PUBLIC

Code Assessment

of the USDOExpress Smart Contracts

Mar 06, 2025

Produced for

OpenEden

by

CHAINSECURITY

Contents

1 Executive Summary 3
2 Assessment Overview 5
3 Limitations and use of report 7
4 Terminology 8
5 Findings 9
6 Resolved Findings 11
7 Informational 14
8 Notes 15

1 Executive Summary

Dear OpenEden team,

Thank you for trusting us to help OpenEden Labs with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of USDOExpress according to Scope to support you in forming an opinion on their security risks.

`USDOExpress` adds instant mint and redeem functionality to the existing USDO token.

We did not find severe issues. However, multiple minor issues related to fees were raised (see Missing Slippage Protection, Fee Can Be Avoided on Small Amounts and Repeated Fees) and the deposit limits are ineffective as described in Ineffective First Deposit Limit. All issues where addressed.

In summary, we find that the codebase provides a high level of security. Yet, it is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	1
• Specification Changed	1
Low-Severity Findings	5
• Code Corrected	1
• Specification Changed	1
• Acknowledged	3

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the USDOExpress repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	08 January 2025	ceaf7861a1bf7bae4c011ee00cefe2b3826eefc0	Initial Version
2	05 February 2025	f5eb61df682661cc71563afd3c14c2e30a188758	Fixes
3	05 March 2025	80cf2cd66356adc8c0f5c83632dc4ca80a7cd6e9	Version 2

For the solidity smart contracts, the compiler version `0.8.18` was chosen.

The following files were in scope:

- `contracts/extensions/USDOExpress.sol`
- `contracts/extensions/USDOExpressPausable.sol`
- `contracts/extensions/USDOMintRedeemLimiter.sol`

2.1.1 Excluded from scope

Other files are not in scope. In particular, USDO itself is not in scope. Third-party systems such as BUIDL and TBILL are not in scope. Libraries such as OpenZeppelin are not in scope.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

The reviewed `USDOExpress` contract allows whitelisted users to instantly mint 1:1 USDO for either USDC or TBILL tokens. Before, USDO could only be minted by an admin, after users performed a relatively time-consuming fiat deposit process. The contract also adds three redemption options. All operations might be subject to a fee, to certain limit, and to the presence of reserves.

To instantly mint USDO, the whitelisted user needs to call `instantMint()`. Additionally, the user needs to specify the underlying (either USDC or TBILL tokens), the amount to exchange for USDO and a destination address. Caller and destination address need to be whitelisted. There are minimum deposit limits for the first deposit and for consecutive deposits. In case the mint request does not exceed the system's limits, the underlying is transferred to the treasury contract and the USDO contract's mint function triggered. We assume that the `_firstDepositAmount` is high enough to mitigate inflation attacks. The `USDOExpress` uses the `USDO` mint function and, hence, needs to have the `MINTER_ROLE`

in the USDO contract. The USDO contract acts similar to a vault in will calculate the depositor's share and mint the USDO to the specified `to` address.

A user has three different ways to redeem his USDO for USDC. All approaches will charge a fee. The caller and the receiver of the USDC needs to be whitelisted in all cases, too. The most basic approach is to call `redeem()`. It will transfer the USDO to the treasury contract and the user needs to wait until the admin manually burns the USDO and transfers the corresponding USDC minus fees to the user. There are two new functions to instantly redeem without the manual review process. Both will first burn the USDO. The difference is that `instantRedeem()` will use TBILL in the redemption process and `instantRedeemSelf()` will use BUIDL in the redemption process. `instantRedeem()` will pull the corresponding amount of TBILL tokens from the treasury contract and use it to call `redeemIns()` on the TBILL contract. Consequently, receiving USDC from the TBILL redemption. The USDC is finally send to the user and fee receiver. The last redemption option is `instantRedeemSelf()`. It will convert the USDO amount to USDC and pull this amount from the BUIDL treasury and call `redeem()` on `_buidlRedemption`. Finally, the USDC is sent to the user and fee receiver, too.

The remaining functions are view functions (e.g., to convert tokens) as well as setters and getters. The contract is pausable and can separately pause mints and redeems. The contract is also upgradable by the `UPGRADE_ROLE` role through OpenZeppelin's UUPS upgrade pattern.

The following roles defined by the contract:

DEFAULT_ADMIN_ROLE: Ultimately trusted as the role can grant and revoke other roles arbitrarily and is allowed to set all critical parameters on the USDOExpress contracts, e.g. fees and limits. It is also able to change the APY on USDO and abuse the minting permissions.

UPGRADE_ROLE: Ultimately trusted as this role can upgrade the contract implementation, making it as powerful as `DEFAULT_ADMIN_ROLE`.

PAUSE_ROLE: Fully trusted as the role can pause and unpause minting and redeeming and block the whole system.

MULTIPLIER_ROLE: Partially trusted to call and update the bonus multiplier in time. See `addBonusMultiplier()`.

WHITELIST_ROLE: Untrusted. This role is used for legal compliance purposes to whitelist users that are allowed to use the instant redeem and mint functionalities provided by the contract.

We want to highlight the following assumptions:

- The `USDOExpress` contract does not hold any relevant tokens.
- No rounding issues or similar will happen in `_buidlRedemption.redeem` such that it will not transfer exactly the `requestAmt` amount.
- We assume that the patched UUPS OpenZeppelin upgrade scheme is used that uses the `onlyProxy` modifier with `upgradeTo()` is used or the implementation contract is initialized accordingly.
- We assume that fees and cost for a minting operation always exceed the profit that potentially can be made through an instant mint and redeem operation as well as through any sandwich attacks on multiplier updates.

Version 2 Changes:

- A KYC whitelist is used to manage access to the redeem and instant mint functionalities. This whitelist is not based on the OZ role access control but on a simple `_kycList` mapping.
- The `redeem` function now burns the USDO, instead of transferring it to treasury address.
- A function to update the `buidlTreasury` address was added.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	3

- Fee Can Be Avoided on Small Amounts Acknowledged
- Ineffective First Deposit Limit Acknowledged
- instantRedeem Can Transfer One Wei Less Than Requested Acknowledged

5.1 Fee Can Be Avoided on Small Amounts

Correctness Low Version 1 Acknowledged

CS-USDOEX-002

The internal function `txsFee()` rounds down when calculating the fee. Given that it is applied to amounts in USDC or TBILL, small amounts can be selected such that the fee is rounded down to zero. For 5 basis points, the optimal amount would be 1999 wei or 0.1999 cents. By breaking up a large sum in chunks of this size, the mint and redeem fees can effectively be circumvented. However, gas fees would most likely make this unprofitable.

Acknowledged

OpenEden Labs acknowledges the issue.

5.2 Ineffective First Deposit Limit

Design Low Version 1 Acknowledged

CS-USDOEX-004

A user that deposits the first time USDC via calling `instantMint()` needs to at least provide `_firstDepositAmount` USDC tokens. However, a user with idle capital can always mint and immediately redeem their deposit in whole or in part. Subsequent deposits can also bypass `_mintMinimum` in a similar way. As a result, there would effectively be no deposit minima as long as redeem liquidity is available. Additionally, users can leverage flash loans if those are available.

Acknowledged:

OpenEden Labs noted the issue.

5.3 instantRedeem Can Transfer One Wei Less Than Requested

Correctness Low Version 1 Acknowledged

CS-USDOEX-006

Due to the conversion to and from TBILL, the `instantRedeem()` function can truncate the transaction amount in an unexpected way, potentially causing confusion for users and integrators.

Assume the redemption fee is zero and the TBILL to USDC rate is 1.088307. A user calls `instantRedeem()` for 1 USDO (10^{18} wei). The function converts the amount to 1 USDC (10^{6} wei) then to 0.918858 TBILL for which the TBILL vault will only give 0.999999 USDC. The caller could reasonably have expected to receive one whole token.

As the TBILL conversion rate grows above 2-to-1, the error can be up to 2 wei, and will grow exponentially.

Acknowledged

OpenEden Labs is aware of the issue and acknowledges it. Furthermore, OpenEden Labs replied that the rate won't grow over 2-1 in reality anyway.

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	1

- Missing Slippage Protection Specification Changed

Low-Severity Findings	2

- Incomplete or Incorrect Natspec Code Corrected
- Repeated Fees Specification Changed

Informational Findings	3

- Code Fragmentation Code Corrected
- Non-indexed Events Code Corrected
- Unused Event Code Corrected

6.1 Missing Slippage Protection

Security Medium Version 1 Specification Changed

CS-USDOEX-001

When redeeming via `instantRedeem()` a user specifies an USDO amount that shall be redeemed. This amount will be burned and converted into a `tbillAmt` which will subtract the fees specified by the protocol. When calling `redeemIns()` on the TBILL contract, additional fees will be charged and calculations performed.

Even though the user can call `previewRedeem()` to estimate the redemption including the protocol fees, it is hard for the user to control third-party fees that are deducted.

The user cannot opt-out in case the fees or conversion rate is unacceptable.

Specification changed:

OpenEden Labs states that the fee will be set to 0 on the tbill side, specifically for requests coming from USDOExpress.

6.2 Incomplete or Incorrect Natspec

Correctness Low Version 1 Code Corrected

CS-USDOEX-003

The following functions have incomplete or missing natspec:

- Many functions in `USDOExpress` either have incomplete or missing natspec
- `USDOMintRedeemLimiter._setTotalSupplyCap`
- `USDOMintRedeemLimiter.__USDOMintRedeemLimiter_init`

In the NatSpec for `instantMint()`, the `amt` parameter is described as "[t]he requested amount of USDO to mint". However, in the actual implementation, it corresponds to the amount of underlying token supplied by the caller.

Code corrected:

The natspec listed above was corrected or added.

6.3 Repeated Fees

Design | Low | Version 1 | Specification Changed

CS-USDOEX-005

`instantRedeem()` charges two different fees. First in `previewRedeem()` and another time third-party fees in `redeemIns`. Besides inconveniences for the user as described in Missing slippage protection, there is no guarantee that in `requestAmt - feeInUsdc` the invariant `requestAmt >= feeInUsdc` holds. As both amounts are calculated in different stages and different amounts with different fees. Additionally, a rational user will never use `instantRedeem()` if fees are charged in `redeemIns`, as it will be more expensive than calling `instantRedeemSelf()`. This might result in an accumulation of tbills in the treasury as users will prefer `instantRedeemSelf()` and withdraw BUIDL form the treasury.

Specification changed:

OpenEden Labs has clarified that the third-party fee in `TBILL.redeemIns` will be waved. (see Missing slippage protection) OpenEden Labs also states that `instantRedeemSelf()` should be the user's first choice, and that `instantRedeem()` should only be used if there isn't enough BUIDL liquidity. This would be reflected in the user interface.

6.4 Code Fragmentation

Informational | Version 1 | Code Corrected

CS-USDOEX-007

The following three errors are defined in `USDOMintRedeemLimiter` but only used in `USDOExpress`:

- `MintLessThanMinimum`
- `TotalSupplyCapExceeded`
- `FirstDepositLessThanRequired`

Code corrected:

The error definitions have been moved to `USDOExpress`.

6.5 Non-indexed Events

Informational | Version 1 | Code Corrected

CS-USDOEX-009

Events can be indexed to make filtering for certain addresses or parameters easier. None of the events defined in `USDOExpress` and `USDOExpressPausable` have indexed fields.

Code corrected:

The following parameters are now indexed:

- `underlying`, `from` and `to` in `InstantMint`,
- `from` and `to` in `InstantRedeem`,
- `from` and `to` in `ManualRedeem` and
- `account` in `UpdateFirstDeposit`.

6.6 Unused Event

Informational | Version 1 | Code Corrected

CS-USDOEX-012

The event `UpdateMintRedeemLimiter` is not used and there is no state variable for the Limiter because it is inherited by the `USDOExpress` contract. This might be an artefact from past versions.

Code corrected:

The event was removed.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Missing Sanity Check

Informational **Version 1**

CS-USDOEX-008

Some setters are missing sanity checks. As all setters have access control, the caller needs to make sure that no mistakes are done. Some sanity checks might still make sense to implement on smart contract level. E.g., tx fees should not exceed `_BPS_BASE`.

7.2 Unconventional Naming

Informational **Version 1**

CS-USDOEX-011

Public fields of `USDOExpress`, namely `_apy`, `_mintFeeRate`, `_redeemFeeRate`, `_increment`, `_lastUpdateTS`, `_timeBuffer`, `_usdo`, `_usdc`, `_tbill`, `_treasury`, `_feeTo`, `_buidl`, `_buidlRedemption`, `_buidlTreasury`, and `_firstDeposit`, start with an underscore. It is a common practice in Solidity development to prefix private fields with an underscore, but not public ones. While there is no difference in terms of functionality, this could confuse users and developers interacting with the contracts through its ABI.

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 Bonus Multiplier Increments Do Not Compound

Note Version 1

The system sets the bonus multiplier increment proportionally to the APY advertised in `updateAPY()`, in such a way that users receive the same amount of interest each day of the year. If a user deposits on the first day and redeems on the 365th day, their gains will reflect the advertised APY. However, since the rewards distribution does not follow an exponential curve, the effective daily APY will be magnified on the first day and decrease on every subsequent day. In particular, a user who deposits only during the first half of the year will receive more than one that deposits on the second half. Furthermore, the effective APY for the second year will be less than the advertised APY.

More specifically, given an APY value `apy`, with `BASE = 1e18` and `BPS_BASE = 1e4` the increment will be `increment = BASE * apy / BPS_BASE / 365`. With a first-day principal of `amount`, the daily yield will be `yield = amount * increment / BASE`, so the balance on day `d` will be `balance(d) = amount + d * yield`. Since balance increases with `d`, the ratio `yield / balance(d) = increment / (base + d * increment)`, which is proportional to the daily APY, decreases with d and tends towards 0 at infinity.

With `apy = 400` and `amount = 10000`, a user who deposits on day 0 and withdraws on day 182 will earn 199.45 USDO, whereas a user who deposits on day 183 and withdraws on day 365 will earn 195.53 USDO.