code4rena

SOFA.org Automator 2.0

Smart Contract Security Assessment

Version 1.0

Audit dates: Jan 08 — Jan 14, 2025

Audited by: peakbolt
spicymeatball

Contents

1. Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2. Executive Summary

2.1 About SOFA.org Automator

In our mission to democratize option trading for the masses, the SOFA team has created Automator as an intuitive way for users to earn strategy yields passively. By leveraging a community of strategy managers, collectively known as Optivisors, users are able to participate in their favorite managed strategies in return for sharing a small upside profit fee.

This win-win model will provide decentralized and on-chain access for depositors to earn a variety of high-quality returns with an evolving base of community-recognized Optivisors.

2.2 Scope

Repository	sofa-org/sofa-protocol
Commit Hash	01cb2b51da23bf40d9a9d5bd445c4acb1dbb05c0

2.3 Audit Timeline

DATE	EVENT
Jan 08, 2025	Audit start
Jan 14, 2025	Audit end
Jan 17, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	4
Informational	0
Total Issues	4

3. Findings Summary

ID	DESCRIPTION	STATUS
L-1	Missing validation for `createAutomator()`	Resolved

L-2	\`harvest()\` could fail to collect protocol fee if automator owner is blacklisted	Resolved
L-3	Negative fee calculation for \`burnProducts()\` should not favor the Automator owner	Acknowledged
L-4	Users could waste credit on \`createAutomator()\` if the specified collateral is unsupported	Resolved

4. Findings

4.1 Low Risk

A total of 4 low risk findings were identified.

[L-1] Missing validation for `createAutomator()`

Severity: Low	Status: Resolved

Context:

- AutomatorFactory.sol#L35-L50

Description:

`createAutomator()` should validate both `feeRate` and `maxPeriod` are valid and acceptable.

Otherwise, if `feeRate > 1e18 - tradingFeeRate`, it will cause `burnProducts()` to take more fees than the profit.

Also if `maxPeriod` has to be an acceptable value. Otherwise owner can create an Automator that has a significantly long withdrawal timelock.

```
    function createAutomator(
>>>     uint256 feeRate,
>>>     uint256 maxPeriod,
        address collateral
    ) external returns (address) {
        require(credits[_msgSender()] > 0, "AutomatorFactory:
insufficient credits");
        credits[_msgSender()] -= 1;
        bytes32 salt = keccak256(abi.encodePacked(_msgSender(),
collateral));
        address _automator = Clones.cloneDeterministic(automator, salt);
        AAVEAutomatorBase(_automator).initialize(_msgSender(),
collateral, feeRate, maxPeriod);
        require(getAutomator[_msgSender()][collateral] == address(0),
"AutomatorFactory: automator already exists");
        getAutomator[_msgSender()][collateral] = _automator;
        automators.push(_automator);
        emit AutomatorCreated(_msgSender(), collateral, _automator,
feeRate, maxPeriod);
        return _automator;
    }
```

Recommendation: Consider validating that `feeRate` is within a reasonable percentage and validate that `maxPeriod` is within an acceptable period.

SOFA.org: Fixed with [@86abeaaa75804..](#)

Zenith: Verified.

[L-2] `harvest()` could fail to collect protocol fee if automator owner is blacklisted

Severity: Low	Status: Resolved

Context:

- AAVEAutomatorBase.sol#L281-L294

Description: `AAVEAutomatorBase` has a `harvest()` that will perform the fee collection for both protocol and the automator owner at the same time.

However, if `owner()` is blocked by the underlying token of the aToken (collateral), `harvest()` will always fail as the `pool.withdraw()` will fail. That means protocol will not be able to collect the protocol fee.

```
    function harvest() external nonReentrant {
        require(totalFee > 0 || totalProtocolFee > 0, "Automator: zero
fee");
        uint256 feeAmount = 0;
        uint256 protocolFeeAmount = 0;
        if (totalFee > 0) {
            feeAmount = pool.withdraw(address(collateral),
uint256(totalFee), owner());
            totalFee = 0;
        }
        if (totalProtocolFee > 0) {
            protocolFeeAmount = pool.withdraw(address(collateral),
totalProtocolFee, IAutomatorFactory(factory).feeCollector());
            totalProtocolFee = 0;
        }
        emit FeeCollected(_msgSender(), feeAmount, totalFee,
protocolFeeAmount, totalProtocolFee);
    }
```

Recommendation: Consider adding a flag to `harvest()` that only performs collection of protocol fee.

SOFA.org: Fixed with @486ebedc2305...

Zenith: Verified. Resolved with a new `harvestByProtocol()` function.

[L-3] Negative fee calculation for `burnProducts()` should not favor the Automator owner

Severity: Low	Status: Acknowledged

Context:

- RCHAutomatorBase.sol#L261

Description: `burnProducts()` will compute and record the profit/loss from the purchased vault products after burning them.

When a profit is made, both protocol and automator owner will receive fees from a portion of the profit.

In the case of a loss, a negative fee is applied for the automator owner (see below) while no protocol fee will be taken.

```
                if (earned < _positions[id]) {
                        //@audit negative fee for automator owner
>>>                 fee -= int256((_positions[id] - earned) * feeRate /
1e18);
                }
```

In solidity, division rounds towards zero for negative numbers (e.g. -5 / 2 = -2).

So the negative fee calculation will cause the fee to slightly favors the Automator owner instead of the users, which is not consistent with the profit scenario.

Recommendation: Consider doing a floor after negative division.

```
function divDown(int256 a, int256 b) public pure returns (int256) {
    int256 quotient = a / b;
    int256 remainder = a % b;

    // If there is a remainder and result is negative, subtract 1 to
round down
    if (remainder != 0 && ((a < 0) != (b < 0))) {
        quotient -= 1;
    }

    return quotient;
}
```

SOFA.org: Acknowledged. To save gas and simplify calculations, it is acceptable for the Automator owner to gain some advantages.

[L-4] Users could waste credit on `createAutomator()` if the specified collateral is unsupported

Severity: Low | Status: Resolved

Context:

- AAVEAutomatorBase.sol#L143-L158
- RCHAutomatorBase.sol#L144-L158

Description: `AutomatorFactory.createAutomator()` will call `initialize()` in both `RCHAutomatorBase` and `AAVEAutomatorBase` to setup the Automator contract with the parameters.

However, it fails to validate that the `collateral_` parameter is a supported asset. This could cause the user to waste credits for `createAutomator()` if the `collateral_` is incorrect as it will not revert in that scenario.

For `AAVEAutomatorBase`, `intialize()` will call pool.getReserveData(), which does not check that the `collateral_` exists, so it is possible for it to return `address(0)` for an unsupported collateral.

```
contract RCHAutomatorBase is ERC1155Holder, ERC20, ReentrancyGuard {
   ...
    function initialize(
        address owner_,
        address collateral_,
        uint256 feeRate_,
        uint256 maxPeriod_
    ) external {
        require(_msgSender() == factory, "Automator: forbidden");
        _owner = owner_;
>>>     collateral = IERC20(collateral_);
        feeRate = feeRate_;
        maxPeriod = maxPeriod_;
        collateral.safeApprove(address(zenRCH), type(uint256).max);
        uint256 salt = uint256(uint160(address(this))) % 65536;
        symbol_ = string(abi.encodePacked("at",
IERC20Metadata(address(collateral)).symbol(), "_", salt.toString()));
    }
```

```
contract AAVEAutomatorBase is ERC1155Holder, ERC20, ReentrancyGuard {
    ...
    function initialize(
        address owner_,
        address collateral_,
        uint256 feeRate_,
        uint256 maxPeriod_
    ) external {
        require(_msgSender() == factory, "Automator: forbidden");
        _owner = owner_;
        collateral = IERC20(collateral_);
        feeRate = feeRate_;
        maxPeriod = maxPeriod_;
>>>     aToken = IAToken(pool.getReserveData(collateral_).aTokenAddress);
        collateral.safeApprove(address(pool), type(uint256).max);
        uint256 salt = uint256(uint160(address(this))) % 65536;
        symbol_ = string(abi.encodePacked("at",
IERC20Metadata(address(collateral)).symbol(), "_", salt.toString()));
    }
```

Recommendation: In the case of `RCHAutomatorBase`, update `initialize()` to validate that `collateral == RCH` since its the only asset that can be used to mint zenRCH.

For `AAVEAutomatorBase`, a straightforward solution is to check `aToken != address(0)` in `initialize()`.

SOFA.org: Fixed in @486ebedc2305...

Zenith: Verified. Resolved with validation on `collateral_`.