



Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 24	2021-06-13	Zapmore, Hebilicious	Final Audit

Audit Notes

Audit Date	2021-06-05 - 2021-06-12
Auditor/Auditors	Hebilicious, Plemonade
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	OB58889510

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
Liquidity Centralization Risk	10
Disruptive Owner Only Functions: Unbounded Fees	12
Disruptive owner only functions: max transaction amount	13
Disruptive Owner Only Functions: Fee Whitelist	14
Unlock Gives Ownership Regardless Of Ownership State	15
Outdated Compiler Version	16
Debug Library Present In Deployed Contract	17
Outdated Comments And Variable Names	18
Static Analysis	19
No Findings	19
On-Chain Analysis	20
No Findings	20
Appendix A - Reviewed Documents	21
Appendix B - Risk Ratings	22
Appendix C - Icons	22
Appendix D - Testing Standard	23

Project Information

Project Name	GiveEarth
Description	A sustainable token for a better world, better environment, and better people.
Website	https://giveearth.net/
Contact	@sidgive
Contact information	@sidgive on TG
Token Name(s)	N/A
Token Short	N/A
Contract(s)	See Appendix A
Code Language	Solidity
Chain	BSC

Executive Summary

The audit of GiveEarth was conducted by two of Obelisks' security experts between the 5th of June 2021 and the 12th of May 2021.

After finishing the full audit, Obelisk Auditing can say that there were some findings with both Low and Medium severity that could cause some issues. The project team added a 48 timelock that mitigates these issues as long as the functions are not called. There were also some Informational findings that pose no risk.

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

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

Summary Table

Audited Part	Severity	Note
Non-Withdrawable BNB	Medium	See Comment
Liquidity Centralization Risk	Medium	See Comment
Disruptive Owner Only Functions: Unbounded Fees	Low	See Comment
Disruptive owner only functions: max transaction amount	Medium	See Comment
Disruptive Owner Only Functions: Fee Whitelist	Low	See Comment
Unlock Gives Ownership Regardless Of Ownership State	Medium	See Comment
Outdated Compiler Version	Informational	N/A
Debug Library Present In Deployed Contract	Informational	N/A
Outdated Comments And Variable Names	Informational	N/A

Introduction

Obelisk was commissioned by GiveEarth on the 4th of June 2021 to conduct a comprehensive audit of GiveEarth's project on Binance Smart Chain. The following audit was conducted between the 5th of June 2021 and the 12th of June 2021 and delivered on the 13th 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.

Going through the contracts, the auditors found multiple instances where there could be a problem for the users. This includes things such as liquidity lock, timelock, and the central control an EAO wallet has to control certain aspects of the project. After bringing these issues up to the project team, they swiftly implemented a 48hour long timelock that mitigated these issues as long as the functions are not called. With 48 hour long timelock, the users would have enough time to see if any of these functions are called.

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

Findings

Manual Analysis

Non-Withdrawable BNB

SEVERITY	Medium
RESOLVED	See Comment In Mitigation
LOCATION	FullSteamAhead.sol: 1064

```
1 function swapAndLiquify(uint256 contractTokenBalance) private lockTheSwap {
2     // split the contract balance into halves
3     uint256 half = contractTokenBalance.div(2);
4     uint256 otherHalf = contractTokenBalance.sub(half);
5
6     // capture the contract's current ETH balance.
7     // this is so that we can capture exactly the amount of ETH that the
8     // swap creates, and not make the liquidity event include any ETH that
9     // has been manually sent to the contract
10    uint256 initialBalance = address(this).balance;
11
12    // swap tokens for ETH
13    swapTokensForEth(half); // <- this breaks the ETH -> HATE swap when swap+liquify is
    triggered
14
15    // how much ETH did we just swap into?
16    uint256 newBalance = address(this).balance.sub(initialBalance);
17
18    // add liquidity to uniswap
19    addLiquidity(otherHalf, newBalance);
20
21    emit SwapAndLiquify(half, newBalance, otherHalf);
22 }
```

DESCRIPTION	The liquidity logic leaves some BNB dust when it runs. Given that the BNB is automatically swapped against the liquidity providers with the taxed tokens, this effectively decreases the token value over time.
RECOMMENDATION	<ul style="list-style-type: none">• Add a function to withdraw that BNB.• Add logic to buy back the token and burn it.• Add logic to distribute the BNB to the token holders.
MITIGATION	GiveEarth Team: "\$GIVE does NOT provide the Automatic Liquidity Pool (LP), setSwapAndLiquifyEnabled has already been set to false. A long enough timelock will be added for the contract."

Obelisk: Mitigated with a timelock as long as the feature never is enabled.

Liquidity Centralization Risk

SEVERITY	Medium
RESOLVED	See Comment In Mitigation
LOCATION	1105

```
1 function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
2     // approve token transfer to cover all possible scenarios
3     _approve(address(this), address(uniswapV2Router), tokenAmount);
4
5     // add the liquidity
6     uniswapV2Router.addLiquidityETH{value: ethAmount}(
7         address(this),
8         tokenAmount,
9         0, // slippage is unavoidable
10        0, // slippage is unavoidable
11        owner(),
12        block.timestamp
13    );
14 }
```

DESCRIPTION	The liquidity gets added to the owner's wallet, which eventually ends up with the owner controlling most of the token liquidity.
RECOMMENDATION	Add the liquidity to a contract designed to handle it, or use a long enough Timelock.
MITIGATION	<p>GiveEarth Team: "\$GIVE does NOT provide the Automatic Liquidity Pool (LP), setSwapAndLiquifyEnabled has already been set to false. A long enough timelock will be added for the contract."</p> <p>Obelisk: Mitigated with a timelock as long as the feature never is enabled.</p>

Disruptive Owner Only Functions: Unbounded Fees

SEVERITY	Low
RESOLVED	See Comment In Mitigation
LOCATION	906

```
1  function setTaxFeePercent(uint256 taxFee) external onlyOwner() {  
2      _taxFee = taxFee;  
3  }  
4  
5  function setLiquidityFeePercent(uint256 liquidityFee) external onlyOwner() {  
6      _liquidityFee = liquidityFee;  
7  }
```

DESCRIPTION	The fees can be set to 100% of the transactions.
RECOMMENDATION	Add a limit to the fees and use a Timelock to warn investors of the settings change.
MITIGATION	Obelisk: Mitigated with a timelock as long as the feature never is enabled.

Disruptive owner only functions: max transaction amount

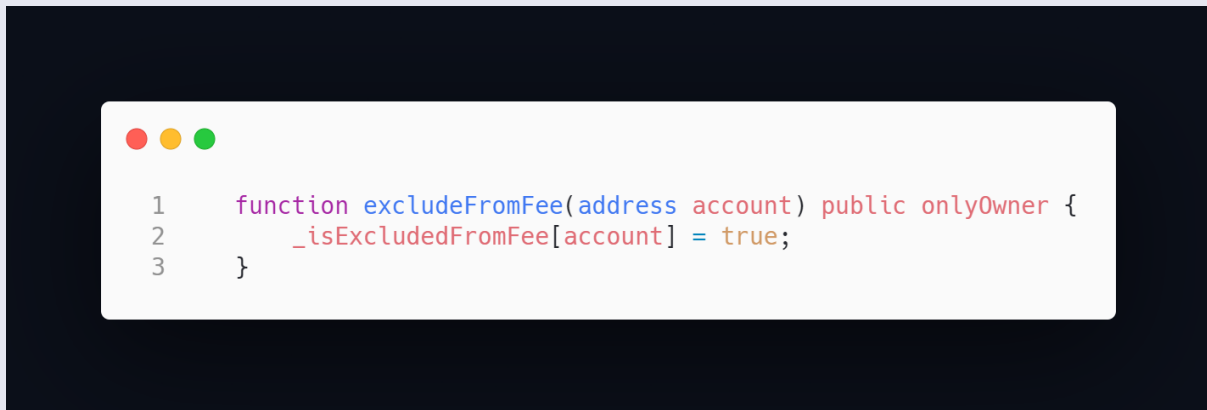
SEVERITY	Low
RESOLVED	See Comment In Mitigation
LOCATION	906

```
1 function setTaxFeePercent(uint256 taxFee) external onlyOwner() {  
2     _taxFee = taxFee;  
3 }  
4  
5 function setLiquidityFeePercent(uint256 liquidityFee) external onlyOwner() {  
6     _liquidityFee = liquidityFee;  
7 }
```

DESCRIPTION	The max transaction amount can be set to 0, preventing any token transfers.
RECOMMENDATION	Add a minimum value to the transaction amount, and use a Timelock to warn investors of the settings change.
MITIGATION	Obelisk: Mitigated with a timelock as long as the feature never is enabled.

Disruptive Owner Only Functions: Fee Whitelist

SEVERITY	Low
RESOLVED	See Comment In Mitigation
LOCATION	898



DESCRIPTION	The owner can allow anyone to bypass the fees.
RECOMMENDATION	Use a timelock to make sure investors can evaluate the fairness of the system.
MITIGATION	<p>GiveEarth Team: "All the fee whitelists will be announced publicly. A long enough timelock will be added for the contract to make sure investors are aware of the fairness of the system."</p> <p>Obelisk: Mitigated with a timelock as long as the feature never is enabled.</p>

Unlock Gives Ownership Regardless Of Ownership State

SEVERITY	Medium
RESOLVED	See Comment In Mitigation
LOCATION	481

```
1 //Locks the contract for owner for the amount of time provided
2 function lock(uint256 time) public virtual onlyOwner {
3     _previousOwner = _owner;
4     _owner = address(0);
5     _lockTime = now + time;
6     emit OwnershipTransferred(_owner, address(0));
7 }
8
9 //Unlocks the contract for owner when _lockTime is exceeds
10 function unlock() public virtual {
11     require(_previousOwner == msg.sender, "You don't have permission to unlock");
12     require(now > _lockTime, "Contract is locked until 7 days");
13     emit OwnershipTransferred(_owner, _previousOwner);
14     _owner = _previousOwner;
15 }
```

DESCRIPTION	The lock and unlock function can be used to bypass the ownership transfers. Calling unlock will give ownership back to the lock caller, regardless of who the current owner is.
RECOMMENDATION	Change the locking logic so that ownership transfers and renounce ownership can't be reverted, or remove those functions.
MITIGATION	Obelisk: The GiveEarth team has transfered the ownership to a Timelock contract without calling the lock or unlock function at any point which mitigates this issue, however the logical flaw still persist.

Outdated Compiler Version

SEVERITY	Informational
RESOLVED	NO
LOCATION	All

DESCRIPTION	The solidity version is 0.6.12.
RECOMMENDATION	Use the latest compiler versions whenever possible.
MITIGATION	N/A

Debug Library Present In Deployed Contract

SEVERITY	Informational
RESOLVED	NO
LOCATION	30



DESCRIPTION	A debug development-only library is present on the deployed contract.
RECOMMENDATION	Make sure to remove this library before deploying the contract. This can be achieved easily with the hardhat-preprocessor plugin, which exposes a removeConsoleLog function that can be used in the hardhat configuration file.
MITIGATION	N/A

Outdated Comments And Variable Names

SEVERITY	Informational
RESOLVED	NO
LOCATION	All

DESCRIPTION	Comments and variable names from previous forks have been re-used and are not matching the logic in all cases.
RECOMMENDATION	When forking, update comments and variable names to describe accurately the current logic.
MITIGATION	N/A

Static Analysis

No Findings

On-Chain Analysis

No Findings




Appendix A - Reviewed Documents

Document	Address
FullSteamAhead.sol	0xa044dc77960ff679916165c4dccc06f10ef61d59
Timelock.sol	0x8b30990ADd17f9732eA4C9fcFC6eA37C46A5EE01

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.

Follow Obelisk Auditing for the Latest Information



ObeliskOrg



ObeliskOrg



Part of Tibereum Group