Byte Masons - Options Compounder

Audit Report

Version 1.1

Zigtur

July 5, 2024

Byte Masons - Options Compounder - 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 Options Compounder

The Options Compounder uses a flash loan to exercise oTokens rewards and swap to the wanted token, meant to be a separated contract used typically by Reaper Strategy contracts.

Security Assessment Summary

Review commit hash - 6d8c60a

Fixes review commit hash - 07d538f

Deployment chains

- Ethereum Mainnet
- BSC
- Optimism
- Mode

Scope

The following smart contract is in scope of the review:

- OptionsCompounder.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

LOW-01 - `initialBalance` is transferred to caller

Description

Scope:

- OptionsCompounder.sol#L274-L288

The `exerciseOptionAndReturnDebt` function distributes the profit from the exercise to the caller with `paymentToken` assets. Before that, it ensures that the profit made is enough by subtracting the initial balance of the compounder from the current balance.

However, this initial balance is then sent to the caller as part of the `gainInPaymentToken`.

This allows any user to drain the `paymentToken` balance left in the compounder.

Note: The impact is limited as the `OptionsCompounder` contract is not supposed to hold any funds.

```
if (
    (
        (assetBalance < flashloanParams.initialBalance)
            || (assetBalance - flashloanParams.initialBalance) <=
            ↪  (totalAmountToPay + flashloanParams.minPaymentAmount)
    )
) {
    revert OptionsCompounder__FlashloanNotProfitableEnough();
}

/* Protected against underflows by statement above */
gainInPaymentToken = assetBalance - totalAmountToPay; // @POC: initial balance
↪  is not subtracted

/* Approve lending pool to spend borrowed tokens + premium */
IERC20(asset).approve(address(lendingPool), totalAmountToPay);
IERC20(asset).safeTransfer(flashloanParams.sender, gainInPaymentToken);
```

Recommendation

The initial balance should not be sent to the caller if the `OptionsCompounder` is supposed to hold funds.

This can be done by subtracting the initial balance from the `gainInPaymentToken` amount.

A patch is given in Appendix.

Resolution

Byte Masons team: Fixed.

Zigtur: Fix reviewed and approved.

LOW-02 - Precise profit in `exerciseOptionAndReturnDebt` will revert

Description

Scope:

- OptionsCompounder.sol#L277

`exerciseOptionAndReturnDebt` reverts if the profit after exercise is not sufficient. This is made through the following check:

```
if (
    (
        (assetBalance < flashloanParams.initialBalance)
            || (assetBalance - flashloanParams.initialBalance) <=
            ↪  (totalAmountToPay + flashloanParams.minPaymentAmount)
    )
) {
    revert OptionsCompounder__FlashloanNotProfitableEnough();
}
```

The second condition will be true when the profit (`assetBalance - initialBalance`) is lower than **or equal to** the expected amount (`totalAmountToPay + flashloanParams.minPaymentAmount`).

This is incorrect as if the profit is equal to the expected amount, the function will revert with error `OptionsCompounder__FlashloanNotProfitableEnough` .

Recommendation

Replace `<=` with `<` to allow the profit to be equal to the expected amount.

A patch is given in Appendix.

Resolution

Byte Masons team: Fixed.

Zigtur: Fix reviewed and approved.

LOW-03 - `exerciseOptionAndReturnDebt` may fail during high volatility periods

Description

Scope:

- OptionsCompounder.sol#L258

When underlying tokens are swapped after an exercise, 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 denial of service during high volatility periods, leading users to not being able to precisely time their exercise through the `OptionsCompounder`.

Note: a too high `maxSwapSlippage` value could lead to a loss of funds for the user.

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 - `_generalSwap` return value is not checked

Description

Scope:

- OptionsCompounder.sol#L264

`_generalSwap` function is used to abstract the swapping call to the `ReaperSwapper`.

In case of a failing swap (`minAmountOut` is not met for example), the `ReaperSwapper` and `_generalSwap` will not revert and will return `0`.

Because `_generalSwap` return value is not checked, the execution flow will continue.

Note: the impact of this issue is null as profitability checks are made after.

Recommendation

Consider ensuring that the value returned by `_generalSwap` is not zero.

Resolution

Byte Masons team: Fixed.

Zigtur: Fix reviewed and approved.

INFO-02 - Approvals are not reset to zero in `exerciseOptionAndReturnDebt`

Description

Scope:

- OptionsCompounder.sol#L250-L252
- OptionsCompounder.sol#L261-L264

Multiple `approve` calls are made to ensure that the tokens can be pulled from the Compounder before exercising and swapping.

However, approvals are not reset to zero after these operations.

Recommendation

Consider reseting the approvals to zero after the `exercise` and `_generalSwap` calls.

Resolution

Byte Masons team: Fixed.

Zigtur: Fix reviewed and approved.

INFO-03 - `exerciseOptionAndReturnDebt` doesn't comply with Solidity style guide

Description

Scope:

- OptionsCompounder.sol#L226

According to Solidity style guide, a private function name should be prefixed by an underscore.

However, `exerciseOptionAndReturnDebt` doesn't comply with this style guide.

Recommendation

Rename `exerciseOptionAndReturnDebt` into `_exerciseOptionAndReturnDebt`.

Resolution

Byte Masons team: Fixed.

Zigtur: Fix reviewed and approved.

INFO-04 - `MIN_NR_OF_FLASHLOAN_ASSETS` variable name is misleading

Description

Scope:

- OptionsCompounder.sol#L40
- OptionsCompounder.sol#L207-L210

The `MIN_NR_OF_FLASHLOAN_ASSETS` seems to indicate "minimum number of flashloan assets".

However, it is used to check that the number of assets is **not greater than this threshold**.

This is misleading as it is used as a **maximum** and not as a minimum.

Recommendation

Rename `MIN_NR_OF_FLASHLOAN_ASSETS` into `MAX_NR_OF_FLASHLOAN_ASSETS`.

Resolution

Byte Masons team: Fixed.

Zigtur: Fix reviewed and approved.

INFO-05 - Multiple access control checks are done in `initialize`

Scope:

- OptionsCompounder.sol#L81-L86

The `initialize` function calls several `public` setters (`setOptionToken` , `setOracle` , ...).

Each of them will execute an access control check to ensure that the caller is the owner.

This is highly inefficient as the same access control check will be executed in each call.

Recommendation

Internal functions without access control could be implemented for each setter. For example, implement `_setOptionToken` which can be called by both `setOptionToken` and `initialize` .

A patch is given in Appendix.

Another way to fix the issue (less recommended) is that `initialize` can directly initialize the storage values.

Resolution

Byte Masons team: Fixed.

Zigtur: Fix reviewed and approved.

INFO-06 - `deadline` should be `block.timestamp`

Description

- OptionsCompounder.sol#L245

The `deadline` used to call `exercise` is set to `type(uint256).max`.

```
bytes memory exerciseParams =
    abi.encode(DiscountExerciseParams({maxPaymentAmount: amount, deadline:
    ↪  type(uint256).max, isInstantExit: false}));
```

The `exercise` is supposed to be executed in the same block.

Recommendation

Consider setting `deadline = block.timestamp`.

Resolution

Byte Masons team: Fixed.

Zigtur: Fix reviewed and approved.

INFO-07 - Incorrect comment in `exerciseOptionAndReturnDebt`

Description

Scope:

- OptionsCompounder.sol#L249

The comment indicates that an approval for spending option token is made.

```
/* Approve spending option token */
IERC20(asset).approve(flashloanParams.exerciserContract, amount);
```

However, this approval is not for option token but for payment token.

Recommendation

Consider fixing the comment with the following:

```
/* Approve spending payment token */
IERC20(asset).approve(flashloanParams.exerciserContract, amount);
```

Resolution

Byte Masons team: Fixed.

Zigtur: Fix reviewed and approved.

INFO-08 - `PERCENTAGE` variable is not used

Description

Scope:

- OptionsCompounder.sol#L41

The `PERCENTAGE` constant variable is set but is never used in the `OptionsCompounder` contract.

Recommendation

Delete the `PERCENTAGE` variable.

Resolution

Byte Masons team: Fixed.

Zigtur: Fix reviewed and approved.

Appendix

LOW-01 - Fix patch

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

```
diff --git a/src/OptionsCompounder.sol b/src/OptionsCompounder.sol
index a388ad7..fa82a9e 100644
--- a/src/OptionsCompounder.sol
+++ b/src/OptionsCompounder.sol
@@ -281,7 +281,7 @@ contract OptionsCompounder is IFlashLoanReceiver,
↪  OwnableUpgradeable, UUPSUpgrad
             }

             /* Protected against underflows by statement above */
-            gainInPaymentToken = assetBalance - totalAmountToPay;
+            gainInPaymentToken = assetBalance - totalAmountToPay -
↪  flashloanParams.initialBalance;

             /* Approve lending pool to spend borrowed tokens + premium */
             IERC20(asset).approve(address(lendingPool), totalAmountToPay);
```

LOW-02 - Fix patch

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

```
diff --git a/src/OptionsCompounder.sol b/src/OptionsCompounder.sol
index a388ad7..4cbc227 100644
--- a/src/OptionsCompounder.sol
+++ b/src/OptionsCompounder.sol
@@ -274,7 +274,7 @@ contract OptionsCompounder is IFlashLoanReceiver,
↪  OwnableUpgradeable, UUPSUpgrad
             if (
                 (
                     (assetBalance < flashloanParams.initialBalance)
-                        || (assetBalance - flashloanParams.initialBalance) <=
↪  (totalAmountToPay + flashloanParams.minPaymentAmount)
+                        || (assetBalance - flashloanParams.initialBalance) <
↪  (totalAmountToPay + flashloanParams.minPaymentAmount)
                 )
             ) {
                 revert OptionsCompounder__FlashloanNotProfitableEnough();
```

INFO-05 - Fix patch

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

```
diff --git a/src/OptionsCompounder.sol b/src/OptionsCompounder.sol
index a388ad7..ffb87d8 100644
--- a/src/OptionsCompounder.sol
+++ b/src/OptionsCompounder.sol
@@ -78,12 +78,12 @@ contract OptionsCompounder is IFlashLoanReceiver,
↪  OwnableUpgradeable, UUPSUpgrad
         initializer
     {
         __Ownable_init();
-        setOptionToken(_optionsToken);
+        _setOptionToken(_optionsToken);
         _setSwapProps(_swapProps);
-        setOracle(_oracle);
-        setSwapper(_swapper);
+        _setOracle(_oracle);
+        _setSwapper(_swapper);
         flashloanFinished = true;
-        setAddressProvider(_addressProvider);
+        _setAddressProvider(_addressProvider);
         __UUPSUpgradeable_init();
         _clearUpgradeCooldown();
     }
@@ -97,10 +97,7 @@ contract OptionsCompounder is IFlashLoanReceiver,
↪  OwnableUpgradeable, UUPSUpgrad
      * @param _optionToken - address of option token contract
      */
     function setOptionToken(address _optionToken) public onlyOwner {
-        if (_optionToken == address(0)) {
-            revert OptionsCompounder__ParamHasAddressZero();
-        }
-        optionsToken = IOptionsToken(_optionToken);
+        _setOptionToken(_optionToken);
     }

     function setSwapProps(SwapProps memory _swapProps) external override
     ↪  onlyOwner {
@@ -108,25 +105,15 @@ contract OptionsCompounder is IFlashLoanReceiver,
↪  OwnableUpgradeable, UUPSUpgrad
     }

     function setOracle(IOracle _oracle) public onlyOwner {
-        if (address(_oracle) == address(0)) {
```

```
-            revert OptionsCompounder__ParamHasAddressZero();
-        }
-        oracle = _oracle;
+        _setOracle(_oracle);
     }

     function setSwapper(address _swapper) public onlyOwner {
-        if (_swapper == address(0)) {
-            revert OptionsCompounder__ParamHasAddressZero();
-        }
-        swapper = _swapper;
+        _setSwapper(_swapper);
     }

     function setAddressProvider(address _addressProvider) public onlyOwner {
-        if (_addressProvider == address(0)) {
-            revert OptionsCompounder__ParamHasAddressZero();
-        }
-        addressProvider = ILendingPoolAddressesProvider(_addressProvider);
-        lendingPool = ILendingPool(addressProvider.getLendingPool());
+        _setAddressProvider(_addressProvider);
     }

     /**
@@ -325,6 +312,35 @@ contract OptionsCompounder is IFlashLoanReceiver,
↪  OwnableUpgradeable, UUPSUpgrad
         _clearUpgradeCooldown();
     }

+    function _setOptionToken(address _optionToken) internal {
+        if (_optionToken == address(0)) {
+            revert OptionsCompounder__ParamHasAddressZero();
+        }
+        optionsToken = IOptionsToken(_optionToken);
+    }
+
+    function _setOracle(IOracle _oracle) internal {
+        if (address(_oracle) == address(0)) {
+            revert OptionsCompounder__ParamHasAddressZero();
+        }
+        oracle = _oracle;
+    }
+
+    function _setSwapper(address _swapper) internal {
+        if (_swapper == address(0)) {
+            revert OptionsCompounder__ParamHasAddressZero();
```

```
+        }
+        swapper = _swapper;
+    }
+
+    function _setAddressProvider(address _addressProvider) internal {
+        if (_addressProvider == address(0)) {
+            revert OptionsCompounder__ParamHasAddressZero();
+        }
+        addressProvider = ILendingPoolAddressesProvider(_addressProvider);
+        lendingPool = ILendingPool(addressProvider.getLendingPool());
+    }
+
     /**
      * Getters **********************************
      */
```