SuperLoyal

Smart Contract Audit
Final Report

January 12, 2023

Introduction

1. About SuperLoyal

The mission at Superloyal is to bring more power, control, and revenue back to brands. Retaining customers and earning their loyalty is more important than ever. But earning loyalty requires more than giving away points for every purchase. Instead of rewarding buying behavior, Superloyal changes it through high-definition buying insights, emotional connections, and brand collaboration.

Visit https://superloyal.com/ to know more about it.

2. About ImmuneBytes

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

The team has secured 205+ blockchain projects by providing security services on different frameworks. The ImmuneBytes team helps start-ups with detailed system analysis, ensuring security and managing the overall project.

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

Documentation Details

The team has provided the following doc for audit:

1. https://github.com/superloyalcom/contracts#readme

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 to find 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 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, including -

1. Structural analysis of the smart contract is checked and verified.
2. An extensive automated testing of all the contracts under scope is conducted.
3. Line-by-line Manual Code review is conducted to evaluate, analyze and identify the potential security risks in the contract.
4. Evaluation of the contract's intended behavior and the documentation shared is imperative to verify the contract behaves as expected.
5. For complex and heavy contracts, adequate integration testing is conducted to ensure that contracts interact acceptably.
6. Storage layout verifications in the upgradeable contract are a must.
7. An important step in the audit procedure is highlighting and recommending better gas optimization techniques in the contract.

Audit Details

- Project Name: SuperLoyal
- Languages: Solidity(Smart contract), Typescript (Unit Testing)
- Smart Contracts: StakingMintableToken, FixSupplyToken
- GitHub Link: https://github.com/superloyalcom/contracts
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

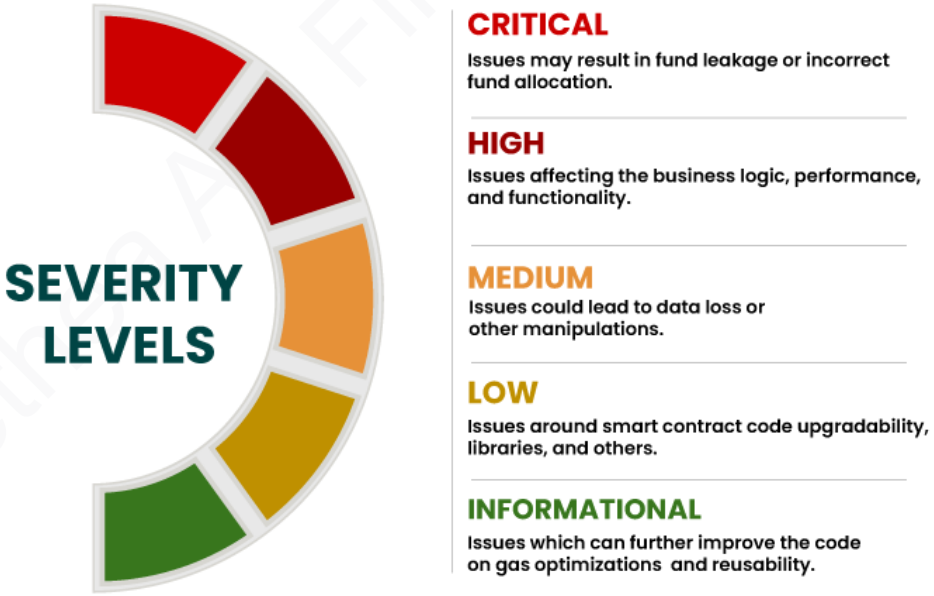
Audit Goals

The audit's focus 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 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 Reference

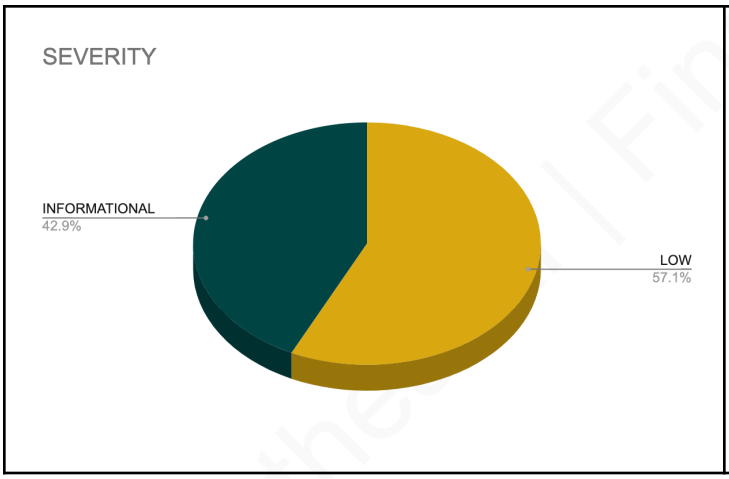
Every issue in this report were assigned a severity level from the following:

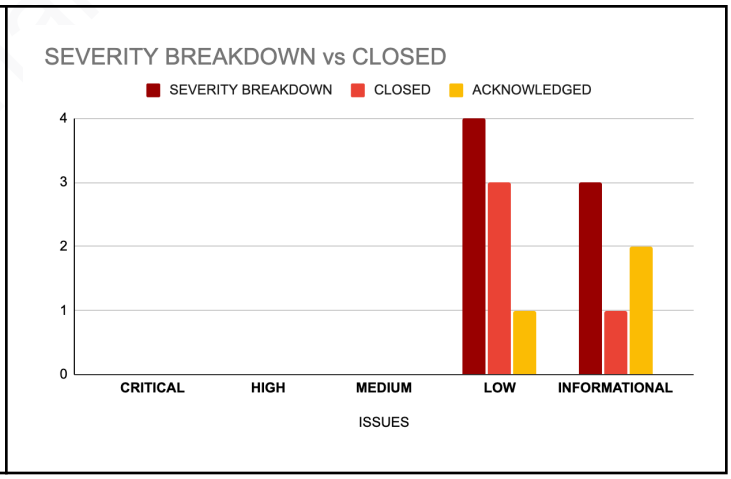


Audit Summary

Team ImmuneBytes has performed a line-by-line manual analysis and automated review of smart contracts. Smart contracts were analyzed mainly for common contract vulnerabilities, exploits, and manipulation hacks. According to the audit:

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





Finding

#	Findings	Risk	Status
01	StakingMintableToken: Return value of an external transfer is ignored	Low	Closed
02	StakingMintableToken: No Events emitted after imperative State Variable modification	Low	Closed
03	StakingMintableToken: Absence of Zero Address and Input Validation	Low	Closed
04	StakingMintableToken: External visibility should be preferred	Low	Acknowledged
05	StakingMintableToken: Infinite Token supply was found	Informatory	Acknowledged
06	StakingMintableToken: Redundant function in contract	Informatory	Closed
07	Unlocked Pragma statements found in the contracts	Informatory	Acknowledged

Critical Severity Issues
No issues were found.

High Severity Issues
No issues were found.

Medium Severity Issues
No issues were found.

Low Severity Issues

1. **StakingMintableToken: Return value of an external transfer is ignored**
   Line no: 45

   Description:
   The withdrawStake() function includes an external call to the stakedToken.transfer() which returns bool based on the successful execution of the token transfer.

   However, it was found that withdrawStake() ignores the boolean return value which can be used as a parameter to ensure that the token transfer was successful.

   Recommendation:
   Return values can be used in the function to ensure a successful transfer of tokens.

   Amended: The team has fixed the issue, and it is no longer present in the code.

2. **StakingMintableToken: No Events emitted after imperative State Variable modification**
   Line no: 39-42

   Description:
   Functions that update an imperative arithmetic state variable contract should emit an event after the update.

   The setRequiredBalance() function in the contract updates a crucial state variable, i.e., requiredBalance but doesn't emit any event on its modification.

   The absence of event emission for important state variables update also makes it difficult to track them off-chain as well.

Recommendation:
As per the best practices in smart contract development, an event should be fired after changing crucial arithmetic state variables.

Amended: The team has fixed the issue, and it is no longer present in the code.

3. **StakingMintableToken: Absence of Zero Address and Input Validation**
   Line no- 23-33

   Description:
   The constructor of the contract includes initializes a few imperative addresses and state variables in the contract.

   However, during the automated testing of the contract, it was found that no input validations is implemented before the initialization.

   Recommendation:
   A require statement should be included in such functions to ensure no invalid arguments are passed to the constructor and state variables are initialized accurately.

   Amended: The team has fixed the issue, and it is no longer present in the code.

4. **StakingMintableToken: External visibility should be preferred**

   Description:
   Functions that are never called throughout the contract should be marked as external visibility instead of public visibility. This will effectively result in Gas Optimization as well.

   Therefore, the following function must be marked as external within the contract:

   - setRequiredBalance()
   - withdrawStake()
   - burn()
   - mint()
   - grantRole()
   - revokeRole()

Recommendation:
If the PUBLIC visibility of the above-mentioned functions is not intended, then the EXTERNAL Visibility keyword should be preferred.

Acknowledged: The team has acknowledged the issue.

Informational

1. **StakingMintableToken: Infinite Token supply was found**
   Line no: 59-62

   Description:
   The current implementation allows the owner to mint an unlimited amount of tokens. The implementation should be reviewed once if a constant max token supply is needed.

   Recommendation:
   If this is not an intended behavior, the contract should include the concept of total supply and impose limitations on the number of tokens that can be minted.

   Acknowledged: The team has acknowledged the issue.

2. **StakingMintableToken: Redundant function in contract**
   Line no: 35-37

   Description:
   The contract includes a canMint() function which returns a bool that ensures whether or not staked balance is greater than the required balance.

   While this function is more of a validation step, it is not used in other functions like mint().

   Recommendation:
   It's considered a better practice to remove unwanted or redundant functions from the contract to optimize the code and enhance readability.

   Amended: The team has fixed the issue, and it is no longer present in the code.

3. **Unlocked Pragma statements found in the contracts**
   Line no: 2

   Description:
   During the code review, it was found that both contracts include unlocked pragma solidity version statements.

   It's not considered a better practice in Smart contract development to do so as it might lead to accidental deployment to a version with unfixed bugs.

   Recommendation:
   It's always recommended to lock pragma statements to a specific version while writing contracts.

   Acknowledged: The team has acknowledged the issue.

Automated Test Results

```
Compiled with Builder
Number of lines: 92 (+ 972 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 2 (+ 10 in dependencies, + 0 tests)

Number of optimization issues: 0
Number of informational issues: 16
Number of low issues: 6
Number of medium issues: 0
Number of high issues: 1

Use: Openzeppelin-Ownable, Openzeppelin-ERC20
ERCs: ERC165, ERC20

+----------------------+-------------+--------------+--------------------+--------------+--------------------+
|         Name         | # functions |     ERCS     |     ERC20 info     | Complex code |      Features      |
+----------------------+-------------+--------------+--------------------+--------------+--------------------+
|    FixSupplyToken    |      30     |    ERC20     |     No Minting     |      No      |                    |
|                      |             |              | Approve Race Cond. |              |                    |
| StakingMintableToken |      61     | ERC20,ERC165 |     ∞ Minting      |      No      | Tokens interaction |
|                      |             |              | Approve Race Cond. |              |                    |
+----------------------+-------------+--------------+--------------------+--------------+--------------------+
```

Test Cases

```
Tokens
  ✓ sent all the fixed supply tokens to address 0 (96ms)
  ✓ deploys a token contract and mints all the tokens into multiple accounts (162ms)
  ✓ throws an error if the arrays are not of equal length (50ms)
  ✓ sent all the fixed supply tokens to address 0
  ✓ forbids non owner from minting (109ms)
  ✓ no tokens can be minted if no stake has been provided (63ms)
  ✓ the owner can add new minters (42ms)
  ✓ non owner can not add new minters (86ms)
  ✓ indicats that minting is disabled
  after providing stake
    ✓ indicats that minting is enabled
    ✓ owner can mint new mintable tokens
    ✓ anyone can burn mintable tokens (48ms)
    ✓ forbids non owner from minting (98ms)
    ✓ the stake can be withdrawn by the owner
    ✓ if the stake is withdrawn by the owner minting is blocked (39ms)


15 passing (2s)
```

Concluding Remarks

While conducting the audits of the SuperLoyal smart contracts, it was observed that the contracts contain only Low severity issues and a few recommendations.

Note:
The SuperLoyal team has fixed the issues based on the auditor's recommendation.

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 complementing this audit is strongly recommended.

Our team does not endorse the SuperLoyal platform or its product, nor is this audit investment advice.

Notes:

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