SHYFT
SMART CONTRACT AUDIT

ZOKYO.

April 12th, 2022 | v. 1.0

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

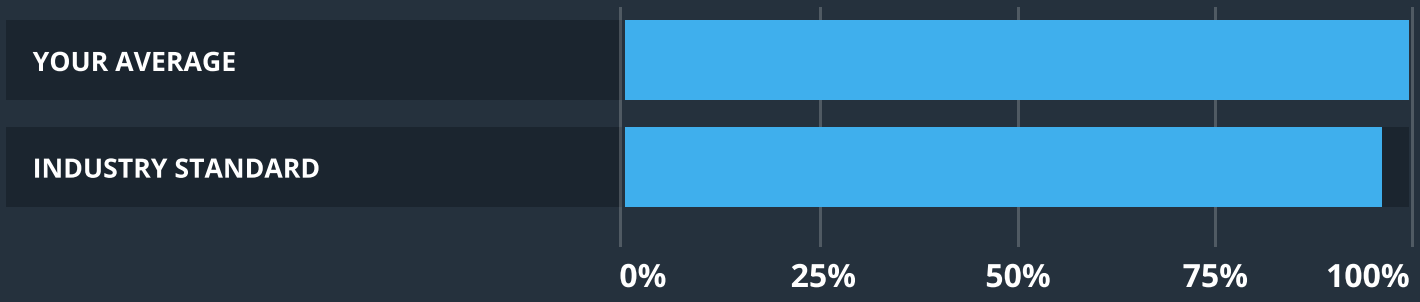
TECHNICAL SUMMARY

This document outlines the overall security of the Shyft smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the Shyft smart contract codebase for quality, security, and correctness.

Contract Status

Testable Code



The testable code is 97%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a security of the contract we at Zokyo recommend that the Shyft put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Shyft smart contract's source code was taken from the repositories provided by Shyft:
https://github.com/ShyftNetwork/StakingModule (master branch)
Audited commit: ad5270459e8484f8cbf15f13d72e890a65ce4780
Last commit (post-audit): 4aa9b7a58a1ac9489bac846ab8be7a037495c4e3
https://github.com/ShyftNetwork/shyft_shyftcorecontracts (alex/inflationRequirements branch)
Audited commit: 59393f880127764cb4afff8efeda589b0ed66ef7
Last commit (post-audit): 59393f880127764cb4afff8efeda589b0ed66ef7

Within the scope of this audit Zokyo auditors have reviewed the following contract(s):

- ShyftBlockReward.sol
- ShyftStaking.sol

Throughout the review process, care was taken to ensure that the contract:

- Implements and adheres to existing standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of resources, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of Shyft smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1	Due diligence in assessing the overall code quality of the codebase.	3	Testing contract logic against common and uncommon attack vectors.
2	Cross-comparison with other, similar smart contracts by industry leaders.	4	Thorough, manual review of the codebase, line-by-line.

EXECUTIVE SUMMARY

There were no critical vulnerabilities found during the audit. Nevertheless, there were found several issues connected to the Eth transferring (the standard issue after the Istanbul fork) and with tracking correct amounts of stakes and rewards.

Though, all issues were resolved by Shyft team and additionally tested by Zokyo Security team.

All other issues are connected to admin actions verification and code style.

Zokyo Security team has prepared the full set of unit-tests to check the correctness of the contracts logic.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

HIGH | RESOLVED

Incorrect Ether transfer

ShyftStaking.sol, unstake() (line 267), notifyRewardAmount() (line 316), finishPrePurchasersMode() (line 424), _getReward() (line457)
After the Istanbul and Berlin updates, sending Eth with transfer() and send() methods is forbidden since it may fail during fallback function call because of the not enough forwarded gas.
Thus, the only correct for m of Eth sending is with the call() method, for example:
(bool success,) = addr.call{value: val}(“”);
require(success, “Failed Eth transfer”);

Recommendation
Use the call() method for Eth sending.

MEDIUM | RESOLVED

Missing checks to distinguish staked funds and reward funds

ShyftStaking.sol, unstake(), finishPrePurchasersMode(), _getReward()
The contract operates with the same currency for the rewards and for the user stakes. In spite of the stakes being tracked within totalSupply variable, there are no checks against that value during the rewards distribution or transfers to the users.
So, in order to mitigate possible scenarios where (as an effect of the calculations with edge cases parameters) rewards amount may exceed the actual distributed value (and get the part of the supplied funds) we recommend to add checks that reward amount does not exceed the distributed value. So, in order to mitigate possible scenarios where (as an effect of the calculations with edge cases parameters) rewards amount may exceed the actual distributed value (and get the part of the supplied funds) we recommend to add checks that reward amount does not exceed the distributed value.

Recommendation:
For the rewards transfers add checks like: rewardToTransfer <= address(this).balance.sub(totalSupply).

MEDIUM | RESOLVED

Missing check for the rewards available

ShyftStaking.sol, finishPrePurchasersMode()
The function relies on the fact that the contract already has rewards distributed. Though there is no security checks confirming this fact. Thus the contract may calculate amounts to be returned incorrectly and draw staked amounts from the users.

Recommendation:
Consider adding checks for the lastUpdateTime to be greater than 0 (so the rewards are started).

MEDIUM | RESOLVED

Incorrect view value

ShyftStaking.sol, rewardPerShft()
The function should return 0 in case if rewards were not distributed yet - in case if lastUpdateTime equals 0. This refers to the period when pre-purchasers are added and rewards are not distributed yet.

Recommendation:
Correct the method.

MEDIUM | UNRESOLVED

Function execution reverts.

ShyftBlockReward.sol, function setupExData().
In case, sender of transaction is administratorAddress, function performs internal transaction of its own function(Lines 294-296), which also check, that a sender is administratorAddress. However, when contract performs this calls, it becomes sender instead of administratorAddress, which means that functions won't update storage variables.

Recommendation:
Instead of performing internal calls, make functions setExRewardsEnabled(), setExCallsEnabled() and setExContractAddress() and call them without using "this" object.

LOW | UNRESOLVED

Missing visibility identifier

ShyftStaking.sol, prePurchasersModeOn (line 87)
ShyftBlockReward.sol: lockExAdministrator, benefactorCoinbase, benefactorDelegate, delegateOriginalBenefactor, blockSealerHistoryLastBlockSealedMapping, blockSealerHistoryMapping, blockSealerHistoryArray.
Visibility is missing (private by default), so should be verified for the correct usage of the storage and access.

Recommendation:
Add visibility identifier.

LOW | UNRESOLVED

Missing unbond per index

ShyftStaking.sol, _unbond()
The function misses the unbond with the index 0 - all ids start from the index 1, which is untraditional and controversial.

Recommendation:
Verify the functionality and check of the calculation should start from 0 index.

INFORMATIONAL | RESOLVED

Constant can be used

ShyftStaking.sol, unbondingPeriod (line 26)
The value is set just once during the initialization and is not changed throughout the contract. Also there is no setter for the variable. Thus it can be set as the constant or the immutable variable constructed in the implementation.

Recommendation:
Consider using the constant.

	ShyftStaking.sol	ShyftBlockReward.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Shyft team

As part of our work assisting Shyft in verifying the correctness of their contract code, our team was responsible for check of the existing set of tests and writing own integration tests using Truffle testing framework up to the full coverage.

It needs to be mentioned, that the code has some original coverage with testing scenarios provided by the Shyft team. All of them were also carefully checked by the auditors' team to be consistent and meaningful.

```
Contract: Shyft Staking
  When providing rewards
  Works
  When there are two normal stakers and two prepurchasers
    ✓ should be equal if marketAveragePrice <= lowestBoundPrice (53ms)
    ✓ should be equal if daoMultiplier is 1 and currentPrice >= marketAveragePrice >=
    lowestBoundPrice (55ms)
    ✓ should be half when dao multiplier is 0.5 and currentPrice >= marketAveragePrice
    >= lowestBoundPrice
    ✓ should be zero if market average price is huge, but still users should have correct
    rewards (86ms)
  When checking staking functionalities
  Succeeds
  On Basic Staking
    ✓ When a normal user is staking (38ms)
    ✓ should revert when passing 0 ether (40ms)
    ✓ When passing prepurchasers
    ✓ reverts when trying to provide wrong prepurchasers input
    ✓ reverts when trying to provide 0 amount
    ✓ reverts when trying to provide same prepurchaser again
    ✓ reverts when trying to add prepurchasers after rewards have been provided (39ms)
  On Providing Rewards
  When there are two normal stakers and two prepurchasers
    ✓ should revert when prepurchaser is trying to normally stake before prepurchasers
    mode goes off
    ✓ should revert if prepurchaser is trying to call stakePrePurchaser() twice
    ✓ should revert if prepurchaser is trying to call stakePrePurchaser() with not the
    whole amount
    ✓ should revert if staker is trying to call stakePrePurchaser()
```

```
And one prepurchaser is not staking before mode goes off
  ✓ His rewards should be returned back to rewardDistribution
  ✓ Rewards should be provided correctly afterwards (60ms)
  ✓ should work when prepurchaser is trying to stake after prepurchaser mode goes off
When checking unstaking functionalities
Works
When there are two normal stakers and two prepurchasers
  ✓ should revert if trying to unbond before prepurchasers period ends
  ✓ should revert if prepurchaser tries to unbond after period but without staking
  ✓ should succeed if trying to unbond after prepurchasers period using unbondAll()
  ✓ should succeed if trying to unbond after prepurchasers period using unbond()
  ✓ should get reward as well when using unbondAll() (51ms)
And one normal staker and prepurchaser have unbonded
  ✓ should revert if trying to unstake before unbonding period ends
  ✓ should revert if trying to unstake not owned unbonding id
  ✓ should revert if trying to unstake more than unbonding id contains
  ✓ should be able to unstake after unbonding period has finished
Contract: ShyftStaking upgrade test
  ✓ Shyft Staking should be upgraded (180ms)
28 passing (3s)
```

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Secured team

As part of our work assisting Shyft team in verifying the correctness of their contract code, our team was responsible for writing integration tests using Truffle testing framework.

Tests were based on the functionality of the code, as well as review of the Shyft contract requirements for details about issuance amounts and how the system handles these.

Contract: ShyftStaking

- ✓ Initializing (3228ms)
- ✓ Reward per shift must be null (55ms)
- ✓ Add prepurchaser (1594ms)
- ✓ Stake proper amount (404ms)
- ✓ Can't stake unproper amount (307ms)
- ✓ Can't stake before release (prepurchasers) (293ms)
- ✓ Must stake the whole amount (976ms)
- ✓ Stake proper amount after release (prepurchsers) (946ms)
- ✓ Can't stake as prepurchaser again (302ms)
- ✓ Can't stake if caller is not added as prepurchaser (309ms)
- ✓ Can't update if last reward update not greater than null (217ms)
- ✓ Can't notify rewards not from reward destributor (265ms)
- ✓ Can't notify rewards if not the whole rewards has been sent (234ms)
- ✓ Rewards must be sent before notifying (if market price is lower than lower bound) (483ms)
- ✓ Rewards must be sent before notifying (if market price is higher than lower bound and lower than current price) (737ms)
- ✓ Rewards must be sent before notifying (if market price is higher than lower bound and higher than current price) (706ms)
- ✓ Get reward works properly (366ms)
- ✓ Finishes prepurchaser mode correct (388ms)
- ✓ Can't stake as prepurchaser if prepurchaser mode is not ON (276ms)
- ✓ Staking as prepurchaser after prepurchaser mode finish (424ms)
- ✓ Can't unbound 0 and more than staked (1284ms)
- ✓ Unbond works properly (724ms)
- ✓ Unbond works properly (prepurchasers) (2193ms)
- ✓ Unbond all works properly (759ms)
- ✓ Can't unstake before unbonding period ends (460ms)
- ✓ Can't unstake not from the owner of unbond (355ms)
- ✓ Unstakes properly (366ms)
- ✓ Can't unstake more than remaining amount in unbonding id (301ms)

✓ Setters (2654ms)
✓ View methods (554ms)

Contract: **ShyftBlockReward**

✓ Should return system address
✓ Should reset administrator (154ms)
✓ Should set administrator (146ms)
✓ Should remove administrator (81ms)
✓ Should set rewards enabled (68ms)
✓ Should set calls enabled (72ms)
✓ Should set ex contract address (99ms)
✓ Should set price feed pair address (82ms)
✓ Should set shyft dao address (115ms)
✓ Should authorize reward reduction (40ms)
✓ Should throttle self reward (59ms)
✓ Should not throttle self reward if amount > max reward (173ms)
✓ Should produce rewards for benefactors (370ms)
✓ Should revert producing rewards if called not by system address (121ms)
✓ Should revert producing rewards if benefactors and kind length mismatch (115ms)
✓ Should produce external reward as 0 if ex calls enabled is false (450ms)
✓ Should produce ex contract reward and should not update ex contract address and benefactor (325ms)
✓ Should not update ex contract during producing rewards (530ms)
✓ Should produce ex contract reward and update ex contract address and benefactor (495ms)
✓ Should set sealer reward as min sealer reward (343ms)
✓ Should produce self throttled reward for benefactor (216ms)
✓ Should set delegator for benefactor (212ms)
✓ Should not set delegator for benefactor if this delegator is already set (266ms)
✓ Should not set delegator for benefactor if benefactor is not a sealer (89ms)
✓ Should not set delegator for benefactor is msg.sender is not authorized (98ms)
✓ Should set coinbase for benefactor (309ms)
✓ Should not set coinbase for benefactor if msg.sender is not authorized

57 passing (30s)

FILE	% STMTS	% BRANCH	% FUNCS
ShyftStaking.sol	100	90.88	100
ShyftBlockReward.s ol	95.83	96.15	96.43
All files	**97.91**	**93.5**	**98.2**

We are grateful to have been given the opportunity to work with Shyft.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Zokyo's Security Team recommends that Shyft put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.