OBELISK

Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 56	2022-03-30	Plemonade, Donut	Audit Final

Audit Notes

Audit Date	2022-01-03 - 2022-03-30
Auditor/Auditors	Plemonade, ByFixter
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	OB556898572

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.

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	ShadeCash
Description	A decentralized protocol for private transactions on Fantom Opera
Website	https://shade.cash/
Contact	https://twitter.com/ShadeCash_
Contact information	@iamnotgayy on TG
Token Name(s)	N/A
Token Short	N/A
Contract(s)	See Appendix A
Code Language	Solidity
Chain	Fantom

Audit of ShadeCash

Obelisk was commissioned by ShadeCash on the 3rd of January 2022 to conduct a comprehensive audit of ShadeCash' contracts. The following audit was conducted between the 3rd of January 2022 and the 30th of April 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.

The audit was done in 3 steps at the same time as the contracts were being developed. As a specific contract was finished, our auditors went through the code and found possible vulnerabilities conveyed to the project team. The project team closed and mitigated some of these vulnerabilities, others are still open. Please read through the audit report to fully understand the audited contracts and the findings attached to them. Keep in mind that no on-chain analysis has been performed.

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 it worth solving these issues.

The team has not reviewed the UI/UX, logic, team, or tokenomics of the ShadeCash 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

Finding	ID	Severity	Status
Tokens With Transfer Fee Not Supported	#0001	Low Risk	Mitigated
Trusting External Contract	#0002	Low Risk	Open
No Timelock Implemented	#0003	Low Risk	Mitigated
Contract Values Can Be Constant Or Immutable (Gas Optimization)	#0004	Informational	Closed
Multiple Contracts In One File	#0005	Informational	Closed
No Events Emitted For Changes To Protocol Values	#0006	Informational	Closed
Withdraw Function Never Withdraws	#0007	High Risk	Closed
SafeTransfer Does Not Require Approval	#0008	High Risk	Closed
Lock Array Recreated Every Time Locks Change	#0009	High Risk	Closed
Unbound Loops On User Locks	#0010	Medium Risk	Closed
Checking Whether The Penalty Receiver Is A Contract Is Done Manually	#0011	Low Risk	Closed
Rounding Precision	#0012	Low Risk	Closed
Inconsistent Logic	#0013	Low Risk	Closed
Global Id For Lock	#0014	Informational	Closed
Blank Elements May Be Left In Array	#0015	Low Risk	Closed
Redundant Division And Multiplication	#0016	Informational	Closed
Loops Can Be Combined	#0017	Informational	Closed
Unbounded Loop	#0018	Informational	Closed
Lock Is Never Removed	#0019	High Risk	Closed

Protocol Values Should Be Public	#0020	Informational	Closed
Mixed Tab and Space Indentation	#0021	Informational	Closed
Incorrect Reward Duration Calculation	#0022	Medium Risk	Open
Shade Points May Not Align With Reward Distributor Epochs	#0023	Medium Risk	Open
Use Safe Transfer	#0024	Low Risk	Closed
Rewards Not Allocated If Last Reward Was Over 20 Weeks Ago	#0025	Low Risk	Partially Closed
Rewards Distributed To Current Epoch May Not Be Collected	#0026	Low Risk	Open
Shade, Reward, and Staking Token Addresses Hard Coded	#0027	Low Risk	Closed
Memory Reference Assignment	#0028	High Risk	Closed
User Can Lock Funds For As Little As A Single block	#0029	Low Risk	Closed
Lock Time Can Be Decreased	#0030	Low Risk	Closed
balanceOfAt Should Not Be Used On Current Block	#0031	Informational	Closed
Block Rate Assumed Constant	#0032	Informational	Open
Mixed Tab and Space Indentation	#0033	Informational	Closed
104 Weeks Will Cause Offset	#0034	Informational	Partially Mitigated
Overall And User Checkpoint Updated Simultaneously	#0035	Informational	Open
Unused Functions	#0036	Informational	Closed
Division Before Multiplication	#0037	Informational	Open

Findings

Code Analysis

Tokens With Transfer Fee Not Supported

FINDING ID	#0001
SEVERITY	Low Risk
STATUS	Mitigated
LOCATION	Rev 1 - LPStaker.sol -> 376-395

```
1     // Deposit LP tokens to for SHADE allocation.
2     function deposit(uint256 amount) public {
3         require(startTime != 0, "Not started");
4
5         UserInfo storage user = userInfo[msg.sender];
6
7         updatePool();
8
9         uint256 pending = user.amount * accSHADEPerShare / 1e12 -
  user.rewardDebt;
10
11        user.amount += amount;
12        user.rewardDebt = user.amount * accSHADEPerShare / 1e12;
13
14        _sendRewards(pending);
15
16        lpToken.safeTransferFrom(address(msg.sender), address(this),
  amount);
17        lpDeposited += amount;
18
19        emit Deposit(msg.sender, amount);
20    }
```

LOCATION	Rev 1 - LPStaker.sol -> 397-416

```
    // Withdraw LP tokens from MasterChef.
    function withdraw(uint256 amount) public {
        UserInfo storage user = userInfo[msg.sender];

        require(user.amount >= amount, "Not enough funds");

        updatePool();

        uint256 pending = user.amount * accSHADEPerShare / 1e12 -
user.rewardDebt;

        user.amount -= amount;
        user.rewardDebt = user.amount * accSHADEPerShare / 1e12;

        _sendRewards(pending);

        lpDeposited -= amount;
        lpToken.safeTransfer(address(msg.sender), amount);

        emit Withdraw(msg.sender, amount);
    }
```

DESCRIPTION	Contract 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.
RECOMMENDATION	Ensure that the deposit token used with this contract will never have a transfer fee. Alternatively, check the token balance before and after the transfer to get the actually received amount. In this case, it is necessary to ensure that there cannot be re-entrancy from the token transfer.
RESOLUTION	The project team has stated this contract will never use a token with a transfer fee for the deposit token.

Trusting External Contract

FINDING ID	#0002
SEVERITY	Low Risk
STATUS	Open
LOCATION	Rev 1 - LPStaker.sol -> 368-373

```
1    uint256 shadeReward = masterPending();
2
3    if (shadeReward != 0) {
4        masterChef.withdraw(masterPoolId, 0);
5        accSHADEPerShare += shadeReward * 1e12 / lpDeposited;

6    }
```

DESCRIPTION	If the *masterPending()* returns the wrong value there could be a discrepancy between the received amounts from the masterchef contract. Also if a transfer fee token is a reward then the amount can also differ.
RECOMMENDATION	Check the token balance before and after the transfer to get the actually received amount. In this case, it is necessary to ensure that there cannot be re-entrancy from the token transfer.
RESOLUTION	Project team comment: "Contract designed to work ONLY with ONE predefined token and CURRENT masterChef contract and it can't have any fees.".

No Timelock Implemented

FINDING ID	#0003
SEVERITY	Low Risk
STATUS	Mitigated
LOCATION	Rev 1 - LPStaker.sol -> 467-469

```
1 function setRewardsStaker(IRewardsStaker newRewardsStaker) external
  onlyOwner {
2     rewardsStaker = newRewardsStaker;
3 }
```

DESCRIPTION	The rewardsStaker contract should use a timelock as rewards are sent to this contract. If the rewardsStaker contract isn't working then it could lock deposit and withdraw and users would have to use *emergencywithdraw*.
RECOMMENDATION	Obelisk recommends a timelock delay of at least 72 hours.
RESOLUTION	Project Team has confirmed that a timelock will be applied on the deployed contract.

Contract Values Can Be Constant Or Immutable (Gas Optimization)

FINDING ID	#0004
SEVERITY	Informational
STATUS	Closed
LOCATION	• Rev 1 - LPStaker.sol -> 299: *IERC20 public shade;* • Rev 1 - LPStaker.sol -> 300: *IMasterChef public masterChef;* • Rev 1 - LPStaker.sol -> 301: *IRewardsStaker public rewardsStaker;*

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 recommended changes were implemented. Reviewed in commit 96048adc10c1d304feaef3bb5d01960dcb0f7a5f

Multiple Contracts In One File

FINDING ID	#0005
SEVERITY	Informational
STATUS	Closed
LOCATION	Rev 1 - LPStaker.sol

DESCRIPTION	The noted files contain multiple contracts.
RECOMMENDATION	Have each contract in its own file.
RESOLUTION	The recommended changes were implemented. Reviewed in commit [96048adc10c1d304feaef3bb5d01960dcb0f7a5f]

No Events Emitted For Changes To Protocol Values

FINDING ID	#0006
SEVERITY	Informational
STATUS	Closed
LOCATION	• Rev 1 - LPStaker.sol -> 476 *function depositToMaster(uint256 pid) external onlyOwner*

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	The recommended changes were implemented. Reviewed in commit [96048adc10c1d304feaef3bb5d01960dcb0f7a5f]

Withdraw Function Never Withdraws

FINDING ID	#0007
SEVERITY	High Risk
STATUS	Closed
LOCATION	Rev 2 - ShadeStaker.sol -> 312-326

```
1     function withdraw(uint256 amount) public nonReentrant {
2         require(amount != 0, "Can't withdraw 0");
3
4         _updateUserLocks(msg.sender);
5         _updateReward(msg.sender);
6         _claimReward(msg.sender);
7
8         Balances storage bal = balances[msg.sender];
9         require(amount <= bal.total - bal.locked, "Not enough
  unlocked tokens to withdraw");
10
11        bal.total -= amount;
12
13        _sendTokensAndPenalty(amount, 0);
14
15        emit Withdrawn(msg.sender, amount);
16    }
```

DESCRIPTION	The function is missing the actual transfer. This would wipe user's balance without sending the user any money.
RECOMMENDATION	Add a *safeTransfer()*.
RESOLUTION	A transfer call was added.

SafeTransfer Does Not Require Approval

FINDING ID	#0008
SEVERITY	High Risk
STATUS	Closed
LOCATION	Rev 2 - ShadeStaker.sol -> 511-525

```
1     function _sendTokensAndPenalty(uint256 tokensAmount, uint256
  penaltyAmount) internal {
2         if (penaltyAmount != 0 && address(penaltyReceiver) !=
  address(0)) {
3             if (penaltyReceiverIsContract) {
4                 stakingToken.approve(address(penaltyReceiver),
  penaltyAmount);
5                 penaltyReceiver.notifyReward(penaltyAmount);

6             } else {
7                 stakingToken.safeTransfer(address(penaltyReceiver),
  penaltyAmount);
8             }
9             emit PenaltyPaid(msg.sender, penaltyAmount);
10            stakingToken.safeTransfer(msg.sender, tokensAmount);
11        } else {
12            stakingToken.safeTransfer(msg.sender, tokensAmount +
  penaltyAmount);
13        }
14        totalSupply -= (tokensAmount + penaltyAmount);
15    }
```

DESCRIPTION	An unnecessary approval is dangerous and should not be in the code. Approving the same amount as the transfer amount will allow the recipient to withdraw the amount themselves again.
RECOMMENDATION	Do not approve for a *safeTransfer*.
RESOLUTION	The call to *approve()* was removed.

Lock Array Recreated Every Time Locks Change

FINDING ID	#0009
SEVERITY	High Risk
STATUS	Closed
LOCATION	Rev 2 - ShadeStaker.sol

DESCRIPTION	Multiple function will update the lock array by creating a new array, copying old values into this array, then replacing the old array. This is a very error prone operation, given how frequently it occurs. Furthermore, the gas costs of updating storage memory can be significant.
RECOMMENDATION	Simplify the logic and save gas by just shifting every value to the left and pop the top of the array. Since random gaps are created only in withdrawLock(uint256 id) shift values there instead as no random gaps have yet to be created. Then you can simply shift without any random gaps in the update function. Due to the complexity withdrawLock(uint256 id) introduces, implementing another datastructure such as OrderedSet, RenounceableQueue or a Linked List could be more beneficial.
RESOLUTION	Locks are not updated consistently. The new implementation uses mappings to avoid re-creating the entire array.

Unbound Loops On User Locks

FINDING ID	#0010
SEVERITY	Medium Risk
STATUS	Closed
LOCATION	• Rev 2 - ShadeStaker.sol -> 167-179: *for (uint i = 0; i < length; i++) {* • Rev 2 - ShadeStaker.sol -> 233-237: *for (uint i = 0; i < locks.length; i++) {* • Rev 2 - ShadeStaker.sol -> 241-247: *for (uint i = 0; i < locks.length; i++) {* • Rev 2 - ShadeStaker.sol -> 371-371: *for (uint i = 0; i < locks.length; i++) {* • Rev 2 - ShadeStaker.sol -> 383-385: *for (uint i = 0; i < lockedCount; i++) {* • Rev 2 - ShadeStaker.sol -> 458-470: *for (uint i = 0; i < locks.length; i++) {* • Rev 2 - ShadeStaker.sol -> 476-478: *for (uint i = 0; i < lockedCount; i++) {*

DESCRIPTION	Certain contracts can add an unlimited number of locks for a user. Iterating over an unbounded array can cause transactions to revert due to the gas limit.
RECOMMENDATION	Provide a limit to the size of the array.
RESOLUTION	A hard limit of 14 locks via the *lockDuration* limits the number of locks.

Checking Whether The Penalty Receiver Is A Contract Is Done Manually

FINDING ID	#0011
SEVERITY	Low Risk
STATUS	Closed
LOCATION	Rev 2 - ShadeStaker.sol -> 513-518

```
            if (penaltyReceiverIsContract) {
                stakingToken.approve(address(penaltyReceiver),
penaltyAmount);
                penaltyReceiver.notifyReward(penaltyAmount);

            } else {
                stakingToken.safeTransfer(address(penaltyReceiver),
penaltyAmount);
            }
```

LOCATION	Rev 2 - ShadeStaker.sol -> 104-108

```
    function setPenaltyReceiver(IPenaltyReceiver newPenaltyReceiver,
bool isContract) public onlyOwner {
        penaltyReceiver = newPenaltyReceiver;
        penaltyReceiverIsContract = isContract;
        emit SetPenaltyReceiver(address(newPenaltyReceiver),
isContract);
    }
```

DESCRIPTION	A boolean parameter is used to determine if the *penaltyReceiver* is a contract.
RECOMMENDATION	Use *isContract* to check if it's a contract. Do note the dangers with this function as described in the implementation comments: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/Address.sol#L36
RESOLUTION	The project team has implemented the recommended change.

Rounding Precision

FINDING ID	#0012
SEVERITY	Low Risk
STATUS	Closed
LOCATION	Rev 2 - ShadeStaker.sol ->286

```
1            uint256 unlockTime = (block.timestamp / rewardsDuration *
  rewardsDuration) + lockDuration;
```

LOCATION	Rev 2 - ShadeStaker.sol -> 348-349 Rev 2 - ShadeStaker.sol -> 389-390

```
1        uint256 penalty = amount / 2;
2        amount -= penalty;
```

DESCRIPTION	A number of locations in the contract, including the ones noted above, have potential rounding errors. For example, at line 286, the math might cause the following rounding: (123 /10 * 10) +2 = 122
RECOMMENDATION	Ensure that the correct precision is used in all arithmetic operations.
RESOLUTION	Project team has stated that the rounding behaviour is intentional.

Inconsistent Logic

FINDING ID	#0013
SEVERITY	Low Risk
STATUS	Closed
LOCATION	Rev 2 - ShadeStaker.sol -> 168

```
1                   if (locks[i].unlockTime > block.timestamp) {
```

LOCATION	Rev 2 - ShadeStaker.sol -> 288

```
1             if (locksLength == 0 || userLocks[account][locksLength-
  1].unlockTime < unlockTime) {
```

LOCATION	Rev 2 - ShadeStaker.sol -> 448

```
1         if (locks[length-1].unlockTime <= block.timestamp) {
```

DESCRIPTION	The noted conditionals are almost the same, but subtly different. In contracts with significant branching, it is important that the conditionals used are clear and easy to understand. In particular, the third one noted (line 288) uses the < operator as opposed to the <= operator.
RECOMMENDATION	Use the same conditional statement for consistency, wherever possible. In this case, also confirm whether the discrepancies are intentional.
RESOLUTION	Operators have been changed to be more consistent.

Global Id For Lock

FINDING ID	#0014
SEVERITY	Informational
STATUS	Closed
LOCATION	Rev 2 - ShadeStaker.sol -> 290-294

```
1                userLocks[account].push(LockedBalance({
2                    amount: amount,
3                    unlockTime: unlockTime,
4                    id: lockIds
5                }));
```

DESCRIPTION	Every deposit is assigned a unique lock. Yet this lockid is never used to fetch the individual lock. Currently, there is no way to check what funds are in a given lock on-chain.
RECOMMENDATION	Remove the lock id mechanism as they add unnecessary complexity.
RESOLUTION	Withdrawing on a per-lock basis has been removed.

Blank Elements May Be Left In Array

FINDING ID	#0015
SEVERITY	Low Risk
STATUS	Closed
LOCATION	Rev 2 - ShadeStaker.sol -> 458-482

```
for (uint i = 0; i < length; i++) {
        if (locks[i].unlockTime > block.timestamp) {
            // if lock not expired adding amount to total locked
            lockedAmount = lockedAmount + locks[i].amount;
            if (length > lockDurationMultiplier) {
                newLocks[lockedCount] = locks[i];
                lockedCount ++;
            }
        } else {
            // if expired delete it
            delete locks[i];
        }
    }

    // let's get rid of empty locks (gaps) on the beginning of array if they are
    // the reason is to not allow array to grow
    if (length > lockDurationMultiplier && lockedCount != 0 && length > lockedCount) {
        delete userLocks[account];
        for (uint i = 0; i < lockedCount; i++) {
            userLocks[account].push(newLocks[i]);
        }
    }

    bal.locked = lockedAmount;
```

DESCRIPTION	The if statement checks the *lockDurationMultiplier* and compares it to the number of locks at the start of the function. There is a likelihood that the user locks will not be correctly updated to use the new locks.
RECOMMENDATION	Clarify the logic of updating the locks.
RESOLUTION	Project team simplified the check logic such that it will always remove blank elements.

Redundant Division And Multiplication

FINDING ID	#0016
SEVERITY	Informational
STATUS	Closed
LOCATION	Rev 2 - ShadeStaker.sol -> 286

```
1            uint256 unlockTime = (block.timestamp / rewardsDuration *
  rewardsDuration) + lockDuration;
```

DESCRIPTION	The division is cancelled out by the multiplication.
RECOMMENDATION	Remove the redundant operations or confirm whether this behaviour is intended.
RESOLUTION	The project team has confirmed that this behaviour is intended.

Loops Can Be Combined

FINDING ID	#0017
SEVERITY	Informational
STATUS	Closed
LOCATION	Rev 2 - ShadeStaker.sol -> 233-248

```
1             for (uint i = 0; i < locks.length; i++) {
2                 if (locks[i].amount != 0 && locks[i].unlockTime >
  block.timestamp) {
3                     locksCount ++;
4                 }
5             }
6             LockedBalance[] memory _userLocks = new LockedBalance[]
  (locksCount);
7             if (locksCount != 0) {
8                 uint256 idx;
9                 for (uint i = 0; i < locks.length; i++) {
10                    if (locks[i].amount != 0 && locks[i].unlockTime >
  block.timestamp) {
11                        _userLocks[idx] = locks[i];
12                        _balances.locked += locks[i].amount;
13                        idx ++;
14                    }
15                }
16            }
```

DESCRIPTION	These loops are nearly identical.
RECOMMENDATION	Combine the functionality of the loops. Before the array is looped through the first time, create a tempArray. Instead of incrementing *locksCount*, push to that tempArray and you will end up with a *tempArray* of equal length without having to loop through it again.
RESOLUTION	The loops were combined.

Unbounded Loop

FINDING ID	#0018
SEVERITY	Informational
STATUS	Closed
LOCATION	• Rev 2 - ShadeStaker.sol -> 145-148: *for (uint256 i = 0; i < rewardsAvailable.length; i++) {* • Rev 2 - ShadeStaker.sol -> 201-205: *for (uint i; i < rewardTokens.length; i++) {* • Rev 2 - ShadeStaker.sol -> 499-507: *for (uint i; i < rewardTokens.length; i++) {*

DESCRIPTION	Iterating over an unbounded array can cause transactions to revert due to the gas limit.
RECOMMENDATION	Provide a limit to the size of the array. Alternatively, pass a lower and upper index as parameters and iterate over a range.
RESOLUTION	An upper bound of 10 tokens was added.

Lock Is Never Removed

FINDING ID	#0019
SEVERITY	High Risk
STATUS	Closed
LOCATION	Rev 3 - ShadeStaker.sol -> 442-453

```
1         LockedBalance[] memory locks = userLocks[msg.sender];
2
3         uint256 amount;
4
5         // AUDIT Finding Id: 4
6         // length can't be more than lockDurationMultiplier (13) + 1
7         for (uint i = 0; i < locks.length; i++) {
8             if (locks[i].id == id) {
9                 amount = locks[i].amount;
10                delete locks[i];
11            }
12        }
```

DESCRIPTION	A local copy of *locks* is made (*userLocks[msg.sender]*). Then a lock in the local copy is deleted. (*delete locks[i];*) The storage lock is thus never deleted.
RECOMMENDATION	Change the reference type from memory to storage.
RESOLUTION	The lock mechanism was changed to use mapping, resolving this issue.

Protocol Values Should Be Public

FINDING ID	#0020
SEVERITY	Informational
STATUS	Closed
LOCATION	• Rev 4 - ShadeStaker_3_mapping.sol -> 442-453: mapping(address=> mapping(address => bool)) public rewardDistributors • Rev 4 - ShadeStaker_3_mapping.sol -> 442-453: mapping(address => bool) public lockStakers

DESCRIPTION	Variables critical to the operation of the protocol should be public or have an associated view function. Mappings are not iterable and therefore it may be challenging to identify all the distributors and lockStakers.
RECOMMENDATION	Add an array that mirrors the contents of the mapping.
RESOLUTION	Arrays were added which track all addresses which ever received the elevated permissions.

Mixed Tab and Space Indentation

FINDING ID	#0021
SEVERITY	Informational
STATUS	Closed
LOCATION	Rev 4 - ShadeStaker_3_mapping.sol

DESCRIPTION	The contract uses mixed tab and space indentation. This can cause the contract indentation to appear “incorrect” (for example on github).
RECOMMENDATION	Use a consistent indentation method.
RESOLUTION	The project team has implemented the recommended change.

Incorrect Reward Duration Calculation

FINDING ID	#0022
SEVERITY	Medium Risk
STATUS	Open
LOCATION	Rev-6 - xShadeRewardsDistributor.sol -> 297-298

```
1                int256 delta = int256(weekCursor -
  oldUserPoint.timeStamp);
2                uint256 balanceOf =
  uint256(SignedMath.max(oldUserPoint.bias - (delta *
  oldUserPoint.slope), 0));
```

DESCRIPTION	The value of *delta* should represent the time elapsed within the current epoch, but because *oldUserPoint* may be uninitialized at that point, it will result in a drastically larger delta value.
RECOMMENDATION	Correctly calculate the time delta.
RESOLUTION	N/A

Shade Points May Not Align With Reward Distributor Epochs

FINDING ID	#0023
SEVERITY	Medium Risk
STATUS	Open
LOCATION	Rev-6 - xShadeRewardDistributor.sol -> 230-236

```
                uint256 epoch = _findTimestampEpoch(_timeCursor);
                Point memory point = xShadeToken.pointHistory(epoch);
                int256 delta;
                if (_timeCursor > point.timeStamp) {
                    delta = int256(_timeCursor - point.timeStamp);
                }
                xShadeSupply[_timeCursor] =
uint256(SignedMath.max(point.bias - (point.slope * delta), 0));
```

LOCATION	Rev-6 - xShadeRewardDistributor.sol -> 297-307

```
                int256 delta = int256(weekCursor -
oldUserPoint.timeStamp);
                uint256 balanceOf =
uint256(SignedMath.max(oldUserPoint.bias - (delta *
oldUserPoint.slope), 0));
                if (balanceOf == 0 && userEpoch > maxUserEpoch) {
                    break;
                }
                if (balanceOf > 0) {
                    if (xShadeSupply[weekCursor] != 0) {
                        rewardsToDistribute += balanceOf *
rewardsPerWeek[weekCursor] / xShadeSupply[weekCursor];
                    }
                }
```

DESCRIPTION	The value of the total supply and user points is assumed to be linear between epochs. If the total supply or user balances are non-linear between epochs, then the calculations of their values will be incorrect.
RECOMMENDATION	Ensure that total supply and balance calculations are consistent between contracts. Alternatively, move the calculation to the xShadeToken contract and allow other addresses to request a value at a given timestamp.
RESOLUTION	Project team comment: "Since totalSupply of xShade goes

down with time we recalculate it every time user claim. Users receive rewards the week after they appear, and by knowing totalSupply at end of the epoch we can calculate it at the given timestamp"

Use Safe Transfer

FINDING ID	#0024
SEVERITY	Low Risk
STATUS	Closed
LOCATION	• Rev-6 - xShadeRewardsDistributor.sol -> 339: *IERC20(rewardToken).transfer(addr, amount);* • Rev-6 - xShadeRewardsDistributor.sol -> 368: *IERC20(rewardToken).transfer(msg.sender, amount);*

DESCRIPTION	Direct transfer functions are called.
RECOMMENDATION	Use openzeppelin's safe transfer functions. These safe transfer functions are used to catch when a transfer fails as well as unusual token behavior.
RESOLUTION	The project team implemented the recommended changes.

Rewards Not Allocated If Last Reward Was Over 20 Weeks Ago

FINDING ID	#0025
SEVERITY	Low Risk
STATUS	Partially Closed
LOCATION	Rev-6 - xShadeRewardDistributor.sol -> 185-203

```
        for (uint256 i = 0; i < 20; i++) {
            nextWeek = thisWeek + WEEK;
            if (block.timestamp < nextWeek) {
                if (sinceLast == 0 && block.timestamp == _lastRewardsTime) {
                    rewardsPerWeek[thisWeek] += rewardsToDistribute;
                } else {
                    rewardsPerWeek[thisWeek] += rewardsToDistribute * (block.timestamp - _lastRewardsTime) / sinceLast;
                }
                break;
            } else {
                if (sinceLast == 0 && nextWeek == _lastRewardsTime) {
                    rewardsPerWeek[thisWeek] += rewardsToDistribute;
                } else {
                    rewardsPerWeek[thisWeek] += rewardsToDistribute * (nextWeek - _lastRewardsTime) / sinceLast;
                }
            }
            _lastRewardsTime = nextWeek;
            thisWeek = nextWeek;
        }
```

DESCRIPTION	The rewards per week are not allocated correctly if the number of weeks to reward will exceed 20 weeks. Note that the calculation logic of the `rewardsPerWeek` is highly inconsistent. For example, some of the branches will never be executed.
RECOMMENDATION	Consolidate the reward distribution logic.
RESOLUTION	The number of weeks was extended to 104 weeks (approximately 2 years).

Rewards Distributed To Current Epoch May Not Be Collected

FINDING ID	#0026
SEVERITY	Low Risk
STATUS	Open
LOCATION	Rev-6 - xShadeRewardDistributor.sol -> 294-312

```
        for (uint256 i = 0; i < 50; i++) {
            if (weekCursor >= _lastRewardsTime) {
                break;
            }
            // ...
        }
        userEpoch = Math.min(maxUserEpoch, userEpoch - 1);
        userEpochOf[addr] = userEpoch;
        timeCursorOf[addr] = weekCursor;
```

DESCRIPTION	Rewards can be distributed to the current epoch, but once a user has collected rewards for an epoch (including the current one), they can no longer collect from it again. This will lead to some rewards being permanently locked.
RECOMMENDATION	Distribute rewards to future epochs only. Note that the reward mechanism is very incoherent and should be re-written to avoid logic errors.
RESOLUTION	Project team comment: "User start receiving rewards the week after he make first deposit" Obelisk comment: "This will not resolve users being unable to collect additional rewards during a week."

Shade, Reward, and Staking Token Addresses Hard Coded

FINDING ID	#0027
SEVERITY	Low Risk
STATUS	Closed
LOCATION	Rev-6 - xShadeRewardDistributor.sol -> 79-81

```
1         xShadeToken =
  IXSHADE(0xE870920B89373503D29578522706230174894 2A2);
2         isRewardsFTM = true; // if true reward token MUST be Wrapped
  FTM
3         rewardToken = 0x15c34D8b356F21112C07cA1811D84101F480a3F1;
```

DESCRIPTION	The shade and reward tokens are hard coded. Since *isRewardsFTM* is also set to true, the *rewardToken* should refer to WFTM. However, the address is invalid.
RECOMMENDATION	Add parameters to the constructor to allow for more flexible deployment.
RESOLUTION	Addresses are passed in as a parameter.

Memory Reference Assignment

FINDING ID	#0028
SEVERITY	High Risk
STATUS	Closed
LOCATION	Rev-6 - XShade.sol -> 410

```
1         Point memory initialLastPoint = lastPoint;
```

LOCATION	Rev-6 - XShade.sol -> 443-444

```
1             lastPoint.timeStamp = timeStamp;
2             lastPoint.blockNumber = initialLastPoint.blockNumber +
  ((blockSlope * (timeStamp - initialLastPoint.timeStamp)) /
  MULTIPLIER);
```

LOCATION	Rev-6 - XShade.sol -> 513-537

```
1     function _depositFor(
2         address account,
3         uint256 amount,
4         uint256 unlockTime,
5         LockedBalance memory locked,
6         LockAction action
7     ) internal {
8         LockedBalance memory oldLocked = locked;
9         // ...
10        _checkpoint(account, oldLocked, locked);
11        // ...
12    }
```

DESCRIPTION	Assignment of memory data does not create a copy by default. Modifying *lastPoint* will change the values of *initialLastPoint*. Modifying *locked* will change the values of *oldLocked*. Example: *uint[] memory memoryArray = [1,2,3];* *uint[] memory pointerArray = memoryArray;*

	pointerArray.push(9); // both point to same array of [1, 2, 3, 9]
	https://www.ajaypalcheema.com/assignment-by-value-vs-reference-in-solidity/

RECOMMENDATION	Create an explicit copy of the data.
RESOLUTION	A explicit copy is now made where necessary.

User Can Lock Funds For As Little As A Single block

FINDING ID	#0029
SEVERITY	Low Risk
STATUS	Closed
LOCATION	Rev-6 - XShade.sol -> 306-316

```
    unlockTime = unlockTime / WEEK * WEEK; // Locktime is rounded down to weeks

    require(unlockTime > block.timestamp, "Can only lock until time in the future");

        uint256 roundedMin = block.timestamp / WEEK * WEEK + MINTIME;
        uint256 roundedMax = block.timestamp / WEEK * WEEK + MAXTIME;
        if (unlockTime < roundedMin) {
            unlockTime = roundedMin;
        } else if (unlockTime > roundedMax) {
            unlockTime = roundedMax;
        }
```

DESCRIPTION	Since the *unlockTime* is rounded down to the week, the funds can be locked for as little as a single block.
RECOMMENDATION	Add a minimum lock time.
RESOLUTION	*MINTIME* is now set to *WEEK * 2* meaning the minimum lock time will be 7 days + 1 block to a maximum of 14 days - 1 block. The locktime will depend on when during the week a lock is created.

Lock Time Can Be Decreased

FINDING ID	#0030
SEVERITY	Low Risk
STATUS	Closed
LOCATION	Rev-6 - XShade.sol -> 333-346

```
1     function increaseLockTime(uint256 unlockTime) external
  nonReentrant notContract notExpired {
2         LockedBalance memory locked = lockedBalances[msg.sender];
3
4         require(locked.amount != 0, "No existing lock found");
5         require(locked.end >= block.timestamp, "Lock expired.
  Withdraw old tokens first");
6
7     uint256 maxUnlockTime = block.timestamp / WEEK * WEEK + MAXTIME;
8     require(locked.end != maxUnlockTime, "Already locked for maximum
  time");
9
10        unlockTime = unlockTime / WEEK * WEEK; // Locktime is rounded
  down to weeks
11    require(unlockTime <= maxUnlockTime, "Can't lock for more than
  max time");
12
13        _depositFor(msg.sender, 0, unlockTime, locked,
  LockAction.INCREASE_LOCK_TIME);
14    }
```

LOCATION	XShade.sol -> 528-532

```
1     if (unlockTime != 0) {
2             locked.end = unlockTime;
3         }
4
5         lockedBalances[account] = locked;
```

DESCRIPTION	A user can use the *increaseLockTime* function to set their unlock time to any timestamp. As a result, they will be able to withdraw immediately.
RECOMMENDATION	Ensure that the lock time can only increase.
RESOLUTION	A *require* was added to ensure that a lock time can not

	decrease.

balanceOfAt Should Not Be Used On Current Block

FINDING ID	#0031
SEVERITY	Informational
STATUS	Closed
LOCATION	Rev-6 - XShade.sol -> 140-185: *function balanceOfAt(address account, uint256 blockNumber)*

DESCRIPTION	Because the balance can change within a block, this function should not be used on the current block number.
RECOMMENDATION	Add this to the documentation or remove the ability to query the current block.
RESOLUTION	A natspec notice was added to the function warning against using this function on the current block.

Block Rate Assumed Constant

FINDING ID	#0032
SEVERITY	Informational
STATUS	Open
LOCATION	Rev-6 - XShade.sol -> 165-177

```
1         if (blockEpoch < epoch) {
2             Point memory point1 = pointHistory[blockEpoch + 1];
3             deltaBlockNumber = point1.blockNumber -
  point0.blockNumber;
4             deltaTimeStamp = point1.timeStamp - point0.timeStamp;
5         } else {
6             deltaBlockNumber = block.number - point0.blockNumber;
7             deltaTimeStamp = block.timestamp - point0.timeStamp;
8         }
9
10        uint256 blockTime = point0.timeStamp;
11        if (deltaBlockNumber != 0) {
12            blockTime += (deltaTimeStamp * (blockNumber -
  point0.blockNumber)) / deltaBlockNumber;
13        }
```

LOCATION	Rev-6 - XShade.sol -> 220-231

```
1         Point memory point = pointHistory[targetEpoch];
2         uint256 delta;
3         if (targetEpoch < epoch) {
4             Point memory pointNext = pointHistory[targetEpoch + 1];
5             if (point.blockNumber != pointNext.blockNumber) {
6                 delta = ((blockNumber - point.blockNumber) *
  (pointNext.timeStamp - point.timeStamp)) / (pointNext.blockNumber -
  point.blockNumber);
7             }
8         } else {
9             if (point.blockNumber != block.number) {
10                delta = ((blockNumber - point.blockNumber) *
  (block.timestamp - point.timeStamp)) / (block.number -
  point.blockNumber);
11            }
12        }
```

DESCRIPTION	The calculation of a timestamp for a given block number is done by interpolating between the values in saved `Point`

	objects. This assumes that the rate of blocks is constant.
RECOMMENDATION	Be aware that the block timestamp may differ when designing the front-end or associated contracts.
RESOLUTION	The project team has acknowledged the issue.

Mixed Tab and Space Indentation

FINDING ID	#0033
SEVERITY	Informational
STATUS	Closed
LOCATION	Rev-6 - XShade.sol

DESCRIPTION	The contract uses mixed tab and space indentation. This can cause the contract indentation to appear “incorrect” (for example on github).
RECOMMENDATION	Use a consistent indentation method.
RESOLUTION	The project team implemented the recommended change.

104 Weeks Will Cause Offset

FINDING ID	#0034
SEVERITY	Informational
STATUS	Partially Mitigated
LOCATION	Rev-6 - XShade.sol -> 58

```
1    uint256 constant MAXTIME = WEEK * 104; // 2 years
```

DESCRIPTION	Since there is a different number of weeks in each year, this will cause an offset after a number of years pass.
RECOMMENDATION	Keep this in mind when designing the front-end.
RESOLUTION	Project team comment: "On front-end we will display unlock time and \~2 years is only aproximisation"

Overall And User Checkpoint Updated Simultaneously

FINDING ID	#0035
SEVERITY	Risk Rating
STATUS	Open
LOCATION	Rev-6 - XShade.sol -> 367-503

```
1    function _checkpoint(
2        address account,
3        LockedBalance memory oldLocked,
4        LockedBalance memory newLocked
5    ) internal {
6        // ...
7    }
```

DESCRIPTION	The XShade contract updates the overall and user checkpoints at the same time. This leads to complex logic which may cause unpredictable behavior.
RECOMMENDATION	Separate and simplify these systems.
RESOLUTION	The project team has acknowledged the issue.

Unused Functions

FINDING ID	#0036
SEVERITY	Informational
STATUS	Closed
LOCATION	Rev-6 - xShadeRewardsDistributor.sol -> 17-19: *function min(int256 a, int256 b) internal pure returns (int256) {*

DESCRIPTION	The noted functions are never used.
RECOMMENDATION	Remove the functions.
RESOLUTION	The noted function was removed.

Division Before Multiplication

FINDING ID	#0037
SEVERITY	Informational
STATUS	Open
LOCATION	Rev-6 - xShade.sol -> 382-383

```
1                 userOldPoint.slope = int256(oldLocked.amount /
  MAXTIME);
2                 userOldPoint.bias = userOldPoint.slope *
  int256(oldLocked.end - block.timestamp);
```

LOCATION	Rev-6 - xShade.sol -> 386-387

```
1                 userNewPoint.slope = int256(newLocked.amount /
  MAXTIME);
2                 userNewPoint.bias = userNewPoint.slope *
  int256(newLocked.end - block.timestamp);
```

DESCRIPTION	The calculations noted use mixed orders of multiplication and division. This may cause rounding errors, resulting in reverted transactions or miscalculations in general.
RECOMMENDATION	Change the calculations to first multiply, then divide.
RESOLUTION	N/A

On-Chain Analysis

Not Analyzed

Appendix A - Reviewed Documents

Document	Address
LPStaker.sol	N/A
ShadeStaker.sol	N/A

Revisions

Revision 1	Zip file
Revision 2	96048adc10c1d304feaef3bb5d01960dcb0f7a5f
Revision 3	0a27b3adb9b30f5aa9a713899ad71c3729579e81
Revision 4	9b413a875788ca04f5c08f7a03397eabe76228f2
Revision 5	b402df61e6ecf03c31238e24067bf1966d683f5c
Revision 6	c534a76a75ca7c12ad3c5ee103a53a4b2bc52213
Revision 7	635dfb28c0bc29f846bdf39406575cb3b82e4867

Imported Contracts

-	-

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 - 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