OBELISK

Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 21	2021-06-24	Zapmore, Plemonade	Audit Final

Audit Notes

Audit Date	2021-06-16 - 2021-06-24
Auditor/Auditors	Plemonade, Hebilicious
Auditor/Auditors Contact Information	tibereum-obelisk@protonmail.com
Notes	Specified code and contracts are audited for security flaws. UI/UX (website), logic, team, and tokenomics are not audited.
Audit Report Number	OB58866657

Disclaimer

This audit is not financial, investment, or any other kind of advice and is for informational purposes only. This report is not a substitute for doing your own research and due diligence. Obelisk is not responsible or liable for any loss, damage, or otherwise caused by reliance on this report for any purpose. Obelisk has based this audit report solely on the information provided by the audited party and on facts that existed before or during the audit being conducted. Obelisk is not responsible for any outcome, including changes done to the contract/contracts after the audit was published. This audit is fully objective and only discerns what the contract is saying without adding any opinion to it. The audit is paid by the project but neither the auditors nor Obelisk has any other connection to the project and has no obligations other than to publish an objective report. Obelisk will always publish its findings regardless of the outcome of the findings. The audit only covers the subject areas detailed in this report and unless specifically stated, nothing else has been audited. Obelisk assumes that the provided information and material were not altered, suppressed, or misleading. This report is published by Obelisk, and Obelisk has sole ownership of this report. Use of this report for any reason other than for informational purposes on the subjects reviewed in this report including the use of any part of this report is prohibited without the express written consent of Obelisk.

Obelisk Auditing

Defi is a relatively new concept but has seen exponential growth to a point where there is a multitude of new projects created every day. In a fast-paced world like this, there will also be an enormous amount of scams. The scams have become so elaborate that it's hard for the common investor to trust a project, even though it could be legit. We saw a need for creating high-quality audits at a fast phase to keep up with the constantly expanding market. With the Obelisk stamp of approval, a legitimate project can easily grow its user base exponentially in a world where trust means everything. Obelisk Auditing consists of a group of security experts that specialize in security and structural operations, with previous work experience from among other things, PricewaterhouseCoopers. All our audits will always be conducted by at least two independent auditors for maximum security and professionalism.

As a comprehensive security firm, Obelisk provides all kinds of audits and project assistance.

Table of Content

Version Notes 2

Audit Notes 2

Disclaimer 2

Obelisk Auditing 3

Project Information 5

Executive Summary 6

Summary Table 7

Introduction 8

Findings 9

Manual Analysis 9

Non-Withdrawable BNB 9

Operator Permissions 11

Transfer Always Taxes 13

Liquidity Distributed To Operator 14

Timelock Bypass 15

Static Analysis 16

No Findings 16

On-Chain Analysis 17

Operator Is Externally Owned Address (EOA) 17

Appendix A - Reviewed Documents 18

Appendix B - Risk Ratings 19

Appendix C - Icons 19

Appendix D - Testing Standard 20

Project Information

Project Name	DSBSwap
Description	DSBSwap is a next-generation automated liquidity mining decentralized exchange running cross-platform with super unique and creative features.
Website	https://dsbswap.com/
Contact	Join DSBTeam
Contact information	https://t.me/DSBSwap_eng on TG
Token Name(s)	Digital Super Bit
Token Short	DSB
Contract(s)	See Appendix A
Code Language	Solidity
Chain	BSC

Executive Summary

The audit of DSBSwap was conducted by two of Obelisks' security experts between the 16th of June 2021 and the 24th of June 2021.

Only the Token contract of the DSBSwap project has been audited by Obelisk. After finishing the full audit, Obelisk auditing can say that there were multiple medium-risk findings that could be a problem. After presenting these findings to the project team, they worked swiftly to solve the issues that could be solved and worked to add a timelock, which they did.

The team has not reviewed the UI/UX, logic, team, or tokenomics of the DSBSwap project.

Please read the full document for a complete understanding of the audit.

Summary Table

Audited Part	Severity	Note
Non-Withdrawable BNB	Low Risk	See Comment
Operator Permissions	Medium / Low Risk	Resolved
Transfer Always Taxes	Informational	See Comment
Liquidity Distributed To Operator	Medium Risk	See Comment
Timelock Bypass	Medium Risk	Resolved
Operator Is Externally Owned Address (EOA)	Medium Risk	Resolved

Introduction

Obelisk was commissioned by DSBSwap on the 15th of June 2021 to conduct a comprehensive audit of DSBSwaps token contract. The following audit was conducted between the 16th of June 2021 and the 24th of June 2021 and delivered on the 24th of June 2021. Two of Obelisk's security experts went through the related contracts using industry standards to find if any vulnerabilities could be exploited.

The comprehensive test was conducted in a specific test environment that utilized exact copies of the published contract. The auditors also conducted a manual visual inspection of the code to find security flaws that automatic tests would not find.

While conducting the audit, the Obelisk security team uses best practices to ensure that the reviewed contracts are thoroughly examined against all angles of attack. This is done by evaluating the codebase and whether it gives rise to significant risks. During the audit, Obelisk assesses the risks and assigns a risk level to each section together with an explanatory comment. Take note that the comments from the project team are their opinion and not the opinion of Obelisk.

After auditing the token contract of DSBSwap, we found two issues that both could be classified as Medium Severity and are both regarding the operator. They are stated as Medium Severity because the operator has permissions that could be dangerous to users as the permissions are vast. The operator is also an Externally Owned Address that is controlled by a person instead of a contract. This without a timelock could pose a risk to users. The project team has added a timelock that solved the operator issues for the future.

There was also a risk with “Liquidity Distributed To Operator” in which the Operator will control the liquidity, and depending on the project’s promise and information, this could be a risk to users if this behavior is against the documentation. This was also solved with a timelock.

Another finding is a Low Severity finding that involves the permanent loss of BNB Liquidity. The loss of BNB Liquidity is disabled and as long as it stays so, it is resolved.

The informational finding poses no risk to the user but is good to know while interacting with the token.

The old timelock had a security flaw, but the new timelock that the team implemented works as it should.

Please see each section of the audit to get a full understanding of the audit.

Findings

Manual Analysis

Non-Withdrawable BNB

SEVERITY	Low Risk
RESOLVED	**Partially**
LOCATION	DSBToken.sol -> 134-164

```
1    function swapAndLiquify() private lockTheSwap transferTaxFree {
2         uint256 contractTokenBalance = balanceOf(address(this));
3         uint256 maxTransferAmount = maxTransferAmount();
4         contractTokenBalance = contractTokenBalance > maxTransferAmount ? maxTransferAmount :
  contractTokenBalance;
5
6         if (contractTokenBalance >= minAmountToLiquify) {
7             // only min amount to liquify
8             uint256 liquifyAmount = minAmountToLiquify;
9
10            // split the liquify amount into halves
11            uint256 half = liquifyAmount.div(2);
12            uint256 otherHalf = liquifyAmount.sub(half);
13
14            // capture the contract's current ETH balance.
15            // this is so that we can capture exactly the amount of ETH that the
16            // swap creates, and not make the liquidity event include any ETH that
17            // has been manually sent to the contract
18            uint256 initialBalance = address(this).balance;
19
20            // swap tokens for ETH
21            swapTokensForEth(half);
22
23            // how much ETH did we just swap into?
24            uint256 newBalance = address(this).balance.sub(initialBalance);
25
26            // add liquidity
27            addLiquidity(otherHalf, newBalance);
28
29            emit SwapAndLiquify(half, newBalance, otherHalf);
30        }
31    }
32
```

DESCRIPTION	When swapping for BNB in order to add liquidity, the ratio of tokens in the liquidity pool is changed. Thus, when adding to liquidity, a small amount of BNB could be leftover. This BNB cannot be retrieved.
RECOMMENDATION	Distribute the leftover BNB to holders or swap it back to DSBToken.
MITIGATED/COMMENT	Project Comment: “Since the contract is deployed on the mainnet, we asked the community's opinion and already

	disabled the liquidity swap feature, so it will not happen anymore.” Obelisk: As long as it stays disabled this can be seen as resolved.

Operator Permissions

SEVERITY	Medium/Low Risk
RESOLVED	**Yes**
LOCATION	DSBToken.sol -> 222-283

```
    function updateTransferTaxRate(uint16 _transferTaxRate) public onlyOperator {
        require(_transferTaxRate <= MAXIMUM_TRANSFER_TAX_RATE, "DSB::updateTransferTaxRate: Transfer tax rate must not exceed the maximum rate.");
        emit TransferTaxRateUpdated(msg.sender, transferTaxRate, _transferTaxRate);
        transferTaxRate = _transferTaxRate;
    }

    /**
     * @dev Update the burn rate.
     * Can only be called by the current operator.
     */
    function updateBurnRate(uint16 _burnRate) public onlyOperator {
        require(_burnRate <= 100, "DSB::updateBurnRate: Burn rate must not exceed the maximum rate.");
        emit BurnRateUpdated(msg.sender, burnRate, _burnRate);
        burnRate = _burnRate;
    }

    /**
     * @dev Update the max transfer amount rate.
     * Can only be called by the current operator.
     */
    function updateMaxTransferAmountRate(uint16 _maxTransferAmountRate) public onlyOperator {
        require(_maxTransferAmountRate <= 10000, "DSB::updateMaxTransferAmountRate: Max transfer amount rate must not exceed the maximum rate.");
        emit MaxTransferAmountRateUpdated(msg.sender, maxTransferAmountRate, _maxTransferAmountRate);
        maxTransferAmountRate = _maxTransferAmountRate;
    }

    /**
     * @dev Update the min amount to liquify.
     * Can only be called by the current operator.
     */
    function updateMinAmountToLiquify(uint256 _minAmount) public onlyOperator {
        emit MinAmountToLiquifyUpdated(msg.sender, minAmountToLiquify, _minAmount);
        minAmountToLiquify = _minAmount;
    }

    /**
     * @dev Exclude or include an address from antiWhale.
     * Can only be called by the current operator.
     */
    function setExcludedFromAntiWhale(address _account, bool _excluded) public onlyOperator {
        _excludedFromAntiWhale[_account] = _excluded;
    }

    /**
     * @dev Update the swapAndLiquifyEnabled.
     * Can only be called by the current operator.
     */
    function updateSwapAndLiquifyEnabled(bool _enabled) public onlyOperator {
        emit SwapAndLiquifyEnabledUpdated(msg.sender, _enabled);
        swapAndLiquifyEnabled = _enabled;
    }

    /**
     * @dev Update the swap router.
     * Can only be called by the current operator.
     */
    function updateDSBSwapRouter(address _router) public onlyOperator {
        DSBSwapRouter = IUniswapV2Router02(_router);
        DSBSwapPair = IUniswapV2Factory(DSBSwapRouter.factory()).getPair(address(this), DSBSwapRouter.WETH());
        require(DSBSwapPair != address(0), "DSB::updateDSBSwapRouter: Invalid pair address.");
        emit DSBSwapRouterUpdated(msg.sender, address(DSBSwapRouter), DSBSwapPair);
    }

```

DESCRIPTION	The Operator has permission to disable the tax, adding liquidity from tax. The Operator can also update min liquidity to add in a transfer, exclude addresses from anti-whale, set max transfer rate, Change router and steal

	tokens that are getting swapped. These settings pose a risk if the operator address is accessed by a malicious actor.
RECOMMENDATION	Consider changing these functions to be more limited in scope as the operator have permission to abuse in many ways and if a malicious actor got access to the dev address it could be disastrous. Also, implement a timelock such that users have time to react to changes that would be disadvantageous to them.
MITIGATED/COMMENT	The project team has transferred the ownership to a timelock contract.

Transfer Always Taxes

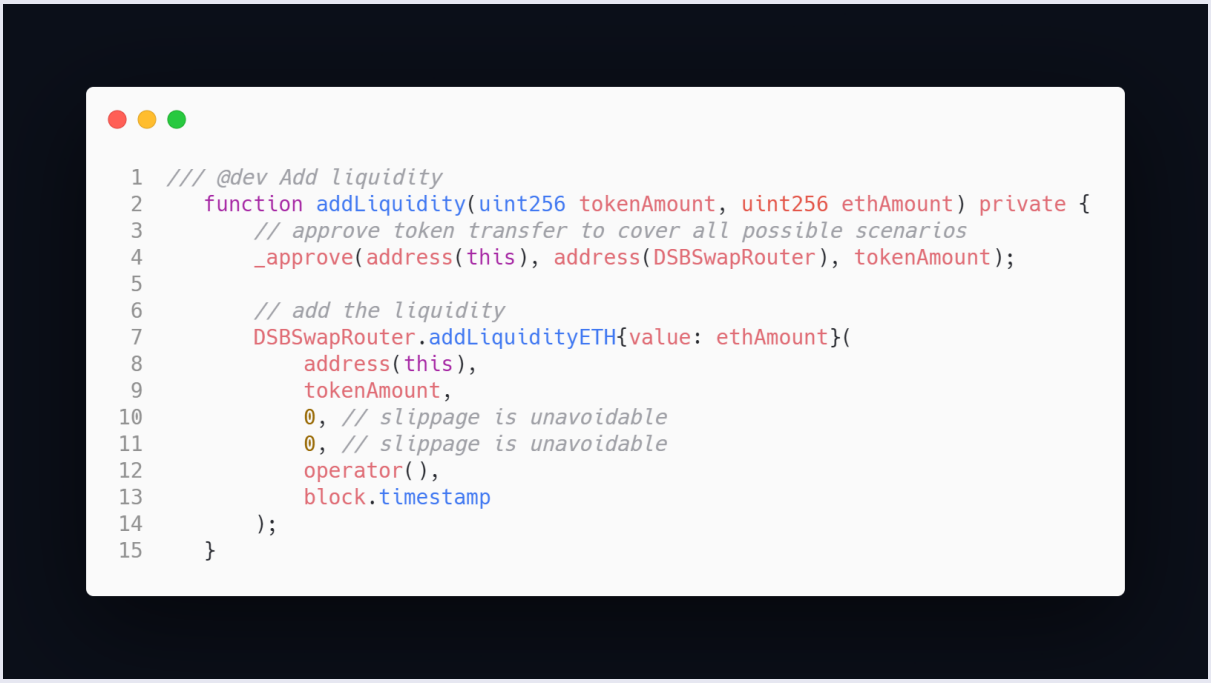
SEVERITY	Informational
RESOLVED	**Partially**
LOCATION	DSBToken.sol -> 100-131

```
1     function _transfer(address sender, address recipient, uint256 amount) internal virtual override
  antiWhale(sender, recipient, amount) {
2         // swap and liquify
3         if (
4             swapAndLiquifyEnabled == true
5             && _inSwapAndLiquify == false
6             && address(DSBSwapRouter) != address(0)
7             && DSBSwapPair != address(0)
8             && sender != DSBSwapPair
9             && sender != owner()
10        ) {
11            swapAndLiquify();
12        }
13
14        if (recipient == BURN_ADDRESS || transferTaxRate == 0) {
15            super._transfer(sender, recipient, amount);
16        } else {
17            // default tax is 5% of every transfer
18            uint256 taxAmount = amount.mul(transferTaxRate).div(10000);
19            uint256 burnAmount = taxAmount.mul(burnRate).div(100);
20            uint256 liquidityAmount = taxAmount.sub(burnAmount);
21            require(taxAmount == burnAmount + liquidityAmount, "DSB::transfer: Burn value invalid");
22
23            // default 95% of transfer sent to recipient
24            uint256 sendAmount = amount.sub(taxAmount);
25            require(amount == sendAmount + taxAmount, "DSB::transfer: Tax value invalid");
26
27            super._transfer(sender, BURN_ADDRESS, burnAmount);
28            super._transfer(sender, address(this), liquidityAmount);
29            super._transfer(sender, recipient, sendAmount);
30            amount = sendAmount;
31        }
32    }
33
```

DESCRIPTION	Token always taxes every address except ***BURN_ADDRESS*** or if ***transferTaxRate*** is 0. Thereby when swapping a token the token pair will still be taxed for sending the token but it will not exchange the funds in the contract and liquefy them at that time. LP tokens are then made and sent to the Operator on the next transfer by a normal user.
RECOMMENDATION	Consider if this is expected behavior to tax for doing these actions as it might incur a big fee for users even when just buying and directly staking.
MITIGATED/COMMENT	Project Comment: “Right, every transfer will tax about 5% for now, we wrote that in the documents, but we just proposed a community vote for changing the transfer tax rate and we will announce the new tax rate soon.”

Liquidity Distributed To Operator

SEVERITY	Medium Risk
RESOLVED	**Partially**
LOCATION	DBSToken.sol -> 185-199



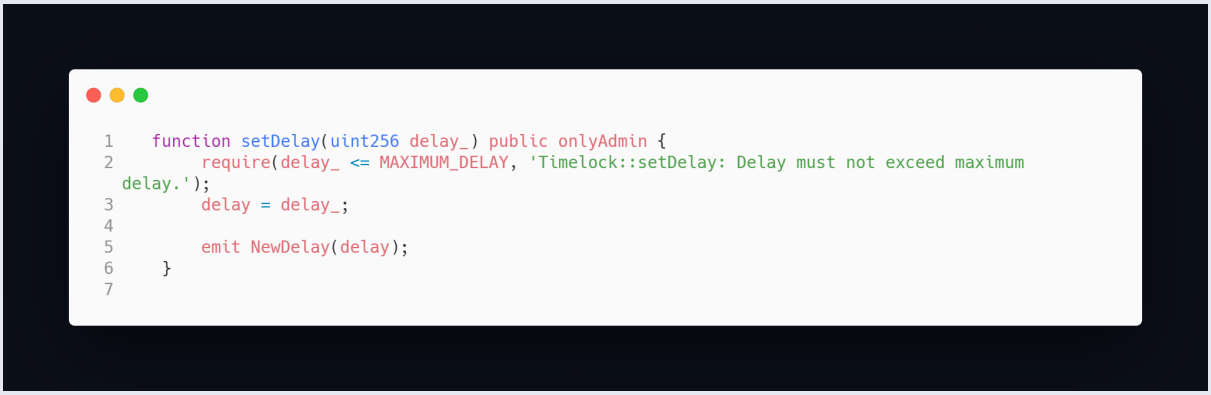
```
/// @dev Add liquidity
    function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
        // approve token transfer to cover all possible scenarios
        _approve(address(this), address(DSBSwapRouter), tokenAmount);

        // add the liquidity
        DSBSwapRouter.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            operator(),
            block.timestamp
        );
    }
```

DESCRIPTION	The contract Operator will receive all liquidity added by the contract. As a result, the Operator will acquire a vast amount of tokens and would probably be able to drain the liquidity pool. The project docs stated that part of the liquidity would be distributed to liquidity providers.
RECOMMENDATION	Transfer the liquidity to a time-locked contract or hold the liquidity directly within the contract itself. Ideally, the distribution of liquidity should not be manually handled.
MITIGATED/COMMENT	Project Comment: “We are developing this contract now and it will launch a few months later in a new section on the DSBSwap platform.” Obelisk: The future funds are now received under a timelock and Obelisk hasn’t audited what's been done with the past funds.

Timelock Bypass

SEVERITY	Medium
RESOLVED	**Yes**
LOCATION	**(old)**Timelock.sol -> 283-288



```
1    function setDelay(uint256 delay_) public onlyAdmin {
2         require(delay_ <= MAXIMUM_DELAY, 'Timelock::setDelay: Delay must not exceed maximum
  delay.');
3         delay = delay_;
4
5         emit NewDelay(delay);
6     }
7
```

DESCRIPTION	The admin is able to set the delay of the timelock as long as it's under the maximum delay. thereby a malicious actor controlling the admin address could bypass the timelock by setting the delay to 0.
RECOMMENDATION	We recommend having only the timelock being able to set a new delay(a change in the delay would thereby have to be done through the timelock). We also recommend adding a minimum delay.
MITIGATED/COMMENT	The project team has implemented the recommended changes.

Static Analysis

No Findings

On-Chain Analysis

Operator Is Externally Owned Address (EOA)

SEVERITY	Medium Risk
RESOLVED	**Yes**

DESCRIPTION	The operator has dangerous permissions and the added liquidity goes directly to this address thus should at least be under a timelock.
RECOMMENDATION	Implement a timelock for the operator
MITIGATED/COMMENT	The project team has changed the operator to be owned by a timelock.

Appendix A - Reviewed Documents

Document	**Address**
DSBToken.sol	0xa1df781126fa49a9f07f4b3cce7b88b78ed3f6ff
(old)Timelock.sol	0x06c4C42F0DC3C01d3E5eF7A4BBaAedF2c57Bb43c
Timelock.sol	0x949aff499010d7436f5cae09d361d6c7a36201ff

Appendix B - Risk Ratings

Risk	Description
High Risk	A fatal vulnerability that can cause immediate loss of Tokens / Funds
Medium Risk	A vulnerability that can cause some loss of Tokens / Funds
Low Risk	A vulnerability that can be mitigated
Informational	No vulnerability

Appendix C - Icons

Icon	Explanation
	Solved by Project Team
	Under Investigation of Project Team
	Unsolved

Appendix D - Testing Standard

An ordinary audit is conducted using these steps.

1. Gather all information
2. Conduct a first visual inspection of documents and contracts
3. Go through all functions of the contract manually (2 independent auditors)
    a. Discuss findings
4. Use specialized tools to find security flaws
    a. Discuss findings
5. Follow up with project lead of findings
6. If there are flaws, and they are corrected, restart from step 2
7. Write and publish a report

During our audit, a thorough investigation has been conducted employing both automated analysis and manual inspection techniques. Our auditing method lays a particular focus on the following important concepts:

- Ensuring that the code and codebase use best practices, industry standards, and available libraries.
- Testing the contract from different angles ensuring that it works under a multitude of circumstances.
- Analyzing the contracts through databases of common security flaws.

OBELISK

Part of Tibereum Group