LOKR

Vault

Smart Contract Audit
Final Report

November 30, 2022

Introduction

1. About LOKR

Lokr provides projects and individuals with a full suite of tools to manage their token economies. Whether it be token creation, distribution, creating cross-chain bridging solutions, or trustless escrows, Lokr has you covered. Lokr lowers the barrier of entry into the crypto space, making blockchain accessible to everyone through simple plug-and-play cross-chain solutions.

Vault:

Vault is a highly-customizable, multi-chain token-locking platform. Designed with the end-user in mind, users can create complex token release models within a few clicks. Vault takes advantage of the latest developments in blockchain technology and, through integration with Oracles, allows users to create event-driven distribution models. The Vault dashboard has a simple and intuitive interface that lets users monitor project token emissions and track their portfolios. Through Vault, projects and users can lock and automate the distribution of tokens, liquidity, and NFTs.

Visit https://www.lokr.io/ 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 LOKR team has provided the following doc for audit:

1. https://immunebytes.notion.site/Standard-Contract-LOKR-lokr-c2e3fc3de9bc450d9cd432ddbe7f6fa2

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: LOKR
- Languages: Solidity(Smart contract), Typescript (Unit Testing)
- Github link: https://github.com/Polkalokr/lkr-lock-app
- Commit hash: 56443309afe541fd2637489ddb26916a06f46abe
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

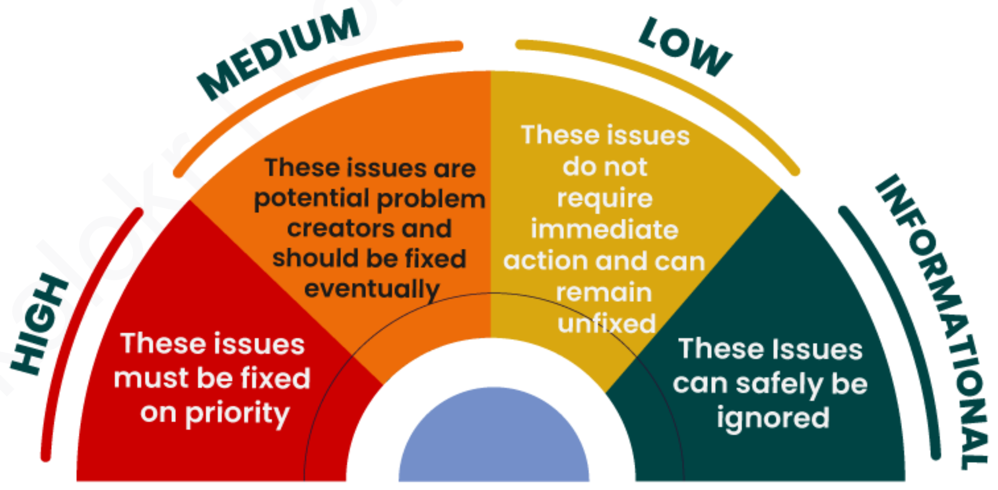
Audit Goals

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:

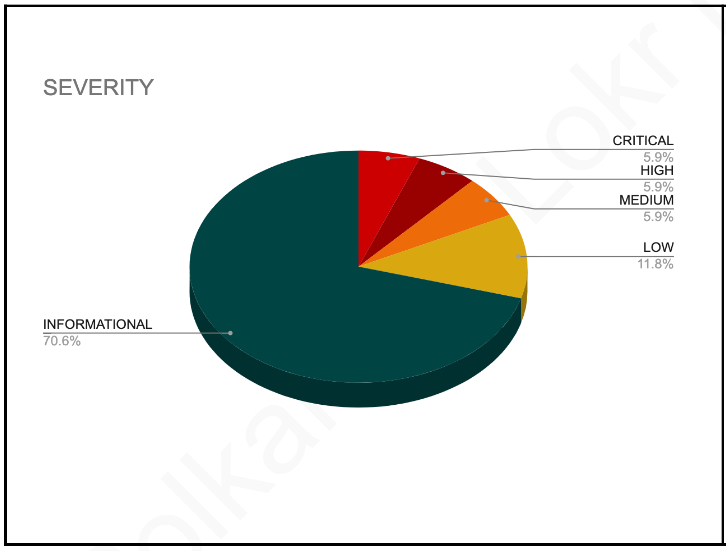

HIGH
These issues must be fixed on priority
MEDIUM
These issues are potential problem creators and should be fixed eventually
LOW
These issues do not require immediate action and can remain unfixed
INFORMATIONAL
These Issues can safely be ignored

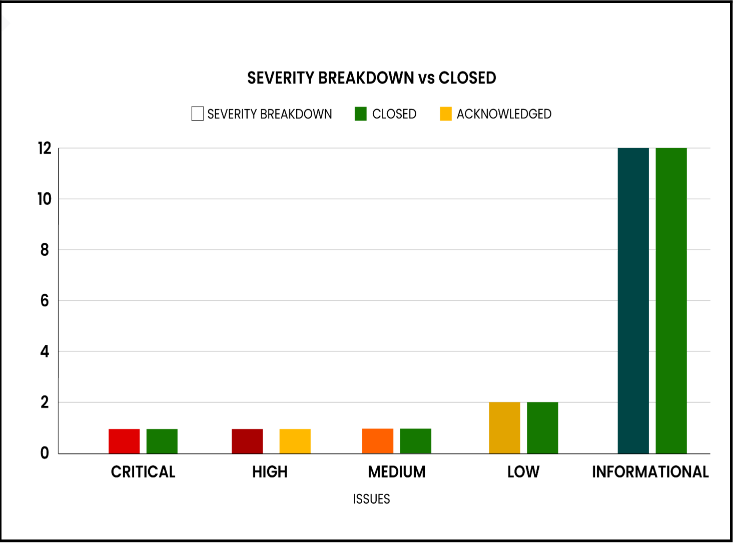
This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process; therefore, running a bug bounty program as a complement to this audit is strongly recommended.

Audit 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	1	-	1	2
Acknowledged	-	1	-	-





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.

Finding Overview

S.No	Findings	Risk	Status
1	Illogical check leading to unreachable data	Critical	Closed
2	Incorrect logic leads to design flaw	High	Acknowledged
3	Insufficient data sanitization	Medium	Closed
4	Emit appropriate events	Low	Closed
5	Misleading error message	Low	Closed
6	Remove unused functions	Informatory	Closed
7	Remove commented code	Informatory	Closed
8	For loop gas optimizations	Informatory	Closed
9	Code style to increase readability	Informatory	Closed
10	Spell checks in codebase	Informatory	Closed
11	Use linting tool for codebase	Informatory	Closed
12	Use integer LITERAL(s)	Informatory	Closed
13	Separate functions for code readability	Informatory	Closed
14	Improve code layout and code style	Informatory	Closed
15	Correction in NatSpec	Informatory	Closed
16	Mark immutable variables	Informatory	Closed
17	Missing of Override keyword from the function	Informatory	Closed

Audit Changes

The LOKR team has implemented the recommendations based on the auditor's initial finding as under:

File Changes

1. **Contracts/DateUnlockSchedule.sol (Low Risk issue # 2, Informatory issues # 3,5 and 6):**
    - Add variable at line #12.
    - Update require statement at line # 42.
    - Update for loop at line # 65.
    - Add some conditions at line # 69-72.
    - Replace 1e18 to variable name EXP at line # 74.
    - Update for loop at line # 101.
    - Replace 1e18 to variable name EXP at line # 107.
    - Update for loop at line # 111.

2. **Contracts/Deployfee.sol(Informatory issues #3,5 and 6, Critical issue #1, Medium Risk issue #1);**
    - Update code from line # 70 to 90.
    - Create new function with name `updateDeployFeeBeneficiaryInternal()--this function is used to update the address of beneficiary to bring deploy fee` at line # 124.
    - Update for loop at line # 266.
    - Add some check from line # 283- 285.
    - Replace for loop with if statement at line # 291.
    - Update for loop at line # 314.
    - Add some check from line # 332- 334.
    - Create new function with name `updateCryptoFeedInternal()--This function update the price feed from oracle` at line # 338.

3. **Contract/DeplositManger.sol(Informatory issues # 3,5, 6 and 7,12):**
    - Add two variable at line # 28 and 29.
    - Update for loop from line # 87-102.
    - Update for loop from line # 133-139.
    - Update for loop from line # 198-202.
    - Update for loop from line # 221-227.
    - Update for loop from line # 247-253.

4. **Contract/DeplositMangerMT.sol (Informatory issues # 3, 5, 6, and 7):**
    - Add two variables at line # 30 and 31.
    - Update for loop from line # 93-108.
    - Update for loop from line # 144-150.
    - Update for loop from line # 219-223.
    - Update for loop from line # 242-248.
    - Update for loop from line # 268-274.
    - Update for loop from line # 300-304.
    - Update for loop from line # 310-322.
    - Update for loop from line # 332-342.

5. **Contracts/EventUnlockSchedule.sol (Informatory issues # 3, 5 and 6)**
    - Update for loop from line # 121-125
    - Update for loop from line # 131-135.
    - Update for loop from line # 165 and 181 .
    - Update for loop from line # 259 and 263.

6. **Contracts/FixedValueDepsoirManager.sol(Informatory issues # 3,5,6, and 7):**
    - Add two variables at line # 30 and 31
    - Add some code in addDeposits function at line # 78 and 79.
    - Variable assigning value changed at line # 84.
    - Update for loop from line # 88- 103.
    - Update for loop from line # 134-140.
    - Update _checkAmount function at line # 197.
    - Update for loop at line # 222 and 226.
    - Update for loop from line # 248-254.

7. **Contracts/FixedvalueLock.sol(Low Issue # 1,Informatory issues # 2,3, 5, 6 and 10)**
    - Add two new events at line # 24 and 26.
    - Add code at line # 146.
    - Update for loop at line # 202 and 210
    - Update addBeneficiaries at line # 227-232.
    - Update removeBeneficiaries at line # 243 and 257.
    - Add code at line # 270.
    - Update for loop from line # 314-319

8. **Contracts/IntervalUnlockSchedule.sol (Informatory issues # 3,5 and 6)**
    - Update for loop from line # 65-70.
    - Update for loop from line # 99-103.
    - Update for loop from line # 109-115.

9. **Contracts/Lock.sol(Low risk issue # 1, Informatory issues # 1, 2, 3, 5, 6 and 10)**
   - Add two new events at line # 22 and 24.
   - Update for loop from line # 197 and 205.
   - Update addBeneficiaries at line # 222-227.
   - Update removeBeneficiaries at line # 238 and 252.
   - Add code at line # 265.
   - Update for loop from line # 309-314.

10. **Contracts/LockFactory.sol (Informatory issues # 3, 5, 6 and 8):**
   - Update for loop from line # 54-59.
   - Add new function getCryptoPriceFeed at line # 111.
   - Update for loop from line # 162-168.
   - Update for loop from line # 176-182.
   - Add new code at line # 248.
   - Add new code from line # 393-444.

11. **Contracts/SplitManagger.sol (Informatory issues # 3, 5,6 and 11)**
   - Update require statement at line # 41.
   - Update for loop from line # 86-90.
   - Update for loop from line # 110-114.

Additional Changes

LOKR Changes With TX-fees

The changes we made in this section is if we mint a token on Mintr platform with Tx fee and any user (token owner or holder) wants to create a lock with a Lokr then the Tx fee will be whitelisted (means no Tx fee will be deducted in scenario of either creating a lock or beneficiary claiming the lock). However if any token is minted from another platform with Tx fee capability then user cannot create a lock on this platform.

File changes

- **Contracts/FixedValueLock.sol :**
  Add require Statement on line # 147 in this function `_initializeBeneficiaries().`

Note: This requires statements to check that those token which have tx-fee capabilities can't lock at vault.

- **Contracts/Lock.sol:**
  Require statement added at on line # 146:
  ```
  require(tokenERC20.balanceOf(address(this)) == _lockedAmount, "ERROR:
  Can't Lock Tokens With TX Fee");
  ```

Note: This requires statements to check that those token which have tx-fee capabilities can't lock at vault.

- **Contracts/LockFactory.sol:**
  - Add Tx-fee-whitelisted role at line # 36.
  - Add address type variable at line # 40.
  - Move code from line # 32-45 at   line # 16-29.
  - Create new function with name setMintrFactory() at line # 102.
  - Add some changes and checks from line # 371 to 379
  - Create new function with name whitelistLokr() at line # 461.

Concluding Remarks

While conducting the audits of the LOKR smart contracts, it was observed that the contracts contain Critical, High, Medium, and Low severity issues.

Note:
The LOKR 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 LOKR platform or its product, nor this audit is 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.