Byte Masons - oToken ZAP Audit Report

Version 1.2

Zigtur

July 5, 2024

Byte Masons - oToken ZAP - Audit Report

Zigtur

July 5th, 2024

Prepared by: Zigtur

Table of Contents

Introduction

Disclaimer

A smart contract security review cannot guarantee the complete absence of vulnerabilities. This effort, bound by time, resources, and expertise, aims to identify as many security issues as possible. However, there is no assurance of 100% security post-review, nor is there a guarantee that the review will uncover all potential problems in the smart contracts. It is highly recommended to conduct subsequent security reviews, implement bug bounty programs, and perform on-chain monitoring.

About Zigtur

Zigtur is an independent blockchain security researcher dedicated to enhancing the security of the blockchain ecosystem. With a history of identifying numerous security vulnerabilities across various protocols in public audit contests and private audits, **Zigtur** strives to contribute to the safety and reliability of blockchain projects through meticulous security research and reviews. Explore previous work here or reach out on X @zigtur.

About oToken

oToken means Options Token. It represents the right to receive different forms of discounted assets in return for the appropriate payment. The option is executed through specialized contracts named exercisers.

The protocol implements a new “zapping” feature as part of the “Discount Exerciser”. This feature allows a user to acquire underlying tokens instantly, without paying payment tokens, at the cost of an instant exit fee.

Security Assessment Summary

Review commit hash - 1293670 & 6cbaea2

Fixes review commit hash - 8d795ca

Deployment chains

- Ethereum Mainnet
- BSC
- Optimism
- Mode

Scope

The audit focuses on the newly implemented “zapping” feature and other modifications done since the last audit.

The following smart contracts are in scope of the review:

- exercise/DiscountExercise.sol
- OptionsToken.sol
- oracles/BalancerOracle.sol
- oracles/ThenaOracle.sol
- exercise/BaseExercise.sol
- oracles/AlgebraOracle.sol
- oracles/UniswapV3Oracle.sol
- helpers/SwapHelper.sol
- VeloSolidMixin.sol

Risk Classification

	Impact: High	**Impact:** Medium	**Impact:** Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Issues

HIGH-01 - Zap and Redeem are not compatible

Description

Scope:

- DiscountExercise.sol#L215

The `_redeem` and `_zap` calculations are not compatible. This makes one more profitable than the user depending on the multiplier configuration.

When the `multiplier` is less than 50%, `_redeem` will be more profitable than `_zap`.

When the `multiplier` is greater than 50%, `_zap` will be more profitable than `_redeem`.

This is due to incompatible calculation

Scenario

In the following scenario, zapping is more profitable than redeeming.

Initial situation:

- price = 1$
- oTokens `amount` = 1000
- `multiplier` = 0.8 (80%)
- instantExitFee = 0.1 (10%)

Redeem:

- payment tokens value = 1$ * 1000 * 0.8 = 800$
- underlying tokens value received = 1000$
- user's profit = 1000$ - 800$ = 200$

Zap:

- discountedUnderlying = 1000 * 0.8 = 800$
- fees = 800$ * 0.1 = 80$
- underlying tokens value received = 720$
- user's profit = 720$

As we can see, zapping with this configuration is more profitable than redeeming.

Proof of Concept

A Foundry unit test file is given in Appendix.

Recommendation

The `_zap` function should return a profit similar to `_redeem` minus the instant fees. This can be fixed by returning `(10_000 - multiplier) * amount` instead of `multiplier * amount`.

However, this fix leads to further incompatibilities. Indeed, `_setMultiplier` allows setting a multiplier greater than `10_000` which will lead to a DOS of `_zap` due to integer underflow.

A patch is given in Appendix. It modifies `_zap` to fix the calculation and reverts when an incompatible multiplier is used.

Resolution

Byte Masons team: Fixed.

Zigtur: Fix reviewed and approved.

MEDIUM-01 - Zapping possible DOS

Description

Scope:

- DiscountExercise.sol#L222-L229

The documentation indicates:

> We limit the amount of funds we leave in an exercise contract at any given time to limit risk.

When the contract is not funded with `underlyingToken` and `feeAmount` is close to `mintAmountToTriggerSwap`, calling `exercise()` with zapping feature will revert due an impossible swap. This causes the user to not get their token when they expected.

A user may want to time his exercise price, which will not always be possible through the zapping feature.

Note: This issue is similar to SR5-oToken [M-01] issue.

Code snippet

The following code snippet from `_zap` explains the issue with `@POC` comments:

```
function _zap(address from, uint256 amount, address recipient,
↪   DiscountExerciseParams memory params)
    internal
    returns (uint256 paymentAmount, address, uint256, uint256)
{
    // ...

    // Fee amount in underlying tokens charged for zapping
    feeAmount += fee;
    if (feeAmount >= minAmountToTriggerSwap) { // @POC: swap when `feeAmount >=
    ↪   minAmountToTriggerSwap`
        uint256 minAmountOut = _getMinAmountOutData(feeAmount,
        ↪   swapProps.maxSwapSlippage, address(oracle));
        /* Approve the underlying token to make swap */
        underlyingToken.approve(swapProps.swapper, feeAmount);
        /* Swap underlying token to payment token (asset) */
        uint256 amountOut = _generalSwap(
            swapProps.exchangeTypes, address(underlyingToken),
            ↪   address(paymentToken), feeAmount, minAmountOut,
            ↪   swapProps.exchangeAddress
        ); // @POC: swap will fail if `underlyingToken` balance is lower than
        ↪   `feeAmount`
        // ...
    }
    // ...
}
```

Recommendation

If the contract is underfunded, the fee swapping should not be executed to allow the user timing his zapping exercise.

The issue can be fixed by adding an `underlyingToken` balance check before executing the fees swap to ensure that the balance is greater than or equal to the `feeAmount`.

A patch is given in Appendix.

Resolution

Byte Masons team: Fixed. Payments to users are prioritized over fees swapping and distribution.

Zigtur: Fix reviewed and approved. Prioritizing payments to users is a good point.

LOW-01 - Lack of decimals checks on most Oracle contracts

Description

Scope:

- AlgebraOracle.sol#L72-L74
- ThenaOracle.sol#L67-L69
- UniswapV3Oracle.sol#L72-L73

Token decimals discrepancies are not handled in the current state of the `DiscountExercise` contract. A finding from a previous security review stated that 18 decimals check for each token should be implemented.

However, only one oracle (BalancerOracle.sol#L80) implements 18 decimals check.

Recommendation

Consider implementing 18 decimals check on each token in `AlgebraOracle`, `ThenaOracle` and `UniswapV3Oracle`.

Resolution

Byte Masons team: Fixed.

Zigtur: Fix reviewed and approved.

LOW-02 - Zapping may fail during high volatility periods

Description

Scope:

- DiscountExercise.sol#L223
- SwapHelper.sol#L89-L97

When fees are swapped during a zapping operation, the `minAmountOut` value is calculated from an oracle price and a `maxSwapSlippage` value set by the administrator.

A too small `maxSwapSlippage` value will lead to zapping denial of service during high volatility periods, leading to users not being able to precisely time their exercise.

Note: a too high `maxSwapSlippage` value will lead to a loss of funds for the protocol due to price manipulation.

Recommendation

The `maxSwapSlippage` value should be chosen precisely to be convenient for users while limiting the impact of price manipulation on the swap.

Moreover, I suggest an off-chain mechanism to monitor `underlyingToken` and `paymentToken` prices and update the `maxSwapSlippage` value when price volatility increases too much.

Resolution

Byte Masons team: Acknowledged.

Zigtur: Acknowledged.

INFO-01 - `PausableUpgradeable` is not initialized in `OptionsToken`

Description

Scope:

- OptionsToken.sol#L61-L63

`OptionsToken` inherits the `PausableUpgradeable` contract. `OptionsToken.initialize` should call the `PausableUpgradeable.__Pausable_init` function.

Note: This has no impact because `PausableUpgradeable.__Pausable_init_unchained` only initializes `paused = false`.

Recommendation

Call `__Pausable_init()` in `OptionsToken.initialize`.

Resolution

Byte Masons team: Fixed.

Zigtur: Fix reviewed and approved.

INFO-02 - Token admin can't be changed without upgrading

Description

Scope:

- OptionsToken.sol#L45
- OptionsToken.sol#L64

The `tokenAdmin` address is defined during the proxy initialization. As there is no setter function to update its value, an upgrade would be needed.

Recommendation

An admin setter may be needed to update the `tokenAdmin` address.

Resolution

Byte Masons team: Acknowledged.

Zigtur: Acknowledged.

INFO-03 - Incorrect Option Token Exercise flow description

Description

The provided documentation describes the flow of an Option Token Exercise. It indicates the following:

> 1. The user approves amount of Options Tokens they wish to spend
> 2. …
> 3. OptionsToken validates the exercise contract, decodes the parameters for the exercise function on the chosen exercise contract

However, the user does not approve the amount they wish to spend and `OptionsToken` contract does not "decode the parameters".

Recommendation

Consider fixing the documentation.

Resolution

Byte Masons team: Fixed.

Zigtur: Fix reviewed and approved.

INFO-04 - OpenZeppelin dependencies does not point to a specific version in `gitmodules`

Description

The codebase expects OpenZeppelin contracts from version `4`. For example, `PausableUpgradeable` should be located in `contracts/security` in OpenZeppelin/openzeppelin-contracts-upgradeable repository.

However, the `.gitmodules` file doesn't configure the correct branch. The main branch will be cloned in the `lib/` folder, which currently corresponds to version `5`. Foundry will not be able to compile the contracts as the `contracts/security` folder doesn't exist in this version.

Recommendation

Consider configuring `.gitmodules` to clone the correct version of OpenZeppelin contracts.

The following `.gitmodules` file can be used:

```
[submodule "lib/forge-std"]
        path = lib/forge-std
        url = https://github.com/foundry-rs/forge-std
[submodule "lib/solmate"]
        path = lib/solmate
        url = https://github.com/rari-capital/solmate
[submodule "lib/create3-factory"]
        path = lib/create3-factory
        url = https://github.com/zeframlou/create3-factory
[submodule "lib/v3-core"]
        path = lib/v3-core
        url = https://github.com/uniswap/v3-core
[submodule "lib/openzeppelin-contracts-upgradeable"]
        path = lib/openzeppelin-contracts-upgradeable
        url = https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable
        branch = v4.9.6
[submodule "lib/openzeppelin-contracts"]
        path = lib/openzeppelin-contracts
        url = https://github.com/OpenZeppelin/openzeppelin-contracts
        branch = v4.9.6
[submodule "lib/vault-v2"]
    path = lib/vault-v2
    url = https://github.com/Byte-Masons/vault-v2
```

Resolution

Byte Masons team: Fixed.

Zigtur: Fix reviewed and approved.

INFO-05 - `safeApprove` is not used

Description

Scope:

- DiscountExercise.sol#L225
- DiscountExercise.sol#L236

The `DiscountExercise` contract uses the `SafeERC20` library for `IERC20`. Transfers are made through `safeTransfer`.

However the contract uses `approve` instead of `safeApprove`.

Recommendation

Consider using `safeApprove` instead of `approve`.

Resolution

Byte Masons team: Acknowledged.

Zigtur: Acknowledged.

INFO-06 - Lack of NatSpec for return values in `OptionsToken.exercise`

Description

Scope:

- OptionsToken.sol#L94-L108

The `OptionsToken.exercise` function shows NatSpec comments for input parameters.

However, return values are not described.

Recommendation

Consider describing the four returned values in NatSpec comments.

Resolution

Byte Masons team: Fixed.

Zigtur: Fix reviewed and approved.

INFO-07 - Fees swapping is not fair

Description

In `_zap`, fees are swapped from `underlyingToken` to `paymentToken` when the accumulated fees `feeAmount` is greater than or equal to a threshold.

This means that a single user will pay swap gas fees for all previous users that used the zapping feature when the threshold is crossed.

Recommendation

None.

Resolution

Byte Masons team: Acknowledged.

Zigtur: Acknowledged.

Appendix

HIGH-01 - Proof of Concept

The following content can be imported in `test/OptionsToken.POC.t.sol`. Then, start the tests with `forge test --mt test_POC -vvv`.

```
// SPDX-License-Identifier: AGPL-3.0
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "forge-std/console2.sol";

import {FixedPointMathLib} from "solmate/utils/FixedPointMathLib.sol";
import {IERC20} from "oz/token/ERC20/IERC20.sol";
import {ERC1967Proxy} from "oz/proxy/ERC1967/ERC1967Proxy.sol";

import {OptionsToken} from "../src/OptionsToken.sol";
import {DiscountExerciseParams, DiscountExercise, BaseExercise, SwapProps,
↪  ExchangeType} from "../src/exercise/DiscountExercise.sol";
import {TestERC20} from "./mocks/TestERC20.sol";
import {IOracle} from "../src/interfaces/IOracle.sol";
import {BalancerOracle} from "../src/oracles/BalancerOracle.sol";
import {MockBalancerTwapOracle} from "./mocks/MockBalancerTwapOracle.sol";

import {ReaperSwapperMock} from "./mocks/ReaperSwapperMock.sol";

contract OptionsTokenTest is Test {
    using FixedPointMathLib for uint256;

    uint16 constant PRICE_MULTIPLIER = 3000; // 0.5
    uint56 constant ORACLE_SECS = 30 minutes;
    uint56 constant ORACLE_AGO = 2 minutes;
    uint128 constant ORACLE_MIN_PRICE = 1e17;
    uint56 constant ORACLE_LARGEST_SAFETY_WINDOW = 24 hours;
    uint256 constant ORACLE_INIT_TWAP_VALUE = 1e18; // @POC: 1$ value
    uint256 constant ORACLE_MIN_PRICE_DENOM = 10000;
    uint256 constant MAX_SUPPLY = 1e27; // the max supply of the options token &
    ↪  the underlying token
    uint256 constant INSTANT_EXIT_FEE = 500;

    address owner;
    address tokenAdmin;
    address[] feeRecipients_;
    uint256[] feeBPS_;
    OptionsToken optionsToken;
    DiscountExercise exerciser;
    IOracle oracle;
    MockBalancerTwapOracle balancerTwapOracle;
    TestERC20 paymentToken;
    address underlyingToken;
    ReaperSwapperMock reaperSwapper;

    function setUp() public {
        // set up accounts
        owner = makeAddr("owner");
        tokenAdmin = makeAddr("tokenAdmin");

        feeRecipients_ = new address[](2);
        feeRecipients_[0] = makeAddr("feeRecipient");
        feeRecipients_[1] = makeAddr("feeRecipient2");

        feeBPS_ = new uint256[](2);
        feeBPS_[0] = 1000; // 10%
        feeBPS_[1] = 9000; // 90%

        // deploy contracts
        paymentToken = new TestERC20();
        underlyingToken = address(new TestERC20());

        address implementation = address(new OptionsToken());
        ERC1967Proxy proxy = new ERC1967Proxy(implementation, "");
        optionsToken = OptionsToken(address(proxy));
        optionsToken.initialize("TIT Call Option Token", "oTIT", tokenAdmin);
        optionsToken.transferOwnership(owner);

        /* Reaper deployment and configuration */
        uint256 slippage = 500; // 5%
        uint256 minAmountToTriggerSwap = 1e5;

        address[] memory tokens = new address[](2);
        tokens[0] = address(paymentToken);
        tokens[1] = underlyingToken;

        balancerTwapOracle = new MockBalancerTwapOracle(tokens);
        // console.log(tokens[0], tokens[1]);
        oracle = IOracle(new BalancerOracle(balancerTwapOracle, underlyingToken,
        ↪   owner, ORACLE_SECS, ORACLE_AGO, ORACLE_MIN_PRICE));

        reaperSwapper = new ReaperSwapperMock(oracle, address(underlyingToken),
        ↪   address(paymentToken));
        deal(underlyingToken, address(reaperSwapper), 1e27);
        deal(address(paymentToken), address(reaperSwapper), 1e27);

        SwapProps memory swapProps = SwapProps(address(reaperSwapper),
        ↪  address(reaperSwapper), ExchangeType.Bal, slippage);

        exerciser = new DiscountExercise(
            optionsToken,
            owner,
            IERC20(address(paymentToken)),
            IERC20(underlyingToken),
            oracle,
            PRICE_MULTIPLIER,
            INSTANT_EXIT_FEE,
            minAmountToTriggerSwap,
            feeRecipients_,
            feeBPS_,
            swapProps
        );
        deal(underlyingToken, address(exerciser), 1e27);

        // add exerciser to the list of options
        vm.startPrank(owner);
        optionsToken.setExerciseContract(address(exerciser), true);
        vm.stopPrank();

        // set up contracts
        balancerTwapOracle.setTwapValue(ORACLE_INIT_TWAP_VALUE);
        paymentToken.approve(address(exerciser), type(uint256).max);
    }

    function test_POCRedeemExerciseHappyPath() public {
        uint256 amount = 1000e18;

        // mint options tokens
        vm.prank(tokenAdmin);
        optionsToken.mint(address(this), amount);

        // mint payment tokens
        uint256 expectedPaymentAmount =
        ↪  amount.mulWadUp(ORACLE_INIT_TWAP_VALUE.mulDivUp(PRICE_MULTIPLIER,
        ↪  ORACLE_MIN_PRICE_DENOM));
        deal(address(paymentToken), address(this), expectedPaymentAmount);

        uint256 paymentTokenValue = paymentTo-
        ↪  ken.balanceOf(address(this)).mulWadUp(ORACLE_INIT_TWAP_VALUE);
        uint256 underlyingTokenValue =
        ↪  IERC20(underlyingToken).balanceOf(address(this));

        // @POC: logs the values before exercise
        console2.log("BEFORE - total account value = ", (paymentTokenValue +
        ↪  underlyingTokenValue) / 1e18);
        console2.log("BEFORE - total paid value    = ", 0);

        DiscountExerciseParams memory params =
            DiscountExerciseParams({maxPaymentAmount: expectedPaymentAmount,
            ↪  deadline: type(uint256).max, isInstantExit: false}); // @POC:
            ↪  Redeem
        optionsToken.exercise(amount, address(this), address(exerciser),
        ↪  abi.encode(params));

        // @POC: logs the values after exercise
        uint256 newpaymentTokenValue = paymentTo-
        ↪  ken.balanceOf(address(this)).mulWadUp(ORACLE_INIT_TWAP_VALUE);
        underlyingTokenValue = IERC20(underlyingToken).balanceOf(address(this));

        console2.log("AFTER  - total account value = ", (newpaymentTokenValue +
        ↪  underlyingTokenValue) / 1e18);
        console2.log("AFTER  - total paid value    = ", paymentTokenValue/1e18);
        console2.log("AFTER  - profit value        = ", (underlyingTokenValue -
        ↪  paymentTokenValue)/1e18);
    }

    function test_POCZapExerciseHappyPath() public {
        uint256 amount = 1000e18;

        // mint options tokens
        vm.prank(tokenAdmin);
        optionsToken.mint(address(this), amount);

        // mint payment tokens
        uint256 expectedPaymentAmount =
        ↪  amount.mulWadUp(ORACLE_INIT_TWAP_VALUE.mulDivUp(PRICE_MULTIPLIER,
        ↪  ORACLE_MIN_PRICE_DENOM));
        deal(address(paymentToken), address(this), expectedPaymentAmount);


        // @POC: logs the values before exercise
        uint256 paymentTokenValue = paymentTo-
        ↪  ken.balanceOf(address(this)).mulWadUp(ORACLE_INIT_TWAP_VALUE);
        uint256 underlyingTokenValue =
        ↪  IERC20(underlyingToken).balanceOf(address(this));
        console2.log("BEFORE - total account value = ", (paymentTokenValue +
        ↪  underlyingTokenValue) / 1e18);
        console2.log("BEFORE - total paid value    = ", 0);

        // exercise options tokens
        DiscountExerciseParams memory params =
            DiscountExerciseParams({maxPaymentAmount: expectedPaymentAmount,
            ↪  deadline: type(uint256).max, isInstantExit: true});
        optionsToken.exercise(amount, address(this), address(exerciser),
        ↪  abi.encode(params));

        // @POC: logs the values after exercise
        paymentTokenValue = paymentTo-
        ↪  ken.balanceOf(address(this)).mulWadUp(ORACLE_INIT_TWAP_VALUE);
        underlyingTokenValue = IERC20(underlyingToken).balanceOf(address(this));

        console2.log("AFTER  - total account value = ", (paymentTokenValue +
        ↪  underlyingTokenValue) / 1e18);
        console2.log("AFTER  - total paid value    = ", 0);
        console2.log("AFTER  - profit value        = ",
        ↪  (underlyingTokenValue)/1e18);
    }
}
```

HIGH-01 - Fix patch

The following patch can be applied through `git apply` to import the recommended fix.

```diff
diff --git a/src/exercise/DiscountExercise.sol b/src/exercise/DiscountExercise.sol
index b8c556f..552cb0e 100644
--- a/src/exercise/DiscountExercise.sol
+++ b/src/exercise/DiscountExercise.sol
@@ -36,6 +36,7 @@ contract DiscountExercise is BaseExercise, SwapHelper, Pausable
↪  {
     error Exercise__InvalidOracle();
     error Exercise__FeeGreaterThanMax();
     error Exercise__AmountOutIsZero();
+    error Exercise__ZapMultiplierIncompatible();

     /// Events
     event Exercised(address indexed sender, address indexed recipient, uint256
      ↪  amount, uint256 paymentAmount);
@@ -212,7 +213,8 @@ contract DiscountExercise is BaseExercise, SwapHelper,
↪  Pausable {
         returns (uint256 paymentAmount, address, uint256, uint256)
     {
         if (block.timestamp > params.deadline) revert Exercise__PastDeadline();
-        uint256 discountedUnderlying = amount.mulDivUp(multiplier, BPS_DENOM);
+        if (multiplier > BPS_DENOM) revert Exercise__ZapMultiplierIncompatible();
+        uint256 discountedUnderlying = amount.mulDivUp(BPS_DENOM - multiplier,
↪  BPS_DENOM);
         uint256 fee = discountedUnderlying.mulDivUp(instantExitFee, BPS_DENOM);
         uint256 underlyingAmount = discountedUnderlying - fee;
```

MEDIUM-01 - Fix patch

The following patch can be applied through `git apply` to import the recommended fix.

```
diff --git a/src/exercise/DiscountExercise.sol b/src/exercise/DiscountExercise.sol
index b8c556f..96cc212 100644
--- a/src/exercise/DiscountExercise.sol
+++ b/src/exercise/DiscountExercise.sol
@@ -218,8 +218,9 @@ contract DiscountExercise is BaseExercise, SwapHelper,
 ↪  Pausable {

         // Fee amount in underlying tokens charged for zapping
         feeAmount += fee;
+        uint256 balance = underlyingToken.balanceOf(address(this));

-        if (feeAmount >= minAmountToTriggerSwap) {
+        if (feeAmount >= minAmountToTriggerSwap && balance >= feeAmount) {
             uint256 minAmountOut = _getMinAmountOutData(feeAmount,
              ↪  swapProps.maxSwapSlippage, address(oracle));
             /* Approve the underlying token to make swap */
             underlyingToken.approve(swapProps.swapper, feeAmount);
```