QuillAudits

Audit Report
August, 2021

Seasonal Tokens

Contents

Scope of Audit

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

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- 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

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 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.
SmartCheck.

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

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	0	2
Closed	0	2	2	4

Introduction

During the period of **August 12, 2021 to August 19, 2021** - QuillAudits Team performed a security audit for SeasonalTokens smart contracts.

The code for the audit was taken from following the official link: https://github.com/seasonaltokens/seasonaltokens

Note	Date	Commit Hash
Version 1	17/08/2021	fe404847f3e60157b92a01b6ca613d17a49c0930
Version 2	19/08/2021	6d9fc8b6e28c4a7539c93b6fa7575b2e7168323c

Issues Found – Code Review / Manual Testing

High severity issues

No issues were found.

Medium severity issues

1. **Divisions performed before multiplication**

   Description
   Due to the fact that Solidity truncates when dividing, performing a division before multiplication can result in truncated amounts being amplified by future calculations. There are a few instances in the codebase where division is performed mid-calculation. For example in Autumn contract:
   L220: _miningTarget = (_miningTarget / 100) * 99
   L222: _miningTarget = (_miningTarget / 99) * 100
   L253: 2 * (2 ** 255 / miningTarget)

   Remediation
   Being mindful of overflows, consider changing the order of operations where possible such that truncating steps are performed last to minimize any unnecessary loss of precision. We recommend changing the order of the divisions in other contracts as well.

   Status: Fixed
   The issue was found fixed in Version 2.

2. Internal functions can be called

Description
approveAndCall() and approve() functions are still callable by the users even though the safeApproveAndCall() and safeApprove() are used.

This can pose the problem that the user can just ignore all the checks in the safeApproveAndCall() and safeApprove() functions, and proceed with the other ones.
Furthermore, it is best practice though to avoid them as they can:

- cause an increase in computations (and unnecessary gas consumption)
- indicate bugs or malformed data structures and they are generally a sign of poor code quality
- cause code noise and decrease readability of the code

Remediation
We recommend changing approveAndCall() and approve() functions to internal.

Status: Fixed
approveAndCall() has been changed to internal in Version 2.

Low level severity issues

3. Lack of error message for require() and revert() functions

Description
The require statements in transfer() and transferFrom() should include error messages that detail the cause of errors.

The error message should also be provided for the revert() statement at line 106.

Remediation
We recommend adding message errors for require() and revert() statements for the above-mentioned functions.

Status: Fixed
The issue was found fixed in Version 2.

4. Missing zero address validation

Description
We've detected missing zero address validation for the entire codebase, and in the transferOwnership() function of Owned contract.

Remediation
Consider implementing require statements where appropriate to validate all user-controlled input, including constructor, to avoid the potential for erroneous values to result in unexpected behaviors or wasted gas.

Status: Fixed
The issue was found fixed in Version 2.

Informational

5. Solidity Private Modifier Do Not Hide Data

To favor explicitness, consider everything that is inside a contract is visible to all observers external to the blockchain. Therefore, the Auditee should be aware that making something private only prevents other contracts from reading or modifying the information, but it will still be visible to the whole world outside of the blockchain.

For example, the following private variables in SeasonalTokens contract are still visible and accessible:

- bytes32 private challengeNumber;
- uint private miningTarget;
- uint public tokensMinted;

Status: Acknowledged by the Auditee

6. Public function that could be declared external

Description

Using the external attribute for functions never called from the contract. Therefore, the following public functions that are never called by the contract should be declared external to save gas:

- acceptOwnership()
- transferAnyERC20Token()
- safeApproveAndCall()
- safeApprove()

Status: Fixed

The issue was found fixed in Version 2.

7. Variables declared as uint instead of uint256

Description

To favor explicitness, consider changing all instances of uint into uint256 in the entire codebase.

Status: Fixed

The issue was found fixed in Version 2.

8. block.timestamp may not be reliable

Description

The Time contract uses the block.timestamp as part of the calculations and time checks.

Nevertheless, timestamps can be slightly altered by miners to favor them in contracts that have logics that depend strongly on them. Consider taking into account this issue and warning the users that such a scenario could happen.

Status: Acknowledged by the Auditee

9. Conformance to Solidity naming conventions

Description

Constants should be named with all capital letters with underscores separating words. Examples: MAX_BLOCKS, TOKEN_NAME, TOKEN_TICKER, CONTRACT_VERSION.

Therefore, we recommend re-naming the following constant variables:

- symbol to SYMBOL
- name to NAME

Status: Fixed

The issue was found fixed in Version 2.

10. Inconsistent coding style

Description

Deviations from the Solidity Style Guide were identified throughout the entire codebase. Taking into consideration how much value a consistent coding style adds to the project's readability, enforcing a standard coding style with the help of linter tools such as Solhint is recommended.

Status: Fixed

The issue was found fixed in Version 2.

Functional test

Function Names	Testing results
mint	Passed
mint	Passed
_numberOfRewardsAvailable	Passed
_numberOfRewardsToGive	Passed
_giveRewards	Passed
_setNextMaxNumberOfRewards	Passed
_getNewChallengeNumber	Passed
_scheduledNumberOfRewards	Passed
_adjustDifficulty	Passed
rewardEra	Passed
getAdjustmentInterval	Passed
getChallengeNumber	Passed
getMiningDifficulty	Passed
getMiningReward	Passed
_getMiningReward	Passed
getNumberOfRewardsAvailable	Passed
getRewardAmountForAchievingTarget	Passed
safeApprove	Passed
safeApproveAndCall	Passed

Automated Testing

Slither

```
INFO:Detectors:
AutumnToken._adjustDifficulty(uint256,uint256,uint256,uint256) (AutumnToken.sol#209-231) performs a multiplication on the result of a division:
        -_miningTarget = (_miningTarget / 100) * 99 (AutumnToken.sol#220)
AutumnToken._adjustDifficulty(uint256,uint256,uint256,uint256) (AutumnToken.sol#209-231) performs a multiplication on the result of a division:
        -_miningTarget = (_miningTarget / 99) * 100 (AutumnToken.sol#222)
AutumnToken.getMiningDifficulty() (AutumnToken.sol#252-254) performs a multiplication on the result of a division:
        -2 * (2 ** 255 / miningTarget) (AutumnToken.sol#253)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
AutumnToken.mint(uint256) (AutumnToken.sol#92-131) uses a dangerous strict equality:
        - singleRewardAmount == 0 (AutumnToken.sol#99)
AutumnToken.mint(uint256,bytes32) (AutumnToken.sol#184-190) uses a dangerous strict equality:
        - require(bool,string)(digest == _challenge_digest,Challenge digest does not match expected digest on token contract [ AbstractERC918.mint() ]) (AutumnToken.sol#187)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Owned.transferOwnership(address)._newOwner (Owned.sol#38) lacks a zero-check on :
                - newOwner = _newOwner (Owned.sol#40)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
AutumnToken.mint(uint256) (AutumnToken.sol#92-131) uses timestamp for comparisons
        Dangerous comparisons:
        - singleRewardAmount == 0 (AutumnToken.sol#99)
AutumnToken._numberOfRewardsAvailable(uint256,uint256,uint256) (AutumnToken.sol#133-147) uses timestamp for comparisons
        Dangerous comparisons:
        - intervalsSinceLastReward > numberAvailable (AutumnToken.sol#140)
        - numberAvailable > MAX_REWARDS_AVAILABLE (AutumnToken.sol#143)
AutumnToken.mint(uint256,bytes32) (AutumnToken.sol#184-190) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(digest == _challenge_digest,Challenge digest does not match expected digest on token contract [ AbstractERC918.mint() ]) (AutumnToken.sol#187)
AutumnToken._adjustDifficulty(uint256,uint256,uint256,uint256) (AutumnToken.sol#209-231) uses timestamp for comparisons
        Dangerous comparisons:
        - timeSinceLastReward * 88 < rewardsGivenNow * REWARD_INTERVAL * 61 (AutumnToken.sol#219)
AutumnToken.rewardEra(uint256) (AutumnToken.sol#234-242) uses timestamp for comparisons
        Dangerous comparisons:
        - timeSinceContractCreation < DURATION_OF_FIRST_ERA (AutumnToken.sol#238)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Pragma version0.8.5 (ApproveAndCallFallBack.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.5 (AutumnToken.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.5 (ERC20.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.5 (ERC918.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.5 (Owned.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.5 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
INFO:Detectors:
Pragma version0.8.5 (ApproveAndCallFallBack.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.5 (AutumnToken.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.5 (ERC20.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.5 (ERC918.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.5 (Owned.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.5 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter AutumnToken.mint(uint256,bytes32)._nonce (AutumnToken.sol#184) is not in mixedCase
Parameter AutumnToken.mint(uint256,bytes32)._challenge_digest (AutumnToken.sol#184) is not in mixedCase
Parameter AutumnToken.rewardEra(uint256)._time (AutumnToken.sol#234) is not in mixedCase
Parameter Owned.transferOwnership(address)._newOwner (Owned.sol#38) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
receiveApproval(address,uint256,address,bytes) should be declared external:
        - ApproveAndCallFallBack.receiveApproval(address,uint256,address,bytes) (ApproveAndCallFallBack.sol#16)
mint(uint256,bytes32) should be declared external:
        - AutumnToken.mint(uint256,bytes32) (AutumnToken.sol#184-190)
getAdjustmentInterval() should be declared external:
        - AutumnToken.getAdjustmentInterval() (AutumnToken.sol#244-246)
getChallengeNumber() should be declared external:
        - AutumnToken.getChallengeNumber() (AutumnToken.sol#248-250)
getMiningDifficulty() should be declared external:
        - AutumnToken.getMiningDifficulty() (AutumnToken.sol#252-254)
getMiningTarget() should be declared external:
        - AutumnToken.getMiningTarget() (AutumnToken.sol#256-258)
getMiningReward() should be declared external:
        - AutumnToken.getMiningReward() (AutumnToken.sol#260-265)
getNumberOfRewardsAvailable(uint256) should be declared external:
        - AutumnToken.getNumberOfRewardsAvailable(uint256) (AutumnToken.sol#271-275)
getRewardAmountForAchievingTarget(uint256,uint256) should be declared external:
        - AutumnToken.getRewardAmountForAchievingTarget(uint256,uint256) (AutumnToken.sol#277-283)
decimals() should be declared external:
        - AutumnToken.decimals() (AutumnToken.sol#285-287)
totalSupply() should be declared external:
        - AutumnToken.totalSupply() (AutumnToken.sol#289-292)
        - ERC20Interface.totalSupply() (ERC20.sol#15)
balanceOf(address) should be declared external:
        - AutumnToken.balanceOf(address) (AutumnToken.sol#301-305)
        - ERC20Interface.balanceOf(address) (ERC20.sol#17)
transfer(address,uint256) should be declared external:
        - AutumnToken.transfer(address,uint256) (AutumnToken.sol#319-331)
        - ERC20Interface.transfer(address,uint256) (ERC20.sol#21)
safeApprove(address,uint256,uint256) should be declared external:
        - AutumnToken.safeApprove(address,uint256,uint256) (AutumnToken.sol#371-377)
transferFrom(address,address,uint256) should be declared external:
        - AutumnToken.transferFrom(address,address,uint256) (AutumnToken.sol#399-413)
        - ERC20Interface.transferFrom(address,address,uint256) (ERC20.sol#25)
```

```
        - ERC20Interface.totalSupply() (ERC20.sol#15)
balanceOf(address) should be declared external:
        - AutumnToken.balanceOf(address) (AutumnToken.sol#301-305)
        - ERC20Interface.balanceOf(address) (ERC20.sol#17)
transfer(address,uint256) should be declared external:
        - AutumnToken.transfer(address,uint256) (AutumnToken.sol#319-331)
        - ERC20Interface.transfer(address,uint256) (ERC20.sol#21)
safeApprove(address,uint256,uint256) should be declared external:
        - AutumnToken.safeApprove(address,uint256,uint256) (AutumnToken.sol#371-377)
transferFrom(address,address,uint256) should be declared external:
        - AutumnToken.transferFrom(address,address,uint256) (AutumnToken.sol#399-413)
        - ERC20Interface.transferFrom(address,address,uint256) (ERC20.sol#25)
allowance(address,address) should be declared external:
        - AutumnToken.allowance(address,address) (AutumnToken.sol#425-429)
        - ERC20Interface.allowance(address,address) (ERC20.sol#19)
safeApproveAndCall(address,uint256,uint256,bytes) should be declared external:
        - AutumnToken.safeApproveAndCall(address,uint256,uint256,bytes) (AutumnToken.sol#468-475)
transferAnyERC20Token(address,uint256) should be declared external:
        - AutumnToken.transferAnyERC20Token(address,uint256) (AutumnToken.sol#484-488)
transferOwnership(address) should be declared external:
        - Owned.transferOwnership(address) (Owned.sol#38-42)
acceptOwnership() should be declared external:
        - Owned.acceptOwnership() (Owned.sol#44-54)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Mythril

```
enderphan@enderphan contracts_slither % myth a --rpc 127.0.0.1:8545 -a 0xe792E99B66C0260bFe1CB633c66a4358c30f5Cd3
^C^C==== Potential denial-of-service if block gas limit is reached ====
SWC ID: 128
Severity: Low
Contract: 0xe792E99B66C0260bFe1CB633c66a4358c30f5Cd3
Function name: gasprice_bit_ether(int128)
PC address: 9889
Estimated Gas Usage: 1810 - 2285
Potential denial-of-service if block gas limit is reached.
A storage modification is executed in a loop. Be aware that the transaction may fail to execute if the loop is unbounded and the necessary gas exceeds the block gas limit.
--------------------
Initial State:

Account: [ATTACKER], balance: 0x0, nonce:0, storage:{}
Account: [SOMEGUY], balance: 0x80020000000000, nonce:0, storage:{}

Transaction Sequence:

Caller: [SOMEGUY], function: approve(address,uint256), txdata: 0x095ea7b300000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000, value: 0x0
Caller: [SOMEGUY], function: gasprice_bit_ether(int128), txdata: 0x23b872dd000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001010101010101010101010101010
101010101010000000000000000000000000000000000000000000000000000000000, value: 0x0
```

SOLHINT LINTER

```
AutumnToken.sol
    2:1    error    Compiler version 0.8.5 does not satisfy the ^0.5.8 semver requirement           compiler-version
   37:5    warning  Constant name must be in capitalized SNAKE_CASE                                  const-name-snakecase
   39:5    warning  Constant name must be in capitalized SNAKE_CASE                                  const-name-snakecase
   78:5    warning  Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)  func-visibility
   82:32   warning  Avoid to make time-based decisions in your business logic                        not-rely-on-time
   83:31   warning  Avoid to make time-based decisions in your business logic                        not-rely-on-time
  106:46   warning  Provide an error message for revert                                              reason-string
  112:61   warning  Avoid to make time-based decisions in your business logic                        not-rely-on-time
  120:65   warning  Avoid to make time-based decisions in your business logic                        not-rely-on-time
  125:31   warning  Avoid to make time-based decisions in your business logic                        not-rely-on-time
  127:76   warning  Avoid to make time-based decisions in your business logic                        not-rely-on-time
  184:35   warning  Variable name must be in mixedCase                                               var-name-mixedcase
  187:9    warning  Error message for require is too long                                            reason-string
  321:9    warning  Provide an error message for require                                             reason-string
  373:9    warning  Error message for require is too long                                            reason-string
  401:9    warning  Provide an error message for require                                             reason-string
  471:9    warning  Error message for require is too long                                            reason-string
```

Results

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Disclaimer

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

Closing Summary

In this report, we have considered the security of the SeasonalTokens platform. We performed our audit according to the procedure described above.

The audit showed several medium, low, and informational severity issues. In the end, the majority of the issues were partially fixed or acknowledged by the Auditee.

Seasonal Tokens

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com