VESYNC

SMART CONTRACT AUDIT

zokyo

June 23th 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security 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 VeSync smart contracts evaluated by the Zokyo Security team.

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

Contract Status

There were 0 critical issues found during the audit. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contracts but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the VeSync team put in place a bug bounty program to encourage further active analysis of the smart contracts.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the VeSync repository:
https://github.com/veSync/contracts

Initial commit - 4dd65dadebf560e668101594a312a5474bb3d18a

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- ExternalBribe.sol
- Minter.sol
- Voter.sol
- VotingEscrow.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most resent vulnerabilities;
- Meets best practices in code readability, etc.

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

01	Due diligence in assessing the overall code quality of the codebase.
02	Cross-comparison with other, similar smart contracts by industry leaders.
03	Thorough manual review of the codebase line by line.

Executive Summary

The codebase was subjected to a security audit by the Zokyo team. No critical issues were discovered during the audit. However, there were identified issues with varying levels of severity, including medium, low, and informational issues. The client team has acknowledged all the findings, which are thoroughly explained in the "Complete Analysis" section.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the VeSync team and the VeSync team is aware of it, but they have chosen to not solved it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. 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

FINDINGS SUMMARY

#	Title	Risk	Status
1	Centralization risk	Medium	Acknowledged
2	Missing VERSION in DOMAIN_TYPEHASH	Low	Acknowledged
3	Lack of events in admin functions	Low	Acknowledged
4	Misuse of ecrecover	Low	Acknowledged
5	Unsafe casting	Low	Acknowledged
6	Using assert excessively	Low	Acknowledged
7	Use _safeMint(...) instead of _mint(...)	Low	Acknowledged
8	Use 2-step ownership transfers	Low	Acknowledged
9	Transfer is unnecessarily triggered	Low	Acknowledged
10	Missing checks for address(0) in setter methods	Low	Acknowledged
11	Missing length checks	Low	Acknowledged
12	Mismatching spec of NFT	Informational	Acknowledged
13	Unreachable code in _reset()	Informational	Acknowledged
14	Divide by zero	Informational	Acknowledged
15	Array indexes input of functions are not validated	Informational	Acknowledged
16	Use solidity time units	Informational	Acknowledged
17	Redundant validation in emission override check function	Informational	Acknowledged
18	Unused imports	Informational	Acknowledged

MEDIUM | ACKNOWLEDGED

Centralization risk

Admin enjoys much authority in contracts. The general theme is that admin has power to call several state changing functions also affecting the tokenomics of the project . Some functions can be highly severe to be left out controlled by one wallet .

Recommendation:

Implement a mutlisig contract as governor or utilize readily implemented mutlisig product.

Comment:

As for this risk in the short term, VeSync team will deploy timelock to mitigate this issue soon, and in the long term. They will switch to multi-signature when the zkSync ecosystem becomes more mature. They plan to deploy a timelock mechanism soon to mitigate the risks involved soon. Additionally, VeSync team aims to transition to more advanced security measures, such as multi-signature, after ZK officially supports Gnosis. This issue will be resolved soon.

Missing VERSION in DOMAIN_TYPEHASH

In contract VotingEscrow, the **DOMAIN_TYPEHASH** is build as following

```
bytes32 public constant DOMAIN_TYPEHASH =
        keccak256(
            "EIP712Domain(string name,uint256 chainId,address
verifyingContract)"
        );
```

While the EIP712 states it has one more parameter `string version` which is missing in this contract.

In the same contract, the delegateBySig method id uses `version` parameter to create the domain separator. This ambiguous behavior can result in an `Invalid signature` revert for any dapp/backend trying to create the legit signature.

Recommendation:
Add `string version` in the DOMAIN_TYPEHASH

```
ytes32 public constant DOMAIN_TYPEHASH
keccak256("EIP712Domain(string name,string version,uint256
chainId,address verifyingContract)");
```

LOW | ACKNOWLEDGED

Lack of events in admin functions

Voter.sol/VotingEscrow.sol - Functions are supposed to be called by privileged wallets to change important parameters that run the protocol. It is advised to have such functions emit events describing the changes that took place. Among these functions:

In `Voter`

```
function setGovernor(address _governor) public
function setEmergencyCouncil(address _council) public
function whitelist(address _token) public
```

In `VotingEscrow`

```
function setTeam(address _team) external
function setArtProxy(address _proxy) external
function unfreeze(uint256 _tokenId) external
function batchUnfreeze(uint256[] memory _tokenIds) external
function setVoter(address _voter) external
function voting(uint256 _tokenId) external
function abstain(uint256 _tokenId) external
function attach(uint256 _tokenId) external
function detach(uint256 _tokenId) external
```

In `Minter`

```
function setEarlyGrowthParams(uint[] memory params) external
function setOverrideGrowthParam(uint param) external
function setTeam(address _team) external
function acceptTeam() external
function setTeamRate(address _teamRate) external
function setWeeklyOverride(address _weeklyOverride) external
```

Recommendation:

Emit events that describe the change that took place.

LOW | ACKNOWLEDGED

Misuse of ecrecover

VotingEscrow.sol - Vulnerability exists in the signature logic of delegateBySig() function. According to the following:

```
address signatory = ecrecover(digest, v, r, s); require(signatory != address(0), "VotingEscrow::delegateBySig: invalid signature");
```

The contract retrieves the address of the signer from the signature via ecrecover() then the retrieved address is being ensured not to be zero address. This is insufficient because an arbitrary signature can also return an arbitrary address.

It is very unlikely for the attacker to exploit this in order to target one specific address by brute force (i.e. probability near zero). But attacker using brute force can be attempting to find just one address among a pool of addresses (i.e. owners of the NFTs). Once the attacker finds a signature that produces an address of any owner (along with the suitable nonce), the impact is big as they move all the delegates to a wallet that they choose.

The severity of the issue is not decided to be high because the strict equality required of nonces[signatory] limit the power of the attacker to carry out such an attempt successfully.

Recommendation:

The address of the owner (i.e. signer) should be involved in the `digest`. Refer to section `How to implement EIP-712` in this posting.

Unsafe casting

VotingEscrow.sol - Casting values without validating the numbers to be within the valid range is commonly used in this contract and it is a discouraged pattern.

In `_checkpoint()`

```
int128(int256(new_locked.end - block.timestamp))

int128(int256(t_i - last_checkpoint))
```

In `_deposit_for()`

```
_locked.amount += int128(int256(_value));
```

In `_balanceOfAtNFT()`

```
997     upoint.bias -= upoint.slope * int128(int256(block_time -
upoint.ts));
```

In `_supply_at()`

```
1047    last_point.bias -= last_point.slope * int128(int256(t_i -
last_point.ts));
```

In `_balanceOfNft()`

```
934     last_point.bias -= last_point.slope * int128(int256(_t) -
int256(last_point.ts));
935     if (last_point.bias < 0) {
936         last_point.bias = 0;
937     }
938     return uint(int256(last_point.bias));
```

In `withdraw()`

```
uint value = uint(int256(_locked.amount));
```

Recommendation:

Implement a `SafeCast` functionality similar to this library: SafeCast

LOW | ACKNOWLEDGED

Using assert excessively

VotingEscrow.sol - `assert` is being utilized unnecessarily in this contract despite that it is meant to be used for a different purpose. The assert statement serves the purpose of testing for internal errors and validating invariants

The require function in Solidity should be utilized to verify the fulfillment of valid conditions, such as inputs or contract state variables, and to validate return values obtained from interactions with external contracts.

There are 13 occurences in which assert is being used in this contract in which almost all are considered a misuse. For the cases in which the developer sees as an invariant and should not be reached, `assert` can be used, otherwise it is advised to use `require` instead.

Recommendation:
Replace `assert` by `require` with an appropriate concise error message.

LOW | ACKNOWLEDGED

Use _safeMint(...) instead of _mint(...)

In contract VotingEscrow.sol, _mint(...) is used to mint NFTs to user addresses whereas it is advised to use _safeMint to ensure that the address is either an EOA or a contract that implements ERC721Receiver.

Recommendation:
Use OpenZeppelin safeMint implementation for the ERC721 tokens.

LOW | ACKNOWLEDGED

Use 2-step ownership transfers

Contract Voting.sol and Contract VotingEscrow.sol have several setters which transfers important ownerships in one step.

function setGovernor(address _governor);
function setEmergencyCouncil(address _council);
function setTeam(address _team) external;

Recommendation:
It is advised to use 2-step ownership where the current owner nominates the next owner and the next owner has to accept it.

LOW | ACKNOWLEDGED

Transfer is unnecessarily triggered

VotingEscrow.sol - `_transferFrom`
ExternalBribe.sol - _safeTransfer and _safeTransferFrom
Following functions does not validate that to is not the same address as from which result in triggering an unnecessary transfer process.

```
function _transferFrom(address _from, address _to, uint256 _tokenId,
address _sender) internal
function transferFrom(address _from, address _to, uint256 _tokenId)
external
function safeTransferFrom(address _from, address _to, uint256 _tokenId,
bytes memory _data) public
function _safeTransfer(address token, address to, uint256 value) internal
function _safeTransferFrom(address token, address from, address to,
uint256 value) internal
```

Recommendation:
Need to make this validation in any of the transfer methods whether the external or internal ones.

Missing checks for address(0) in setter methods

In contract Voting.sol, the following methods are missing zero-address checks.

- function initialize(address[] memory _tokens, address _minter) external;

This method does not check if _minter is a zero-address or not.

- function setGovernor(address _governor);

For setGoverner(), there is no check for zero-address for _governer parameter and the method checks that only the governor should be able to set a new governor. So, suppose accidentally governor is set as address(0). In that case, it will be impossible to set the governor again; all the methods callable only by the governor can not be called anymore.

- function setEmergencyCouncil(address _council);

The above case of the setGovernor method applies to the setEmergencyCouncil method as well.

In contract VotingEscrow.sol, the following methods are missing zero-address checks:

- function setTeam(address _team) external;

For setTeam(), there is no check for zero-address for _teamparameter and the method checks that only the team should be able to set a new team. So, suppose accidentally team is set as address(0). In that case, it will be impossible to set the team again; all the methods callable only by the team can not be called anymore.

- function setArtProxy(address _proxy) external;

This method does not check if _proxy is zero-address or not.

In contract Voting.sol, the following methods are missing zero-address checks:

- function setTeam(address _team) external;

There is no check for zero-address for _team parameter.

Recommendation:

Add checks to revert in case zero-address parameters are passed in the above-mentioned scenarios.

Missing length checks

In contract Voter.sol, the method `claimBribes(address[] memory _bribes, address[][] memory _tokens,uint _tokenId) ` and in contract Minter.sol, the method initializeToken(address[] memory targets, uint[] memory accounts) have two array parameters that are accessed in the for loop and it is missing a check to see if both arrays are of same lengths.

Recommendation:
Add a validation check at the beginning of the function to ensure that the arrays have the same length.

Mismatching spec of NFT

VotingEscrow.sol - A natspec comment mismatches what the function actually does:

```
/// @dev Returns the number of NFTs owned by `_owner`.
///      Throws if `_owner` is the zero address. NFTs assigned to the
zero address are considered invalid.
/// @param _owner Address for whom to query the balance.
function _balance(address _owner) internal view returns (uint) {
    return ownerToNFTokenCount[_owner];
}
```

The comment which describes a spec is not actually implemented in the function.

```
/// Throws if `_owner` is the zero address. NFTs assigned to the zero
address are considered invalid.
```

Recommendation:

Implement the spec or if the spec is not needed, remove the comment.

Unreachable code in _reset()

Voter.sol - In `_reset()` function we have:

```
uint256 _votes = votes[_tokenId][_pool];
if (_votes != 0) {
    _updateFor(gauges[_pool]);
    weights[_pool] -= _votes;
    votes[_tokenId][_pool] -= _votes;
    if (_votes > 0) {
        IBribe(internal_bribes[gauges[_pool]])._withdraw(uint256(_votes),
_tokenId);
        IBribe(external_bribes[gauges[_pool]])._withdraw(uint256(_votes),
_tokenId);
        _totalWeight += _votes;
    } else {
        _totalWeight -= _votes;
    }
...
```

Where `votes[_tokenId][_pool]` and accordingly `_votes` are of type `uint256`. Since the value of `_votes >= 0` , we end up having `_totalWeight -= _votes`; serving no purpose. The only case in which the code is reachable (i.e. at `_votes == 0`) the subtraction has no effect on `_totalWeight`.

Recommendation:

Developer might need to fix this in some way in case the subtraction of `_votes` is supposed to has an effect. In the case in which there is no need to subtract value of `_votes` then it is better to remove the unneeded line of code.

Divide by zero

Voter.sol - Variables being the divisor of division are not ensured to be non-zero.
In `notifyRewardAmount()` function
`uint256 _ratio = amount * 1e18 / totalWeight;`

In `_vote()` function
`uint256 _poolWeight = _weights[i] * _weight / _totalVoteWeight;`

`totalWeight` represents a state of contract while `_totalVoteWeight` is derived from the arguments of the function and they can be holding zero value. Having the functions revert for invalid inputs or state as a shown reason makes it less time consuming to debug when the transaction is reverted.

Recommendation:
Require these values to be non-zero so that it reverts showing a clearer message about the error and devs know what kind of mitigation to work on.

Array indexes input of functions are not validated

Voter.sol - Indexes by array are not ensured to be within the valid range. Arguments of function `updateForRange(): start, end` are not assured to be within the valid range of `pools` array.

```
function updateForRange(uint start, uint end) public {
    for (uint i = start; i < end; i++) {
        _updateFor(gauges[pools[i]]);
    }
}

also,
function distribute(uint start, uint finish) public {
    for (uint x = start; x < finish; x++) {
        distribute(gauges[pools[x]]);
    }
}
```

Having the function revert for invalid inputs as a shown reason makes it less time consuming to debug when the transaction is reverted.

Recommendation:

Require that `end > start` and that `end <= pools.length` .

INFORMATIONAL | ACKNOWLEDGED

Use solidity time units

In contract VotingEscrow.sol, MAXTIME, and iMAXTIME can use time units for readability instead.

```
uint internal constant MAXTIME = 365 * 86400;
  int128 internal constant iMAXTIME = 365 * 86400;
```

Recommendation:
Use time units for MAXTIME, and iMAXTIME values

INFORMATIONAL | ACKNOWLEDGED

Redundant validation in emission override check function

The Minter.emission_override_enabled function is used to check if the emission override feature is enabled. It verifies three conditions: the current timestamp is less than _launchTime + OVERRIDE_ALLOWED_DURATION, weeklyOverride is greater than 0, and weeklyOverride is less than weekly. However, the third condition is redundant as the setWeeklyOverride function ensures that the new setted weeklyOverride value is less than the current weekly value.

Recommendation:
Remove redundant check.

Unused imports

The code base contains unused imports not referenced in the contracts that should be removed to improve code cleanliness and eliminate unnecessary dependencies.

Voter.sol:

```
import 'contracts/libraries/Math.sol';
```

ExternalBribe.sol:

```
mport 'contracts/interfaces/IGauge.sol';
```

Recommendation:
Remove the mentioned imports.

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

We are grateful for the opportunity to work with the VeSync team.

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

Zokyo Security recommends the VeSync team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo