OBELISK

Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 33	2022-08-22	DoD4uFN, Donut	Audit Final

Audit Notes

Audit Date	2022-08-09 - 2022-08-21
Auditor/Auditors	DoD4uFN, mechwar
Auditor/Auditors Contact Information	contact@obeliskauditing.com
Notes	Specified code and contracts are audited for security flaws. UI/UX (website), logic, team, and tokenomics are not audited.
Audit Report Number	OB569687511

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. In instances where an auditor or team member has a personal connection with the audited project, that auditor or team member will be excluded from viewing or impacting any internal communication regarding the specific audit.

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.

Audit Information

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

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.

Table of Contents

Project Information

Name	Based
Description	BASED is an algorithmic token pegged to TOMB. The protocol's underlying mechanism dynamically adjusts BASED supply, trading its price up or down relative to the price of TOMB.
Website	https://basedfinance.io/
Contact	https://twitter.com/BasedFinance_io
Contact information	@@athena_goddazz on TG
Token Name(s)	N/A
Token Short	N/A
Contract(s)	See Appendix A
Code Language	Solidity
Chain	Fantom

Audit of Based

Obelisk was commissioned by Based on the 7th of August 2022 to conduct a comprehensive audit of Baseds' additional contracts. The following audit was conducted between the 8th of August 2022 and the 21st of August 2022. Two of Obelisk's security experts went through the related contracts manually using industry standards to find if any vulnerabilities could be exploited either by the project team or users.

During the audit of Based's additional contracts, we found multiple instances of risky code. The project team worked fast to solve most of the issues in their contracts with only issues #3 and #5 still being open, and issue #7 being partially open. Issue #3 is handled by the project team in a temporary way and will be permanently fixed during a re-deploy. Issue #5 is safe as long as the project team follows their own comment on the issue. Likewise, issue #7 should be ok as long as the project team is aware of the potential issue and follows through according to their comment. This audit is conducted on already deployed contracts which means that the contracts need a redeploy in order to be solved on-chain.

The informational findings are good to know while interacting with the project but don't directly damage the project in its current state, hence it's up to the project team if they deem that it's worth solving these issues, however, please take note of them.

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

This document is a summary of the findings that the auditors found. Please read the full document for a complete understanding of the audit.

Summary Table

Code Analysis

Finding	ID	Severity	Status
Users Staked Tokens Could Be Drained	#0001	High Risk	Closed
Missing Emergency NFT Withdraw Function	#0002	High Risk	Closed
Reward Tokens Can Be Withdrawn	#0003	Medium Risk	Open
Tokens With Transfer Fee Not Supported	#0004	Medium Risk	Closed
Unbounded Loop	#0005	Low Risk	Open
Rewards Can Be Lost	#0006	Low Risk	Closed
No Limit For Protocol Values	#0007	Low Risk	Partially Closed
Contract Values Can Be Constant Or Immutable	#0008	Informational	Closed
Unused Variables	#0009	Informational	Closed
Unused Events	#0010	Informational	Closed
Using Safe Math In Solidity ^0.8.0	#0011	Informational	Closed
Inconsistent Error Messages	#0012	Informational	Closed
No Events Emitted For Changes To Protocol Values	#0013	Informational	Partially Closed

On-Chain Analysis

Finding	ID	Severity	Status
N/A	#0014	N/A	Open

Findings

Code Analysis

Users Staked Tokens Could Be Drained

FINDING ID	#0001
SEVERITY	High Risk
STATUS	Closed
LOCATION	SmeltRewardPool.sol -> 473-497

```
  function governanceAllocationAdjustment(
        uint256 _pid,
        uint256 _amount,
        address _teamMember
    ) external onlyOperator {
        PoolInfo storage pool = poolInfo[_pid];
        require (pool.token == stater , "team pool only");
//==================make sure this is TEAM POOL ONLY
        UserInfo storage user = userInfo[_pid][_teamMember];
        updatePool(_pid);
        uint256 _pending =
user.amount.mul(pool.accSmeltPerShare).div(1e18).sub(user.rewardDebt);
        if (_pending > 0) {
            safeSmeltTransfer(protocolFundAddress, _pending);
            emit RewardPaid(protocolFundAddress, _pending);
        }
        if (_amount < user.amount){
            uint256 cut = user.amount.sub(_amount);
            stater.safeTransfer(protocolFundAddress, cut);
        } else if(_amount > user.amount) {
            uint256 bonus = _amount.sub(user.amount);
            stater.safeTransferFrom(protocolFundAddress, address(this),
bonus);
        }
        user.amount = _amount;
        user.rewardDebt =
user.amount.mul(pool.accSmeltPerShare).div(1e18);
        emit TeamMemberAllocationAdjusted(_teamMember, _amount);
    }
```

LOCATION	SmeltRewardPool.sol -> 463-467

```
function setTeamToken (address _teamToken) public onlyOperator
 {
      require (_teamToken != address(0), "cant be 0 address");
      stater = IERC20(_teamToken);
 }
```

DESCRIPTION	The function *setTeamToken()* can be set to any ERC20 token that is the token of an existing staking pool. By calling *governanceAllocationAdjustment()*, with an address of a particular user and their staking pool ID, this function could drain the unsuspecting user's staked tokens to the *protocolFundAddress* address. Then, *governanceAllocationAdjustment()* can be used again, with a teamMember's address, to allocate these funds from *protocolFundAddress* to themselves and use the *withdraw()* function to drain the funds. Also, the function *governanceAllocationAdjustment()* can be used by the team to assign to themselves from 0% up to 100% of the accrued deposit fees (as long as the *stater* can be changed to any ERC-20).
RECOMMENDATION	Set the *stater* once in the constructor of the contract or make the *stater* a constant.
RESOLUTION	The project team has implemented the recommended changes. Note: *Changes have not been deployed*.

Missing Emergency NFT Withdraw Function

FINDING ID	#0002
SEVERITY	High Risk
STATUS	Closed
LOCATION	SmeltRewardPool.sol -> 343-352

```
1     // Withdraw without caring about rewards. EMERGENCY ONLY.
2     function emergencyWithdraw(uint256 _pid) public {
3         PoolInfo storage pool = poolInfo[_pid];
4         UserInfo storage user = userInfo[_pid][msg.sender];
5         uint256 _amount = user.amount;
6         user.amount = 0;
7         user.rewardDebt = 0;
8         pool.token.safeTransfer(msg.sender, _amount);
9         emit EmergencyWithdraw(msg.sender, _pid, _amount);
10    }
```

DESCRIPTION	The smart contract implements every functionality twice, once for ERC-20 and once for ERC-721 tokens. The emergency withdrawal functionality is implemented only for ERC-20 tokens.
RECOMMENDATION	Implement the emergency withdrawal for ERC-721 tokens.
RESOLUTION	The project team has implemented the recommended changes. Note: *Changes have not been deployed*.

Reward Tokens Can Be Withdrawn

FINDING ID	#0003
SEVERITY	Medium Risk
STATUS	Open
LOCATION	SmeltRewardPool.sol -> 499-507

```
1   function governanceRecoverUnsupported(
2         IERC20 _token,
3         uint256 _amount,
4         address _to
5     ) external onlyOperator {
6         require(address(_token) == address(smelt), "reward token
  only");
7         _token.safeTransfer(_to, _amount);
8     }
9 }
```

DESCRIPTION	The reward tokens can be withdrawn by the operator using the *governanceRecoverUnsupported()* function. These kinds of functions are usually used to withdraw tokens that aren't being used by the contract.
RECOMMENDATION	Adjust the function to only withdraw unused tokens.
RESOLUTION	The project team introduced a timelock of 30 days after the pool's end time that the reward tokens and the pool tokens aren't withdrawable. After this period, any token can be withdrawn. However, a new function was introduced, called *moveRewardsToUpdatedContract()* which allows the *owner* of the contract to move the reward tokens to any address, without the time restrictions that were introduced to the *governanceRecoverUnsupported()* function. If the intention is to move the reward tokens to an updated contract, a sufficient timelock (72 hours) is recommended. Project team comment: *"Owner and Operator is transferred to a multisig. Will have moveRewardsToUpdatedContract function removed in updated version."*

Tokens With Transfer Fee Not Supported

FINDING ID	#0004
SEVERITY	Medium Risk
STATUS	Closed
LOCATION	SmeltRewardPool.sol -> 235-240

```
1         if (_amount > 0) {
2             pool.token.safeTransferFrom(_sender, address(this),
  _amount);
3             uint256 depositDebt =
  _amount.mul(pool.depositFee).div(10000);
4             user.amount = user.amount.add(_amount.sub(depositDebt));
5             pool.token.safeTransfer(protocolFundAddress, depositDebt);
6         }
```

DESCRIPTION	The *deposit()* function does not support fees on transfer tokens. If the deposited token has a fee on transfer, there can be a discrepancy in the actually received amount. This discrepancy can cause other users to lose their deposits.
RECOMMENDATION	Check the token balance before and after the transfer to get the actually received amount.
RESOLUTION	The project team has implemented the recommendation. The reward pool does not allow any tokens with transfer fees. Note: *Changes have not been deployed*.

Unbounded Loop

FINDING ID	#0005
SEVERITY	Low Risk
STATUS	Open
LOCATION	SmeltRewardPool.sol -> 167-172

```
1    function massUpdatePools() public {
2        uint256 length = poolInfo.length;
3        for (uint256 pid = 0; pid < length; ++pid) {
4            updatePool(pid);
5        }
6    }
```

DESCRIPTION	Iterating over an unbounded array can cause transactions to revert due to the gas limit.
RECOMMENDATION	It is recommended to add an input variable to *massUpdatePools()* that bounds the loop to a max limit. The function *add()* which calls *massUpdatePools()* would also need the same input variable.
RESOLUTION	Project team comment: *"The number of pools will not exceed 10."* Obelisk comment: *"We recommend adding a limit to the number of pools to add just in case"*

Rewards Can Be Lost

FINDING ID	#0006
SEVERITY	Low Risk
STATUS	Closed
LOCATION	SmeltRewardPool.sol -> 379-389

```
1     // Safe SMELT transfer function, in case if rounding error causes
  pool to not have enough SMELTs.
2     function safeSmeltTransfer(address _to, uint256 _amount) internal {
3         uint256 _smeltBalance = smelt.balanceOf(address(this));
4         if (_smeltBalance > 0) {
5             if (_amount > _smeltBalance) {
6                 smelt.safeTransfer(_to, _smeltBalance);
7             } else {
8                 smelt.safeTransfer(_to, _amount);
9             }
10        }
11    }
```

DESCRIPTION	The function *safeSmeltTransfer()* transfers the reward tokens to the user. If the smart contract doesn't have enough reward tokens to send to the user, it will send fewer rewards. Although, the existing functionality will reset the user's rewards to 0, even if he doesn't receive rewards.
RECOMMENDATION	Don't allow users to withdraw and abandon their rewards using the regular *withdraw()* functionality.
RESOLUTION	The project team has implemented the recommended changes. Note: *Changes have not been deployed*.

No Limit For Protocol Values

FINDING ID	#0007
SEVERITY	Low Risk
STATUS	Partially Closed
LOCATION	SmeltRewardPool.sol -> 395-448

```
    function add(
        bool _isNftPool,
        IERC20 _token,
        IERC721 _nft,
        uint256 _depFee,
        uint256 _allocPoint,
        bool _withUpdate,
        uint256 _lastRewardTime
    ) public onlyOperator {
        // ...
    }
```

LOCATION	SmeltRewardPool.sol -> 451-461

```
    function set(uint256 _pid, uint256 _allocPoint) public onlyOperator
{
        massUpdatePools();
        PoolInfo storage pool = poolInfo[_pid];

        if (pool.isStarted) {
            totalAllocPoint = totalAllocPoint.sub(pool.allocPoint).add(
                _allocPoint
            );
        }
        pool.allocPoint = _allocPoint;
    }
```

DESCRIPTION	The following values *_allocPoint* and *_depFee* can be set arbitrarily high, potentially breaking the functionality of the contract.
RECOMMENDATION	Add a reasonable upper limit to the values of *_allocPoint* and *_depFee*. A lower limit may be useful as well.
RESOLUTION	The project team has partially implemented the recommendation. There is a limit for *_depFee*, but still no limit for *_allocPoint*.

	Project team comment: "*alloc points limits will limit the flexibility of expanding at certain times. Have exact numbers and timeframes to emmit correctly.*" Obelisk team comment: "*Setting a high allocation limit (perhaps 10000) should hopefully provide flexibility without allowing unlimited control.*" Note: *Changes have not been deployed*.

Contract Values Can Be Constant Or Immutable

FINDING ID	#0008
SEVERITY	Informational
STATUS	Closed
LOCATION	• SmeltRewardPool.sol -> 19: *IERC20 public smelt;* • SmeltRewardPool.sol -> 64: *uint256 public poolStartTime;* • SmeltRewardPool.sol -> 67: *uint256 public poolEndTime;* • SmeltRewardPool.sol -> 69: *address public protocolFundAddress;* • SmeltRewardPool.sol -> 72: *uint256 public smeltPerSecond = 0.00115 ether;* • SmeltRewardPool.sol -> 73: *uint256 public runningTime = 800 days;*

DESCRIPTION	Variables which do not change during the operation of a contract can be marked *constant* or *immutable* to reduce gas costs and improve code readability.
RECOMMENDATION	Mark these variables as *constant* or *immutable* as appropriate.
RESOLUTION	The project team has implemented the recommended changes. Note: *Changes have not been deployed*.

Unused Variables

FINDING ID	#0009
SEVERITY	Informational
STATUS	Closed
LOCATION	• SmeltRewardPool.sol -> 74: *uint256 public constant TOTAL_REWARDS = 80000 ether;*

DESCRIPTION	The noted variables are never used.
RECOMMENDATION	Remove the variables or incorporate them into the contract functionality.
RESOLUTION	The project team has implemented the recommended changes. Note: *Changes have not been deployed*.

Unused Events

FINDING ID	#0010
SEVERITY	Informational
STATUS	Closed
LOCATION	• SmeltRewardPool.sol -> 87: *event StringFailure(string stringFailure);*

DESCRIPTION	The noted events are never used.
RECOMMENDATION	Remove the events.
RESOLUTION	The project team has implemented the recommended changes. Note: *Changes have not been deployed*.

Using Safe Math In Solidity ^0.8.0

FINDING ID	#0011
SEVERITY	Informational
STATUS	Closed
LOCATION	SmeltRewardPool.sol -> 17

```
1 using SafeMath for uint256;
```

DESCRIPTION	The *SafeMath* library is imported in *SmeltRewardPool.sol*, while the contract is using Solidity ^0.8.0, in which the compiler has built-in overflow check.
RECOMMENDATION	Remove the *SafeMath* library from the contract.
RESOLUTION	The project team has implemented the recommended changes. Note: *Changes have not been deployed*.

Inconsistent Error Messages

FINDING ID	#0012
SEVERITY	Informational
STATUS	Closed
LOCATION	• SmeltRewardPool.sol -> 223: *require (pool.isNftPool == false , "Pool not for ERC20");* • SmeltRewardPool.sol -> 310: *require (pool.isNftPool == false , "pool for nfts");*

DESCRIPTION	The error messages for the same require statements are different.
RECOMMENDATION	Keep the error messages consistent across the smart contract.
RESOLUTION	The same messages appear in different required statements. *SmeltRewardPool.sol:227* *SmeltRewardPool.sol:255* Project team comment: *“have appropriate edits for comments.“*

No Events Emitted For Changes To Protocol Values

FINDING ID	#0013
SEVERITY	Informational
STATUS	Partially Closed
LOCATION	SmeltRewardPool.sol -> 395-448

```
    function add(
        bool _isNftPool,
        IERC20 _token,
        IERC721 _nft,
        uint256 _depFee,
        uint256 _allocPoint,
        bool _withUpdate,
        uint256 _lastRewardTime
    ) public onlyOperator {
        // ...
    }
```

LOCATION	SmeltRewardPool.sol -> 499-507

```
  function governanceRecoverUnsupported(
        IERC20 _token,
        uint256 _amount,
        address _to
    ) external onlyOperator {
        require(address(_token) == address(smelt), "reward token only");
        _token.safeTransfer(_to, _amount);
    }
}
```

LOCATION	• SmeltRewardPool.sol -> 463: *function setTeamToken (address _teamToken) public onlyOperator* • SmeltRewardPool.sol -> 468: *function setOperator(address _operator) external onlyOperator* • SmeltRewardPool.sol -> 451: *function set(uint256 _pid, uint256 _allocPoint) public onlyOperator*

DESCRIPTION	Functions that change important variables should emit events such that users can more easily monitor the change.

RECOMMENDATION	Emit events from these functions.
RESOLUTION	Events were added to the functions except *add()*, *set()* and *governanceRecoverUnsupported()*. Project team comment: "*event for add, set are not added to avoid frontrunning for users who call through contract.*" Obelisk comment: "*Front will generally be watching the mem pool to watch for upcoming transactions and will not be deterred by a lack of events.*" Note: *Changes have not been deployed*.

On-Chain Analysis

Not Deployed Yet

External Addresses

Externally Owned Accounts

Not Deployed Yet

External Contracts

These contracts are not part of the audit scope.

Not Deployed Yet

External Tokens

These contracts are not part of the audit scope.

Not Deployed Yet

Appendix A - Reviewed Documents

Deployed Contracts

Document	Address
SmeltRewardPool.sol	N/A

Libraries And Interfaces

Revisions

Revision 1	a63e16383f5220f6eddd738fee6512dff0fb78a9
Revision 2	a3f3b3de971a801f781d2c50d6d2bc7a779a0aec

Imported Contracts

OpenZeppelin	TBD

Appendix B - Risk Ratings

Risk	Description
High Risk	Security risks that are ***almost certain*** to lead to ***impairment or loss of funds***. Projects are advised to fix as soon as possible.
Medium Risk	Security risks that are ***very likely*** to lead to ***impairment or loss of funds*** with ***limited impact***. Projects are advised to fix as soon as possible.
Low Risk	Security risks that can lead to ***damage to the protocol***. Projects are advised to fix. Issues with this rating might be used in an exploit with other issues to cause significant damage.
Informational	Noteworthy information. Issues may include code conventions, missing or conflicting information, gas optimizations, and other advisories.

Appendix C - Finding Statuses

Closed	Contracts were modified to permanently resolve the finding.
Mitigated	The finding was resolved on-chain. The issue may require monitoring, for example in the case of a time lock.
Partially Closed	Contracts were modified to partially fix the issue
Partially Mitigated	The finding was resolved by project specific methods which cannot be verified on chain. Examples include compounding at a given frequency, or the use of a multisig wallet.
Open	The finding was not addressed.

Appendix D - Glossary

Contract Structure

Contract: An address with which provides functionality to users and other contracts. They are implemented in code and deployed to the blockchain.
Protocol: A system of contracts which work together.
Stakeholders: The users, operators, owners, and other participants of a contract.

Security Concepts

Bug**:** A defect in the contract code.
Exploit**:** A chain of events involving bugs, vulnerabilities, or other security risks which damages a protocol.
Funds: Tokens deposited by users or other stakeholders into a protocol.
Impairment: The loss of functionality in a contract or protocol.
Security risk: A circumstance that may result in harm to the stakeholders of a protocol. Examples include vulnerabilities in the code, bugs, excessive permissions, missing timelock, etc.
Vulnerability: A vulnerability is a flaw that allows an attacker to potentially cause harm to the stakeholders of a contract. They may occur in a contract's code, design, or deployed state on the blockchain.

Appendix E - Audit Procedure

A typical Obelisk audit uses a combination of the three following methods:

Manual analysis consists of a direct inspection of the contracts to identify any security issues. Obelisk auditors use their experience in software development to spot vulnerabilities. Their familiarity with common contracts allows them to identify a wide range of issues in both forked contracts as well as original code.

Static analysis is software analysis of the contracts. Such analysis is called “static” as it examines the code outside of a runtime environment. Static analysis is a powerful tool used by auditors to identify subtle issues and to verify the results of manual analysis.

On-chain analysis is the audit of the contracts as they are deployed on the block-chain. This procedure verifies that:

- deployed contracts match those which were audited in manual/static analysis;
- contract values are set to reasonable values;
- contracts are connected so that interdependent contract function correctly;
- and the ability to modify contract values is restricted via a timelock or DAO mechanism. (We recommend a timelock value of at least 72 hours)

Each obelisk audit is performed by at least two independent auditors who perform their analysis separately.

After the analysis is complete, the auditors will make recommendations for each issue based on best practice and industry standards. The project team can then resolve the issues, and the auditors will verify that the issues have been resolved with no new issues introduced.

Our auditing method lays a particular focus on the following important concepts:

- Quality code and the use of best practices, industry standards, and thoroughly tested libraries.
- Testing the contract from different angles to ensure that it works under a multitude of circumstances.
- Referencing the contracts through databases of common security flaws.

Follow Obelisk Auditing for the Latest Information

 ObeliskOrg

 ObeliskOrg

OBELISK

Part of Tibereum Group