OpenZeppelin | security

Origin Dollar Dripper & Uniswap Additions

ORIGIN

April 20, 2023

This security assessment was prepared by OpenZeppelin.

Table of Contents

Summary

Type	DeFi
Timeline	From 2023-03-20 To 2023-04-03
Languages	Solidity

Total Issues	26 (21 resolved, 1 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	2 (2 resolved)
Low Severity Issues	10 (7 resolved, 1 partially resolved)
Notes & Additional Information	14 (12 resolved)

Scope

We audited the changes made to the OriginProtocol/origin-dollar repository in pull request #1203 up to commit `281852b` and pull request #1258 up to `2ddb8d0`.

In scope were the following contracts:

```
contracts
├── interfaces
│   ├── IUniswapV3Strategy.sol
│   ├── IVault.sol
│   ├── IVaultValueChecker.sol
│   └── uniswap
│       └── v3
│           ├── INonfungiblePositionManager.sol
│           └── IUniswapV3Helper.sol
├── proxies
│   └── Proxies.sol
├── strategies
│   └── uniswap
│       ├── UniswapV3LiquidityManager.sol
│       ├── UniswapV3Strategy.sol
│       └── UniswapV3StrategyStorage.sol
├── utils
│   ├── InitializableAbstractStrategy.sol
│   └── UniswapV3Helper.sol
└── vault
    ├── VaultAdmin.sol
    ├── VaultCore.sol
    └── VaultStorage.sol
```

System Overview

Origin Dollar (OUSD) is an ERC-20-compliant stablecoin backed by USDT, USDC, and DAI. An explanation for how it works, including how it generates yield, is available in our previous audit report. The reviewed code augments this functionality with two new features.

Firstly, the "Dripper" mechanism is integrated into the core protocol, ensuring that the yield generated by the system is gradually released to OUSD holders using the standard rebase mechanism. In equilibrium, the incoming yield should match the released amount, but the new mechanism helps to stabilize any volatility in the investment strategies.

Secondly, the reviewed code introduces a new investment strategy to add liquidity to a Uniswap V3 pool. However, in contrast to the other strategies, two stablecoins need to be invested together at a specific ratio, which introduces the possibility that not all capital can be invested simultaneously. Therefore, the Uniswap strategy nominates an existing strategy per stablecoin to be the "reserve strategy" and excess funds will be directed there, to be retrieved when required. The core contracts have been modified accordingly to accommodate some strategies interacting with other ones.

Security Model and Trust Assumptions

The overall structure of the project is the same, so the security architecture is still described by our previous stablecoin and governance audit reports. However, the new functionality makes some incremental modifications to the security architecture.

In the interest of simplicity, the project has always accepted a close but imperfect match between the collateral depositors and the yield beneficiaries. This is a consequence of pooling the collateral and yield distribution, and the discontinuous nature of claiming yield and rewards from the strategies. The new Dripper functionality exacerbates this discrepancy, since some OUSD holders will receive the yield earned before they joined, and some will redeem their OUSD tokens without receiving the yield that is due to be distributed. This should have a minor effect during normal operations.

In addition to the trust assumptions placed on the new Uniswap strategy (which are equivalent to those placed in the other strategies), the reserve strategy mechanism introduces dependencies between the different strategies. For example, rebalancing the Uniswap strategy may require funds to be withdrawn from the reserve strategy, which requires the reserve strategy to be functioning correctly and to have sufficient liquidity to fulfill the request.

Lastly, it should be noted that the Uniswap strategy needs to be actively managed, and a new Operator role has been introduced to help manage it. However, this role is restricted in several important ways:

- The operator address can be set or replaced by the strategist or governor address.
- All powers granted to the operator are also granted to the strategist and governor addresses.
- The governor and strategist addresses can pause the ability of the operator to rebalance any Uniswap position, or to swap the assets between stablecoins.
- The governor and strategist addresses can set bounds on the possible positions and swaps the operator can specify.
- The governor and strategist addresses can set a maximum amount of value invested in Uniswap V3.
- The governor and strategist addresses can set a maximum amount the operator can lose before its powers are revoked. This is important because losses are handled by simply pausing rebasing until the protocol becomes profitable again. In the meantime, the protocol may be undercollateralized.

Medium Severity

M-01 Function `withdrawAssetFromActivePositionOnlyVault` may fail

The `withdrawAssetFromActivePositionOnlyVault` function is supposed to withdraw part of the active position's liquidity in order to retrieve at least a minimum `amount` of an asset.

To integrate with Uniswap's interface, the corresponding liquidity is first calculated, and then the position is reduced by that amount. The minimum amount to receive for each token is provided as slippage protection.

However, the `_calculateLiquidityToWithdraw` function may miscalculate the token bounds. Specifically, the liquidity calculation is limited by the smaller of the two (price-adjusted) token limits. Therefore, the token values represent the maximum amount of tokens that can be withdrawn. Since they are treated as a minimum for slippage purposes, the withdrawal will fail.

One option to address this discrepancy would be to continue rounding up when calculating the token bound for liquidity calculations, but rounding down when using that bound for slippage protection. Another would be to remove the slippage protection entirely, since the withdrawal is executed atomically with the calculation, so it doesn't protect against a pool that was imbalanced before the calculation. However, the most natural resolution would be to use the one-sided liquidity calculations instead. In this case, extra caution is required so that the price parameters are specified appropriately with respect to the position's price boundaries and the current active price, although this could be addressed by validating that the position has enough of the desired token. Finally, if the slippage bounds are still desired, the `getAmountsForLiquidity` function can still be used, but this time it will calculate the scaled values directly.

Update: *Resolved in pull request #1284 at commit 6349d32. The `withdrawAssetFromActivePositionOnlyVault` function has been removed entirely.*

M-02 Inconsistent design assumptions

The vault modifications to support the `UniswapV3Strategy` contract make inconsistent design assumptions.

In particular, when allocating funds, if the Uniswap strategy is the default strategy, the funds are deposited into the reserve strategy instead. However, when redeeming OUSD, the funds will be taken directly from the Uniswap strategy, which will attempt to take them from the active Uniswap position. This could create a scenario where OUSD cannot be redeemed because the funds are not retrieved from where they are invested. Relatedly, since funds are allocated to the reserve strategy directly, the `deposit` function of the Uniswap strategy is unused.

On the other hand, if the Uniswap strategy should not be a default strategy, then this requirement should be enforced at the code level and the extra logic in the `_allocate` function can be removed. In either case, its `deposit` function remains unused. It is also worth noting that this function cannot be used as written because the `nonReentrant` modifier on the `depositToUniswapV3Reserve` function would prevent it from being executed.

Consider deciding how the Uniswap v3 strategy will be used and updating the token flows accordingly.

Update: *Resolved in pull request #1284 at commit 6349d32. The Uniswap v3 strategy can no longer be a default strategy.*

Low Severity

L-01 Default values in implementation

The `UniswapV3Strategy` contract contains default values for some contract variables, including the `maxTVL` and `maxPositionValueLostThreshold`. These will be set in the implementation, but will not be initialized in the proxy where they are required.

Consider setting the variable defaults in the initializer. A similar observation applies to the Vault default values, although the contract is already deployed so there is a limited advantage to updating it.

Update: *Resolved in pull request #1290 at commit f70c57b.*

L-02 Empty positions are not closed

When closing a position, the `UniswapV3LiquidityManager` contract will exit early whenever the position has no liquidity. This could occur if the position is reduced by a privileged role or by the vault. In this scenario, the usual closing activities will be bypassed, which means:

- It will remain the active position if it already was.
- The `UniswapV3PositionClosed` event will not be emitted.
- The earned fees will not be collected.

If the position is closed as part of a swap and rebalance or a regular rebalance operation, it will no longer be active, so the fees cannot be collected directly either. The unclaimed fees could be recovered by reactivating the position and then collecting the fees.

In the interest of consistency, consider performing the closing activities on empty positions.

Update: *Resolved in pull request #1291 at commit 7767110.*

L-03 Inconsistent fee accounting

There are two contexts where a fraction of the yield is diverted away from OUSD token holders.

Firstly, the governor address can reserve up to 50% of the yield for itself. This bypasses the drip schedule and also allows the governor to retrieve the funds in any supported stablecoin regardless of the stablecoin price. Secondly, the governor address can direct up to 50% of the remaining yield to a trustee address, which receives OUSD tokens in accordance with the drip schedule. They should exchange these tokens for OGN, or could redeem the OUSD through the standard mechanism that releases a basket of stablecoins, scaled by their prices whenever the oracle indicates they are worth more than $1.

Consider documenting why there are two different mechanisms to extract fees, and how the funds are intended to be used. If possible, consider merging them into a single fee that could be split and redirected outside the core contract, which might require a mechanism to support the automatic buyback. Alternatively, in the interest of simplicity and predictability, consider issuing OUSD tokens to the governor instead of creating a new mechanism for protocol reserves.

Update: *Acknowledged, not resolved.*

L-04 Missing return values

The `increaseActivePositionLiquidity` function of the `UniswapV3LiquidityManager` contract should return the token amounts but does not return anything. Consider returning the correct values.

Update: *Resolved in pull request #1292 at commit f6c8e8c.*

L-05 Missing validation

After minting a new position, the `UniswapV3LiquidityManager` contract obtains a token ID from Uniswap and then overwrites the corresponding position record. In practice, Uniswap should provide unique identifiers so this will not cause a problem. Nevertheless, in the interest of defensive programming, consider validating that the record is empty before saving the new value.

Update: *Resolved in pull request #1293 at commit 49af9f3.*

L-06 Potentially locked ETH

The fallback function of the `VaultCore` contract and the fallback function of the `UniswapV3Strategy` contract are both `payable` but none of the expected target functions require ETH.

Consider removing the `payable` modifier or introducing a mechanism to withdraw any ETH sent to the contract.

Update: *Partially resolved in pull request #1296 at commit 91f9e0c. The fallback function of the `VaultCore` contract remains `payable`. The Origin team indicated that this may be necessary for future upgrades.*

L-07 Rebasing trustee is overpaid

When distributing rewards, the trustee address first receives new tokens and then the yield is distributed amongst rebasing token holders, which may include the trustee address (if it opted in to rebasing). In this scenario, the trustee will receive more tokens than expected, and the other OUSD token holders will receive less.

Consider distributing the yield *before* minting the trustee's fee. Note that this would require increasing the supply by only `(yield - fee)`, rather than the full `yield`, because the subsequent minting operation will increase the supply again.

Update: *Acknowledged, not resolved. The Origin team stated:*

> *We will keep this as is, since the vault code is simpler this way. For all of our use cases we will be sending these funds to contracts in which case the calculations will be correct.*

L-08 Tokens are not reinvested

After a position is closed directly, the received tokens (including the fees) remain in the strategy contract. Similarly, after a particular asset is removed from the Uniswap position, the other asset remains in the strategy contract. For consistency with the rest of the contract and the intended design philosophy, consider reinvesting these funds in the reserve strategy contracts.

Update: *Resolved in pull request #1297 at commit 6447916.*

L-09 UniswapV3Strategy implementation not disabled

The `UniswapV3Strategy` contract is intended to be deployed behind a proxy. In this scenario, it is good practice to disable the implementation contract as much as possible, typically through a mechanism like the `disableInitializers` function in the OpenZeppelin contracts library.

The `Initializable` contract in the current codebase does not have this option. This means the contract deployer will become the governor and will have the power to manage the contract, including setting the liquidity manager implementation, which is the target of a `delegatecall`. This could result in the contract executing a `selfdestruct` operation, which would disable the user-facing proxy contract.

Consider using the latest OpenZeppelin `Initializable` contract and calling `disableInitializers` in the implementation's constructor. Alternatively, consider setting the governor to the zero address in the implementation's constructor.

Update: *Resolved in pull request #1299 at commit 2ecc91a.*

L-10 Unsafe ABI encoding

It is not an uncommon practice to use `abi.encodeWithSignature` or `abi.encodeWithSelector` to generate calldata for a low-level call. However, the first option is not typo-safe and the second option is not type-safe. The result is that both of these methods are error-prone and should be considered unsafe.

Within `UniswapV3Strategy.sol` there are two occurrences of unsafe ABI encodings. Specifically:

- On line 293
- On line 317

Consider replacing all the occurrences of unsafe ABI encodings with `abi.encodeCall`, which checks whether the supplied values actually match the types expected by the called function and also avoids errors caused by typographical mistakes.

Update: *Resolved in pull request #1298 at commit 7c11174. The code uses `abi.encodeWithSelector` to match the target compiler version.*

Notes & Additional Information

N-01 Code simplification

We identified some opportunities for code simplification as follows:

- In the `_depositAll` function of the `UniswapV3StrategyStorage` contract, the conditions on lines 198 and 204 can be simplified, as the second term of the logical OR statement also covers its first term.
- In the `deposit` function of the `UniswapV3Strategy` contract the if statement could be simplified by using the conditional to choose a threshold, and then comparing the amount with that threshold.
- In the `_getTickPositionKey` function of the `UniswapV3LiquidityManager` contract there is a redundant type cast since `upperTick` already is of type `int24`.
- In the `_calculateLiquidityToWithdraw` function of the `UniswapV3LiquidityManager` contract, consider passing both `minAmountX`

parameters (instead of `amount`) to the liquidity calculation so that the function can be moved outside of the `if` statement.

Update: *Resolved in pull request #1300 at commit 2869118.*

N-02 Constants not using `UPPER_CASE` format

Throughout the codebase there are constants not using `UPPER_CASE` format. For instance:

- The `liquidityManagerImplPosition` constant declared on line 150 in `UniswapV3StrategyStorage.sol`
- The `adminImplPosition` constant declared on line 92 in `VaultStorage.sol`

According to the Solidity Style Guide, constants should be named with all capital letters with underscores separating words. For better readability, consider following this convention.

Update: *Resolved in pull request #1300 at commit d6276a0.*

N-03 Inconsistent drip schedule

This issue was raised with the Origin team during the audit and they indicated that this is intentional behavior. Nevertheless, we believe the discussion is instructive so it is included here for reference.

The new Dripper functionality is intended to release yield to OUSD token holders at a constant rate over a specified number of blocks. However, the funds are only released during a rebase, at which point the next release rate is calculated based on the remaining reserves. This ensures that new yield is automatically included but it also distorts the release schedule.

In the extreme case, if a rebase occurs every block then funds will be released at an exponentially decaying rate (that technically never ends). Consider specifying a minimum drip rate to augment or replace the drip duration.

Update*: Acknowledged, not resolved. The Origin team stated (abridged):*

> *The drip rate is actually exponentially decaying towards the incoming yield rate. [...] Our biggest priority is the simplicity of the implementation. If we stop getting yield in, but still have small positive end-user yield for a while, that is just fine, and is even our preference.*

N-04 Lack of indexed event parameters

The `UniswapV3StrategyStorage` contract contains several events that would benefit from having an indexed parameter to improve the ability of off-chain services to search for and filter for specific events. Our suggestions are to index all the `address` parameters in the first four events.

Update: *Resolved in pull request #1300 at commit 763fff2.*

N-05 Misleading comments

By updating the comments listed below the readability of the codebase could be improved:

- The explanatory comment for function `resetAllowanceOfTokens` of contract `UniswapV3Strategy` only mentions the position manager, while it removes the allowance from the swap router as well.
- In the `_closeActivePositionOnlyByVault` function of the `UniswapV3LiquidityManager` contract there is an incomplete comment line.

Update: *Resolved in pull request #1300 at commit acd5898.*

N-06 Missing underscore prefix for non-external functions and variables

A number of non-external functions and variables do not have an underscore prefix.

In the `UniswapV3LiquidityManager` contract, the internal functions:

- `getPositionValue`
- `ensureTVL`
- `swapsNotPausedAndWithinLimits`
- `rebalanceNotPaused`
- `rebalanceNotPausedAndWithinLimits`
- `updatePositionNetVal`
- `ensureNetLossThreshold`

In the `UniswapV3Strategy` contract, the internal function `onlyPoolTokens`.

In the `UniswapV3StrategyStorage` contract, the internal variables: - `helper` - `swapRouter` - `ticksToTokenId`

Consider adding an underscore prefix to all internal functions and variables, following the Solidity Style Guide.

Update: *Resolved in pull request #1300 at commit 2006a4a.*

N-07 Naming suggestions

The following naming changes may improve the readability of the codebase:

- Dripper's rate value is stored under struct field `perBlock`. However, yield drips are calculated with respect to a per-second rate. Consider renaming "perBlock" to the more accurate "perSecond".
- Consider renaming the `NetLossValueReset` event to "NetLostValueReset" to be consistent with the respective variable and the rest of the codebase. Similarly, consider renaming the `ensureNetLossThreshold` function to "ensureNetLostValueThreshold".

Update: *Resolved in pull request #1285 at commit d2b9a14 and pull request #1300 at commit 1eee3da.*

N-08 Non-canonical gap variable

The `UniswapV3StrategyStorage` contract contains a gap of 100 storage slots after its variable declarations. This is non-canonical for two reasons:

- the gap provides flexibility to add future variables to the contract layout without repositioning the variables in descendant contracts. In this case, the purpose of the contract is to capture all of the relevant variables, so there shouldn't be any in the descendant contracts.
- the gap typically expands the storage size to a round number (like 50 or 100) so all versions of the contract have a predictable size, and the storage layout of descendant contracts starts at a predictable slot number.

Consider adjusting the size of the gap so the total storage layout occupies 100 slots. Alternatively, if the descendant contracts will never have any variables, consider removing the gap entirely.

Update: *Resolved in pull request #1300 at commit c14672a.*

N-09 `require` statements with multiple conditions

Throughout the codebase, there are `require` statements that require multiple conditions to be satisfied. This could lead to misleading error messages being thrown in the following two cases:

- The `require` statement on line 897 of `UniswapV3LiquidityManager.sol`
- The `require` statement on line 912 of `UniswapV3LiquidityManager.sol`

To raise the most helpful error messages for failing require statements, consider having a single `require` statement per condition for these cases.

Update: *Resolved in pull request #1300 at commit 39e377a.*

N-10 State identifier visibility not explicitly declared

Throughout the codebase, there are state identifiers that lack an explicitly declared visibility. For instance:

- The constants and immutables in `VaultCore.sol`
- The constant `MINT_MINIMUM_ORACLE` in `VaultStorage.sol`

For clarity, consider always explicitly declaring the visibility of identifiers, even when the default visibility matches the intended visibility.

Update: *Resolved in pull request #1286 at commit 790cbef.*

N-11 Todo comments in the code

The following instances of TODO/Fixme comments were found in the codebase. These comments should be tracked in the project's issue backlog and resolved before the system is deployed:

- The `TODO` comment on line 146 in UniswapV3LiquidityManager.sol
- The `TODO` comment on line 134 in VaultCore.sol

During development, having well-described TODO/Fixme comments will make the process of tracking and solving them easier. Without this information, these comments may age and important information for the security of the system may be forgotten by the time it is released to production.

Consider removing all instances of TODO/Fixme comments and instead tracking them in the issues backlog. Alternatively, consider linking each inline TODO/Fixme to the corresponding issues backlog entry.

***Update**: Acknowledged, not resolved. The Origin team stated:*

> *The vault TODO is a known change we are planning on making, but we are waiting until we have a comprehensive fuzzing test suite on the vault and strategies.*

N-12 Typographical errors

Consider addressing the following typographical errors in the codebase.

In `VaultAdmin` contract:

- In line 372: "DriperDurationChanged" should be "DripperDurationChanged"

In `VaultCore` contract:

- In line 390 : "driper" should be "dripper"
- In line 463: "invarient" should be "invariant"
- In lines 458, 473, 484: 'ETH' should be 'USD'

In `UniswapV3Strategy` contract:

- In line 43: "delegecall" should be "delegatecall"

In `UniswapV3StrategyStorage` contract:

- In line 189: there is a redundant "back" word

In `UniswapV3LiquidityManager` contract:

- In lines 665, 712: "form" should be "from"
- In lines 119, 148: "Liquidiate" should be "Liquidate"
- Line 919 is redundant and incorrect
- Line 27: "exlcuding" should "excluding"

Update: *Resolved in pull request #1287 at commit 4b36c61 and pull request #1300 at commits a9cabad and 0a6750f.*

N-13 Unused function

The `_checkBalance` function (with no arguments) in the `VaultCore` contract is unused. Consider removing it from the codebase.

Update: *Resolved in pull request #1288 at commit 9516876.*

N-14 Unused imports

Throughout the codebase, imports on the following lines are unused and could be removed:

- Import `InitializableAbstractStrategy` of `UniswapV3LiquidityManager.sol`
- Import `IStrategy` of `UniswapV3LiquidityManager.sol`
- Import `IUniswapV3Pool` of `UniswapV3LiquidityManager.sol`
- Import `Strings` of `UniswapV3LiquidityManager.sol`
- Import `InitializableAbstractStrategy` of `UniswapV3Strategy.sol`
- Import `Tick` of `UniswapV3Helper.sol`
- Import `PositionKey` of `UniswapV3Helper.sol`
- Import `Strings` of `VaultCore.sol`
- Import `IVault` of `VaultCore.sol`
- Import `IStrategy` of `VaultStorage.sol`
- Import `IUniswapV3Strategy` of `VaultStorage.sol`
- Import `Helpers` of `VaultStorage.sol`

Consider removing unused imports to improve the overall clarity and readability of the codebase.

Update: *Resolved in pull request #1312 at commit ce15980, pull request #1289 at commit dbc208a and pull request #1332 at commit 4355edf.*

Monitoring Recommendations

While audits help in identifying code-level issues in the current implementation and potentially the code deployed in production, the Origin team is encouraged to consider incorporating monitoring activities in the production environment. Ongoing monitoring of deployed contracts helps identify potential threats and issues affecting production environments. With the goal of providing a complete security assessment, the monitoring recommendations section raises several actions addressing trust assumptions and out-of-scope components that can benefit from on-chain monitoring.

The previous audit report had a recommendation to monitor unusual user actions, investment protocol health, and privileged actions. These recommendations would naturally apply to the new Uniswap V3 strategy and its corresponding administrative actions.

In addition, it is also recommended to monitor:

- Changes to the `dripperReserve` as well as the frequency and size of `YieldReceived` events. This should help validate that the distribution is following the expected model, or identify any unexpected variations.
- The `NetLostValueChanged` event to identify how well the active Uniswap V3 position is being managed.

Lastly, the following safety conditions are checked on relevant state changes but they should be validated continuously to detect unexpected violations. The second one in particular could be surreptitiously violated if the operator makes a trade on a biased Uniswap pool.

- The active position must be worth less than the `maxTVL` parameter.
- The total value lost by the operator on Uniswap trades must be less than the `maxPositionValueLostThreshold` parameter.

Conclusions

No critical or high-severity issues have been found, which indicates an overall healthy protocol design and implementation. Several minor vulnerabilities have been identified, and fixes have been proposed. The UniswapV3 strategy may require additional clarification in specific areas, particularly with regard to its functionality as a default strategy for an asset. The audit team appreciated the numerous explanatory comments within the codebase, which helped clarify certain design choices. The Origin team has been very responsive and willing to provide context and detailed explanations as needed.