RD AUDITORS

FARMING SMART CONTRACT, CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: WallStreetBets
Prepared on: 27 May 2021
Platform: Binance Smart Chain
Language: Solidity

TABLE OF CONTENTS

THIS DOCUMENT MAY CONTAIN CONFIDENTIAL INFORMATION ABOUT ITS SYSTEMS AND INTELLECTUAL PROPERTY OF THE CUSTOMER AS WELL AS INFORMATION ABOUT POTENTIAL VULNERABILITIES AND METHODS OF THEIR EXPLOITATION.

THE REPORT CONTAINING CONFIDENTIAL INFORMATION CAN BE USED INTERNALLY BY THE CUSTOMER OR IT CAN BE DISCLOSED PUBLICLY AFTER ALL VULNERABILITIES ARE FIXED - UPON DECISION OF CUSTOMER.

Document

Name	Farming Smart Contract Code Review and Security Analysis Report of WallStreetBets
Platform	BSC / Solidity
File 1	StratSimple.sol
MD5 hash	2B88E801F6EDEB700E9C82D148A981C0
SHA256 hash	16E2A1BD060E95347402A5F005FBBECCC14794C483223BE642F15FD2DB43A28C
File 2	StratWSBToken.sol
MD5 hash	D4395224BBA6A7AC817C9FAF287C1E17
SHA256 hash	AD55976AEEFD1C6F805EDB72A2E1B1DBAD870ECC93132F234B0E9645A4FEDC18
File 3	WSB_JPOW+DarkPool.sol
MD5 hash	CC64A9A3DE11E5DD2875406D232688EE
SHA256 hash	9C71CEDB08CC20D54AB7610854A8204C7373C965712838F99F151D3A251185D7
Date	27/05/2021

Introduction

RD Auditors (Consultant) was contracted by WallStreetBets (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report represents the findings of the security assessment of the customer`s smart contracts and its code review conducted between 20 - 27 May 2021.

This contract consists of three files.

Project Scope

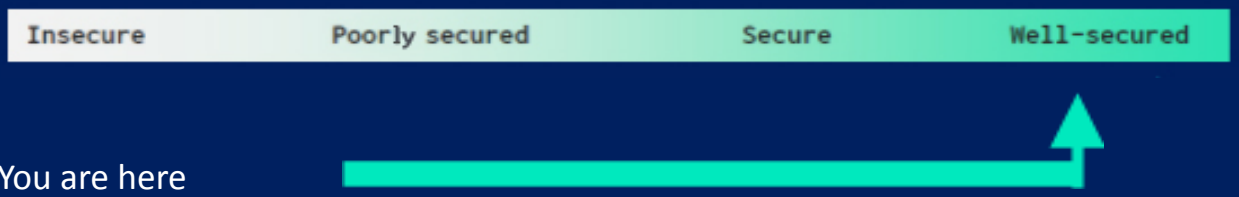
The scope of the project is a smart contract.

We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but is not limited to):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level
- Flash loan attack

Executive Summary

According to the assessment, the customer's solidity smart contract is well secured.



Automated checks are with smartDec, Mythril, Slither and remix IDE. All issues were performed by our team, which included the analysis of code functionality, manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the AS-IS section and all issues found are located in the audit overview section.

We found 0 critical, 0 high, 0 medium, 0 low and 0 very low level issues.

Code Quality

Please find a link that, within this report, contains SafeMath, Address, ownable, safeERC20 and Enumerableset from the popular open source.

The libraries within this smart contract are part of a logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned to a specific address and its properties/methods can be reused many times by other contracts.

The WallStreetBets team has also conducted unit tests using scripts provided through the same github link which fortify functionality and security of the contract, which also helped us to determine the integrity of the code in an automated way.

Overall, the code is well commented. Commenting provides rich documentation for functions, return variables and more and also helps auditors to promptly cover the flow behind code logic. Use of Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

We were given the WallStreetBets Farming Smart Contract as a zip file:

The hash of that file is mentioned in the table. As mentioned above, It's recommended to write comments in the smart contract code, so anyone can quickly understand the programming flow as well as complex code logic.

Comments are very helpful in understanding the overall architecture of the protocol. It also provides a clear overview of the system components, including helpful details, like the lifetime of the background script.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure. Those were based on well known industry standard open source projects and even core code blocks that are written well and systematically.

AS-IS Overview

WSB DApp solves the fees and market manipulation problem that have solely benefited centralized financial institutions for the past 100 Years.

File And Function Level Report

File: StratSimple.sol

Contract: ERC20
Inherit: Context, IERC20
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	name	read	Passed	All Passed	No Issue	Passed
2	symbol	read	Passed	All Passed	No Issue	Passed
3	decimals	read	Passed	All Passed	No Issue	Passed
4	totalSupply	read	Passed	All Passed	No Issue	Passed
5	balanceOf	read	Passed	All Passed	No Issue	Passed
6	transfer	write	Passed	All Passed	No Issue	Passed
7	allowance	read	Passed	All Passed	No Issue	Passed
8	approve	write	Passed	All Passed	No Issue	Passed
9	transferFrom	write	Passed	All Passed	No Issue	Passed
10	increaseAllowance	write	Passed	All Passed	No Issue	Passed
11	decreaseAllowance	write	Passed	All Passed	No Issue	Passed
12	_transfer	write	Passed	All Passed	No Issue	Passed
13	_mint	write	Passed	All Passed	No Issue	Passed

14	_burn	write	Passed	All Passed	No Issue	Passed
15	_approve	write	Passed	All Passed	No Issue	Passed
16	_setUpDecimals	write	Passed	All Passed	No Issue	Passed
17	_beforeTokenTransfer	write	Passed	All Passed	No Issue	Passed

Contract: Ownable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	owner	read	Passed	All Passed	No Issue	Passed
2	renounceownership	write	Passed	All Passed	No Issue	Passed
3	transferOwnership	write	Passed	All Passed	No Issue	Passed

Contract: Pausable
Inherit: Context
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	Paused	read	Passed	All Passed	No Issue	Passed
2	_Pause	write	Passed	All Passed	No Issue	Passed
3	_unpause	write	Passed	All Passed	No Issue	Passed

Contract: StratSimple
Inherit: Ownable, ReentrancyGuard, Pausable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	deposit	write	Passed	All Passed	No Issue	Passed
2	withdraw	write	Passed	All Passed	No Issue	Passed
3	pause	write	Passed	All Passed	No Issue	Passed
4	unpause	write	Passed	All Passed	No Issue	Passed
5	setEntranceFeeFactor	write	Passed	All Passed	No Issue	Passed
6	setGov	write	Passed	All Passed	No Issue	Passed
7	IncaseTokensGetStuck	write	Passed	All Passed	No Issue	Passed

File: StratWSBToken.sol

Contract: ERC20
Inherit: Context, IERC20
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	name	read	Passed	All Passed	No Issue	Passed
2	symbol	read	Passed	All Passed	No Issue	Passed
3	decimals	read	Passed	All Passed	No Issue	Passed
4	totalSupply	read	Passed	All Passed	No Issue	Passed
5	balanceOf	read	Passed	All Passed	No Issue	Passed
6	transfer	write	Passed	All Passed	No Issue	Passed
7	allowance	read	Passed	All Passed	No Issue	Passed
8	approve	write	Passed	All Passed	No Issue	Passed

9	transferFrom	write	Passed	All Passed	No Issue	Passed
10	increaseAllowance	write	Passed	All Passed	No Issue	Passed
11	decreaseAllowance	write	Passed	All Passed	No Issue	Passed
12	_transfer	write	Passed	All Passed	No Issue	Passed
13	_mint	write	Passed	All Passed	No Issue	Passed
14	_burn	write	Passed	All Passed	No Issue	Passed
15	_approve	write	Passed	All Passed	No Issue	Passed
16	_setUpDecimals	write	Passed	All Passed	No Issue	Passed
17	_beforeTokenTransfer	write	Passed	All Passed	No Issue	Passed

Contract: Ownable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	owner	read	Passed	All Passed	No Issue	Passed
2	renounceownership	write	Passed	All Passed	No Issue	Passed
3	transferOwnership	write	Passed	All Passed	No Issue	Passed

Contract: Pausable
Inherit: Context
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	Paused	read	Passed	All Passed	No Issue	Passed
2	_Pause	write	Passed	All Passed	No Issue	Passed
3	_unpause	write	Passed	All Passed	No Issue	Passed

Contract: StratWSBToken
Inherit: Ownable, ReentrancyGuard, Pausable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	deposit	write	Passed	All Passed	No Issue	Passed
2	withdraw	write	Passed	All Passed	No Issue	Passed
3	pause	write	Passed	All Passed	No Issue	Passed
4	unpause	write	Passed	All Passed	No Issue	Passed
5	setEntranceFeeFactor	write	Passed	All Passed	No Issue	Passed
6	setGov	write	Passed	All Passed	No Issue	Passed
7	IncaseTokensGetStuck	write	Passed	All Passed	No Issue	Passed
8	setExitFeeFactor	write	Passed	All Passed	No Issue	Passed

File: WSB_JPOW+DarkPool.sol

Contract: Context
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	_msgSender	read	Passed	All Passed	No Issue	Passed
2	_msgData	read	Passed	All Passed	No Issue	Passed

Contract: ERC20
Inherit: context, IERC20
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	name	read	Passed	All Passed	No Issue	Passed
2	symbol	read	Passed	All Passed	No Issue	Passed
3	decimals	read	Passed	All Passed	No Issue	Passed
4	totalSupply	read	Passed	All Passed	No Issue	Passed
5	balanceOf	read	Passed	All Passed	No Issue	Passed
6	transfer	write	Passed	All Passed	No Issue	Passed
7	allowance	read	Passed	All Passed	No Issue	Passed
8	approve	write	Passed	All Passed	No Issue	Passed
9	transferFrom	write	Passed	All Passed	No Issue	Passed
10	increaseAllowance	write	Passed	All Passed	No Issue	Passed
11	decreaseAllowance	write	Passed	All Passed	No Issue	Passed
12	_transfer	write	Passed	All Passed	No Issue	Passed
13	_mint	write	Passed	All Passed	No Issue	Passed
14	_burn	write	Passed	All Passed	No Issue	Passed
15	_approve	write	Passed	All Passed	No Issue	Passed
16	_setUpDecimals	write	Passed	All Passed	No Issue	Passed
17	_beforeTokenTransfer	write	Passed	All Passed	No Issue	Passed

Contract: Ownable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	owner	read	Passed	All Passed	No Issue	Passed
2	renounceownership	write	Passed	All Passed	No Issue	Passed
3	transferOwnership	write	Passed	All Passed	No Issue	Passed

Contract: JPOW
Inherit: Ownable, ReentrancyGuard
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	PoolLength	read	Passed	All Passed	No Issue	Passed
2	add	write	Passed	All Passed	No Issue	Passed
3	set	write	Passed	All Passed	No Issue	Passed
4	getMultiplier	read	Passed	All Passed	No Issue	Passed
5	PendingWSB	read	Passed	All Passed	No Issue	Passed
6	StakeWantTokens	read	Passed	All Passed	No Issue	Passed
7	massUpdatePools	write	Passed	All Passed	No Issue	Passed
8	UpdatePool	write	Passed	All Passed	No Issue	Passed
9	deposit	write	Passed	All Passed	No Issue	Passed
10	withdraw	write	Passed	All Passed	No Issue	Passed
11	emergencyWithdraw	write	Passed	All Passed	No Issue	Passed
12	safeWSBTransfer	write	Passed	All Passed	No Issue	Passed
13	getWSB	write	Passed	All Passed	No Issue	Passed
14	SetFundSource	write	Passed	All Passed	No Issue	Passed
15	incaseTokensGetStuck	write	Passed	All Passed	No Issue	Passed
16	SetWSBPerBlock	write	Passed	All Passed	No Issue	Passed

Contract: DarkPool
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	transferERC20	write	Passed	All Passed	No Issue	Passed

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to lost tokens etc.
High	High level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial functions.
Medium	Medium level vulnerabilities are important to fix; however, they cannot lead to lost tokens.
Low	Low level vulnerabilities are most related to outdated, unused etc. These code snippets cannot have a significant impact on execution.
Lowest Code Style/ Best Practice	Lowest level vulnerabilities, code style violations and information statements cannot affect smart contract execution and can be ignored.

Audit Findings

Critical

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

No medium severity vulnerabilities were found.

Low

No low severity vulnerabilities were found.

Very Low

No very low severity vulnerabilities were found.

Discussion

File: WSB_JPOW+DarkPool.sol

1. Hardcoded addresses/bytes should be double checked before deploying for production.

2. Line 1449 burnAddress has not been used and it can be removed.

```
1445
1446    address public WSB = 0x22168882276e5D5e1da694343b41DD7726eeb288;
1447    address public fundSource; //source of WSB tokens to pull from
1448
1449    address public burnAddress = 0x000000000000000000000000000000000000dEaD;
1450
```

3. Concerning “massUpdatePools” the developer is aware of “gas spending” for loops. This can be written as a loop with batch processing.

```
1561    // Update reward variables for all pools. Be careful of gas spending!
1562    function massUpdatePools() public {
1563        uint256 length = poolInfo.length;
1564        for (uint256 pid = 0; pid < length; ++pid) {
1565            updatePool(pid);
1566        }
1567    }
```

Note: In view of the most recent flash loan attacks, as far as this contract is concerned, it has no such vulnerability.

Conclusion

We were given a contract file and have used all possible tests based on the given object. The contract is written systematically, so it is ready to go for production.

Since possible test cases can be unlimited and developer level documentation (code flow diagram with function level description) not provided, for such an extensive smart contract protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

The security state of the reviewed contract is now “well secured”

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyse the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

RD Auditors Disclaimer

The smart contracts given for audit have been analysed in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Because the total number of test cases are unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

RD
AUDITORS

Email: info@rdauditors.com

Website: www.rdauditors.com