HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Vaultka
Date: 10 November, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Vaultka
Approved By	Paul Fomichov | Lead Solidity SC Auditor at Hacken OÜ
Tags	ERC20 token; Staking;
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://www.vaultka.com/
Changelog	17.08.2023 - Initial Review 03.10.2023 - Second Review 10.11.2023 - Update on the Second Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Vaultka (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

System Overview

Vaultka is a staking protocol with the following contracts:

- *VKA Token: ERC20 token with 100,000,000 max supply*
- *esVKA Token: token used to represent escrowed VKA token that can be exchanged 1:1 for VKA tokens or 0.5:1 depending on the staking time decided by the user*
- *DualStaking: contract to stake VKA and esVKA tokens for rewards*
- *SingleStaking: contract to stake POD tokens and earn esVKA rewards primarily*
- *RewarderPerSec: to calculate the reward rate for the staking*
- *Claim: Allows users to claim their rewards*
- *Treasury: To concentrate the rewards accrued by the protocol and allow revenue split*
- *Boosting: To boost the staking rewards for VKA stakers*

Privileged roles

- The owner of DualStaking, SingleStaking, Treasury, Claim, Boosting, Vester3Months, Vester12Months and esVKAToken will be able to access the onlyOwner() functions and modify state variables inside those contracts
- The handler of the Vester3Months and Vester12Months will be able to access the functions deposit and withdraw for accounts
- The handler of the esVKAToken will be able to transferFrom and burn esVKATokens
- The handler of Claim will be able to pause the contract, call the functions notifyClaimable(), notifyAdditionalClaimable() and transferExpired()

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **1** out of **10**.

- Functional requirements are provided, but invalidated by the change of code in the deployment stage.
- Technical description is not provided.
- The development environment is described.
- NatSpec is provided.

Code quality

The total Code Quality score is **5** out of **10**.

- The development environment is configured.
- Best Practices are not followed.

Test coverage

Code coverage of the project is **70.85%** (branch coverage)

- Tests are not sufficient.

Security score

As a result of the audit, the code contains **1** medium severity issue. The security score is **9** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **6.4**. The system users should acknowledge all the risks summed up in the risks section of the report.

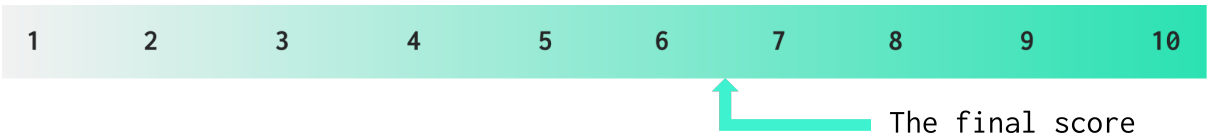

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
17 August 2023	11	18	14	6
03 October 2023	1	3	3	0
10 November 2023	0	1	0	0

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Risks

- **The client deployed unchecked code to the blockchain and subsequently initiated a Token Generation Event. The VKAToken.sol contract was submitted for review and then modified prior to deployment. (Contract address on Arbitrum: 0xAFccb724e3aec1657fC9514E3e53A0E71e80622D)."**

  Additionally, the client inaccurately and publicly claimed that Hacken audited the code when they made changes to the deployed code.

  Given that the client proceeded with the Token Generation Event, misrepresented the audit status by Hacken, and altered the code pre-deployment from what was submitted for the audit raises concerns. These actions compromise the relevance of the audit and raise questions about the assurance against future modifications and deployments of other contracts.

- Centralized protocol with upgradeable contracts
- **There are out of scope contracts and the client cannot provide the code needed to verify the correctness of the contract in the scope, this lowers severely the security of this audit.**
- Out of scope contract `LeveragedVault` has an important role of token burning, but it is not provided in the codebase.
- withdrawalTimeLock can change after the user has deposited funds
- In the `DualStaking` contract, some rewards are provided as USDC tokens. The rewards are calculated using 18 decimals, but USDC has only 6 decimals on certain chains.
- The `Owner` of `esVKAToken` has the potential to whitelist who can transfer or receive tokens using `setRecipientAllowed()`. This can effectively lock tokens in a wallet if `isRecipientAllowed[_handler]` is set to `false`.
- The function `safeesVKATransfer()` in `SingleStaking.sol` will send to the users as many tokens as available in the contract. In case there are not enough tokens, the user will have to wait until the protocol managers refill the contract with more tokens.
- In the contract `VaultkaTreasury`, the function `withdrawFees()` was added in order to recover the leftover tokens once the project becomes obsolete. However, the function can be used at any moment to withdraw the fees collected for `Protocol`, `Buyback` or `DualStaking`.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

■■■■ Critical

C01. Denial Of Service

Impact	High
Likelihood	High

The claim contract has a pause function, but no function to unpause the contract.

Path: /contracts/Claim.sol

Recommendation: Remove the possibility to pause or add the possibility to unpause.

Found in: 67124d3

Status: Fixed (3407a61).

C02. Race Condition

Impact	High
Likelihood	High

There are 2 calls in *allocateRevenueSplit()* to the *UniswapV2Router* with *amountOutMin* set to 0, this will cause a race condition and the swap will be frontrunned.

Path: /contracts/Treasury.sol : allocateRevenueSplit()

Recommendation: Use an oracle to know the amountOutMin desired.

Found in: 67124d3

Status: Mitigated (3407a61)

Resolution:

The solution proposed does not use oracles, it uses the expectAmountsOut variable that could still be susceptible to attacks, but it is much less likely, the variable uniswapSlippage should be kept on a reasonable value for this solution to not create problems and the amounts of supportedAssets should be kept under a certain amount to avoid a DoS vulnerability.

C03. Requirements Violation

Impact	High
Likelihood	High

The documentation states that there is a maximum amount of VKA tokens and that esVKA tokens can be swapped for VKA tokens 1:1 in the 12

months vester contract, this contradicts the maximum supply for the VKA tokens.

Path: /contracts/Tokens/VKAToken.sol

Recommendation: Fix the mismatch between the code and the documentation.

Found in: 67124d3

Status: Mitigated (3407a61)

Resolution:

The maximum supply cannot be checked since it is a variable passed in the constructor of both the esVKAToken and the VKAToken, the owner of the protocol needs to verify that the maximum supply is compliant to the functional requisites.

C04. Highly Permissive Role Access

Impact	**High**
Likelihood	**High**

The handler can send tokens from any account with the *transferFrom()* function.

Protocols should not be able to access users' funds without their permission.

Path: /contracts/Tokens/esVKAToken.sol transferFrom();

Recommendation: Document properly the behavior or remove it.

Found in: 67124d3

Status: Fixed (3407a61).

C05. Unverifiable Logic

Impact	**High**
Likelihood	**High**

In the *SingleStaking.sol* contract, there is an interaction with a contract implementing *ILeverageVault* that is out of scope.

In *RewarderPerSec.sol*, the contract implementing IMasterChefVaultka interface is used, but it is out of scope.

Paths:
./contracts/Staking/SingleStaking.sol
./contracts/Staking/RewarderPerSec.sol

Recommendation: Add the unverifiable logic to the scope.

Found in: 67124d3

Status: Mitigated (3407a61)

Resolution:

The client cannot provide the code needed to verify the correctness of the contract in the scope, this lowers severely the security of this audit.

C06. Data Consistency

Impact	High
Likelihood	High

The function *notifyAdditionalClaimable()* updates the value *hasClaimable[user][currentId]* instead of *hasClaimable[user][id]*, resulting in inconsistent data being stored and unexpected behavior.

Path: ./contracts/Claim.sol: notifyAdditionalClaimable().

Recommendation: Use id instead of currentId when updating hasClaimable.

Found in: 3407a61

Status: Fixed (3407a61).

■ ■ ■ High

H01. Highly Permissive Owner Access

Impact	High
Likelihood	Medium

The handler can use the function *claimForAccount()* to manipulate the claim of users of the platform

Paths: /contracts/Vesting/Vester3Months.sol : claimForAccount();

/contracts/Vesting/Vester12Months.sol : claimForAccount();

Recommendation: Document or remove the functionality.

Found in: 67124d3

Status: Fixed (3407a61).

H02. Undocumented Functionality; Highly Permissive Role Access

Impact	High

Likelihood	Medium

The owner could set a very small time in the *Claim.sol* contract and withdraw all the expired transfers, or a very long time and effectively lock the rewards forever.

Additionally, users have the same time constraint (claimable.startTime + claimable.claimablePeriod) to claim their rewards than the handler to retrieve them. Hence, there is no guarantee the users will get their rewards.

Path: /contracts/Claim.sol : transferExpired();

Recommendation: Add reasonable limits to claimable period, and add some extra time for the users to be able to get their tokens before the function transferExpired() can be called.

Found in: 67124d3

Status: Mitigated (3407a61)

Resolution:

A variable called MIN_CLAIMABLE_PERIOD has been added, the users will have a minimum of 7 days and a maximum of 730 days to claim.

H03. Missing Validation

Impact	High
Likelihood	Medium

tokenPerSec should not be higher than 1e30, in the function *setRewardRate()* it is possible to set it higher because there is a missing validation.

setrewardrate allows to break the rule of setting *tokenpersec* > 1e30

Path: /contracts/Staking/RewarderPerSec : setRewardRate();

Recommendation: Add the validation.

Found in: 67124d3

Status: Fixed (3407a61).

H04. Requirements Violation

Impact	High
Likelihood	Medium

The documentation states that esVKA is a non transferable token, this is not true, the token presents transfer functions.

Path: /contracts/Tokens/esVKAToken.sol : _transfer(), transfer(), fransferFrom();

Recommendation: Fix the mismatch between the documentation and the code.

Found in: 67124d3

Status: Fixed (3407a61).

H05. Undocumented Functionality

Impact	High
Likelihood	Medium

In the *Vester3Months* contract, a user could deposit esVKA token, get slashed and withdraw half of the deposited tokens, this functionality is not documented.

Path: /contracts/Vesting/Vester3Months.sol : deposit(), withdraw();

Recommendation: Do not slash the reward unless it is withdrawn as VKA tokens.

Found in: 67124d3

Status: Fixed (3407a61).

H06. Lost Claimable Funds

Impact	High
Likelihood	Medium

In the *Claim.sol* contract, if the *notifyClaimable()* function gets call on someone that has already claimable funds, the old claimable funds will be overwritten.

Path: /contracts/Claim.sol : notifyClaimable();

Recommendation: Add the rewards instead of overriding it.

Found in: 67124d3

Status: Mitigated (3407a61)

Resolution:

Client Response: "*the claimableBalance mapping includes a user and id, each id representing a different reward campaign. Thus, the claimable funds are not overwritten but stored in a different id*".

H07. Highly Permissive Role Access

Impact	High
Likelihood	Medium

In the *SingleStaking.sol* contract a fee of 100% could be applied.

In Treasury.sol, fee, dualStakingFee, protocolFee and buyBackFee have no limits.

Paths:
./contracts/Treasury.sol: setFee(), setProtocolFees().

Recommendation: Limit the fees to a reasonable number.

Found in: 67124d3

Status: Mitigated (Intended behavior) (3407a61).

H08. Unfinalized Code

Impact	Medium
Likelihood	High

The code has a lot of notes and todos with unfinished code and proposed functionalities that are not implemented.

Paths: /contracts/Staking/SingleStaking.sol : updatePool(), esVKAPerSec;
/contracts/Staking/DualStaking.sol : claimProtocolFee();
/contracts/Tokens/VKAToken.sol : constructor();

Recommendation: Finalize the unfinalized code.

Found in: 67124d3

Status: Fixed (3407a61).

H10. Undocumented Functionality

Impact	High
Likelihood	Medium

The state variable ACC_TOKEN_PRECISION in RewarderPerSec is 1e36 but there is no reason for the value to be that high, and the original MasterChef contract uses a different value.

In SingleStaking, ACC_ESVKA_PRECISION has 1e18, which is a newly different value.

There is no test provided to make sure this value keeps a consistency when used during the code.

There is no additional documentation or information about the order of magnitude of input or stored variables' decimals, so it is very difficult to determine this issue.

Path: ./contracts/Staking/RewarderPerSec.sol: ACC_TOKEN_PRECISION.

Recommendation: It is recommended to provide additional information and tests to make sure the decimals consistency is maintained.

Found in: 67124d3

Status: Mitigated (3407a61)

Resolution:

1e36 was changed to 1e18, but no explanation was given to why the 1e36 was there in the first place, this issue is considered fixed, but without a proper documentation it is not possible to ensure that everything in the code follows the functional requirements.

H11. Loss of Funds

Impact	**High**
Likelihood	Medium

The function *safeesVKATransfer()* will send to the users as many tokens as available in the contract, but in case there are not enough tokens, it will send them less than they are entitled.

Path: ./contracts/Staking/SingleStaking.sol: safeesVKATransfer().

Recommendation: It is recommended to implement a check to make sure the users will receive all tokens they are owed or to record the missing tokens in the protocol so that those tokens can be received later.

Found in: 67124d3

Status: Fixed (3407a61).

H12. Locked Funds

Impact	**High**
Likelihood	Medium

In allocateRevenueSplit(), a portion of the funds is kept in the contract as feesCollectedForBuyback.

However, when the protocol is no longer used, the funds will get stuck since there is no method to retrieve the leftover funds.

Path: ./contracts/Treasury.sol: allocateRevenueSplit().

Recommendation: Consider adding a method to retrieve leftover funds.

Found in: 67124d3

Status: Fixed (3407a61).

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

H13. Requirements Violation

Impact	High
Likelihood	Medium

In _getNextClaimableAmount(), there are missing time checks to make sure that:

- timeDiff cannot be higher than the total vesting time
- the claim time starts after the timestamp set by the protocol, not before
- the claim time ends no longer than the timestamp set by the protocol

Due to the lack of said checks, timeDiff can be higher than expected, resulting in a higher claimableAmount than expected.

Paths:
./contracts/Vesting/Vester3Months.sol: _getNextClaimableAmount().
./contracts/Vesting/Vester12Months.sol: _getNextClaimableAmount().

Recommendation: Implement the necessary checks to make sure the claimable period follows the recommended checks or equivalent.

Found in: 67124d3

Status: Mitigated (3407a61)

(Intended behavior, Vaultka team takes full responsibility for this feature.)

Medium

M01. Contradiction

Impact	Medium
Likelihood	Medium

The vesting duration is a variable instead of a constant of 3 or 12 months.

Path: /contracts/Vesting/Vester3Months.sol : vestingDuration;
/contracts/Vesting/Vester12Months.sol : vestingDuration;

Recommendation: Fix the mismatch.

Found in: 67124d3

Status: Fixed (3407a61).

M02. Highly Permissive Role Access

Impact	High

Likelihood	Medium

The owner can withdraw reward tokens from the *Claim* contract using the *withdrawToken()* function.

Protocols should not be able to access users' funds without their permission.

Path: ./contracts/Claim.sol : withdrawToken().

Recommendation: Limit the functionality so that user rewards cannot be affected or add a time interval so that users can have some time to withdraw their tokens before the transfer is triggered.

Since this is a highly sensitive function, it is recommended to use multi-signature for the owner account.

Found in: 67124d3

Status: Mitigated (3407a61)

Resolution:

Client Response: *"The claim contract is designed to distribute rewards to users on a nominative basis. We don't "steal" users' funds as it is our plain right to distribute rewards or not, and users do not have any funds staked"*.

M03. Contradiction

Impact	Medium
Likelihood	Medium

There is a comment in *notifyRewardAmount()* in the *DualStaking.sol* contract that explains the math that is not followed later.

Path: /contracts/Staking/DualStaking.sol : notifyRewardAmount();

Recommendation: Fix the mismatch.

Found in: 67124d3

Status: Fixed (3407a61).

M04. Undocumented Functionality

Impact	Medium
Likelihood	Medium

The case where there is no staked amount in the *DualStaking.sol* contract is never documented, in that case, the rewards sent with the *receiveProtocolFees()* function are not accounted for and therefore the transfer makes no sense.

Path: /contracts/Staking/DualStaking.sol : receiveProtocolFees();

Recommendation: Consider providing additional documentation to explain this case. Additionally, the transfer should not be performed if accUSDCperTokens is not updated.

Found in: 67124d3

Status: Fixed (3407a61).

M05. Undocumented Functionality

Impact	Medium
Likelihood	Medium

Document what should happen in the function *unstakeAndLiquidate()* in case the *rewarder* address is 0 and the pool does not allow double rewards.

Path: /contracts/Staking/SingleStaking.sol : unstakeAndLiquidate();

Recommendation: Remove the functionality or mention it in documentation.

Found in: 67124d3

Status: Fixed (3407a61).

M06. Undocumented Functionality

Impact	Medium
Likelihood	Medium

The *RewarderPerSec* contract implements an interface of itself, it is not clear if *IRewarder* is the same contract as *RewarderPerSec*.

Path: /contracts/Staking/RewarderPerSec

Recommendation: Remove the functionality or mention it in documentation.

Found in: 67124d3

Status: Fixed (3407a61).

M07. Undocumented Functionality

Impact	Medium
Likelihood	Medium

esVKA tokens should not be transferable, they are, but to be transferred in the vester the user needs to be allowed.

Path: /contracts/Tokens/esVKATokens.sol : _transfer();

Recommendation: Remove the functionality or mention it in documentation.

Found in: 67124d3

Status: Mitigated (Intended Behavior) (3407a61).

M08. Highly Permissive Role Access

Impact	**High**
Likelihood	**Low**

The owner can withdraw esTokens from the 12 month vester contract.

Path: /contracts/Vesting/Vester12Months.sol withdrawToken();

Recommendation: Limit the functionality.

Found in: 67124d3

Status: Fixed (3407a61).

M09. Best Practice Violation - Checks-Effects-Interactions Pattern

Impact	**High**
Likelihood	**Low**

State variables are updated after the external calls to the token contract.
As explained in Solidity Security Considerations, it is best practice to follow the checks-effects-interactions pattern when interacting with external contracts to avoid reentrancy-related issues.

Paths:
./contracts/Staking/RewarderPerSec : oneVKAReward();
./contracts/Staking/DualStaking : getReward(), notifyRewardAmount(), receiveProtocolFees();
./contracts/SingleStaking: updatePool(), deposit(), updateBoostMultiplier().
./contracts/Claim.sol: notifyClaimable().
./contracts/Treasury.sol: allocateRevenueSplit().
./contracts/Vesting/Vester3Months.sol: withdraw(), _deposit().

Recommendation: Follow the checks-effects-interactions pattern when interacting with external contracts.

Found in: 67124d3

Status: Fixed (3407a61).

M10. Highly Permissive Owner Access

Impact	**Medium**

Likelihood	Medium

In the *DualStaking.sol* contract, there is a withdrawal timelock for the funds that is not documented nor limited by code; funds might be locked forever with the current implementation.

Path: /contracts/Staking/DualStaking.sol

Recommendation: Limit the timelock and document it.

Found in: 67124d3

Status: Fixed (3407a61).

M11. Undocumented Functionality

Impact	High
Likelihood	Low

The withdrawalTimeLock can be updated by the Owner at any moment. This means a user can stake tokens expecting a certain lock period, but it can change at any point without users' notice.

For example, a user can stake some tokens expecting a month duration, but it can change to one year.

Path: ./contracts/Staking/DualStaking.sol: setWithdrawalTimeLock().

Recommendation: Document this behavior to notify users about the possibility or implement a system in which this situation is not possible.

Found in: 67124d3

Status: Fixed (Limited to 48 hours) (3407a61).

M12. Division Before Multiplication

Impact	Low
Likelihood	High

Since Solidity language does not have floating point numbers, performing divisions before multiplications results in a loss of precision.

Paths:
./contracts/Staking/SingleStaking.sol: pendingTokens(), updatePool(), withdraw(), unstakeAndLiquidate(), updateBoostMultiplier(), _settlePendingESVKA().

Recommendation: It is recommended to perform divisions after multiplications to avoid loss of precision.

Found in: 67124d3

Status: Fixed (3407a61).

M13. Data Consistency

Impact	High
Likelihood	Low

When users stake or withdraw tokens into *DualStaking* contract the variable accUSDCperTokens is not updated.

As a consequence, new deposits will not decrease the reward per token in USDC, resulting in unfair rewards and front-running situations since latest withdrawals will have none.

Additionally, withdrawals will not increase the reward per token in USDC, and thus the latest users to withdraw will not benefit from it.

Path: /contracts/Staking/DualStaking.sol: stake(), withdraw().

Recommendation: accUSDCperTokens should be updated every time totalStakedAmount changes or the behavior should be documented.

Found in: 67124d3

Status: Mitigated (3407a61)

Resolution:

Client Response: "*Unlike the master-chef style emissions that occur over a specified timeframe, the USDC fee rewards involve taking a snapshot at the moment of receiveProtocolFees being called . This allows users to claim the whole share of rewards at the moment of receiveProtocolFees being called. As a result, behaviors such as 'new deposits not diluting the reward per token' and 'withdrawals not increasing the reward per token' are indeed expected.*"

M14. Uncontrolled Loop of Storage Interactions; DOS

Impact	High
Likelihood	Low

The function *massUpdatePools()* can become unusable if the stored variable *poolInfo* increases enough.

The number of operations that must be performed in order to get all the transactions a user performs depends on stored data, and it can reach the block Gas limit. Eventually, it can block all functions interacting with massUpdatePools(), resulting in a Denial of Service.

A solution for this issue is to add indexes as function parameters to interact with a given amount of pools instead of all at the same time.

Path: ./contracts/Staking/SingleStaking.sol: massUpdatePools().

Recommendation: It is recommended to implement a function to update all pools with fixed array indexes in case the original massUpdatePools() will fail for excess Gas. Alternatively, add the bounds in massUpdatePools() directly.

Found in: 67124d3

Status: Mitigated (3407a61).

Resolution:

Client Response: "*In implementation we will have a small number of pools.*"

M15. Requirements Violation

Impact	**High**
Likelihood	**Low**

The function decreaseRewardRate() is called weekly according to the provided information in its NatSpec, but there is no enforcement in the code (e.g. time check).

Path: ./contracts/Staking/RewarderPerSec.sol: decreaseRewardRate();

Recommendation: Update the code or documentation so that they match.

Found in: 67124d3

Status: Fixed (3407a61).

M16. Data Consistency; Denial Of Service

Impact	**High**
Likelihood	**Low**

assetPath in the *addAsset()* function in the *Treasury.sol* contract will be overwritten with the wrong path if the variable *_pathToUSDC* has more than 1 element and the variable *_path* is *true*.

Path: ./contracts/Treasury.sol : addAsset();

Recommendation: Do not overwrite the variable, add the paths one after the other.

Found in: 67124d3

Status: Mitigated (3407a61).

Resolution:

Client Response: "*Overwriting the path is an intended behavior.*"

M17. Data Consistency; MissingCheck

Impact	High
Likelihood	Low

In `notifyAdditionalClaimable()`, there is no check to make sure the id exists.

As a consequence, given the case this function is called for an id before using `notifyClaimable()`, there will be inconsistencies with the recorded data and result in unexpected behavior.

Path: ./contracts/Claim.sol : notifyAdditionalClaimable();

Recommendation: Consider implementing a check to make sure the id exists.

Found in: 67124d3

Status: Fixed (3407a61).

M18. Inefficient Gas Model

Impact	Medium
Likelihood	Medium

In `removeAsset()`, the array supportedAssets is not reduced when an asset is removed. Instead, the element of the array corresponding to the removed asset is substituted by address(0).

As a consequence, the array keeps getting bigger and more expensive to read without benefit, since the asset position does not affect other places of the code.

Path: ./contracts/VaultkaTreasury.sol : removeAsset();

Recommendation: Consider removing assets instead of writing address(0), by substituting it for the last element of the array and then using pop().

Found in: 67124d3

Status: Fixed (3407a61).

H09. Highly Permissive Role Access

Impact	Medium
Likelihood	Medium

The owner can withdraw reward tokens and USDC tokens from the *DualStaking.sol* contract using the *recoverERC20()* function.

Protocols should not be able to access users' funds without their permission.

Path: /contracts/Staking/DualStaking : recoverERC20();

Recommendation: Limit the functionality so that user rewards cannot be affected or add a time interval so that users can have some time to withdraw their tokens before the transfer is triggered.

Since this is a highly sensitive function, it is recommended to use multi-signature for the owner account.

Found in: 67124d3

Status: Acknowledged

Low

L01. Missing Zero Address Validation

Impact	Low
Likelihood	Low

Address parameters are being used without checking against the possibility of 0x0.

This can lead to unwanted external calls to 0x0.

Paths:
./contracts/Staking/SingleStaking.sol : constructor(), dev(), setTreasuryAddr(), setInvestorAddr(), udateBoostMultiplier().
./contracts/Treasury.sol : initialize(), setUniRouters(), addAsset(), setBuyBackpath().
./contracts/Staking/DualStaking.sol: constructor().
./contracts/Staking/RewarderPerSec.sol: onesVKAReward(), pendingTokens().
./contracts/Boosting.sol: getBoostMultiplier(), getBoostMultiplierWithDeposit().
./contracts/Claim.sol: setHandler(), notifyClaimable(), notifyAdditionalClaimable(), transferExpired().
./contracts/Vesting/Vesting3Months.sol: constructor, setHandler(), depositForAccount(), claimForAccount(), claimable().
./contracts/Vesting/Vesting3Months.sol: constructor, setHandler(), depositForAccount(), claimable().

Recommendation: Implement zero address checks.

Found in: 67124d3

Status: Fixed (3407a61).

L02. Style Guide Violation

Impact	Low
Likelihood	Low

The provided projects should follow the official guidelines. Especially pay attention to ‘Order of Layout’. Following the Solidity Style guidelines facilitates conde comprehension, increases readability and makes supporting code easier. The following rules have been violated:

- variables names must be in mixed case
- the order of functions should be followed

Path: /contracts/*

Recommendation: Follow the official Solidity guidelines.

Found in: 67124d3

Status: Fixed (3407a61).

L03. Redundant SafeMath

Impact	Low
Likelihood	Low

Using the *SafeMath* library on a contract that uses solidity 0.8.0 and higher is an inefficient Gas model because the compiler already handles underflows and overflows.

Path: /contracts/*

Recommendation: Remove the library and use the standard math operators “+-*/” instead of safemath’s functions “sub,mul,div…”.

Found in: 67124d3

Status: Fixed (3407a61).

L04. Inefficient Gas Model

Impact	Low
Likelihood	Low

The *updatePool()* function gets called twice in the *deposit()* function.

Path: /contracts/Staking/SingleStaking.sol : deposit();

Recommendation: Limit the calls of the *updatedPool()*.

Found in: 67124d3

Status: Fixed (3407a61).

L05. Best Practice Violation

Impact	Low
Likelihood	Low

It is a best practice to check the return values for token transfers.

Paths:
./contracts/Staking/SingleStaking.sol : updatePool(), safeesVKATransfer();
./contracts/Treasury.sol: allocateRevenueSplit().

Recommendation: Use *SafeTransfer* to transfer tokens.

Found in: 67124d3

Status: Fixed (3407a61).

L06. Redundant View Functions

Impact	Low
Likelihood	Low

The following functions return state variables that are already public, and thus can already be read without creating additional functions.

Redundant code should be removed from the contracts for simplification and decrease of the contract deployment Gas cost.

Paths:
./contracts/Staking/DualStaking.sol : balanceOf(), balanceOfTokens(), totalStaked(), stakedAmount().
./contracts/Tokens/esVKAToken.sol: balanceOf(), allowance().
./contracts/Treasury.sol: getAssets().
./contracts/Vesting/Vester3Months.sol: balanceOf(), getTotalVested().
./contracts/Vesting/Vester12Months.sol: balanceOf(), getTotalVested().

Recommendation: Consider removing redundant functions.

Found in: 67124d3

Status: Fixed (3407a61).

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

L07. Inefficient Gas Model; Redundant State Variable Access

Impact	Medium
Likelihood	Low

The state variable dr.debtUSDC is updated twice when calling stake() and withdraw(), since it has been already set in claimProtocolFee(), spending more Gas.

Path: /contracts/Staking/DualStaking.sol : stake(), withdraw(), claimProtocolFee();

Recommendation: Consider removing the storage access from stake() and withdraw().

Found in: 67124d3

Status: Fixed (3407a61).

L08. Unused Variable

Impact	Medium
Likelihood	Low

The call to get bonusTokenSymbol is unused since the function rewarderBonusTokenSymbol() does not return the symbol.

Path: /contracts/Staking/SingleStaking.sol: pendingTokens(), rewarderBonusTokenSymbol().

Recommendation: Consider removing the bonusTokenSymbol variable in pendingTokens() and the returned variable with the same name in rewarderBonusTokenSymbol().

Found in: 67124d3

Status: Fixed (3407a61).

L09. Redundant SafeERC20

Impact	Low
Likelihood	Low

Using the *SafeERC20* library on a token contract is unnecessary.

Path: /contracts/Tokens/esVKAToken.sol

Recommendation: Remove the library.

Found in: 67124d3

Status: Fixed (3407a61).

L10. Inefficient Gas Model

Impact	Low
Likelihood	Low

In notifyClaimable(), the state variable currentId is read multiple times for a lot of Gas, instead of using a memory variable.

In allocateRevenueSplit(), the state variable supportedAssets[i] is read multiple times for a lot of Gas, instead of using a memory variable.

Paths:
./contracts/Claim.sol: notifyClaimable().
./contracts/Treasury.sol: allocateRevenueSplit().

Recommendation: Consider creating a memory variable for currentId to be used for all cases in which that variable is read.

Found in: 67124d3

Status: Fixed (3407a61).

L11. Inefficient Gas Model

Impact	Low
Likelihood	Medium

In notifyClaimable() and notifyAdditionalClaimable(), the array _users is read at every iteration of the for loop to calculate its length.

In claimMultiple(), a similar situation appears with _ids.

By caching the length, creating a new *memory* variable *length* to be used within the iteration, it is possible to save Gas.

Paths:
./contracts/Claim.sol: notifyClaimable(), notifyAdditionalClaimable().

Recommendation: It is recommended to cache the length of said arrays.

Found in: 67124d3

Status: Fixed (3407a61).

Informational

I01. Unindexed Events

Having indexed parameters in the events makes it easier to search for these events using indexed parameters as filters.

Paths: /contracts/Vesting/Vester3Months.sol Claim(), Deposit(), Withdraw();

/contracts/Vesting/Vester12Months.sol Claim(), Deposit(), Withdraw();

Recommendation: Use the “indexed” keyword to the event parameters.

Found in: 67124d3

Status: Fixed (3407a61).

I02. Functions Should Be Declared External

In order to save Gas, public functions that are never called in the contract should be declared as external.

Paths:
./contracts/Staking/DualStaking.sol : totalStaked(), stakedAmount(), harvest();
./contracts/Staking/SingleStaking.sol : add(), set(), emergencyWithdraw(), dev(), setDevPercent(), setTreasuryAddr(), setTreasuryPercent(), setInvestorAddr(), setInvestorPercent(), updateEmissionRate(), getUserAmonut(), deposit(), withdraw(), unstakeAndLiquidate(), updateBoostMultiplier().
./contracts/Vesting/Vester3Months.sol : getTotalVested(), balanceOf(), transfer(), allowance(), approve() ;
./contracts/Vesting/Vester12Months.sol : getTotalVested(), balanceOf(), transfer(), allowance(), approve(), transferFrom();
./contracts/Boosting.sol : setStakingContracts(), setMaxBoostPrecision(), getBoostMultiplier(), getBoostMultiplierWithDeposit();
./contracts/Staking/RewarderPerSec.sol: decreaseRewardRate().
./contracts/Treasury.sol: setesVKAStaking(), setUniRouters(), setProtocolReserveFund(), setFee(), setProtocolFees(), addAsset(), removeAsset(), setBuyBackpath(), allocateRevenueSplit().

Recommendation: Use the external attribute for functions never called from the contract.

Found in: 67124d3

Status: Fixed (3407a61).

I03. State Variables Can Be Declared Immutable

Some variables are assigned a value only in the constructor and never changed later.

To lower the Gas fees, these variables can be declared as immutable.

Paths:
./contracts/Staking/DualStaking.sol: stakingToken1, stakingToken2, rewardToken, usdcToken.
./contracts/Staking/SingleStaking.sol: esVKAToken, boostContract,

startTimestamp.
./contracts/Vesting/Vester3Months.sol: vestingDuration, esToken, claimableToken.
./contracts/Vesting/Vester12Months.sol: vestingDuration, esToken, claimableToken.

Recommendation: Declare mentioned variables as immutable.

Found in: 67124d3

Status: Acknowledged

I04. Long Uint Literals

In the VKA and esVKA contracts, there are various uint with long literals that are not properly separated to aid readability.

Path: ./contracts/Tokens/*

Recommendation: Rewrite the long literals.

Found in: 67124d3

Status: Fixed (3407a61).

I05. Missing Events for Critical Value Updates

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Paths:
./contracts/Staking/DualStaking.sol: setTreasury(), setWithdrawalTimeLock().
./contracts/RewardsPerSec.sol: constructor() → tokenPerSec.
./contracts/SingleStaking.sol: constructor() → devAddr, treasuryAddr, investorAddr, esVKAPerSec, devPercent, treasuryPercent, investorPercent; setLeverageVault(); setDoubleRewardsAsset(), setDevPercent(), setTreasuryPercent(), setTreasuryAddr(); setInvestorAddr(), setInvestorPercent().
./contracts/Tokens/esVKAToken.sol: setRecipientAllowed(), setInPrivateTransferMode(), setHandler().
./contracts/Boosting.sol: setStakingContracts(), setMaxBoosPrecision().
./contracts/Claim.sol: constructor().
./contracts/Treasury.sol: initialize() → uniRouter, fee.
./contracts/Vester/Vester3Months.sol: setHandler().
./contracts/Vester/Vester12Months.sol: setHandler().

Recommendation: Consider emitting *events* in said functions.

Found in: 67124d3

Status: Acknowledged

I06. Best Practice Violation; Non-Explicit Variable Unit Sizes

Variable type uint is used without explicitly setting its size.

Path: ./contracts/Staking/DualStaking.sol: pendingRewardsUSDC().

Recommendation: It is a best practice to explicitly set the size of uint variable types.

Found in: 67124d3

Status: Fixed (3407a61).

I07. Reference To Other Contracts

There are constant references to Joe contracts instead of their own.

Path: ./contracts/Staking/RewardsPerSec.sol.

Recommendation: Consider updating the references to the own protocol instead of other protocols.

Found in: 67124d3

Status: Fixed (3407a61).

I09. Variables That Can Be Set Constant

The variables BOOST_PRECISION and BASE_MULTIPLIER are hardcoded in the constructor and never change. Instead, it is recommended to set those variables as *constant* to save Gas.

Paths:
./contracts/Boosting.sol: BOOST_PRECISION, BASE_MULTIPLIER.

Recommendation: Consider using the keyword constant for said variables.

Found in: 67124d3

Status: Acknowledged

I10. Typo

At line 11 of the *DualStaking.sol* contract, it is written accpet instead of accept.

Paths:
./contracts/Staking/DualStaking.sol

Recommendation: Fix the typo.

Found in: 67124d3

Status: Fixed (3407a61).

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	**Critical**	**High**	**Medium**
Medium Likelihood	**High**	**Medium**	**Low**
Low Likelihood	**Medium**	**Low**	**Low**

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/Vaultka-Project/Vaultka-tokenomics
Commit	67124d365185e18bc30d9c4d9c652c323917b568
Requirements	$VKA Tokenomics (Vaultka).pdf
Contracts	File: contracts/Boosting.sol SHA3: d18eef9e06aba97aa9a3deb6bb053ec9da667f94e6497ff314a9468aad763815 File: contracts/Claim.sol SHA3: e4a32d3f8382b21f92ec5e83b8b90dce9a1514d7cfad08e7c0433dfb3f72f232 File: contracts/VaultkaTreasury.sol SHA3: c00fdc1209c0576982375b5490b5c665c686479c47175c7d5989e36ee100c17e File: contracts/Staking/DualStaking.sol SHA3: d20ea58be35b6b32dc2f3164e686af60f0696a55a8946047bbb655fa485ff7e3 File: contracts/Staking/RewarderPerSec.sol SHA3: ecde7e79998e65c4408e5e9c4f8f0b7abab3432981cfe6a1f260902c2becaffa File: contracts/Staking/SingleStaking.sol SHA3: 96585fe223343d22868bf95775fab4c5500ef4ec97d4e9ea91120b499e7e8f04 File: contracts/Tokens/esVKAToken.sol SHA3: 4e6d6b3de60b21258f3b11ba231e6803d572153041c802ea8f419e923bb86336 File: contracts/Tokens/VKAToken.sol SHA3: d8510bb53d9e2ef9b316a097c78581794351529ebe6e8531e0998d86fb0c9042 File: contracts/Vesting/Vester12Months.sol SHA3: 54fc653bad71f823ff982aafeb88a8aaf9a6cc87ed39787b95ea8ec8a8f027f0 File: contracts/Vesting/Vester3Months.sol SHA3: 4632cca12640d963589a6259058a124569be1e92e69ea8ba4b9f57d19d923592 File: contracts/interfaces/IesVKAStaking.sol SHA3: 5b454dc66a6e2d33cb9cd6404f5b2759f1b4bc6284f22ea7bd8215310c55e3d0 File: contracts/interfaces/IGMXStrategy.sol SHA3: e79928b1e43118e1bf5cf85d9d40538476a3a66761c14a3edad111fee298f284 File: contracts/interfaces/ILeverageVault.sol SHA3: fc2bf76eaa5274e5d21d4cd1710bfcc6afeee40bc011dea6c1810cb6ef871bd3 File: contracts/interfaces/ISwapRouter.sol SHA3: cb12f9bb69683710c8ba8e5fcc2116a9c3a822eeac9066e93c0ca2eee0d09d26 File: contracts/interfaces/ITokenBurnable.sol SHA3: ccb86bf020fab01cab638f4135e4c88175607eeebf904f54b6c8875bb13e728c File: contracts/interfaces/IUnirouter.sol SHA3: d0d7abbbb7d09870f8efbf0a9e5b5dede3347faa46b520a8f2a50def32f57a0d

Second review scope

Repository	https://github.com/Vaultka-Project/Vaultka-tokenomics
Commit	3407a6150129772ec494663a13989b4391b4c527
Requirements	https://docs.vaultka.com/welcome-to-vaultka/overview
Contracts	File: contracts/Boosting.sol SHA3: c94b6486a752c7a700eb4c0bef351bcc02c6e110a9a6a5676c44f7c27cfc83a1 File: contracts/Claim.sol SHA3: 471d609bdf2bec8c069b0232ce2972280e0e610d01f327cd3e0fa9fe27ee7b44 File: contracts/VaultkaTreasury.sol SHA3: 4cc72c68e4f65fcdc604f77f7effc84855070e333ec9d4318748b005443cd3f5 File: contracts/Staking/DualStaking.sol SHA3: d553fc8583bd3460caf804fe310bcb3109a1abac0529909dd1386275ee6a9238 File: contracts/Staking/RewarderPerSec.sol SHA3: e6c704596f5682df872890d1f92bd1349aca7306096cf54398cd912e35553241 File: contracts/Staking/SingleStaking.sol SHA3: 2dae9afe852235f18f5ba4e51471ba19dae5acd6205e546f33edda7a842923a7 File: contracts/Tokens/esVKAToken.sol SHA3: d5b9a89c8b1927f3813fe97e82af3cccd8decb90259830418eff3d71844d2d30 File: contracts/Tokens/VKAToken.sol SHA3: 096d8c38f93fb90cafe8c2e9312f0b2b391dbb9033ada6a0e85d1528aa40a1c9 File: contracts/Vesting/Vester12Months.sol SHA3: 55b4f40c4519713438613cff2894a9eeb87ba9d16876f70f8f899aec022a124b File: contracts/Vesting/Vester3Months.sol SHA3: c31ef1b0744554068e1e95149ca0f07491dd111784a7566d20f2d7952cf477da File: contracts/interfaces/IesVKAStaking.sol SHA3: 7a47a4775cfe213f1a5921cfb596020f915cf8aabf2f06734be0fa5fdef6563c File: contracts/interfaces/IGMXStrategy.sol SHA3: e79928b1e43118e1bf5cf85d9d40538476a3a66761c14a3edad111fee298f284 File: contracts/interfaces/ILeverageVault.sol SHA3: fc2bf76eaa5274e5d21d4cd1710bfcc6afeee40bc011dea6c1810cb6ef871bd3 File: contracts/interfaces/ISwapRouter.sol SHA3: cb12f9bb69683710c8ba8e5fcc2116a9c3a822eeac9066e93c0ca2eee0d09d26 File: contracts/interfaces/ITokenBurnable.sol SHA3: ccb86bf020fab01cab638f4135e4c88175607eeebf904f54b6c8875bb13e728c File: contracts/interfaces/IUnirouter.sol SHA3: d0d7abbbb7d09870f8efbf0a9e5b5dede3347faa46b520a8f2a50def32f57a0d