QuillAudits

AUDIT REPORT

February, 2025

For

Table of Content

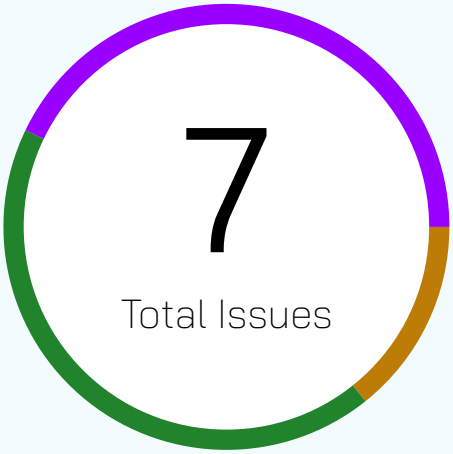
Table of Content

Executive Summary

Project name	Growt
Overview	Growt is a DeFi project leveraging an ERC20 token with a unique vesting contract mechanism for token distribution. It introduces a custom token with adjustable decimals, primarily set to 6 for precision. The project's standout feature is its Vesting contract, which manages token allocations across various categories like pre-seed, team, marketing, and liquidity, among others. This contract calculates and releases tokens based on predefined vesting periods, ensuring a controlled and phased distribution. This approach aims to provide a fair and transparent token economy, supporting the project's growth and sustainability.
Update code Received	2025-02-06
Contracts In Scope	Vesting and token
Audit Scope	The scope of this Audit was to analyze the Growt Smart Contracts for quality, security, and correctness. https://testnet.bscscan.com/address/0x8739d144AE0F4a899e18D9C01BA7EB3A5C5FB9F5#readContract
Language	Solidity
Blockchain	BSC
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	24th January 2025 - 1st Feb 2025

Updated Code received	3rd February 2025
Review 2	6th February 2025
Fixed in	https://drive.google.com/drive/folders/1sU546aqMS-dygl7CXPJN0b6EqtkGKe6jE?usp=sharing
Mainnet Address	Vesting: https://bscscan.com/address/0xF47893302E123D90f516B447d0C96B0e8FCEad23#code Token: https://bscscan.com/address/0xb18cf91B28957df955770386EEdf7f2e19A12431#code

Number of Issues per Severity



High	0 (0.00%)
Medium	1 (14.29%)
Low	3 (42.86%)
Informational	3 (42.86%)

Severity

Issues	High	Medium	Low	Informational
Open	0	0	0	0
Resolved	0	**1**	**2**	**3**
Acknowledged	0	0	**1**	0
Partially Resolved	0	0	0	0

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
- ☑ 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 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

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistical 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 Issues

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

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

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

Total Amount of token minted exceeds the expected total supply

Resolved

Path

Vesting.sol

Description

Based on the excel sheet provided by the Growt team, total suppply of Growt tokens(6 decimals) is expected to be 500000000000000 tokens.
But calculations in the Vesting smart contract mints 500000003952000 tokens in end of 67th month.

Recommendation

Consider to reevaluating the calculation in the smart contract or change the total supply.

Low Severity Issues

Missing zero address checks

Resolved

Path

Vesting.sol

Description

constructor of the contract is missing to check the input address params are not zero address.
- preSeed_
- team_
- marketing_
- liquidity_
- marketMaking_
- airDrop_
- preSeedReward_
- reserve_

Recommendation

Consider to add zero address checks for aforementioned addresses.

Missing Event emission

Resolved

Path

Vesting.sol

Description

releaseTokens function in the contract, mints new tokens to certain addresses stored in the contract. Therefore it is making state changes. State changing functions should emit events which is a best smart contract code practice.
Release event is defined in the contract but never used in the funtion.

```
event Release(
 address indexed userAddress,
 uint256 amount
);
```

Recommendation

Consider to emit the Release event in releaseTokens function

Not checking the return value

Acknowledged

Path

Vesting.sol

Description

releaseTokens function returns a boolean value to show the status of the token release. This function is invoked in constructor but return value is not checked.
As well check the return values of mint functions in the releaseTokens function.

Recommendation

Consider verifying the return value of the releaseTokens and mint functions.

Informational Severity Issues

Use !=0 instead of >0

Resolved

Path

Vesting.sol

Description

In releaseTokens function, Growt tokens are minted after checking the amount of tokens is more than zero

It is cheaper to use != 0 than > 0 for uint256 values .

```
function releaseTokens() public returns(bool, uint256 total) {
    ...
    // @audit uint256 is always greater than 0 so make it != 0 to be gas efficient
    if (preSeedAmount > 0) {
        released[preSeed] += preSeedAmount;
        _growtContract.mint(preSeed, preSeedAmount);
    }
    if (teamAmount > 0) {
        released[team] += teamAmount;
        _growtContract.mint(team, teamAmount);
    }
    if (marketingAmount > 0) {
        released[marketing] += marketingAmount;
        _growtContract.mint(marketing, marketingAmount);
    }
    if (liquidityAmount > 0) {
        released[liquidity] += liquidityAmount;
        _growtContract.mint(liquidity, liquidityAmount);
    }
    if (marketMakingAmount > 0) {
        released[marketMaking] += marketMakingAmount;
        _growtContract.mint(marketMaking, marketMakingAmount);
    }
    if (airDropAmount > 0) {
        released[airDrop] += airDropAmount;
        _growtContract.mint(airDrop, airDropAmount);
    }
    if (preSeedRewardAmount > 0) {
        released[preSeedReward] += preSeedRewardAmount;
        _growtContract.mint(preSeedReward, preSeedRewardAmount);
    }
    if (reserveAmount > 0) {
        released[reserve] += reserveAmount;
        _growtContract.mint(reserve, reserveAmount);
    }
    ...
```

Recommendation

Consider using != 0 instead of > 0 in the function.

Missing and incorrect docstrings

Resolved

Path

Vesting.sol

Description

constructor, releaseTokens and getAvailableTokenAmount functions miss some parts of docstrings. @param and @return tags provide rich documentation for functions about their parameters and return variables.
getAvailableTokenAmount function's @dev tag is misleading because it just returns the available number of tokens for a receiver.

```
/**
* @dev Function let release available tokens //@audit misleading explanation
*/
function getAvailableTokenAmount (uint8 vestingPeriod_, address receiver) public view re-
turns(uint256) {
```

Recommendation

Consider using the NATSPEC to complete the docstrings and change the incorrect docstring.

Use standard time literals for better readability

Resolved

Description

MONTH constant in the contract is not using the standard solidity (time units).(https://docs.soliditylang.org/en/latest/units-and-global-variables.html#time-units)

uint32 internal constant MONTH = 30 * 24 * 3600; // @audit use standard time units for better readability

Recommendation

Consider to change 30 * 24 * 3600 to 30 days in the contract.

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 Growt. We performed our audit according to the procedure described above.

Some issues of low,Informational and medium severity were found. Some suggestions, gas optimizations and best practices are also provided in order to improve the code quality and security posture. In the end, Growt Team Resolved almost all Issues and Acknowledged one.

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the submitted smart contract source code, including its compilation, deployment, and intended functionality.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

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

7+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

February, 2025

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com