HALBORN

CreditSwap - Core Contracts

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **January 9th, 2024 - February 9th, 2024**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	02/06/2024
0.2	Document Updates	02/07/2024
0.3	Draft Review	02/09/2024
1.0	Remediation Plan	02/22/2024
1.1	Remediation Plan Review	02/26/2024

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

CreditSwap engaged Halborn to conduct a security assessment on their smart contract beginning on January 9th, 2024 and ending on February 9th, 2024. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided four weeks for the engagement and assigned a full-time security engineer to verify the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified some security risks that were mostly addressed by the CreditSwap team.

1.3 SCOPE

1. **IN-SCOPE :**

   - 94e776796b02828a254eb4674b6a030ef6ac8420

In-scope Contracts :

- ./creditUsd/CreditUSD.sol
- ./creditUsd/CreditUSDMinter.sol
- ./util/DefaultCreditorAutomation.sol
- ./nft/DebtorNFT.sol
- ./nft/DebtorStaking.sol
- ./nft/CreditorNFT.sol
- ./nft/StakingPool.sol
- ./nft/NFTBase.sol
- ./oracle/BaseOracleUSD.sol
- ./oracle/AggregatedChainlinkOracle.sol
- ./oracle/WBTCOracle.sol
- ./LoanVault.sol
- ./ProtocolController.sol
- ./governance/token/VeCreditSwapToken.sol
- ./governance/token/CSProtocolToken.sol
- ./governance/token/lib/TokenWrapper.sol
- ./governance/token/lib/TokenCheckpointer.sol
- ./governance/RewardAuctionHouse.sol
- ./governance/RewardDistributor.sol
- ./governance/CSGovernor.sol
- ./governance/CSTimelock.sol
- ./lib/TimestampLib.sol
- ./lib/Beacon.sol
- ./lib/BeaconProxy.sol
- ./lib/AccessControl.sol

- ./lib/RenderNFT.sol
- ./lib/Staking.sol
- ./CreditSwapMarket.sol

REMEDIATION COMMIT IDs :

- b902b8d
- ba38416
- 2a344e1
- 59ca73e
- 4aa6f24
- d1a2e0e

1.4 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 bridge code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of 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)

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) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 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) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 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

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	0	1	3	7

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) INADEQUATE PROPOSAL THRESHOLD SETTING IN CREDITSWAP GOVERNOR CONTRACT	Critical (10)	SOLVED - 02/06/2024
(HAL-02) NON-STANDARD TOKENS CAN LEAD TO SILENT FAILURES	Medium (6.2)	SOLVED - 02/18/2024
(HAL-03) INCOMPATIBILITY WITH TRANSFER-ON-FEE OR DEFLATIONARY TOKENS	Low (2.5)	SOLVED - 02/18/2024
(HAL-04) MISSING ERC721HOLDERUPGRADEABLE ERC721HOLDER INIT() CALL ON THE CREDITUSDMINTER CONTRACT	Low (2.5)	SOLVED - 02/18/2024
(HAL-05) ROBUST ALLOWANCE	Low (2.5)	SOLVED - 02/18/2024
(HAL-06) PAUSE FUNCTIONALITY IN LIQUIDATION PROCESS TO HANDLE UNEXPECTED SITUATIONS	Informational (0.0)	ACKNOWLEDGED
(HAL-07) CACHE ARRAY LENGTHS	Informational (0.0)	SOLVED - 02/18/2024
(HAL-08) USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS TO SAVE GAS	Informational (0.0)	SOLVED - 02/18/2024
(HAL-09) NO NEED TO INITIALIZE VARIABLES WITH DEFAULT VALUES	Informational (0.0)	SOLVED - 02/18/2024
(HAL-10) REVERT STRING SIZE OPTIMIZATION	Informational (0.0)	SOLVED - 02/18/2024
(HAL-11) LATESTROUNDDATA CALL MAY RETURN STALE RESULTS	Informational (0.0)	ACKNOWLEDGED
(HAL-12) MISSING IMPLEMENTATION OF TOKENURI FUNCTION IN NFTBASE CONTRACT	Informational (0.0)	ACKNOWLEDGED

FINDINGS & TECH DETAILS

4.1 (HAL-01) INADEQUATE PROPOSAL THRESHOLD SETTING IN CREDITSWAP GOVERNOR CONTRACT - CRITICAL(10)

Description:

The CSGovernor contract, which inherits from Governor, GovernorVotes, GovernorVotesQuorumFraction, and GovernorTimelockControl, includes a critical configuration issue in its constructor, specifically regarding the proposalThreshold function. The proposalThreshold function, which determines the minimum number of votes required to propose governance actions, is currently set to return 0. This configuration effectively means there is no minimum vote threshold for proposing governance actions. Determine and set a meaningful proposalThreshold value that balances accessibility with the need to prevent spam and low-quality proposals. This threshold should reflect the governance system's size and the token distribution.

Code Location:

CSGovernor.sol#L32

Listing 1

```
contract CSGovernor is Governor, GovernorVotes,
↳ GovernorVotesQuorumFraction, GovernorTimelockControl {
    mapping(uint256 proposalId => mapping(address account =>
↳ uint256 weight)) public accountWeights;
    mapping(uint256 proposalId => mapping(uint8 support => uint256
↳  totalWeight)) public proposalWeights;

    constructor(IVotes _token, TimelockController _timelock)
        Governor("CreditSwapGovernor")
        GovernorVotes(_token)
        GovernorVotesQuorumFraction(4)
        GovernorTimelockControl(_timelock)
    {}

```

```
12      function votingDelay() public pure override returns (uint256)
↳ {
13          return 1 days;
14      }
15
16      function votingPeriod() public pure override returns (uint256)
↳  {
17          return 7 days;
18      }
19
20      function proposalThreshold() public pure override returns (
↳ uint256) {
21          return 0; // TBD
22      }
23  }
```

Proof Of Concept:

Listing 2

```
1     function testSetVotingPeriodDuringActiveProposal() public {
2       //1. initiate a proposal
3       _propose(0);
4       //2. change fullWeightDuration while the proposal is still
↳ active
5       assertEq(governor.votingPeriod(), initParams.votingPeriod);
6       vm.prank(initParams.owner);
7       governor.relay(
8         address(governor),
9         0,
10        abi.encodeWithSelector(governor.setVotingPeriod.selector,
↳ 121_212)
11      );
12      assertEq(governor.proposalThreshold(), 0);
13    }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:C/A:C/D:N/Y:N/R:N/S:U (10)

Recommendation:

To address this issue, it’s essential to determine and set a meaningful `proposalThreshold` value.

Remediation Plan:

SOLVED: The `CreditSwap team` solved the issue by adjusting proposal threshold according to total supply.

Commit ID : b902b8d

4.2 (HAL-02) NON-STANDARD TOKENS CAN LEAD TO SILENT FAILURES - MEDIUM (6.2)

Description:

Currently, the contract does not adequately handle atypical ERC20 tokens on the staking. According to the ERC20 specification, tokens should return “false” when a transfer fails, but it does not guarantee that the function will revert. This discrepancy could potentially lead to silent failures, making it difficult to detect issues when they occur.

For instance, from the below, It can be seen that LDO does not revert If the user balance is not enough for the transfer.

Listing 3

```
...
var previousBalanceFrom = balanceOfAt(_from, block.number);
if (previousBalanceFrom < _amount) {
    return false;
}
...
```

Code Location:

VeCreditSwapToken.sol#L124-L125

Listing 4

```
    /// @dev Internal method used for public staking
    /// @param amount amount to lockup in the stake
    /// @param duration in seconds for the stake
    /// @param to address to receive ownership of the stake
    function _stake(uint256 amount, uint256 duration, address to)
↳ internal {
        require(to != address(0), "Staking: To the zero address");
        require(amount <= type(uint128).max, "Staking: Too much");
```

```
8         require(amount > 0, "Staking: Not enough");
9
10        // duration checked inside previewPoints
11        (uint256 points, uint256 end) = previewPoints(amount,
↳ duration);
12        require(points + totalSupply() <= type(uint192).max, "
↳ Staking: Max points exceeded");
13        _updateRewards(to);
14        lockups[to].push(
15            Lockup({
16                amount: uint128(amount), // max checked in require
↳  above
17                end: uint128(end),
18                points: points
19            })
20        );
21        _mint(to, points);
22        underlyingToken.transferFrom(msg.sender, address(this),
↳ amount); // Important that it's sender
23
24        if (!hasDelegationSet[to] && delegates(to) == address(0))
↳ {
25            // Delegate voting power to the receiver, if
↳ unregistered
26            _delegate(to, to);
27        }
28
29        emit Stake(to, lockups[to].length - 1, amount, end, points
↳ );
30    }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:M/Y:N/R:N/S:C (6.2)

Recommendation:

Consider checking pre- / post-balance of contract during the transfer. To ensure proper handling of atypical tokens and compliance with ERC20 standards, it is advisable to incorporate OpenZeppelin's SafeTransferLib. This library is specifically designed to handle edge cases and provide a

consistent behavior for token transfers.

Remediation Plan:

SOLVED: The CreditSwap team solved the issue by using safe transfer.

Commit ID : ba38416

4.3 (HAL-03) 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:

DefaultCreditorAutomation.sol#L82-L86

Listing 5

```
    function _deposit(address asset, uint256 amount) internal {
        IERC20(asset).safeTransferFrom(msg.sender, address(this),
↳ amount);
        emit Deposit(asset, msg.sender, amount);
    }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/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.

Remediation Plan:

SOLVED: The CreditSwap team solved the issue by not white-listing fee-on-transfer tokens.

4.4 (HAL-04) MISSING ERC721HOLDERUPGRADEABLE ERC721HOLDER INIT() CALL ON THE CREDITUSDMINTER CONTRACT - LOW (2.5)

Description:

The contracts, which are designed to be upgradeable, miss proper initialization functions. The initial setup for a ERC721Holder upgradeable contract is crucial to ensure that the contract can be safely upgraded in the future without losing state or introducing vulnerabilities.

Code Location:

CreditUSDMinter.sol#L20

Listing 6

```
contract CreditUSDMinter is ICreditUSDMinter,
↳ AccessControlExternal, ERC721HolderUpgradeable {
    using SafeERC20 for ICreditUSD;

    uint16 constant DENOMINATOR = 10000;
}
```

BVSS:

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

Recommendation:

It is crucial to include an initialization procedure for the ERC721HolderUpgradeable within the CreditUSDMinter contract. This can be achieved by calling the init() function or equivalent initializer in the contract's constructor, or a dedicated initialization function for

upgradeable contracts.

Remediation Plan:

SOLVED: The CreditSwap team solved the issue by using safe transfer.

Commit ID : 2a344e1

4.5 (HAL-05) ROBUST ALLOWANCE - LOW (2.5)

Description:

Non-standard tokens like USDT will revert the transaction when a contract or a user tries to approve an allowance when the spender allowance is already set to a non-zero value. For that reason, the previous allowance should be decreased before increasing allowance in the related function.

Code Location:

Listing 7

```
File: nft/CreditorNFT.sol

181:          IERC20(debt).approve(vault, debtAmount);

```

BVSS:

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

Recommendation:

Consider approving with zero first through safeApprove function.

Remediation Plan:

SOLVED: The CreditSwap team solved the issue by using force approve.

Commit ID : 79d8d0b

4.6 (HAL-06) PAUSE FUNCTIONALITY IN LIQUIDATION PROCESS TO HANDLE UNEXPECTED SITUATIONS - INFORMATIONAL (0.0)

Description:

The current implementation of the liquidate function in a loan contract lacks the capability to be paused in response to unexpected situations. This function is critical as it handles the liquidation of collateral in the event of a loan default or breach of the liquidation threshold. Integrate a pause mechanism, typically through a pause function controlled by an admin role or a decentralized governance process. This function should be able to halt the liquidation process immediately when activated.

Code Location:

LoanVault.sol#L172

Listing 8

```
    function liquidate(address to) external onlyStatus(LoanStatus.
↳ ACTIVE) {
        withdrawableInterest = _claimInterest();
        if (healthFactor() > liquidationThreshold) revert
↳ LoanHealthy();
        status = LoanStatus.LIQUIDATED;
        uint256 currentDebt = IERC20(debt).balanceOf(address(this)
↳ );
        if (currentDebt < debtAmount) revert InvalidDebt();

        // unstake token if staked
        debtorNFT.forceUnstaking(id);

        uint256 collateralBalance = IERC20(collateral).balanceOf(
↳ address(this));
        if (collateralBalance < withdrawableInterest) {
            withdrawableInterest = collateralBalance;
```

```
14          }
15          uint256 remainingCollateral = collateralBalance -
↳ withdrawableInterest;
16          uint256 protocolFee = (remainingCollateral * _fees.
↳ protocolLiquidationFee) / FEE_DENOMINATOR;
17          uint256 creditorFee = (remainingCollateral *
↳ liquidationFee) / FEE_DENOMINATOR;
18          remainingCollateral -= protocolFee + creditorFee;
19
20          withdrawableDebt = currentDebt;
21          uint256 interestWithFee = withdrawableInterest +
↳ creditorFee;
22          // Add Creditor Fee to his withdrawable Interest
23          withdrawableInterest = (interestWithFee <
↳ collateralBalance) ? interestWithFee : collateralBalance;
24
25          protocolFee = _takeFee(collateral, protocolFee);
26          if (remainingCollateral != 0) IERC20(collateral).
↳ safeTransfer(to, remainingCollateral);
27
28          _automatedFundsWithdrawal();
29
30          emit LoanLiquidated(currentDebt, remainingCollateral);
31      }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider adding pause mechanism in the function.

Remediation Plan:

ACKNOWLEDGED: The CreditSwap team acknowledged the issue. The Team claims that there is no need for pause functionality for now.

4.7 (HAL-07) CACHE ARRAY LENGTHS - INFORMATIONAL (0.0)

Description:

In a for loop, the length of an array can be put in a temporary variable to save some gas. This has been done already in several other locations in the code.

In the above case, the solidity compiler will always read the length of the array during each iteration. That is,

- if it is a storage array, this is an extra sload operation (100 additional extra gas (EIP-2929) for each iteration except for the first),
- if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first),
- if it is a calldata array, this is an extra calldataload operation (3 additional gas for each iteration except for the first)

Code Location:

Listing 9

```
File: CreditSwapMarket.sol

53:         for (uint256 i = 0; i < allowedTokens_.length; i++) {

```

Listing 10

```
File: nft/CreditorNFT.sol

150:         for (uint256 i = 0; i < collateral.length; ++i) {

```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

In a for loop, store the length of an array in a temporary variable.

Remediation Plan:

SOLVED: The CreditSwap team solved the issue by caching array length.

Commit ID : 59ca73e

4.8 (HAL-08) USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS TO SAVE GAS - INFORMATIONAL (0.0)

Description:

Custom errors are available from solidity version 0.8.4. Custom errors save ~50 gas each time they're hit by avoiding having to allocate and store the revert string. Not defining the strings also saves deployment gas.

Code Location:

Listing 11

```
File: governance/VeCreditSwapToken.sol

62:         revert("Transfers disabled");

66:         revert("Transfers disabled");

108:         require(to != address(0), "Staking: To the zero
↳ address");

109:         require(amount <= type(uint128).max, "Staking: Too
↳ much");

110:         require(amount > 0, "Staking: Not enough");

114:         require(points + totalSupply() <= type(uint192).max,
↳ "Staking: Max points exceeded");

141:         require(block.timestamp >= end, "Staking: End of
↳ lockup not reached");

142:         require(end != 0, "Staking: Already unstaked this
↳ lockup");

```

```
171:          require(newEnd > oldEnd, "Staking: New lockup must be longer");

192:          require(duration >= minStakeDuration, "Staking: Too short");

193:          require(duration <= 1461 days, "Staking: Too long");
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider replacing all revert strings with custom errors.

Remediation Plan:

SOLVED: The CreditSwap team solved the issue by using custom errors.

Commit ID : 4aa6f24

4.9 (HAL-09) NO NEED TO INITIALIZE VARIABLES WITH DEFAULT VALUES - INFORMATIONAL (0.0)

Description:

Initialization to 0 or false is not necessary, as these are the default values in Solidity.

Code Location:

Listing 12

```
File: CreditSwapMarket.sol

53:         for (uint256 i = 0; i < allowedTokens_.length; i++) {

```

Listing 13

```
File: LoanVault.sol

262:         uint256 interest = 0;

```

Listing 14

```
File: nft/CreditorNFT.sol

150:         for (uint256 i = 0; i < collateral.length; ++i) {

```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Remove the initialization values of 0 or false.

Remediation Plan:

SOLVED: The CreditSwap team solved the issue by removing default initialization.

Commit ID : d1a2e0e

4.10 (HAL-10) REVERT STRING SIZE OPTIMIZATION - INFORMATIONAL (0.0)

Description:

Shortening revert strings to fit in 32 bytes will decrease deploy time gas and will decrease runtime gas when the revert condition has been met.

Code Location:

Listing 15

```
File: governance/VeCreditSwapToken.sol

141:          require(block.timestamp >= end, "Staking: End of
↳ lockup not reached");

142:          require(end != 0, "Staking: Already unstaked this
↳ lockup");

171:          require(newEnd > oldEnd, "Staking: New lockup must be
↳  longer");
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Shorten the revert strings to fit in 32 bytes. Alternatively, the code could be modified to use custom errors, introduced in Solidity 0.8.4.

Remediation Plan:

SOLVED: The CreditSwap team solved the issue by using custom errors.

Commit ID : 4aa6f24

4.11 (HAL-11) LATESTROUNDDATA CALL MAY RETURN STALE RESULTS - INFORMATIONAL (0.0)

Description:

The return values of the latestRoundData() call are:

- roundId: The round ID. Oracles provide periodic data updates to the aggregators. Data feeds are updated in rounds. Rounds are identified by their roundId, which increases with each new round. This increase may not be monotonic. Knowing the roundId of a previous round allows contracts to consume historical data.
- answer: The data that this specific feed provides, in this case, the price of an asset.
- startedAt: Timestamp of when the round started.
- updatedAt: Timestamp of when the round was updated.
- answeredInRound: The round ID of the round in which the answer was computed.

In the current implementation, There is no check for stale prices.

Code Location:

AggregatedChainlinkOracle.sol#L24

Listing 16

```
    function latestPrice() public view virtual override returns (
↳ uint256) {
        (, int256 answer, , , ) = _feed.latestRoundData();
        assert(answer > 0);
        uint256 adjustedDecimalsAnswer = uint256(answer) * 10 **
↳ (18 - _decimals);
        return adjustedDecimalsAnswer;
    }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

`latestPrice` calls out to a Chainlink oracle receiving the `latestRoundData()`. If there is a problem with Chainlink starting a new round and finding consensus on the new value for the oracle (e.g. Chainlink nodes abandon the oracle, chain congestion, vulnerability/attacks on the Chainlink system) consumers of this contract may continue using outdated stale or incorrect data (if oracles are unable to submit no new round is started).

Remediation Plan:

ACKNOWLEDGED: The CreditSwap team acknowledged the issue.

4.12 (HAL-12) MISSING IMPLEMENTATION OF TOKENURI FUNCTION IN NFTBASE CONTRACT - INFORMATIONAL (0.0)

Description:

The NFTBase contract, serving as the foundational element for debtor and creditor NFTs within our ecosystem, currently has the tokenURI function implementation commented out. This function is crucial for ERC721 compliance as it provides a unique URI for each token's metadata, which is essential for rendering the NFTs on marketplaces and user interfaces. The absence of an active tokenURI function means that NFTs minted from this contract lack accessible metadata, significantly reducing their interoperability and visibility across platforms that rely on this standard method to retrieve NFT details.

Code Location:

NFTBase.sol

Listing 17

```
1 abstract contract NFTBase is INFTBase, ERC721Upgradeable {}
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Uncomment and complete the implementation of the tokenURI function to ensure it returns a valid URI for each token's metadata.

Remediation Plan:

ACKNOWLEDGED: The CreditSwap team acknowledged the issue.

RECOMMENDATIONS OVERVIEW

RECOMMENDATIONS OVERVIEW

Throughout the analysis of the CreditSwap core contracts and related functionalities, several critical and notable issues have been identified. This overview consolidates the recommendations for remediation, aiming to enhance the security, efficiency, and standard compliance of the system. Implementing these recommendations will significantly contribute to the robustness and reliability of CreditSwap's ecosystem.

- Determine and set a meaningful proposalThreshold that reflects the governance system's size and token distribution to prevent spam and ensure only well-supported proposals are considered.
- Implement re-entrancy guards to prevent potential exploits during external calls within the redeemOffer function, ensuring the integrity of offer redemption processes.
- Adapt contract interactions to accurately handle atypical ERC20 tokens, including fee-on-transfer or deflationary tokens, to avoid silent failures and ensure contract functions perform as expected.
- Modify the latestPrice function to dynamically handle feeds with more than 18 decimals, ensuring accurate price data is provided regardless of the feed's decimal configuration.
- Uncomment and complete the implementation of the tokenURI function to ensure ERC721 compliance and enhance the interoperability and usability of NFTs within the ecosystem.
- Integrate a pause mechanism in critical functions like liquidation to provide a means to halt operations in response to unexpected situations or threats.

AUTOMATED TESTING

6.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats, Slither was run on the all-scoped 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.

Slither results:

```
INFO:Detectors:
Governor._executeOperations(uint256,address[],uint256[],bytes[],bytes32) (node_modules/@openzeppelin/contracts/governance/Governor.sol#437-448) sends eth to arbitrary user
        Dangerous calls:
        - (success,returndata) = targets[i].call{value: values[i]}(calldatas[i]) (node_modules/@openzeppelin/contracts/governance/Governor.sol#445)
Governor.relay(address,uint256,bytes) (node_modules/@openzeppelin/contracts/governance/Governor.sol#658-661) sends eth to arbitrary user
        Dangerous calls:
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/governance/Governor.sol#659)
TimelockController._execute(address,uint256,bytes) (node_modules/@openzeppelin/contracts/governance/TimelockController.sol#412-415) sends eth to arbitrary user
        Dangerous calls:
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/governance/TimelockController.sol#413)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#123-282) has bitwise-xor operator ^ instead of the exponentiation operator **:
        - inverse = (3 * denominator) ^ 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#184)
PRBMath.mulDiv(uint256,uint256,uint256) (node_modules/@prb/math/contracts/PRBMath.sol#394-478) has bitwise-xor operator ^ instead of the exponentiation operator **:
        - inverse = (3 * denominator) ^ 2 (node_modules/@prb/math/contracts/PRBMath.sol#460)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-exponentiation
INFO:Detectors:
Reentrancy in CreditUSDMinter.burn(uint256) (contracts/creditUsd/CreditUSDMinter.sol#85-93):
        External calls:
        - debt = _repay(id) (contracts/creditUsd/CreditUSDMinter.sol#88)
                - returndata = address(token).functionCall(data) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#96)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#87)
                - ICreditUSD(creditUSD).safeTransferFrom(msg.sender,address(this),debt) (contracts/creditUsd/CreditUSDMinter.sol#122)
                - ICreditUSD(creditUSD).burn(vault.debt(),amount) (contracts/creditUsd/CreditUSDMinter.sol#123)
                - ICreditUSD(creditUSD).safeTransfer(feeCollector,debt - amount) (contracts/creditUsd/CreditUSDMinter.sol#124)
        External calls sending eth:
        - debt = _repay(id) (contracts/creditUsd/CreditUSDMinter.sol#88)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#87)
        State variables written after the call(s):
        - delete mintRecords[id] (contracts/creditUsd/CreditUSDMinter.sol#90)
        CreditUSDMinter.mintRecords (contracts/creditUsd/CreditUSDMinter.sol#32) can be used in cross function reentrancies:
        - CreditUSDMinter._repay(uint256) (contracts/creditUsd/CreditUSDMinter.sol#115-125)
        - CreditUSDMinter.burn(uint256) (contracts/creditUsd/CreditUSDMinter.sol#85-93)
        - CreditUSDMinter.mint(uint256,uint256) (contracts/creditUsd/CreditUSDMinter.sol#64-83)
        - CreditUSDMinter.mintRecords (contracts/creditUsd/CreditUSDMinter.sol#32)
        - CreditUSDMinter.outstandingDebt(uint256) (contracts/creditUsd/CreditUSDMinter.sol#95-103)
Reentrancy in LoanVault.liquidate(address) (contracts/LoanVault.sol#172-202):
        External calls:
        - debtorNFT.forceUnstaking(id) (contracts/LoanVault.sol#180)
        - protocolFee = _takeFee(collateral,protocolFee) (contracts/LoanVault.sol#196)
                - returndata = address(token).functionCall(data) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#96)
                - IERC20(token).safeTransfer(feeCollector,amount) (contracts/LoanVault.sol#223)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#87)
        - IERC20(collateral).safeTransfer(to,remainingCollateral) (contracts/LoanVault.sol#197)
        - _automatedFundsWithdrawal() (contracts/LoanVault.sol#199)
                - returndata = address(token).functionCall(data) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#96)
                - IERC20(collateral).safeTransfer(to,interest) (contracts/LoanVault.sol#239)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#87)
                - IERC20(debt).safeTransfer(to,debt_) (contracts/LoanVault.sol#240)
        External calls sending eth:
        - protocolFee = _takeFee(collateral,protocolFee) (contracts/LoanVault.sol#196)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#87)
        - _automatedFundsWithdrawal() (contracts/LoanVault.sol#199)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#87)
        State variables written after the call(s):
        - withdrawableInterest = collateralBalance (contracts/LoanVault.sol#184)
        LoanVault.withdrawableInterest (contracts/LoanVault.sol#40) can be used in cross function reentrancies:
        - LoanVault._withdrawFunds(address) (contracts/LoanVault.sol#234-242)
        - LoanVault.healthFactor() (contracts/LoanVault.sol#258-271)
        - LoanVault.liquidate(address) (contracts/LoanVault.sol#172-202)
        - LoanVault.repay(address) (contracts/LoanVault.sol#159-170)
        - LoanVault.withdrawableInterest (contracts/LoanVault.sol#40)
        - withdrawableInterest = interestWithFee (contracts/LoanVault.sol#194)
        LoanVault.withdrawableInterest (contracts/LoanVault.sol#40) can be used in cross function reentrancies:
        - LoanVault._withdrawFunds(address) (contracts/LoanVault.sol#234-242)
        - LoanVault.healthFactor() (contracts/LoanVault.sol#258-271)
INFO:Detectors:
TimelockController.getOperationState(bytes32) (node_modules/@openzeppelin/contracts/governance/TimelockController.sol#207-218) uses a dangerous strict equality:
        - timestamp == 0 (node_modules/@openzeppelin/contracts/governance/TimelockController.sol#209)
TimelockController.getOperationState(bytes32) (node_modules/@openzeppelin/contracts/governance/TimelockController.sol#207-218) uses a dangerous strict equality:
        - timestamp == _DONE_TIMESTAMP (node_modules/@openzeppelin/contracts/governance/TimelockController.sol#211)
RewardAuctionHouse.startAuction(address,uint256) (contracts/governance/RewardAuctionHouse.sol#54-83) uses a dangerous strict equality:
        - currentBalance == 0 (contracts/governance/RewardAuctionHouse.sol#66)
VeCreditSwapToken._currentRewardIndex() (contracts/governance/VeCreditSwapToken.sol#227-234) uses a dangerous strict equality:
        - _totalSupply == 0 (contracts/governance/VeCreditSwapToken.sol#229)
VeCreditSwapToken._currentRewardIndex() (contracts/governance/VeCreditSwapToken.sol#227-234) uses a dangerous strict equality:
        - newRewards == 0 (contracts/governance/VeCreditSwapToken.sol#232)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in VeCreditSwapToken._stake(uint256,uint256,address) (contracts/governance/VeCreditSwapToken.sol#107-132):
        External calls:
        - underlyingToken.transferFrom(msg.sender,address(this),amount) (contracts/governance/VeCreditSwapToken.sol#124)
        State variables written after the call(s):
        - _delegate(to,to) (contracts/governance/VeCreditSwapToken.sol#128)
                - _delegatee[account] = delegatee (node_modules/@openzeppelin/contracts/governance/utils/Votes.sol#170)
        Votes._delegatee (node_modules/@openzeppelin/contracts/governance/utils/Votes.sol#38) can be used in cross function reentrancies:
        - Votes._delegate(address,address) (node_modules/@openzeppelin/contracts/governance/utils/Votes.sol#168-174)
        - Votes.delegates(address) (node_modules/@openzeppelin/contracts/governance/utils/Votes.sol#127-129)
Reentrancy in StakingPool.fundRewards(uint256) (contracts/nft/StakingPool.sol#44-54):
        External calls:
        - IERC20(rewardToken).safeTransferFrom(msg.sender,address(this),amount) (contracts/nft/StakingPool.sol#46)
        State variables written after the call(s):
        - rewardDepositExpires = block.timestamp + addedRewardTime (contracts/nft/StakingPool.sol#48-50)
        StakingPool.rewardDepositExpires (contracts/nft/StakingPool.sol#17) can be used in cross function reentrancies:
        - StakingPool._currentRewardIndex() (contracts/nft/StakingPool.sol#73-84)
        - StakingPool._updateRewardIndex() (contracts/nft/StakingPool.sol#64-71)
        - StakingPool.fundRewards(uint256) (contracts/nft/StakingPool.sol#44-54)
        - StakingPool.rewardDepositExpires (contracts/nft/StakingPool.sol#17)
        - StakingPool.updateYearlyReward(uint256) (contracts/nft/StakingPool.sol#34-42)
        - rewardDepositExpires = rewardDepositExpires + addedRewardTime (contracts/nft/StakingPool.sol#48-50)
        StakingPool.rewardDepositExpires (contracts/nft/StakingPool.sol#17) can be used in cross function reentrancies:
        - StakingPool._currentRewardIndex() (contracts/nft/StakingPool.sol#73-84)
        - StakingPool._updateRewardIndex() (contracts/nft/StakingPool.sol#64-71)
        - StakingPool.fundRewards(uint256) (contracts/nft/StakingPool.sol#44-54)
        - StakingPool.rewardDepositExpires (contracts/nft/StakingPool.sol#17)
        - StakingPool.updateYearlyReward(uint256) (contracts/nft/StakingPool.sol#34-42)
Reentrancy in LoanVault.liquidate(address) (contracts/LoanVault.sol#172-202):
        External calls:
        - debtorNFT.forceUnstaking(id) (contracts/LoanVault.sol#180)
        State variables written after the call(s):
        - withdrawableInterest = collateralBalance (contracts/LoanVault.sol#184)
        LoanVault.withdrawableInterest (contracts/LoanVault.sol#40) can be used in cross function reentrancies:
        - LoanVault._withdrawFunds(address) (contracts/LoanVault.sol#234-242)
        - LoanVault.healthFactor() (contracts/LoanVault.sol#258-271)
        - LoanVault.liquidate(address) (contracts/LoanVault.sol#172-202)
        - LoanVault.repay(address) (contracts/LoanVault.sol#159-170)
        - LoanVault.withdrawableInterest (contracts/LoanVault.sol#40)
        - withdrawableInterest = interestWithFee (contracts/LoanVault.sol#194)
        LoanVault.withdrawableInterest (contracts/LoanVault.sol#40) can be used in cross function reentrancies:
        - LoanVault._withdrawFunds(address) (contracts/LoanVault.sol#234-242)
        - LoanVault.healthFactor() (contracts/LoanVault.sol#258-271)
INFO:Detectors:
        - schedule < block.timestamp (node_modules/@openzeppelin/contracts-upgradeable/access/extensions/AccessControlDefaultAdminRulesUpgradeable.sol#423)
TimelockController.getOperationState(bytes32) (node_modules/@openzeppelin/contracts/governance/TimelockController.sol#207-218) uses timestamp for comparisons
        Dangerous comparisons:
        - timestamp == 0 (node_modules/@openzeppelin/contracts/governance/TimelockController.sol#209)
        - timestamp == _DONE_TIMESTAMP (node_modules/@openzeppelin/contracts/governance/TimelockController.sol#211)
        - timestamp > block.timestamp (node_modules/@openzeppelin/contracts/governance/TimelockController.sol#213)
votes.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (node_modules/@openzeppelin/contracts/governance/utils/Votes.sol#142-161) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp > expiry (node_modules/@openzeppelin/contracts/governance/utils/Votes.sol#150)
ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (node_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol#44-67) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp > deadline (node_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol#53)
Time._getFullAt(Time.Delay,uint48) (node_modules/@openzeppelin/contracts/utils/types/Time.sol#74-77) uses timestamp for comparisons
        Dangerous comparisons:
        - effect <= timepoint (node_modules/@openzeppelin/contracts/utils/types/Time.sol#76)
CreditSwapMarket.redeemOffer(ICreditSwapMarket.Offer,bytes) (contracts/CreditSwapMarket.sol#58-101) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp < offer.startTimestamp (contracts/CreditSwapMarket.sol#59)
        - block.timestamp > offer.endTimestamp (contracts/CreditSwapMarket.sol#60)
CreditSwapMarket._handlePayment(address,address,uint256,address) (contracts/CreditSwapMarket.sol#128-143) uses timestamp for comparisons
        Dangerous comparisons:
        - msg.value < value (contracts/CreditSwapMarket.sol#132)
        - fee != 0 (contracts/CreditSwapMarket.sol#134)
        - refund != 0 (contracts/CreditSwapMarket.sol#136)
        - fee != 0 (contracts/CreditSwapMarket.sol#141)
LoanVault.remainingInterest() (contracts/LoanVault.sol#252-256) uses timestamp for comparisons
        Dangerous comparisons:
        - _interest.minDurationEnd > block.timestamp (contracts/LoanVault.sol#254)
LoanVault.healthFactor() (contracts/LoanVault.sol#258-271) uses timestamp for comparisons
        Dangerous comparisons:
        - interestData.lastUpdate != block.timestamp (contracts/LoanVault.sol#263)
CreditUSDMinter.burn(uint256) (contracts/creditUsd/CreditUSDMinter.sol#85-93) uses timestamp for comparisons
        Dangerous comparisons:
        - msg.sender != mintRecords[id].owner (contracts/creditUsd/CreditUSDMinter.sol#86)
RewardAuctionHouse.startAuction(address,uint256) (contracts/governance/RewardAuctionHouse.sol#54-83) uses timestamp for comparisons
        Dangerous comparisons:
        - currentAuctionId != 0 && auctions[asset][currentAuctionId - 1].end > block.timestamp (contracts/governance/RewardAuctionHouse.sol#57)
RewardAuctionHouse.bidOnAuction(address,uint256) (contracts/governance/RewardAuctionHouse.sol#85-108) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp >= auction.end (contracts/governance/RewardAuctionHouse.sol#92)
RewardAuctionHouse.finalizeAuction(address) (contracts/governance/RewardAuctionHouse.sol#110-125) uses timestamp for comparisons
        Dangerous comparisons:
        - auction.end > block.timestamp (contracts/governance/RewardAuctionHouse.sol#114)
RewardAuctionHouse.currentAuction(address) (contracts/governance/RewardAuctionHouse.sol#152-157) uses timestamp for comparisons
        Dangerous comparisons:
        - id == 0 || auctions[asset][id - 1].end < block.timestamp (contracts/governance/RewardAuctionHouse.sol#154)
VeCreditSwapToken._stake(uint256,uint256,address) (contracts/governance/VeCreditSwapToken.sol#107-132) uses timestamp for comparisons
        Dangerous comparisons:
```

- As a result of the tests carried out with the Slither tool, some results were obtained and reviewed by Halborn. Based on the results reviewed, some vulnerabilities were determined to be false positives. The actual vulnerabilities found by Slither are already included in the report findings.

THANK YOU FOR CHOOSING

// HALBORN