Blaize.Security

March 27th, 2023 / V. 1.0

GEROBI

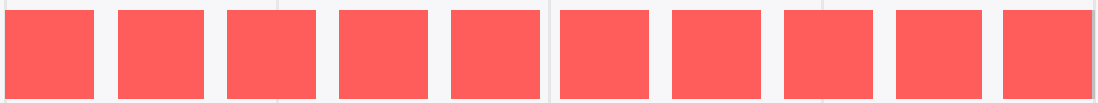
SMART CONTRACT AUDIT

TABLE OF CONTENTS

AUDIT RATING

Gerobi contract's source code was taken from the contract deployed on Aurora testnet.

SCORE **10** /10



The scope of the project includes **Gerobi** set of contracts:

1/ GerobiERC20.sol

Code was delivered as a contract deployed on Aurora testnet:

https://explorer.aurora.dev/address/0xEA4c81B51F02d0EC3d43278A4CB83CDc68d68b9b/contracts#address-tabs

Latest audited code was delivered as a contract deployed on Aurora testnet:

https://explorer.aurora.dev/address/0xeF15465E9Cc468202b94b4b04a92e4ec3cBd4EC8/contracts#address-tabs

TECHNICAL SUMMARY

During the audit, we examined the security of smart contracts for the Gerobi protocol. Our task was to find and describe any security issues in the smart contracts of the platform. This report presents the findings of the security audit of the **Gerobi** smart contracts conducted between **March 24th, 2023** and **March 27th, 2023.**

Testable code

The code is 100% testable, which corresponds to the industry standard of 95%.

The scope of the audit includes the unit test coverage, which is based on the smart contract code, documentation and requirements presented by the Gerobi team. The coverage is calculated based on the set of Hardhat framework tests and scripts from additional testing strategies. However, to ensure the security of the contract, the Blaize.Security team suggests that the Binaryx team launch a bug bounty program to encourage further active analysis of the smart contracts.

THE GRAPH OF VULNERABILITIES DISTRIBUTION:

The table below shows the number of the detected issues and their severity. A total of 1 problems were found. 0 issues were fixed or verified by the Binaryx team.

	FOUND	FIXED/VERIFIED
Critical	0	0
High	0	0
Medium	0	0
Low	0	0
Lowest	1	0

SEVERITY DEFINITION

Critical

The system contains several issues ranked as very serious and dangerous for users and the secure work of the system. Requires immediate fixes and a further check.

High

The system contains a couple of serious issues, which lead to unreliable work of the system and migh cause a huge data or financial leak. Requires immediate fixes and a further check.

Medium

The system contains issues that may lead to medium financial loss or users' private information leak. Requires immediate fixes and a further check.

Low

The system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Requires fixes.

Lowest

The system does not contain any issues critical to the secure work of the system, yet is relevant for best practices

AUDITING STRATEGY AND TECHNIQUES APPLIED/PROCEDURE

We have scanned this smart contract for commonly known and more specific vulnerabilities:

- Unsafe type inference;
- Timestamp Dependence;
- Reentrancy;
- Implicit visibility level;
- Gas Limit and Loops;
- Transaction-Ordering Dependence;
- Unchecked external call - Unchecked math;
- DoS with Block Gas Limit;
- DoS with (unexpected) Throw;
- Byte array vulnerabilities;
- Malicious libraries;
- Style guide violation;
- ERC20 API violation;
- Uninitialized state/storage/ local variables;
- Compile version not fixed.

Procedure

We checked the contract for the following parameters:

- Whether the contract is secure;
- Whether the contract corresponds to the documentation;
- Whether the contract meets the best practices in the efficient use of gas, code readability.

Automated analysis:

Scanning contracts by several publicly available automated analysis tools such as Mythril, Solhint, Slither, and Smartdec. Manual verification of all the issues found with tools.

Manual audit:

Manual analysis of smart contracts for security vulnerabilities. We checked smart contract logic and compared it with the one described in the documentation.

EXECUTIVE SUMMARY

Audited contracts represent ERC20 token with standard OpenZeppelin implementation. The contract also inherits ERC20Permit contract.

Auditors verified compatibility with the ERC20 standard, checked that the token inherits standard OpenZeppelin contracts (most of standard contracts from 4.8.0 version). Auditors also prepared set of tests to check the standard functionality (transfer, approve, balances, permit) and correct token parameters.

Token will have ticker “Gerobi Token” / “GRB” with the initial supply minted just once during the construction and transferred to the recipient chosen by the deployer.

Audit detected the only issue with the solc version used for contracts.

	RATING
Security	10
Gas usage and logic optimization	10
Code quality	9.9
Test coverage	10
Total	10

COMPLETE ANALYSIS

LOWEST-1	Unresolved

Inaccurate version pragma.

Contracts use solc ^0.8.5 - with the floating version. Locking the pragma version helps ensure that the contract is not accidentally deployed using a different version. In addition, older versions may contain bugs and vulnerabilities, and be less optimized in terms of gas. It is recommended to use the latest stable version of Solidity and specify the fixed pragma: pragma solidity 0.8.19

Recommendation:

Specify the fixed latest stable version of Solidity in the pragma statement (which is 0.8.19 now).

Post-audit:

In the latest code version, solc was updated to 0.8.9

	GerobiERC20.sol
✔ Re-entrancy	Pass
✔ Access Management Hierarchy	Pass
✔ Arithmetic Over/Under Flows	Pass
✔ Delegatecall Unexpected Ether	Pass
✔ Default Public Visibility	Pass
✔ Hidden Malicious Code	Pass
✔ Entropy Illusion (Lack of Randomness)	Pass
✔ External Contract Referencing	Pass
✔ Short Address/Parameter Attack	Pass
✔ Unchecked CALL Return Values	Pass
✔ Race Conditions/Front Running	Pass
✔ General Denial Of Service (DOS)	Pass
✔ Uninitialized Storage Pointers	Pass
✔ Floating Points and Precision	Pass
✔ Tx.Origin Authentication	Pass
✔ Signatures Replay	Pass
✔ Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES, PREPARED BY BLAIZE SECURITY TEAM

GerobiERC20

Deployment

- ✓ Should deploy with correct name
- ✓ Should deploy with correct symbol
- ✓ Should deploy with correct decimal
- ✓ Should mint tokens for recipient when deployed

ERC20 functionality

Approve

- ✓ Should get right approved balance
- ✓ Should change allowance when increasing
- ✓ Should change allowance when decreasing

Transfers

- ✓ Should get right balance after transfer
- ✓ Shouldn't add balance when transfer to yourself
- ✓ Shouldn't transfer more that balance
- ✓ Shouldn't transfer using transferFrom without approve
- ✓ Should decrease allowance when using transferFro

Permit

- ✓ Should approve tokens using signature

13 passing (370ms)

TEST COVERAGE RESULTS

FILE	% STMTS	% BRANCH	% FUNCS
GerobiERC20.sol	100	100	100
All files	**100**	**100**	**100**

DISCLAIMER

The information presented in this report is an intellectual property of the customer, including all the presented documentation, code databases, labels, titles, ways of usage, as well as the information about potential vulnerabilities and methods of their exploitation. This audit report does not give any warranties on the absolute security of the code. Blaize.Security is not responsible for how you use this product and does not constitute any investment advice.

Blaize.Security does not provide any warranty that the working product will be compatible with any software, system, protocol or service and operate without interruption. We do not claim the investigated product is able to meet your or anyone else's requirements and be fully secure, complete, accurate, and free of any errors and code inconsistency.

We are not responsible for all subsequent changes, deletions, and relocations of the code within the contracts that are the subjects of this report.

You should perceive Blaize.Security as a tool, which helps to investigate and detect the weaknesses and vulnerable parts that may accelerate the technology improvements and faster error elimination.