SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Openeden
Date: 17 October, 2023

Document

Name	Smart Contract Code Review and Security Analysis Report for Openeden
Approved By	Przemyslaw Swiatowiec | Lead Solidity SC Auditor at Hacken OÜ Eren Gonen | SC Auditor at Hacken OÜ Seher Saylik | SC Auditor at Hacken OÜ
Tags	ERC20 stable token
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://openeden.com/
Changelog	09.10.2023 - Initial Review 17.10.2023 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Openeden (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

Openeden, is an project that manages the TBILL stable ERC20 tokens. Users can deposit USDC to mint TBILL tokens, entitling them to redeem assets in proportion to their TBILL holdings. TBILL tokens are stored in whitelisted wallets, and the project's code governs deposit, withdrawal, and management functions, ensuring proper asset handling. It is built with the following contracts:

- *OpenEden T-Bills* — Simple ERC-20 token used for shares.
  It has the following attributes:
  - Name: OpenEden T-Bills
  - Symbol: TBILL
  - Decimals: 6
  - Total supply: No Max Supply Limit.
- *Controller* — The Controller contract provides mechanisms for pausing and unpausing specific operations (deposit and withdraw) in a system.
- *FeeManager* - The FeeManager contract provides mechanisms to manage various fee-related parameters in a system. This includes settings for transaction fees, deposit and withdrawal limits, management fee rates, and special considerations for weekends.
- *OpenEdenVaultV2* - OpenEdenVaultV2 is an upgradeable vault contract designed for managing deposits and withdrawals, charging fees, integrating with KYC systems, and operating under specific time-based rules.
- *TBillPriceOracle* - Oracle contract provides a way to manage and update TBill prices with constraints on how much the price can deviate from previous values.
- *Timelock* - Imports TimelockController from Openzeppelin.
- *OEPausable* - The contract is designed to introduce "pausing" functionality into a contract by inheritance. This pausing mechanism can be utilized for emergency scenarios or other use cases to temporarily halt certain operations of a contract.

Privileged roles

- The DEFAULT_ADMIN_ROLE of the Controller contract can:
  - Pause and unpause deposits and withdrawals.
- The OPERATOR_ROLE of the Controller contract can:
  - Pause and unpause deposits and withdrawals.
- The Owner of the FeeManager contract can:

- The owner can set various fee-related parameters like transaction fees, deposit and withdrawal limits, etc.
- The owner inherits the capabilities provided by the OpenZeppelin's Ownable contract, such as the ability to transfer ownership or renounce ownership.

- The Owner of the OpenEdenVaultV2 can:
    - Set the treasury for the vault.
    - Set the treasury specific to 'q' (qTreasury).
    - Toggle whether the USDC/USD price is fixed.
    - Set various addresses, such as FeeManager, KycManager, Operator, USDC Price Feed, TBill Price Feed, and Controller.
    - Authorize contract upgrades.
    - Upgrade the contract.
- The Operator of the OpenEdenVaultV2 can:
    - Initiate off-ramp operations to transfer underlying assets to designated treasuries.
    - Process the withdrawal queue.
    - Update the epoch and set whether it is a weekend.
    - Claim the service fee.
- The DEFAULT_ADMIN_ROLE of the TBillPriceOracle can:
    - Grant and revoke the OPERATOR_ROLE.
    - Can update the maximum price deviation.
    - Can manually update the close NAV price.
- The OPERATOR_ROLE of the TBillPriceOracle can:
    - Can update the price.
    - Can update the close NAV price.

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 **10** out of **10**.

- Functional requirements are provided.
- Technical description is provided.
- NatSpec is provided.

Code quality

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

Test coverage

Code coverage of the project is **93.88%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- The scenarios involving multiple user are tested thoroughly.
- Negative case coverage is partially missing for the *OEPausable.sol*, *DoubleQueueModified.sol*, and *OpenEdenVaultV2.sol* contracts.

Security score

As a result of the audit, the code contains **1** medium and **1** low severity issues. 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: **9.1**. The system users should acknowledge all the risks summed up in the risks section of the report.

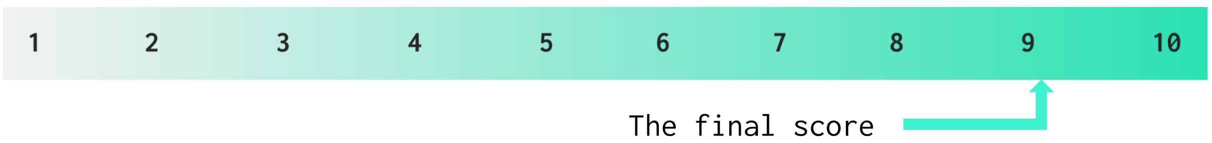

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
09 October 2023	5	1	0	2

17 October 2023	1	1	0	0

Risks

- At any given time, the owner or operator holds the capability to pause both withdrawals and deposits.
- The transaction fees within the system can be configured to any value, including 100%, which implies that users may receive nothing in return when attempting to deposit or redeem tokens. The **fee** is calculated and collected when the operator executes the withdrawal queue, **based on the latest fee rate, not the rate at the time of user redemption.**
- The **TBILL price** used in calculations is provided by an oracle through an **off-chain** mechanism and the implementation to sustain **stable coin mechanism** is also handled **off-chain**.
- The owner has the authority to **withdraw any token including USDC** from the contract.
- Only the operator has the authority to execute redemption requests, and the timing of execution is determined by the operator's discretion.
- The **Treasury**, **oplTreasury** and **KycManager** contracts are beyond the scope of this audit. The reliability of these contracts cannot be confirmed.
- The system employs a KYC process. There is a potential risk where a user, after receiving KYC approval and depositing tokens, could be subsequently banned. This would result in their tokens being permanently locked in their account.
- The redemption process relies on a centralized operator's discretion, and any issues or risks related to this operator's actions, such as insufficient USDC balance or centralization concerns, can result in users being unable to access their deposited TBILL tokens or the promised USDC tokens, potentially compromising the trustworthiness of the redemption process.
- There exists a risk where the backend system could fail to accurately set the **isWeekend** flag, which is crucial for enforcing the intended fee structure and deposit limits. In this case, users might be able to deposit funds under incorrect fee structures.
- The Open Eden operators can ban users, which may result in declining withdrawal requests if a user has already deposited funds. This mechanism was introduced to protect against withdrawal queue block due to USDC blacklisting mechanism.

Findings

■■■■ Critical

C01. Denial of Service; Assets Locked in Withdrawal Queue Due to Absent Verifications

Impact	High
Likelihood	High

The withdraw operations in the system are solely processed by the operator, starting from the front of the queue, and executing only first the frontmost operation. Users initiate withdrawal requests via the *redeem()* function, which allows them to specify transfer amounts and receiver addresses. This setup introduces a critical vulnerability that could lead to a collapse of the entire withdrawal queue mechanism through three distinct scenarios:

Scenario 1:

A malicious user sets the receiver address to the zero address when executing the redeem function. The redeem function lacks a restriction to prevent this action, allowing the user's request to be added to the withdrawal queue. Subsequently, when the operator decides to execute withdrawal operations from the queue, the malicious user's request, along with the upcoming ones, will be permanently stuck. This is because transfers to the zero address are prohibited in the provided ERC20 contract, effectively locking up these assets.

Scenario 2:

The fee for users’ redeem requests is not charged during the users call of redeem function. It’s charged when the operator process the withdrawals. When the calculated transaction fee for a user is smaller than the minTxsFee specified in the contract, and the user has no available assets in the withdrawal queue data, both the transaction and so the withdrawal queue will always fail.

Scenario 3:

Another scenario involves the use of a blacklisted USDC user as the recipient. If the recipient address is blacklisted by the USDC contract, any attempt to transfer USDC tokens to that address will also cause a reversion, leading to a similar denial of service condition.

All of these scenarios result in users' assets / orders being locked within the contract, rendering the system non-functional and severely damaging its reputation.

Path: ./contracts/OpenEdenVaultV2.sol: redeem(), processWithdrawalQueue()

Proof of Concept:

1. A malicious user calls the redeem() function and intentionally sets the receiver address to the Ethereum 0x00 address (also known as the zero address).
2. The contract does not have any checks to prevent this action, so the malicious user's request is added to the withdrawal queue.
3. The operator, responsible for processing withdrawals, attempts to process the queue. When the operator reaches the malicious user's request, the withdrawal operation fails. This is because transfers to the zero address are prohibited in standard ERC20 contracts.
4. As a result, the entire withdrawal queue becomes jammed. The operator cannot process this request nor any subsequent requests in the queue.

Recommendation: In the redeem function, add a check to ensure that the receiver address is not set to the zero address. Reject the transaction if the receiver address is invalid.

Charge the fee during the redeeming process and do not allow processing if the amount to be received is zero after the charging.

Enhance the withdrawal queue mechanism to handle and skip invalid or malicious requests gracefully, ensuring that the entire system does not become paralyzed due to such requests. To cancel an order, original assets must be refunded to the corresponding user.

All given recommendations should be implemented.

Found in: 4aed24d

Status: Fixed (Revised commit: 8fc01c)

Remediation: Following fixes were introduced:

1. The *cancel()* function was added. Using this function contract operators can cancel withdraw request for banned user. Open Eden will ban users that are blacklisted by USDC or other users that may block withdrawal queue.
2. KYC verification was added to the *redeem()* function, so it is not possible to add non-KYC user or zero address as withdrawal receiver.

It is important to acknowledge that these fixes offer a robust solution to prevent Denial of Service (DoS) in the event of a single withdrawal failure within the queue. However, it is crucial to emphasize that this solution exclusively applies to banned users. A new concern has arisen concerning the potential for queue lock caused by non-banned users (L06 issue in this report).

C02. Missing KYC Verification in *'Deposit()'* Function

Impact	High
Likelihood	High

The contract allows users to deposit tokens without verifying their KYC status, contrary to the specifications mentioned in the public documentation. The current implementation lacks this verification, which can lead to unintended consequences and potential loss of funds for non-KYC users.

The deposit function in the contract is designed to accept tokens from users. However, there is a missing check to verify if both user depositing tokens (msg_sender) and address receiving share (receiver) have been approved through the KYC process. This oversight means that even users without KYC approval can deposit tokens, which is a deviation from the intended behavior as described in the public documentation. The contract has other restrictions in place that prevent non-KYC approved users from transferring or withdrawing their tokens. If a non-KYC approved user deposits tokens, they will be unable to transfer or withdraw these tokens, effectively locking their funds permanently in the contract.

Impact:

1. **Regulatory Impact:** Allowing deposits without KYC can lead to regulatory issues, as KYC is a crucial step in preventing money laundering and ensuring compliance with financial regulations.
2. **Trust Issues:** Users and stakeholders might lose trust in the platform if it does not adhere to its own documented procedures.
3. **Potential for Exploitation:** Malicious actors might exploit this oversight to deposit illicit funds or engage in other fraudulent activities.

Proof of Concept:

1. The non-kyc user calls the deposit function of the OpenEdenVaultV2 contract to deposit a certain amount of *USDC* tokens.
2. The contract accepts the deposit and emits a ProcessDeposit event, even though the non-kyc user has not been approved through the KYC process.

3. The non-kyc user attempts to redeem a portion of their deposited tokens. The redemption attempt fails because the user has not passed the KYC verification, but their funds are already locked in the contract due to the successful deposit.
4. The non-kyc user's *USDC* funds are effectively locked in the *OpenEdenVaultV2* contract, and the *TBILL* token locked in their wallet, as they can neither transfer *TBILL* tokens nor redeem their *USDC* tokens.

Path: ./contracts/OpenEdenVaultV2.sol: deposit()

Recommendation: Modify the deposit function to include a check that verifies the KYC status of the *msg.sender* and *reciever*. If the user has not passed the KYC verification, the function should revert with an appropriate error message.

Found in: 4aed24d

Status: Fixed (Revised commit: 8fc01c)

Remediation: KYC validation was added to the *deposit()* function.

■■■ High

No high severity issues found.

■■ Medium

M01. Unrestricted Fee Configuration

Impact	High
Likelihood	Low

The system owner possesses the authority to set transaction fees for both weekends and weekdays to any value, including 100%. A fee rate of 100% implies that users will receive nothing when attempting to deposit or redeem tokens.

Moreover, charging fees for reedem operation does not happen in the redeem function immediately. The fee is taken when the operator process the withdrawal. If the fee is changed during this period of time, users will pay different amount of fees than promised.

This unrestricted control over fee configuration by the owner may lead to the imposition of excessive fees, resulting in users receiving no tokens for their transactions and potential dissatisfaction with the system. It introduces the risk of unfair and unjust fee charging and a potential loss of trust in the platform.

Path: ./contracts/feeManager.sol: setTransactionFeeWeekday(), setTransactionFeeWeekend()

Recommendation: Set reasonable boundaries for the transaction fees and mention these max and min (if there is) limits in the documentation. Additionally, calculate and apply the fee within the redeem function based on the current fee rate, rather than collecting the fee after a user places a redemption order.

Found in: 4aed24d

Status: Acknowledged

Comment: The Open Eden team decided to not fix this issue due to commercial reasons.

■ Low

L01. Missing Zero Address Validation

Impact	Low
Likelihood	Low

The zero address validation check is not implemented for the following functions:

1. *setOplTreasury()*
2. *setFeeManager()*
3. *setKycManager()*
4. *setOperator()*
5. *setUsdcPriceFeed()*
6. *setTBillPriceFeed()*
7. *setController()*
8. *setTreasury()*
9. *setQTreasury()*

Setting one of aforementioned parameters to zero address (0x0) results in breaking Open Eden main business flow.

Paths: ./contracts/OpenEdenVaultV2.sol : deposit(), redeem(), setOplTreasury(), setFeeManager(), setKycManager(), setOperator(), setUsdcPriceFeed(), setTBillPriceFeed(), setController(), setTreasury(), setQTreasury()

Recommendation: Implement zero address validation for the given parameters.

Found in: 4aed24d

Status: Fixed (Revised commit: 8fc01c)

Remediation: Zero address checks were implemented for aforementioned functions.

L02. Possibility of Dangerous Assumption on USDC Peg When Calculating TBillUsdcRate

Impact	Medium
Likelihood	Low

The contract's design assumes that the USDC stablecoin will always maintain a 1:1 peg with the US dollar when the *fixedPriceOn* flag is true. This assumption can lead to potential vulnerabilities and financial risks.

The *tbillUsdcRate()* function, which plays a crucial role in calculating shares and total assets during deposit and withdrawal operations, makes a potentially dangerous assumption when the *fixedPriceOn* boolean flag is set to true. It assumes that the USDC stablecoin will always be pegged to 1 USD. This assumption can introduce several risks and vulnerabilities.USDC, although designed to be pegged to the US dollar, might not always maintain this peg. Factors like market dynamics, liquidity issues, or unforeseen events can cause deviations from the 1:1 peg.

For example, in 10-13th March 2023, because of several bank collapses USDC was depegged for several days to values up to 0.88. Consider following scenario:

1. OpenEden vault contract has *fixedPriceOn* flag set to true.
2. USDC depegs to 0.9.
3. For simplicity let's assume that TBILL token exchange rate is 1.000000.
4. Users are depositing many USDC into contract.
5. OpenEden cannot easily offRamp USDC and buy expected amount of treasury bills as USDC is valuated less than USD. As a result, OpenEden operators cannot fulfill protocol promises and have to wait for USDC to stabilize.
6. However, if depeg event was unnoticed by protocol operators, then USD to fullfill protocol promises (exchanging USDC to TBills) have to be substituted from protocol USD reserves resulting in protocol profit decrease.

Path: ./contracts/OpenEdenVaultV2.sol: tbillUsdcRate(), filipFixedPrice()

Recommendation: Instead of relying on a fixed assumption, use trusted external oracles to fetch the current USDC price. This ensures that the contract always operates with the most accurate and up-to-date information.

Found in: 4aed24d

Status: Fixed (Revised commit: 8fc01c)

Remediation: The *fixedPriceOn* flag was removed. The USDC price is taken from the oracle.

L03. Usage of Deprecated Oracle Functions

Impact	Medium
Likelihood	Low

The codebase contains the usage of deprecated oracle function for getting USDC/USD price, the *latestAnswer*. It is crucial to update the codebase to use the latest recommended oracle function, *latestRoundData*, and ensure that the return data is handled correctly to avoid issues such as receiving stale or incorrect price data.

The *latestRoundData* function provides more comprehensive data, including not only the latest answer (price) but also additional information such as timestamps, answers from specific rounds, and more. Transitioning to this function is essential to ensure the accuracy and reliability of the price data obtained from the oracle.

Path: ./contracts/OpenEdenVaultV2.sol: tbillUsdcRate()

Recommendation: Use *latestRoundData* function to bring the USDC price and check all of its return values. Set a reasonable stale price delay time. A stale delay time of 1 to 5 minutes is commonly used in many applications. Ensure that the price that is returned is greater than zero.

Found in: 4aed24d

Status: Fixed (Revised commit: 8fc01c)

Remediation: The *onlyValidPrice* modifier was introduced, which checks price staleness against newly introduced *maxTimeDelay*, which is set during contract initialization and later could be modified by contract owner by *setMaxTimeDelay* function.

L04. Overpermisive offRamp Function

Impact	Medium
Likelihood	Low

The offRamp() function should be used to withdraw underlying asset (USDC) to treasury wallet.

It was observed that this function could be used to withdraw treasury share tokens as contract operator can specify an address of token that should be withdrawn. Withdrawing treasury share tokens can lead to Denial of Service (DoS) as such tokens act as users escrow (after redeem and before withdrawal queue processing). Changing escrow balance can lead to withdrawal queue processing error.

Path: ./contracts/OpenEdenVaultV2.sol: offRamp()

Recommendation: Restrict the *offRamp()* function so it can be only used to transfer underlying token (USDC).

Found in: 4aed24d

Status: Fixed (Revised commit: 8fc01c)

Remediation: The *offRamp()* is restricted to only be used to transfer underlying token (USDC).

L05. Missing Protection Against USDC Depeg

Impact	Medium
Likelihood	Low

The USDC depeg refers to a situation where the value of the USDC stablecoin loses its stability and is no longer effectively equivalent to one US dollar. In a depegged state, the USDC's value can fluctuate independently of the USD, and it may be worth more or less than one US dollar.

A USDC depeg event, often likened to a black swan event, carries the potential for significant adverse consequences on Open Eden, as outlined below:

1. Users depositing USDC to acquire share tokens may find themselves receiving fewer shares than anticipated due to slippage. This becomes especially precarious when the frontend application uses a hardcoded USDC price instead of reflecting real market dynamics.
2. The process for users to withdraw their underlying tokens involves a two-step procedure comprising redemption and withdrawal queue processing. Should a USDC depeg event occur during any stage of the withdrawal process, it could significantly alter the final amount of USDC received by the user.

Path: ./contracts/OpenEdenVaultV2.sol: deposit(), redeem(), processWithdrawalQueue()

Recommendation: It is strongly advisable to implement safeguards to mitigate the impact of potential USDC depeg events. This can be achieved through the following measures:

- Consider implementing a slippage mechanism that allows users to specify the precise quantity of tokens they intend to receive when making deposits. This empowers users to proactively manage their expectations and minimize the impact of sudden price fluctuations.
- An alternative approach is to temporarily suspend deposits and withdrawals in situations where the USDC/USD price experiences significant volatility. This precautionary step can help

safeguard the system's stability during periods of uncertainty and prevent unexpected losses for users.

Found in: 4aed24d

Status: Fixed (Revised commit: 8fc01c)

Remediation: The *maxDepeg* parameter was introduced, which validates (in *onlyValidPrice* modifier) if the USDC depeg is in expected range. The *maxDepeg* parameter can be set by contract owner using *setMaxDepeg()* function after the contract initialization.

L06. Missing Failover Mechanism to Unlock Withdrawal Queue Blocked by Non-Banned Users

Impact	Medium
Likelihood	Low

The *cancel()* function plays a critical role in managing withdrawal requests for banned users, specifically those who have been blacklisted by the internal mechanism of USDC. These requests must be canceled because attempting to transfer funds to a blacklisted account results in transaction reversal and queue processing blockage, preventing other users from withdrawing their funds.

While the *cancel()* function is effective in addressing some aspects of the queue blocking issue, it does not provide a comprehensive solution, as it is tailored exclusively to banned users. Several concerns persist regarding non-blocked users:

1. Denial of Service incidents may not always originate from banned users. Various scenarios, such as changes in Know Your Customer (KYC) status for the withdrawal receiver between the redemption and queue processing stages, can also lead to disruptions.
2. A separate issue pertains to the fees applied during the withdrawal processing. Users might still encounter discrepancies between the promised or agreed-upon fee amounts and the actual fees deducted. In exceptionally rare cases, if fees are altered between redemption and withdrawal queue processing, a user's withdrawal request could fail, causing queue blockages.

Path: ./contracts/OpenEdenVaultV2.sol: cancel()

Recommendation: Implementing a failover mechanism is strongly advised to address situations where a valid (non-banned) user's withdrawal request experiences a reversal, resulting in the blocking of the withdrawal queue.

Found in: 8fc01c

Status: Acknowledged

Comment: The Open Eden acknowledged this issue.

Informational

I01. Redundant Declaration

The boolean variables' default value is always false in Solidity. Therefore, declaring them as false in the constructor function is redundantly consuming Gas.

Path: ./contracts/security/OEPausable.sol: constructor()

Recommendation: Remove the boolean variables' declarations from the constructor.

Found in: 4aed24d

Status: Fixed (Revised commit: 8fc01c)

Remediation: Redundant declaration was removed.

I02. Solidity Style Guides Violation

Contract readability and code quality are influenced significantly by adherence to established style guidelines. In Solidity programming, there exist certain norms for code arrangement and ordering. These guidelines help to maintain a consistent structure across different contracts, libraries, or interfaces, making it easier for developers and auditors to understand and interact with the code.

The suggested order of elements within each contract, library, or interface is as follows:

- Type declarations
- State variables
- Events
- Modifiers
- Functions

Functions should be ordered and grouped by their visibility as follows:

- Constructor
- Receive function (if exists)
- Fallback function (if exists)
- External functions
- Public functions

- Internal functions
- Private functions

Within each grouping, view and pure functions should be placed at the end.

Furthermore, following the Solidity naming convention and adding NatSpec annotations for all crucial functionalities are strongly recommended. These measures aid in the comprehension of code and enhance overall code quality.

Path: ./*

Recommendation: Consistent adherence to the official Solidity style guide is recommended. This enhances readability and maintainability of the code, facilitating seamless interaction with the contracts. Providing comprehensive NatSpec annotations for functions and following Solidity's naming conventions further enrich the quality of the code.

Found in: 4aed24d

Status: Fixed (Revised commit: 8fc01c)

Remediation: Code was refactored to align with Solidity Style Guides.

I03. Commented Code Parts

Following commented code parts were observed:

1. *TBillPriceOracle* lines 6, 188-189, 195-201 (*console.log*)
2. *FeeManager* line 3.
3. *DoubleQueueModified* line 5 (SafeCast import).
4. *Controller* line 4.

The presence of commented-out code indicates an unfinished implementation, potentially causing confusion for both developers and users and decreasing code readability.

Path: ./contracts/feeManager.sol,

./contracts/oracle/TBillPriceOracle.sol,

./contracts/DoubleQueueModified.sol,

./contracts/Controller.sol

Recommendation: Remove commented parts of code.

Found in: 4aed24d

Status: Fixed (Revised commit: 8fc01c)

Remediation: Commented code was removed.

I04. State Variables Default Visibility

Visibility for following variables were not specified:

- *_transactionFeeWeekday*
- *_transactionFeeWeekend*
- *_firstDeposit*
- *_minDeposit*
- *_maxDeposit*
- *_minWithdraw*
- *_maxWithdraw*
- *_managementFeeRate*
- *_maxWeekendDepositPct*
- *_maxWeekendAggregatedDepositPct*
- *_minTxsFee*
- *firstDepositMap*
- *depositAmountMap*
- *withdrawAmountMap*
- *withdrawalInfo*
- *withdrawalQueue*

Specifying state variables visibility helps to catch incorrect assumptions about which inheriting contract can access the variable. Every variable with unspecified visibility is by default treated as internal.

Path: ./contracts/feeManager.sol,

./contracts/OpenEdenVaultV2.sol

Recommendation: To increase code readability, it is recommended to explicitly define visibility as public, internal, or private for all state variables.

Found in: 4aed24d

Status: Fixed (Revised commit: 8fc01c)

Remediation: Aforementioned variables are defined as private.

I05. Typo in the Function Name

The function *filipFixedPrice* has a typo in its name.

Path: ./contracts/OpenEdenVaultV2.sol: filipFixedPrice()

Recommendation: Change the name to *flipFixedPrice*.

Found in: 4aed24d

Status: Fixed (Revised commit: 8fc01c)

Remediation: The *filipFixedPrice()* function was removed.

I06. Usage of Toggle Switch Mechanism

The *flipFixedPrice* function incorporates a toggle-switch mechanism, which can pose a risk if inadvertently invoked several times and is not configured for the intended action.

Path: ./contracts/OpenEdenVaultV2.sol: filipFixedPrice()

Recommendation: Consider implementing a Boolean-control mechanism where *true* signifies the *fixed price* is enabled, and *false* indicates the opposite to enhance clarity and reduce the risk of accidental double invocation.

Found in: 4aed24d

Status: Fixed (Revised commit: 8fc01c)

Remediation: The *filipFixedPrice()* function was removed.

I07. Missing Event Indexes

Use indexed events to keep track of a smart contract's activity after it is deployed, which is helpful in reducing overall Gas.

Path: ./contracts/interfaces/IOpenEdenVault.sol

./contracts/oracle/TBillPriceOracle.sol

Recommendation: Add missed *indexed* keywords to easier tracking smart contract information.

Found in: 4aed24d

Status: Acknowledged

Comment: The Open Eden team is not indexing aforementioned events, so *indexed* is not necessary.

I08. Mismatch Between Contract Name and Filename

Consistency between contract names and their filenames is a recommended best practice in Solidity development. It ensures clarity and reduces the risk of errors. The *FeeManager* contract does not adhere to this guideline, which can lead to violation of official guides.

Path: ./contracts/feeManager.sol

Recommendation: Change the filename to *FeeManager.sol* to match the contract's name.

Found in: 4aed24d

Status: Acknowledged

Comment: The Open Eden acknowledged this issue.

I09. Use Custom Errors Instead Of Error Strings To Save Gas

Custom errors were introduced in Solidity version 0.8.4, and they offer several advantages over traditional error handling mechanisms:

1. Gas Efficiency: Custom errors can save approximately 50 Gas each time they are hit because they avoid the need to allocate and store revert strings. This efficiency can result in cost savings, especially when working with complex contracts and transactions.
2. Deployment Gas Savings: By not defining revert strings, deploying contracts becomes more gas-efficient. This can be particularly beneficial when deploying contracts to reduce deployment costs.
3. Versatility: Custom errors can be used both inside and outside of contracts, including interfaces and libraries. This flexibility allows for consistent error handling across different parts of the codebase, promoting code clarity and maintainability.

Path: ./contracts/*

Recommendation: To save Gas, it is recommended to use custom errors instead of strings.

Found in: 4aed24d

Status: Acknowledged

Comment: The Open Eden acknowledged this issue.

I10. Floating Pragma

The project uses floating pragmas ^0.8.0, ^0.8.4 and ^0.8.9.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version, which may include bugs that affect the system negatively.

Path: ./contracts/security/OEPausable.sol

./contracts/security/Timelock.sol

./contracts/Controller.sol

./contracts/DoubleQueueModified.sol

./contracts/feeManager.sol

./contracts/OpenEdenVaultV2.sol

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Found in: 4aed24d

Status: Fixed (Revised commit: 8fc01c)

Remediation: The pragma version was fixed to 0.8.9.

I11. Missing Overflow Check in Queue Implementation

The code contains a function called *pushBack*, which inserts an item at the end of the queue. However, there is a concern that there is no check for overflow. It's possible for the end index to overflow and exceed the begin index, although it is quite rare for this to happen and to reach to the max value of uint256 variable when the queue increases by only one.

Path: ./contracts/DoubleQueueModified.sol

Recommendation: Consider removing unchecked code blocks, so transaction reverts on overflow.

Found in: 4aed24d

Status: Acknowledged

Comment: The Open Eden acknowledged this issue.

I12. Checks-Effect-Interaction Pattern Violation

The Checks-Effects-Interactions pattern is violated inside following functions:

1. The *_processDeposit()*: During the function, the state variables are updated after the external token transfer call via *_deposit()*.
2. The *_processWithdraw()*: During the function, the state variables are updated after the token transfer call via *_addToWithdrawalQueue()*.
3. The *processWithdrawalQueue()*. During the function, the state variables are updated after the external token transfer call via *_withdraw()*.

This may lead to reentrancies, race conditions, and denial of service vulnerabilities during implementation of new functionality.

Path: ./contracts/OpenEdenVaultV2.sol : _processDeposit(), _processWithdraw(), processWithdrawalQueue()

Recommendation: Follow common best practices, and implement the functions according to the Checks-Effects-Interactions pattern. Modify the *_processDeposit(), _processWithdraw(), processWithdrawalQueue()* functions to update the state variable before making the external token transfer call.

Found in: 4aed24d

Status: Fixed (Revised commit: 8fc01c)

Remediation: The code was refactored to support Checks-Effect-Interaction pattern.

I13. Setting Insufficient Max Deviation of TBill Price Parameter Can Lead to Value Loss Due to Frontrunning

The Treasury Bills exchange rate is governed by the TBillPriceOracle contract, where the price parameter is subject to modification through the *updatePrice* function. While there is a built-in safeguard to prevent excessive deviations from the previous price (*_maxPriceDeviation*), it is important to note that this parameter can be dynamically adjusted both during contract initialization and through the *updateMaxPriceDeviation* function.

However, a potential security concern arises in scenarios where the Oracle authority permits substantial deviations from the previous price. In such cases, malicious actors can exploit this vulnerability by closely monitoring *updatePrice* transactions. When a significant price movement occurs, these malicious actors can swiftly execute front-running transactions, depositing USDC before the TBill price is updated. Subsequently, once the TBill price is modified, these actors can proceed to withdraw USDC, thereby gaining an unfair advantage and potentially compromising the integrity of the system.

Path: ./contracts/oracle/TBillPriceOracle.sol: constructor(), updateMaxPriceDeviation()

Recommendation: It is recommended to set *_maxPriceDeviation* with extra cautious or define deviation parameter upper limit.

Found in: 4aed24d

Status: Acknowledged

Comment: The Open Eden team is using internal procedure to verify *_maxPriceDeviation* parameter.

I14. Missing Validation for Setting Deposit and Withdrawal Limits

Ensuring that minimum values are less than or equal to their corresponding maximum values is crucial for the logical consistency of a contract. The current implementation does not enforce this rule, which can lead to unexpected results and hinder the contract's usability.

The contract allows the owner to set the *_minDeposit* value greater than *_maxDeposit* and *_minWithdraw* value greater than *_maxWithdraw*. There is no input validation to ensure that the minimum values are less than or equal to the corresponding maximum values. This oversight can lead to unexpected behavior and potential transaction reverts when users interact with the contract.

Path: ./contracts/feeManager.sol: setMinDeposit(), setMaxDeposit(), setMinWithdraw(), setMaxWithdraw()

Recommendation: Add checks in the functions that set deposit and withdrawal limits to ensure that minimum values are always less than or equal to the corresponding maximum values.

Found in: 4aed24d

Status: Fixed (Revised commit: 8fc01c)

Remediation: Maximum and minimum deposit values are validated minimum cannot be greater than maximum.

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.

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.

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/OpenEdenHQ/openeden.vault.v2.audit
Commit	4aed24dac07c442ad0fca131b4749d950465d5be
Whitepaper	
Requirements	https://docs.openeden.com/category/introduction
Technical Requirements	https://docs.openeden.com/category/introduction
Contracts	File: contracts/Controller.sol SHA3: 77d87bd87d71c4d12233ff6dcd917118b73f186b6982a512ae94af122cbf58f7 File: contracts/DoubleQueueModified.sol SHA3: eccafcca981a8b15948b6a70b987591798bcb64ad508bc2be9543c63cf6bf972 File: contracts/feeManager.sol SHA3: 9290ee062ead4d835dbf43daee76f39d7e96ba910d2d1a87a39f4216d15c7ac9 File: contracts/KycManager.sol SHA3: ca9af66472cd9ff5892c690a46c33da2784217f447c4781958cc32483958b43c File: contracts/OpenEdenVaultV2.sol SHA3: ef3e10fedc12e840210abb4e12942aaad80cfa1e91aee80604fd157e362ea6a8 File: contracts/interfaces/IFeeManager.sol SHA3: 4904db7caecad5f9277e127554d83ff57ce0e141cc204573ee128007c12c108a File: contracts/interfaces/IKycManager.sol SHA3: a906c0e50bf5429d255e85ef6719bd2c9482a99473a2425eb9af009950d7a93f File: contracts/interfaces/IOpenEdenVault.sol SHA3: e133ea0f9a41ea185a084648170e08cdabfc4373bd58416890c9cf8bdb8ce6f7 File: contracts/interfaces/IPriceFeed.sol SHA3: 03d9593c94c35b0f0a6806ab0988abb9775215e94f9fed072fb42daa931c0ef2 File: contracts/oracle/TBillPriceOracle.sol SHA3: 4062c6868e92cd2d52220f9ce8211aaa77087bf1bb811c6407d0de6734389879 File: contracts/security/OEPausable.sol SHA3: ff72063dc549a1eb9e8941de037efea2d183da27efece0ddb77bccc16a800fa1 File: contracts/security/Timelock.sol SHA3: e27795e7af78e18e4646c0db800f03297e181196599f66d2acd3b8f878bd9c37