Sheesha Finance

Smart Contract Audit Report

IMMUNE BYTES

Audits

Sheesha FINANCE

March 18, 2021

Introduction

1. About Sheesha Finance

The team at Sheesha Finance is inspired to bring a powerful DeFi platform to the industry, one that is trusted, well-funded, and heavily supported by a strong community. Sheesha is here for the long haul and, as such, will be rolling out our implementation in phases to ensure everything is correct and safe for their users!

Participating in Sheesha Finance is a straightforward approach known as a liquidity generation event. This event allows anyone to participate by contributing ETH/BNB and receiving a portion of Liquidity Provision (LP) tokens in the process. These tokens can be staked and should be for a variety of benefits.

Visit https://sheesha.finance/ to know more

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

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

1. Sheesha Finance - Smart Chain.docx
2. https://sheeshafinance.medium.com/

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: Sheesha Finance
- Languages: Solidity(Smart contract)
- Github Link/Smart Contract Address for audit: 36adeff12204228c1473b00199c1a35bfb61e943
- Deployed Smart Contract Address (kovan):
  - SHEESHAGlobals - 0x75b7524c355F67cff7D16e51f0C5CdF26bBBb2fe
  - LGE - 0xc78d2255e9cf5952C40e16C3B3D2e02BABeB0a18
  - SHEESHAVault - 0xB0C32604Ea44b814Fff891D943e787c5B04488fD
  - SHEESHAVaultLP - 0x8F46f6E392d27a4b2d2C98004335299d33f7DB55

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:

Admin/Owner Privileges can be misused either intentionally or unintentionally.

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	Admin/Owner Privileges	High	Medium	Low
Open	2	-	5	11
Closed	-	-	-	-

Admin/Owner Privileges

The **admin/owner** of **Sheesha Finance** smart contracts has various privileges over the smart contracts. These privileges can be misused either intentionally or unintentionally (in case admin's private key gets hacked). We assume that these extra rights will always be used appropriately. Some of these admin rights are listed below.

1. **All address variables in SHEESHAGlobals can be changed.**
   In **SHEESHAGlobals.sol,** admin/owner has the right to change all the state variable using the **initialize**(), **setSheeshaToken**(), **setSheeshaVaultAddress**() and **setSheeshaVaultLPAddress**() functions. Any ethereum address can be set as the **SHEESHATokenAddress**, **SHEESHAGlobalsAddress**, **SHEESHAVaultAddress**, **SHEESHAVaultLPAddress**, **SHEESHAWETHUniPair** and **UniswapFactory**.

2. **Admin has the right to change the reward rate for staked tokens.**
   In the **SheeshaVault** and **SheeshaVaultLP** contract, the admin has the right to change the **allocPoint** (reward amount per block) of any type of staked token using the **set**() function. Any **uint256** amount can be set as the **allocPoint**.

   Recommendation:
   Consider hardcoding predefined ranges or validations for variables. Also consider adding some governance for admin rights for smart contracts or use a multi-sig wallet as admin/owner address.

Medium severity issues

1. **transferVaultRewards() and transferVaultLPewards() can be called with any address as input.**
   In **SHEESHA.sol**, **transferVaultRewards() and transferVaultLPewards()** are admin protected functions. These functions are intended to send **SHEESHA** tokens to the respected Vaults. However these functions take an address variable as input, to which the **SHEESHA** tokens are sent. Any address can be given as input to these functions which creates room for human error. Also there is no logic present to restrict the calling of these functions only once, hence the functions can be called multiple times.

   Recommendation:
   Consider adding some validations in the mentioned functions so that they can only be called only once with a predefined address as input..

2. **Funds can be drained by emergencyDrain24hAfterLiquidityGenerationEventIsDone() function.**
   In **SHEESHA.sol**, **emergencyDrain24hAfterLiquidityGenerationEventIsDone()** is defined as an admin protected function which is intended to pull out funds from the contract in case of a bug/attack. In case where the admin's private key gets compromised, this function can be used to transfer all ETH and SHEESHA tokens from the **LGE** contract to the admin wallet.

3. **PoolInfo.lastRewardBlock is not updated in updatePool() function.**
   In the **SheeshaVault** and **SheeshaVaultLP** contract, **PoolInfo.lastRewardBlock** is intended to store the block number when the Pool's state was updated. However **lastRewardBlock** is only updated when the Pool is initialised for the first time and when the contract's pool token balance is zero. The **lastRewardBlock** should be updated with every deposit, claim and withdraw transaction which is currently not happening. This can lead to serious miscalculations in the contract's state/ledger.

   Recommendation:
   Consider updating the state of **PoolInfo.lastRewardBlock** at the end of **updatePool**() function.

4. **isActive() function always returns *false* value.**
   In **SheeshaVaultLP** contract, **Userinfo.status** stores whether a user has interacted with the contract previously or not. This value is read by **isActive()** function. However no logic is present inside the smart contract which updates the value of **Userinfo.status**. Since this value is never updated, the **isActive**() function will always return a false value.

   Recommendation:
   Consider adding the logic to update the **Userinfo.status** value in the smart contract.

5. **Unwanted updation of UserInfo.checkpoint variable.**
   In **SheeshaVaultLP** contract, there are certain scenarios when **UserInfo.checkpoint** for a user gets updated unintentionally.
   Since the **isActive**() always returns false value, whenever a user deposits some lpTokens more than once then his **checkpoint** gets updated.
   The **deposit**() function can also be used for claiming the staking rewards (when the input amount is provided as zero). In this case as well the **checkpoint** value gets updated.

   Recommendation:
   Consider implementing a more robust logic to update the **UserInfo.checkpoint** value.

Low severity issues

1. **Updation of userCount and userList are not handled properly.**
   In the **addLiquidity**() function of LGE smart contract, **userCount uint** variable **and userList** mapping are updated whenever a user adds ETH liquidity to the contract. However, if a user adds liquidity more than once, the userCount variable will be incremented and the user's address will get stored with more than one **uint** key in **userCount** mapping. So the scenario of addition of liquidity more than once is not handled properly. As the **userCount** and **userList** variables are not used inside the smart contract for any crucial purpose this issue won't impact the working of the contract.

   Recommendation:
   Consider handling the scenario of addition of liquidity more than once by one user.

2. **Redundant variables stored in SHEESHA smart contract.**
   The **SHEESHA** smart contract stores these state variables **devAddress, teamAddress, marketingAddress** and **reserveAddress**, these values are never used inside the smart contract. The storing of unnecessary data in Ethereum smart contracts are never recommended.

   Recommendation:
   Consider removing the mentioned state variables if not necessary.

3. **SHEESHA contract should be marked as abstract.**
   Since the **SHEESHA** smart contract is not meant to be deployed on its own and is intended to be inherited by **LGE** smart contract, it should be marked as **abstract**. This will remove the possibility of **SHEESHA** contract deployment by mistake.

4. **Unnecessary calculations in updatePool() function increases the transaction's gas cost.**
   The **updatePool**() function in the **SHEESHAVault** contract performs the following calculations -
   tokenRewards.mul(sheeshaPerBlock).div(percentageDivider)
   Since all of these three values are hardcoded in the contract itself, the end result of the above statement is also known. The repeated execution of this statement with every transaction increases the transaction cost which can be avoided. Similar is the case with **updatePool**() function in **SHEESHAVaultLP** contract.

Recommendation:
Consider hardcoding the end result of the calculation instead of computing it every time or at least do not use SafeMath for those arithmetic operations (the values will not overflow).

5. **createUniswapPairMainnet() function should be declared as *internal.***
   The **createUniswapPairMainnet**() function in **LGE** smart contract is intended to be called only once by the **constructor**() at the time of deployment. The function also has necessary checks which prevents its execution after a successful deployment. So the function cannot be called externally but still is marked is **external**

   Recommendation:
   Consider declaring the functions as **internal** instead of **external**.

6. **Approval is only needed once in the claimAndStakeLP() function*.***
   At **Line 161** of **LGE.sol**, the contract approves the **SHEESHAVaultLP** contract to move the max uint amount of LP tokens. Currently this approval is done with every user's transaction which is not necessary. The contract runs totally fine if the approval is only done once. The repeated approval also increases the transaction cost.

   Recommendation:
   Consider doing the approval transaction only when needed.

7. **Unnecessary condition implemented in add() functions*.***
   In the **add**() function of **SHEESHVault** and **SHEESHVaultLP** contract, this statement gets executed
   uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
   Since the **startBlock** is set to **block.timestamp** in the **constructor**(), the **lastRewardBlock** is always set to current **block.timestamp** (since current block timestamp will always be greater than startBlock).
   So the ternary operation can be avoided which will save some gas cost as well.

   Recommendation:
   Consider directly assigning the **block.timestamp** value to **lastRewardBlock**.

8. ***SHEESHAGlobals* contract should inherit the *ISHEESHAGlobals* interface.**
   Since all the **SHEESHA** contract fetch other contract's address using the **SHEESHAGlobals** instance on **ISHEESHAGlobals** interface, it is recommended that

SHEESHAGlobals contract should inherit the **ISHEESHAGlobals** interface so that the contract interface and contract implementation can align with each other.

9. **Spelling mistakes in function names.**
   The **getElpasedMonth**() in **SheeshaVaultLP** contract and **transferVaultLPewards**() in **SHEESHA** contract are spelled incorrectly.

   Recommendation:
   Consider correcting the mentioned function names.

10. **Unused variable - stakeCount**
    The **stakeCount** variable in the **LGE** contract is never used. Unused state variables increase the size of deployed bytecode.

    Recommendation:
    Consider removing the unused variables.

11. **Spelling mistake in contract's documentation.**
    There are spelling mistakes at **Line 91, 126** and **134** of **LGE** contract and **Line 196** of **SheeshaVaultLP** contract.
    Misspelled words and misleading documentation creates confusion in other developers/auditors minds. Consider avoiding them.

Unit Test

No unit tests were provided by the Sheesha Finance 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.

Concluding Remarks

While conducting the audits of Sheesha Finance smart contracts, it was observed that the contracts contain Admin/Owner Privileges, Medium, and Low severity issues, along with a few areas of recommendations.

Our auditors suggest that Medium, Low severity issues should be resolved by Sheesha Finance developers. Resolving the Admin/Owner Privileges and areas of recommendations are up to Sheesha Finance'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 Sheesha Finance platform or its product neither this audit is investment advice.
Notes:

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

ImmuneBytes Pvt Ltd.