GROW

SMART CONTRACT AUDIT REPORT

Prepared by:

BlockAudit

Date Of Enrollment:

November 09th, 2023 - November 17th, 2023

Visit : www.blockaudit.report

TABLE OF CONTENTS

INTRODUCTION 2-3

- Summary 2
- Overview 3

FINDINGS 4-15

- **Finding Overview 4**
- M-01 5
- L-01 6
- L-02 7
- L-03 8
- L-04 9
- L-05 10
- I-01 11
- I-02 12
- G-01 13
- G-02 14
- G-03 15

APPENDIX 16

DISCLAIMER 18

ABOUT 20

SUMMARY

This Audit Report mainly focuses on the extensive security of **GROW** Smart Contracts. With this report, we attempt to ensure the reliability and correctness of the smart contract by complete and rigorous assessment of the system's architecture and the smart contract codebase.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts producedby industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified inpublic;
- Provide more transparency on privileged activities once the protocol is live.

OVERVIEW

Project Summary

Project Name	GROW
Logo	
Platform	-
Language	Solidity
Code Link	-

File Summary

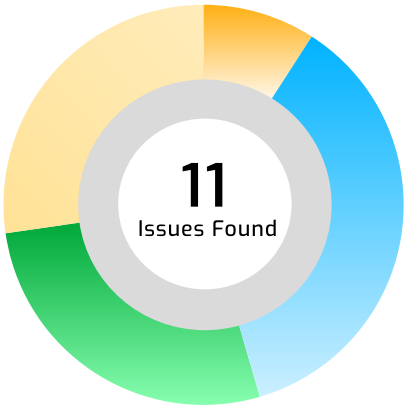
ID	File Name	Audit Status
GROW	Grow.sol	Pass

Audit Summary

Date of Delivery	17 November 2023
Audit Methodology	Code Analysis. Automatic Assesment, Manual Review
Audit Result	Passed
Audit Team	BlockAudit Report Team

FINDINGS

Critical	0	0.0%
High	0	0.0%
Medium	1	0.0%
Low	4	0.0%
Informational	3	0.0%
Ownership	0	0.0%
Gas Optimization	3	0.0%



Vulnerability Findings Summary

ID	Type	Instances	Severity	Status
M-01	Unsafe ERC20 Transfer Function Usage	289 / 291 / 428	Medium	**Resolved**
L-01	Use 2-step transfer of Ownership	7	Low	**Acknowledged**
L-02	Lack of 0-address check in setTreasuryWallet()	441 - 442	Low	**Acknowledged**
L-03	Avoid the use of floating pragma	3	Low	**Acknowledged**
L-04	Missing Event emission for multiple functions	-	Low	**Acknowledged**
L-05	Unused receive() function will lock Ether in contract	482	Informational	**Acknowledged**
I-01	Remove TODO	95 - 158 / 164 / 487	Informational	**Acknowledged**
I-02	Removal of Commented-Out Code for Better Code Quality	85	Informational	**Acknowledged**
G-01	Use Custom Errors instead of Revert Strings to save Gas	235 - 281 / 311 / 421 / 425	Gas Optimisation	**Acknowledged**
G-02	Use Pre-increment instead of Post-increment to save gas	212 / 218 / 224 / 230	Gas Optimisation	**Acknowledged**
G-03	Explicitly initializing variables with their default values wastes gas.	97 / 138 / 301	Gas Optimisation	**Acknowledged**

M-01

Type	Unsafe ERC20 Transfer Function Usage
Severity	Medium
File	Grow.sol
Line	289 / 291 / 428
Status	**Resolved**

Description

Some tokens (like USDT) don't correctly implement the EIP20 standard and their transfer/transferFrom function return void instead of a success boolean. Calling these functions with the correct EIP20 function signatures will always revert.
The ERC20.transfer() and ERC20.transferFrom() functions return a boolean value indicating success. This parameter needs to be checked for success. Some tokens do not revert if the transfer failed but return false instead.
Tokens that don't actually perform the transfer and return false are still counted as a correct transfer and tokens that don't correctly implement the latest EIP20 spec, like USDT, will be unusable in the protocol as they revert the transaction because of the missing return value.

Remediation

Recommend using OpenZeppelin's SafeERC20 versions with the safeTransfer and safeTransferFrom functions that handle the return value check as well as non-standard-compliant tokens.

Snapshot

```
TypeScript
  token.sol:
    289:             USDT.transferFrom(msg.sender, owner(), purchaseTotal);
    291:             USDC.transferFrom(msg.sender, owner(), purchaseTotal);
    428:         IERC20(swapPair).transfer(msg.sender, LpTokenBalance);
```

L-01

Type	Use 2-Step Transfer Of Ownership
Severity	Low
File	Grow.sol
Line	7
Status	**Acknowledged**

Description

Single-step ownership transfers add the risk of setting an unwanted owner by accident (this includes address(0)) if the ownership transfer is not done with excessive care.

Remediation

Consider employing 2 step ownership transfer mechanisms for this critical ownership.

Snapshot

```Solidity
import "@openzeppelin/contracts/access/Ownable.sol";
```

L-02

Type	Lack Of 0-Address Check In SetTreasuryWallet()
Severity	Low
File	Grow.sol
Line	441 - 442
Status	**Acknowledged**

Description

The setTreasuryWallet() function in the token.sol contract lacks a check for a zero address (0x0) when setting the setTreasuryWallet value. However, many other setter functions in the contract properly include a check for a zero address.

Remediation

Add a check for 0-addresses to the setTreasuryWallet() function.

Snapshot

```
Solidity
import "@openzeppelin/contracts/access/Ownable.sol";
```

L-03

Type	Avoid The Use Of Floating Pragma
Severity	Low
File	Grow.sol
Line	441 - 442
Status	**Acknowledged**

Description

Contracts should be deployed using the same compiler version/flags with which they have been tested. Locking the floating pragma, i.e. by not using ^ in pragma solidity ^0.8.17, ensures that contracts do not accidentally get deployed using an older compiler version with unfixed bugs.

Remediation

Remove ^ in "pragma solidity ^0.8.17" and change it to "pragma solidity 0.8.17" to be consistent with the rest of the contracts.

Snapshot

```
pragma solidity ^0.8.17;
```

L-04

Type	Missing Event Emission For Multiple Functions
Severity	Low
File	Grow.sol
Line	-
Status	**Acknowledged**

Description

Events are essential for non-contract tools to track changes and prevent users from being surprised by unexpected modifications. However, the current implementation of the contract lacks event emission for its functions.

Remediation

By emitting events, the contract can enhance its transparency, improve user experience, and facilitate composability.

Snapshot

None

L-05

Type	Unused Receive() Function Will Lock Ether In Contract
Severity	Informational
File	Grow.sol
Line	482
Status	**Acknowledged**

Description

The token.sol contract contains an unused receive() function. This function is payable, meaning that it can receive Ether. However, the function does not do anything with the Ether that it receives. This means that if a user sends Ether to the contract using the receive() function, the Ether will be locked in the contract and cannot be recovered.

Remediation

The following mitigation steps can be taken to address this vulnerability:

1. Remove the receive() function from the contract.
2. If the receive() function is needed, then modify the function to do something with the Ether that it receives.

Snapshot

```
token.sol:
  482:     receive() external payable {}
```

I-01

Type	Unused Receive() Function Will Lock Ether In Contract
Severity	Informational
File	Grow.sol
Line	85
Status	**Acknowledged**

Description

The token.sol contract contain TODO comments. These comments indicate that there are open questions or issues that need to be resolved before the contracts can be deployed.

Remediation

Remove the TODO comments or resolve them.

Snapshot

```
token.sol:
  85:     // TODO:
```

I-02

Type	Removal Of Commented-Out Code For Better Code Quality
Severity	Informational
File	Grow.sol
Line	85
Status	**Acknowledged**

Description

The token.sol contract contains lines of code that are commented out but serve no purpose. Commented-out code segments do not contribute to the functionality of the contracts but can clutter the codebase, making it harder for developers to understand the actual logic of the contracts. These lines of code should be removed to improve the overall code quality and clarity of the contracts.

Remediation

Remove the TODO comments or resolve them.

Snapshot

```
token.sol:
   95:     // uint256 constant maxMint = 50_000_000;
  100:     // uint256 maxBuy = 750_000;
  103:     // amounts for mainnet
  109:     // supply of tokens at each tier for TESTNET
  110:     // uint256 tier1 = 100;
  111:     // uint256 tier2 = 100;
  112:     // uint256 tier3 = 100;
  113:     // uint256 tier4 = 100;
  165:         // _mint(msg.sender, 10_000_000 * 10 ** decimals());
  406:         // _approve(address(this), address(swapRouter), tokenAmount)
  433:     // _addLockTime should be the number of seconds added to the current lock
  461:         // require(feeBuyTotal <= 100, "too much tax");
  475:         // require(feeSellTotal <= 100, "too much tax");
```

G-01

Type	Use Custom Errors Instead Of Revert Strings To Save Gas
Severity	Informational
File	Grow.sol
Line	235 - 281 / 311 / 421 / 425
Status	**Acknowledged**

Description

Custom errors from Solidity 0.8.4 are cheaper than revert strings (cheaper deployment cost and runtime cost when the revert condition is met)

Starting from Solidity v0.8.4, there is a convenient and gas-efficient way to explain to users why an operation failed through the use of custom errors. Until now, you could already use strings to give more information about failures (e.g., revert("Insufficient funds.");), but they are rather expensive, especially when it comes to deploy cost, and it is difficult to use dynamic information in them.

Custom errors are defined using the error statement, which can be used inside and outside of contracts (including interfaces and libraries).

Remediation

Use Custom errors instead of revert string to save gas

Snapshot

```
token.sol:
  235:         require(amt + amountSold <= currentMax, "Cannot buy more");
  240:         require(amt + amountSold <= currentMax, "Cannot buy more");
  245:         require(tokens > 0, "cannot buy zero");
  246:         require(amountSold < maxMint, "Sold Out");
  247:         require(amtBought[msg.sender] + tokens <= maxBuy, "cannot buy more");
  249:             require(block.timestamp <= nextChange, "Sale is over");
  256:         require(amountSold + tokens <= currentMax, "cannot buy more at this tier");
  259:         require(msg.value == purchaseTotal, "wrong purchase amt");
  265:         require(sent, "Failed to send Ether");
  270:         require(tokens > 0, "cannot buy zero");
  271:         require(amountSold < maxMint, "Sold Out");
  272:         require(amtBought[msg.sender] + tokens <= maxBuy, "cannot buy more");
  274:             require(block.timestamp <= nextChange, "Sale is over");
  281:         require(amountSold + tokens <= currentMax, "cannot buy more at this tier");
  311:             require(block.timestamp >= influencerLock, "Tokens locked");
  421:         require(block.timestamp >= timeOfDeployment + lockTime, "Lock duration has not Passed");
  425:         require (LpTokenBalance > 0, "Insufficient LP Tokens in the Contract");
```

G-02

Type	Use Pre-Increment Instead Of Post-Increment To Save Gas
Severity	Informational
File	Grow.sol
Line	212 / 218 / 224 / 230
Status	**Acknowledged**

Description

++i costs less gas compared to i++ or i += 1 for unsigned integer, as pre-increment is cheaper (about 5 gas per iteration). This statement is true even with the optimizer enabled.

Remediation

Use ++i instead of i++ to increment the value of an uint variable.

Snapshot

```
token.sol:
  212:              timeTier++;
  218:              timeTier++;
  224:              timeTier++;
  230:              timeTier++;
```

G-03

Type	Explicitly Initializing Variables With Their Default Values Wastes Gas.
Severity	Informational
File	Grow.sol
Line	97 / 138 / 301
Status	**Acknowledged**

Description

If a variable is not set/initialized, it is assumed to have the default value (0 for uint, false for bool, address(0) for address...). Explicitly initializing it with its default value is an anti-pattern and wastes gas.

As an example: uint256 i = 0 should be replaced with uint256 i;

Remediation

Remove explicit initializations for default values

Snapshot

```
token.sol:
   97:     uint256 public amountSold = 0;
  138:     uint256 timeTier = 0;
  301:         uint256 taxAmount = 0;
```

APPENDIX

Auditing Approach and Methodologies applied

The Block Audit Report team has performed rigorous testing of the project including the analysis of the code design patterns where we reviewed the smart contract architecture to ensure it is structured along with the safe use of standard inherited contracts and libraries. Our team also conducted a formal line by line inspection of the Smart Contract i.e., a manual review, to find potential issues including but not limited to

- Race conditions
- Zero race conditions approval attacks
- Re-entrancy
- Transaction-ordering dependence
- Timestamp dependence
- Check-effects-interaction pattern (optimistic accounting)
- Decentralized denial-of-service attacks
- Secure ether transfer pattern
- Guard check pattern
- Fail-safe mode
- Gas-limits and infinite loops
- Call Stack depth

In the Unit testing Phase, we coded/conducted custom unit tests written against each function in the contract to verify the claimed functionality from our client. In Automated Testing, we tested the Smart Contract with our standard set of multifunctional tools to identify vulnerabilities and security flaws. The code was tested in collaboration of our multiple team members and this included but not limited to;

- Testing the functionality of the Smart Contract to determine proper logic has been followed throughout the whole process.
- Analyzing the complexity of the code in depth and in detail line-by-line manual review of the code.
- Deploying the code on testnet using multiple clients to run live tests.
- Analyzing failure preparations to check how the Smart Contract performs in case of any bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analyzing the security of the on-chain data.

Issue Categories:

Every issue in this report was assigned a severity level from the following:

Critical Severity Issues

Issues of Critical Severity leaves smart contracts vulnerable to major exploits and can lead to asset loss and data loss. These can have significant impact on the functionality/performance of the smart contract.
We recommend these issues must be fixed before proceeding to MainNet..

High Severity Issues

Issues of High Severity are not as easy to exploit but they might endanger the execution of the smart contract and potentially create crucial problems.
Fixing these issues is highly recommended before proceeding to MainNet.

Medium Severity Issues

Issues on this level are not a major cause of vulnerability to the smart contract, they cannot lead to data-manipulations or asset loss but may affect funtionality.
It is important to fix these issues before proceeding to MainNet.

Low Severity Issues

Issues at this level are very low in their impact on the overall functionality and execution of the smart contract. These are mostly code-level violations or improper formatting.
These issues can be remain unfixed or can be fixed at a later date if the code is redeployed or forked.

Informational Findings

These are finding that our team comes accross when manually reviewing a smart contract which are important to know for the owners as well as users of a contract.
These issues must be acknowledged by the owners before we publish our report.

Ownership Priviledges

Owner of a smart contract can include certain rights and priviledges while deploying a smart contract that might be hidden deep inside the codebase and may make the project vulnerable to rug-pulls or other types of scams.
We at BlockAudit believe in transparency and hence we showcase Ownership priviledges separately so the owner as well as the investors can get a better understanding about the project.

Gas Optimization

Solidity gas optimization is the process of lowering the cost of operating your Solidity smart code. The term "gas" refers to the level of processing power required to perform specific tasks on the Ethereum network.
Each Ethereum transaction costs a fee since it requires the use of computer resources. It will deduct a fee anytime any function in the smart contract is invoked by the contract's owner or users.

DISCLAIMER

This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for the client to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that the client should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for the client to conduct the client's own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

By reading this report or any part of it, the client agrees to the terms of this disclaimer. If the client does not agree to the terms, then please immediately cease reading this report, and delete and destroy any/all copies of this report downloaded and/or printed by the client. This report is provided for information purposes only and stays on a non-reliance basis, and does not constitute investment advice. No one/ NONE shall have any rights to rely on the report or its contents, and BlockAudit and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives).

(BlockAudit) owes no duty of care towards the client or any other person, nor does BlockAudit claim any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and BlockAudit hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effects in relation to the report.

Except and only to the extent that it is prohibited by law, BlockAudit hereby excludes all liability and responsibility, and neither the client nor any other person shall have any claim against BlockAudit, for any amount or kind of loss or damage that may result to the client or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report.

The analysis of the security is purely based on the received smart contracts alone. No related/third-party smart contracts, applications or operations were reviewed for security. No product code has been reviewed.

Note: The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the **GROW** team put a bug bounty program in place to encourage further analysis of the smart contracts by other third parties

About BlockAudit

BlockAudit is an industry leading security organisation that helps web3 blockchain based projects with their security and correctness of their smart-contracts. With years of experience we have a dedicated team that is capable of performing audits in a wide variety of languages including HTML, PHP, JS, Node, React, Native, Solidity, Rust and other Web3 frameworks for DApps, DeFi, GameFi and Metaverse platforms.

With a mission to make web3 a safe and secure place BlockAudit is commited to provide it's partners with a budget and investor friendly security Audit Report that will increase the value of their projects significantly.

 *www.blockaudit.report*

 *team@blockaudit.report*

 *@BlockAudit*

 *github.com/Block-Audit-Report*