Kurobi

Smart Contract Audit Report

April 06, 2021

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 Kurobi

Kurobi is a decentralized video platform for paid 1:1 video calls with your audience. Kurobi aims to provide a platform where users can schedule calls with a verified person, receive instant payment for their time, and make transactions on-chain. This is a tool to encourage users to offer their skills, valuable time, mentorship, counseling, tutorship, and lots more to others while earning some cash in return.

Visit https://kurobi.io/ to know more about it.

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

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

1. Litepaper.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: Kurobi
- Languages: Solidity(Smart contract), Javascript(Unit Testing)
- Github commit hash for audit: 5263b0377a77b1b3bfc54b058d6cd6e09336959f
    - Kurobi Token (kovan): 0xfb08ca5d45e2f3fbeef125befefff6be35357124
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

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	-	-	2
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.

Admin/Owner Privileges

There are **NO** Admin Privileges present in the Kurobi ERC20 token contract.

Low severity issues

1. ***_msgSender()* should be used instead of *msg.sender***
   In the **constructor**() of **Kurobi** token contract, **msg.sender** is used to mint the initial supply of token to the deployer of the contract. Since the OpenZeppelin's contracts use a **_msgSender**() function to access the address of the caller, it is recommended to use the same coding conventions throughout the contract to maintain consistency in code readability.

   Recommendation:
   Consider changing **msg.sender** with **_msgSender()**.

2. **Kurobi contract should only inherit *ERC20Burnable*.**
   The **Kurobi** token contract present in **KurobiToken.sol** inherits **ERC20** and **ERC20Burnable** contracts.

   ```
   contract Kurobi is ERC20, ERC20Burnable { … }
   ```

   Since the **ERC20Burnable** itself inherits the **ERC20** contract there is no need to inherit **ERC20** again.

   Recommendation:
   Consider only inheriting the **ERC20Burnable** contract for **Kurobi** token.

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.

Unit Test

No unit tests were provided by the Kurobi 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.

Automated Auditing

Solhint Linting Violations

Solhint is an open-source project for linting solidity code, providing both security and style guide validations. It integrates seamlessly into most mainstream IDEs. We used Solhint as a plugin within our VScode for this analysis. Some Linting violations were detected by Solhint in Kurobi token contract, it is recommended to use Solhint's npm package to lint the contracts.

Contract Library

Contract-library contains the most complete, high-level decompiled representation of all Ethereum smart contracts, with security analysis applied to them in real-time. We performed analysis using contract Library on the Kovan address of the Kurobi Token contract used during manual testing:

- Kurobi Token: 0xfb08ca5d45e2f3fbeef125befefff6be35357124

It raises no major concern for the contract.

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 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.

The concerns slither raises have already been covered in the manual audit section.

```
ERC20.constructor(string,string).symbol (contracts/KurobiToken.sol#446) shadows:
        - ERC20.symbol() (contracts/KurobiToken.sol#463-465) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Address.isContract(address) (contracts/KurobiToken.sol#281-290) uses assembly
        - INLINE ASM (contracts/KurobiToken.sol#288)
Address._functionCallWithValue(address,bytes,uint256,string) (contracts/KurobiToken.sol#374-395) uses assembly
        - INLINE ASM (contracts/KurobiToken.sol#387-390)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Pragma version0.6.12 (contracts/KurobiToken.sol#5) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
solc-0.6.12 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (contracts/KurobiToken.sol#308-314):
        - (success) = recipient.call{value: amount}() (contracts/KurobiToken.sol#312)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (contracts/KurobiToken.sol#374-395):
        - (success,returndata) = target.call{value: weiValue}(data) (contracts/KurobiToken.sol#378)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Redundant expression "this (contracts/KurobiToken.sol#23)" inContext (contracts/KurobiToken.sol#17-26)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Variable Kurobi.TOTAL_SUPPLY (contracts/KurobiToken.sol#737) is too similar to ERC20._totalSupply (contracts/KurobiToken.sol#431)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
Kurobi.slitherConstructorConstantVariables() (contracts/KurobiToken.sol#733-743) uses literals with too many digits:
        - TOTAL_SUPPLY = 10000000 * 10 ** uint256(DECIMALS) (contracts/KurobiToken.sol#737)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
name() should be declared external:
        - ERC20.name() (contracts/KurobiToken.sol#455-457)
symbol() should be declared external:
        - ERC20.symbol() (contracts/KurobiToken.sol#463-465)
decimals() should be declared external:
        - ERC20.decimals() (contracts/KurobiToken.sol#480-482)
totalSupply() should be declared external:
        - ERC20.totalSupply() (contracts/KurobiToken.sol#487-489)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (contracts/KurobiToken.sol#494-496)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (contracts/KurobiToken.sol#506-509)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (contracts/KurobiToken.sol#525-528)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (contracts/KurobiToken.sol#542-546)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (contracts/KurobiToken.sol#560-563)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (contracts/KurobiToken.sol#579-582)
burn(uint256) should be declared external:
        - ERC20Burnable.burn(uint256) (contracts/KurobiToken.sol#709-711)
burnFrom(address,uint256) should be declared external:
        - ERC20Burnable.burnFrom(address,uint256) (contracts/KurobiToken.sol#724-729)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:./contracts/KurobiToken.sol analyzed (7 contracts with 72 detectors), 23 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

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 Kurobi smart contract, it was observed that the contracts contain only Low severity issues, along with several areas of recommendations.

Our auditors suggest that Low severity issues should be resolved by Kurobi 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 Kurobi 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.