SMART CONTRACT AUDIT

zokyo

August 30th 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

TECHNICAL SUMMARY

This document outlines the overall security of the Vaultka smart contracts evaluated by the Zokyo Security team.

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

Contract Status

There was one critical issue 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 Vaultka 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 Vaultka repository:
https://github.com/Vaultka-Project/sakev2-contract

Last changes -8bf16c604d99a7cedc698b6194ac36bfe4e073c7

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

- SakeVaultV2.sol
- Water.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 recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Vaultka smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were 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 audit revealed one issue of critical severity and another with high severity. Furthermore, there were findings of medium, low severity, and informational issues. These are comprehensively described 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 Vaultka team and the Vaultka team is aware of it, but they have chosen to not solve 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	An Attacker Can Make Victim's Deposit Be Locked Forever	Critical	Resolved
2	Water deposit is exposed to inflation attack	High	Resolved
3	No slippage protection	Medium	Resolved
4	Centralization Risk	Medium	Acknowledged
5	Users Might Receive Undesired Tokens In fullfilledRequestSwap	Medium	Resolved
6	Return Value Of repayDebt Not Checked	Low	Resolved
7	openPosition exposed to cross-function reentrance	Low	Resolved
8	Cooldown Period Is 48-72 hours In Docs But 24 hours In Code	Low	Resolved
9	Uninitialized reentrance guard	Low	Resolved
10	Users Will Be Forced To Liquidations Due To Missing Functionality To Update Position	Informational	Acknowledged
11	closePositionRequest value not updated on request fulfillment	Informational	Resolved
12	Events not emitted on calling critical functions	Informational	Resolved
13	Misleading revert message	Informational	Invalid
14	Comment mismatch	Informational	Resolved
15	Lower and upper bounds of the new withdrawal fees are not validated	Informational	Resolved

#	Title	Risk	Status
16	Misleading comments	Informational	Resolved
17	Typos	Informational	Acknowledged
18	Missing Documentation	Informational	Acknowledged
19	Floating pragma	Informational	Acknowledged
20	Structs Can Be Packed Together To Save A Lot Of Gas	Informational	Acknowledged
21	No information returned about the new positionId created	Informational	Acknowledged

CRITICAL-1 | RESOLVED

An Attacker Can Make Victim's Deposit Be Locked Forever

Inside the deposit() function in Water.sol let's consider the following scenario →

1.) Victim deposits x amount of assets into the Water contract, expecting good returns. He can withdraw at block.timestamp + lockTime

2.) It's almost time to withdraw for the victim, now an attacker calls deposit() with `assets` as 1 (1 USDC) and the receiver as a victim.

3.) The `userTimelock` for the victim gets reset again to block.timestamp + lockTime.

If this griefing keeps happening, the victim won't be able to withdraw ever (Due to check at L209)

Recommendation:
The functionality to deposit for others should be removed.

Comment: The new fix adds another critical bug where a user can deposit to another account and instantly remove the funds bypassing the lockup.

Comment: The client fixed the following issue in commit: 7d32cf53cebcb6c81bc4755bbfd6befe4ac8522b. However, the paremter `address _receiver` should be removed as it has no functionality.

HIGH-1 | RESOLVED

Water deposit is exposed to inflation attack

Water.sol - The ERC4626 version used is exposed to inflation attack. That leads to having function `deposit()` in the water contract to be exploited by attackers to achieve illegitimate gains. The description of the attack scenario as well as the mitigation is described: https://docs.openzeppelin.com/contracts/4.x/erc4626.

Recommendation:
Follow the mitigation suggested in the link.

MEDIUM-1 | RESOLVED

No slippage protection

SakeVaultV2.sol - In function `_swap()` which is invoked by external functions: `openPosition()` and `fulfilledRequestSwap()`. As `_swap()` interacts with an external Decentralized Exchange (DEX), it can be exposed to price slippage on execution. The functions mentioned do not consider requiring the output amount to be greater than the desired minimum amount.

For instance, `velaShares` is the output amount received of vlp after staking USDC that resulted from `_swap`. `velaShares` can end up being less than the desirable amount that the user had in mind at the beginning of the execution of `openPosition()`. The same should be considered in `fulfilledRequestSwap()`.

Recommendation:
Add an argument in `openPosition()` and `fulfilledRequestSwap()` that represents the desirable output amount to be set by the user.

Fix: Client acknowledges that slippage amount of swap is to be handled via Kyberswap by passing information about it through the calldata.

Centralization Risk

The whole system grants too much power to the owner, for example - inside the SakeVault.sol the owner can call `withdrawAllESVELA` to withdraw all the esVELA to the feeReceiver which can be set to the owner himself using the changeProtocolFee function. The owner has the privilege to all the critical functionalities and can be manipulated in a way that harms the protocol (Another example can be setting MAX_LEVERAGE and MIN_LEVERAGE to arbitrary amounts, adding weird - ERC20 tokens into the whitelist of tokens which can have unintended behavior in the ERC4626 vault if a token is said a fee on transfer).

Recommendation:
Ensure the owner is a multisig and add suitable checks wherever necessary, even on functions with onlyOwner.

Users Might Receive Undesired Tokens In fullfilledRequestSwap

Say a user closed his position (closePosition() in SakeVaultV2.sol) and does not want to withdraw in USDC.
`closePositionAmount[_user][_positionID] = amountAfterFee;` and `closePositionRequest[_user][_positionID] = true;` would be set in this case.
Now imagine, me as an attacker call `fulfilledRequestSwap` at L744 but with `_outputAsset` as a token which is pretty volatile or in extreme conditions a meme coin, whereas the user wanted to swap to a more stable coin (all whitelisted).

The user gets his withdraw amount in an asset which might lose value quickly and is highly intended.

Recommendation:
Ensure this check on the function →

```
UserInfo storage _userInfo = userInfo[_user][_positionID];
require(msg.sender == _userInfo.user)
```

Comment: Client fixed the following issue in commit:
7d32cf53cebcb6c81bc4755bbfd6befe4ac8522b

LOW-1 | RESOLVED

Return Value Of repayDebt Not Checked

The function repayDebt inside Water.sol at L153 returns a bool indicating the success of the debt repayment operation.
This function is used inside SakeVaultV2.sol at L698 where the return value of this call is not checked.
Due to this, the call might fail silently and a return would happen at the next line.
This means that when closing a position (fully closing) if DTV goes beyond the liquidation threshold, a user won't actually be liquidated (if the repaydebt call fails) and the function executes normally.
Similar to L708 , where the call might fail and the debt won't be repaid to the Water contract, and the rest of the function executes normally.

Recommendation:
Add a check `require(success)` after a repayDebt call like `(bool success) = IWater(water).repayDebt()`

LOW-2 | RESOLVED

openPosition exposed to cross-function reentrance

SakeVaultV2.sol - Function `openPosition()` is not guarded by `nonReentrant` modifier which makes it susceptible to be reentered from other related functions like `closePosition()`. Functions like `openPosition(), closePosition(), liquidatePosition(), fulfulliedRequestSwap()` involve related asset transfers with updating related mappings of contract state. Hence, it is required that they are all guarded by a reentrance guard.

Recommendation:
Add `nonReentrant` modifier to `openPosition` to prevent the function from being reentered from the other mentioned functions.

LOW-3 | RESOLVED

Cooldown Period Is 48-72 hours In Docs But 24 hours In Code

Inside the docs the cooldown period is mentioned to be 48-72 hours, but in the code (Water.sol at L62) it is initialized to 24 hours.
The logic should match the intended usage, furthermore the code mention "The execution frequency will be improved in SAKE v2 to further shorten the cooldown to exactly 48 hours"

Recommendation:
The value for lock time should be initialized according to the docs/intention.

LOW-4 | RESOLVED

Uninitialized reentrance guard

SakeVaultV2.sol - In `function initialize()` the initializer of `ReentrancyGuardUpgradeable` (i.e. `__ReentrancyGuard_init()`) is not invoked. This is a bug because the default value of `_status` is `0` while the initializer sets it to `1`. The severity of the issue is limited though because once the modifier `nonReentrant` is used once the value of `_status` becomes the proper expected value.

Recommendation:
Invoke `__ReentrancyGuard_init()` in body of `initialize()`.

INFORMATIONAL-1 | ACKNOWLEDGED

Users Will Be Forced To Liquidations Due To Missing Functionality To Update Position

A user can create a new position using the function `openPosition()` inside SakeVaultV2.sol at L564.
As the DTV for the user's position comes close to the liquidation threshold, the user can do nothing to prevent the liquidation, since there is no mechanism to update a user's position or a repay mechanism. The only functionalities available are openPosition , closePosition, and liquidatePosition.

Recommendation:
Introduce a function where the user specifies his position (from the userInfo mapping) and the function updates the position with the given parameters.

Comment from a client - This feature may not be common when applying to tradfi, but we believe it does not pose any risk to users. Users can always close their position and prevent liquidation.

INFORMATIONAL-2 | RESOLVED

closePositionRequest value not updated on request fulfillment

SakeVaultV2.sol - In `fulfilledRequestSwap()`, we have the mapping `closePositionRequest` is not updated for the user. The mapping is expected to be true in order here:

```
require(closePositionRequest[_user][_positionID], "SAKE: Close only open position");
```

It is expected to be reassigned to `false` after that, which signifies that the user does not have a claimable amount of asset to be swapped and withdrawn. The severity of the issue is limited though by having `closePositionAmount[_user][_positionID]` assigned to zero.

Recommendation:
Update the value of `closePositionRequest[_user][_positionID]`.

Events not emitted on calling critical functions

```
function setAssetWhitelist(address _asset, bool _status) public
onlyOwner

function setFeeSplit(uint256 _feeSplit) public onlyOwner

function setUtilRate(uint256 _utilRate) public onlyOwner
```

SakeVaultV2.sol/Water.sol - No events emitted on changing significant parameters of the contracts by admin and assets transfers.

Recommendation:
Emit an event in the body of the admin functions.

Fix: Issue is partially fixed in commit 9b0f573 but functions that cause asset transfer like withdrawVesting, claim, withdrawAllVELA, withdrawAllESVELA are not addressed as well as setUtilRate which is in Water.sol .

Fix: As of commit a7edc1e, client added UtilRateChanged event to be emitted on calling setUtilRate. Other functions are unchanged though acknowledged by client.

INFORMATIONAL-4 | INVALID

Misleading revert message

SakeVaultV2.sol - Function `setFeeSplit()` shows a misleading revert message:
`require(_feeSplit <= 90, "Fee split cannot be more than 100%");`

Recommendation:
To be replaced by "Fee split cannot be more than 90%"

INFORMATIONAL-5 | RESOLVED

Comment mismatch

The comment at L388 should be at L385 and vice versa in SakeVaultV2.sol.

Recommendation:
Swap the comments.

Lower and upper bounds of the new withdrawal fees are not validated

SakeVaultV2.sol - In `function changeProtocolFee`, the argument `newWithdrawalFee` should be within certain determined bounds. In the body of the function the `newWithdrawalFee` is not being validated to be a proper input to be the new value of `feeConfiguration.withdrawalFee`.

```
function changeProtocolFee(address newFeeReceiver, uint256
newWithdrawalFee) external onlyOwner {
    feeConfiguration.withdrawalFee = newWithdrawalFee;
    feeConfiguration.feeReceiver = newFeeReceiver;
    emit ProtocolFeeChanged(newFeeReceiver, newWithdrawalFee);
}
```

Recommendation:
Add require statement to validate the `newWithdrawalFee` to be a proper value. Also, check `newFeeReceiver` to assert that it is a non-zero address.

Fix: In commit 9b0f573, function name has changed to setProtocolFee. Address input is validated to be non-zero address, but numerical uint values are not validated despite expected to be within a certain range.

Misleading comments

SakeVaultV2.sol - In function `getUpdatedDebtAndValue()`: comments in lines 385 and 388 do not reflect the conditions in the if branches that they belong to.

```
378     if (currentPosition > previousValueInUSDC) {
379         profitOrLoss = currentPosition - previousValueInUSDC;
380         // Call the `profitSplit` function to calculate the reward
split to water and the amount owed to water
381         (rewardSplitToWater, ) = profitSplit(profitOrLoss);
382         // The amount owed to water is the user's leverage amount
plus the reward split to water
383         owedToWater = leverage + rewardSplitToWater;
384     } else if (previousValueInUSDC > currentPosition) {
385         // If the current position and previous value are the same,
the amount owed to water is just the leverage amount
386         owedToWater = leverage;
387     } else {
388         // If the current position is less than the previous value,
the amount owed to water is the leverage amount
389         owedToWater = leverage;
390     }
```

Recommendation:
Swap the mentioned comments.

INFORMATIONAL-8 | ACKNOWLEDGED

Typos

The revert string for require at L109 inside SakeVaultV2 should be "Sake: positionID is not valid.

The revert string for require at L207 inside SakeVaultV2 should be "Fee split cannot be more than 90%" instead.

INFORMATIONAL-9 | ACKNOWLEDGED

Missing Documentation

The contract SakeVaultV2 lacks documentation for all the state variables plus functions like claim etc. Proper natspec documentation is necessary to increase readability and increase the overall quality of the codebase.

INFORMATIONAL-10 | ACKNOWLEDGED

Floating pragma

The solidity version is not locked throughout the codebase if a version of solidity has a bug that might propagate. Always ensure to test and write contracts at a fixed solidity version.

Recommendation:
Ensure the solidity version is fixed.

INFORMATIONAL-11 | ACKNOWLEDGED

Structs Can Be Packed Together To Save A Lot Of Gas

Inside the UserInfo struct in SakeVaultV2 , if we declare the `bool liquidated` just below the `address user` that will save
one storage slot saving ~20000 gas.

In FeeConfiguration struct `withdrawalFee` can be made into a uint128 ad that will result in one less storage slot.

Inside struct Datas , `profits` can be easily converted to uint128 as it is ample for accounting profits, and then `profits` and `inFull` can be clubbed together inside the same slot to save one storage slot.

INFORMATIONAL-12 | ACKNOWLEDGED

No information returned about the new positionId created

SakeVaultV2.sol - In `openPosition()`, a new position is created, but the value of the id about that position is missing.

Recommendation:
Return its value or emit it in an event.

	SakeVaultV2.sol Water.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 Vaultka 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 Vaultka team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo