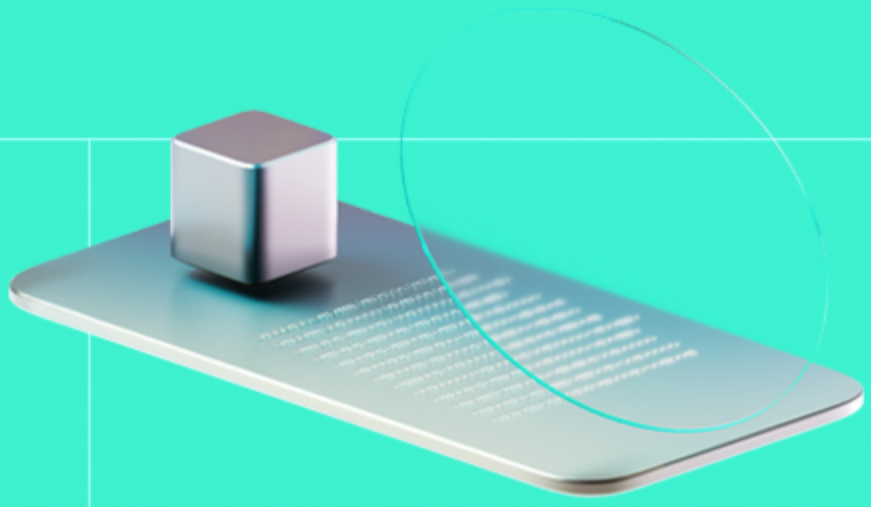
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: OpenEden

Date: 10/12/2024

We express our gratitude to the OpenEden team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

OpenEden offers 24/7, on-chain access to tokenized US Treasury securities for Web3 CFOs, DAO treasury managers, and buy-side institutional investors.

Document

Name	Smart Contract Code Review and Security Analysis Report for OpenEden
Audited By	Stepan Chekhovskoi
Approved By	Przemyslaw Swiatowiec
Website	https://openeden.com
Changelog	29/11/2024 - Preliminary Report
	10/12/2024 - Final Report
Platform	Ethereum, Arbitrum, Base
Language	Solidity
Tags	DeFI, Vault
Methodology	https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/OpenEdenHQ/openeden.vault.v2.audit
Initial Commit	980995254706f34187440f9755c3507b6f647588
Final Commit	1299050d098a626fffa2a652545ee40abb9f1d7a

Audit Summary

The system users should acknowledge all the risks summed up in the risks section of the report

3	1	1	1
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	0
High	0
Medium	2
Low	1

Vulnerability	Severity
F-2024-7411 - Potential Redeem Process DoS due to KYC User Ban	Medium
F-2024-7422 - Possible Invalid Convert Rate due to Lack of Oracle Output Validation	Medium
F-2024-7417 - Unchecked Integers Casting	Low

Documentation quality

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

Code quality

- The code structure is clear.

Test coverage

Code coverage of the project is **70%** (branch coverage).

- Tests cover most of the functionality partially missing negative cases.

Table of Contents

System Overview

The TBILL protocol is a decentralized framework designed to enable secure and efficient subscription and redemption of tokenized Treasury Bills (TBILLs). At its core, the architecture integrates smart contracts to manage user interactions and enforce critical operations such as KYC verification, fee management, and price updates.

The audit covers the `OpenEdenVault` contract upgrade to the version 4. The following features are implemented:

- Allow cancelling out the txnFee for USDO mints for certain users.
- Cancelled redemption requests are returned to sender instead of quarantined wallet.
- Allow the ability to manually bypass firstDeposit rule for applicable wallet.
- Allow setting of multiple operators.
- Supporting instant redeem by using BUIDL.

Privileged roles

- The `PartnerShip` contract Owner is able to set arbitrary partnership relations.
- The `OpenEdenVaultV4Impl` contract Owner is able to set the contract Maintainer and configure treasury, fee manager, KYC manager contract addresses.
- The `OpenEdenVaultV4Impl` contract Maintainer is able to set the contract Operators, configure partnership, buidl contract addresses, cancel user redemption requests.
- The `OpenEdenVaultV4Impl` contract Operators are able to withdraw any token from the contract to treasury and update epochs.

Potential Risks

- **Single Points of Failure and Control**: The project is mostly centralized, introducing single points of failure and control. This centralization can lead to vulnerabilities in decision-making and operational processes, making the system more susceptible to targeted attacks or manipulation.
- **System Reliance on External Contracts**: The functioning of the system relies on fee manager, buidl redemption, KYC manager external contracts. Any flaws or vulnerabilities in these contracts adversely affect the audited project, potentially leading to security breaches or loss of funds.
- **Potentially Insufficient Funds**: The contract does not guarantee that there would be available enough underlying funds for redemption. User funds are transferred to Treasury contracts out of the audit scope.
- **External Calls in Iteration Risks**: Making external calls within loops increases the risk of gas exhaustion, potentially leading to failed transactions and reduced contract reliability, especially when processing large datasets.
- **Reentrancy Assumptions**: The project relies on the underlying (USDC) token does not delegate execution control to third parties. In case the token is changed or upgraded, Operators may break the contract state consistency reentering the `processWithdrawalQueue` function.
- **Deprecated Safe Approve**: The project utilizes deprecated `safeApprove` method of `SafeERC20` library for buidl token approval. The contract ensures the redemption process consumes the full allowance reducing the approve DoS risk due to non-zero allowance.

Findings

Vulnerability Details

F-2024-7411 - Potential Redeem Process DoS due to KYC User Ban - Medium

Description:

The `processWithdrawalQueue` function of the `OpenEdenVaultV4` contract validates if each user in queue is KYC authorized.

```
for (uint count = 0; count < _len; ) {
    bytes memory data = withdrawalQueue.front();
    (address sender, address receiver, uint256 shares, bytes32 prevId) = _dec
odeData(data);
    require(_validateKyc(sender, receiver), "invalid kyc");
    ...
}
```

The function processes users one-by-one without the skip ability. This may lead to the function DoS in case any user in the queue is not KYC authorized.

To be included to the queue the user need to execute `redeem` function which validates the user is KYC authorized. However, the KYC status is processed by another contract and might be changed to banned in between `redeem` and `processWithdrawalQueue` function calls.

```
function redeem(... address _receiver) ... {
    address sender = _msgSender();
    require(_validateKyc(sender, _receiver), "invalid kyc");
    ...
}

function _validateKyc(address _from, address _to) ... {
    return
        kycManager.isKyc(_from) &&
        kycManager.isKyc(_to) &&
        !kycManager.isBanned(_from) &&
        !kycManager.isBanned(_to);
}
```

The contract maintainer is able to execute `cancel` function which cancels first request in the queue unblocking the `processWithdrawalQueue` function.

Assets:

- OpenEdenVaultV4Impl.sol [https://github.com/OpenEdenHQ/openeden.vault.v2.audit/]

Status: Mitigated

Classification

Impact: 3/5

Likelihood: 3/5

Exploitability: Independent

Complexity: Simple

Severity: Medium

Recommendations

Remediation: Consider automatically cancel requests from the banned users in the `processWithdrawalQueue` function rather than blocking the function execution.

```
uint256 totalCancelShares;
...
for (uint count = 0; count < _len; ) {
    bytes memory data = withdrawalQueue.front();
    (address sender, address receiver, uint256 shares, bytes32 prevId) = _dec
odeData(data);
    if (!_validateKyc(sender, receiver)) {
        withdrawalQueue.popFront();
        unchecked {
            withdrawalInfo[receiver] -= shares;
            totalCancelShares += shares
            ++count;
        }
        _transfer(address(this), sender, shares);
        emit ProcessRedeemCancel(sender, receiver, shares, prevId);
        continue;
    };
    ...
}
...
emit ProcessWithdrawalQueue(
    totalWithdrawAssets,
    totalBurnShares,
    totalCancelShares,
```

```
    totalFees
);
```

Resolution: The Finding is mitigated according to the notice.

> The KYC service is part of the project and is managed by the same owner. The owner is able to order transactions in appropriate way to avoid the DoS state. Transfers for banned users are not allowed.

The malicious user may poison the queue with withdraw requests to temporarily avoid ban, however, this is ineffective due to specified `minWithdraw` amount.

F-2024-7422 - Possible Invalid Convert Rate due to Lack of Oracle Output Validation - Medium

Description:

The `tbillUsdcRate` function of the `OpenEdenVaultV4` contract integrates Oracle for price fetch. However, it lacks required price validations.

```
function tbillUsdcRate() ... {
    uint256 tbillUsdPrice = tbillUsdPriceFeed.latestAnswer();
    rate = (tbillUsdPrice * tbillDecimalScaleFactor) / ONE;
}
```

Due to maintenance stop oracle data might become expired. The returned price might be zero indicating Oracle invalid status.

This may lead to the conversions happen at expired or invalid rate.

Assets:

- OpenEdenVaultV4Impl.sol [https://github.com/OpenEdenHQ/openeden.vault.v2.audit/]

Status: Fixed

Classification

Impact: 3/5

Likelihood: 3/5

Exploitability: Independent

Complexity: Simple

Severity: Medium

Recommendations

Remediation:

Consider using the `latestRoundData` function, validate the `updatedAt` and `price` parameters.

```
(, int256 answer, , uint256 updatedAt, ) = tbillUsdPriceFeed.latestRoundData(
);
require(answer > 0 ...);
require(block.timestamp - updatedAt < heartbeat ...);
uint256 tbillUsdPrice = uint256(answer);
```

Resolution: The Finding is fixed in the commits `60039ad620f4e6eea6ccd5a95349b42c53b63af7` and `1299050d098a626fffa2a652545ee40abb9f1d7a`.

The necessary validations are implemented.

F-2024-7417 - Unchecked Integers Casting - Low

Description:

The `txsFee` function performs unchecked integer casting which may cause integer overflows.

```
function txsFee(... uint256 _assets) ... {
    ...
    if (...) {
        ...
        pFee = (int256(_assets) * pfeePct) / int256(BPSUNIT);
    }
    ...
}
```

Passing `_assets` parameter over `2^256 / 2 - 1` causes integer overflow and the function returns unexpected result.

While the situation is unlikely to impact the contract operation as such assets amount is not reasonable, the function does not handle the edge case and returns corrupted value confusing off-chain users.

Assets:

- OpenEdenVaultV4Impl.sol [https://github.com/OpenEdenHQ/openeden.vault.v2.audit/]

Status: Accepted

Classification

Impact: 3/5

Likelihood: 2/5

Exploitability: Independent

Complexity: Simple

Severity: Low

Recommendations

Remediation: Consider validating the `_assets` parameter is in expected range.

```
require(_assets < 2**255, ...);
```

Resolution: The Finding is accepted by the Client team with the notice.

The described flaw does not impact real use cases, the current implementation is more Gas efficient.

Observation Details

F-2024-7415 - Is Now Weekend Check Automation - Info

Description:

The `OpenEdenVaultV4` contract implements `isWeekend` state flag which is used for fee calculation. The flag is set manually by the contract operator.

```
bool public isWeekend;
function setWeekendFlag(bool _isWeekend) external onlyOperator {
    isWeekend = _isWeekend;
    emit SetWeekendFlag(_isWeekend);
}
```

The `block.timestamp` can be used to get the week day number. The `block.timestamp / 60 / 60 / 24 % 7` expression returns weekday where 2 is Saturday and 3 is Sunday.

The automation allows to safe some Gas in runtime.

Status: Mitigated

Recommendations

Remediation:

Consider automatically checking if now is weekend in the `txsFee` function.

```
uint256 daynum = block.timestamp / 3600 / 24 % 7;
uint256 isWeekend = daynum == 2 || daynum == 3;
```

Resolution:

The Finding is mitigated according to the notice.

> The flag is used to identify weekends and holidays. This logic cannot be automated.

F-2024-7416 - Revert Messages Optimization - Info

Description:

Using revert string messages increases the size of a smart contract and its deployment costs.

Since Solidity version `0.8.4` custom errors are available. Custom Errors provide a way for developers to define descriptive and semantically meaningful error conditions without relying on string messages.

The revert strings are used in the `PartnerShip`, `DoubleQueueModified`, and `OpenEdenVaultV4Impl` contracts.

Assets:

- OpenEdenVaultV4Impl.sol [https://github.com/OpenEdenHQ/openeden.vault.v2.audit/]
- PartnerShip.sol [https://github.com/OpenEdenHQ/openeden.vault.v2.audit/]
- DoubleQueueModified.sol [https://github.com/OpenEdenHQ/openeden.vault.v2.audit/]

Status: Fixed

Recommendations

Remediation:

To reduce gas costs during contract deployment, consider using custom errors instead of revert strings.

Solidity compiler supports the use of Custom Errors within require statements since version `0.8.26` for the IR pipeline and since version `0.8.27` for the legacy pipeline. For the older Solidity versions, the pattern `if (!condition) revert CustomError()` should be used.

Resolution:

The Finding is fixed in the commit `6bcbbcb79aaf1532f651cc6746433533451882df`.

The checks are reworked to employ Custom Errors.

F-2024-7429 - Lack Of Two-Step Ownership Transfer Mechanism - Info

Description: The `PartnerShip` contract follows the single-step ownership transfer model.

This allows accidental ownership transfers to arbitrary invalid addresses, resulting in permanent ownership loss.

Assets:

- PartnerShip.sol [https://github.com/OpenEdenHQ/openeden.vault.v2.audit/]

Status: Mitigated

Recommendations

Remediation: Consider changing the ownership management model to a two-step process by replacing the `Ownable` dependency with `Ownable2Step`.

Resolution: The Finding is mitigated according to the notice.

> The contract is owned by multi-signature wallet what lowers the described risk.

F-2024-7439 - Lack of Fees Transfer to Treasury - Info

Description:

The `processWithdrawalQueue` function of the `OpenEdenVaultV4` processes withdraw requests sequentially taking fee from each request.

Due to typo in if-condition (`totalFees > 0` instead of `totalFee > 0`), there is a possibility that the withdrawal fees are not transferred to the treasury (`totalFees == 0 && totalFee > 0` is possible as `totalFees` is updated after the condition check).

```
uint256 totalFees;

for (uint count = 0; count < _len; ) {
    ...
    (uint256 oeFee, int256 pFee, uint256 totalFee) = txsFee(
        ActionType.REDEEM,
        sender,
        assets
    );

    if (totalFees > 0) {
        SafeERC20Upgradeable.safeTransfer(
            IERC20Upgradeable(underlying),
            oplTreasury,
            totalFee
        );
    }

    ...
    unchecked {
        ...
        totalFees += totalFee;
    }
    ...
}
```

The problem appears in case only one withdrawal request is passed to the `processWithdrawalQueue` function or there is a chain of no-fee requests and the last one is not fee exempted.

This may lead to the withdrawal fee is not transferred to the treasury. However, the system provides centralized ability to transfer any stuck funds from the contract to the treasury using the `offRamp` function.

Assets:

- OpenEdenVaultV4Impl.sol [https://github.com/OpenEdenHQ/openeden.vault.v2.audit/]

Status: Fixed

Recommendations

Remediation: Consider fixing typo in the if-condition.

```
if (totalFee > 0) {
    SafeERC20Upgradeable.safeTransfer(
        IERC20Upgradeable(underlying),
        oplTreasury,
        totalFee
    );
}
```

Resolution: The Finding is fixed in the commit `3f1db9f7bda5170d9a8cd5ac8c248b122b340c9c`.

The `totalFee` variable is renamed to `fee` to avoid confusion.

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. Definitions

Severities

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
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 will not have a significant impact on code execution.

Potential Risks

The "Potential Risks" section identifies issues that are not direct security vulnerabilities but could still affect the project’s performance, reliability, or user trust. These risks arise from design choices, architectural decisions, or operational practices that, while not immediately exploitable, may lead to problems under certain conditions. Additionally, potential risks can impact the quality of the audit itself, as they may involve external factors or components beyond the scope of the audit, leading to incomplete assessments or oversight of key areas. This section aims to provide a broader perspective on factors that could affect the project's long-term security, functionality, and the comprehensiveness of the audit findings.

Appendix 2. Scope

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

Scope Details	
Repository	https://github.com/OpenEdenHQ/openeden.vault.v2.audit
Initial Commit	980995254706f34187440f9755c3507b6f647588
Final Commit	1299050d098a626fffa2a652545ee40abb9f1d7a
Whitepaper	N/A
Requirements	https://docs.openeden.com
Technical Requirements	README.md

Asset	Type
DoubleQueueModified.sol [https://github.com/OpenEdenHQ/openeden.vault.v2.audit/]	Smart Contract
interfaces/IBuidlRedemption.sol [https://github.com/OpenEdenHQ/openeden.vault.v2.audit/]	Smart Contract
interfaces/IController.sol [https://github.com/OpenEdenHQ/openeden.vault.v2.audit/]	Smart Contract
interfaces/IFeeManagerV3.sol [https://github.com/OpenEdenHQ/openeden.vault.v2.audit/]	Smart Contract
interfaces/IKycManager.sol [https://github.com/OpenEdenHQ/openeden.vault.v2.audit/]	Smart Contract
interfaces/IOpenEdenVaultV4.sol [https://github.com/OpenEdenHQ/openeden.vault.v2.audit/]	Smart Contract
interfaces/IPartnerShipV4.sol [https://github.com/OpenEdenHQ/openeden.vault.v2.audit/]	Smart Contract
interfaces/IPriceFeed.sol [https://github.com/OpenEdenHQ/openeden.vault.v2.audit/]	Smart Contract
interfaces/ITypes.sol [https://github.com/OpenEdenHQ/openeden.vault.v2.audit/]	Smart Contract
OpenEdenVaultV4Impl.sol [https://github.com/OpenEdenHQ/openeden.vault.v2.audit/]	Smart Contract
PartnerShip.sol [https://github.com/OpenEdenHQ/openeden.vault.v2.audit/]	Smart Contract

Appendix 3. Additional Valuables

Verification of System Invariants

During the audit of OpenEden, Hacken followed its methodology by performing fuzz-testing on the project's main functions. Echidna, a tool used for fuzz-testing, was employed to check how the protocol behaves under various inputs. Due to the complex and dynamic interactions within the protocol, unexpected edge cases might arise. Therefore, it was important to use fuzz-testing to ensure that several system invariants hold true in all situations.

Fuzz-testing allows the input of many random data points into the system, helping to identify issues that regular testing might miss. A specific Echidna fuzzing suite was prepared for this task, and throughout the assessment, 1 invariant was tested over 500K runs. This thorough testing ensured that the system works correctly even with unexpected or unusual inputs.

Invariant	Test Result	Run Count
DoubleQueueModified library keeps queue start index below the end index	Passed	500K+

Additional Recommendations

The smart contracts in the scope of this audit could benefit from the introduction of automatic emergency actions for critical activities, such as unauthorized operations like ownership changes or proxy upgrades, as well as unexpected fund manipulations, including large withdrawals or minting events. Adding such mechanisms would enable the protocol to react automatically to unusual activity, ensuring that the contract remains secure and functions as intended.

To improve functionality, these emergency actions could be designed to trigger under specific conditions, such as:

- Detecting changes to ownership or critical permissions.
- Monitoring large or unexpected transactions and minting events.
- Pausing operations when irregularities are identified.

These enhancements would provide an added layer of security, making the contract more robust and better equipped to handle unexpected situations while maintaining smooth operations.