QuillAudits

AUDIT REPORT

January 2025

For

GALILEO

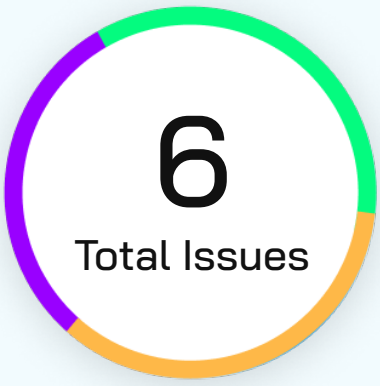
Table of Content

Table of Content

Executive Summary

Project Name	Galileo
Project URL	*https://www.galileoprotocol.io/*
Overview	Galileo Protocol is a cutting-edge tokenisation platform that aims to transform how luxury goods and real-world assets are owned and authenticated. The protocol is built to support EVM compatible blockchains and utilises advanced AI and machine learning algorithms to provide a secure and transparent way for collectors and investors to verify the authenticity of their valuable assets
Audit Scope	*https://github.com/Galileo-Protocol-io/Galileo_Staking*
Contracts in Scope	contracts/GalileoStaking.sol contracts/GalileoSoulBoundToken.sol
Commit Hash	8a51424285eb2b4e55037e604d2908feade3e9b1
Language	solidity
Blockchain	EVM
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	25th December 2024 - 7th January 2025
Updated Code Received	16th January 2025
Review 2	20th January 2025 - 22nd January 2025
Fixed In	a5f7d1343c983de0a6ed07569e73fa73ec508fdd

Number of Issues per Severity



High	0 (0%)
Medium	2 (33%)
Low	2 (33%)
Informational	2 (33%)

Severity

Issues	High	Medium	Low	Informational
Open	0	0	0	0
Resolved	0	**2**	**2**	**1**
Acknowledged	0	0	0	**1**
Partially Resolved	0	0	0	0

Checked Vulnerabilities

- Access Management
- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic Computations Correctness
- Race conditions/front running
- SWC Registry
- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC's conformance
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls

Checked Vulnerabilities

- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Multiple Sends
- ✓ Using suicide
- ✓ Using delegatecall
- ✓ Upgradeable safety
- ✓ Using throw
- ✓ Using inline assembly
- ✓ Style guide violation
- ✓ Unsafe type inference
- ✓ Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code
- Use of best practices
- Code documentation and comments, match logic and expected behavior
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper
- Implementation of ERC standards
- Efficient use of gas
- Code is safe from re-entrancy and other vulnerabilities

The following techniques, methods, and tools were used to review all the smart contracts:

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Techniques and Methods

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms Used for Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistic analysis

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Medium Severity Issues

1. Stake function allows staking NFTs with token ID 0

Resolved

Path
contracts/GalileoStaking.sol

Function Name
stake

Description
The _stakeTokens function allows users to stake NFTs with a token ID of 0. However, unstaking of token ID 0 is disallowed, leading to an inconsistency in the staking logic. If a user stakes a token with ID 0, they might face issues retrieving their NFT later, as the unstaking functionality does not account for this scenario.

Impact
- Users can stake NFTs with token ID 0 but will not be able to unstake them due to unstaking restrictions.
- This could result in locked NFTs that cannot be retrieved, leading to frustration and potential loss of assets for users.

Recommendation
Add a check in the _stakeTokens function to prevent staking of NFTs with token ID 0.

Chain ID is not used in the signature, allowing replay attacks on other EVM chains

Resolved

Path
contracts/GalileoStaking.sol

Function Name
_recover

Description
The _recover function uses EIP-712 typed data hashing to generate a digest for recovering the signer from a provided signature. However, the chain ID is not included in the hashed data. This omission allows the same signature to be reused (replayed) on other EVM-compatible chains that share the same contract address and data structure, potentially causing unintended actions.

Impact
Signatures could be maliciously reused across chains, leading to unauthorized staking on different chains.

Recommendation
Include the chain ID in the EIP-712 encoded data. This ensures the signature is tied to a specific blockchain, preventing replay attacks on other chains.

Low Severity Issues

3. The ‘already staked’ check can be bypassed in stakeLeoxTokens by passing tokenId as 0

Resolved

Path
contracts/GalileoStaking.sol

Function Name
stakeLeoxTokens()

Description
The stakeLeoxTokens function checks if the token is already staked by verifying the tokenId. However, this check can be bypassed by passing 0 as the tokenId. Since the tokenId field in the StakePerCitizen struct defaults to 0, the condition stakePerCitizen.tokenId != tokenId will evaluate as false, allowing the function logic to proceed. This creates an inconsistency and can potentially lead to staking logic being executed for an invalid token.

Impact
- Users may bypass staking validations for specific token IDs, potentially leading to undefined or unintended behavior.
- It may result in reward miscalculations or incorrect updates in staking data.

Recommendation
Add a check to disallow a tokenId of 0 in the stakeLeoxTokens function.

4. No check for emergency state in the _stakeTokens and stakeLeoxTokens functions

Resolved

Path
contracts/GalileoStaking.sol

Function Name
_stakeTokens and stakeLeoxTokens

Description
The current implementation of the stake and _stakeTokens functions does not verify whether an emergency state has been declared. If the contract is in an emergency state, allowing users to stake tokens may result in undesirable behavior, such as locking funds in an unsafe environment or exposing the protocol to additional risks.

Adding a check to ensure that staking is not permitted during emergencies would enhance security and user protection.

Impact
- Users may unknowingly stake tokens during an emergency, potentially losing access to their funds if the protocol remains in an unstable state.

Recommendation
Add an emergency check for both of the functions and ensure that staking functions check this state before allowing any staking operation.

Informational Issues

5. Rewards mapping set to zero without withdrawal in _emergencyUnstake

Acknowledged

Path
contracts/GalileoStaking.sol

Function Name
_emergencyUnstake

Description
In the _emergencyUnstake function, the reward mapping (state.rewards[recipient][collectionAddress][tokenId]) is reset to 0 without first allowing users to withdraw their accrued rewards. This can lead to users losing their rewards, especially in scenarios where the protocol does not provide an explicit mechanism to recover those rewards during or after an emergency.

Impact
- Users may permanently lose their accrued rewards during an emergency unstaking process.
- This could lead to dissatisfaction and loss of trust in the protocol.

Recommendation
Before setting the rewards mapping to 0, transfer the accumulated rewards to the user. This ensures that users do not lose any rewards.

6. Incorrect comment in configureNewCollection regarding maxLeox validation

Resolved

Path
contracts/GalileoStaking.sol

Function Name
configureNewCollection

Description
The comment in the configureNewCollection function inaccurately describes the logic of the if block. Specifically, it states:
// Ensure that maxLeox does not decrease compared to the previous tier
However, the implementation actually enforces that maxLeox values must decrease (i.e., be in descending order) compared to the previous tier. This discrepancy between the comments and the logic leads to confusion, as it contradicts the intention verified in the written tests.

Impact
- Misleading documentation could confuse developers and auditors, leading to incorrect assumptions about the function's behavior.

Recommendation
Modify the comment to accurately describe the enforced descending order of maxLeox

Functional Tests

Some of the tests performed are mentioned below:

- Should allow staking of NFT and LEOX and 0 token id as well
- Test to see if more than allowed tokens can be staked per citizen ID
- Test to see if rewards are wrong
- Test to see if less tokens results in calculating different amount of rewards
- Should revert Admin to withdraw tax and the smart contract does not have amount
- Should return the expected rewards if there is only one staker
- Should return the expected rewards if there are 2 stakers
- Should calculate the collect points
- Should return rewards per token allocation

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of Galileo. We performed our audit according to the procedure described above.

Some issues of High,low,medium and informational severity were found. Some suggestions, gas optimizations and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Galileo. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Galileo. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of Galileo to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

6+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1K+** Projects Secured

Follow Our Journey

AUDIT REPORT

January 2025

For

GALILEO

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillhash.com