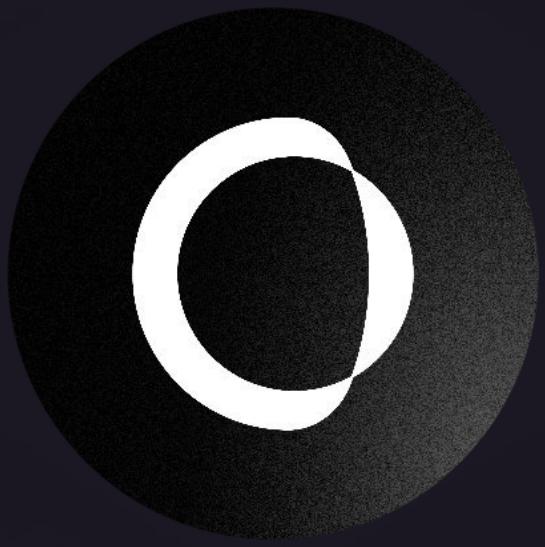
SHERLOCK

Security Review For Usual Labs



Collaborative Audit Prepared For: **Usual Labs**
Lead Security Expert(s): **eeyore**
nirohgo
xiaoming90
Date Audited: **February 7 - February 12, 2025**

Introduction

Auditing the Hook Contract and simplified EVK deployment script from EulerV2, focusing on our custom implementation and the debt-token hook integration with Usual Protocol's RWA-Collateralization logic. The audit will assess potential attack vectors, not limited to EulerV2, but also those arising from incorporating the debt-token into collateralization and the Usual Stability Loan feature.

Scope

Repository: usual-dao/pegasus

Audited Commit: ecb77a5df2f65d8a6df0f622b0c3adfbc11ede58

Files:

- packages/solidity/src/oracles/EulerOracle.sol

Repository: euler-xyz/euler-usual

Audited Commit: fe7af6f5f18ce706f11b0727d10744cd2ca7e4af

Files:

- script/Deploy.s.sol

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
0	1	5

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue M-1: eUSD0 shares locked as RWA collateral can be redeemed and re-deposited to Usual by anyone, enabling an exploiter to gain from vault interest on the Dao's account

Source: https://github.com/sherlock-audit/2025-02-usual-stability-loan/issues/49

Summary

Vulnerability Detail

On Usual, anyone can redeem USD0 for any of the supported RWAs (as long as they are premissioned on the RWA they specify). Since eUSD0 is permissionless as a token, anyone holding USD0 can redeem it out of Usual (for a redemption fee currently set at 0.1%), and re-deposit it to mint USD0 when they choose to. In addition, since redeem/withdraw are not blocked on the USD0 vault, they can also redeem directly from the vault.

This fact can have the following negative consequences.

1. Anyone who redeems eUSD0 from Usual early can hold on to it as it gains interest and then use it to mint back USD0 including the accrued interest. The interest rate is set as 5% annual, if a large part of the vault funds is borrowed out, the eUSD0 vault share value will increase at close to that rate, in which case the interest will cover the 0.1% redemption fee within days.
2. If large quantities of the eUSD0 deposited to Usual are extracted and held by others, the operational flow of USL can be disputed. For example, if a borrow is repaid and the Dao wishes to deleverage, there might not be enough eUSD0 shares in Usual to enable de-leveraging the full repaid amount.

Impact

1. Loss of vault interest for the Dao - As detailed above. The probability of this scenario depends on conditions that effect the risk/reward of the exploiter: the vault utilization (low utilization means low interest per share which might not cover the redemption fee), time to maturity (the earlier the exploit takes place the higher the potential gain)
2. Operational Disruptions - inability for the Dao to deleverage at will, depending on availability of eUSD0 shares in Usual.
3. in conjunction with issue #51 this issue can be a more severe impact (see issue 51 scenario 2 for details).

Code Snippet

Tool Used

Manual Review

Recommendation

1. It is recommended to make the eUSD0 RWA collateral type permissioned on Usual, so that only the Dao (or a limited number of trusted users) will be able to interact (swap/redeem) with that collateral type.
2. The same access control can be applied to the vault redeem/withdraw actions, preventing direct vault withdrawals from anyone not authorized by the Dao.

Discussion

Usual

ACK as a possibility; NACK on the severity. The attacker would

- pay 1 USD0 to receive 1 USD worth of vaultshare
- pay a 0.1% redemption fee
- and only receive 4.85% yield, as if they just deposited into the vault directly if they were able to.

In any case we will prevent this attack vector. We will not allow redemptions for the time being by not granting the Collateral Treasury allowance for the vaultsharetoken to DaoCollateral.sol until we add a conditional redemption to certain RWA's in the core protocol

Issue L-1: Potential Risk Of Vault's upgradability

Source: https://github.com/sherlock-audit/2025-02-usual-stability-loan/issues/45

Summary

Vulnerability Detail

Note

The FactoryGovernor (https://docs.euler.finance/developers/periphery/governors/) has the right to upgrade the vault's implementation. The upgrade process is documented in the Euler documentation. The risk of a malicious upgrade is low as any upgrade has to be proposed by Euler DAO multisig, undergo a timelock, and can be vetoed by Euler Security Council multisig. Having said that, nothing is risk-free, and there is no guarantee the process will always work as intended or the Governor/Timelock contracts are 100% secure. Thus, the risk of upgradability to USUAL is still presented here for completeness.

The first iteration of the deployed vaults will be upgradable and will be an instance of the BeaconProxy. Therefore, it is possible for Euler to update the implementation of the eUSD0 and eUSDPP vaults.

The following are the risks identified:

1. Euler can update the implementation to transfer all the USD0 within the eUSD0 vault to an external wallet. The maximum loss would be the USD0 that USUAL treasury deposited that no one borrows.
2. Euler can update the implementation to transfer all the USD0++ collateral tokens within the eUSD0PP vault to an external wallet. Similarly, the maximum amount of USD0++ that can be lost is roughly the same as the USD0 deposited due to the LTV limitation.
3. USUAL protocol allows users to swap RWA (e.g., USYC, M^0) into USD0. If the vault shares (eUSD0) are one of the accepted RWA token, which means that `$.tokenMapping.isUsd0Collateral(eUSD0)` is true, it is possible for Euler to mint a large amount of eUSD0 and swap it for USD0 or USDC via the `DaoCollateral` contract.
4. USUAL protocol will call the Oracle Wrapper (`EulerOracle`), which internally calls the `EulerRouter(ROUTER).getQuote()` that fetches the price from the vault's native convert function (e.g., convertToAsset) to determine the value of the vault shares (eUSD0). As such, Euler can update the implementation of the vault's convert function to return an incorrect pricing that is either deflated or inflated.

```
// deploy and configure the USD0++ vault
IEVault eUSD0PP =
    IEVault(eVaultFactory.createProxy(address(0), true, abi.encodePacked(USD0PP,
    ↪  oracleRouter, USD)));
```

```
// deploy and configure the USD0 vault
IEVault eUSD0 = IEVault(eVaultFactory.createProxy(address(0), true,
↪  abi.encodePacked(USD0, oracleRouter, USD)));
```

https://github.com/euler-xyz/euler-vault-kit/blob/23cdc17bc9d24c16bc8e395515d13c12a129388d/src/GenericFactory/GenericFactory.sol#L78

```
File: GenericFactory.sol
111:     /// @param upgradeable If true, the proxy will be an instance of the
↪  BeaconProxy. If false, a minimal meta proxy
112:     /// will be deployed
..SNIP..
116:     function createProxy(address desiredImplementation, bool upgradeable,
↪  bytes memory trailingData)
```

Impact

Refer to above for the risk identified.

Code Snippet

N/A

Tool Used

Manual Review

Recommendation

Review if it is necessary for the vaults to be upgradable. If the vaults have to be upgraded, the risks can be reduced by the following measures:

- Disallowing vault shares (eUSD0) to be swapped for USD0 and USDC within the USUAL protocol. In this case, if Euler is compromised, the loss will only be limited to the assets (USD0 and USD0++) within the Euler vaults.
- Ensure that none of the callers of `EulerOracle` oracle depends on the price returned to perform "mission-critical" operations (e.g., minting of USD0/USDP++, rebalancing of RWA) since the price can be manipulated if Euler is compromised.
- Setup a monitoring system to alert the team when the following events occur:
  - Pause Guardian switches the EVault implementation to a read-only proxy. This indicates that it is like that Euler team has detected or identified in-progress hacking attempt on the system

- Upgrade being queued in the Timelock
- Vault implementation being updated

Discussion

Usual

ACK: We are implementing a monitoring setup to alert us as outlined, as well as cooperating closely with the Euler DAO to be able to veto potential malicious governance.

Issue L-2: Flash loan of USD0++ tokens from the Euler's eUSD0PP vault

Source: https://github.com/sherlock-audit/2025-02-usual-stability-loan/issues/46

Summary

Vulnerability Detail

Following is the extract from the audit documentation:

> USD0++: This vault is effectively a normal "escrow" vault, which means that anybody can deposit into it, but it does not have any LTVs configured, so it cannot be loaned out.

When deploying the USD0++ vault, the `setHookConfig` is set to `address(0), 0`, which means that all operations (including flash loan) will be enabled.

```
// deploy and configure the USD0++ vault
IEVault eUSD0PP =
    IEVault(eVaultFactory.createProxy(address(0), true, abi.encodePacked(USD0PP,
    ↪  oracleRouter, USD)));
oracleRouter.govSetResolvedVault(address(eUSD0PP), true);
eUSD0PP.setHookConfig(address(0), 0);
```

Following is the flash loan implementation taken from the Euler vault. The access control check at Line 147 will pass because `vaultStorage.hookedOps` is empty. Next, the USD0++ tokens residing in the eUSD0PP vault will be transferred to the caller in Line 153. In addition, this is an interest-free flash loan, as the borrower simply needs to return the same amount back at the end of the transaction.

https://github.com/euler-xyz/euler-vault-kit/blob/cecb8650c7ff9d2d67790ccaf7402f3eadae4734/src/EVault/modules/Borrowing.sol#L208

```
File: Borrowing.sol
145:     function flashLoan(uint256 amount, bytes calldata data) public virtual
↪  nonReentrant {
146:         address account = EVCAuthenticate();
147:         callHook(vaultStorage.hookedOps, OP_FLASHLOAN, account);
148:
149:         (IERC20 asset,,) = ProxyUtils.metadata();
150:
151:         uint256 origBalance = asset.balanceOf(address(this));
152:
153:         asset.safeTransfer(account, amount);
154:
155:         IFlashLoan(account).onFlashLoan(data);
```

```
156:
157:         if (asset.balanceOf(address(this)) < origBalance) revert
↪  E_FlashLoanNotRepaid();
158:     }
```

Allowing the flash loan of USD0++ might have the following side-effects:

- A sudden increase in the circulating supply of the USD0++ (Only for single block)
- A sudden increase in the circulating supply of the USD0 (Only for single block). USD0++ obtained can be redeemed for USD0 at a floor price of 0.87. However, the attacker has to ensure that its operation earns more USD0 at the end so that it can mint USD0++ to repay the flash loan
- Present inconsistent views of circulating supply (Both USD0++ and USD0) to external contracts

Impact

Refer to above for the side-effects.

Code Snippet

N/A

Tool Used

Manual Review

Recommendation

Disable flash loan on the eUSD0PP vault to reduce the attack surface.

Discussion

Usual

ACK: The spec wasnt required and has been rescinded accordingly. We already have 340 million USD0++ at Morpho that is flashloanable, and preventing flash-loans is a band-aid at most.

Issue L-3: Interest gained by shares that were locked in Usual for leverage can not be collected by the Yield Treasury

Source: https://github.com/sherlock-audit/2025-02-usual-stability-loan/issues/48

Summary

Vulnerability Detail

The Post-Repay/Liquidation deleverage flow presented is as follows: The borrowed USD0 amount+interest (weather repaid by the borrower or extracted from the borrower's collateral) is split, where the interest part is collected by the Yield Treasury (a multisig wallet) and the principal part is burned in Usual to extract locked eUSD0 shares for deleverage.

This flow assumes that all the accrued interest can be collected by the Yield Treasury, however if the vault funds are leveraged by using eUSD0 as RWA, this assumption doesn't hold for the following reason: any interest accrued in the vault is accounted for as share price increase. If some of the shares are held as Usual RWA collateral while the price increases, it means more USD0 will have to be burned to extract those shares. In other words, the protocol's benefit from this part of the interest is in the form of over-collateralization by the same RWA that existed before the leverage (by burning more USD0 to extract the shares then was minted when they were deposited).

Example:

1. Dao initial deposit to the vault: 100,000 USD0 (Dao gets 100,000 eUSD0 shares)
2. Dao leverages once by using all shares to mint another 100,000 USD0 and deposits those to the vault as well (now there's 100,000 shares as RWA in Usual and 100,000 shares held by the Dao)
3. All the funds are borrowed from the vault as two 100,000 borrows, and returned after some time with 10% interest.
4. When the first borrow is returned (100,000 + 10,000 interest), according to the presented flow, the funds are extracted from the vault, 10,000 is sent to the Yield Treasury and the rest (100,000) is burned on Usual for the locked shares. However, since the share price is now 1.1, the 100,000 USD0 burned enables a redeem of only 90909 eUSD0 shares.
5. When the second borrow is returned, the Dao doesn't have enough eUSD0 shares to fully redeem the vault. It will need to bring USD0 from another source to burn on Usual and fully release the shares, and only then fully redeem the vault.

Impact

1. Operational Complexity: From an operational perspective, if the current flow is implemented as is, the Dao will need to use the interest paid to the Yield Treasury (or any other source) to fully release the vault shares and redeem the vault. (until a new flow is implemented)
2. Financial Impact: Any DAO financial commitment based on the expectation that all interest will be available for use through the Yield Treasury will be broken.

Code Snippet

Tool Used

Manual Review

Recommendation

It is recommended that the post-repay/liquidation flow is changed to: As long as there is still leverage in the system (some eUSD0 is still used as RWA collateral) all the repaid/liquidated funds (principal+interest) are used for burning/deleverage on Usual, when leverage is eliminated, the interest can be sent to the Yield Treasury.

Discussion

Usual

ACK: Incorrect flow diagram, will be updated. We would and will follow the recommendation outlined here, which is also the easiest way in terms of complexity. Any Interest accrued was able to be minted as USD0 to the YIELD_TREASURY to start with, so it was a given for us that we would use it that way.

Issue L-4: `LIQUIDATION_LTV` is too high

Source: https://github.com/sherlock-audit/2025-02-usual-stability-loan/issues/50

Summary

The `LIQUIDATION_LTV` value of `1e4 - 1` leaves little buffer for liquidation delays before bad debt starts accumulating.

Vulnerability Detail

With the current setting of `1e4 - 1`, liquidation must occur, and the debt must be repaid within approximately 17.5 hours after reaching `LIQUIDATION_LTV`. Otherwise, accrued interest will exceed the deposited collateral.

Since the vault prevents debt socialization (`CFG_DONT_SOCIALIZE_DEBT`) and the only liquidator is the DAO, the DAO will need to cover the accumulating interest. As a result, profits will diminish over time.

Additionally, with the introduction of the NAV oracle, `LLTV` will be influenced not only by time but also by fluctuations in the `USD0` price. This means that interest gains from borrowing may not provide a sufficient buffer for bad debt, leading to real losses for the DAO rather than just reduced interest revenue.

Impact

Reduced interest earnings for the Yield Treasury or potential real losses for DAO after NAV oracle introduction.

Code Snippet

```
uint16 internal constant LIQUIDATION_LTV = 1e4 - 1;
```

Tool Used

Manual Review

Recommendation

Consider lowering the `LIQUIDATION_LTV` by a few basis points to allow for a sufficient buffer in case of unexpected liquidation delays.

Discussion

Usual

Ack. We will keep the liquidation window tight, and consider our liquidation window in the future.

Issue L-5: Possible side-effects of setting the USD0 vault to CFG_DONT_SOCIALIZE_DEBT

Source: https://github.com/sherlock-audit/2025-02-usual-stability-loan/issues/51

Summary

Vulnerability Detail

While the Dao is the only one allowed to deposit to the USD0 vault, there are some niche scenarios where having the vault configured as CFG_DONT_SOCIALIZE_DEBT might have unexpected effects:

Scenario 1:

1. Assume the vault is leveraged by the Dao (e.g. 3x)
2. Assume (hypothetically) that the vault enters a bad debt situation such as a third of the debt is forgone.
3. Since the vault is set as CFG_DONT_SOCIALIZE_DEBT, the vault share price remains the same, leaving the last to redeem to bear the loss.
4. When the Dao starts to de-leverage, the first cycles (redeeming shares from the vault and then burning the received USD0 on Usual for more shares) might use up all the available funds, leaving the vault empty before the Dao can withdraw its original investment.
5. If debt was socialized, each de-leverage cycle would have absorbed part of the bad debt, possibly enabling Dao to retrieve part of its original investment (but causing the Usual UDS0 system to absorb more of the bad debt).

In this scenario the choice is between:

A. Socialize debt - The Usual USD0 system absorbs more bad debt (increasing the chances of undercollateralization/depeg)

B. Not Socialize debt - Dao initial investment absorbs more of the bad debt (increasing the chances of loss of initial deposit+interest) Which option is preferable is left as a design decision of the Protocol.

Scenario 2:

Issue #49 (non-Dao ownership/redemption of vault shares) can have more severe impacts when combined with this issue:

1. Assume the same scenario as above, only this time a user redeems and holds all the uSD0 shares that are kept as Usual RWA collateral.
2. When the vault enters bad debt, the user can:

A. Deposit the shares back to Usual and mint USD0. Because debt is not socialized the user can mint USD0 based on the original price, possibly causing USD0 to become undercollateralized (depegged) since the shares are not actually worth the price the oracle quotes for them.

B. Frontrun the Dao in withdrawing from the vault, leaving the Dao with nothing to withdraw.

Impact

While extreme bad debt scenarios are highly unlikely on the USL vault, they can still happen (in cases of sudden RWA price drops for example).

Scenario 1: As mentioned, a design choice rather than a definite impact

Scenario 2: Possible under-collateralization/depeg of USD0 as well as loss of initial Dao deposit to the vault.

Code Snippet

```
eUSD0.setConfigFlags(CFG_DONT_SOCIALIZE_DEBT);
```

Tool Used

Manual Review

Recommendation

If issue #49 is resolved, choosing a configuration is a design choice as described in scenario 1.

Discussion

Usual

Ack. As pointed out, this is a design choice, with no direct benefit from choosing the A Scenario instead.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.