QuillAudits

# Audit Report
March, 2024

For

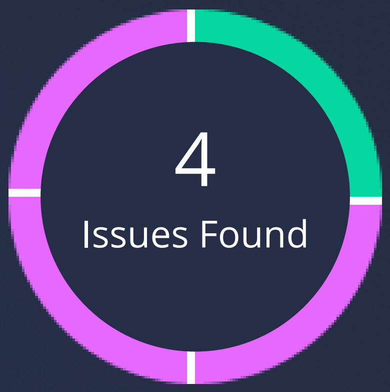
Velar

## Table of Content

## Executive Summary

| | |
|---|---|
| **Project Name** | Velar Token |
| **Overview** | The Velar token is an ERC20 Token Contract deployed on Ethereum. VELAR, is a key component of the Velar ecosystem. Velar also utilizes the Stacks blockchain as its underlying technology, enabling the integration of DeFi features on the Bitcoin blockchain. |
| **Timeline** | 18th March 2024 |
| **Updated Code Received** | NA |
| **Second Review** | NA |
| **Method** | Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives. |
| **Audit Scope** | This audit aimed to analyze the Velar Token Codebase for quality, security, and correctness. |
| **Source Code** | *https://etherscan.io/address/0x033BbDe722EA3Cdcec73cFFEA6581DF9F9C257de#code* |
| **Fixed In** | NA |

## Number of Security Issues per Severity



| | |
|---|---|
| 🟥 High | 🟨 Medium |
| 🟩 Low | 🟪 Informational |

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

## Checked Vulnerabilities

- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ DoS with Block Gas Limit
- ✓ Transaction-Ordering Dependence
- ✓ Use of tx.origin
- ✓ Exception disorder
- ✓ Gasless send
- ✓ Balance equality
- ✓ Byte array
- ✓ Transfer forwards all gas
- ✓ ERC20 API violation
- ✓ Malicious libraries
- ✓ Compiler version not fixed
- ✓ Redundant fallback function
- ✓ Send instead of transfer
- ✓ Style guide violation
- ✓ Unchecked external call
- ✓ Unchecked math
- ✓ 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-20 token 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.

## 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

Hardhat, Foundry.

## Types of Severity

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

### High Severity Issues

A high severity issue or vulnerability means 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 impacts 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 issues were identified in the initial audit and 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.

## Low Severity Issues

### 1. Avoid using floating pragma

**Path**

Velar.sol

**Description**

Floating pragma should only be used when a contract is intended for consumption by other developers.

Locking the pragma helps ensure that contracts are not accidentally deployed using, for example, the latest compiler, which may have higher risks of undiscovered bugs.

**Recommendation**

Use locked pragma instead.

**Status**

**Acknowledged**

## Informational Issues

### 2. Consider defining a constant

**Path**

Velar.sol

**Description**

The contract uses literals with many digits, potentially leading to human error or misleading, particularly when the number of zeros is large.

However, in the provided smart contract code, the _mint function is called with a clear multiplication of two numbers, '1000000000' and '10**decimals()'.

Since the decimals function overrides the standard 18 decimals with 6, the actual number minted is '1000000000 * 10**6', which is intentional and correct for the token's intended design. There is no inherent vulnerability in having a large number of tokens; it is a design decision that affects the granularity and distribution of the token.

**Recommendation**

Consider defining a constant for the initial supply or using explanatory comments to make the code clearer to future reviewers and maintainers. This will improve readability and prevent human error.

**Status**

**Acknowledged**

### 3. 6 decimal places can cause problem in future

**Path**

Velar.sol

**Description**

According to the protocol’s whitepaper, the Velar token is intended to be used in the entire velar ecosystem, including Governance, Staking, etc.

In that case, integrating the Velar token might create much more redundant complexity in calculations in future smart contract developments. It will become more prone to rounding issues if not handled properly.

Here 1 token would be **1*(10**6) = 1000000000** Wei. It may happen that any other smart contract uses/accepts this token for some reason. That smart contract calculates the token amount sent by the user assuming its 18 decimal token, which can result in unwanted outcomes.

**Eg:** care needs to be taken in this type of scenario.
**User** sends 1 token (**"1*(10**6)"** in this case) to a smart contract.
The smart contract which accepts this token checks the token amount sent by **User** which was **1*(10**8) = 100000000**
While calculating amount sent by the **User**, smart contract uses **18 decimals** and expects **1 token** sent to be **1*(10**18) = 1000000000000000000**
In this case this condition will fail since token amount sent by **User** is **1000000** i.e **1*(10**6)** and not **1*(10**18)**

**Recommendation**

If possible, consider changing the decimal place to 18 since the Velar token is used in more places mentioned above other than the fiat aspect. This will make integration and calculation much easier during developing more sophisticated DeFi products.

**Velar Team's Comment**

We need 6 decimals because thats standard on stx (128bit uints).

**Status**

**Resolved**

## 4. Remove Unused Interfaces

**Description**

The contract inherited a couple of Interfaces intended to aid achieve the safety creation of ERC20 token. However, there are some Interfaces that were left unused in the contract.

**Remediation**

It is recommended to remove unused interfaces that are no longer needed to build the contract.

**Status**

**Acknowledged**

## Functional Tests Cases

- ✓ Should get the name of the token
- ✓ Should get the symbol of the token
- ✓ Should get the total supply of the token when deployed
- ✓ Should approve another account to spend token
- ✓ Should transfer tokens to other address
- ✓ Should approve another account to spend token

## 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 the Velar codebase. We performed our audit according to the procedure described above.

Some issues of Low and informational severity were found, Some suggestions 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 Velar smart contracts. 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 Velar smart contracts. 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 the Velar to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

## About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

**1000+**
Audits Completed

**$30B**
Secured

**1M**
Lines of Code Audited

## Follow Our Journey

# Audit Report March, 2024

For

Velar

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com