



Security Assessment

VybeSale SaleBase Options

Verified on 12/22/2025

SUMMARY

Project

VybeSale SaleBase Options

CHAIN

Binance Smart Chain

METHODOLOGY

Manual & Automatic Analysis

FILES

Single

DELIVERY

12/22/2025

TYPE

Standard Audit



4

0

0

1

1

2

4

Total Findings

Critical

Major

Medium

Minor

Informational

Resolved

■ 0 Critical

An exposure that can affect the contract functions in several events that can risk and disrupt the contract

■ 0 Major

An opening & exposure to manipulate the contract in an unwanted manner

■ 1 Medium

An opening that could affect the outcome in executing the contract in a specific situation

■ 1 Minor

An opening but doesn't have an impact on the functionality of the contract

■ 2 Informational

An opening that consists information but will not risk or affect the contract

■ 4 Resolved

ContractWolf's findings has been acknowledged & resolved by the project

STATUS
✓ **AUDIT PASSED**

TABLE OF CONTENTS | VybeSale SaleBase Options

| **Summary**

Project Summary
Findings Summary
Disclaimer
Scope of Work
Auditing Approach

| **Project Information**

Token/Project Details
Inheritance Graph
Call Graph

| **Findings**

Issues
SWC Attacks
CW Assessment
Fixes & Recommendation
Audit Comments

DISCLAIMER | VybeSale SaleBase Options

ContractWolf audits and reports should not be considered as a form of project's "Advertisement" and does not cover any interaction and assessment from "Project Contract" to "External Contracts" such as PancakeSwap, UniSwap, SushiSwap or similar.

ContractWolf does not provide any warranty on its released report and should not be used as a decision to invest into audited projects.

ContractWolf provides a transparent report to all its "Clients" and to its "Clients Participants" and will not claim any guarantee of bug-free code within its **SMART CONTRACT**.

ContractWolf's presence is to analyze, audit and assess the Client's Smart Contract to find any underlying risk and to eliminate any logic and flow errors within its code.

Each company or project should be liable to its security flaws and functionalities.

SCOPE OF WORK | VybeSale SaleBase Options

VybeSale SaleBase Options team has agreed and provided us with the files that need to be tested (*Github, BSCscan, Etherscan, Local files etc*). The scope of audit is the main contract.

The goal of this engagement is to identify if there is a possibility of security flaws in the implementation of smart contract and its systems.

ContractWolf will be focusing on contract issues and functionalities along with the project claims from smart contract to their website, whitepaper, repository which has been provided by **VybeSale SaleBase Options**.

AUDITING APPROACH | VybeSale SaleBase Options

Every line of code along with its functionalities will undergo manual review to check for security issues, quality of logic and contract scope of inheritance. The manual review will be done by our team that will document any issues that they discovered.

METHODOLOGY

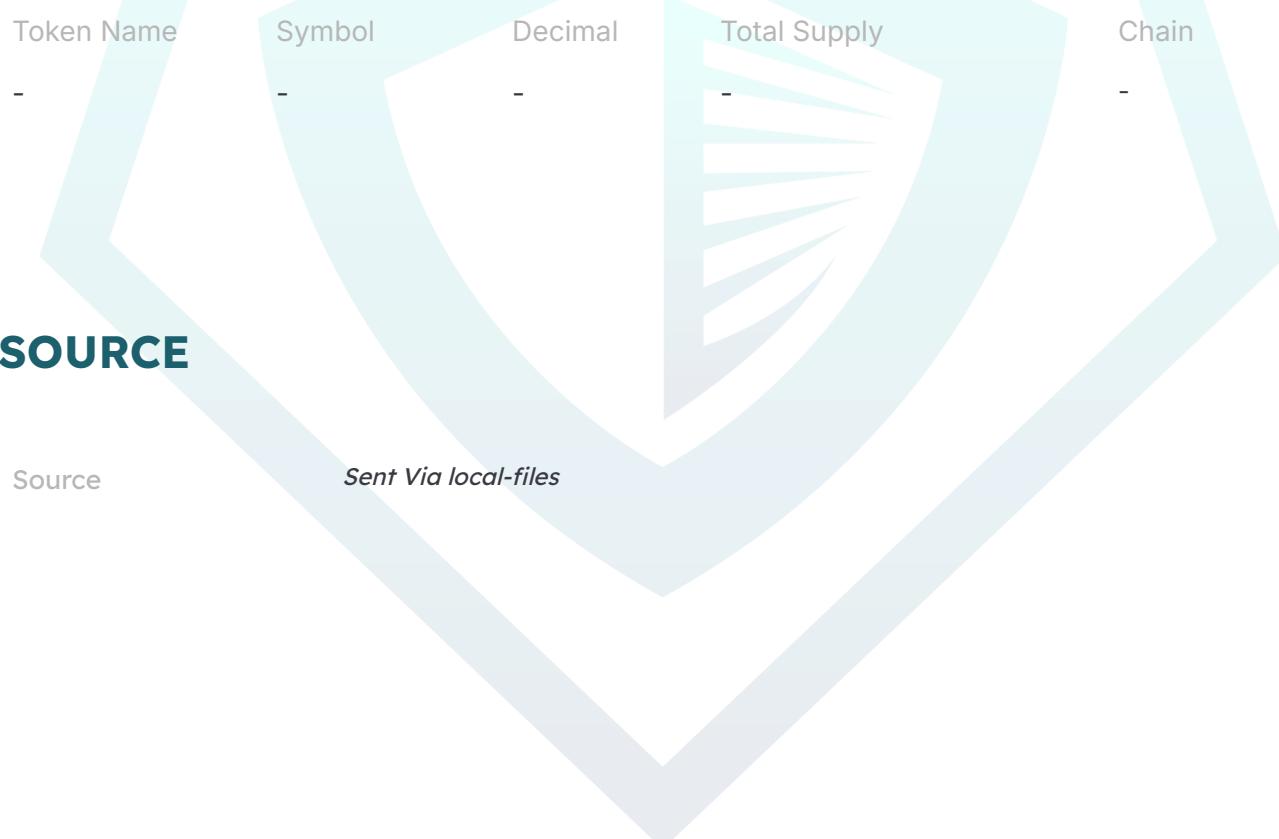
The auditing process follows a routine series of steps :

1. Code review that includes the following :
 - Review of the specifications, sources and instructions provided to ContractWolf to make sure we understand the size, scope and functionality of the smart contract.
 - Manual review of code. Our team will have a process of reading the code line-by-line with the intention of identifying potential vulnerabilities, underlying and hidden security flaws.
2. Testing and automated analysis that includes :
 - Testing the smart contract function with common test cases and scenarios to ensure that it returns the expected results.
3. Best practices and ethical review. The team will review the contract with the aim to improve efficiency, effectiveness, clarifications, maintainability, security and control within the smart contract.
4. Recommendations to help the project take steps to eliminate or minimize threats and secure the smart contract.

TOKEN DETAILS | VybeSale SaleBase Options



A multi-chain launchpad. Run presales, fair launches, or let us build your custom token. Comes with built-in LP locker, token locker, airdrop tool, and optional professional auditing. Launch safer, faster, across every major chain.



FINDINGS | VybeSale SaleBase Options



This report has been prepared to state the issues and vulnerabilities for VybeSale SaleBase Options through this audit. The goal of this report findings is to identify specifically and fix any underlying issues and errors

ID	Title	File & Line #	Severity	Status
N/A	-getCurrentStageId() Logic	Bonus.sol, L: 22	Medium	● Resolved
SWC-103	Floating Pragma is set	SaleBase.sol, Bonus.sol, Affiliate.sol, BuyBack.sol, Vesting.sol	Informational	● Resolved
SWC-131	Presence of Unused Variables	SaleBase.sol, L: 40	Informational	● Resolved
N/A	Unbounded Array	Bonus.sol, L: 41	Minor	● Resolved

SWC ATTACKS | VybeSale SaleBase Options

Smart Contract Weakness Classification and Test Cases

ID	Description	Status
SWC-100	Function Default Visibility	Passed
SWC-101	Integer Overflow and Underflow	Passed
SWC-102	Outdated Compiler Version	Passed
SWC-103	Floating Pragma	Passed
SWC-104	Unchecked Call Return Value	Passed
SWC-105	Unprotected Ether Withdrawal	Passed
SWC-106	Unprotected SELF DESTRUCT Instruction	Passed
SWC-107	Reentrancy	Passed
SWC-108	State Variable Default Visibility	Passed
SWC-109	Uninitialized Storage Pointer	Passed
SWC-110	Assert Violation	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed
SWC-112	Delegatecall to Untrusted Callee	Passed
SWC-113	DoS with Failed Call	Passed
SWC-114	Transaction Order Dependence	Passed
SWC-115	Authorization through tx.origin	Passed
SWC-116	Block values as a proxy for time	Passed
SWC-117	Signature Malleability	Passed
SWC-118	Incorrect Constructor Name	Passed
SWC-119	Shadowing State Variables	Passed
SWC-120	Weak Sources of Randomness from Chain Attributes	Passed
SWC-121	Missing Protection against Signature Replay Attacks	Passed
SWC-122	Lack of Proper Signature Verification	Passed

ID	Description	Status
SWC-123	Requirement Violation	● Passed
SWC-124	Write to Arbitrary Storage Location	● Passed
SWC-125	Incorrect Inheritance Order	● Passed
SWC-126	Insufficient Gas Griefing	● Passed
SWC-127	Arbitrary Jump with Function Type Variable	● Passed
SWC-128	DoS With Block Gas Limit	● Passed
SWC-129	Typographical Error	● Passed
SWC-130	Right-To-Left-Override control character(U+202E)	● Passed
SWC-131	Presence of unused variables	● Passed
SWC-132	Unexpected Ether balance	● Passed
SWC-133	Hash Collisions With Multiple Variable Arguments	● Passed
SWC-134	Message call with hardcoded gas amount	● Passed
SWC-135	Code With No Effects	● Passed
SWC-136	Unencrypted Private Data On-Chain	● Passed

CW ASSESSMENT

VybeSale SaleBase Options

ContractWolf Vulnerability and Security Tests

ID	Name	Description	Status
CW-001	Multiple Version	Presence of multiple compiler version across all contracts	✓
CW-002	Incorrect Access Control	Additional checks for critical logic and flow	✓
CW-003	Payable Contract	A function to withdraw ether should exist otherwise the ether will be trapped	✓
CW-004	Custom Modifier	major recheck for custom modifier logic	✓
CW-005	Divide Before Multiply	Performing multiplication before division is generally better to avoid loss of precision	✓
CW-006	Multiple Calls	Functions with multiple internal calls	✓
CW-007	Deprecated Keywords	Use of deprecated functions/operators such as block.blockhash() for blockhash(), msg.gas for gasleft(), throw for revert(), sha3() for keccak256(), callcode() for delegatecall(), suicide() for selfdestruct(), constant for view or var for actual type name should be avoided to prevent unintended errors with newer compiler versions	✓
CW-008	Unused Contract	Presence of an unused, unimported or uncalled contract	✓
CW-009	Assembly Usage	Use of EVM assembly is error-prone and should be avoided or double-checked for correctness	✓
CW-010	Similar Variable Names	Variables with similar names could be confused for each other and therefore should be avoided	✓
CW-011	Commented Code	Removal of commented/unused code lines	✓
CW-012	SafeMath Override	SafeMath is no longer needed starting with Solidity v0.8+. The compiler now has built-in overflow checking.	✓

FIXES & RECOMMENDATION

SWC-103 | A FloatingPragma is Set

Code

```
pragma solidity ^0.8.17;
```

The compiler version should be a fixed one to avoid undiscovered compiler bugs. Fixed version sample below

```
pragma solidity 0.8.17;
```

SWC-131 | Presence of unused variables

Code

```
address immutable dead = 0x00000000000000000000000000000000dEaD;
```

Unused variables are allowed in Solidity and they do not pose a direct security issue. It is best practice to remove them if unused.



_getCurrentStageId() Logic

The function has three issues:

- (1) potential underflow if sale hasn't started
- (2) incorrect cumulative duration logic (checks each stage independently instead of cumulative), and;
- (3) returns a timestamp instead of stage ID when no stage matches.

```

function _getCurrentStageId(
    uint256 _saleStartTime
) internal view returns (uint256) {
    uint256 currentTime = block.timestamp;
    uint256 elapsedTime = currentTime - _saleStartTime; // Underflow if sale not
started

    for (uint256 i = 0; i < stages.length; i++) {
        if (elapsedTime <= stages[i].duration) { // Wrong logic - should be
cumulative
            return i;
        }
    }

    return currentTime; // Returns timestamp instead of stage ID!
}

```

Recommendation

Add a check at the start to return stage 0 if the sale hasn't started, preventing underflow. Track cumulative duration across stages instead of checking each stage independently. Initialize a cumulative duration variable to 0, then in the loop add each stage's duration to the cumulative total and check if elapsed time is within that cumulative range. If no stage matches, return stages.length to indicate all stages have passed, rather than returning a timestamp.

Unbounded Array Growth

There's no maximum limit on the number of bonus stages that can be added, which can lead to gas DoS attacks and increased gas costs in `_getCurrentStageId()`.

```
function addBonusStage(
    Sales.BonusParameters memory params
) external onlyController {
    stages.push(
        BonusStage({
            duration: params.duration,
            minContribution: params.minContribution,
            bonusPercentage: params.bonusPercentage,
            bonusSpots: params.bonusSpots,
            spotsTaken: 0
        })
    );
}
```

Recommendation

Add a constant `MAX_STAGES` set to a reasonable limit (e.g., 50) at the contract level. At the beginning of `addBonusStage()`, add a require statement that checks `stages.length < MAX_STAGES` before allowing a new stage to be added. This prevents unbounded array growth and protects against gas DoS attacks. The error message should clearly indicate "Max stages reached" to inform the caller why the operation failed.



CONTRACTWOLF

Blockchain Security - Smart Contract Audits