ithacaprotocol.io - EVM Contracts

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **December 19th, 2023 - January 5th, 2024**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	01/02/2024
0.2	Draft Review	01/05/2024
0.3	Draft Review	01/05/2024
1.0	Remediation Plan	01/29/2024
1.1	Remediation Plan Review	01/30/2024
1.2	Remediation Plan Review	01/30/2024

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

The ithacaprotocol.io EVM smart contracts are a decentralized options and collateralized funding protocol.

ithacaprotocol.io engaged Halborn to conduct a security assessment on their smart contracts beginning on December 19th, 2023 and ending on January 5th, 2024. The security assessment was scoped to the smart contracts provided in the ithaca-protocol/ithaca-sc GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

1.2 ASSESSMENT SUMMARY

Halborn was provided two weeks for the engagement and assigned a full-time security engineer to review the security of the smart contracts in scope. The security team consists of a blockchain and smart contract security experts with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some security risks, which were mostly addressed by ithacaprotocol.io. The main ones were the following:

- Distribute the yield correctly during withdrawals.
- Use `forceApprove()` instead of `approve()` to also handle non-standard-compliant tokens.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (solgraph).
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Static Analysis of security for scoped contract, and imported functions (Slither).
- Testnet deployment (Foundry, Brownie).

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A)	1
	Specific (AO:S)	0.2
Attack Cost (AC)	Low (AC:L)	1
	Medium (AC:M)	0.67
	High (AC:H)	0.33
Attack Complexity (AX)	Low (AX:L)	1
	Medium (AX:M)	0.67
	High (AX:H)	0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N)	1
	Partial (R:P)	0.5
	Full (R:F)	0.25
Scope (s)	Changed (S:C)	1.25
	Unchanged (S:U)	1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

Code repositories:

1. ithacaprotocol.io Smart Contracts

- Repository: ithaca-protocol/ithaca-sc
- Commit ID: 1c4a399fd8d28083072d8c42832e71250c441c28
- Smart contracts in scope:
    - contracts/strategies/Strategy.sol
    - contracts/strategies/AaveV3Strategy.sol
    - contracts/registry/Registry.sol
    - contracts/ledger/Ledger.sol
    - contracts/ledger/proxy/LedgerBeaconProxy.sol
    - contracts/ledger/proxy/LedgerBeacon.sol
    - contracts/fundlock/Fundlock.sol
    - contracts/access/AccessController.sol
    - contracts/access/AccessRestricted.sol
    - contracts/access/Roles.sol
    - contracts/validator/TokenValidator.sol
- Fixed commit ID (Final): d09fef4d35a28e97ead2ee02402417f9b1a2642d

Out-of-scope

- Third-party libraries and dependencies.
- Economic attacks.

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	1	3	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) UNCLAIMED YIELD IS LOST DURING WITHDRAWALS	High (7.5)	SOLVED - 01/29/2024
(HAL-02) APPROVE IS INCOMPATIBLE WITH NON-STANDARD ERC20 TOKENS	Medium (5.0)	SOLVED - 01/06/2024
(HAL-03) FUNDS CANNOT BE UNUTILIZED WITH ADJUSTFUND	Low (2.5)	SOLVED - 01/06/2024
(HAL-04) INACCURATE TOTALVALUE CALCULATION	Low (2.5)	SOLVED - 01/29/2024
(HAL-05) INCOMPATIBILITY WITH TRANSFER-ON-FEE OR DEFLATIONARY TOKENS	Low (2.5)	RISK ACCEPTED
(HAL-06) MISSING TOKEN VALIDATION CAN LEAD TO READ IMPROPER DATA	Informational (0.8)	ACKNOWLEDGED
(HAL-07) USING GAS INEFFICIENT PARAMETER STORAGE	Informational (0.0)	SOLVED - 01/06/2024

FINDINGS & TECH DETAILS

4.1 (HAL-01) UNCLAIMED YIELD IS LOST DURING WITHDRAWALS - HIGH (7.5)

Description:

It was identified that the withdraw() function in the Fundlock contract does not credit the latest yield to the user. It is noted that, the yield is lost and cannot be claimed later because the user's yield index is updated.

Code Location:

The balance is stored in a local variable before calling the distributeYield() function. The unclaimed yield is lost because the function uses the local variable without updating it for further calculation.

Listing 1: contracts/fundlock/Fundlock.sol (Lines 85-95)

```
79     function withdraw(address token, uint256 amount) external {
80         if (amount == 0) revert ZeroAmount();
81         address tokenValidator = IRegistry(registry).
   ↳ tokenValidator();
82         if (!ITokenValidator(tokenValidator).isWhitelisted(token))
   ↳  {
83             revert NotWhitelisted(token);
84         }
85         uint256 balance = _balances[msg.sender][token];
86         if (amount > balance) {
87             revert InsufficientFunds(amount, balance);
88         }
89 
90         address tokenStrategy = tokenStrategies[token];
91         if (tokenStrategy != address(0)) {
92             IStrategy(tokenStrategy).distributeYield(msg.sender);
93         }
94 
95         _balances[msg.sender][token] = balance - amount;
96         uint8 slot = _findEmptySlot(msg.sender, token);
97         _fillSlot(msg.sender, token, slot);
```

```
        _withdrawals[msg.sender][token][slot] = Withdrawal({
            amount: amount,
            timestamp: uint32(block.timestamp)
        });

        emit Withdraw(msg.sender, token, amount, slot);
    }
```

Proof of Concept:

The unclaimed yield is not distributed to client2 during withdrawal:

```
  AaveV3Strategy
    Yield distribution
Test yield distribution during withdrawal:
  client1: deposits 1000000000
  client2: deposits 1000000000
  client1 fundlock balance: BigNumber { value: "1000000000" }
  client2 fundlock balance: BigNumber { value: "1000000000" }
  client1: calls distributeYield
  client2: withdraws 1000000000
  client1 fundlock balance: BigNumber { value: "1012417955" }
  client2 fundlock balance: BigNumber { value: "0" }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:H/R:N/S:U (7.5)

Recommendation:

The yield should be distributed before the balance is read during the withdrawal.

Remediation Plan:

SOLVED: The ithacaprotocol.io team solved the issue in commit 9ef13e1 by fixing the yield distribution.

4.2 (HAL-02) APPROVE IS INCOMPATIBLE WITH NON-STANDARD ERC20 TOKENS - MEDIUM (5.0)

Description:

Some tokens do not correctly implement the EIP20 standard, and their approve function returns void instead of a success boolean. Calling these functions with the correct EIP20 function signatures will always revert. Tokens that do not correctly implement the latest EIP20 spec, like USDT on Ethereum, will be unusable in the mentioned contracts as they revert the transaction because of the missing return value.

Some tokens also require that the allowance be set to 0 before issuing a new approve call. Calling the approve function when the allowance is not zero reverts the transaction with these types of tokens.

Code Location:

The approve() function is used in the AaveV3Strategy contract:

Listing 2: contracts/strategies/AaveV3Strategy.sol (Line 93)

```
    function _utilize(uint256 _amount) internal override {
        _lastAccruedYield += _accruedYieldLatest();
        _lastGeneratedYield = 0;
        IERC20(supplyingAsset).approve(aavePool, _amount);
        IAaveV3Pool(aavePool).supply(supplyingAsset, _amount,
↳ address(this), 0);
    }
```

The forceApprove() function of the SafeERC20 wrapper is meant to be used with tokens that require the approval to be set to zero before setting it to a non-zero value or having no return value, such as USDT.

Listing 3: @openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol

```
    /**
     * @dev Set the calling contract's allowance toward `spender`
↳ to `value`. If `token` returns no value,
     * non-reverting calls are assumed to be successful. Meant to
↳ be used with tokens that require the approval
     * to be set to zero before setting it to a non-zero value,
↳ such as USDT.
     */
    function forceApprove(IERC20 token, address spender, uint256
↳ value) internal {
        bytes memory approvalCall = abi.encodeCall(token.approve,
↳ (spender, value));

        if (!_callOptionalReturnBool(token, approvalCall)) {
            _callOptionalReturn(token, abi.encodeCall(token.
↳ approve, (spender, 0)));
            _callOptionalReturn(token, approvalCall);
        }
    }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:H/A:N/D:N/Y:N/R:N/S:U (5.0)

Recommendation:

It is recommended to use OpenZeppelin's SafeERC20 and the forceApprove() function to also handle non-standard-compliant tokens.

Remediation Plan:

SOLVED: The ithacaprotocol.io team solved the issue in commit 8c58c167 by using OpenZeppelin's SafeERC20 and the forceApprove() function.

4.3 (HAL-03) FUNDS CANNOT BE UNUTILIZED WITH ADJUSTFUND - LOW (2.5)

Description:

It was identified that it is only possible to withdraw funds from the strategies by releasing withdrawals. If the strategy earns funds or the managing ratio setting is decreased by the admin with the setMaxManagingRatio() function, the additional funds will not be withdrawn from the strategy. Therefore, the managing ratio correlates to 100%, and it is not possible to lower it.

Code Location:

The adjustFund() function does not withdraw the excess funds from the strategy. The managing ratio can only be increased:

Listing 4: contracts/strategies/Strategy.sol

```
    function adjustFund() external onlyRole(UTILITY_ACCOUNT_ROLE)
↳ {
        uint256 expectUtilizeAmount = (totalValueAll() *
↳ maxManagingRatio) /
            MULTIPLIER;
        uint256 managingFund_ = managingFund();
        if (expectUtilizeAmount > managingFund_) {
            uint256 _shortage = expectUtilizeAmount -
↳ managingFund_;
            IFundlock(fundlock).utilizeFund(supplyingAsset,
↳ _shortage);
            _utilize(_shortage);
            emit FundPulled(fundlock, _shortage);
        }
    }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:C/D:N/Y:N/R:F/S:U (2.5)

Recommendation:

It is recommended to modify the adjustFund() function not just to supply the shortage, but also to withdraw the excess over the managing ratio from the strategy.

Remediation Plan:

SOLVED: The ithacaprotocol.io team solved the issue in commit ff9a7241 by implementing the recommendation.

4.4 (HAL-04) INACCURATE TOTALVALUE CALCULATION - LOW (2.5)

Description:

It was identified that the totalValueAll() function does not exclude the funds stored in the withdrawal queue from the return value. This results in an inaccurate yield calculation, as the strategy does not earn interest on funds stored in the withdrawal queue.

Code Location:

Listing 5: contracts/strategies/Strategy.sol (Line 78)

```
61     function managingFund() public view returns (uint256) {
62         return IERC20(yieldBearingAsset).balanceOf(address(this));
63     }
64
65     function currentManagingRatio() external view returns (uint256
 ↳ ) {
66         return (managingFund() * MULTIPLIER) / totalValueAll();
67     }
68
69     function availableFund() public view returns (uint256) {
70         return IERC20(supplyingAsset).balanceOf(fundlock);
71     }
72
73     function totalValueAll() public view returns (uint256) {
74         return availableFund() + managingFund();
75     }
76
77     function adjustFund() external onlyRole(UTILITY_ACCOUNT_ROLE)
 ↳ {
78         uint256 expectUtilizeAmount = (totalValueAll() *
 ↳ maxManagingRatio) /
79             MULTIPLIER;
80         uint256 managingFund_ = managingFund();
81         if (expectUtilizeAmount > managingFund_) {
82             uint256 _shortage = expectUtilizeAmount -
 ↳ managingFund_;
```

```
                IFundlock(fundlock).utilizeFund(supplyingAsset, _shortage);
                _utilize(_shortage);
                emit FundPulled(fundlock, _shortage);
            }
        }
```

The totalValueAll() function is used in the yield calculation:

Listing 6: contracts/strategies/AaveV3Strategy.sol (Lines 39-43)

```
36      function distributeYield(address account) external override {
37          uint256 totalYield = _lastAccruedYield + _accruedYieldLatest();
38          uint256 yieldGenerated = totalYield - _lastGeneratedYield;
39          uint256 totalValueAll_ = totalValueAll();
40          if (totalValueAll_ != 0) {
41              _yieldIndex +=
42                  (yieldGenerated * MULTIPLIER) /
43                  (totalValueAll_ - totalYield);
44              _lastGeneratedYield += yieldGenerated;
45              uint256 shares = IFundlock(fundlock).balanceSheet(
46                  account,
47                  supplyingAsset
48              );
49              uint256 userYield_ = (shares *
50                  (_yieldIndex - _yieldIndexOf[account])) / MULTIPLIER;
51              _yieldIndexOf[account] = _yieldIndex;
52              if (userYield_ > 0) {
53                  IFundlock(fundlock).distributeYield(
54                      account,
55                      supplyingAsset,
56                      userYield_
57                  );
58              }
59          }
60      }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:C/D:N/Y:N/R:F/S:U (2.5)

Recommendation:

It is recommended to correct the total value calculation in the strategy contracts.

Remediation Plan:

SOLVED: The ithacaprotocol.io team solved the issue in commit d09fef4d by deducting the funds stored in the withdrawal queue from the return value in the totalValueAll() function.

4.5 (HAL-05) INCOMPATIBILITY WITH TRANSFER-ON-FEE OR DEFLATIONARY TOKENS - LOW (2.5)

Description:

It was identified that the several contracts assume that the safeTransferFrom() and safeTransfer() calls transfer the full amount of tokens. This may not be true if the tokens being transferred are fee-on-transfer tokens, causing the received amount to be lesser than the accounted amount. For example, DGX (Digix Gold Token) and CGT (CACHE Gold) tokens apply transfer fees, and the USDT (Tether) token also has a currently disabled fee feature.

It was also identified that the contract assumes that its token balance does not change over time without any token transfers, which may not be true if the tokens being transferred were deflationary/rebasing tokens. For example, the supply of AMPL (Ampleforth) tokens automatically increases or decreases every 24 hours to maintain the AMPL target price.

In these cases, the contracts may not have the full token amounts, and the associated functions may revert.

Code Location:

Several functions assume that the ERC20 contracts transfer the full amount of tokens:

Listing 7: contracts/fundlock/Fundlock.sol (Lines 67-68)

```
            IERC20(token).safeTransferFrom(msg.sender, address(this),
 ↳ amount);
            _balances[client][token] = _balances[client][token] +
 ↳ amount;
            emit Deposit(client, token, amount);
        }
```

Listing 8: contracts/fundlock/Fundlock.sol (Line 136)

```
        uint256 availableBalance = IERC20(token).balanceOf(address
  ↳ (this));
        if (availableBalance < withdrawal.amount) {
            uint256 shortage = withdrawal.amount -
  ↳ availableBalance;
            IStrategy(tokenStrategy).returnFund(shortage);
        }
        IERC20(token).safeTransfer(msg.sender, withdrawal.amount);
```

Listing 9: contracts/strategies/Strategy.sol (Lines 82-84)

```
        if (expectUtilizeAmount > managingFund_) {
            uint256 _shortage = expectUtilizeAmount -
  ↳ managingFund_;
            IFundlock(fundlock).utilizeFund(supplyingAsset,
  ↳ _shortage);
            _utilize(_shortage);
            emit FundPulled(fundlock, _shortage);
        }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:M/Y:N/R:P/S:U (2.5)

Recommendation:

It is recommended to get the exact received amount of the tokens being transferred by calculating the difference of the token balance before and after the transfer and using it to update all the variables correctly.

It is also recommended that all tokens are thoroughly checked and tested before they are used to avoid tokens that are incompatible with the contracts.

Remediation Plan:

RISK ACCEPTED: The ithacaprotocol.io team made a business decision to accept the risk of this finding and not alter the contracts because they do not plan to support such tokens.

4.6 (HAL-06) MISSING TOKEN VALIDATION CAN LEAD TO READ IMPROPER DATA - INFORMATIONAL (0.8)

Description:

It was identified that the getTokenDetails() function in the TokenValidator contract does not revert if the token is not whitelisted and returns zero values. This function is used in the initialize() function of the Ledger contract. If the return values are not checked, you could initialize the Ledger contract with an invalid data.

Code Location:

The getTokenDetails() function returns zero values if the token is not whitelisted.

Listing 10: contracts/validator/TokenValidator.sol

```
    function getTokenDetails(
        address token
    ) external view returns (uint8 precision, uint8
 decimalPrecisionDiff) {
        TokenDetails memory tokenPrecision = _whitelist[token];
        precision = tokenPrecision.precision;
        decimalPrecisionDiff = tokenPrecision.decimalPrecisionDiff
 ;
    }
```

The values returned from the getTokenDetails() function are not validated in the initialize() function:

Listing 11: contracts/ledger/Ledger.sol (Lines 46-51)

```
    function initialize(
        address accessController_,
        address underlyingCurrency_,
        address strikeCurrency_
    ) external initializer {
        __AccessRestricted_init_unchained(accessController_);
        underlyingCurrency = underlyingCurrency_;
        strikeCurrency = strikeCurrency_;
        registry = msg.sender;

        address tokenValidator = IRegistry(registry).
↳ tokenValidator();
        (, uint8 diffUnderlying) = ITokenValidator(tokenValidator)
            .getTokenDetails(underlyingCurrency);
        (, uint8 diffStrike) = ITokenValidator(tokenValidator).
↳ getTokenDetails(
            strikeCurrency
        );

        _underlyingMultiplier = int256(10 ** diffUnderlying);
        _strikeMultiplier = int256(10 ** diffStrike);
    }
```

Proof of Concept:

```
 TokenValidator
   Halborn Tests: TokenValidator
         Calling getTokenDetails() if token is not whitelisted:
         -   precision: 0
         -   decimalPrecisionDiff: 0
     ✓ #getTokenDetails() returns 0 values if token is not whitelisted (42ms)
```

BVSS:

AO:A/AC:L/AX:H/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (0.8)

Recommendation:

It is recommended to validate the return values of the getTokenDetails() function.

Remediation Plan:

ACKNOWLEDGED: The ithacaprotocol.io team made a business decision to acknowledge this finding and not alter the contracts. New Ledger contracts will only be deployed by the Registry, which adds the strike and underlying tokens to the whitelist of the TokenValidator before deployment.

4.7 (HAL-07) USING GAS INEFFICIENT PARAMETER STORAGE - INFORMATIONAL (0.0)

Description:

It was identified that the addTokensToWhitelist() function in the TokenValidator contract uses a memory parameter storage instead of calldata. If a reference type function parameter is read-only, it is cheaper in gas to use calldata instead of memory. calldata is a non-modifiable, non-persistent area where function arguments are stored, and behaves mostly like memory.

Code Location:

Listing 12: contracts/validator/TokenValidator.sol

```
    function addTokensToWhitelist(
        AddTokenToWhitelistParams[] memory tokens
    ) external onlyRole(ADMIN_ROLE) {
        uint256 tokensLength = tokens.length;
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U - 0.0 - None (0.0)

Recommendation:

Consider using calldata instead of memory.

Remediation Plan:

SOLVED: The ithacaprotocol.io team solved the issue in commit 9d54d87 by using calldata instead of memory.

AUTOMATED TESTING

5.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.
The security team assessed all findings identified by the Slither software, however, findings with severity Information and Optimization are not included in the below results for the sake of report readability.

Results:

contracts/strategies/Strategy.sol

Slither results for Strategy.sol	
Finding	**Impact**
Reentrancy in Strategy.adjustFund() (contracts/strategies/Strategy.sol#77-87): External calls: - IFundlock(fundlock).utilizeFund(supplyingAsset,_shortage) (contracts/strategies/Strategy.sol#83) Event emitted after the call(s): - FundPulled(fundlock,_shortage) (contracts/strategies/Strategy.sol#85)	Low
Reentrancy in Strategy.returnFund(uint256) (contracts/strategies/Strategy.sol#89-94): External calls: - IERC20(supplyingAsset).safeTransfer(fundlock,amountRecieved) (contracts/strategies/Strategy.sol#92) Event emitted after the call(s): - FundReturned(fundlock,amountRecieved) (contracts/strategies/Strategy.sol#93)	Low
End of table for Strategy.sol	

contracts/strategies/AaveV3Strategy.sol

Slither results for AaveV3Strategy.sol	
Finding	**Impact**
AaveV3Strategy.userYield(address) (contracts/strategies/AaveV3Strategy.sol#62-79) uses a dangerous strict equality: - totalValueAll_ == 0 (contracts/strategies/AaveV3Strategy.sol#68)	Medium
AaveV3Strategy._utilize(uint256) (contracts/strategies/AaveV3Strategy.sol#90-95) ignores return value by IERC20(supplyingAsset).approve(aavePool,_amount) (contracts/strategies/AaveV3Strategy.sol#93)	Medium
AaveV3Strategy.initialize(address,address,address,address,uint256,address).aavePool_ (contracts/strategies/AaveV3Strategy.sol#24) lacks a zero-check on : - aavePool = aavePool_ (contracts/strategies/AaveV3Strategy.sol#33)	Low
End of table for AaveV3Strategy.sol	

contracts/registry/Registry.sol

Slither results for Registry.sol	
Finding	**Impact**
Reentrancy in Registry.deployLedger(address,address,uint8,uint8) (contracts/registry/Registry.sol#59-101): External calls: - ITokenValidator(tokenValidator).addTokensToWhitelist(params) (contracts/registry/Registry.sol#84) - beaconProxy = new LedgerBeaconProxy(ledgerBeacon,abi.encodeWithSignature(initialize(address,address,address),_getAccessController(),underlying,strike)) (contracts/registry/Registry.sol#86-94) State variables written after the call(s): - _ledgers[underlying][strike] = deployedLedger (contracts/registry/Registry.sol#97) Registry._ledgers (contracts/registry/Registry.sol#22-23) can be used in cross function reentrancies: - Registry.deployLedger(address,address,uint8,uint8) (contracts/registry/Registry.sol#59-101)	Medium
Reentrancy in Registry.deployLedger(address,address,uint8,uint8) (contracts/registry/Registry.sol#59-101): External calls: - ITokenValidator(tokenValidator).addTokensToWhitelist(params) (contracts/registry/Registry.sol#84) - beaconProxy = new LedgerBeaconProxy(ledgerBeacon,abi.encodeWithSignature(initialize(address,address,address),_getAccessController(),underlying,strike)) (contracts/registry/Registry.sol#86-94) State variables written after the call(s): - _deployedLedgers[deployedLedger] = true (contracts/registry/Registry.sol#98)	Low
Reentrancy in Registry.deployLedger(address,address,uint8,uint8) (contracts/registry/Registry.sol#59-101): External calls: - ITokenValidator(tokenValidator).addTokensToWhitelist(params) (contracts/registry/Registry.sol#84) - beaconProxy = new LedgerBeaconProxy(ledgerBeacon,abi.encodeWithSignature(initialize(address,address,address),_getAccessController(),underlying,strike)) (contracts/registry/Registry.sol#86-94) Event emitted after the call(s): - LedgerDeployed(deployedLedger) (contracts/registry/Registry.sol#100)	Low
End of table for Registry.sol	

contracts/ledger/Ledger.sol

Slither results for Ledger.sol	
Finding	**Impact**
Ledger._initializeData(ILedger.FundMovementParam[],uint256).i (contracts/ledger/Ledger.sol#171) is a local variable never initialized	Medium
Ledger._initializeData(ILedger.FundMovementParam[],uint256).ctr (contracts/ledger/Ledger.sol#172) is a local variable never initialized	Medium
Ledger._validateAndCountAmounts(ILedger.FundMovementParam[]).i (contracts/ledger/Ledger.sol#102) is a local variable never initialized	Medium
Ledger._processPositionUpdates(ILedger.PositionParam[]).i (contracts/ledger/Ledger.sol#122) is a local variable never initialized	Medium
Ledger.initialize(address,address,address).strikeCurrency_- (contracts/ledger/Ledger.sol#39) lacks a zero-check on : - strikeCurrency = strikeCurrency_ (contracts/ledger/Ledger.sol#43)	Low
Ledger.initialize(address,address,address).underlyingCurrency _ (contracts/ledger/Ledger.sol#38) lacks a zero-check on : - underlyingCurrency = underlyingCurrency_ (contracts/ledger/Ledger.sol#42)	Low
Reentrancy in Ledger.updateFundMovements(ILedger.FundMovementParam[ ],uint64) (contracts/ledger/Ledger.sol#81-89): External calls: - _processFundMovement(fundMovements,backendId,transferCount) (contracts/ledger/Ledger.sol#87) - IFundlock(fundlock).updateBalances(clients,tokens,amounts,backend Id) (contracts/ledger/Ledger.sol#143) Event emitted after the call(s): - FundMovementsUpdated(backendId) (contracts/ledger/Ledger.sol#88)	Low
End of table for Ledger.sol	

contracts/fundlock/Fundlock.sol

Slither results for Fundlock.sol	
Finding	**Impact**
Reentrancy in Fundlock.release(address,uint8) (contracts/fundlock/Fundlock.sol#112-140): External calls: - IStrategy(tokenStrategy).distributeYield(msg.sender) (contracts/fundlock/Fundlock.sol#125) State variables written after the call(s): - _withdrawals[msg.sender][token][index] = Withdrawal(0,0) (contracts/fundlock/Fundlock.sol#128-131)Fundlock._withdrawals (contracts/fundlock/Fundlock.sol#32-33) can be used in cross function reentrancies: - Fundlock._fundFromWithdrawal(address,address,uint256) (contracts/fundlock/Fundlock.sol#288-326) - Fundlock.fundsToWithdraw(address,address,uint8) (contracts/fundlock/Fundlock.sol#184-192) - Fundlock.fundsToWithdrawTotal(address,address) (contracts/fundlock/Fundlock.sol#201-217) - Fundlock.release(address,uint8) (contracts/fundlock/Fundlock.sol#112-140) - Fundlock.withdraw(address,uint256) (contracts/fundlock/Fundlock.sol#79-104)	Medium

Finding	Impact
Reentrancy in Fundlock.withdraw(address,uint256) (contracts/fundlock/Fundlock.sol#79-104): External calls: - IStrategy(tokenStrategy).distributeYield(msg.sender) (contracts/fundlock/Fundlock.sol#92) State variables written after the call(s): - _balances[msg.sender][token] = balance - amount (contracts/fundlock/Fundlock.sol#95) Fundlock._balances (contracts/fundlock/Fundlock.sol#28-29) can be used in cross function reentrancies: - Fundlock._updateBalance(address,address,int256) (contracts/fundlock/Fundlock.sol#328-350) - Fundlock.balanceSheet(address,address) (contracts/fundlock/Fundlock.sol#169-174) - Fundlock.deposit(address,address,uint256) (contracts/fundlock/Fundlock.sol#57-70) - Fundlock.distributeYield(address,address,uint256) (contracts/fundlock/Fundlock.sol#226-236) - Fundlock.withdraw(address,uint256) (contracts/fundlock/Fundlock.sol#79-104)	Medium

Finding	Impact
Reentrancy in Fundlock.setTokenStrategy(address,address) (contracts/fundlock/Fundlock.sol#254-265): External calls: - IStrategy(tokenStrategy).returnFund(type()(uint256).max) (contracts/fundlock/Fundlock.sol#261) State variables written after the call(s): - tokenStrategies[token] = strategy (contracts/fundlock/Fundlock.sol#263)Fundlock.tokenStrategies (contracts/fundlock/Fundlock.sol#34) can be used in cross function reentrancies: - Fundlock.deposit(address,address,uint256) (contracts/fundlock/Fundlock.sol#57-70) - Fundlock.distributeYield(address,address,uint256) (contracts/fundlock/Fundlock.sol#226-236) - Fundlock.release(address,uint8) (contracts/fundlock/Fundlock.sol#112-140) - Fundlock.setTokenStrategy(address,address) (contracts/fundlock/Fundlock.sol#254-265) - Fundlock.tokenStrategies (contracts/fundlock/Fundlock.sol#34) - Fundlock.utilizeFund(address,uint256) (contracts/fundlock/Fundlock.sol#219-224) - Fundlock.withdraw(address,uint256) (contracts/fundlock/Fundlock.sol#79-104)	Medium
Fundlock._findEmptySlot(address,address).i (contracts/fundlock/Fundlock.sol#272) is a local variable never initialized	Medium
Fundlock.fundsToWithdrawTotal(address,address).withdrawnSum (contracts/fundlock/Fundlock.sol#205) is a local variable never initialized	Medium
Fundlock._fundFromWithdrawal(address,address,uint256).index (contracts/fundlock/Fundlock.sol#294) is a local variable never initialized	Medium
Fundlock._fundFromWithdrawal(address,address,uint256).fundedSum (contracts/fundlock/Fundlock.sol#293) is a local variable never initialized	Medium
Fundlock.updateBalances(address[],address[],int256[],uint64).i (contracts/fundlock/Fundlock.sol#160) is a local variable never initialized	Medium

Finding	Impact
Fundlock.fundsToWithdrawTotal(address,address).k (contracts/fundlock/Fundlock.sol#206) is a local variable never initialized	Medium
Reentrancy in Fundlock.withdraw(address,uint256) (contracts/fundlock/Fundlock.sol#79-104): External calls: - IStrategy(tokenStrategy).distributeYield(msg.sender) (contracts/fundlock/Fundlock.sol#92) State variables written after the call(s): - _fillSlot(msg.sender,token,slot) (contracts/fundlock/Fundlock.sol#97) - _withdrawalSlots[user][token] |= uint8(1 << slot) (contracts/fundlock/Fundlock.sol#281) - _withdrawals[msg.sender][token][slot] = Withdrawal(amount,uint32(block.timestamp)) (contracts/fundlock/Fundlock.sol#98-101)	Low
Reentrancy in Fundlock.deposit(address,address,uint256) (contracts/fundlock/Fundlock.sol#57-70): External calls: - IStrategy(tokenStrategy).distributeYield(client) (contracts/fundlock/Fundlock.sol#65) - IERC20(token).safeTransferFrom(msg.sender,address(this),amount) (contracts/fundlock/Fundlock.sol#67) State variables written after the call(s): - _balances[client][token] = _balances[client][token] + amount (contracts/fundlock/Fundlock.sol#68)	Low
Reentrancy in Fundlock.release(address,uint8) (contracts/fundlock/Fundlock.sol#112-140): External calls: - IStrategy(tokenStrategy).distributeYield(msg.sender) (contracts/fundlock/Fundlock.sol#125) State variables written after the call(s): - _emptySlot(msg.sender,token,index) (contracts/fundlock/Fundlock.sol#132) - _withdrawalSlots[user][token] &= uint8(1 << slot) (contracts/fundlock/Fundlock.sol#285)	Low

Finding	Impact
Reentrancy in Fundlock.withdraw(address,uint256) (contracts/fundlock/Fundlock.sol#79-104): External calls: - IStrategy(tokenStrategy).distributeYield(msg.sender) (contracts/fundlock/Fundlock.sol#92) Event emitted after the call(s): - Withdraw(msg.sender,token,amount,slot) (contracts/fundlock/Fundlock.sol#103)	Low
Reentrancy in Fundlock.release(address,uint8) (contracts/fundlock/Fundlock.sol#112-140): External calls: - IStrategy(tokenStrategy).distributeYield(msg.sender) (contracts/fundlock/Fundlock.sol#125) - IStrategy(tokenStrategy).returnFund(shortage) (contracts/fundlock/Fundlock.sol#136) - IERC20(token).safeTransfer(msg.sender,withdrawal.amount) (contracts/fundlock/Fundlock.sol#138) Event emitted after the call(s): - Release(msg.sender,token,withdrawal.amount,index) (contracts/fundlock/Fundlock.sol#139)	Low
Reentrancy in Fundlock.deposit(address,address,uint256) (contracts/fundlock/Fundlock.sol#57-70): External calls: - IStrategy(tokenStrategy).distributeYield(client) (contracts/fundlock/Fundlock.sol#65) - IERC20(token).safeTransferFrom(msg.sender,address(this),amount) (contracts/fundlock/Fundlock.sol#67) Event emitted after the call(s): - Deposit(client,token,amount) (contracts/fundlock/Fundlock.sol#69)	Low
Reentrancy in Fundlock.setTokenStrategy(address,address) (contracts/fundlock/Fundlock.sol#254-265): External calls: - IStrategy(tokenStrategy).returnFund(type()(uint256).max) (contracts/fundlock/Fundlock.sol#261) Event emitted after the call(s): - TokenStrategySet(token,strategy) (contracts/fundlock/Fundlock.sol#264)	Low

Finding	Impact
Fundlock._fundFromWithdrawal(address,address,uint256) (contracts/fundlock/Fundlock.sol#288-326) uses timestamp for comparisons Dangerous comparisons: - withdrawal.timestamp + tradeLock > block.timestamp (contracts/fundlock/Fundlock.sol#296)	Low
Fundlock.release(address,uint8) (contracts/fundlock/Fundlock.sol#112-140) uses timestamp for comparisons Dangerous comparisons: - uint32(block.timestamp) <= withdrawal.timestamp + releaseLock (contracts/fundlock/Fundlock.sol#116)	Low
Fundlock.fundsToWithdrawTotal(address,address) (contracts/fundlock/Fundlock.sol#201-217) uses timestamp for comparisons Dangerous comparisons: - withdrawal.timestamp + tradeLock > block.timestamp (contracts/fundlock/Fundlock.sol#209-210)	Low
End of table for Fundlock.sol	

contracts/validator/TokenValidator.sol

Slither results for TokenValidator.sol	
Finding	**Impact**
TokenValidator.addTokensToWhitelist(ITokenValidator.AddTokenToWhitelistParams[]).i (contracts/validator/TokenValidator.sol#70) is a local variable never initialized	Medium
TokenValidator.addTokensToWhitelist(ITokenValidator.AddTokenToWhitelistParams[]) (contracts/validator/TokenValidator.sol#65-100) has external calls inside a loop: decimals = IERC20Metadata(info.token).decimals() (contracts/validator/TokenValidator.sol#81)	Low
End of table for TokenValidator.sol	

contracts/ledger/proxy/LedgerBeaconProxy.sol

Slither did not identify any vulnerabilities in the contract.

contracts/ledger/proxy/LedgerBeacon.sol

Slither did not identify any vulnerabilities in the contract.

contracts/access/AccessController.sol

Slither did not identify any vulnerabilities in the contract.

contracts/access/AccessRestricted.sol

Slither did not identify any vulnerabilities in the contract.

contracts/access/Roles.sol

Slither did not identify any vulnerabilities in the contract.

Results summary:

The findings obtained as a result of the Slither scan were reviewed. The vulnerabilities identified by Slither were determined to be false-positives.

THANK YOU FOR CHOOSING

// HALBORN