezedAVSTokens.add(avsEarnings - amount) (stakin_contract.sol#130)
        - st = StakeInfo(++ stakeIdLast,currDay,numDaysStake,amount,avsEarnings - amount) (stakin_contract.sol#133-140)
        - stakeList[sender].push(st) (stakin_contract.sol#141)
        - totalStakedAVS = totalStakedAVS.add(amount) (stakin_contract.sol#154)
        - ++ totalStakers (stakin_contract.sol#152)
        - unfreezedAVSTokens = unfreezedAVSTokens.sub(avsEarnings - amount) (stakin_contract.sol#129)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
```

```
INFO:Detectors:
Reentrancy in AlgoVestStaking.AVSOwnerWithdraw(uint256) (stakin_contract.sol#86-104):
        External calls:
        - require(bool,string)(avsAddress.transfer(sender,amount),StakingAVS: Could not send AVS tokens) (stakin_contract.sol#97-100)
        Event emitted after the call(s):
        - AVSTokenOutcome(sender,amount,_currentDay()) (stakin_contract.sol#103)
Reentrancy in AlgoVestStaking.AVSTokenDonation(uint256) (stakin_contract.sol#75-84):
        External calls:
        - require(bool,string)(avsAddress.transferFrom(sender,address(this),amount),StakingAVS: Could not get AVS tokens) (stakin_contract.sol#77-80)
        Event emitted after the call(s):
        - AVSTokenIncome(sender,amount,_currentDay()) (stakin_contract.sol#83)
Reentrancy in AlgoVestStaking.stakeEnd(uint256,uint256) (stakin_contract.sol#157-262):
        External calls:
        - avsAddress.transfer(sender,avsTokensToReturn) (stakin_contract.sol#183)
        Event emitted after the call(s):
        - AVSTokenOutcome(sender,avsTokensToReturn - st.stakedAVS,currDay) (stakin_contract.sol#184-188)
        - StakeEnd(sender,st.stakeId,avsTokensToReturn - st.stakedAVS,servedNumOfDays,currDay) (stakin_contract.sol#189-195)
Reentrancy in AlgoVestStaking.stakeEnd(uint256,uint256) (stakin_contract.sol#157-262):
        External calls:
        - avsAddress.transfer(sender,st.stakedAVS.add((avsTokensToReturn_scope_0.sub(st.stakedAVS)).mul(98).div(100))) (stakin_contract.sol#215-220)
        Event emitted after the call(s):
        - AVSTokenOutcome(sender,(avsTokensToReturn_scope_0.sub(st.stakedAVS)).mul(98).div(100),currDay) (stakin_contract.sol#221-225)
        - StakeEnd(sender,st.stakeId,avsTokensToReturn_scope_0 - st.stakedAVS,servedNumOfDays,currDay) (stakin_contract.sol#227-233)
Reentrancy in AlgoVestStaking.stakeStart(uint256,uint256) (stakin_contract.sol#106-155):
        External calls:
        - require(bool,string)(avsAddress.transferFrom(sender,address(this),amount),StakingAVS: AVS token transfer failed) (stakin_contract.sol#117-120)
        Event emitted after the call(s):
        - AVSTokenIncome(sender,amount,currDay) (stakin_contract.sol#122)
        - StakeStart(sender,amount,avsEarnings - amount,numDaysStake,currDay,stakeIdLast) (stakin_contract.sol#143-150)
        - TokenFreezed(sender,avsEarnings - amount,currDay) (stakin_contract.sol#131)
```

Slither didn't raise any critical issue with UNICORN contracts. The contract was well tested and all the minor issues that were raised have been documented in the report. All other vulnerabilities of importance have already been covered in the manual audit section of the report.

SmartCheck

SmartCheck is a tool for automated static analysis of Solidity source code for security vulnerabilities and best practices. SmartCheck translates Solidity source code into an XML-based intermediate representation and checks it against XPath patterns. SmartCheck shows significant improvements over existing alternatives in terms of false discovery rate (FDR) and false negative rate (FNR). It gave the following result for the Token and Vesting smart contract:

https://tool.smartdec.net/scan/939d96d8789a43368dfb5e2eb1a57eef

SmartCheck did not detect any high severity issue. All the considerable issues raised by SmartCheck are already covered in the code review section of this report.

Mythril

Mythril is a security analysis tool for EVM bytecode. It detects security vulnerabilities in smart contracts built for Ethereum. It uses symbolic execution, SMT solving and taint analysis to detect a variety of security vulnerabilities.
Mythril did not detect any high severity issue. All the considerable issues raised by Mythril are already covered in the code review section of this report.

Remix IDE

Remix is a powerful, open source tool that helps you write Solidity contracts straight from the browser. Written in JavaScript, Remix supports both usage in the browser and locally.

The staking smart contracts were tested for various compiler versions. The smart contract compiled without any error on explicitly setting compiler version **0.6.0>= and <0.6.12.**

It is recommended to use any fixed compiler versions from 0.6.0 to 0.6.12.
The contract failed to compile from version 0.7.0 onwards as few functionalities in solidity have been deprecated – for example now() is no longer supported from 0.7.0 onwards.

Closing Summary

Overall, the smart contracts are well written and adhere to ERC-20 guidelines. Several issues of medium and low severity were found during the audit. There were no critical or major issues found that can break the intended behaviour. Issues shared in Initial report were fixed and were reviewed by the auditor.

Disclaimer

QuillHash audit is not a security warranty, investment advice, or an endorsement of the AlgoVest 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 AlgoVest Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

hello@quillhash.com