Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

2 Audit Methodology

3 Project Overview

3.1 Project Introduction

3.2 Vulnerability Information

4 Code Overview

4.1 Contracts Description

4.2 Visibility Description

4.3 Vulnerability Summary

5 Audit Result

6 Statement

1 Executive Summary

On 2023.10.08, the SlowMist security team received the Struct Finance team's security audit application for Struct Finance, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

Struct Finance is a DeFi platform offering tailored structured financial products to cater to the distinct risk-return profiles of retail and institutional investors. Our innovative Tranching mechanism, the first in our planned lineup of product offerings, enables diversified investment opportunities across a wide array of markets through interest rate vaults.

The core contracts include Product module, Factory module, SP Token module, YieldSource module and GAC module.

1. Product module

The fixed and enhanced yield contract containing core functions of the protocol: depositing, claiming excess, and withdrawals. Interacts with the Yield Source contract to supply and remove liquidity from the underlying protocol.

2. Factory module

Creates new FEYProduct contracts (using clones) with user-defined configurations such as tranche start and end times, tranche tokens, fixed rate, etc. It also has methods to whitelist tokens and token pairs.

3. SP Token module

An ERC1155 standard token is used to represent user's position in each tranche, Every tranche is assigned a unique ID in the StructSPToken and is minted at a 1:1 ratio for deposits.

Note that FEYProduct and FEYProductFactory components have a new implementation for each integration (platform). The implementation of StructSPToken remains the same, but there will be a new deployment for each integration.

4. YieldSource module

Helps the FEYProduct contract to interact with the external protocols.

5. Global Access Control (GAC) module

Enforces the access control methods for the entire protocol implemented by the `GACManaged` contract which inherits the GlobalAccessControl for the roles. All the core contracts will inherit the GACManaged contract.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Compatibility issue between supplyTokens function and deflationary tokens	Design Logic Audit	Medium	Fixed

NO	Title	Category	Level	Status
N2	Emergency Withdrawal Deficiency	Design Logic Audit	High	Fixed
N3	Risk of protocol denial of service by manipulating `ammRate`	Design Logic Audit	Critical	Fixed
N4	Missing of event records	Others	Suggestion	Fixed
N5	Compatibility issue between `_increaseAllowanceAndSwap` operation and deflationary tokens	Design Logic Audit	Medium	Acknowledged
N6	Compatibility issue between redeemTokens function and deflationary tokens	Design Logic Audit	Medium	Fixed
N7	Gas optimization	Gas Optimization Audit	Suggestion	Acknowledged
N8	Risk of excessive privilege	Authority Control Vulnerability Audit	Medium	Acknowledged
N9	Redundant function	Gas Optimization Audit	Suggestion	Fixed
N10	There is a flaw in the amountInMax calculation when tranche tokens are allocated	Others	Low	Fixed

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/struct-defi/struct-core

commit: bbf16ca72dd7f271aa4daabd3ed402001574f7d1

Fixed Version:

https://github.com/struct-defi/struct-core

commit: 2582b1cc3000c73b30a6f419de598583f989a945

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

FEYAutoPoolProduct			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	-
invest	External	Can Modify State	nonReentrant gacPausable
removeFundsFromLP	External	Can Modify State	nonReentrant gacPausable
processRedemption	External	Can Modify State	nonReentrant gacPausable
setSlippage	External	Can Modify State	onlyRole
getTokenRate	Public	-	-
_depositToLP	Private	Can Modify State	-
_chargeFee	Private	Can Modify State	-
_allocateToTranches	Private	Can Modify State	-
getSrFrFactor	Public	-	-

FEYProduct			
Function Name	Visibility	Mutability	Modifiers
_checkSpAndTrancheTokenBalances	Private	-	-
deposit	External	Payable	nonReentrant gacPausable
depositFor	External	Payable	nonReentrant gacPausable onlyRole
claimExcess	External	Can Modify State	nonReentrant gacPausable
withdraw	External	Can Modify State	nonReentrant gacPausable
claimExcessAndWithdraw	External	Can Modify State	nonReentrant gacPausable
rescueTokens	External	Can Modify State	onlyRole
forceUpdateStatusToWithdrawn	Public	Can Modify State	-
getUserInvestmentAndExcess	External	-	-
getUserTotalDeposited	External	-	-
getInvestorDetails	External	-	-
getCurrentState	External	-	-
getTrancheInfo	External	-	-
getTrancheConfig	External	-	-
getProductConfig	External	-	-
_deposit	Internal	Can Modify State	validateBalances
_claimExcess	Private	Can Modify State	-
_transferTokens	Private	Can Modify State	-
_calculateUserShareAndTransfer	Internal	Can Modify State	-

FEYProduct			
_forceUpdateStatusToWithdrawn	Internal	Can Modify State	-
<Receive Ether>	External	Payable	-
initialize	External	Can Modify State	-
invest	External	Can Modify State	-
removeFundsFromLP	External	Can Modify State	-

FEYAutoPoolProductFactory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
totalProducts	External	-	-
createProduct	External	Payable	gacPausable
setPoolStatus	External	Can Modify State	onlyRole
setYieldSource	External	Can Modify State	onlyRole
isMintActive	External	-	-
isTransferEnabled	External	-	-
_deployProduct	Private	Can Modify State	-
_makeInitialDeposit	Private	Can Modify State	-
_validateProductConfig	Private	-	-
_validatePool	Private	-	-
_getTrancheCapacityValues	Private	-	-
_getInitialDepositValueUSD	Private	-	-

FEYFactoryConfigurator			
Function Name	Visibility	Mutability	Modifiers
setStructPriceOracle	External	Can Modify State	onlyRole
setMinimumTrancheDuration	External	Can Modify State	onlyRole
setMaximumTrancheDuration	External	Can Modify State	onlyRole
setManagementFee	External	Can Modify State	onlyRole
setPerformanceFee	External	Can Modify State	onlyRole
setLeverageThresholdMinCap	External	Can Modify State	onlyRole
setLeverageThresholdMaxCap	External	Can Modify State	onlyRole
setTokenStatus	External	Can Modify State	onlyRole
setTrancheCapacity	External	Can Modify State	onlyRole
setMaxFixedRate	External	Can Modify State	onlyRole
setFEYProductImplementation	External	Can Modify State	onlyRole
setMinimumDepositValueUSD	External	Can Modify State	onlyRole

AutoPoolYieldSource			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
supplyTokens	External	Payable	gacPausable nonReentrant onlyRole
recompoundRewards	Public	Can Modify State	gacPausable onlyRole
queueForRedemption	External	Can Modify State	gacPausable nonReentrant onlyRole
redeemTokens	External	Can Modify State	gacPausable nonReentrant onlyRole
sharesToTokens	External	-	-

AutoPoolYieldSource			
setMaxIterations	External	Can Modify State	onlyRole
harvestRewards	External	Can Modify State	onlyRole
rescueTokens	External	Can Modify State	onlyRole
updateFarmInfo	External	Can Modify State	onlyRole
emergencyWithdrawFromFarm	Public	Can Modify State	onlyRole
emergencyWithdrawFromAutoPool	Public	Can Modify State	onlyRole
emergencyWithdrawAndRescue	External	Can Modify State	onlyRole
_depositAPTToFarm	Internal	Can Modify State	-
_recompoundRewards	Internal	Can Modify State	-
_harvestRewards	Internal	Can Modify State	-
_increaseAllowanceAndSwap	Internal	Can Modify State	-
_recompoundRewards	Internal	Can Modify State	-
_getTokenRate	Internal	-	-
_rescueTokens	Internal	Can Modify State	-
_updateFarmInfo	Internal	Can Modify State	-
getRoundInfo	External	-	-

YieldSource			
Function Name	Visibility	Mutability	Modifiers
_tokenToShares	Internal	-	-

YieldSource			
_sharesToTokens	Internal	-	-
<Receive Ether>	External	Payable	-

StructERC1155			
Function Name	Visibility	Mutability	Modifiers
uri	Public	-	-
setApprovalForAll	Public	Can Modify State	-
safeTransferFrom	Public	Can Modify State	-
safeBatchTransferFrom	Public	Can Modify State	-
balanceOfBatch	Public	-	-
supportsInterface	Public	-	-
_mint	Internal	Can Modify State	-
_batchMint	Internal	Can Modify State	-
_batchBurn	Internal	Can Modify State	-
_burn	Internal	Can Modify State	-
_beforeTokenTransfer	Internal	Can Modify State	-
_asSingletonArray	Private	-	-

StructSPToken			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
safeTransferFrom	Public	Can Modify State	gacPausable
safeBatchTransferFrom	Public	Can Modify State	gacPausable
mint	Public	Can Modify State	gacPausable onlyRole

StructSPToken			
mintBatch	Public	Can Modify State	gacPausable onlyRole
burn	Public	Can Modify State	onlyRole gacPausable
burnBatch	Public	Can Modify State	onlyRole gacPausable
_beforeTokenTransfer	Internal	Can Modify State	gacPausable
setURI	External	Can Modify State	onlyRole
setFeyProductFactory	External	Can Modify State	onlyRole
_setURI	Internal	Can Modify State	-
uri	Public	-	-
supportsInterface	Public	-	-

Rewarder			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
allocateRewards	External	Can Modify State	nonReentrant gacPausable onlyRole
claimRewards	External	Can Modify State	nonReentrant gacPausable
rescueTokens	External	Can Modify State	nonReentrant onlyRole
calculateRewards	Public	-	-
_calculateRewardForTranche	Internal	-	-
_validateAllocateRewards	Internal	-	-
getAllocationDetails	Public	-	-

4.3 Vulnerability Summary

[N1] [Medium] Compatibility issue between supplyTokens function and deflationary tokens

Category: Design Logic Audit

Content

In the AutoPoolYieldSource contract, the supplyTokens function will transfer srToken and jrToken from PRODUCT, which are used to deposit to autoPoolVault and update the corresponding shares.

Unfortunately, deflationary tokens are not handled in the function. When srToken and jrToken are deflationary tokens, the number of tokens received by the AutoPoolYieldSource contract will be less than the `_amountAIn` and `_amountBIn` deposited to autoPoolVault. This will prevent the autoPoolVault contract from successfully transferring srToken and jrToken, and ultimately result in PRODUCT being unable to perform normal invest operations.

Code location: contracts/protocol/yield-sources/AutoPoolYieldSource.sol#L221

```
    function supplyTokens(uint256 _amountAIn, uint256 _amountBIn)
        external
        payable
        gacPausable
        nonReentrant
        onlyRole(PRODUCT)
        returns (uint256 _investedTokenA, uint256 _investedTokenB)
    {
        ...
        tokenA.safeTransferFrom(msg.sender, address(this), _amountAIn);
        tokenB.safeTransferFrom(msg.sender, address(this), _amountBIn);
        ...
        (, _investedTokenA, _investedTokenB) = autoPoolVault.deposit(_amountAIn,
 _amountBIn);
        ...
    }
```

Solution

It is recommended that the supplyTokens function obtains the balance difference between srToken and jrToken tokens before and after the transfer as `_amountAIn` and `_amountBIn` to deposit to autoPoolVault.

Consider the following fixes:

```
    function supplyTokens(uint256 _amountAIn, uint256 _amountBIn)
        ...
    {
        ...
        uint256 _amountAInBefore = tokenA.balanceOf(address(this));
        uint256 _amountBInBefore = tokenB.balanceOf(address(this));

        tokenA.safeTransferFrom(msg.sender, address(this), _amountAIn);
        tokenB.safeTransferFrom(msg.sender, address(this), _amountBIn);

        _amountAIn = tokenA.balanceOf(address(this)) - _amountAInBefore;
        _amountBIn = tokenB.balanceOf(address(this)) - _amountBInBefore;
        (, _investedTokenA, _investedTokenB) = autoPoolVault.deposit(_amountAIn,
_amountBIn);
        ...
    }
```

Status

Fixed

[N2] [High] Emergency Withdrawal Deficiency

Category: Design Logic Audit

Content

In the AutoPoolYieldSource contract, the GOVERNANCE role can perform emergency withdrawals from farm and autoPoolVault contracts through the emergencyWithdrawAndRescue, emergencyWithdrawFromAutoPool and emergencyWithdrawFromFarm functions. However, in the emergencyWithdrawFromAutoPool function only the value of `totalAutoPoolShareTokens` is cleared and not the value of `totalShares`, which will result in an error at the step of calculating the number of shares on the next investment, and the contract won't work properly.

Code Location: contracts/protocol/yield-sources/AutoPoolYieldSource.sol#L431-434

```
    function emergencyWithdrawFromAutoPool() public onlyRole(GOVERNANCE) {
        totalAutoPoolShareTokens = 0;
        autoPoolVault.emergencyWithdraw();
    }
```

Solution

It is recommended that the value of totalShares is also cleared when making an emergency withdrawal. In the meantime, make sure the product is in a suspended state.

Status

Fixed

[N3] [Critical] Risk of protocol denial of service by manipulating `ammRate`

Category: Design Logic Audit

Content

In the FEYAutoPool protocol, it compares the token price rate obtained from the Chainlink oracle and Traderjoe LBQuoter to check whether there is a large deviation in the price. The next step will be taken when the price check passes, otherwise revert will be performed.

When the price is obtained through findBestPathFromAmountIn of the Traderjoe LBQuoter contract, it will check all the pools of v1, v2, and v2.1 to obtain the largest amountOut during swap as the optimal price. However, it should be noted that the amountIn parameter passed in when calling the findBestPathFromAmountIn function of the FEYAutoPool protocol is only 1 token amount. This would allow a malicious user to cost-effectively create a low-liquidity but high-price pair in some version of Traderjoe DEX, causing the ammRate to deviate far from the chainlinkRate. by making the validPrice false in this way, it would never pass the subsequent price checking, causing the The FEYAutoPool protocol is at risk of being denied service.

Code location:

contracts/protocol/yield-sources/AutoPoolYieldSource.sol#L545-L547

contracts/protocol/yield-sources/AutoPoolYieldSource.sol#L636-L637

```
    function _getTokenRate(address _asset1, address _asset2, address[] memory _path)
        internal
        view
        returns (bool, uint256, uint256, uint256)
    {
        ...
        ILBQuoter.Quote memory quote =
            lbQuoter.findBestPathFromAmountIn(_path, uint128(10 **
IERC20Metadata(_asset1).decimals()));
        ...
    }

    function _increaseAllowanceAndSwap(uint256 _amount, IERC20Metadata _token,
  address[] memory _path) internal {
        ...
        (bool _validPrice, uint256 _exchangeRate,,) = _getTokenRate(address(_token),
  _path[_path.length - 1], _path);

        require(_validPrice, Errors.PFE_RATEDIFF_EXCEEDS_DEVIATION);
        ...
   }
```

contracts/protocol/libraries/helpers/Helpers.sol#L265-L296

```
    function getTrancheTokenRateV2(
        IStructPriceOracle _structPriceOracle,
        address[] storage _path,
        ILBQuoter _lbQuoter,
        uint256 _amountOut
    ) external view returns (bool, uint256, uint256, uint256) {
        ...
        if (_amountOut == 0) {
            quote = _lbQuoter.findBestPathFromAmountIn(_path, uint128(10 **
  IERC20Metadata(_path[0]).decimals()));
            // no need to divide by the amountIn because it is equivalent to WAD
            _ammRate = tokenDecimalsToWei(IERC20Metadata(_path[1]).decimals(),
  quote.amounts[1]);
        } else {
            _amountOut = weiToTokenDecimals(IERC20Metadata(_path[1]).decimals(),
  _amountOut);
            quote = _lbQuoter.findBestPathFromAmountOut(_path, uint128(_amountOut));
            _ammRate = tokenDecimalsToWei(IERC20Metadata(_path[1]).decimals(),
  quote.amounts[1]) * Constants.WAD
                / tokenDecimalsToWei(IERC20Metadata(_path[0]).decimals(),
  quote.amounts[0]);
        }
        ...
    }
```

Solution

When getting the best price via findBestPathFromAmountIn, the balance of all tokens owned by the current contract should be queried as amountIn.

Status

Fixed

[N4] [Suggestion] Missing of event records

Category: Others

Content

There is no corresponding event logged when a sensitive parameter in the contract is modified.

Code Location:

contracts/protocol/products/autopool/FEYAutoPoolProduct.sol#L276-279

```
    function setSlippage(uint256 _newSlippage) external onlyRole(GOVERNANCE) {
        require(_newSlippage < Constants.MAX_SLIPPAGE, Errors.VE_INVALID_SLIPPAGE);
        slippage = _newSlippage;
    }
```

contracts/protocol/tokenization/StructSPToken.sol#L177-187

```
    function setFeyProductFactory(IFEYFactory _feyProductFactory) external
onlyRole(GOVERNANCE) {
        feyProductFactory = _feyProductFactory;
    }

    ...

    function _setURI(string memory newuri) internal virtual {
        _uri = newuri;
    }
```

Solution

It is recommended to record events when sensitive parameters are modified for self-inspection or community review.

Status

Fixed; The project team response: we decided to skip the event of the _setURI as we will not be calling the function.

[N5] [Medium] Compatibility issue between `_increaseAllowanceAndSwap` operation and deflationary tokens

Category: Design Logic Audit

Content

In the AutoPoolYieldSource contract, when performing the `_recompoundRewards` operation, the `_increaseAllowanceAndSwap` function will be used to swap the joeToken and rewardToken2 tokens into srToken and jrToken for compound interest. It should be noted that rewardToken2 may be a deflationary token, and in the `_increaseAllowanceAndSwap` function, only the swapExactTokensForTokens interface of the lbRouter contract is used for token exchange, but this interface is not suitable for the exchange of deflationary tokens. Therefore, if APT_FARM exists and rewardToken2 is a deflationary token, the protocol will not be able to perform compound interest operations.

Code location:

contracts/protocol/yield-sources/AutoPoolYieldSource.sol#L563

contracts/protocol/yield-sources/AutoPoolYieldSource.sol#L599

```
    function _recompoundRewards(uint256 _reward1Harvested, uint256 _reward2Harvested,
uint256 _wavaxBalanceBefore)
        internal
    {
        ...
        if (numRewards > 1 && _reward2Harvested > 0) {
            if (address(_rewardToken2) != address(tokenA) && address(_rewardToken2) !=
address(tokenB)) {
                if (isReward2Native) {
                    IWETH9(WAVAX).deposit{value: _reward2Harvested}();
                } else {
                    if (address(rewardToken2) != address(WAVAX)) {
                        _increaseAllowanceAndSwap(_reward2Harvested, _rewardToken2,
reward2ToNativeSwapPath);
                    }
                }
                _hasNativeReward = true;
            }
        }
        ...
    }
    function _increaseAllowanceAndSwap(uint256 _amount, IERC20Metadata _token,
address[] memory _path) internal {
        ...
        lbRouter.swapExactTokensForTokens(_amount, _minOut, _route, address(this),
block.timestamp + 1);
    }
```

Solution

It is recommended to use the `try-catch` method in the `_increaseAllowanceAndSwap` function for token exchange. First call the swapExactTokensForTokens interface to try swap, and when it fails, try to use the swapExactTokensForTokensSupportingFeeOnTransferTokens interface for swap.

Status

Acknowledged; The project team response: We made an attempt at your suggestion, but ultimately decided to not implement it because we cannot effectively test swapExactTokensForTokensSupportingFeeOnTransferTokens without a token that takes a fee on transfer, and none of the existing autopool tokens nor rewards have a fee on transfer. If TraderJoe ever lists an autopool with a token that takes a fee on transfer, we can implement this logic specifically for that yield source contract. In the case where one of the reward tokens for an existing autopool becomes a fee on transfer token, we will rescue all tokens from the yield source and deploy it with the fix.

[N6] [Medium] Compatibility issue between redeemTokens function and deflationary tokens

Category: Design Logic Audit

Content

In the AutoPoolYieldSource contract, the keeper role redeems the invested funds by calling the redeemTokens function. This function calls the redeemQueuedWithdrawal() method on the AutoPoolVault contract and then the vault contract sends the tokens to the AutoPoolYieldSource contract. It should be noted that the received tokenA and tokenB tokens may be deflationary tokens. In the case of deflationary tokens, the actual number of tokens received from vault will be less than the tokenAReceived or tokenBReceived returned, which will then lead to incorrect share calculations afterwards.

Code Location: contracts/protocol/yield-sources/AutoPoolYieldSource.sol#L333

```
    function redeemTokens() external gacPausable nonReentrant onlyRole(KEEPER) {
            ...
            (uint256 tokenAReceived, uint256 tokenBReceived) =
                autoPoolVault.redeemQueuedWithdrawal(_roundId, address(this));

            for (uint256 _productIndex; _productIndex < _productsLength;) {
                uint256 _tokenARedeemed =
                    _roundInfo.shares[_productIndex].mulDiv(tokenAReceived,
_roundInfo.totalShares);
                uint256 _tokenBRedeemed =
                    _roundInfo.shares[_productIndex].mulDiv(tokenBReceived,
_roundInfo.totalShares);
               ...
            }
            ...
        }
        ...
    }
```

Solution

It is recommended that the redeemTokens function use the difference between the balances of tokenA and tokenB before and after receiving as tokenAReceived and tokenBReceived to calculate share.

Status

Fixed

[N7] [Suggestion] Gas optimization

Category: Gas Optimization Audit

Content

In the FEYAutoPoolProduct contract, the initialize function uses the necessary parameters to initialize the product. It saves the necessary information of srToken and jrToken through `_srDecimals`, `_jrDecimals`, `trancheTokenSr`, `trancheTokenJr` global variables. But this information has already been recorded in trancheConfig, and there is no need to spend additional gas to store the already stored information.

In the Helpers contract, the getInvestedAndExcess function is used to obtain the user's investment amount and its excess. When `prefixSum - depositAmount < invested`, it will recalculate userInvested and excess, but it does not return directly after the calculation is completed, but recalculates the excess parameter again, which is unnecessary.

Code location:

contracts/protocol/products/autopool/FEYAutoPoolProduct.sol#L75-L76

contracts/protocol/products/autopool/FEYAutoPoolProduct.sol#L86-L90

```
    function initialize(
        ...
    ) external override {
        ...

        trancheConfig[DataTypes.Tranche.Senior] = _initConfig.configTrancheSr;
        trancheConfig[DataTypes.Tranche.Junior] = _initConfig.configTrancheJr;
        ...
        _srDecimals = _initConfig.configTrancheSr.decimals;
        _jrDecimals = _initConfig.configTrancheJr.decimals;

        trancheTokenSr = _initConfig.configTrancheSr.tokenAddress;
        trancheTokenJr = _initConfig.configTrancheJr.tokenAddress;
        ...
    }
```

contracts/protocol/libraries/helpers/Helpers.sol#L87-L90

```
    function getInvestedAndExcess(DataTypes.Investor storage investor, uint256
invested)
        external
        view
        returns (uint256 userInvested, uint256 excess)
    {
        ...
            if (prefixSum - depositAmount < invested) {
                userInvested += (depositAmount + invested - prefixSum);
                excess = investor.userSums[length - 1] - userInvested;
            }
        }
        excess = investor.userSums[length - 1] - userInvested;
    }
```

Solution

If the design is not intended, we recommend only saving necessary parameters through trancheConfig. And when the modifications to userInvested and excess are completed, return the getInvestedAndExcess function.

Status

Acknowledged; The team project response: we agreed that it is redundant, but we are reading the token decimals and token addresses multiple times hence we initially had to load the entire trancheConfig struct into memory every time just to read the token decimals which incurred more gas, iirc. so we decided to store the values seperately.

[N8] [Medium] Risk of excessive privilege

Category: Authority Control Vulnerability Audit

Content

In the FEYProduct contract, the GOVERNANCE role can transfer any ERC20 tokens away from a product contract by calling the rescueTokens function. If the privilege is lost or misused, this may have an impact on the user's assets.

In the FEYFactoryConfigurator contract, the GOVERNANCE role can set the performanceFee by calling the setPerformanceFee function. If the privilege is lost or misused, this may have an impact on the user's assets.

And the GOVERNANCE role can withdraw investment funds from autoPoolVault to the specified address through the emergencyWithdrawAndRescue function in the AutoPoolYieldSource contract. Performing this operation when there is no emergency in the contract will bring huge risks to the user's funds.

Code Location: contracts/protocol/products/FEYProduct.sol#L178-180

```
    function rescueTokens(IERC20Metadata _token, address _recipient) external
onlyRole(GOVERNANCE) {
        _token.safeTransfer(_recipient, Helpers._getTokenBalance(_token,
address(this)));
    }
```

contracts/protocol/products/FEYFactoryConfigurator.sol#L127-130

```
    function setPerformanceFee(uint256 _performanceFee) external onlyRole(GOVERNANCE)
{
        performanceFee = _performanceFee;
        emit PerformanceFeeUpdated(_performanceFee);
    }
```

contracts/protocol/yield-sources/AutoPoolYieldSource.sol#L441-L448

```
    function emergencyWithdrawAndRescue(address _recipient) external
onlyRole(GOVERNANCE) {
        emergencyWithdrawFromFarm();
        emergencyWithdrawFromAutoPool();
        uint256 _amountA = tokenA.balanceOf(address(this));
        uint256 _amountB = tokenB.balanceOf(address(this));
        _rescueTokens(tokenA, _amountA, _recipient, false);
        _rescueTokens(tokenB, _amountB, _recipient, false);
    }
```

Solution

The role of GOVERNANCE is that multi-signature contracts can well mitigate single-point risks, but it still cannot completely solve the excessive special effects risks involving user funds. Therefore, for the rescueTokens and emergencyWithdrawAndRescue functions, it is a better solution to transfer the recovered funds to a specific contract. The contract is supervised by the community, and the operations involving funds in the contract are controlled through the time lock contract, and the contract address Should be hardcoded into the protocol. This can well solve the community trust problem.

Status

Acknowledged; After communicating with the project team, the project team stated that the team plans to use governance management privileged roles in the future to completely address this risk.

[N9] [Suggestion] Redundant function

Category: Gas Optimization Audit

Content

In the Helpers contract, the getTrancheTokenRate function is not used by other contracts in the protocol, and there is currently a getTrancheTokenRateV2 function that replaces getTrancheTokenRate. Therefore getTrancheTokenRate is a redundant function.

Code location: contracts/protocol/libraries/helpers/Helpers.sol#L230-L255

```
    function getTrancheTokenRate(
        IStructPriceOracle _structPriceOracle,
        address _asset1,
        address _asset2,
        address[] storage _path,
        IJoeRouter _router
    ) external view returns (bool, uint256, uint256, uint256) {
        ...
    }
```

Solution

If the design is not expected, it is recommended to remove redundant functions.

Status

Fixed

[N10] [Low] There is a flaw in the amountInMax calculation when tranche tokens are allocated

Category: Others

Content

In the FEYAutoPoolProduct contract, the `_allocateToTranches` function is used to allocate tranche tokens. When `_jrToSwap < _receivedJr`, it will be exchanged for tokens through the `_swapToExact` function. In the `_swapToExact` function, it will use slippage to calculate `_amountInMax`, but when the difference between `_jrToSwap` and `_receivedJr` is very small, the value of `_amountInMax` may exceed `_receivedJr`. Therefore, the amountIn required for the swapTokensForExactTokens operation may exceed `_receivedJr`, which will cause the jrToken token balance in the contract to be unable to meet the swap requirements, ultimately causing the `_allocateToTranches` operation to fail.

The current MEV situation is extremely common, so this error is very likely to occur when the difference between `_jrToSwap` and `_receivedJr` is small.

Code location: contracts/protocol/products/autopool/FEYAutoPoolProduct.sol#L441

```
    function _allocateToTranches(uint256 _receivedSr, uint256 _receivedJr, uint256
 _srFrFactor) private {
        ...
        } else if (_receivedSr < _srFrFactor) {
            ...
            } else {
                _swapToExact(_seniorDelta, juniorTokenToSeniorTokenSwapPath,
 address(this));
            }
        }
    }
```

Solution

It is recommended that `amountInMax` be no larger than `_receivedJr`.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002310170001	SlowMist Security Team	2023.10.08 - 2023.10.17	Medium Risk

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 critical risk, 1 high risk, 4 medium risk, 1 low risk, and 3 suggestions. All the findings were fixed or acknowledged. The code was not deployed to the mainnet. Due to excessive privilege issues, the audit conclusion of the current protocol is medium risk. The project team plans to deploy governance for managing privileged roles in the near future to address this risk.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.