verichains

SECURITY AUDIT OF

SYNSTATION STAKING SMART CONTRACTS

SynStation STAKING

Public Report

Oct 21, 2024

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.
ERC721	The ERC-721 introduces a standard for NFT, in other words, this type of Token is unique and can have different value than another Token from the same Smart Contract, maybe due to its age, rarity or even something else like its visual.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Oct 21, 2024. We would like to thank the SynStation Organization for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the SynStation Staking smart contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified a small issue in the contract code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About SynStation Staking smart contracts

SynStation is the first and trusted DeFi hub on *Soneium*, aims to be the best community owned DeFi protocol.

1.2. Audit Scope

This audit focused on identifying security flaws in code and the design of the SynStation Staking smart contracts.

It was conducted on commit `b82e2792c4e61d883f8d466ef4e13d0102620f9f` from git repository *https://github.com/WASD3Rplay/synstation-staking-contract/tree/main/contracts/v1*

SHA256 Sum	File
`607d0bd119b6871ebed51c697f2beb197e3ecd2b98484ae66670f77a7c8a39d9`	`SynstationDepositWrapper.sol`
`d3b5357c2dfb5546da9ca50922d2d63a5a2c2bb2bc24277a74b062535fa1b0a1`	`WstETHWrapper.sol`
`a447e9db2b4e1d76c5336bb3dfa86401c4fe062c24436aa9a32d43fde1e82a7c`	`SynstationPreStaking.sol`

1.3. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit

- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

SynStation Organization acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. SynStation Organization understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, SynStation Organization agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the SynStation Organization will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the SynStation Organization, the final report will be considered fully accepted by the SynStation Organization without the signature.

2. AUDIT RESULT

2.1. Overview

The SynStation Staking smart contracts are written in Solidity language and utilize `OpenZeppelin`'s contract libraries, emphasizing modularity and security.

2.1.1. Deposit contract

`WstETHWrapper` is a wrapper contract which provides users ability to wrap `ETH` and `stETH` to `wstETH` and deposit it to the staking contract.

- `function deposit(address _token ,uint256 _amount)`: Users can call this function and pass `_token` to be either `ETH` or `stETH` and `_amount` to deposit. They need to send `ETH` value along to be equal to the `_amount` if they want to deposit `ETH`. The function will wrap the `_token` to `wstETH` and deposit it to the staking contract.

2.1.2. Staking contract

`SynstationPreStaking` is the staking contract in the SynStation Staking smart contracts which allows users to deposit and withdraw tokens. It extends `AccessControlUpgradeable` contract which provides role-based access control mechanism and also the ability to be upgraded.

There are 3 roles in the contract: `DEFAULT_ADMIN_ROLE`, `ADMIN_ROLE`, and `LISTING_ROLE` which are all set to the same address when init contract. All the roles can be easily customized by the `DEFAULT_ADMIN_ROLE` at any time.

2.1.2.1. Listing role

Addresses with `LISTING_ROLE` can add pools with supported token and update pool's deposit cap. After pool is added, users can now stake their supported token to the pool.

- `function add(IERC20 _want, uint256 _depositCap)`: This function is used to add a pool with `_want` token and `_depositCap` to the staking contract. The `_want` token is the token that users can stake to the pool and `_depositCap` is the maximum amount of `_want` token that can be deposited to the pool. The function will add the pool to the `poolInfo` array and emit an `AddNewPool` event.
- `function setDepositCap(uint256 _pid, uint256 _depositCap)`: This function is used to update the deposit cap of the pool with `_pid` to `_depositCap`. The function will update the `depositCap` of the pool in the `poolInfo` array and emit an `SetDepositCap` event.

2.1.2.2. Admin role

Addresses with `ADMIN_ROLE` can pause/unpause pools. When a pool is paused, users can no longer deposit tokens to the pool but can still withdraw their staked. Admin role can also

withdraw any `ERC20` tokens and native tokens from the contract, this is to rescue the fund in emergency case.

- `function setPause(uint256 _pid, bool _paused)`: This function is used to pause/unpause the pool with `_pid`. When `_paused` is `true`, users can no longer deposit tokens to the pool but can still withdraw their staked. The function will update the `paused` status of the pool in the `poolInfo` array and emit an `SetPause` event.
- `function rescueFunds(IERC20 token, uint256 amount)`: This function is used to withdraw any `ERC20` tokens from the contract. The function will transfer the `amount` of `token` to the caller.
- `rescueETH(uint256 amount)`: This function is used to withdraw native tokens from the contract. The function will transfer the `amount` of native tokens to the caller.

2.1.2.3. Users

Users can deposit and withdraw tokens to/from the pool but the tokens must not exceed the deposit cap of the pool.

- `function deposit(uint256 pid, uint256 amount, address to)`: This function is used to deposit `amount` of tokens to the pool with `_pid`. The function will transfer the `amount` of tokens from the caller to the contract and update the user's staked amount and last deposit time in the `userInfo` mapping. The function will also update the pool's total staked amount and emit a `Deposit` event.
- `function withdraw(uint256 pid, uint256 desiredAmount, address to)`: This function is used to withdraw `desiredAmount` of tokens from the pool with `_pid`. If the `desiredAmount` is more than user's staked amount, it will be set to the staked amount to avoid withdrawing more than staked. The function will then transfer the tokens from the contract to the caller and update the user's staked amount in the `userInfo` mapping. The function will also update the pool's total staked amount and emit a `Withdraw` event.

2.2. Findings

During the audit process, the audit team had identified a small issue in the contract code.

SynStation Organization fixed the code, according to Verichains's private report, in commit `9c5e72f84f2fd5fc1fb07f89d1b138a455e94171`.

2.2.1. Should require token address to be supported tokens LOW

Affected files:

- SynstationDepositWrapper.sol
- WstETHWrapper.sol

The deposit contract only supports to deposit `wstETH` to staking contract so it allows users to deposit `ETH` and `stETH` by wrapping them to `wstETH`. However, the deposit function does not strictly check the `_token` address so `ETH` could be stuck in some case.

For example, a user deposit `ETH` by calling `deposit` function with `ETH` value but mistaken pass `_token` as `wstETH` address, the `deposit` function will not revert and the user will lose the `ETH` without deposit anything.

Another example is when a user want to deposit `stETH` but mistaken pass `ETH` value with `deposit` function, the `ETH` value will be stuck in the contract.

```
function deposit(
    address _token,
    uint256 _amount
) external payable virtual returns (uint256) {
    _handleWrapProcess(_token, _amount);

    return _handleDeposit();
}

function _handleWrapProcess(
    address _token,
    uint256 _amount
) internal override {
    if (_token == address(0)) {
        require(msg.value == _amount, "!value-mismatch");

        _ethToStEth(_amount);
        _stEthToWstEth(_amount);
    }

    if (_token == address(STETH)) {
        IERC20(_token).safeTransferFrom(msg.sender, address(this), _amount);
        _stEthToWstEth(_amount);
    }
}
```

RECOMMENDATION

- Require `msg.value == 0` when `_token` is not `address(0)`.
- Require `_token` to be either `address(0)` or `address(STETH)`.

UPDATES

- **Oct 21, 2024**: The issue has been acknowledged and fixed by SynStation Organization team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Oct 17, 2024*	Private Report	Verichains Lab
1.1	*Oct 21, 2024*	Public Report	Verichains Lab

Table 2. Report versions history