SHELLREBELS

SMART CONTRACT AUDIT REPORT

Prepared by:

BlockAudit

Date Of Enrollment:

December 14th, 2022 - December 16th, 2022

Visit : www.blockaudit.report

TABLE OF CONTENTS

INTRODUCTION 2-3

- Summary 2
- Overview 3

FINDINGS 4-11

- **Finding Overview 4**
- SWL01 5
- SWL02 7
- SWL03 8
- SWL04 9
- SWL05 10
- SWL06 11
- SWL07 12
- SWL08 13
- SWL09 14
- SWL10 15
- SWL11 16
- SWL12 17

APPENDIX 18

DISCLAIMER 20

ABOUT 22

SUMMARY

This Audit Report mainly focuses on the extensive security of **SHELLREBELS** 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	SHELLREBELS
Logo	
Platform	BSC
Language	Solidity
Code Link	https://bscscan.com/address/0x700f75f3c8dbeaea25068f477e68083dc7cfc41d#code

File Summary

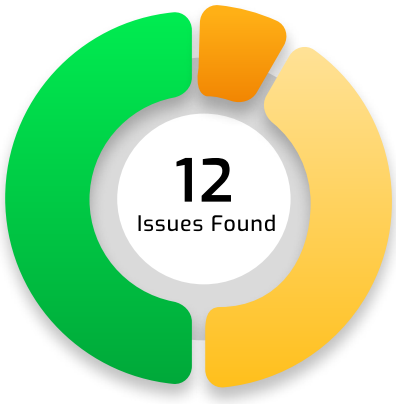
ID	File Name	Audit Status
SWL	SWO-Token.sol	Pass

Audit Summary

Date of Delivery	16 Dec 2022
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	8.34%
Low	5	41.66%
Informational	0	0.0%
Ownership	0	0.0%
Gas Optimization	6	50.0%



Vulnerability Findings Summary

ID	Type	Line	Severity	Status
SWL01	Centralisation Risk	871-940	Medium	Acknowledged
SWL02	Floating pragma	10	Low	Acknowledged
SWL03	Use A More Recent Version Of Solidity	10	Low	Acknowledged
SWL04	Remove Commented Out Codes	389	Low	Acknowledged
SWL05	No Use Of Safemath	3	Low	Acknowledged
SWL06	Two Step Ownership Recommended	342-346	Low	Acknowledged
SWL07	Using Custom Error	329/339/348/373/582/583/602/623/650/651/940/941/950/951/952	Gas Optimization	Acknowledged
SWL08	++i costs less gas compared to i++ or i += 1 (same for --i vs `i-- or i -= 1)	860	Gas Optimization	Acknowledged
SWL09	`ARRAY.LENGTH` SHOULD NOT BE LOOKED UP IN EVERY LOOP OF A FOR-LOOP	860	Gas Optimization	Acknowledged
SWL10	`X += Y` Costs More Gas Than `X = X + Y` For State Variables	1050/1054/1058/1062	Gas Optimization	Acknowledged
SWL11	Using > 0 costs more gas than != 0 when used on a uint in a require() statement	383	Gas Optimization	Acknowledged
SWL12	IT COSTS MORE GAS TO INITIALIZE VARIABLES WITH THEIR DEFAULT VALUE THAN LETTING THE DEFAULT VALUE BE APPLIED	707/710/775/776	Gas Optimization	Acknowledged

SWL01

Type	Centralisation Risk
Severity	Medium
File	SWL.sol
Line	871-940
Status	**Acknowledged**

Description

The owner can manipulate fees like foundation fees , marketing fee , liquidity fee , their is no lower limit.

Remediation

Always remember a contract should contain lower limit upper limit to set the fees

Snapshot

```
function setMarketingWallet(address payable wallet) external onlyOwner{
    marketingWalletAddress = wallet;
}


function setFoundationWallet(address addr) public onlyOwner {
    foundationWalletAddress = addr;
}


function isExcludedFromFees(address account) public view returns(bool) {
    return _isExcludedFromFees[account];
}

function setDividendExempt(address val) public onlyOwner {
    isDividendExempt[val] = true;
}

function setBlack(address val) public onlyOwner{
    _isBlackList[val] = true;
}

function unBlack(address val) public onlyOwner{
    _isBlackList[val] = false;
}


function setSwapTokensAtAmount(uint256 amount) public onlyOwner {
    swapTokensAtAmount = amount;
}


function setLiquidityFee(uint256 val) public onlyOwner {
    liquidityFee = val;
}


function setMarketingFee(uint256 val) public onlyOwner {
    marketingFee = val;
}


function setFoundationFee(uint256 val) public onlyOwner {
    foundationFee = val;
}


function setDeadFee(uint256 val) public onlyOwner {
    deadFee = val;
}


function setLpRewardFee(uint256 val) public onlyOwner {
    lpRewardFee = val;
}


function setMinPeriod(uint256 number) public onlyOwner {
    minPeriod = number;
}


function setLiquidityReceiveAddress(address val) public onlyOwner {
    liquidityReceiveAddress = val;
}


function resetLPRewardLastSendTime() public onlyOwner {
    dividendTracker.resetLPRewardLastSendTime();
}
```

SWL02

Type	Floating Pragma
Severity	Low
File	SWL.sol
Line	10
Status	**Acknowledged**

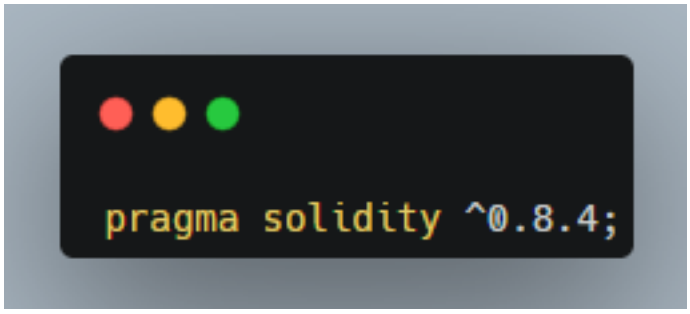
Description

In the contracts, floating pragmas should not be used. Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively..

Remediation

Use Fix Version of Solidity.

Snapshot



SWL03

Type	Use a more recent version of solidity
Severity	Low
File	SWL.sol
Line	10
Status	**Acknowledged**

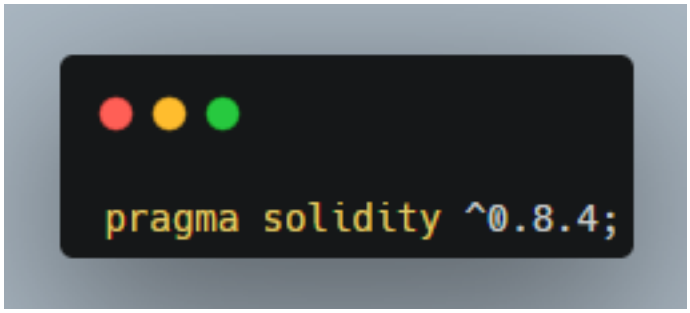
Description

Use a solidity version of at least 0.8.10 to have external calls skip contract existence checks if the external call has a return value

Remediation

Use the latest Solidity version at least 0.8.16

Snapshot



SWL04

Type	Remove Commented Out Codes
Severity	Low
File	SWL.sol
Line	389
Status	**Acknowledged**

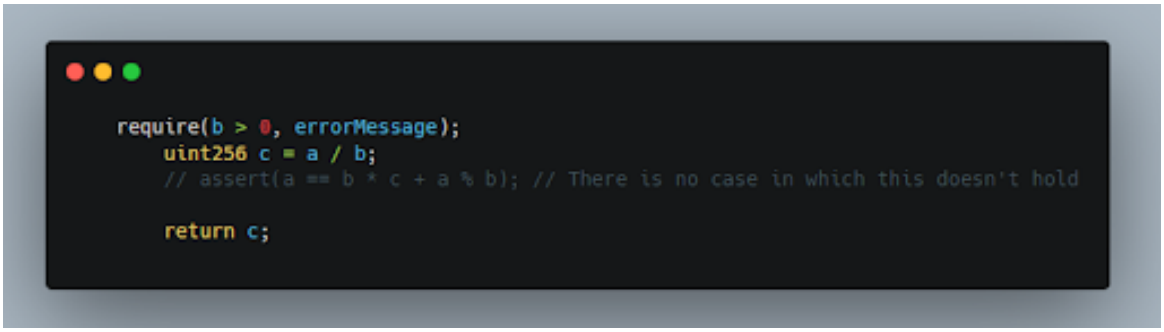
Description

Before deploying a contract, one should check for the commented codes since their principles don't work.

Remediation

Remove Commented out codes

Snapshot



```
    require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
```

SWL05

Type	Using safemath for solidity version 8.4 and above is redundant
Severity	Low
File	SWL.sol
Line	349
Status	**Acknowledged**

Description

Using safemath for solidity version 8.4 and above is redundant

Remediation

Ignore using Safemath library for solidity version 0.8.4 and above

Snapshot

```
library SafeMath {
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
```

SWL06

Type	Two step transfer of Ownership is recommended
Severity	Low
File	SWL.sol
Line	342-346
Status	**Acknowledged**

Description

This function checks the new owner is not the zero address and proceeds to write the new owner's address into the owner's state variable. If the nominated EOA account is not a valid account, it is entirely possible the owner may accidentally transfer ownership to an uncontrolled account, breaking all functions with the onlyOwner() modifier. Lack of two-step procedure for critical operations leaves them error-proneif the address is incorrect, the new address will take on the functionality of the new role immediately

Remediation

Consider implementing a two step process where the owner nominates an account and the nominated account needs to call an acceptOwnership() function for the transfer of ownership to fully succeed. This ensures the nominated EOA account is a valid and active account.

Snapshot

```
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(newOwner != address(0), "Ownable: new owner is the zero address");
    emit OwnershipTransferred(_owner, newOwner);
    _owner = newOwner;
}
```

SWL07

Type	Using of Custom Error is recommended
Severity	Gas Optimisation
File	SWL.sol
Line	329/339/348/373/582/583/602/623/650/651/940/941/950/951/952
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

Using of Custom Error is recommended

Snapshot

```
329:        require(owner() == _msgSender(), "Ownable: caller is not the owner");
339:        require(newOwner != address(0), "Ownable: new owner is the zero address");
348:        require(c >= a, "SafeMath: addition overflow");
373:        require(c / a == b, "SafeMath: multiplication overflow");
582:        require(sender != address(0), "ERC20: transfer from the zero address");
583:        require(recipient != address(0), "ERC20: transfer to the zero address");
602:        require(account != address(0), "ERC20: mint to the zero address");
623:        require(account != address(0), "ERC20: burn from the zero address");
650:        require(owner != address(0), "ERC20: approve from the zero address");
651:        require(spender != address(0), "ERC20: approve to the zero address");
940:        require(newValue >= 300000 && newValue <= 750000, "distributorGas must be
between 300,000 and 750,000");
941:        require(newValue != distributorGas, "Cannot update distributorGas to same
value");
950:        require(from != address(0), "ERC20: transfer from the zero address");
951:        require(to != address(0), "ERC20: transfer to the zero address");
952:        require(!_isBlackList[from], "ERC20: transfer to the zero address");
```

SWL08

Type	`++i` costs less gas compared to `i++` or `i += 1` (same for `--i` vs `i--` or `i -= 1`)
Severity	Gas Optimisation
File	SWL.sol
Line	860
Status	**Acknowledged**

Description

++i costs less gas as compared to i++ for unsigned integer, as per-increment is cheaper (it's about 5 gas per iteration cheaper)

Snapshot

SWL09

Type	`ARRAY.LENGTH` SHOULD NOT BE LOOKED UP IN EVERY LOOP OF A FOR-LOOP
Severity	Gas Optimisation
File	SWL.sol
Line	860
Status	**Acknowledged**

Description

Reading array length at each iteration of the loop takes 6 gas (3 for mload and 3 to place memory_offset) in the stack.Caching the array length in the stack saves around 3 gas per iteration.

Remediation

Storing the array's length in a variable before the for-loop, and use it instead .

Snapshot

SWL10

Type	`x += y` costs more gas than `x = x + y` for state variables
Severity	Gas Optimisation
File	SWL.sol
Line	1050/1054/1058/1062
Status	**Acknowledged**

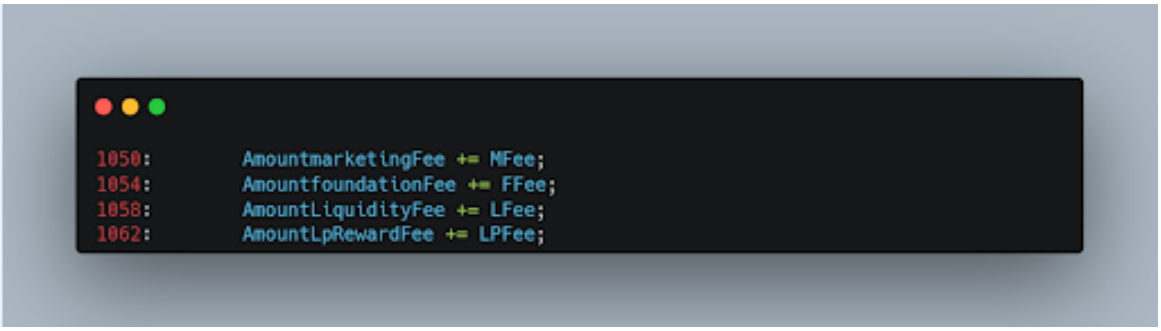
Description

`x += y` costs more gas than `x = x + y` for state variables

Remediation

Use of `x = x + y` instead is recommended

Snapshot



SWL11

Type	Using > 0 costs more gas than != 0 when used on a uint in a require() statement
Severity	Gas Optimisation
File	SWL.sol
Line	383
Status	**Acknowledged**

Description

> 0 is less efficient than != 0 for unsigned integers (with proof)
!= 0 costs less gas compared to > 0 for unsigned integers in require statements with the optimizer enabled (6 gas) Proof: While it may seem that > 0 is cheaper than !=, this is only true without the optimizer enabled and outside a require statement. If you enable the optimizer at 10k AND you're in a require statement, this will save gas

Remediation

Changing > 0 with != 0. Also, please enable the Optimizer.

Snapshot

```
383:        require(b > 0, errorMessage);
```

SWL12

Type	Variables: No need to explicitly initialize variables with default values
Severity	Gas Optimisation
File	SWL.sol
Line	707/710/775/776
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.

Remediation

We can use `uint number;` instead of `uint number = 0;`

Snapshot

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.

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 **SHELLREBELS** 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