Origin Protocol OETH

Security Testing and Assessment

May 8, 2023

Prepared for Original Protocol

Narya.ai

Table of Content

Summary

Overview

From April 17, 2023, to May 8, 2023, Origin Protocol engaged Narya.ai to evaluate the security of its OETH in the GitHub repository: https://github.com/OriginProtocol/origin-dollar (commit 2a8975e91e1371fa23b268b30c8959f95027dafb).

OETH is a vault protocol allowing users to deposit ETH to receive OETH tokens. OETH is a self-custodial, yield-generating token based on OUSD. The vault has the ability to employ multiple strategies for yield generation and drip the proceeds to smooth out the returns.

Project Scope

We reviewed and tested the smart contracts that implement OETH. Other security-critical components of OETH, such as off-chain services and the web front end, are not included in the scope of this security assessment. We recommend a further review of those components.

Summary of Findings

Severity	# of Findings
High	0
Medium	0
Low	5
Informational	0
Gas	1
Total	6

Throughout the testing, we identified 5 medium-severity issues and 1 Gas issue:

- 1 issue of an unused mapping `isUpgraded`.
- 1 issue of asymmetric ACL in VaultAdmin.
- 1 issue of maximum approval for the platform to spend on behalf of the `Generalized4626Strategy`.
- 1 issue of `Dripper::collect()` not collecting anything on the first call.
- 1 issue of no funds sent to the vault when the dripper has a `dripDuration` bigger than `remaining`.

- 1 gas issue concerning an incorrect modifier used in Harvester.

Disclaimer

This report should not be used as investment advice.

Narya.ai uses an automatic testing technique to test smart contracts' security properties and business logic rapidly and continuously. However, we do not provide any guarantees on eliminating all possible security issues. The technique has its limitations: for example, it may not generate a random edge case that violates an invariant during the allotted time. Its use is also limited by time and resource constraints.

Unlike time-boxed security assessment, Narya.ai advises continuing to create and update security tests throughout the project’s lifetime. In addition, Narya.ai recommends proceeding with several other independent audits and a public bug bounty program to ensure smart contract security.

Project Overview

Origin is divided into multiple components:

- A ERC20 token (OUSD/OETH) that is pegged to a stable (DAI/USDT/USDC/ETH), with a concept of user shares.
- A vault that accepts stablecoins and mints the OETH/OUSD token. It is also in charge of investing the funds in strategies to generate the yield.
- Strategies that invest the funds and produce returns.
- A harvester that collects the rewards from the strategies, swap them into stablecoins and send them to the dripper.
- A dripper that drips out rewards to the vault over a period of time to smooth out the returns.

Tests

Tested Invariants and Properties

We relied on the Narya engine that used a smart fuzzing approach to test the following 23 invariants and properties of the smart contracts. When specified, the tests have a variant adaptation for the live OETH and/or OUSD Vaults.

ID	Invariant/Property Description	Found Bug(s)
01	The OUSD contract cannot change owners unless requested by the current owner (including live OETH/OUSD Vaults).	Passed
02	The Vault contract cannot change owners unless requested by the current owner (including live OETH/OUSD Vaults).	Passed
03	The VaultAdmin contract cannot change owners unless requested by the current owner (including live OETH/OUSD Vaults).	Passed
04	The Harvester contract cannot change owners unless requested by the current owner (including live OETH/OUSD Vaults).	Passed
05	The Strategy contract cannot change owners unless requested by the current owner (including live OETH/OUSD Vaults).	Passed
06	The Dripper contract cannot change owners unless requested by the current owner (including live OETH/OUSD Vaults).	Passed

07	User's funds locked in the vault can be reclaimed at any time by the user and cannot be less than 90% of the original deposited value (including live OETH/OUSD Vaults).	Passed
08	When the Vault has locked funds from user deposits, the Vault balance balance cannot be completely drained (including live OETH/OUSD Vaults).	Passed
09	A user that locked funds for OETH/OUSD shares, can send them to anyone and, they in turn, are able to redeem them.	Passed
10	When using the meta OUSD strategy, vault functionality (mint and redeem) is not affected (including live OUSD Vaults).	Passed
11	When using strategies, users can still withdraw at any time from the vault.	Passed
12	When depositing/withdrawing/reallocating directly into strategies, validate that the amount of tokens transferred checks out.	Passed
13	Platform Token can only be changed by the governor.	Passed
14	Reward Token can only be changed by the governor.	Passed
15	The strategist can only be set by the governor (including live OETH/OUSD Vaults).	Passed
16	The governor can pull out any ERC20 assets from the Dripper (including live OETH/OUSD Vaults).	Passed
17	The governor can pull out any ERC20 assets from the Harvester (including live OETH/OUSD Vaults).	Passed
18	The governor can pull out any ERC20 assets from the Strategy.	Passed
19	Only the governor can set the Vault Admin implementation (including live OETH/OUSD Vaults).	Passed
20	Only the governor can enable strategies in the harvester.	Passed
21	When simulating a flash loan, checks that a user that deposits and redeems in the same transaction cannot withdraw more funds than what was originally deposited (including live OETH/OUSD Vaults).	Passed
22	The platform cannot withdraw any arbitrary amount of tokens from the strategy at any time.	3. Approval should be done for the amounts

		transferred and not for type(uint).max
23	Collect() on the dripper should get the dripped funds and send them to the vault.	4. Dripper::collect() does not collect anything on the first call

Key Findings and Recommendations

1. Unused mapping `isUpgraded`

Severity: **Low**

Description

The mapping is never set anywhere in the contract or its parent contracts. The only use is in the getter `creditsBalanceOfHighres()` which uses a default state (0), so it will always return false.

Code 1 contracts/contracts/token/OUSD.sol#L52

```
    mapping(address => uint256) public isUpgraded;
```

Impact

Contracts or users that rely on `creditsBalanceOfHighres()` would get an incorrect result, potentially affecting their functionality. Additionally wasting gas reading the state variable when the result is predictably false.

Failed Invariant

Found during code review.

Recommendation

If not used, this should be removed.

Remediation

This issue has been acknowledged by Origin Protocol and fixed at commit

2. Asymmetric ACL in vaultAdmin

Severity: **Low**

Description

pauseRebase() requires onlyGovernorOrStrategist but unpauseRebase() uses onlyGovernor which prevents a strategist from unpausing a vault that was paused by him/her.

Code 2 contracts/contracts/vault/VaultAdmin.sol#L406

```
    function pauseRebase() external onlyGovernorOrStrategist {
        rebasePaused = true;
        emit RebasePaused();
    }

    /**
     * @dev Set the deposit paused flag to true to allow rebasing.
     */
    function unpauseRebase() external onlyGovernor {
        rebasePaused = false;
        emit RebaseUnpaused();
    }
```

Impact

A strategist can find himself/herself locking the rebase mechanism without any way to revert the action without the help of the contract's governor.

Failed Invariant

Found during code review.

Recommendation

unpauseRebase() should use the same modifier `onlyGovernorOrStrategist`.

Remediation

This issue has been acknowledged by Origin Protocol and fixed at commit

3. Approval should be done for the amounts transferred and not for type(uint).max

Severity: **Low**

Description

pauseRebase() requires onlyGovernorOrStrategist but unpauseRebase() uses onlyGovernor which prevents a strategist from unpausing a vault that was paused by him/her.

Code 3 contracts/contracts/strategies/Generalized4626Strategy.sol#LL82C1-L92C6

```
    function _abstractSetPToken(address _asset, address _pToken)
        internal
        override
    {
        shareToken = IERC20(_pToken);
        assetToken = IERC20(_asset);

        // Safe approval
        shareToken.safeApprove(platformAddress, type(uint256).max);
        assetToken.safeApprove(platformAddress, type(uint256).max);
    }
```

Code 4 contracts/contracts/strategies/Generalized4626Strategy.sol#L134

```
    function safeApproveAllTokens() external override {
        assetToken.safeApprove(platformAddress, type(uint256).max);
        shareToken.safeApprove(platformAddress, type(uint256).max);
    }
```

Impact

The risk is that if the platform contract is compromised, it could steal all the shares and assets in the `Generalized4626Strategy` contract.

Failed Invariant

```
    function invariantMaliciousPlatform() public {
    // only governor/strategist/vault may withdraw from strategy
        require(balanceOf(address(strategy)) == 100,
            "strategy LP was drained");
    }
```

Recommendation

Approval should be done for the exact amounts that are expected from the platform contract to use.

Remediation

This issue has been acknowledged by Origin Protocol and fixed at commit

4. Dripper::collect() does not collect anything on the first call

Severity: **Low**
File(s): Dripper.sol

Description

When `setDripDuration()` is called for the first time, `_collect()` is called, and `drip.perBlock` is set to 0 as no funds are available at the time. Later when funds are available and a user wants to collect drips, `_availableFunds()` will use `drip.perBlock` which was previously set to 0 during setup. This makes the amount to send to be 0. The user has to call a second time collect() to be able to collect funds.

Code 5 contracts/contracts/harvest/Dripper.sol#L131

```
function _collect() internal {
        // Calculate send
        uint256 balance = IERC20(token).balanceOf(address(this));
        uint256 amountToSend = _availableFunds(balance, drip);
        uint256 remaining = balance - amountToSend;
        // Calculate new drip perBlock
        //   Gas savings by setting entire struct at one time
        drip = Drip({
            perBlock: uint192(remaining / dripDuration),
            lastCollect: uint64(block.timestamp)
        });
        // Send funds
        IERC20(token).safeTransfer(vault, amountToSend);
    }
}
```

Impact

When calling `collect()` for the first time, no rewards will be sent to the vault.

Failed Invariant

```
    function testDripper(uint amount) public {
        vm.assume(amount > 10 && amount < 1 ether);

        deal(DAI, bob, amount);

        // mint for bob and send to alice
        vm.startPrank(bob);
```

```
        uint oldDaiBalance = IERC20(DAI).balanceOf(address(bob));

        IERC20(DAI).approve(address(vault), amount);
        vault.mint(DAI, amount, 0);

        vault.allocate();

        require(IERC20(USDC).balanceOf(address(strategy)) == 100,
            "no rewards given");

        require(balanceOf(address(strategy)) == amount,
            "no shares given");

        require(IERC20(DAI).balanceOf(address(this)) == amount,
            "vault didnt receive asset");

        vault.redeem(ousd.balanceOf(bob), 0);

        require(IERC20(DAI).balanceOf(address(bob)) >= oldDaiBalance * 9 /
10,
            "Lost more than 90% (accounting for rounding issues)");

        require(IERC20(USDC).balanceOf(address(strategy)) == 100,
            "rewards disappeared from strategy");

        // actually due to known rounding issues, the redeemed ousd
        // won't give you back the same amount as you deposited
        // this means, strategy will have leftover shares

        harvester.harvestAndSwap(address(strategy), rewardRecipient);

        // console.log("dripper",
IERC20(USDT).balanceOf(address(dripper)));
        // console.log("rewardRecipient",
IERC20(USDT).balanceOf(address(rewardRecipient)));


        require(IERC20(USDT).balanceOf(address(dripper)) > 0,
            "dripper didnt receive funds");

        vm.warp(block.timestamp + 1 minutes);
        dripper.collect();
```

```
        // vm.warp(block.timestamp + 1 minutes);
        // dripper.collect();
        require(IERC20(USDT).balanceOf(address(vault)) > 0,
            "vault didnt receive funds");

        // console.log(IERC20(USDT).balanceOf(address(vault)));

        vm.stopPrank();
    }
```

Recommendation

When calling `_collect()` for the first time, it should accurately calculate `drip.perBlock`.

Remediation

This issue has been acknowledged by Origin Protocol and fixed at commit

5. No funds sent to vault when the dripper has a dripDuration bigger than remaining

Severity: **Low**
File(s): Dripper.sol

Description

If `dripDuration` is bigger than `remaining`, no tokens will be sent to the vault, even if the dripper contract has some balance. This might occur when the dripper hasn't accumulated enough funds to have `remaining` bigger than `dripDuration`.

Code 6 contracts/contracts/harvest/Dripper.sol#L130

```
function _collect() internal {
        // Calculate send
        uint256 balance = IERC20(token).balanceOf(address(this));
        uint256 amountToSend = _availableFunds(balance, drip);
        uint256 remaining = balance - amountToSend;
        // Calculate new drip perBlock
        //   Gas savings by setting entire struct at one time
        drip = Drip({
            perBlock: uint192(remaining / dripDuration),
            lastCollect: uint64(block.timestamp)
        });
        // Send funds
        IERC20(token).safeTransfer(vault, amountToSend);
    }
}
```

Impact

When trying to send the accumulated rewards to the vault (for eg. to be reinvested), `collect()` might fail if the `dripDuration` is bigger than the remaining rewards balance. In that case, only the governor can withdraw the rewards manually using `transferToken()`.

Failed Invariant

Found during code review.

Recommendation

Increase the precision of `remaining` (for eg. by multiplying with 1e18) when doing the division in order to prevent the rounding to zero.

Remediation

This issue has been acknowledged by Origin Protocol and fixed at commit

6. Modifier not correct

Severity: Gas

Description

The modifier allows the vault to call it but the Vault doesn't make any calls to this function.

Code 7 contracts/contracts/harvest/Harvester.sol#L195

```
    function setSupportedStrategy(address _strategyAddress, bool _isSupported)
        external
        onlyVaultOrGovernor
    {
        supportedStrategies[_strategyAddress] = _isSupported;
        emit SupportedStrategyUpdate(_strategyAddress, _isSupported);
    }
```

Impact

If the caller of this function is not the Governor, then an additional check to see if the caller is the Vault is made. This could be removed for better gas efficiency.

Failed Invariant

Found during code review.

Recommendation

Remove the check on the msg.sender to see if it is the Vault, as the Vault will never call this function.

Remediation

This issue has been acknowledged by Origin Protocol and fixed at commit

Fix Log

ID	Title	Severity	Status
01	1. Unused mapping `isUpgraded`	**Low**	
02	2. Asymmetric ACL in vaultAdmin	**Low**	
03	3. Approval should be done for the amounts transferred and not for type(uint).max	**Low**	
04	4. Dripper::collect() does not collect anything on the first call	**Low**	
05	5. No funds sent to vault when the dripper has a dripDuration bigger than remaining	**Low**	
06	6. Modifier not correct	**Gas**	

* The issues have been acknowledged by Origin but won't be fully fixed at the time of this report. Check the former description of each issue for details.

Appendix

Severity Categories

Severity	Description
Gas	Gas optimization.
Informational	The issue does not pose a security risk but is relevant to best security practices.
Low	The issue does not put assets at risk such as functions being inconsistent with specifications or issues with comments.
Medium	The issue puts assets at risk not directly but with a hypothetical attack path, stated assumptions, or external requirements. Or the issue impacts the functionalities or availability of the protocol.
High	The issue directly results in assets being stolen, lost, or compromised.