CHAIN TROOPERS

Scion Finance

Contracts

Security Assessment Report

September 9th, 2022

Version 1.1

CONFIDENTIAL

Table of Contents

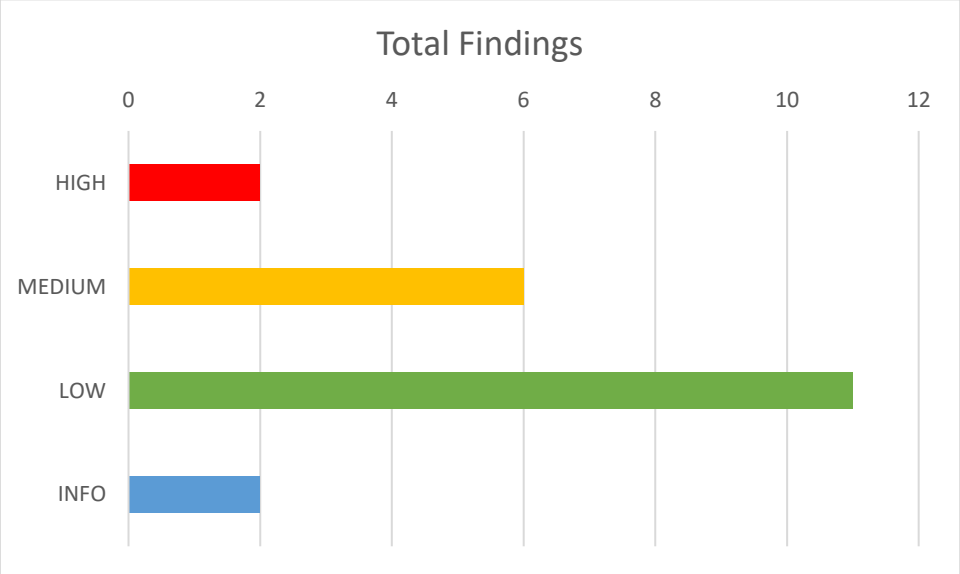
1 Executive Summary

1.1 Introduction

The report contains the results of Scion Finance Contracts security assessment that took place from July 18th, 2022, to August 1st, 2022. The security engineers performed an in-depth manual analysis of the provided functionalities, and uncovered issues that may be used by adversaries to affect the confidentiality, the integrity, and the availability of the in-scope components.

All the identified vulnerabilities are presented in the report, including their impact and the proposed mitigation strategy, and are ordered by their severity.

In total, the team identified nineteen (19) vulnerabilities. There were also two (2) informational issues of no-risk.



All the identified vulnerabilities are presented in the report, including their impact and the proposed mitigation strategy, and are ordered by their severity.

A retesting phase was carried out on September 7th, 2022, and the results are presented in Section 6.

1.2 Assessment Results

The assessment results revealed that the in-scope application components were mainly vulnerable to two (2) Data Validation issues of HIGH risk. More precisely, it was identified that there is no upper bound for the max slippage *('5.1.1 - No upper bound in max slippage potentially allows Sandswitch attack at "HedgedLP.sol" and "IMX.sol"')*, allowing the managers to perform operations at any possible slippage value. Such transactions can be abused by adversaries and exploited in a *sandwich* attack, especially in markets with low liquidity. Furthermore, the team found that a number of functionalities can affect the liquidity of the strategy on-demand, and as a result can be exploited to perform a flash loan attack *('5.1.2 - Flash Loan attack using "closePosition()", "removeLiquidity()", "rebalance()" and "rebalanceLoan()' functionalities at "HedgedLP.sol" and "IMX.sol"')*. A compromised manager wallet can be used to decrease the position even if the price offset is not in favor of the strategy, reduce the liquidity of underlying tokens to zero on demand, and profit from the price difference. It should be noted that both attacks require a compromised Manager wallet.

The in-scope components were also affected by six (6) Access Control, Data Validation, Administration and Business Logic vulnerabilities of MEDIUM risk. Regarding the Access Control issues, it was found that the permissions of the HedgeLP strategy do not comply with the documented permissions model and provides "managers" with increased privileges, allowing them to withdraw tokens from farms on demand, decrease the position, or even control the acceptable price offset up to a range *('5.2.1 - Excessive and Inconsistent permissions in emergency functionalities "withdrawFromFarm()", "redeemCollateral()", "removeLiquidity()" at "*HedgedLP.sol"*', '5.2.3 - Excessive and Inconsistent permissions in "setMaxPriceMismatch" functionality potentially allows Sandwitch attack at "HedgedLP.sol" and "*IMX.sol"*')*.

In reference to the Data Validation MEDIUM-risk issues, it was identified that certain functions of the Base strategy are not validating the provided addresses to not be the zero address *('5.2.2 - Unvalidated addresses in governance functionalies "setVault()" and "emergencyWithdraw()" at "BaseStrategy.sol"')*. An accidental transfer of several tokens to the zero address is equivalent to burning that number of tokens. Regarding the Administration issue of MEDIUM-risk, it was found that the initialization scripts deploy the Time Lock contract with a very small minimum delay requirement *('5.2.4 - Insufficient and Inconsistent time lock delay set*

at "02_*timelock.ts"')*, making it impossible for the users to react to a proposed change.

Furthermore, in reference to the Business Logic MEDIUM-risk issues, it was found that the order of operations in certain functions can introduce a loss of precision due to round-off errors *('5.2.5 - Order of operations can introduce round-off errors in "HedgedLP.sol"', '5.2.6 - Order of operations can introduce round-off errors in "IMXFarm.sol"')*. These errors occur because the division of a number takes place before a multiplication.

The examined contracts were also affected by eleven (11) Administration and Data Validation LOW-risk issues. Regarding the LOW-risk Administration issues, the team identified that many admin functionalities do not emit the appropriate event when an action is taken *('5.3.1 - Event not emitted in "closePosition" functionality at "HedgedLP.sol"', '5.3.2 - Event not emitted in "withdrawFromFarm" functionality at "HedgedLP.sol"', '5.3.3 - Event not emitted in "removeLiquidity" functionality at "HedgedLP.sol"', '5.3.4 - Event not emitted in "redeemCollateral" functionality at "HedgedLP.sol"', '5.3.5 - Event not emitted in "mint" functionality at "BaseStrategy.sol"', '5.3.6 - Event not emitted in "redeemUnderlying" functionality at "BaseStrategy.sol"')*, potentially affecting the credibility and the confidence in the system.

In reference to the LOW risk Data Validation issues, it was found that the lower and upper bounds of several configuration parameters are not being validated and can lead to unexpected behavior if they are accidentally initialized with empty or zeroed parameters *('5.3.7 - Lack of validation for minimum delay at "ScionTimelock.sol"', '5.3.10 - Unvalidated parameters in "setMaxTvl()", "setSafeCollateralRatio()", "setMinLoanHeath()", "setMaxPriceMismatch()" and "setRebalanceThreshold()" at "HedgedLP.sol" and "IMX.sol"')*. Moreover, the team identified that many smart contracts are using a floating pragma or support deprecated and outdated Solidity versions *('5.3.8 - Floating pragma in multiple contracts', '5.3.9 - Outdated compiler version supported at "USDCftmSPIRITscream.sol"')*. In Solidity programming, multiple APIs are only supported in some specific versions. Finally, certain functionalities do not return any value, even if a return type is specified in their definition *('5.3.11 - No return value in functions at "IMXFarm.sol"')*, making it impossible for the caller function to verify the result.

There were also two (2) findings of no-risk (INFORMATIONAL). Chaintroopers recommend the immediate mitigation of all HIGH and MEDIUM-risk issues. It is also advisable to address all LOW and INFORMATIONAL findings to enhance the overall security posture of the components.

1.2.1 Retesting Results

Results from retesting carried out in September 2022, determined that all HIGH-risk vulnerabilities (2 out of 21 findings) have been successfully mitigated (see sections 5.1.1 and 5.1.2).

Furthermore, two (2 out of 6) vulnerabilities of MEDIUM risk, nine (8 out of 11) vulnerabilities of LOW risk and one (1 out 2) INFORMATIONAL issue have been sufficiently addressed (see sections 5.2.1, 5.2.3, 5.3.1, 5.3.2, 5.3.3, 5.3.4, 5.3.5, 5.3.6, 5.3.9, 5.3.10 and 5.4.2).

One (1) LOW-risk issue has been partially fixed, and its risk is downgraded to INFORMATIONAL finding (see section 5.3.8), while the risk of four (4 out of 6) MEDIUM-risk vulnerabilities and two (2) LOW-risk vulnerabilities has been accepted as minor issues, intentional use cases, mitigated at the UI layer or not in current scope (see sections 5.2.2, 5.2.4, 5.2.5, 5.2.6, 5.3.7 and 5.3.11).

One (1 out of 2) finding of INFORMATIONAL risk remains OPEN (see section 5.4.1).

1.3 Summary of Findings

The following findings were identified in the examined source code:

Vulnerability Name	Status	Retest Status	Page
No upper bound in max slippage potentially allows Sandswitch attack at "HedgedLP.sol" and "IMX.sol"	HIGH	CLOSED	15
Flash Loan attack using "withdrawFromFarm()", "closePosition()", "removeLiquidity()", "rebalance()" and "rebalanceLoan()' functionalities at "HedgedLP.sol" and "IMX.sol"	HIGH	CLOSED	17
Excessive and Inconsistent permissions in emergency functionalities "withdrawFromFarm()", "redeemCollateral()", "removeLiquidity()" at "HedgedLP.sol"	MEDIUM	CLOSED	20
Unvalidated addresses in governance functionalies "setVault()", and "emergencyWithdraw()" at "BaseStrategy.sol"	MEDIUM	ACCEPTED RISK	23
Excessive and Inconsistent permissions in "setMaxPriceMismatch" functionality potentially allows Sandwitch attack at "HedgedLP.sol" and "IMX.sol"	MEDIUM	CLOSED	25
Insufficient and Inconsistent time lock delay set at "02_timelock.ts"	MEDIUM	ACCEPTED RISK	28
Order of operations can introduce round-off errors in "HedgedLP.sol"	MEDIUM	ACCEPTED RISK	30
Order of operations can introduce round-off errors in "IMXFarm.sol"	MEDIUM	ACCEPTED RISK	32
Event not emitted in "closePosition" functionality at "HedgedLP.sol"	LOW	CLOSED	34

Event not emitted in "withdrawFromFarm" functionality at "HedgedLP.sol"	LOW	CLOSED	36
Event not emitted in "removeLiquidity" functionality at "HedgedLP.sol"	LOW	CLOSED	38
Event not emitted in "redeemCollateral" functionality at "HedgedLP.sol"	LOW	CLOSED	40
Event not emitted in "mint" functionality at "BaseStrategy.sol"	LOW	CLOSED	42
Event not emitted in "redeemUnderlying" functionality at "BaseStrategy.sol"	LOW	CLOSED	44
Lack of validation for minimum delay at "ScionTimelock.sol"	LOW	ACCEPTED RISK	46
Floating pragma in multiple contracts	LOW	INFO	49
Outdated compiler version supported at "USDCftmSPIRITscream.sol"	LOW	CLOSED	52
Unvalidated parameters in "setMaxTvl()", "setSafeCollateralRatio()", "setMinLoanHeath()", "setMaxPriceMismatch()" and "setRebalanceThreshold()" at "HedgedLP.sol" and "IMX.sol"	LOW	CLOSED	54
No return value in functions at "IMXFarm.sol"	LOW	ACCEPTED RISK	56
Unvalidated parameters in "mint()" and "redeemUnderlying()" at "BaseStrategy.sol"	INFO	INFO	58
Debugging utility in "HedgedLP.sol"	INFO	CLOSED	60

2 Assessment Description

2.1 Target Description

Scion Finance is an on-chain, permissionless hedge fund/farm that offers low-risk, high-yield delta-neutral investment strategies that generate consistent returns independent of market direction.

It offers users market-making strategies that provide exposure to the returns of farming volatile assets while hedging the market risks of those assets. For example, a user can earn yield from providing liquidity to the USDC/AVAX with a USDC-only portfolio, without holding any AVAX. These kinds of strategies are called delta-neutral because they are not dependent on any market direction to make a profit (unlike, going long or short ETH for example).

The hedged Uniswap market-making strategy enables users to seamlessly borrow a volatile asset from a lending protocol and provide it as LP (along with a stable asset) to a Uniswap-like DEX.

2.2 In-Scope Components

The following list specifies the repository that was subject to this audit. Tests are excluded.

- *https://github.com/scion-finance/contracts*

The scope included the Base strategy, the HedgedLP strategy, the changes in the Vault mechanism, and the operational functionalities of the IMX Strategy.

Component	Commit Identifier
contracts	*d2046c805a6cb1da673258948a26062d43e70cf9*
contracts (dev) - Audit Updates #2 - Retest	*ab3a7a862f655e240b1bf8d3b66ca1bb4e286405*

3 Methodology

3.1 Assessment Methodology

Chaintroopers' methodology attempts to bridge the penetration testing and source code reviewing approaches in order to maximize the effectiveness of a security assessment.

Traditional pentesting or source code review can be done individually and can yield great results, but their effectiveness cannot be compared when both techniques are used in conjunction.

In our approach, the application is stress tested in all viable scenarios though utilizing penetration testing techniques with the intention to uncover as many vulnerabilities as possible. This is further enhanced by reviewing the source code in parallel to optimize this process.

When feasible our testing methodology embraces the Test-Driven Development process where our team develops security tests for faster identification and reproducibility of security vulnerabilities. In addition, this allows for easier understanding and mitigation by development teams.

Chaintroopers' security assessments are aligned with OWASP TOP10 and NIST guidance.

This approach, by bridging penetration testing and code review while bringing the security assessment in a format closer to engineering teams has proven to be highly effective not only in the identification of security vulnerabilities but also in their mitigation and this is what makes Chaintroopers' methodology so unique.

3.2 Smart Contracts

The testing methodology used is based on the empirical study "Defining Smart Contract Defects on Ethereum" by J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo, and T. Chen, in IEEE Transactions on Software Engineering, and the security best practices as described in "Security Considerations" section of the solidity wiki.

The following is a non-exhaustive list of security vulnerabilities that are identified by our methodology during the examination of the in-scope contract:

- Unchecked External Calls
- Strict Balance Equality
- Transaction State Dependency
- Hard Code Address
- Nested Call
- Unspecified Compiler Version
- Unused Statement
- Missing Return Statement
- Missing Reminder
- High Gas Consumption Function Type
- DoS Under External Influence
- Unmatched Type Assignment
- Re-entrancy
- Block Info Dependency
- Deprecated APIs
- Misleading Data Location
- Unmatched ERC-20 standard
- Missing Interrupter
- Greedy Contract
- High Gas Consumption Data Type

In Substrate Pallets, the list of vulnerabilities that are identified also includes:

- Static or Erroneously Calculated Weights
- Arithmetic Overflows
- Unvalidated Inputs
- Runtime Panic Conditions
- Missing Storage Deposit Charges
- Non-Transactional Dispatch Functions
- Unhandled Errors &Unclear Return Types
- Missing Origin Authorization Checks

4 Scoring System

4.1 CVSS

All issues identified as a result of Chaintroopers' security assessments are evaluated based on Common Vulnerability Scoring System version 3.1 (https://www.first.org/cvss/).

With the use of CVSS, taking into account a variety of factors a final score is produced ranging from 0 up to 10. The higher the number goes the more critical an issue is.

The following table helps provide a qualitative severity rating:

Rating	CVSS Score
None/Informational	0.0
Low	0.1-3.9
Medium	4.0-6.9
High	7.0-8.9
Critical	9.0-10.0

Issues reported in this document contain a CVSS Score section, this code is provided as an aid to help verify the logic of the team behind the evaluation of a said issue. A CVSS calculator can be found in the following URL:

https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator

5 Identified Findings

5.1 High Severity Findings

5.1.1 No upper bound in max slippage potentially allows Sandswitch attack at "HedgedLP.sol" and "IMX.sol"

Description	HIGH

It was found that no upper bound for the max slippage is set, allowing the managers to perform "*rebalance*()" and "*closePosition*()" operations at any possible slippage value. Slippage refers to the discrepancy between the expected price of a trade and the price at which the trade is executed. The max slippage defines the maximum amount that an order can move against the trader before it is cancelled. In the specific case, it was identified that the managers have no limitation on performing these operations.

The issue exists at the following location:

```
File: /contracts/src/strategies/HedgedLP.sol
44:         modifier checkPrice(uint256 maxSlippage) {
45:                 if (maxSlippage == 0) maxSlippage = maxPriceMismatch;
46:                     require(getPriceOffset() <= maxSlippage, "HLP:
PRICE_MISMATCH");
47:                 _;
48:         }
```

For example, a manager can perform an operation with a Slippage of even 100% by providing the 10000 number. This type of issues is usually exploited during a *sandwich* attack. Sandwiching occurs when transactions are inserted around another user's transaction, to profit from the victim's slippage tolerance

Impact

A manager can perform one of the allowed operations, such as "*rebalance*()", even with 100% max slippage. This order can be abused especially in markets with low liquidity which have increased slippage. This type of issues is usually

exploited during a *sandwich* attack. In this scenario, the attack can be performed by other wallets that they control to withdraw funds.

While a "*rebalance*()" or "*closePosition*()" operation with high maximum slippage can be a legitimate transaction under certain extreme financial cases, it can also be abused by malicious managers whose private keys have been compromised in order to exhaust the available funds.

Recommendation

It is recommended to set an upper bound on the maximum slippage that is allowed to be used, that will be controlled by a time locked or multisig account. This account will be able to authorize the usage of high slippage under specific cases.

CVSS Score

AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:H/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.1.2 *Flash Loan attack using "closePosition()", "removeLiquidity()", "rebalance()" and "rebalanceLoan()' functionalities at "HedgedLP.sol" and "IMX.sol"*

Description	HIGH

The team identified that the "*closePosition*()", "*removeLiquidity*()", "*rebalance*()" and "*rebalanceLoan*()' functionalities can affect the liquidity of the strategy, and as a result can be exploited to perform a flash loan attack. The above functionalities can be used by compromised manager wallets to decrease the position even if the price offset is not in favor of the strategy and reduce the LP underlying tokens at zero on demand, to profit from the price difference.

Flash loan attacks are essentially a very rapid crypto pump-and-dump that leverages the quick and collateral-free borrowing available via some DeFi platforms. These loans are “secured” by setting a tight time limit in which repayment must be made; if the borrower does not make it by the end of the window, the entire transaction is invalidated automatically. In a basic flash loan attack, the borrower immediately uses the large amount of funds to buy a large amount of a crypto asset, triggering a sell-off. This artificially drops the price on that exchange, due to the on-chain price calculation of the system, at least until the loan repayment window closes. During this time, the attackers snap up the now undervalued crypto asset and sell it at another exchange that is maintaining normal market prices.

In the specific case, an adversary, who is also the manager in a deployed HedgedLP strategy, can borrow a collateral-free loan in a DeFi platform (e.g., USDC or DAI from AAVE).

In general, users will place funds of an underlying token (e.g., USDC) in the HedgedLP strategy which will try to borrow a volatile asset from a lending protocol and provide it as LP (along with a stable asset) to a Uniswap-like DEX. So, the strategy will use a part of this underlying token as a collateral to borrow a short liquidity token (e.g., 68% of the USDC deposit funds will be used as collateral to borrow MOVR from Moonwell) and placed in a farm. The rest of the

funds of the deposit (32%) will be also used in the farm as part of the delta-neutral strategy.

Then, the adversary will use the borrowed funds to affect the underlying token to short token ratio. At the same time, the adversary can use the "*closePosition*()", and "*removeLiquidity*()" functions to forcefully dump the tokens, swap them with the necessary ones, and repay the strategy's loan, even if the price offset is not in favor of the strategy. During this operation, the adversary will use the borrowed funds to profit from the price difference and repay the loan.

The "*rebalance*()" and the "*rebalanceLoan*()" can also be used to perform the attack at a lesser extent. The "*rebalance*()" functionality is used to decrease LP in order to increase collateral and borrow more funds or to remove all LP and repay loan. In this case, the decision is taken based on the underlying token to short token (u/s) ratio. Similarly, the "*rebalanceLoan*()" functionality can be used by any user of the platform to remove LP and repay loan if the price difference is above a certain level. Currently, the examined system utilizes the following mechanism to identify the price offsets that these operations are allowed to be performed:

```
File: contracts/src/strategies/HedgedLP.sol
520:  function getPriceOffset() public view returns (uint256 offset) {
521:        uint256 minPrice = _shortToUnderlying(1e18);
522:        uint256 maxPrice = _oraclePriceOfShort(1e18);
523:        (minPrice, maxPrice) = maxPrice > minPrice ? (minPrice,
maxPrice) : (maxPrice, minPrice);
524:        offset = ((maxPrice - minPrice) * BPS_ADJUST) / maxPrice;
525:  }
```

And the "*_shortToUnderlying*" and "*_oraclePriceOfShort*" functions:

```
File: /contracts-test/src/strategies/mixins/ICompound.sol
090:  function _oraclePriceOfShort(uint256 amount) internal view override
returns (uint256) {
091:        return
092:              (amount                                                *
oracle().getUnderlyingPrice(address(cTokenBorrow()))) /
```

```
093:              oracle().getUnderlyingPrice(address(cTokenLend()));
094:  }
095:
096:  function  _oraclePriceOfUnderlying(uint256  amount)  internal  view
override returns (uint256) {
097:        return
098:              (amount                                                *
oracle().getUnderlyingPrice(address(cTokenLend())) /
099:              oracle().getUnderlyingPrice(address(cTokenBorrow()));
100:  }
```

Impact

An adversary can abuse the forced withdrawal functionality to remove liquidity while the price offset is not in favor of the strategy. The issue can be mainly exploited by malicious managers or by compromised manager wallets that are able to use the "*closePosition*()" and "*removeLiquidity*()" functionalities and at certain extend using the "rebalance()" functionality. Users are also able to perform a similar attack under certain conditions using the "*rebalanceLoan*()" functionality.

Recommendation

It is recommended to apply specific, limited, upper bounds in price offsets at which functions such as *"closePosition()", "removeLiquidity()", "rebalance*()" and "*rebalanceLoan*()' can be used.

Furthermore, it is advisable to investigate if the functions "closePosition()" and "removeLiquidity()" should be accessible by the managers.

CVSS Score

AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:H/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2 Medium Severity Findings

5.2.1 Excessive and Inconsistent permissions in emergency functionalities "withdrawFromFarm()", "redeemCollateral()", "removeLiquidity()" at "HedgedLP.sol"

Description	MEDIUM

It was identified that the permissions of the HedgeLP strategy do not comply with the permissions model of the rest application components regarding the "managers" privileges. The hedged market-making strategy enables users to seamlessly borrow a volatile asset from a lending protocol and provide it as LP (along with a stable asset) to a Uniswap-like DEX.

According to the documentation, every strategy has three (3) roles, the owner, the manager, and the vault. The owner can perform sensitive configurations ("*setMinLoanHealth", "setMaxPriceMismatch", "setRebalanceThreshold", "setManager",* and *"setVault*"), the Vault is only allowed to "*mint*", "*redeemUnderlying*" and perform "*emergencyWithdraw*", while the managers and the owner are only allowed to "*rebalance*", to "*harvest*", to "*setMaxTvl*" and to "*closePosition*".

However, in the specific case it was found that the managers are also allowed to perform several other functionalities, such as to remove liquidity, to withdraw stuck collateral and to withdraw LP tokens from farm.

The "OnlyAuth()" modifier is defined at the following location:

```
File: /contracts/src/strategies/BaseStrategy.sol
21:         modifier onlyAuth() {
22:                 require(msg.sender == owner() || _managers[msg.sender]
== true, "Strat: NO_AUTH");
23:                 _;
24:         }
```

and the permissions are defined at:

```
File: /contracts/src/strategies/HedgedLP.sol
296:         // in case of emergency - withdraw lp tokens from farm
297:         function withdrawFromFarm() public onlyAuth {
298:                 _withdrawFromFarm(_getFarmLp());
299:         }
300:
301:         // in case of emergency - withdraw stuck collateral
302:              function redeemCollateral(uint256 repayAmnt, uint256
withdrawAmnt) public onlyAuth {
303:                 _repay(repayAmnt);
304:                 _redeem(withdrawAmnt);
305:         }
306:
307:         // in case of emergency - remove LP
308:          function removeLiquidity(uint256 removeLp) public onlyAuth
{
309:                 _removeLiquidity(removeLp);
310:         }
```

Impact

While the emergency functionalities "withdrawFromFarm()", "redeemCollateral()", and "removeLiquidity()" can be legitimate transactions under certain extreme financial cases, they can also be abused by malicious managers whose private keys have been compromised in order to tamper the available funds, as described in *'5.1.2 - Flash Loan attack using "closePosition()", "removeLiquidity()", "rebalance()" and "rebalanceLoan()' functionalities at "HedgedLP.sol" and "IMX.sol"'.*

Furthermore, since the used privileges are not in compliance with the published documentation, this issue affects the credibility and the confidence in the system.

Recommendation

It is advisable to allow the specific operations only to the Owner account. A multisig approach can be used to protect emergency operations that cannot be protected using a time lock wallet.

If this is not possible, it is recommended to apply specific, limited, price offsets in which these functionalities can be used by the managers, and to update the documentation to reflect the implemented permissions.

CVSS Score

AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:L/A:L/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.2 Unvalidated addresses in governance functionalies "setVault()" and "emergencyWithdraw()" at "BaseStrategy.sol"

Description	MEDIUM

The team identified that the *"setVault()" and "emergencyWithdraw"* functions of BaseStrategy are not validating the provided addresses to not be the zero address. Transferring a number of tokens to the zero address is equivalent to burning that amount of tokens.

The issue is located at:

```
File: /contracts/src/strategies/BaseStrategy.sol
110:         function setVault(address vault_) external onlyOwner {
111:                 _vault = vault_;
112:                 emit VaultUpdate(vault_);
113:         }
114:
118:             function emergencyWithdraw(address recipient, IERC20[]
calldata tokens)
119:                 external
120:                 override
121:                 onlyVault
122:         {
123:                 for (uint256 i = 0; i < tokens.length; i++) {
124:                         IERC20 token = tokens[i];
125:                   uint256 balance = token.balanceOf(address(this));
126:                     if (balance != 0) token.safeTransfer(recipient,
balance);
127:                 }
128:                   // send ETH to vault (no reason it should go to
recipient / owner)
129:                                 if (address(this).balance > 0)
SafeETH.safeTransferETH(vault(), address(this).balance);
130:                 emit EmergencyWithdraw(recipient, tokens);
131:         }
```

Impact

The vault can use the "*emergencyWithdraw*" function to transfer an amount of tokens to the zero address (either accidentally or on purpose), effectively burning that amount of tokens. The owner can also set the zero address as one of the managers or the vault address.

Recommendation

It is advisable to verify that the address is not the zero address.

The functions assert and require can be used to check for conditions and throw an exception if the condition is not met. The control can also be implemented with a simple check:

```
if (vault_ == address(0)) revert InvalidReceiver();
```

CVSS Score

AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:L/A:H/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.3 Excessive and Inconsistent permissions in "setMaxPriceMismatch" functionality potentially allows Sandwitch attack at "HedgedLP.sol" and "IMX.sol"

Description	MEDIUM

It was identified that the "setMaxPriceMismatch" permissions of the HedgeLP and IMX strategy do not comply with the permissions model of the rest application components regarding the "managers" privileges. The hedged market-making strategy enables users to seamlessly borrow a volatile asset from a lending protocol and provide it as LP (along with a stable asset) to a Uniswap-like DEX. The "setMaxPriceMismatch" adjusts the max price if needed. According to the documentation, only the owner is allowed to use this functionality.

The "OnlyAuth()" modifier is defined at the following location:

```
File: /contracts/src/strategies/BaseStrategy.sol
21:         modifier onlyAuth() {
22:               require(msg.sender == owner() || _managers[msg.sender]
== true, "Strat: NO_AUTH");
23:                 _;
24:         }
```

and the permissions allowing managers to adjust the price up to "maxAllowedMismatch" are defined in the following location:

```
File: /contracts/src/strategies/HedgedLP.sol
097:         // manager can adjust max price if needed
098:            function setMaxPriceMismatch(uint256 maxPriceMismatch_)
public onlyAuth {
099:                 require(msg.sender == owner() || maxAllowedMismatch
>= maxPriceMismatch_, "HLP: TOO LARGE");
100:                 maxPriceMismatch = maxPriceMismatch_;
101:                 emit SetMaxPriceMismatch(maxPriceMismatch_);
102:         }
```

```
File: /contracts/src/strategies/IMX.sol
113:         // manager can adjust max price if needed
114:            function setMaxPriceMismatch(uint256 maxPriceMismatch_)
public onlyAuth {
115:                 require(msg.sender == owner() || maxAllowedMismatch
>= maxPriceMismatch_, "HLP: TOO LARGE");
116:                 maxPriceMismatch = maxPriceMismatch_;
117:                 emit SetMaxPriceMismatch(maxPriceMismatch_);
118:         }
```

The max allowed mismatch is configured to 3%:

```
File: /contracts/src/strategies/HedgedLP.sol
34:         uint256 constant maxAllowedMismatch = 300; // manager cannot
make price mismatch more than 3%
35:           uint256 public minLoanHealth = 1.15e18; // how close to
liquidation we get
```

In the specific case, managers can adjust the price up to 3%. This type of issues is usually exploited during a sandwich attack. Sandwiching occurs when transactions are inserted around another user's transaction, to profit from the victim's slippage tolerance.

Impact

A manager can trick other managers who perform "rebalance()" and "closePosition()" operations with 0% maxslippage to use 3% instead by changing the max price mismatch value. This type of issues is usually exploited during a sandwich attack. In this scenario, the attack can be performed by other wallets that they control to withdraw funds.

Furthermore, since the used privileges are not in compliance with the published documentation, this issue affects the credibility and the confidence in the system.

Recommendation

It is advisable to allow the specific operation only to the Owner account. A multisig approach can be used to protect emergency operations that cannot be protected using a time lock wallet.

If this is not possible, it is recommended to evaluate if the 3% is an accepted upper bound offset that be used by the managers, and to update the documentation to reflect the implemented permissions.

CVSS Score

AV:N/AC:L/PR:L/UI:R/S:U/C:N/I:L/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.4 Insufficient and Inconsistent time lock delay set at "02_timelock.ts"

Description	MEDIUM

It was found that the initialization scripts deploy the "ScionTimelock.sol" contract with a very small minimum delay requirement. The time lock enforces a minimum delay between the proposition and the execution on all `*onlyOwner*` maintenance operations. However, in the examined case, the minimum delay is set to one (1) minute:

```
File: /contracts/deploy/02_timelock.ts
20:
21:   const minDelay = 60 * 1; // 1m
22:
23:   const proposers = [deployer, team1, timelockAdmin];
24:   const executors = [deployer, manager, team1];
25:
26:   await deploy('ScionTimelock', {
27:     from: deployer,
28:     log: true,
29:     skipIfAlreadyDeployed: true,
30:     args: [minDelay, proposers, executors],
31:   });
```

It should be noted that the documentation mentions 5 days

https://docs.scion.finance/security/permissions-architecture

Impact

The users are not able to react to a proposed change and either accept it or withdraw their assets if they do not agree.

Recommendation

It is recommended to increase the minimum delay to an acceptable time window (e.g., a minimum of two (2) days).

Furthermore, it is advisable to update the documentation according to the selected value.

CVSS Score

AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:H/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.5 Order of operations can introduce round-off errors in "HedgedLP.sol"

Description	MEDIUM

The team identified that the order of operations in "*_rebalanceLoan*()" function can introduce a loss of precision due to round-off errors. A roundoff error, also called rounding error, is the difference between the result produced by a given algorithm using exact arithmetic and the result produced by the same algorithm using finite-precision, rounded arithmetic. Rounding errors are due to inexactness in the representation of real numbers and the arithmetic operations done with them. These types of errors occur when division of a number takes place before a multiplication.

In the specific case the *"targetHealth"* is calculated by dividing with *"_safeCollateralRatio"*, and then multiplied with collateral and 1e18.

The issue exists at the following location:

```
File: /contracts/src/strategies/HedgedLP.sol
134:         function _rebalanceLoan(uint256 _loanHealth) internal {
135:                 ...
138:                 // get back to our target _safeCollateralRatio
139:                         uint256 targetHealth = (10000 * 1e18) /
_safeCollateralRatio;
140:                     uint256 addCollateral = (1e18 * ((collateral *
targetHealth) / _loanHealth - collateral)) /
141:                      ((targetHealth * 1e18) / _getCollateralFactor()
+ 1e18);
142:
...
151:         }
```

Impact

The current implementation introduces a small loss on precision due to the order of multiplication and division. This can affect the *"targetHealth"* and the *"addCollateral"* values

Recommendation

It is recommended to rearrange the mathematical operations and try to perform all multiplications before any divisions, in order to increase precision.

CVSS Score

AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:N/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.6 Order of operations can introduce round-off errors in "IMXFarm.sol"

Description	MEDIUM

The team identified that the order of operations in "*_optimalUBorrow*()" function in "*IMXFarm*" contract can introduce a loss of precision due to round-off errors. A roundoff error, also called rounding error, is the difference between the result produced by a given algorithm using exact arithmetic and the result produced by the same algorithm using finite-precision, rounded arithmetic. Rounding errors are due to inexactness in the representation of real numbers and the arithmetic operations done with them. These types of errors occur when division of a number takes place before a multiplication.

The issue exists at the following location:

```
File: /contracts/src/strategies/adapters/IMXFarm.sol
226:         // borrow amount of underlying for every 1e18 of deposit
227:       function _optimalUBorrow() internal override returns (uint256
uBorrow) {
228:                                (uint256 price0, uint256 price1) =
collateralToken.getPrices();
229:                 if (_flip) (price0, price1) = (price1, price0);
230:
231:                uint256 l = collateralToken.liquidationIncentive();
232:                 // this is the adjusted safety margin - how far we
stay from liquidation
233:                  uint256 s = (collateralToken.safetyMarginSqrt() *
_getSafetyMarginSqrt()) / 1e18;
234:                 uBorrow = (1e18 * (2e18 - (l * s) / 1e18)) / ((l *
1e18) / s + (l * s) / 1e18 - 2e18);
235:         }
236:
```

Impact

The current implementation introduces a small loss on precision due to the order of multiplication and division. This can affect the "*uBorrow*" value

Recommendation

It is recommended to rearrange the mathematical operations and try to perform all multiplications before any divisions, to increase precision.

CVSS Score

AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3 Low Severity Findings

5.3.1 Event not emitted in "closePosition" functionality at "HedgedLP.sol"

Description	LOW

It was identified that the authorized command "closePosition" does not emit an event. A contract can emit events when it wants to notify external entities like users, chain explorers, or dApps about changes or conditions in the blockchain. When an event is emitted, it stores the arguments passed in transaction logs. These logs are stored on blockchain and are accessible using address of the contract till the contract is present on the blockchain

The issue exists at:

```
File: /contracts/src/strategies/HedgedLP.sol
291:           // note: one should call harvest immediately before close
position
292:                 function closePosition(uint256 maxSlippage) public
checkPrice(maxSlippage) onlyAuth {
293:                 _closePosition();
294:         }
```

Which calls "*_closePosition*()" at:

```
File: /contractst/src/strategies/HedgedLP.sol
311:
312:         function _closePosition() internal returns (uint256) {
313:                 _decreaseLpTo(0);
314:                 uint256 shortPosition = _updateAndGetBorrowBalance();
315:               uint256 shortBalance = _short.balanceOf(address(this));
316:                 if (shortPosition > shortBalance) {
317:                         pair()._swapTokensForExactTokens(
318:                                 shortPosition - shortBalance,
319:                                 address(_underlying),
320:                                 address(_short)
321:                         );
322:                 } else if (shortBalance > shortPosition) {
```

```
323:                         pair()._swapExactTokensForTokens(
324:                                 shortBalance - shortPosition,
325:                                 address(_short),
326:                                 address(_underlying)
327:                         );
328:                 }
329:                 _repay(_short.balanceOf(address(this)));
330:                                         uint256 collateralBalance =
_updateAndGetCollateralBalance();
331:                 _redeem(collateralBalance);
332:                 return _underlying.balanceOf(address(this));
333:         }
334:
```

Impact

Events are necessary to notify the off-chain world of successful state transitions. Administration functionalities should emit the corresponding events throughout the system's life cycle to provide credibility and confidence in the system.

In the specific case, if the authorized command "*closePosition*" is called, no event will be emitted.

Recommendation

It is recommended to emit an event related to this functionality.

CVSS Score

AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.2 Event not emitted in "withdrawFromFarm" functionality at "HedgedLP.sol"

Description	LOW

It was identified that the authorized command "*withdrawFromFarm*" does not emit an event. A contract can emit events when it wants to notify external entities like users, chain explorers, or dApps about changes or conditions in the blockchain. When an event is emitted, it stores the arguments passed in transaction logs. These logs are stored on blockchain and are accessible using address of the contract till the contract is present on the blockchain.

The issue exists at:

```
File: /contracts/src/strategies/HedgedLP.sol
295:
296:         // in case of emergency - withdraw lp tokens from farm
297:         function withdrawFromFarm() public onlyAuth {
298:                 _withdrawFromFarm(_getFarmLp());
299:         }
```

Impact

Events are necessary to notify the off-chain world of successful state transitions. Administration functionalities should emit the corresponding events throughout the system's life cycle to provide credibility and confidence in the system.

In the specific case, if the authorized command "*withdrawFromFarm*" is called in case of emergency, no event will be emitted.

Recommendation

It is recommended to emit an event related to this functionality.

CVSS Score

AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.3 Event not emitted in "removeLiquidity" functionality at "HedgedLP.sol"

Description	LOW

It was identified that the authorized command "*removeLiquidity*" does not emit an event. A contract can emit events when it wants to notify external entities like users, chain explorers, or dApps about changes or conditions in the blockchain. When an event is emitted, it stores the arguments passed in transaction logs. These logs are stored on blockchain and are accessible using address of the contract till the contract is present on the blockchain

The issue exists at:

```
File: /contracts/src/strategies/HedgedLP.sol
307:          // in case of emergency - remove LP
308:          function removeLiquidity(uint256 removeLp) public onlyAuth {
309:                  _removeLiquidity(removeLp);
310:          }
```

Then, the _removeLiquidity depends on the inherited contract. For example:

```
File: /contracts/src/strategies/mixins/IUniLp.sol
34:          function _removeLiquidity(uint256 liquidity)
35:                  internal
36:                  virtual
37:                  override
38:                  returns (uint256, uint256)
39:          {
40:                   IERC20(address(pair())).safeTransfer(address(pair()),
liquidity);
41:                                                  (address   tokenA,   )   =
UniUtils._sortTokens(address(underlying()), address(short()));
42:                         (uint256 amountToken0, uint256 amountToken1) =
pair().burn(address(this));
43:                  return
44:                          tokenA == address(underlying())
45:                                  ? (amountToken0, amountToken1)
46:                                  : (amountToken1, amountToken0);
```

```
47:         }
48:
```

Impact

Events are necessary to notify the off-chain world of successful state transitions. Administration functionalities should emit the corresponding events throughout the system's life cycle in order to provide credibility and confidence in the system.

In the specific case, if the authorized command "*removeLiquidity*" is called in case of emergency, no event will be emitted.

Recommendation

It is recommended to emit an event related to this functionality.

CVSS Score

AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.4 Event not emitted in "redeemCollateral" functionality at "HedgedLP.sol"

Description	LOW

It was identified that the authorized command "*redeemCollateral*" does not emit an event. A contract can emit events when it wants to notify external entities like users, chain explorers, or dApps about changes or conditions in the blockchain. When an event is emitted, it stores the arguments passed in transaction logs. These logs are stored on blockchain and are accessible using address of the contract till the contract is present on the blockchain

The issue exists at:

```
File: /contracts/src/strategies/HedgedLP.sol
301:         // in case of emergency - withdraw stuck collateral
302:               function redeemCollateral(uint256 repayAmnt, uint256
withdrawAmnt) public onlyAuth {
303:                 _repay(repayAmnt);
304:                 _redeem(withdrawAmnt);
305:         }
```

Impact

Events are necessary to notify the off-chain world of successful state transitions. Administration functionalities should emit the corresponding events throughout the system's life cycle in order to provide credibility and confidence in the system.

In the specific case, if the authorized command "*redeemCollateral*" is called in case of emergency, no event will be emitted.

Recommendation

It is recommended to emit an event related to this functionality.

CVSS Score

AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.5 Event not emitted in "mint" functionality at "BaseStrategy.sol"

Description	LOW

It was found It was identified that the authorized command "*mint*" does not emit an event. A contract can emit events when it wants to notify external entities like users, chain explorers, or dApps about changes or conditions in the blockchain. When an event is emitted, it stores the arguments passed in transaction logs. These logs are stored on blockchain and are accessible using address of the contract till the contract is present on the blockchain

The issue exists at:

```
File: /contracts/src/strategies/BaseStrategy.sol
82:   // PUBLIC METHODS
83:   function mint(uint256 amount) external onlyVault returns (uint256
errCode) {
84:         uint256 newShares = _deposit(amount);
85:         _shares += newShares;
86:         errCode = 0;
87:   }
```

Impact

Events are necessary to notify the off-chain world of successful state transitions. Administration functionalities should emit the corresponding events throughout the system's life cycle to provide credibility and confidence in the system.

In the specific case, if the authorized command "*mint*" is called, no event will be emitted.

Recommendation

It is recommended to emit an event related to this functionality.

CVSS Score

AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.6 *Event not emitted in "redeemUnderlying" functionality at "BaseStrategy.sol"*

Description	LOW

It was identified that the authorized command "*redeemUnderlying*" does not emit an event. A contract can emit events when it wants to notify external entities like users, chain explorers, or dApps about changes or conditions in the blockchain. When an event is emitted, it stores the arguments passed in transaction logs. These logs are stored on blockchain and are accessible using address of the contract till the contract is present on the blockchain

The issue exists at:

```
File: /contracts/src/strategies/BaseStrategy.sol
89:   function redeemUnderlying(uint256 amount)
90:         external
91:         override
92:         onlyVault
93:         returns (uint256 errCode)
94:   {
95:         uint256 burnShares = _withdraw(amount);
96:         _shares -= burnShares;
97:         errCode = 0;
98:   }
```

Impact

Events are necessary to notify the off-chain world of successful state transitions. Administration functionalities should emit the corresponding events throughout the system's life cycle in order to provide credibility and confidence in the system.

In the specific case, if the authorized command "*redeemUnderlying*" is called, no event will be emitted.

Recommendation

It is recommended to emit an event related to this functionality.

CVSS Score

AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.7 Lack of validation for minimum delay at "ScionTimelock.sol"

Description	LOW

The team identified that the constructor and the "*updateDelay*" function of "*ScionTimelock.sol*", are not validating the provided delay parameter. Providing incorrect value can lead to unexpected results in other functionalities of the inherited contracts. For example, a very low or zero parameter would make the time lock control completely ineffective, while a very high value can make future management tasks impossible.

The issue exists at:

```
File: /contracts/src/ScionTimelock.sol
70:         constructor(
71:                 uint256 minDelay,
72:                 address[] memory proposers,
73:                 address[] memory executors
74:         ) {
....
93:                 _minDelay = minDelay;
94:                 emit MinDelayChange(0, minDelay);
95:         }
```

And in:

```
File: /contracts/src/ScionTimelock.sol
360:          */
361:         function updateDelay(uint256 newDelay) external virtual {
362:             require(msg.sender == address(this), "TimelockController:
caller must be timelock");
363:                 emit MinDelayChange(_minDelay, newDelay);
364:                 _minDelay = newDelay;
365:         }
```

Impact

If the contract is accidentally initialized with empty, zeroed, or extremely low "*minDelay*" parameter, the time lock control will not be effectively enforced. For example, currently the deploy script sets the minimum delay only to one (1) minute, making it almost impossible for most users to react:

```
File: /contracts-test/deploy/02_timelock.ts
20:
21:   const minDelay = 60 * 1; // 1m
22:
23:   const proposers = [deployer, team1, timelockAdmin];
24:   const executors = [deployer, manager, team1];
25:
26:   await deploy('ScionTimelock', {
27:     from: deployer,
28:     log: true,
29:     skipIfAlreadyDeployed: true,
30:     args: [minDelay, proposers, executors],
31:   });
```

Moreover, setting a high value as minimum delay can disable future management tasks such as changing the trust in strategies, adding strategies, setting fee and harvest window, and controlling the managers.

Furthermore, adversaries who control the operators and executors roles can easily set the new delay to zero (0) using the "*updateDelay*()" function. It should be noted that changing the delay is only possible through the timelock itself. It is thus possible to set it to 0, but only after a duration equal to the old delay.

Recommendation

It is recommended to set hard minimum and maximum delay limits. For example, an implementation as the following can be used:

```
contract ScionTimelock is AccessControl {
...
uint256 public immutable hardMinDelay;
uint256 public immutable hardMaxDelay;
...
```

```
constructor(
                uint256 minDelay,
                address[] memory proposers,
                address[] memory executors
        ) {
        require(minDelay >= hardMinDelay);
        require(minDelay <= hardMaxDelay);
        ...
}
...
function updateDelay(uint256 newDelay) external virtual {
                require(msg.sender == address(this), "TimelockController:
caller must be timelock");
                require(newDelay >= hardMinDelay);
                require(newDelay <= hardMaxDelay);
                emit MinDelayChange(_minDelay, newDelay);
                _minDelay = newDelay;
        }
```

The issue has been discussed in the past also in a similar Timelock contract at OpenZeppelin:

https://github.com/OpenZeppelin/openzeppelin-contracts/issues/2642

CVSS Score

AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:L/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.8 Floating pragma in multiple contracts

Description	LOW

It was found that many smart contracts are using a floating pragma. In Solidity programming, multiple APIs are only be supported in some specific versions. In each contract, the pragma keyword is used to enable certain compiler features or checks. If a contract does not specify a compiler version, developers might encounter compile errors in the future code reuse because of the version gap.

The issue exists at:

- *ScionTimelock.sol:4:pragma solidity ^0.8.0;*
- *interfaces/imx/IImpermax.sol:2:pragma solidity >=0.8.0;*
- *interfaces/uniswap/IUniswapV2Router01.sol:2:pragma solidity ^0.8.0;*
- *interfaces/uniswap/IWETH.sol:2:pragma solidity ^0.8.0;*
- *interfaces/uniswap/IStakingRewards.sol:2:pragma solidity ^0.8.0;*
- *interfaces/uniswap/IUniswapV2Factory.sol:2:pragma solidity ^0.8.0;*
- *interfaces/uniswap/IUniswapV2Pair.sol:2:pragma solidity ^0.8.0;*
- *interfaces/AllowedPermit.sol:2:pragma solidity >=0.8.0;*
- *interfaces/Strategy.sol:2:pragma solidity >=0.8.0;*
- *interfaces/chainlink/AggregatorV2V3Interface.sol:2:pragma solidity ^0.8.0;*
- *interfaces/compound/ICompPriceOracle.sol:2:pragma solidity ^0.8.0;*
- *interfaces/compound/IClaimReward.sol:2:pragma solidity ^0.8.0;*
- *interfaces/compound/ICTokenInterfaces.sol:2:pragma solidity ^0.8.0;*
- *interfaces/compound/IComptroller.sol:2:pragma solidity ^0.8.0;*
- *interfaces/compound/InterestRateModel.sol:2:pragma solidity ^0.8.0;*
- *libraries/FixedPointMathLib.sol:2:pragma solidity >=0.8.0;*
- *libraries/SafeETH.sol:2:pragma solidity >=0.8.0;*
- *libraries/UniUtils.sol:2:pragma solidity ^0.8.0;*
- *libraries/SafeCastLib.sol:2:pragma solidity >=0.8.0;*
- *libraries/Bytes32AddressLib.sol:2:pragma solidity >=0.8.0;*
- *strategies/mixins/ICompound.sol:2:pragma solidity ^0.8.0;*
- *strategies/mixins/ILending.sol:2:pragma solidity ^0.8.0;*
- *strategies/mixins/IFarmableLp.sol:2:pragma solidity ^0.8.6;*
- *strategies/mixins/IBase.sol:2:pragma solidity ^0.8.0;*
- *strategies/mixins/ILp.sol:2:pragma solidity ^0.8.0;*

- *strategies/mixins/IIMXFarm.sol:2:pragma solidity ^0.8.6;*
- *strategies/mixins/IUniLp.sol:2:pragma solidity ^0.8.0;*
- *strategies/mixins/IFarmable.sol:2:pragma solidity ^0.8.0;*
- *strategies/BaseStrategy.sol:2:pragma solidity ^0.8.0;*
- *strategies/IMX.sol:2:pragma solidity ^0.8.0;*
- *strategies/adapters/MiniChefFarm.sol:2:pragma solidity ^0.8.0;*
- *strategies/adapters/CompoundFarm.sol:2:pragma solidity ^0.8.0;*
- *strategies/adapters/MasterChefFarm.sol:2:pragma solidity ^0.8.0;*
- *strategies/adapters/CompMultiFarm.sol:2:pragma solidity ^0.8.0;*
- *strategies/adapters/IMXFarm.sol:2:pragma solidity ^0.8.0;*
- *strategies/adapters/Compound.sol:2:pragma solidity ^0.8.0;*
- *strategies/implementations/USDCmovrSOLARwell.sol:2:pragma solidity ^0.8.0;*
- *strategies/implementations/USDCftmSPIRITscream.sol:2:pragma solidity ^0;*
- *strategies/implementations/USDCimxWEVE.sol:2:pragma solidity ^0.8.0;*
- *strategies/implementations/USDCglmrSTELLAwell.sol:2:pragma solidity ^0.8.0;*
- *strategies/implementations/USDCavaxJOEqi.sol:2:pragma solidity ^0.8.0;*
- *strategies/implementations/USDCavaxPNGqi.sol:2:pragma solidity ^0.8.0;*
- *strategies/implementations/USDCftmSPOOKYscream.sol:2:pragma solidity ^0.8.0;*
- *strategies/HedgedLP.sol:2:pragma solidity ^0.8.0;*
- *vault/ScionBeaconProxy.sol:2:pragma solidity ^0.8.10;*
- *vault/ScionVaultFactoryV0.sol:2:pragma solidity >=0.8.0;*
- *vault/VaultUpgradable.sol:2:pragma solidity >=0.8.0;*
- *vault/modules/VaultRouterModule.sol:2:pragma solidity >=0.8.0;*
- *vault/ScionVaultFactory.sol:2:pragma solidity >=0.8.0;*

For example, in case of "*ScionTimelock.sol*" where the following directive exists "*pragma solidity ^0.8.0;",* the source file with the line above does not compile with a compiler earlier than version 0.8.9, and it also does not work on a compiler starting from version 0.9.0 (this second condition is added by using ^). The exact version of the compiler is not fixed, so that bugfix releases are still possible.

In other cases, such as "*vault/modules/VaultRouterModule.sol*", where the "pragma solidity >=0.8.0" is defined, any future version will be accepted.

Impact

Since different versions of Solidity may contain different APIs, developers might encounter compile errors in the future code reuse.

Recommendation

Source files should be annotated with a pragma version to reject compilation with previous or future compiler versions that might introduce incompatible changes.

It is advisable to remove the ">=" directives for the compiler version. Furthermore, it is recommended to avoid using the "^" directive to avoid using nightly builds.

CVSS Score

AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.9 Outdated compiler version supported at "USDCftmSPIRITscream.sol"

Description	LOW

The team identified that the "*USDCftmSPIRITscream*.sol" strategy specifies a less secure and outdated Solidity compiler version. In Solidity programming, multiple APIs only be supported in some specific versions. In each contract, the pragma keyword is used to enable certain compiler features or checks.

The issue exists at:

```
File:contractst/src/strategies/implementations/USDCftmSPIRITscream.sol
2: pragma solidity ^0;
```

The source file with the line above will compile with any compiler that starts with 0, as far as it supports the pragma directives.

For example, version 0.6.12 can be used to compile the file.

Impact

Developers are allowed to use outdated and less secure compilers to compile the specific file. For example, if the compiler 0.6.12 is selected, safe math will not be enabled by default.

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version. For example, if the 0.6.12 is used, the known bugs can be found at:

https://docs.soliditylang.org/en/v0.8.15/bugs.html

Recommendation

It is recommended to specify a specific, more updated compiler version.

CVSS Score

AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.10 Unvalidated parameters in "setMaxTvl()", "setSafeCollateralRatio()", "setMinLoanHeath()", "setMaxPriceMismatch()" and "setRebalanceThreshold()" at "HedgedLP.sol" and "IMX.sol"

Description	LOW

It was found The team identified that the *"setMaxTvl()", "setSafeCollateralRatio()", "setMinLoanHeath()", "setMaxPriceMismatch()"* and *"setRebalanceThreshold()*" functions of HedgedLP and IMX strategies are not validating the provided numeric input parameters. Providing incorrect values such as zero amounts can lead to unexpected results in other functionalities of the affected contracts.

The issue is located at:

```
File: /contracts/src/strategies/HedgedLP.sol
108:
109:         function setMaxTvl(uint256 maxTvl_) public onlyAuth {
110:                 _maxTvl = maxTvl_;
111:                 emit SetMaxTvl(maxTvl_);
112:         }
```

```
File: /contracts/src/strategies/HedgedLP.sol
82:         function setSafeCollateralRatio(uint256 safeCollateralRatio_)
public onlyOwner {
83:                 _safeCollateralRatio = safeCollateralRatio_;
84:                 emit SetSafeCollateralRaio(safeCollateralRatio_);
85:         }
```

```
File: /contracts/src/strategies/HedgedLP.sol
092:             function setMinLoanHeath(uint256 minLoanHealth_) public
onlyOwner {
093:                 minLoanHealth = minLoanHealth_;
094:                 emit setMinLoanHealth(minLoanHealth_);
095:         }
096:
```

```
File: /contracts/src/strategies/HedgedLP.sol
098:        function setMaxPriceMismatch(uint256 maxPriceMismatch_) public
onlyAuth {
099:                  require(msg.sender == owner() || maxAllowedMismatch
>= maxPriceMismatch_, "HLP: TOO LARGE");
100:                 maxPriceMismatch = maxPriceMismatch_;
101:                 emit SetMaxPriceMismatch(maxPriceMismatch_);
102:         }
```

```
File: /contracts/src/strategies/HedgedLP.sol
104:           function setRebalanceThreshold(uint16 rebalanceThreshold_)
public onlyOwner {
105:                 rebalanceThreshold = rebalanceThreshold_;
106:                 emit SetRebalanceThreshold(rebalanceThreshold_);
107:         }
108:
```

Impact

If the functions are accidentally set with empty / zeroed parameters, other functionalities will not be usable.

Recommendation

It is advisable to verify that the important parameters are not zero.

CVSS Score

AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:BI20X/MA:X

5.3.11 No return value in functions at "IMXFarm.sol"

Description	LOW

The team identified that the "*_addLiquidity*()" and "*_harvestFarm*()" functionalities of the IMXFarm adapter do not return any value, even if a return type is specified in their definition.

The issue exists in the following location:

```
File: /contracts/src/strategies/adapters/IMXFarm.sol
51:               function _addLiquidity(uint256 amntUnderlying, uint256
amntShort)
52:                 internal
53:                 virtual
54:                 override
55:                 returns (uint256)
56:         {
57:                 uint256 borrowU = (_optimalUBorrow() * amntUnderlying)
/ 1e18;
58:             uint256 borrowS = amntShort + _underlyingToShort(borrowU);
59:
60:                 bytes memory data = abi.encode(amntUnderlying);
61:                       sBorrowable.borrowApprove(address(sBorrowable),
amntShort);
62:
63:                 // mint collateral
64:                 bytes memory borrowBData = abi.encode(
65:                         CalleeData({
66:                             callType: CallType.ADD_LIQUIDITY_AND_MINT,
67:                                 data: abi.encode(
68:                                         AddLiquidityAndMintCalldata({
uAmnt: amntUnderlying + borrowU, sAmnt: borrowS })
69:                                 )
70:                         })
71:                 );
72:                 // borrow borrowableB
73:                 bytes memory borrowAData = abi.encode(
```

```
74:                         CalleeData({
75:                                 callType: CallType.BORROWB,
76:                                    data: abi.encode(BorrowBCalldata({
borrowAmount: borrowU, data: borrowBData }))
77:                         })
78:                 );
79:
80:                 // flashloan borrow then add lp
81:                     sBorrowable.borrow(address(this), address(this),
borrowS, borrowAData);
82:         }
83:
```

```
File: /contracts/src/strategies/adapters/IMXFarm.sol
202:         function _harvestFarm(HarvestSwapParms[] calldata swapParams)
203:                 internal
204:                 override
205:                 returns (uint256[] memory harvested)
206:         {}
```

Impact

When declaring a function with a return type definition, it is expected that a value will be returned. Other application components might try to unsuccessfully verify or use this value.

Recommendation

It is recommended to examine if a return value is required or the definition should change.

CVSS Score

AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.4 Informational Findings

5.4.1 *Unvalidated parameters in "mint()" and "redeemUnderlying()" at "BaseStrategy.sol"*

Description	INFO

The team identified that the "*mint()*" and "*redeemUnderlying*()" functions of BaseStrategy are not validating the provided numeric input parameters. Providing incorrect values such as zero amounts can lead to unexpected results.

The issue is located at:

```
File: /contracts/src/strategies/BaseStrategy.sol
83:             function mint(uint256 amount) external onlyVault returns
(uint256 errCode) {
84:                 uint256 newShares = _deposit(amount);
85:                 _shares += newShares;
86:                 errCode = 0;
87:         }
```

```
File: /contracts/src/strategies/BaseStrategy.sol
89:         function redeemUnderlying(uint256 amount)
90:                 external
91:                 override
92:                 onlyVault
93:                 returns (uint256 errCode)
94:         {
95:                 uint256 burnShares = _withdraw(amount);
96:                 _shares -= burnShares;
97:                 errCode = 0;
98:         }
```

Impact

If the functions are accidentally set with empty / zeroed parameters, no operation will occur.

This issue is marked as INFORMATIONAL.

Recommendation

It is advisable to verify that the important parameters are not zero.

CVSS Score

AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.4.2 Debugging utility in "HedgedLP.sol"

Description	INFO

It was identified that the HedgedLP contract contains a debugging utility. Hardhat comes built-in with Hardhat Network, a local Ethereum network designed for development. When running contracts and tests on Hardhat Network it is possible to print logging messages and contract variables calling *console.log()* from the Solidity code. To perform this operation, it is required to import "*hardhat/console.sol"* in the contract code.

```
File: /contracts/src/strategies/HedgedLP.sol
14: import "hardhat/console.sol";
```

Impact

Since no logging functionality is used, the issue is marked as INFORMATIONAL

Recommendation

It is recommended to remove hardhat console from production code.

CVSS Score

AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

6 Retest Results

6.1 Retest of High Severity Findings

All the High-risk vulnerabilities were found to be successfully addressed:

- *5.1.1 - No upper bound in max slippage potentially allows Sandswitch attack at "HedgedLP.sol" and "IMX.sol"*
- *5.1.2 - Flash Loan attack using "closePosition()", "removeLiquidity()", "rebalance()" and "rebalanceLoan()' functionalities at "HedgedLP.sol" and "IMX.sol"*

A check for max 20% slippage has been added:

```
File: /contracts-dev-
ab3a7a862f655e240b1bf8d3b66ca1bb4e286405/src/strategies/HedgedLP.sol
29:         uint256 public constant maxPriceOffset = 2000; // maximum
offset for rebalanceLoan & manager  methods 20%
...
45:         modifier checkPrice(uint256 maxSlippage) {
46:                 if (maxSlippage == 0)
47:                         ...
49:                 else require(maxSlippage <= maxPriceOffset ||
isGuardian(msg.sender), ""HLP: MAX_MISMATCH"");
50:                 ...
52:         }
53:
```

The closePosition and removeLiquidity methods are now onlyGuardian, where guardian is a cold wallet.

```
File: /contracts-dev-
ab3a7a862f655e240b1bf8d3b66ca1bb4e286405/src/strategies/HedgedLP.sol
311:         // note: one should call harvest immediately before close
position
312:         function closePosition(uint256 maxSlippage) public
checkPrice(maxSlippage) onlyGuardian {
313:                 _closePosition();
314:                 emit UpdatePosition();
315:         }
316:
```

```
317:         // in case of emergency - remove LP
318:         function removeLiquidity(uint256 removeLp, uint256
maxSlippage)
319:                 public
320:                 checkPrice(maxSlippage)
321:                 onlyGuardian
322:         {
323:                 _removeLiquidity(removeLp);
324:                 emit UpdatePosition();
325:         }
326:
```

In IMX.sol the closePosition is onlyGuardian and rebalance code is commented out.

6.2 Retest of Medium Severity Findings

The following MEDIUM-risk vulnerabilities were found to be successfully addressed:

- *5.2.1 - Excessive and Inconsistent permissions in emergency functionalities "withdrawFromFarm()", "redeemCollateral()", "removeLiquidity()" at "HedgedLP.sol"*
- *5.2.3 - Excessive and Inconsistent permissions in "setMaxPriceMismatch" functionality potentially allows Sandwitch attack at "HedgedLP.sol" and "IMX.sol"*

The onlyGuardian permission was introduced:

```
File: /contracts-dev-
ab3a7a862f655e240b1bf8d3b66ca1bb4e286405/src/strategies/HedgedLP.sol
103:         // guardian can adjust max default price mismatch if needed
104:         function setMaxDefaultPriceMismatch(uint256
maxDefaultPriceMismatch_) public onlyGuardian {
105:                 require(maxDefaultPriceMismatch_ >= 25, ""HLP:
BAD_INPUT""); // no less than .25%
106:                 require(
107:                         msg.sender == owner() || maxAllowedMismatch >=
maxDefaultPriceMismatch_,
108:                         ""HLP: BAD_INPUT""
109:                 );
```

```
110:                 maxDefaultPriceMismatch = maxDefaultPriceMismatch_;
111:                 emit
SetMaxDefaultPriceMismatch(maxDefaultPriceMismatch_);
112:         }
317:         // in case of emergency - remove LP
318:         function removeLiquidity(uint256 removeLp, uint256
maxSlippage)
319:                 public
320:                 checkPrice(maxSlippage)
321:                 onlyGuardian
322:         {
323:                 _removeLiquidity(removeLp);
324:                 emit UpdatePosition();
325:         }
326:
327:         // in case of emergency - withdraw lp tokens from farm
328:         function withdrawFromFarm() public onlyGuardian {
329:                 _withdrawFromFarm(_getFarmLp());
330:                 emit UpdatePosition();
331:         }
332:
333:         // in case of emergency - withdraw stuck collateral
334:         function redeemCollateral(uint256 repayAmnt, uint256
withdrawAmnt) public onlyGuardian {
335:                 _repay(repayAmnt);
336:                 _redeem(withdrawAmnt);
337:                 emit UpdatePosition();
338:         }
```

And in IMX:

```
File: /contracts-dev-
ab3a7a862f655e240b1bf8d3b66ca1bb4e286405/src/strategies/IMX.sol
113:         // manager can adjust max price if needed
114:         function setMaxPriceMismatch(uint256 maxPriceMismatch_) public
onlyGuardian {
115:                 require(msg.sender == owner() || maxAllowedMismatch >=
maxPriceMismatch_, ""HLP: TOO LARGE"");
116:                 maxPriceMismatch = maxPriceMismatch_;
117:                 emit SetMaxPriceMismatch(maxPriceMismatch_);
118:         }
```

```
317:          // in case of emergency - remove LP
318:          function removeLiquidity(uint256 removeLp, uint256
maxSlippage)
319:                  public
320:                  checkPrice(maxSlippage)
321:                  onlyGuardian
322:          {
323:                  _removeLiquidity(removeLp);
324:                  emit UpdatePosition();
325:          }
326:
327:          // in case of emergency - withdraw lp tokens from farm
328:          function withdrawFromFarm() public onlyGuardian {
329:                  _withdrawFromFarm(_getFarmLp());
330:                  emit UpdatePosition();
331:          }
332:
333:          // in case of emergency - withdraw stuck collateral
334:          function redeemCollateral(uint256 repayAmnt, uint256
withdrawAmnt) public onlyGuardian {
335:                  _repay(repayAmnt);
336:                  _redeem(withdrawAmnt);
337:                  emit UpdatePosition();
338:          }
```

And for rebalance and rebalanceLoan:

```
File: /contracts-dev-
ab3a7a862f655e240b1bf8d3b66ca1bb4e286405/src/strategies/HedgedLP.sol
138:          function rebalanceLoan() public nonReentrant {
139:                  // limit offset to maxPriceOffset manager to prevent
misuse
140:                  if (isGuardian(msg.sender)) {} else if
(isManager(msg.sender))
141:                          require(getPriceOffset() <= maxPriceOffset,
""HLP: MAX_MISMATCH"");
….
}
…
290:          function rebalance(uint256 maxSlippage)
291:                  external
```

```
292:                  onlyManager
293:                  checkPrice(maxSlippage)
46:                  if (maxSlippage == 0)
47:                          maxSlippage = maxDefaultPriceMismatch;
48:                          // manager accounts cannot set maxSlippage
bigger than maxPriceOffset
49:                  else require(maxSlippage <= maxPriceOffset ||
isGuardian(msg.sender), ""HLP: MAX_MISMATCH"");
…
```

The risk of the following MEDIUM-risk vulnerabilities was accepted:

- *5.2.2 - Unvalidated addresses in governance functionalies "setVault()" and "emergencyWithdraw()" at "BaseStrategy.sol"*
- *5.2.4 - Insufficient and Inconsistent time lock delay set at "02_timelock.ts"*
- *5.2.5 - Order of operations can introduce round-off errors in "HedgedLP.sol"*
- *5.2.6 - Order of operations can introduce round-off errors in "IMXFarm.sol"*

The team responded that the probability of passing address(0) is similar to passing an non-existent or incorrect address. Furthermore, the inconsistent lock delay is an intentional setting for initial deployment that will be updated to the necessary delay via the UI and can be verified Scion UI and Etherscan. Finally, Scion team reported that small imprecision due to order of operations can be tolerated in the reported case.s

6.3 Retest of Low Severity Findings

The following Low-risk vulnerabilities were found to be successfully addressed:

- *5.3.1 - Event not emitted in "closePosition" functionality at "HedgedLP.sol"*
- *5.3.2 - Event not emitted in "withdrawFromFarm" functionality at "HedgedLP.sol"*
- *5.3.3 - Event not emitted in "removeLiquidity" functionality at "HedgedLP.sol"*
- *5.3.4 - Event not emitted in "redeemCollateral" functionality at "HedgedLP.sol"*
- *5.3.5 - Event not emitted in "mint" functionality at "BaseStrategy.sol"*
- *5.3.6 - Event not emitted in "redeemUnderlying" functionality at "BaseStrategy.sol"*
- *5.3.9 - Outdated compiler version supported at "USDCftmSPIRITscream.sol"*

- *5.3.10 - Unvalidated parameters in "setMaxTvl()", "setSafeCollateralRatio()", "setMinLoanHeath()", "setMaxPriceMismatch()" and "setRebalanceThreshold()" at "HedgedLP.sol" and "IMX.sol"*

The following events are now emitted:

```
File: /contracts-dev-ab3a7a862f655e240b1bf8d3b66ca1bb4e286405/src/strategies/HedgedLP.sol
311:            // note: one should call harvest immediately before close position
312:             function closePosition(uint256 maxSlippage) public checkPrice(maxSlippage) onlyGuardian {
313:                 _closePosition();
314:                 emit UpdatePosition();
315:         }
316:
317:         // in case of emergency - remove LP
318:         function removeLiquidity(uint256 removeLp, uint256 maxSlippage)
319:                 public
320:                 checkPrice(maxSlippage)
321:                 onlyGuardian
322:         {
323:                 _removeLiquidity(removeLp);
324:                 emit UpdatePosition();
325:         }
327:         // in case of emergency - withdraw lp tokens from farm
328:         function withdrawFromFarm() public onlyGuardian {
329:                 _withdrawFromFarm(_getFarmLp());
330:                 emit UpdatePosition();
331:         }
333:         // in case of emergency - withdraw stuck collateral
334:                 function redeemCollateral(uint256 repayAmnt, uint256 withdrawAmnt) public onlyGuardian {
335:                 _repay(repayAmnt);
336:                 _redeem(withdrawAmnt);
337:                 emit UpdatePosition();
338:         }
```

And in BaseStrategy:

```
File: /contracts-dev-ab3a7a862f655e240b1bf8d3b66ca1bb4e286405/src/strategies/BaseStrategy.sol
91:   function mint(uint256 amount) external onlyVault returns (uint256 errCode) {
92:         uint256 newShares = _deposit(amount);
93:         _shares += newShares;
94:         errCode = 0;
95:         emit Deposit(msg.sender, amount);
96:   }
098:          function redeemUnderlying(uint256 amount)
099:                  external
100:                 override
101:                 onlyVault
102:                 returns (uint256 errCode)
103:         {
104:                 uint256 burnShares = _withdraw(amount);
105:                 _shares -= burnShares;
106:                 errCode = 0;
107:                 emit Withdraw(msg.sender, amount);
108:         }
109:
```

The compiler support has been configured:

```
File: /contracts-dev-ab3a7a862f655e240b1bf8d3b66ca1bb4e286405/src/strategies/implementations/USDCftmSPIRITscream.sol
1: // SPDX-License-Identifier: MIT
2: pragma solidity 0.8.16;
```

And a parameter validation control has been introduced, with the exception of maxTVL in which the zero value is allowed to be used:

```
File: /contracts-dev-ab3a7a862f655e240b1bf8d3b66ca1bb4e286405/src/strategies/HedgedLP.sol
02: 86:        function setSafeCollateralRatio(uint256 safeCollateralRatio_) public onlyOwner {
03:  87:                            require(safeCollateralRatio_ >= 1000 && safeCollateralRatio_ <= 8500, ""HLP: BAD_INPUT"");
04: 88:                 _safeCollateralRatio = safeCollateralRatio_;
05: 89:                 emit SetSafeCollateralRaio(safeCollateralRatio_);
06: 90:         }
```

```
07: 91:
08: 095:
09: 096:          // OWNER CONFIG
10: 097:             function setMinLoanHeath(uint256 minLoanHealth_) public
onlyOwner {
11: 098:                  require(minLoanHealth_ > 1e18, ""HLP: BAD_INPUT"");
12: 099:                  minLoanHealth = minLoanHealth_;
13: 100:                  emit setMinLoanHealth(minLoanHealth_);
14: 101:          }
15: 102:
16: 103:         // guardian can adjust max default price mismatch if needed
17:  104:                       function  setMaxDefaultPriceMismatch(uint256
maxDefaultPriceMismatch_) public onlyGuardian {
18: 105:                     require(maxDefaultPriceMismatch_ >= 25, ""HLP:
BAD_INPUT""); // no less than .25%
19: 106:                  require(
20: 107:                         msg.sender == owner() || maxAllowedMismatch
>= maxDefaultPriceMismatch_,
21: 108:                          ""HLP: BAD_INPUT""
22: 109:                  );
23: 110:                 maxDefaultPriceMismatch = maxDefaultPriceMismatch_;
24:   111:                                                                emit
SetMaxDefaultPriceMismatch(maxDefaultPriceMismatch_);
25: 112:          }
26: 113:
27: 114:          function setRebalanceThreshold(uint16 rebalanceThreshold_)
public onlyOwner {
28: 115:                  // rebalance threshold should not be lower than 1%
(2% price move)
29: 116:                         require(rebalanceThreshold_ >= 100, ""HLP:
BAD_INPUT"");
30: 117:                  rebalanceThreshold = rebalanceThreshold_;
31: 118:                  emit SetRebalanceThreshold(rebalanceThreshold_);
32: 119:          }
33: 120:
34: 121:          function setMaxTvl(uint256 maxTvl_) public onlyGuardian {
35: 122:                  _maxTvl = maxTvl_;
36: 123:                  emit SetMaxTvl(maxTvl_);
37: 124:          }
38:
```

However, the issues have not been fixed in IMX. It should be noted that all related functionalities in IMX are commented out, and as a result the issues are marked as CLOSED.

One (1) Low-risk vulnerability was partially fixed, and its risk was downgraded to INFORMATIONAL issue:

- *5.3.8 - Floating pragma in multiple contracts*

. The following floating pragma still exists:

```
src/strategies/mixins/IIMXFarm.sol:2:pragma solidity ^0.8.6;
```

The risk of the following Low-risk vulnerabilities was accepted:

- *5.3.7 - Lack of validation for minimum delay at "ScionTimelock.sol"*
- *5.3.11 - No return value in functions at "IMXFarm.sol"*

The team responded that they a check for the minimum delay will be implemented on the front-end, while the WIP is not in scope of the audit.

6.4 Retest of Informational Findings

The following INFORMATIONAL issue was found to be successfully addressed:

- *5.4.2 - Debugging utility in "HedgedLP.sol"*

The import was commented out:

```
File:                                                    /contracts-dev-
ab3a7a862f655e240b1bf8d3b66ca1bb4e286405/src/strategies/HedgedLP.sol
14: // import "hardhat/console.sol";
15:
```

While the following issue remains OPEN:

- *5.4.1 - Unvalidated parameters in "mint()" and "redeemUnderlying()" at "BaseStrategy.sol"*

References & Applicable Documents

Ref.	Title	Version
N/A	*N/A*	*N/A*

Document History

Revision	Description	Changes Made By	Date
0.2	*Initial Draft*	*Chaintroopers*	*July 29th, 2022*
1.0	*First Version*	*Chaintroopers*	*August 1st, 2022*
1.1	*Added retest results*	*Chaintroopers*	*September 9th, 2022*