Ethereum STYK

Smart Contract Audit Report

IMMUNE BYTES

Audits

December 28, 2020

Introduction

1. About Ethereum STYK

Ethereum STYK is an Intelligent smart contract. The name STYK is a play on words as the objective of the smart contract is to own a stake (portion) of the smart contract and to stake (staking) the % you hold in the contract. The entire contract works on the percentage (%) each user owns of the contract and NOT how many tokens each user owns.

This smart contract is:

- Verified & Fully decentralized with NO administrator keys (Zero human intervention)
- Crypto compliant (Based on current law for decentralized smart contracts)
- Halal investment compliant (no gambling, no borrowing, no lending, no interest, clarity on how dividends are derived, giving back some profits to the poor)
- Halal investment allows an additional 20% of the world's population to participate
- Increase volatility = more dividends
- Five layers of dividends and rewards
- Fair team based referral model - Monthly additional rewards/dividends
- Less stagnation
- More liquidity
- Inflation management to allow for more participation

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.

IMMUNE BYTES

Documentation Details

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

1. STYK 2 – Premine Smart Contract 3(.docx)

Audit Process & Methodology

ImmueBytes 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: Ethereum STKY
- Languages: Solidity(Smart contract), Javascript(Unit Testing)
- Github commit hash for audit: 8162f6b4f909489f95d436576f98ef222376c8c8
- Kovan Testnet address: 0xbfae05757d8401268039027e216bb516eccf8d41

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	7	3	6
Closed	-	-	-

High severity issues

1. **Incorrect calculation of user's token percentage in multiple functions.**
   In the **STYK** smart contract, there are multiple instances where the user's token balance percentage is calculated with respect to the total supply of STYK. The implementation of these calculations are found to be incorrect at **lines 517, 660, 725**.
   The calculation for user's token percentage is calculate as **(totalSupply * 100) / userBalance** which is incorrect. The correct implementation should be **(userBalance * 100) / totalSupply**. Since these calculation errors are present in crucial functions of smart contract they can have a major impact on the contract's internal ledger for user's funds.

   Recommendation:
   Consider correcting the implementation at all the mentioned places.

2. **Re-entrancy in STYKRewardsPayOuts() function.**
   In the **STYKRewardsPayOuts**() function of **STYK** contract, an external call for ETH **transfer** is performed to distribute rewards to users. However the function's implementation is incorrect and opens up a possible attack vector for **reentrancy** attack. The contract updates its internal balance records after the external ETH transfer call **(Lines 587 - 591)** which is inappropriate. All smart contracts must follow the Checks-Effects-Interactions patterns and must update its internal records prior to any external call. Due to this attack vector funds can be drained from STYK smart contract.

   More details on reentrancy attack can be found here https://docs.soliditylang.org/en/v0.6.12/security-considerations.html#re-entrancy

   Recommendation:
   Consider following the Checks-Effects-Interactions pattern, i.e, all internal states must be updated before any external call (ETH transfer) to avoid reentrancy.

3. **monthlyRewardsPayOuts**() **function can be called infinite times by a user.**
   The **monthlyRewardsPayOuts**() function is intended to distribute monthly rewards to users, however this function does not track whether a user has already claimed the monthly reward or not. Due to this the function can be called infinite times by any user who is eligible for monthly rewards until the contract gets drained out of ETH.

Recommendation:
Consider keeping the track of users who have already claimed the monthly reward for a respective month. State variables should be used to store this data.

4. **Re-entrancy in earlyAdopterBonus() function.**
   In the **earlyAdopterBonus**() function of STYK contract, an external call for ETH **transfer** is performed to distribute ETH to users. However the function's implementation is incorrect and opens up a possible attack vector for **reentrancy** attack. The contract updates its internal records after the external ETH transfer call **(Lines 729 - 731)** which is inappropriate. All smart contracts must follow the Checks-Effects-Interactions patterns and must update its internal records prior to any external call. Due to this attack vector funds can be drained from STYK smart contract.

   More details on reentrancy attack can be found here https://docs.soliditylang.org/en/v0.6.12/security-considerations.html#re-entrancy

   Recommendation:
   Consider following the Checks-Effects-Interactions pattern, i.e, all internal states must be updated before any external call (ETH transfer) to avoid reentrancy.

5. **The setStakingRequirement() function is open for anyone to call.**
   Anyone who can interact with Ethereum blockchain can call **setStakingRequirement()** and can set any **uint** value as stakingRequirement. stakingRequirement variable is used in purchaseTokens() function and can have a huge impact on the contract's working.

   Recommendation:
   Consider making the **setStakingRequirement()** function admin protected so that only whitelisted users/admins can change the **stakingRequirement**.

6. **The setAuctionExpiryTime() function is open for anyone to call.**
   Anyone who can interact with Ethereum blockchain can call **setAuctionExpiryTime()** and can set any **uint** value as **auctionExpiryTime**.The **stakingRequirement** variable is used in **purchaseTokens**() function and can have a huge impact on the contract's working.

Recommendation:
Consider making the **setAuctionExpiryTime()** function admin protected so that only whitelisted users/admins can change the **auctionExpiryTime**.

7. **setInflationTime(), setStakeTime**() and **setAuctionEthLimit**() **functions are open for anyone to call.**
   Any Ethereum user can call these functions and can set any **uint** value as **inflationTime**, **stakeTime** and **auctionEthLimit** respectively.These variables are used at multiple places inside the STYK smart contract and can have a huge impact on the contract's working.

   Recommendation:
   Consider making the mentioned functions admin protected so that only whitelisted users/admins can change the crucial state variables.

Medium severity issues

1. **Anyone can change the token name and symbol.**
   Any Ethereum user can call **setName**() and **setSymbol**() functions and can set any string value as **name** and **symbol** of the STYK token.The **name** and **symbol** of any token on ethereum is expected to be a **constant** by the project's users and community.

   Recommendation:
   Consider removing the functions to update the **name** and **symbol** of the token.

2. **Looping through unbounded arrays.**
   The **STYKRewardsPayOuts**() function of **STYK** contract loops through an unbounded array **userAddress** twice which can become an issue when the number of elements in the array becomes large. As the size of the array increases the function call will use more gas and after a certain point the transactions can start reverting due to ethereum **block gas limit.**

   Recommendation:
   Consider implementing the smart contract logic in such a way that looping through unbounded arrays is not needed.

3. **STYK token do not follow the ERC20 standard.**
   The implementation of **STYK** token differs majorly from the standard **ERC20** token interface. The STYK token does not implement all the functions specified in the ERC20 token standard, specifically it lacks the **approve**(), **transferFrom**() and **allowance**() function. Not following the ERC20 standard can hinder the interaction of STYK token with other Ethereum projects.
   More details on ERC20 token specification can be found here https://eips.ethereum.org/EIPS/eip-20.

Low severity issues

1. **In PriceConsumerV3 AggregatorV3Interface address is hardcoded to a kovan address.**
   The **constructor**() of **PriceConsumerV3** contains a hardcoded address for **AggregatorV3Interface** contract which should be manually changed for every ethereum network deployment. This opens up a window for human error by the developer/deployer of the contract. Ideally the address parameter should be given as a constructor parameter at the time of contract deployment.

2. **Inconsistent use of balanceOf() and tokenBalanceLedger_.**
   The **balanceOf**() function and **tokenBalanceLedger_** mapping in the STYK contract returns the STYK token balance of a user. However both of them are used inside the smart contract at various places, while both the methods return the same result it is still recommended to only use one of them throughout the contract to maintain the consistency. This will result in less confusion for other developers/auditors interacting with the STYK token.

3. **Compiler warning for No receive() function.**
   From Solidity 0.6.0 onwards the **unnamed fallback**() function is broken up into two functions, **fallback**() and **receive**(). While the **fallback**() function is implemented in the STYK contract it still lacks the **receive**() function. The solidity compiler also shows a warning for the same.
   More details about the breaking change can be found here https://docs.soliditylang.org/en/v0.7.4/060-breaking-changes.html#semantic-and-syntactic-changes

   Recommendation:
   Since the STYK token expects ETH to be transferred to the contract it is recommended to explicitly provide a **receive**() function with the **fallback**().

4. **Events should be emitted in consistent syntax.**
   In the STYK contract there are various events that are emitted in different functions. The contract mostly uses the **emit** keyword for emitting events however at **Line 287** it uses the older syntax for emitting the **Transfer** event. It is always recommended to use consistent coding styles throughout the contract/project.

5. **Spelling mistake for an external function.**
   The **calculateInfaltionHours**() function at **Line 508** of STYK contract has a spelling error. Instead of **inflation** it spells as **inflation**. Since this is an external function it will be accessed by off chain elements like frontend and users.

Unit Test

The unit tests provided by the STYK team are failing.

```
Contract: STYK
  1) Should deploy smart contract properly
  > No events were emitted
  [Testcase 1: To purchase the token]
    ✓ Token Buy (203ms)
  [Testcase 2: To check for monthly rewards]
    ✓ Token Buy (768ms)
  [Testcase 3: To check for non - qualifying monthly rewards]
    ✓ Token Buy (607ms)
  [Testcase 4: To send payouts to early adopter that qualify]
    ✓ Token Buy (541ms)
  [Testcase 5: To check for user not qualifying for early adopter bonus]
    ✓ Token Buy (469ms)


5 passing (6s)
1 failing

1) Contract: STYK
     Should deploy smart contract properly:
   Error: STYK has not been deployed to detected network (network/artifact mismatch)
    at Object.checkNetworkArtifactMatch (node_modules/truffle/build/webpack:/packages/contract/lib/utils/index.js:245:1)
    at Function.deployed (node_modules/truffle/build/webpack:/packages/contract/lib/contract/constructorMethods.js:85:1)
    at processTicksAndRejections (internal/process/task_queues.js:93:5)
    at Context.<anonymous> (test/STYKTest.js:27:24)
```

Coverage Report

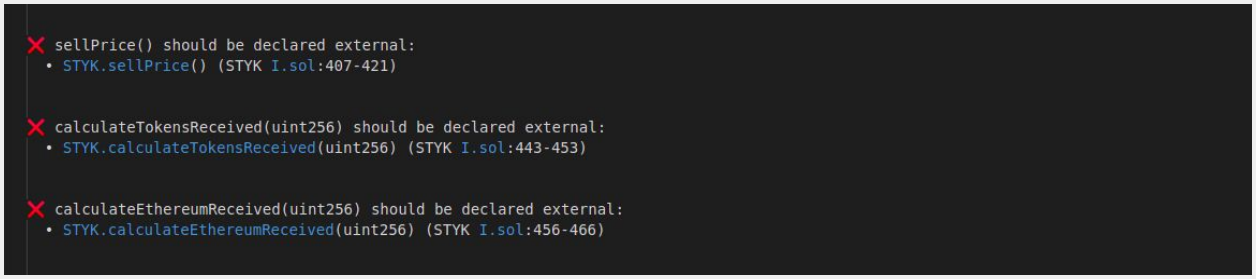
Test coverage of smart contracts is **not** 100%.

```
----------------------------|----------|----------|----------|----------|----------------|
File                        |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
----------------------------|----------|----------|----------|----------|----------------|
 contracts/                 |     43.7 |    33.78 |    40.32 |    43.75 |                |
  AggregatorV3Interface.sol |      100 |      100 |      100 |      100 |                |
  PriceConsumerV3.sol       |    33.33 |      100 |       50 |    33.33 |          59,66 |
  STYK I.sol                |    40.55 |    31.25 |    33.96 |    40.83 |... 672,689,715 |
  SafeMath.sol              |    83.33 |       50 |    85.71 |    78.95 | 66,112,113,116 |
----------------------------|----------|----------|----------|----------|----------------|
All files                   |     43.7 |    33.78 |    40.32 |    43.75 |                |
----------------------------|----------|----------|----------|----------|----------------|

> Istanbul reports written to ./coverage/ and ./coverage.json
> solidity-coverage cleaning up, shutting down ganache server
Error: ❌ 1 test(s) failed under coverage.
    at plugin (/home/akshay/workspace/gigs/STYK/STYK/node_modules/solidity-coverage/plugins/truffle.plugin.js:121:27)
    at runMicrotasks (<anonymous>)
    at processTicksAndRejections (internal/process/task_queues.js:93:5)
Truffle v5.1.58 (core: 5.1.58)
Node v14.15.1
```

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

Slither Tool Result



Recommendations

1. **The contract should follow the proper Solidity Style Guide.**
   Consider following the Solidity guide. It is intended to provide coding/naming convention for writing solidity code.

   Recommendation:
   Solidity style guide can be found using the following link https://docs.soliditylang.org/en/v0.6.12/style-guide.html

Concluding Remarks

While conducting the audits of Ethereum STYK 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 Ethereum STYK developers. Resolving the areas of recommendations are up to Ethereum STYK'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 Ethereum STKY 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.