HaggleX

Smart Contract Audit Report

March 19, 2021

IMMUNE BYTES

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Introduction

1. About HaggleX

HaggleX has come to offer an all in one service where users can trade, exchange their coins, stake, lend, buy and sell cryptocurrency/Giftcards with our secured escrow service and also pay for utility services using HaggleX Wallet (platform). These added services bring more significant advantage to HaggleX, giving the exchange an edge when compared to existing crypto exchange services.

Visit https://www.hagglex.com/ to know more about.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provide professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 15+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

Visit http://immunebytes.com/ to know more about the services.

Documentation Details

HaggleX team has provided documentation for the purpose of conducting the audit. The documents are:

1. https://docs.google.com/document/d/1CBpXXaRayakNDFH7OEYW2vRbKhaJA90xcSzkrvmc9KA/edit?usp=drivesdk
2. https://ico.hagglex.com/docs/hagglex-whitepaper.pdf

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

IMMUNE BYTES

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: HaggleX
- Languages: Solidity(Smart contract)
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck
- Github commits hash/Smart Contract Address for audit: 9c2d7821bf5740afc4437c9a2bd3ace7ec883137

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level References

Every issue in this report was assigned a severity level from the following:

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	1	1	3
Closed	-	-	-

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

High severity issues

1. In function staking, line number 1076 gives control to the user to set the array length of variable **_staking[_msgSender()][]**. This is critical because the function **unstake** loops over this array length to delete an index from **_staking**. In case the array length is large enough, the function unstake can easily run out of gas and there is no way the array length can be decreased in any way externally.

```
1067    function stake(uint amount_, uint stakeType_) external {
1068        _burn(_msgSender(), amount_);
1069        stakedSupply += amount_;
1070        Stake memory temp;
1071        temp.amount = amount_;
1072        temp.startTime = block.timestamp;
1073        temp.stakeType = stakeType_;
1074        temp.lastWithdrawTime = block.timestamp;
1075        temp.noOfWithdrawals = 0;
1076        _staking[_msgSender()].push(temp);
1077        _stakingOptions[stakeType_].totalStaked += amount_;
1078        emit staked(_msgSender(), amount_, _stakingOptions[stakeType_].lockedTime);
1079    }
```

We recommend that it is ensured that there is some limit on the length of the array so that unstake can never run out of gas. Another alternative could be to not implement the **_removeIndexInArray** function to delete the index and make sure that the functions in the contract using the _staking variable are aware that an index is deleted. One way this can be done is by maintaining a mapping in place of the array, maintaining a latest stake index mapping for all addresses and adding another variable to the Stake struct to maintain the unstake or deleted status. There can be other ways to handle this as well.

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Medium severity issues

1. Account addresses should not be non-updateable as this could address recovery problems. If the private key is lost for any of the addresses, the address stored in the contract cannot be updated to a new address. This could cause funds to eventually go to addresses without a private key which means it will be lost forever.

```
696     address private CORE_TEAM = 0x821e90F8a572B62604877A2d5aa740Da0abDa07F;
697     address private ADVISORS = 0x7610269058d82eC1F2E66a0ff981056622D387F6;
698     address private CORE_INVESTORS = 0x811a622EB3e2a1a0Af8aE1a9AAaE8D1DC6016534;
699     address private RESERVE = 0x609B1D6C5E6E2B48bfCe70eD7f88bAA3ECB9ca9d;
700     address private CHARITY = 0x76a6a41282434a784e88Afc91Bd3E552A6F560f1;
701     address private FOUNDING_STAFF = 0xAbE2526c2F8117B081817273d56fa09c4bcBDc49;
702     address private AIRGRAB = 0xF985905b1510d51fd3da563D530263481F7c2a18;
703
704     address private LEADERSHIP_BOARD = 0x7B9A1CF604396160F543f0FFaE50E076e15f39c8;
705     address private UNIVERSAL_BASIC_INCOME = 0x9E7481AeD2585eC09066B8923570C49A38E06eAF;
706     address private DEVELOPMENT = 0xaC92741D5BcDA49Ce0FF35a3D5be710bA870B260;
```

We recommend all account addresses to be updateable by the owner or some other recovery strategy should be kept in mind to make sure funds never go to a lost address.

Low severity issues

1. The function **mintToMultipleAddresses** takes in an array of addresses as input but takes a single amount parameter to mint the same amount to all the provided addresses. This makes the bulk mint function less flexible.

   We recommend also making the amount parameter as an array and adding a require statement that ensures that the length of addresses and amounts arrays are the same. If this is an intended behavior then this issue can be ignored.

2. There are multiple instances in the contract where the array length is calculated inside the for loop definition. As an example:

```
883     function mintToMultipleAddresses(address[] memory _addresses, uint _amount) external
        onlyOwner {
884         for(uint i = 0; i < _addresses.length; i++){
885             _mint(_addresses[i],  _amount);
886         }
887     }
```

   In such cases, address.length is called each time the for loop is executed which uses a lot of gas.

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

We recommend storing the array length in a variable before the for loop and use that variable in the for loop definition. This ensures array length is calculated only once which saves gas.

The following functions can be improved:

- **mintToMultipleAddresses**
- **_removeIndexInArray**

3. We recommend making variables **STAKERS_PERCENTAGE, LEADERSHIP_BOARD_PERCENTAGE,UNIVERSAL_BASIC_INCOME_PERCENTAGE, DEVELOPMENT_PERCENTAGE** constants if they won't be updated later. This optimises gas costs on deployment and variable usage.

Recommendations

1. Follow solidity style guide for better readability: https://docs.soliditylang.org/en/v0.7.5/style-guide.html.
   For example, Functions should be grouped according to their visibility and ordered:
   - constructor
   - receive function (if exists)
   - fallback function (if exists)
   - external
   - public
   - internal
   - private
   - Within a grouping, place the view and pure functions last.

2. Add Natspecs comments to all functions for better understanding regarding what the parameters mean and what the function does. Function description is explained nicely in the code but it is missing parameter descriptions which is as important.

3. The following functions are missing a zero check for addresses in parameters. Although these are onlyOwner functions and can be taken care of by the owner but we recommend that they are added:
   - Address ***newOwner*** in function ***transferOwnership*** *on line* ***275***
   - Address ***minter_*** in function ***setMinter*** *on line* ***816***

4. We recommend avoiding comparing variables to a boolean constant. This is unnecessary and will save gas costs.

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Examples:
Line **513:**
"***require(_paused == false, "HaggleX Token: token transfer is unavailable");***"
can be replaced with
"***require(!_paused, "HaggleX Token: token transfer is unavailable");***"

5. We recommend avoiding using literals with too many digits as it is hard to read and could easily be mistaken. For example,
   - **_mint(CORE_TEAM,100000000000000000000000)**
   - **getTotalSupply() >= 100000000000000000000000 && getTotalSupply() <= 111680000000000000000000**

   We recommend using Ether suffix.

Unit Test

No unit tests were provided by the HaggleX team.

Recommendation:
Our team suggests that the developers should write extensive test cases for the contracts.

Coverage Report

Coverage report cannot be generated without unit test cases.

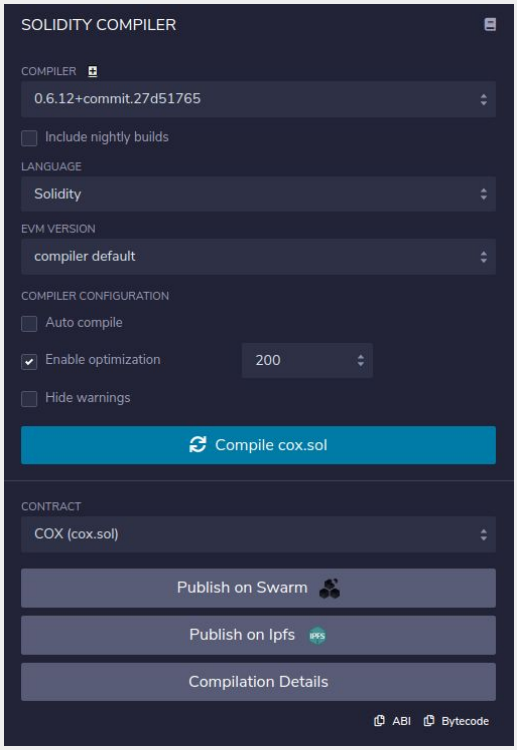
Recommendation:
We recommend 100% line and branch coverage for unit test cases.

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Automated Auditing

Remix Compiler Warnings

No errors were thrown by the compiler:



Contract Library

Contract-library contains the most complete, high-level decompiled representation of all Ethereum smart contracts, with security analysis applied to these in real time.

We performed analysis using contract Library on the kovan address of the HAG contract: 0xe48e2041eabc90eff82d1e5509b22a23663d1870

Analysis summary can be accessed here:
https://contract-library.com/contracts/Kovan/0xe48e2041eabc90eff82d1e5509b22a23663d1870
It did not return any issue during the analysis.

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Slither

Slither, an open-source static analysis framework. This tool provides rich information about Ethereum smart contracts and has the critical properties. While Slither is built as a security-oriented static analysis framework, it is also used to enhance the user's understanding of smart contracts, assist in code reviews, and detect missing optimizations.

```
 INFO:Detectors:
HaggleXToken (HaggleToken.sol#679-1110) contract sets array length with a user-controlled value:
	- _staking[_msgSender()].push(temp) (HaggleToken.sol#1076)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#array-length-assignment
INFO:Detectors:
HaggleXToken.withdrawStakeReward(uint256) (HaggleToken.sol#1030-1037) uses a dangerous strict equality:
	- require(bool,string)(isStakeLocked(stake_) == true,Withdrawal no longer available, you can only
Unstake now!) (HaggleToken.sol#1031)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
HaggleXToken.stake(uint256,uint256).temp (HaggleToken.sol#1070) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
Owned.transferOwnership(address).newOwner (HaggleToken.sol#275) lacks a zero-check on :
		- _newOwner = newOwner (HaggleToken.sol#276)
HaggleXToken.setMinter(address).minter_ (HaggleToken.sol#816) lacks a zero-check on :
		- _minter = minter_ (HaggleToken.sol#817)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
HaggleXToken.isStakeLocked(uint256) (HaggleToken.sol#895-898) uses timestamp for comparisons
	Dangerous comparisons:
	- stakingTime < _stakingOptions[_staking[_msgSender()][stake_].stakeType].lockedTime
(HaggleToken.sol#897)
HaggleXToken.getRemainingLockTime(uint256) (HaggleToken.sol#903-910) uses timestamp for comparisons
	Dangerous comparisons:
	- stakingTime < _stakingOptions[_staking[_msgSender()][stake_].stakeType].lockedTime
(HaggleToken.sol#905)
HaggleXToken.withdrawStakeReward(uint256) (HaggleToken.sol#1030-1037) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(isStakeLocked(stake_) == true,Withdrawal no longer available, you can only
Unstake now!) (HaggleToken.sol#1031)
	- require(bool,string)(block.timestamp >= _staking[_msgSender()][stake_].lastWithdrawTime + 600,Not
yet time to withdraw reward) (HaggleToken.sol#1032)
HaggleXToken.withdrawLeadershipBoardReward() (HaggleToken.sol#1039-1045) uses timestamp for
comparisons
	Dangerous comparisons:
	- require(bool,string)(block.timestamp >= lastWithdrawTime + 600,Not yet time to withdraw Leadership
Board reward) (HaggleToken.sol#1041)
HaggleXToken.withdrawUBIReward() (HaggleToken.sol#1047-1053) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(block.timestamp >= lastWithdrawTime + 600,Not yet time to withdraw Leadership
Board reward) (HaggleToken.sol#1049)
HaggleXToken.withdrawDevelopmentReward() (HaggleToken.sol#1055-1061) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(block.timestamp >= lastWithdrawTime + 600,Not yet time to withdraw Leadership
Board reward) (HaggleToken.sol#1057)
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

```
HaggleXToken.unstake(uint256) (HaggleToken.sol#1087-1095) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(isStakeLocked(stake_) != true,HaggleX Token:Stake still locked!)
(HaggleToken.sol#1088)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
ERC20._pause() (HaggleToken.sol#512-515) compares to a boolean constant:
        -require(bool,string)(_paused == false,HaggleX Token: token transfer is unavailable)
(HaggleToken.sol#513)
ERC20._unpause() (HaggleToken.sol#520-523) compares to a boolean constant:
        -require(bool,string)(_paused == true,HaggleX Token: token transfer is available) (HaggleToken.sol#521)
ERC20._blacklist(address) (HaggleToken.sol#528-531) compares to a boolean constant:
        -require(bool,string)(_blacklists[_address] == false,HaggleX Token: account already blacklisted)
(HaggleToken.sol#529)
ERC20._whitelist(address) (HaggleToken.sol#536-539) compares to a boolean constant:
        -require(bool,string)(_blacklists[_address] == true,HaggleX Token: account already whitelisted)
(HaggleToken.sol#537)
ERC20._transfer(address,address,uint256) (HaggleToken.sol#558-571) compares to a boolean constant:
        -require(bool,string)(_paused == false,HaggleX Token: token contract is not available)
(HaggleToken.sol#561)
ERC20._transfer(address,address,uint256) (HaggleToken.sol#558-571) compares to a boolean constant:
        -require(bool,string)(_blacklists[sender] == false,HaggleX Token: sender account already blacklisted)
(HaggleToken.sol#562)
ERC20._transfer(address,address,uint256) (HaggleToken.sol#558-571) compares to a boolean constant:
        -require(bool,string)(_blacklists[recipient] == false,HaggleX Token: sender account already blacklisted)
(HaggleToken.sol#563)
ERC20._mint(address,uint256) (HaggleToken.sol#582-594) compares to a boolean constant:
        -require(bool,string)(_paused == false,HaggleX Token: token contract is not available)
(HaggleToken.sol#584)
ERC20._mint(address,uint256) (HaggleToken.sol#582-594) compares to a boolean constant:
        -require(bool,string)(_blacklists[account] == false,HaggleX Token: account to mint to already blacklisted)
(HaggleToken.sol#585)
ERC20._burn(address,uint256) (HaggleToken.sol#607-619) compares to a boolean constant:
        -require(bool,string)(_blacklists[account] == false,HaggleX Token: account to burn from already
blacklisted) (HaggleToken.sol#610)
ERC20._burn(address,uint256) (HaggleToken.sol#607-619) compares to a boolean constant:
        -require(bool,string)(_paused == false,HaggleX Token: token contract is not available)
(HaggleToken.sol#609)
ERC20._approve(address,address,uint256) (HaggleToken.sol#635-646) compares to a boolean constant:
        -require(bool,string)(_paused == false,HaggleX Token: token contract approve is not available)
(HaggleToken.sol#638)
ERC20._approve(address,address,uint256) (HaggleToken.sol#635-646) compares to a boolean constant:
        -require(bool,string)(_blacklists[owner] == false,HaggleX Token: owner account already blacklisted)
(HaggleToken.sol#639)
ERC20._approve(address,address,uint256) (HaggleToken.sol#635-646) compares to a boolean constant:
        -require(bool,string)(_blacklists[spender] == false,HaggleX Token: spender account already blacklisted)
(HaggleToken.sol#640)
HaggleXToken.withdrawStakeReward(uint256) (HaggleToken.sol#1030-1037) compares to a boolean constant:
        -require(bool,string)(isStakeLocked(stake_) == true,Withdrawal no longer available, you can only Unstake
now!) (HaggleToken.sol#1031)
HaggleXToken.unstake(uint256) (HaggleToken.sol#1087-1095) compares to a boolean constant:
        -require(bool,string)(isStakeLocked(stake_) != true,HaggleX Token:Stake still locked!)
(HaggleToken.sol#1088)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Pragma version0.6.12 (HaggleToken.sol#3) necessitates a version too recent to be trusted. Consider deploying
with 0.6.11
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

```
solc-0.6.12 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Variable Owned._owner (HaggleToken.sol#265) is not in mixedCase
Variable Owned._newOwner (HaggleToken.sol#266) is not in mixedCase
Parameter ERC20.blacklisted(address)._address (HaggleToken.sol#415) is not in mixedCase
Event HaggleXTokenstaked(address,uint256,uint256) (HaggleToken.sol#682) is not in CapWords
Event HaggleXTokenunstaked(address,uint256) (HaggleToken.sol#683) is not in CapWords
Parameter HaggleXToken.blacklist(address)._address (HaggleToken.sol#837) is not in mixedCase
Parameter HaggleXToken.whitelist(address)._address (HaggleToken.sol#844) is not in mixedCase
Parameter HaggleXToken.mintToMultipleAddresses(address[],uint256)._addresses (HaggleToken.sol#883) is not
in mixedCase
Parameter HaggleXToken.mintToMultipleAddresses(address[],uint256)._amount (HaggleToken.sol#883) is not in
mixedCase
Variable HaggleXToken.STAKERS_PERCENTAGE (HaggleToken.sol#690) is not in mixedCase
Variable HaggleXToken.LEADERSHIP_BOARD_PERCENTAGE (HaggleToken.sol#691) is not in mixedCase
Variable HaggleXToken.UNIVERSAL_BASIC_INCOME_PERCENTAGE (HaggleToken.sol#692) is not in
mixedCase
Variable HaggleXToken.DEVELOPMENT_PERCENTAGE (HaggleToken.sol#693) is not in mixedCase
Variable HaggleXToken.CORE_TEAM (HaggleToken.sol#696) is not in mixedCase
Variable HaggleXToken.ADVISORS (HaggleToken.sol#697) is not in mixedCase
Variable HaggleXToken.CORE_INVESTORS (HaggleToken.sol#698) is not in mixedCase
Variable HaggleXToken.RESERVE (HaggleToken.sol#699) is not in mixedCase
Variable HaggleXToken.CHARITY (HaggleToken.sol#700) is not in mixedCase
Variable HaggleXToken.FOUNDING_STAFF (HaggleToken.sol#701) is not in mixedCase
Variable HaggleXToken.AIRGRAB (HaggleToken.sol#702) is not in mixedCase
Variable HaggleXToken.LEADERSHIP_BOARD (HaggleToken.sol#704) is not in mixedCase
Variable HaggleXToken.UNIVERSAL_BASIC_INCOME (HaggleToken.sol#705) is not in mixedCase
Variable HaggleXToken.DEVELOPMENT (HaggleToken.sol#706) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (HaggleToken.sol#21)" inContext (HaggleToken.sol#15-24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
HaggleXToken.constructor() (HaggleToken.sol#727-768) uses literals with too many digits:
        - _mint(CORE_TEAM,100000000000000000000000) (HaggleToken.sol#760)
HaggleXToken.constructor() (HaggleToken.sol#727-768) uses literals with too many digits:
        - _mint(RESERVE,100000000000000000000000) (HaggleToken.sol#763)
HaggleXToken.constructor() (HaggleToken.sol#727-768) uses literals with too many digits:
        - _mint(AIRGRAB,100000000000000000000000) (HaggleToken.sol#766)
HaggleXToken.getHalvedReward() (HaggleToken.sol#970-1011) uses literals with too many digits:
        - getTotalSupply() >= 100000000000000000000000 && getTotalSupply() <=
1116800000000000000000000 (HaggleToken.sol#974)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
HaggleXToken.ADVISORS (HaggleToken.sol#697) should be constant
HaggleXToken.AIRGRAB (HaggleToken.sol#702) should be constant
HaggleXToken.CHARITY (HaggleToken.sol#700) should be constant
HaggleXToken.CORE_INVESTORS (HaggleToken.sol#698) should be constant
HaggleXToken.CORE_TEAM (HaggleToken.sol#696) should be constant
HaggleXToken.DEVELOPMENT (HaggleToken.sol#706) should be constant
HaggleXToken.DEVELOPMENT_PERCENTAGE (HaggleToken.sol#693) should be constant
HaggleXToken.FOUNDING_STAFF (HaggleToken.sol#701) should be constant
HaggleXToken.LEADERSHIP_BOARD (HaggleToken.sol#704) should be constant
HaggleXToken.LEADERSHIP_BOARD_PERCENTAGE (HaggleToken.sol#691) should be constant
HaggleXToken.RESERVE (HaggleToken.sol#699) should be constant
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

```
HaggleXToken.STAKERS_PERCENTAGE (HaggleToken.sol#690) should be constant

HaggleXToken.UNIVERSAL_BASIC_INCOME (HaggleToken.sol#705) should be constant
HaggleXToken.UNIVERSAL_BASIC_INCOME_PERCENTAGE (HaggleToken.sol#692) should be constant
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Slither:./HaggleToken.sol analyzed (6 contracts with 72 detectors), 72 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither raised one high severity issue related to user controlled array length assignment which has already been covered in the Manual report. The medium issues raised were not related and the low level, informational issues have been covered in the manual report.

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Concluding Remarks

While conducting the audits of HaggleX smart contract, it was observed that the contracts contain High, Medium, and Low severity issues, along with several areas of recommendations.

Our auditors suggest that High, Medium, Low severity issues should be resolved by HaggleX developers. Resolving the areas of recommendations are up to the team's discretion. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the HaggleX platform or its product neither this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the one audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes Pvt Ltd.