RENASCENCE

Philand Audit Report

Version 2.0

Audited by:

peakbolt

SpicyMeatball

Koolex

bytes032

August 16, 2024

Contents

1 Introduction

1.1 About Renascence

Renascence Labs was established by a team of experts including HollaDieWaldfee, MiloTruck, alexxander and bytes032.

Our founders have a distinguished history of achieving top honors in competitive audit contests, enhancing the security of leading protocols such as Reserve Protocol, Arbitrum, MaiaDAO, Chainlink, Dodo, Lens Protocol, Wenwin, PartyDAO, Lukso, Perennial Finance, Mute and Taurus.

We strive to deliver tailored solutions by thoroughly understanding each client's unique challenges and requirements. Our approach goes beyond addressing immediate security concerns; we are dedicated to fostering the enduring success and growth of our partners.

More of our work can be found here.

1.2 Disclaimer

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

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

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

1.3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

1.3.1 Impact

- High - Funds are **directly** at risk, or a **severe** disruption of the protocol's core functionality
- Medium - Funds are **indirectly** at risk, or **some** disruption of the protocol's functionality
- Low - Funds are **not** at risk

1.3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

2 Executive Summary

2.1 About Philand

Phi is a whole new Web3 world created directly from ENS & wallet activities, enabling the easy visualization of on-chain identities. It encourages users to interact with various web3 protocols, which provides a positive feedback loop to the entire Crypto ecosystem.

It's mission is to visualize on-chain identity and create an open and inclusive metaverse land system that is based on universal web3 building blocks, such as ENS domains and wallet activity. Through familiar gamification like Simcity, Animal Crossing, and Habbo Hotel, Phi contributes to the ecosystem by onboarding the next million people to crypto.

2.2 Overview

Project	Philand
Repository	phi-protocol
Commit Hash	92b2e5e269da...
Mitigation Hash	928f2c2a8af8...
Date	29 July 2024 - 7 August 2024

2.3 Issues Found

Severity	**Count**
High Risk	6
Medium Risk	13
Low Risk	12
Informational	4
Total Issues	**35**

3 Findings Summary

ID	Description	Status
H-1	Reward distribution is vulnerable to flash loan attacks	Resolved
H-2	Reward distribution will fail if one of the curators is address zero	Resolved
H-3	Anyone can call `depositRewards` function	Resolved
H-4	Incorrect validation for artist in `PhiNFT1155.updateRoyalties()`	Resolved
H-5	Minter can avoid paying `creatorFee` during minting	Resolved
H-6	Creator can bypass the bonding curve and perform two trades at the same price	Resolved
M-01	Reward distribution may fail because of the rounding error	Resolved
M-02	Users can batch trade even if the `Cred` contract is paused	Resolved
M-03	USDT/USDC will fail to work with `ContributeRewards.sol`	Resolved
M-04	Referral reward will be unclaimable if referral address is not specified	Resolved
M-05	Anyone can deny reward initialization via `setRewardInfo`	Resolved
M-06	`createCred()` and `updateCred()` are vulnerable to signature replay attack	Resolved
M-07	Missing `credId` validation for `_handleSignal()` and `_executeBatchTrade()`	Resolved
M-08	Incorrect ETH transfer in `_processClaim()` could cause minting to fail	Resolved
M-09	Incorrect use of `msg.value` for `batchClaim()` could cause batch minting to fail	Resolved
M-10	PhiFactory won't be able to pause the NFT	Resolved
M-11	There are no slippage checks in the cred trading functions	Resolved
M-12	Cred creator can implement a "honeypot" for cred traders	Resolved
M-13	Exploiting fixed MAX_SUPPLY to block user participation in Credential Signaling	Resolved
L-01	Missing `credId` validation in `CuratorRewardsDistributor`	Resolved
L-02	Incorrect validation in `PhiRewards.handleRewardsAndGetValueSent()`	Resolved
L-03	Missing refund of excess mint fee in `createArtFromFactory()`	Resolved
L-04	Excess payment should be refunded to the caller when signalling on behalf	Resolved
L-05	Missing validation for creator in `_credCredInternal()`	Resolved
L-06	`_updateCuratorSignalBalance()` fails to remove `credId` properly and cause duplicate `credId`	Resolved

ID	Description	Status
L-07	Incorrect increment of `artIdCounter` will cause mismatch of `artId` during `createArt()`	Resolved
L-08	It is possible to batch trade zero amount of creds	Resolved
L-09	createCred function has the whenNotPaused modifier, but updateCred function does not	Resolved
L-10	Lack of upper bound check for mintFee	Acknowledged
L-11	Unnecessary receive() function in multiple contracts	Resolved
L-12	Missing artId_ validation: a potential source of unexpected behavior	Resolved
I-1	Duplicate `verifier` validation for `PhiRewards.depositRewards()`	Resolved
I-2	Unnecessary ETH transfer in `Cred`	Resolved
I-3	`nonReentrant` modifier is not initialized	Resolved
I-4	Duplicate `credType` check	Resolved

4 Findings

High Risk

[H-1] Reward distribution is vulnerable to flash loan attacks

Context: CuratorRewardsDistributor.sol#L104-L105

Description: When the `distribute` function is called curators receive rewards shares based on the number of signals they have for a given cred ID:

```
    function distribute(uint256 credId) external {
        ---SNIP---
        for (uint256 i = 0; i < distributeAddresses.length; i++) {
            address user = distributeAddresses[i];

            uint256 userSignals = credContract.getSignalNumber(credId, user);
»           uint256 userRewards = (distributeAmount * userSignals) / totalSignals;
```

A distributor can significantly increase his number of signal with a flash loan and take most of the distributed reward for himself. After the attack is done, he returns the loan and sells his signals. This will discourage honest curators from purchasing and holding signals.

Recommendation: Add a cooldown period after a user signals for a cred, during which he can't perform the unsignal. This will prevent attacker from returning the flash loan.

Client: Fixed in https://github.com/ZaK3939/phi-protocol/commit/1a9c8f8f14ee754b60b979d768d317b8927983

Renascence: Verified, a cooldown period is added during which users cannot unsignal a cred.

[H-2] Reward distribution will fail if one of the curators is address zero

Context: CuratorRewardsDistributor.sol#L82 RewardControl.sol#L82

Description: The `distribute` function will send rewards to all curators according to the amount of signal they have for a given cred ID:

```
    function distribute(uint256 credId) external {
        uint256 totalBalance = balanceOf[credId];
        if (totalBalance == 0) {
            revert NoBalanceToDistribute();
        }

        address[] memory distributeAddresses =
        credContract.getCuratorAddresses(credId, 0, 0);

        ---SNIP---

        //slither-disable-next-line arbitrary-send-eth
»       phiRewardsContract.depositBatch{ value: distributeAmount }(
            distributeAddresses, amounts, reasons, "deposit from curator rewards
            distributor"
        );
```

However, the `RewardControl` control has a check that reverts the transaction if one of the addresses is 0. As a result, the whole distribution can be blocked if someone signal a cred for a zero address.

Recommendation: Add zero address validation when user purchases signals for another address.

Client: Fixed in https://github.com/ZaK3939/phi-protocol/pull/43/commits/f6241e15abfdb8311e621f75175b0482664150c2

Renascence: Verified, address zero verification is added to all trade functions in `Cred.sol`.

[H-3] Anyone can call `depositRewards` function

Context: PhiRewards.sol#L78-L124

Description: The `depositRewards` function updates users' internal balances based on the arguments provided:

```
    function depositRewards(
        uint256 artId_,
        uint256 credId_,
        bytes calldata addressesData_,
        uint256 artistTotalReward_,
        uint256 referralTotalReward_,
        uint256 verifierTotalReward_,
        uint256 curateTotalReward_,
        bool chainSync_
    )
        external
        payable
    {
        (address minter_, address receiver_, address referral_, address verifier_) =
            abi.decode(addressesData_, (address, address, address, address));

        if (receiver_ == address(0) || minter_ == address(0) || verifier_ ==
        address(0) || verifier_ == address(0)) {
            revert InvalidAddressZero();
        }

        if (referral_ != address(0)) {
            if (referral_ == minter_) {
                artistTotalReward_ += referralTotalReward_;
                referralTotalReward_ = 0;
            } else {
»               balanceOf[referral_] += referralTotalReward_;
            }
        }

»      balanceOf[verifier_] += verifierTotalReward_;
»      balanceOf[receiver_] += artistTotalReward_;

        bytes memory rewardsData;
        if (chainSync_ && address(curatorRewardsDistributor) != address(0)) {
            //slither-disable-next-line arbitrary-send-eth
            curatorRewardsDistributor.deposit{ value: curateTotalReward_ }(credId_,
            curateTotalReward_);
            rewardsData = abi.encode(artistTotalReward_, referralTotalReward_,
            verifierTotalReward_, curateTotalReward_);
        } else {
```

```
»          balanceOf[receiver_] += curateTotalReward_;
            rewardsData =
                abi.encode(artistTotalReward_ + curateTotalReward_,
                referralTotalReward_, verifierTotalReward_, 0);
        }
```

As we can see, it doesn't require permissions, meaning users can freely call it and update their balances with arbitrary values, which can then be withdrawn for ETH.

Recommendation: This function should be internal and called from `handleRewardsAndGetValueSent`.

Client: Fixed in https://github.com/ZaK3939/phi-protocol/pull/43/commits/2babc0dad67304cd5515ea57c107b

Renascence: Verified, `depositRewards` is now an internal function.

[H-4] Incorrect validation for artist in `PhiNFT1155.updateRoyalties()`

Context:

- PhiNFT1155.sol#L77-L82

Description: `PhiNFT1155.updateRoyalties()` has a `onlyArtCreator` modifier that ensure that only the artist for the particular art can update the royalties.

However, `onlyArtCreator` incorrectly checks the caller against the `artAddress` instead of the `artist`. This prevents the artist from updating the royalties for the NFT. This could result in lost royalties if the artist wish to increase the royalty amount.

```
    modifier onlyArtCreator(uint256 tokenId_) {
        uint256 artId = _tokenIdToArtId[tokenId_];
        //@audit this should be artData(artId).artist instead
»>      address artist = phiFactoryContract.artData(artId).artAddress;
        if (msg.sender != artist && msg.sender != owner()) revert NotArtCreator();
        _;
    }

    function updateRoyalties(
        uint256 tokenId_,
        RoyaltyConfiguration memory configuration
    )
        external
        onlyArtCreator(tokenId_)
    {
        _updateRoyalties(tokenId_, configuration);
    }
```

Recommendation: Fix the validation by checking against `artist` instead of `artAddress` as follows,

```
    modifier onlyArtCreator(uint256 tokenId_) {
        uint256 artId = _tokenIdToArtId[tokenId_];
-        address artist = phiFactoryContract.artData(artId).artAddress;
+        address artist = phiFactoryContract.artData(artId).artist;
        if (msg.sender != artist && msg.sender != owner()) revert NotArtCreator();
        _;
    }
```

Client: Fixed as per recommendation in https://github.com/ZaK3939/phi-protocol/pull/43/commits/48ba575f35f8c674e5ce139d76ca337158dcaed3.

Renascence: Issue has been resolved as per recommendation.

[H-5] Minter can avoid paying `creatorFee` during minting

Context: PhiFactory.sol#L332-L333

Description: PhiFactory allows minting of NFT using `merkleClaim()`, which will verify the minter is eligible to mint using merkle proof.

However, it fails to validate the `credCreator_` in this case, allowing the minter to set it to his own address and avoid paying credCreator the `creatorFee` during minting.

This will cause credCreators to lose their `creatorFee` as it can be easily exploited by anyone.

```
function merkleClaim(
    bytes32[] calldata proof_,
    bytes calldata encodeData_,
    MintArgs calldata mintArgs_,
    bytes32 leafPart_
)
    external
    payable
    whenNotPaused
{

    //@audit missing validation for credCreator_
    (address minter_, address ref_, address credCreator_, uint256 artId_) =
        abi.decode(encodeData_, (address, address, address, uint256));

}
```

Recommendation: Verify the `credCreator` address or obtain it from a verified source.

Client: Fixed as per recommendation in https://github.com/ZaK3939/phi-protocol/commit/087cc67f7c7ace51d92dff386332b119efe77b75.

Renascence: Issue has been resolved as per recommendation. The `credCreator` for merkle claim is obtained from the verified `arts[]` storage variable.

[H-6] Creator can bypass the bonding curve and perform two trades at the same price

Context: Cred.sol#L649

Description: When a batch trade is executed, the contract will precalculate prices for each cred ID and call `_executeBatchTrade`:

```
    function batchSignalCred(
        uint256[] calldata credIds_,
        uint256[] calldata amounts_,
        address holder_
    )
        public
        payable
    {
»       (uint256 totalCost, uint256[] memory prices, uint256[] memory protocolFees,
uint256[] memory creatorFees) =
            _validateAndCalculateBatch(credIds_, amounts_, true);

        if (totalCost > msg.value) revert InsufficientBatchPayment();

        _executeBatchSignal(credIds_, amounts_, holder_, prices, protocolFees,
        creatorFees);

        uint256 refund = msg.value - totalCost;
        if (refund > 0) {
            _msgSender().safeTransferETH(refund);
        }
    }

    function batchUnSignalCred(uint256[] calldata credIds_, uint256[] calldata
    amounts_) public {
»       (uint256 totalPayout, uint256[] memory prices, uint256[] memory protocolFees,
uint256[] memory creatorFees) =
            _validateAndCalculateBatch(credIds_, amounts_, false);

        _executeBatchUnSignal(credIds_, amounts_, prices, protocolFees, creatorFees);

        _msgSender().safeTransferETH(totalPayout);
    }
```

Inside of the `_executeBatchTrade` function the curator balances and cred supplies are updated:

```
    function _executeBatchTrade(
        uint256[] calldata credIds_,
        uint256[] calldata amounts_,
        address trader,
        uint256[] memory prices,
        uint256[] memory protocolFees,
        uint256[] memory creatorFees,
        bool isSignal
    )
        internal
    {
        for (uint256 i = 0; i < credIds_.length; ++i) {
```

```
            uint256 credId = credIds_[i];
            uint256 amount = amounts_[i];

            _updateCuratorSignalBalance(credId, trader, amount, isSignal);

            if (isSignal) {
                creds[credId].currentSupply += amount;
                address(this).safeTransferETH(prices[i]);
            } else {
                creds[credId].currentSupply -= amount;
            }

            creds[credId].latestActiveTimestamp = block.timestamp;

            protocolFeeDestination.safeTransferETH(protocolFees[i]);
»           creds[credId].creator.safeTransferETH(creatorFees[i]);
```

Note, that the fee is sent to the creator inside the loop. This allows a creator to send another `signalCred` or `unsignalCred` transaction in the `receive` callback. Since the prices in batch trading are precalculated it is possible to perform two trades at the same price. Consider the following scenario:

- Bob creates two creds ID2 - 1 token, ID3 - 2 tokens
- then he batch sells ID2 - 1 token and ID3 - 1 token
- in the receive fee callback for ID2 he sells his second ID3 token at the same price as his first, because supply wasn't updated yet.

This will allow creator to receive more ETH from trades and in worst case scenario break the accounting and prevent users from selling their credits as the contract may have insufficient funds.

Recommendation: Add `nonReentrant` modifier to the `_executeBatchTrade` and `_handleSignal` functions.

Client: Fixed in https://github.com/ZaK3939/phi-protocol/pull/43/commits/b530b4148f44fd17f7ac386c808f60c23e185b6a

Renascence: Verified, `nonReentrant` modifier is added to `_executeBatchTrade`.

Medium Risk

[M-01] Reward distribution may fail because of the rounding error

Context: CuratorRewardsDistributor.sol#L105-L117 RewardControl.sol#L74

Description: When reward is distributed among the curators, the `depositBatch` function expects `msg.value` to be equal to the sum of all curators' rewards:

```
    function depositBatch(
        address[] calldata recipients,
        uint256[] calldata amounts,
        bytes4[] calldata reasons,
        string calldata comment
    )
        external
        payable
    {
        ---SNIP---

        uint256 expectedTotalValue;
        for (uint256 i = 0; i < numRecipients; i++) {
            expectedTotalValue += amounts[i];
        }

»      if (msg.value != expectedTotalValue) {
            revert InvalidDeposit();
        }
```

Unfortunately this condition may not be met in some cases, let's examine the following scenario:

- 1 ether distribution is called, and there are 3 curators with 1 signal each
- according to the formula they should all receive 333 333 333 333 333 333 wei
- in the `depositBatch` a comparison is made, 1e18 != 999 999 999 999 999 999, and the transaction reverts

Recommendation: Calculate the sum of all rewards, and if it is less than the total amount to distribute, send the remainder to the distributor or the last curator in the list

Client: Fixed in https://github.com/ZaK3939/phi-protocol/pull/43/commits/a0ec7d5977d854a192f060ed02a2764978d798f6

Renascence: Verified, the sum of all rewards is transferred to `depositBatch`, the distributor receives the remainder.

[M-02] Users can batch trade even if the `Cred` contract is paused

Context: Cred.sol#L622-L631

Description: The `Cred.sol` contract owner can pause the contract and disable signal/unsignal operations for single creds. However, batch operations with creds are allowed even in the paused state:

```
function _executeBatchTrade(
    uint256[] calldata credIds_,
    uint256[] calldata amounts_,
    address trader,
    uint256[] memory prices,
    uint256[] memory protocolFees,
    uint256[] memory creatorFees,
    bool isSignal
)
    internal
```

This makes the contract pausing useless, since users can still trade signals.

Recommendation: Add `whenNotPaused` modifier to the `_executeBatchTrade` function.

Client: Fixed in https://github.com/ZaK3939/phi-protocol/pull/43/commits/74c70345419e35626233e6d6f23a0

Renascence: Verified, added `whenNotPaused` modifier to `_executeBatchTrade` function.

[M-03] USDT/USDC will fail to work with `ContributeRewards.sol`

Context: ContributeRewards.sol#L89 ContributeRewards.sol#L128 ContributeRewards.sol#L116

Description: Some tokens like USDT(USDC), do not return a boolean value when transferred, resulting in users not being able to use them as rewards for contributors:

```
if (!info.rewardToken.transferFrom(msg.sender, address(this), _totalRewardAmount))
revert TransferFailed();
```

Recommendation: Use SafeERC20 library from OpenZeppelin.

Client: Fixed in https://github.com/ZaK3939/phi-protocol/pull/43/commits/be4c2c0f0daf5d1a940875b1f1e947ac49c121d6, https://github.com/ZaK3939/phi-protocol/pull/43/commits/5286de9d85f6265e51be04b80cc590f11a911041

Renascence: Verified, SafeERC20 library is used to transfer tokens

[M-04] Referral reward will be unclaimable if referral address is not specified

Context: PhiRewards.sol#L98-L105

Description: Users pay a fee when they claim an art from the factory. This fee is split between the referral, artist and verifier in the `depositRewards` function:

```
    function depositRewards(
        uint256 artId_,
        uint256 credId_,
        bytes calldata addressesData_,
        uint256 artistTotalReward_,
        uint256 referralTotalReward_,
        uint256 verifierTotalReward_,
        uint256 curateTotalReward_,
        bool chainSync_
    )
        external
        payable
    {
        (address minter_, address receiver_, address referral_, address verifier_) =
            abi.decode(addressesData_, (address, address, address, address));

        if (receiver_ == address(0) || minter_ == address(0) || verifier_ ==
        address(0) || verifier_ == address(0)) {
            revert InvalidAddressZero();
        }

»      if (referral_ != address(0)) {
            if (referral_ == minter_) {
                artistTotalReward_ += referralTotalReward_;
                referralTotalReward_ = 0;
            } else {
»               balanceOf[referral_] += referralTotalReward_;
            }
        }
```

A situation may arise when the referral address is not specified, in which case the referral part of the reward will not be distributed and will remain in the contract.

Recommendation: Add referral reward to `artistTotalReward` if referral address is not specified.

Client: Fixed in https://github.com/ZaK3939/phi-protocol/pull/43/commits/51720029e5de9410cca10993e7804

Renascence: Verified, referral reward is transferred to the artist in the absence of a referral address.

[M-05] Anyone can deny reward initialization via `setRewardInfo`

Context: ContributeRewards.sol#L74-L76

Description: `ContributeRewards.setRewardInfo()` will ensure that new `rewardSetter` is the same address as the previous `rewardSetter` for the specified `credId`.

However, when `rewardCount[CredId] == 0`, there is no check on the `msg.sender`. That means anyone can set `rewardInfo` for any `credId` and prevent others from setting `RewardInfo` for that `credId`. That is because, once the `rewardSetter` is set, only the first who set the `rewardInfo` can set future `rewardInfo` for the `credId`.

```
function setRewardInfo(
    uint256 credId,
    uint256 _claimPeriodEnds,
    bytes32 _merkleRoot,
    address _rewardToken,
    uint256 _totalRewardAmount,
    bool _isCheckMinted
)
    external
{
    if (!credContract.isExist(credId)) revert CredDoesNotExist(credId);
    if (rewardCount[credId] > 0 && rewardInfos[credId][rewardCount[credId] -
    1].rewardSetter != msg.sender) {
        revert Unauthorized(msg.sender);
    }
```

Recommendation: Remove the `rewardSetter` validation to make `setRewardInfo` permissionless so that it is not restricted to any `rewardSetter`. Only use `rewardSetter` to verify `closeAndSweep()`.

```
    function setRewardInfo(
        uint256 credId,
        uint256 _claimPeriodEnds,
        bytes32 _merkleRoot,
        address _rewardToken,
        uint256 _totalRewardAmount,
        bool _isCheckMinted
    )
        external
    {
        if (!credContract.isExist(credId)) revert CredDoesNotExist(credId);
-        if (rewardCount[credId] > 0 && rewardInfos[credId][rewardCount[credId] -
1].rewardSetter != msg.sender) {
-            revert Unauthorized(msg.sender);
-        }

        uint256 rewardId = rewardCount[credId];
        RewardInfo storage info = rewardInfos[credId][rewardId];
        info.claimPeriodEnds = _claimPeriodEnds;
        info.merkleRoot = _merkleRoot;
        info.rewardToken = IERC20(_rewardToken);
        info.totalRewardAmount = _totalRewardAmount;
```

```
        info.rewardSetter = msg.sender;
        info.isCheckMinted = _isCheckMinted;
        info.isOpen = true;

        rewardCount[credId]++;
        if (!info.rewardToken.transferFrom(msg.sender, address(this),
        _totalRewardAmount)) revert TransferFailed();

        emit RewardInfoSet(
            msg.sender, credId, rewardId, _claimPeriodEnds, _merkleRoot, _rewardToken,
            _totalRewardAmount
        );
    }
```

Client: Fixed as per recommendation in https://github.com/ZaK3939/phi-protocol/pull/43/commits/556eadd57ce98431a363f6465aec7d0149d3a39f.

Renascence: Issue has been resolved as per recommendation.

[M-06] `createCred()` and `updateCred()` are vulnerable to signature replay attack

Context:

- Cred.sol#L216-L224

Description: `createCred()` and `updateCred()` will verify that the `signedData_` are from the `PhiSignerAddress` using the signature.

However, there are no anti-replay mechanisms to prevent the re-use of the signature, which will allow anyone to use the signature and create/update credentials with the same `signedData_`, which could be outdated or not meant for the specific user.

```
function createCred(
    address creator_,
    bytes calldata signedData_,
    bytes calldata signature_,
    uint16 signalRoyalty_,
    uint16 unSignalRoyalty_
)
    public
    payable
    whenNotPaused
{
    if (_recoverSigner(keccak256(signedData_), signature_) != phiSignerAddress)
    revert AddressNotSigned();
    (
        ,
        address bondingCurve,
        string memory credURL,
        string memory credType,
        string memory verificationType,
        bytes32 merkleRoot_
    ) = abi.decode(signedData_, (uint256, address, string, string, string, bytes32));
```

Recommendation: Recommend to add anti-replay mechanisms to `createCred()` and `updateCred()` so that the signatures have an expiry date and is verified against the sender.

Client: Fixed as per recommendation in https://github.com/ZaK3939/phi-protocol/pull/43/commits/3686012064b12808463339b5678d9f0e22b75a80.

Renascence: Issue has been resolved as per recommendation. Both `createCred()` and `updateCred()` will verified `expiresIn` and `sender` that are in `signedData_` to ensure that the signature is limited to specific caller for a limited period of time, to prevent replay attack using others old signature.

[M-07] Missing `credId` validation for `_handleSignal()` and `_executeBatchTrade()`

Context: Cred.sol#L548-L595

Description: Both `_handleSignal()` and `_executeBatchTrade()` handles the signal purchases/sales for credentials but they fail to check that the `credId` is valid.

This issue could allow anyone to purchase the signal for the next `credId`, even when it does not exists yet.

An attacker can abuse this and frontrun `createCred()` to deny the credCreator from creating credentials as the first signal would have been bought by the attacker, causing the price to increase. In the worst case, if the credCreator paid excess ETH, the attacker could cause the credCreator to pay more due to the increased price.

Recommendation:

```
    function _handleSignal(uint256 credId_, uint256 amount_, bool isSignal, address
    sender_) internal whenNotPaused {
        if (amount_ == 0) {
            revert InvalidAmount();
        }

+       if (!isExist(credId_)) {
+           revert InvalidCredId();
+       }
        ...
    }
```

```
    function _executeBatchTrade(
        uint256[] calldata credIds_,
        uint256[] calldata amounts_,
        address trader,
        uint256[] memory prices,
        uint256[] memory protocolFees,
        uint256[] memory creatorFees,
        bool isSignal
    )
        internal
        whenNotPaused
    {
        for (uint256 i = 0; i < credIds_.length; ++i) {
            uint256 credId = credIds_[i];
+            if (!isExist(credId)) {
+                revert InvalidCredId();
+            }
```

Client: Fixed as per recommendation in https://github.com/ZaK3939/phi-protocol/pull/43/commits/db8c24735d73b15fc50e07946cc7e76afe00926a and https://github.com/ZaK3939/phi-protocol/pull/43/commits/04f3dc7cdabadd45f44522fa207e243dce44c22a.

Renascence: Issue has been resolved as per recommendation.

[M-08] Incorrect ETH transfer in `_processClaim()` could cause minting to fail

Context:

- PhiFactory.sol#L708-L719

Description: `_processClaim()` will refund excess ETH payment after accounting for the `mintFee`.

However, it fails to consider the ETH fund when sending ETH to `claimFromFactory()` as it is using `msg.value` instead of `mintFee`.

```
function _processClaim(
    uint256 artId_,
    address minter_,
    address ref_,
    address verifier_,
    uint256 quantity_,
    bytes32 data_,
    string memory imageURI_
)
    private
{
    PhiArt storage art = arts[artId_];

    // Handle refund
    uint256 mintFee = getArtMintFee(artId_, quantity_);
    if ((msg.value - mintFee) > 0) {
        _msgSender().safeTransferETH(msg.value - mintFee);
    }
    protocolFeeDestination.safeTransferETH(mintProtocolFee * quantity_);
```

```
    (bool success_,) = art.artAddress.call{ value: msg.value - mintProtocolFee *
    quantity_ }(
        abi.encodeWithSignature(
            "claimFromFactory(uint256,address,address,address,uint256,bytes32,string)
            ",
            artId_,
            minter_,
            ref_,
            verifier_,
            quantity_,
            data_,
            imageURI_
        )
    );

    if (!success_) revert ClaimFailed();
}
```

Recommendation: `claimFromFactory()` should send `mintFee - mintProtocolFee * quantity_` as follows:

```
-       (bool success_,) = art.artAddress.call{ value: msg.value - mintProtocolFee *
quantity_ }(
+       (bool success_,) = art.artAddress.call{ value: mintFee - mintProtocolFee *
quantity_ }(
            abi.encodeWithSignature(
                "claimFromFactory(uint256,address,address,address,uint256,bytes32,str
                ing)",
                artId_,
                minter_,
                ref_,
                verifier_,
                quantity_,
                data_,
                imageURI_
            )
        );
```

Client: Fixed as per recommendation in https://github.com/ZaK3939/phi-protocol/pull/43/commits/25104eaeb588a5cb173694169626a8ee1054e2b4.

Renascence: Issue has been resolved as per recommendation.

[M-09] Incorrect use of `msg.value` for `batchClaim()` could cause batch minting to fail

Context:

- PhiFactory.sol#L688-L722

Description: `PhiFactory` allows batch minting of NFT using `batchClaim()`, which is using `_processClaim()` to handle each mint.

However, `_processClaim()` is using `msg.value` to process each claim, which is actually meant for all the claims. This will cause batch claim to fail as it will refund the payment for subsequent claim right after the first claim.

```
function _processClaim(
    uint256 artId_,
    address minter_,
    address ref_,
    address verifier_,
    uint256 quantity_,
    bytes32 data_,
    string memory imageURI_
)
    private
{
    PhiArt storage art = arts[artId_];

    // Handle refund
    uint256 mintFee = getArtMintFee(artId_, quantity_);
    if ((msg.value - mintFee) > 0) {
        _msgSender().safeTransferETH(msg.value - mintFee);
    }
    protocolFeeDestination.safeTransferETH(mintProtocolFee * quantity_);

    (bool success_,) = art.artAddress.call{ value: msg.value - mintProtocolFee *
    quantity_ }(
        abi.encodeWithSignature(
            "claimFromFactory(uint256,address,address,address,uint256,bytes32,string)
            ",
            artId_,
            minter_,
            ref_,
            verifier_,
            quantity_,
            data_,
            imageURI_
        )
    );

    if (!success_) revert ClaimFailed();
}
```

Recommendation: For batch, we should pass in a new `msgValue` variable as a separate parameter for each claim. then verify that the sum is equal to `msg.value`.

Client: Fixed as per recommendation in https://github.com/ZaK3939/phi-protocol/pull/43/commits/b9262a9b8816a575cfe4953b39b6368502dcd3a3.

Renascence: Issue has been resolved as per recommendation.

[M-10] PhiFactory wont be able to pause the NFT

Context: PhiNFT1155.sol#L128-L135

Description: The `PhiNFT1155` token owner can put token in the paused state and prevent functions `createArtFromFactory` and `claimFromFactory` from being called:

```
/// @notice Pauses the contract.
function pause() external onlyOwner {
    _pause();
}

/// @notice Unpauses the contract.
function unPause() external onlyOwner {
    _unpause();
}
```

Unfortunately, it is impossible to pause the token, because the owner is the `PhiFactory` contract that creates and initializes the token:

```
    function initialize(
        uint256 credChainId_,
        uint256 credId_,
        string memory verificationType_,
        address protocolFeeDestination_
    )
        external
        initializer
    {
»       __Ownable_init(msg.sender);
```

and there are no functions in the factory that can externally call `pause` and `unpause` functions.

Recommendation: Create functions that would call `pause` and `unpause` functions of the token from the `PhiFactory`.

Client: Fixed in https://github.com/ZaK3939/phi-protocol/pull/43/commits/a0264638d50fa5e10491e2cf2ef73248cdea8d79

Renascence: Verified, factory contract can pause NFT tokens

[M-11] There are no slippage checks in the cred trading functions

Context: Cred.sol#L159-L199

Description: The `Cred.sol` contract allows users to creds using the bonding curve algorithm. The price is updated based on the current cred supply and in general increases with each cred purchased:

```
function _curve(uint256 targetAmount_) private pure returns (uint256) {
    return (TOTAL_SUPPLY_FACTOR * CURVE_FACTOR * 1 ether) / (TOTAL_SUPPLY_FACTOR -
    targetAmount_)
        - CURVE_FACTOR * 1 ether - INITIAL_PRICE_FACTOR * targetAmount_ / 1000;
}
```

A user who wishes to purchase (signal) or sell (unsignal) creds is vulnerable to sandwich attacks. Consider the following scenario:

- Alice calls `unsignalCred` function to sell 1 cred for 0.1 ETH
- Bob frontruns her transaction and sells 10 creds moving price down
- Alice's transaction is executed at the lower price than she expected

Recommendation: Allow users to specify `maxPrice` and `minPrice` for buy and sell prices respectively and revert the trade if max/min amounts conditions are not met:

```
if(price + protocolFee + creatorFee > maxPrice) revert // for signal
if(price - protocolFee - creatorFee < minPrice) revert // for unsignal
```

Client: Fixed in https://github.com/ZaK3939/phi-protocol/pull/43/commits/25538da90e7f3d9b985c7b13a6374

Renascence: Verified, users can now specify price bounds for their trades.

[M-12] Cred creator can implement a "honeypot" for cred traders

Context: Cred.sol#L590 Cred.sol#L649

Description: The cred creator receives fees from every cred trade in the form of ETH tokens:

```
    function _handleSignal(uint256 credId_, uint256 amount_, bool isSignal, address
    sender_) internal whenNotPaused {
        if (amount_ == 0) {
            revert InvalidAmount();
        }

        PhiCred storage cred = creds[credId_];

        uint256 supply = cred.currentSupply;
        address creator = cred.creator;

        (uint256 price, uint256 protocolFee, uint256 creatorFee) =
```

```
            IBondingCurve(cred.bondingCurve).getPriceData(credId_, supply, amount_,
            isSignal);
        ---SNIP---
        protocolFeeDestination.safeTransferETH(protocolFee);
»       creator.safeTransferETH(creatorFee);
    }

    function _executeBatchTrade(
        uint256[] calldata credIds_,
        uint256[] calldata amounts_,
        address trader,
        uint256[] memory prices,
        uint256[] memory protocolFees,
        uint256[] memory creatorFees,
        bool isSignal
    )
        internal
    {
        for (uint256 i = 0; i < credIds_.length; ++i) {
            uint256 credId = credIds_[i];
            uint256 amount = amounts_[i];
            ---SNIP---
            protocolFeeDestination.safeTransferETH(protocolFees[i]);
»           creds[credId].creator.safeTransferETH(creatorFees[i]);
```

This allows a malicious creator to set up a contract that will revert the trade in the `receive` callback, creating a "honeypot" for traders. Foe example, Bob creates a cred and purchases 100 creds for himself, then he configures his contract to revert all `unSignal` transactions (contract can monitor the price change) and let the cred price grow. Once the price reaches a certain value, he can sell his creds for a significant profit.

Recommendation: Deposit creator fees into the `RewardControl` and let creator withdraw it afterwards.

Client: Fixed in https://github.com/ZaK3939/phi-protocol/pull/43/commits/212fa61a6743bfb65e91b7a1f0f36a2

Renascence: Verified, creator fee is sent to the `RewardControl` contract.

[M-13] Exploiting fixed MAX_SUPPLY to block user participation in Credential Signaling

Context: Cred.sol#L28

Description: The Cred contract allows users to purchase signals for credentials up to a maximum supply of 999. The contract doesn't implement individual purchase limits or protection against front-running attacks.

An attacker could exploit the current implementation to prevent legitimate users from participating in the signaling process. This attack leverages front-running and the fixed MAX_SUPPLY to effectively deny service to targeted users.

Attack steps:

1. Monitor the mempool for pending signal purchase transactions.
2. Front-run legitimate transactions with a higher gas price.
3. Purchase signals up to MAX_SUPPLY, causing subsequent transactions to fail.

4. Sell signals and repeat the process to maintain control.

This attack doesn't necessarily aim for profit but rather to prevent specific users from participating in the signaling process for certain credentials. It's particularly effective against credentials with lower royalty fees and could be used by wealthy actors to establish monopolies.

```
uint16 private constant MAX_SUPPLY = 999;
```

MAX_SUPPLY = 999

The vulnerability arises from:

1. The lack of individual purchase limits.
2. The fixed MAX_SUPPLY of 999.

Recommendation: To mitigate this issue, consider implementing one or more of the following measures:

1. Implement individual purchase limits.
2. Implement a cooldown period for large purchases.
3. Implement a dynamic pricing model that increases the cost for larger purchases

Client:

we already introduces SIGNAL_LOCK_PERIOD for signal and unsignal lock period

https://github.com/ZaK3939/phi-protocol/pull/43/commits/1a9c8f8f14ee754b60b979d768d317b892798320

Renascence: This fixes the issue from occurring in the same block which makes it more expensive.

Low Risk

[L-01] Missing `credId` validation in `CuratorRewardsDistributor`

Context:

- CuratorRewardsDistributor.sol#L68-L80

Description: `CuratorRewardsDistributor` allows deposit and distribution of rewards based on `credId`.

However, there are no checks to ensure that `credId` exists, which could cause the deposited reward to be lost if the `credId` is wrong.

```
function deposit(uint256 credId, uint256 amount) external payable {
    //@audit missing credId validation would allow deposit for non-existing credId
    if (msg.value != amount) {
        revert InvalidValue(msg.value, amount);
    }
    balanceOf[credId] += amount;
    emit Deposit(credId, amount);
}

function distribute(uint256 credId) external {
    //@audit missing credId validation
    uint256 totalBalance = balanceOf[credId];
    if (totalBalance == 0) {
        revert NoBalanceToDistribute();
    }
   ...
 }
```

Recommendation: Add `credId` validation checks for both `deposit()` and `distribute()`.

```
function deposit(uint256 credId, uint256 amount) external payable {
+       if (!credContract.isExist(credId)) revert InvalidCredId();
}

    function distribute(uint256 credId) external {
+       if (!credContract.isExist(credId)) revert InvalidCredId();
}
```

Client: Fixed as per recommendation in https://github.com/ZaK3939/phi-protocol/pull/43/commits/c6983a815e250b84804fb9b9588d8e8efaf78a74.

Renascence: Issue has been resolved as per recommendation.

[L-02] Incorrect validation in `PhiRewards.handleRewardsAndGetValueSent()`

Context: Description:

`PhiRewards.handleRewardsAndGetValueSent()` has an incorrect validation for the reward amount (`msg.value`), which do not validate `msg.value`.

```
function handleRewardsAndGetValueSent(
    uint256 artId_,
    uint256 credId_,
    uint256 quantity_,
    uint256 mintFee_,
    bytes calldata addressesData_,
    bool chainSync_
)
    external
    payable
{
    if (
        computeMintReward(quantity_, mintFee_)
            != (artistReward + mintFee_ + referralReward + verifierReward +
            curateReward) * quantity_
    ) {
        revert InvalidDeposit();
    }

    this.depositRewards{ value: msg.value }(
        artId_,
        credId_,
        addressesData_,
        quantity_ * (mintFee_ + artistReward),
        quantity_ * referralReward,
        quantity_ * verifierReward,
        quantity_ * curateReward,
        chainSync_
    );
}
```

Recommendation:

```
    function handleRewardsAndGetValueSent(
        uint256 artId_,
        uint256 credId_,
        uint256 quantity_,
        uint256 mintFee_,
        bytes calldata addressesData_,
        bool chainSync_
    )
        external
        payable
    {
-       if (
-           computeMintReward(quantity_, mintFee_)
```

```
-                != (artistReward + mintFee_ + referralReward + verifierReward +
curateReward) * quantity_
-         ) {

+       if (computeMintReward(quantity_, mintFee_) != msg.value) {
            revert InvalidDeposit();
        }

        this.depositRewards{ value: msg.value }(
            artId_,
            credId_,
            addressesData_,
            quantity_ * (mintFee_ + artistReward),
            quantity_ * referralReward,
            quantity_ * verifierReward,
            quantity_ * curateReward,
            chainSync_
        );
    }
```

Client: Fixed as per recommendation in https://github.com/ZaK3939/phi-protocol/pull/43/commits/726692863cc1b8b38959bbbf1d1fb292435b05fd.

Renascence: Issue has been resolved as per recommendation.

[L-03] Missing refund of excess mint fee in `createArtFromFactory()`

Context: PhiNFT1155.sol#L138-L150

Description: `PhiNFT1155.createArtFromFactory()` fails to refund any excess ETH that is paid by the artist to create the art for the credential.

```
function createArtFromFactory(uint256 artId_) external payable onlyPhiFactory
whenNotPaused returns (uint256) {
    _artIdToTokenId[artId_] = tokenIdCounter;
    _tokenIdToArtId[tokenIdCounter] = artId_;
    uint256 artFee = phiFactoryContract.artCreateFee();
    protocolFeeDestination.safeTransferETH(artFee);
    emit ArtCreated(artId_, tokenIdCounter);
    uint256 createdTokenId = tokenIdCounter;

    unchecked {
        tokenIdCounter += 1;
    }
    return createdTokenId;
}
```

Recommendation: Update to refund any excess payment after paying `artFee`.

```
    function createArtFromFactory(uint256 artId_) external payable onlyPhiFactory
    whenNotPaused returns (uint256) {
        _artIdToTokenId[artId_] = tokenIdCounter;
        _tokenIdToArtId[tokenIdCounter] = artId_;
        uint256 artFee = phiFactoryContract.artCreateFee();
        protocolFeeDestination.safeTransferETH(artFee);
        emit ArtCreated(artId_, tokenIdCounter);
        uint256 createdTokenId = tokenIdCounter;

        unchecked {
            tokenIdCounter += 1;
        }

+        if ((msg.value - artFee) > 0) {
+            _msgSender().safeTransferETH(msg.value - artFee);
+        }

        return createdTokenId;
    }
```

Client:

Renascence:

[L-04] Excess payment should be refunded to the caller when signalling on behalf

Context:

- Cred.sol#L582

Description: It is possible for any users to purchase and sell signals on behalf of another user using `signalCredFor()`.

However, when that occur, the excess payment are actually incorrectly sent to the signal recipient instead of the caller.

```
    function _handleSignal(uint256 credId_, uint256 amount_, bool isSignal, address
    sender_) internal whenNotPaused {

        if (isSignal) {
            cred.currentSupply += amount_;
            address(this).safeTransferETH(price);
            uint256 excessPayment = msg.value - price - protocolFee - creatorFee;
            if (excessPayment > 0) {
            //@audit this should refund to the caller for signal on behalf
»>            sender_.safeTransferETH(excessPayment);
            }
```

Recommendation: Refund the excess payment to the caller instead.

```
    function _handleSignal(uint256 credId_, uint256 amount_, bool isSignal, address
    sender_) internal whenNotPaused {

        ...
        if (isSignal) {
            cred.currentSupply += amount_;
            address(this).safeTransferETH(price);
            uint256 excessPayment = msg.value - price - protocolFee - creatorFee;
            if (excessPayment > 0) {
-                sender_.safeTransferETH(excessPayment);
+               _msgSender().safeTransferETH(excessPayment);
            }
```

Client: Fixed as per recommendation in https://github.com/ZaK3939/phi-protocol/pull/43/commits/a432f9f125ea7243f50d63b6c44e1f460b9f148a.

Renascence: Issue has been resolved as per recommendation.

[L-05] Missing validation for creator in `_credCredInternal()`

Context: Cred.sol#L505-L520

Description: `_credCredInternal()` fails to check that the `creator` is not `address(0)`, which is actually used for `isExist()` to check the existence of the `credId`.

```
function _createCredInternal(
    address creator_,
    string memory credURL_,
    string memory credType_,
    string memory verificationType_,
    address bondingCurve_,
    uint16 signalRoyalty_,
    uint16 unSignalRoyalty_
)
    internal
    whenNotPaused
    returns (uint256)
{
    if (!credType_.eq("BASIC") && !credType_.eq("ADVANCED")) {
        revert InvalidCredType();
    }

...
}

function isExist(uint256 credId_) external view returns (bool) {
    return creds[credId_].creator != address(0);
}
```

Recommendation:

```
    function _createCredInternal(
        address creator_,
        string memory credURL_,
        string memory credType_,
        string memory verificationType_,
        address bondingCurve_,
        uint16 signalRoyalty_,
        uint16 unSignalRoyalty_
    )
        internal
        whenNotPaused
        returns (uint256)
    {

+        if (creator_ == address(0)) {
+            revert InvalidAddressZero();
+        }
```

Client: Fixed as per recommendation in https://github.com/ZaK3939/phi-protocol/commit/b7e9a24afc1ca5384aad65b33eac33d0a319676f.

Renascence: Issue has been resolved as per recommendation.

[L-06] `_updateCuratorSignalBalance()` fails to remove `credId` properly and cause duplicate `credId`

Context: Cred.sol#L602-L617

Description: `_updateCuratorSignalBalance()` fails to remove the `credId_` from `_credIdsPerAddress[sender_]` when the `sender_` sold all his signals.

This will cause the next signal for the same credId_ to insert duplicate `credId` into `_credIdsPerAddress[sender_]`, causing duplicate credentials during retrieval.

```
function _updateCuratorSignalBalance(uint256 credId_, address sender_, uint256
amount_, bool isSignal) internal {
    (, uint256 currentSignals) = signalBalance[credId_].tryGet(sender_);

    if (isSignal) {
        if (currentSignals == 0 && !_credIdExistsPerAddress[sender_][credId_]) {
            _credIdsPerAddress[sender_].push(credId_);
            _credIdExistsPerAddress[sender_][credId_] = true;
        }
        signalBalance[credId_].set(sender_, currentSignals + amount_);
    } else {
        if ((currentSignals - amount_) == 0) {
           //@audit credId should be removed from _credIdsPerAddress
            _credIdExistsPerAddress[sender_][credId_] = false;
        }
        signalBalance[credId_].set(sender_, currentSignals - amount_);
    }
}
```

Recommendation: The issue could be fixed as follows:

```
+    mapping(address curator => uint256 arrayLength) private
_credIdsPerAddressArrLength;
+    mapping(address curator => mapping (uint256 credId => uint256 index)) private
_credIdsPerAddressCredIdIndex;


    function _updateCuratorSignalBalance(uint256 credId_, address sender_, uint256
    amount_, bool isSignal) internal {
        (, uint256 currentSignals) = signalBalance[credId_].tryGet(sender_);

        if (isSignal) {
            if (currentSignals == 0 && !_credIdExistsPerAddress[sender_][credId_]) {
-                  _credIdsPerAddress[sender_].push(credId_);
+               _addCredIdPerAddress(credId_, sender_);

                _credIdExistsPerAddress[sender_][credId_] = true;
            }
            signalBalance[credId_].set(sender_, currentSignals + amount_);
        } else {
            if ((currentSignals - amount_) == 0) {
+                _removeCredIdPerAddress(credId_, sender_);
                _credIdExistsPerAddress[sender_][credId_] = false;
            }
            signalBalance[credId_].set(sender_, currentSignals - amount_);
        }

    }

+    function _addCredIdPerAddress(uint256 credId_, address sender_) public {
+        _credIdsPerAddress[sender_].push(credId_);
+        _credIdsPerAddressCredIdIndex[sender_][credId_] =
_credIdsPerAddressArrLength;
+        _credIdsPerAddressArrLength[sender_]++;
+    }

+    function _removeCredIdPerAddress(uint256 credId_, address sender_) public {
+        if (_credIdsPerAddress[sender_].length == 0) revert EmptyArray();

+        uint256 indexToRemove = _credIdsPerAddressCredIdIndex[sender_][credId_];
+        if (indexToRemove >= _credIdsPerAddress[sender_].length)  revert
IndexOutofBounds();

+        uint256 credIdToRemove = _credIdsPerAddress[sender_][indexToRemove];
+        if (credId_ != credIdToRemove) revert WrongCredId();

+        uint256 lastIndex = _credIdsPerAddress[sender_].length - 1;
+        uint256 lastCredId = _credIdsPerAddress[sender_][lastIndex];
+         _credIdsPerAddress[sender_][indexToRemove] = lastCredId;

+        if (indexToRemove < lastIndex) {
+             _credIdsPerAddressCredIdIndex[sender_][lastCredId] = indexToRemove;
+        }

+        _credIdsPerAddress[sender_].pop();
```

```
+         delete _credIdsPerAddressCredIdIndex[sender_][credIdToRemove];

+         if (_credIdsPerAddressArrLength[sender_] > 0) {
+             _credIdsPerAddressArrLength[sender_]--;
+         }
+     }
```

Client: Fixed as per recommendation in https://github.com/ZaK3939/phi-protocol/pull/43/commits/d41ea990b580c73f36a2087ea785e833ad510ea4.

Renascence: Issue has been resolved as per recommendation.

[L-07] Incorrect increment of `artIdCounter` will cause mismatch of `artId` during `createArt()`

Context: PhiFactory.sol#L171-L187

Description: `_createERC1155Data` uses `artIdCounter` while `createERC1155Internal()` uses `artIdCounter++`.

That means the `artId` will not match during `createArt()` as the validation and execution are using different `artId`.

```
function createArt(
    bytes calldata signedData_,
    bytes calldata signature_,
    CreateConfig memory createConfig_
)
    external
    payable
    nonReentrant
    whenNotPaused
    returns (address)
{
    _validateArtCreationSignature(signedData_, signature_);
    (, string memory uri_, bytes memory credData) = abi.decode(signedData_, (uint256,
    string, bytes));
    ERC1155Data memory erc1155Data = _createERC1155Data(artIdCounter, createConfig_,
    uri_, credData);
    uint256 newArtId = artIdCounter++;
    return createERC1155Internal(newArtId, erc1155Data);
}
```

Recommendation:

```
-         uint256 newArtId = artIdCounter++;
-         return createERC1155Internal(newArtId, erc1155Data);

+         address artAddress = createERC1155Internal(artIdCounter, erc1155Data);
+         artIdCounter++;
+         return artAddress;

    }
```

Client: Fixed as per recommendation in https://github.com/ZaK3939/phi-protocol/pull/43/commits/72350754c62ed67927c1cb8dbe698d57e5ed0d58.

Renascence: Issue has been resolved as per recommendation.

[L-08] It is possible to batch trade zero amount of creds

Context: Cred.sol#L622-L653

Description: Unlike single trades, `_executebatchTrade` doesn't have validation for zero amounts:

```
    function _executeBatchTrade(
        uint256[] calldata credIds_,
        uint256[] calldata amounts_,
        address trader,
        uint256[] memory prices,
        uint256[] memory protocolFees,
        uint256[] memory creatorFees,
        bool isSignal
    )
        internal
    {
        for (uint256 i = 0; i < credIds_.length; ++i) {
            uint256 credId = credIds_[i];
»           uint256 amount = amounts_[i];
```

This allows users to add themselves or other addresses to the curator list for a given cred. Because the curator list is dynamic, it is possible to increase it's size and cause Out-of-Gas error in functions that iterate over it. For example, `CuratorRewardsDistributor.sol` searches the entire list for balances:

```
    function distribute(uint256 credId) external {
        uint256 totalBalance = balanceOf[credId];
        if (totalBalance == 0) {
            revert NoBalanceToDistribute();
        }

»      address[] memory distributeAddresses =
credContract.getCuratorAddresses(credId, 0, 0);
```

Recommendation: Verify amounts in the `_executeBatchTrade` function.

Client: Fixed is https://github.com/ZaK3939/phi-protocol/pull/43/commits/13da37cf4417d6fdf335af98910bd36

Renascence: Verified, the contract validates cred amounts in the batch trade.

[L-09] createCred function has the whenNotPaused modifier, but updateCred function does not

Context: src/Cred.sol:L205

Description: The Cred contract contains two key functions: createCred for creating new credentials and updateCred for updating existing ones. These functions have different pause controls, which could lead to inconsistent behavior.

The createCred function includes the whenNotPaused modifier, preventing the creation of new credentials when the contract is paused. However, the updateCred function lacks this modifier, allowing updates to existing credentials even when the contract is paused. This inconsistency could lead to unexpected behavior and potential security risks.

Recommendation: To ensure consistent behavior and improve security, consider applying the whenNotPaused modifier to the updateCred function as well.

Client: fixed ZaK3939/phi-protocol@c9694c5 (#43)

Renascence: Resolved

[L-10] Lack of upper bound check for mintFee

Context: PhiFactory.sol:L189

Description: The PhiFactory contract allows the creation of art pieces (NFTs) with associated mint fees. These mint fees are set during the art creation process and can be updated later.

Currently, there is no upper bound check for the mintFee when creating or updating art pieces. While this allows for flexibility, it also introduces a potential risk where extremely high mint fees could be set, either accidentally or maliciously. This could lead to unexpected behavior, user frustration, or even render the art piece effectively unmintable.

In both the `createArt` and `updateArtSettings` functions, there's no check to ensure that mintFee is within a reasonable range.

Recommendation: Implement an upper bound check for the mintFee to prevent potential issues.

Client: Acknowledged

[L-11] Unnecessary receive() function in multiple contracts

Context:

- Cred.sol:L802
- PhiFactory.sol:L755
- CuratorRewardsDistributor.sol:L126
- PhiRewards.sol:L167

Description: Several contracts in codebase include an empty receive() function, which allows them to accept ETH without any specific handling.

Recommendation: If these receive() functions are not explicitly needed for the contract's functionality, they should be removed to prevent potential issues and improve the overall security and clarity of the codebase.

Client:

Fixed in https://github.com/ZaK3939/phi-protocol/pull/43/commits/45fe224b211dfe80435b8b51897e1eeab951e

Renascence: The fix has been applied

[L-12] Missing artId_ validation: a potential source of unexpected behavior

Context: PhiFactory.sol:L441

Description: The function lacks validation to ensure the artId_ corresponds to an existing artwork. This omission may result in unexpected behavior.

Recommendation: Verify the provided artId_ actually exists in the system

Client:

Fixed at https://github.com/ZaK3939/phi-protocol/pull/43/commits/0b3c59e2b6b68e99a3322f157baed68d19e

Renascence: Verified

Informational

[I-1] Duplicate `verifier` validation for `PhiRewards.depositRewards()`

Context:

- PhiRewards.sol#L94-L96

Description: Within `PhiRewards.depositRewards()`, the `verifier` validation is duplicated.

```
function depositRewards(
    uint256 artId_,
    uint256 credId_,
    bytes calldata addressesData_,
    uint256 artistTotalReward_,
    uint256 referralTotalReward_,
    uint256 verifierTotalReward_,
    uint256 curateTotalReward_,
    bool chainSync_
)
    external
    payable
{
    (address minter_, address receiver_, address referral_, address verifier_) =
        abi.decode(addressesData_, (address, address, address, address));

    if (receiver_ == address(0) || minter_ == address(0) || verifier_ == address(0)
    || verifier_ == address(0)) {
        revert InvalidAddressZero();
    }
```

Recommendation: Remove the redundant `verifier` validation.

```
    function depositRewards(
        uint256 artId_,
        uint256 credId_,
        bytes calldata addressesData_,
        uint256 artistTotalReward_,
        uint256 referralTotalReward_,
        uint256 verifierTotalReward_,
        uint256 curateTotalReward_,
        bool chainSync_
    )
        external
        payable
    {
        (address minter_, address receiver_, address referral_, address verifier_) =
            abi.decode(addressesData_, (address, address, address, address));

-       if (receiver_ == address(0) || minter_ == address(0) || verifier_ ==
address(0) || verifier_ == address(0)) {
+       if (receiver_ == address(0) || minter_ == address(0) || verifier_ ==
address(0)) {
            revert InvalidAddressZero();
        }
```

Client: Fixed as per recommendation in https://github.com/ZaK3939/phi-protocol/pull/43/commits/b782ba07d888c456aae6a4ee8b3079b68cb33aeb.

Renascence: Issue has been resolved as per recommendation.

[I-2] Unnecessary ETH transfer in `Cred`

Context:

- Cred.sol#L579
- Cred.sol#L641

Description: In `Cred`, the creation of credential will perform an unnecessary ETH transfer to the contract itself. This is not required as the functions are payable, which means the ETH would have been transferred into the contract by the caller.

```
function _handleSignal(uint256 credId_, uint256 amount_, bool isSignal, address
sender_) internal whenNotPaused {

    if (isSignal) {
        cred.currentSupply += amount_;
        address(this).safeTransferETH(price);

}

function _executeBatchTrade(
    uint256[] calldata credIds_,
    uint256[] calldata amounts_,
    address trader,
    uint256[] memory prices,
    uint256[] memory protocolFees,
    uint256[] memory creatorFees,
    bool isSignal
)
    internal
{
    ...
        if (isSignal) {
            creds[credId].currentSupply += amount;
            address(this).safeTransferETH(prices[i]);
}
```

Recommendation: Remove the `address(this).safeTransferETH(price);` as follows.

```
    function _handleSignal(uint256 credId_, uint256 amount_, bool isSignal, address
    sender_) internal whenNotPaused {

        if (isSignal) {
            cred.currentSupply += amount_;
-            address(this).safeTransferETH(price);

}
```

```
    function _executeBatchTrade(
        uint256[] calldata credIds_,
        uint256[] calldata amounts_,
        address trader,
        uint256[] memory prices,
        uint256[] memory protocolFees,
        uint256[] memory creatorFees,
        bool isSignal
    )
        internal
    {
        ...
            if (isSignal) {
                creds[credId].currentSupply += amount;
-                address(this).safeTransferETH(prices[i]);
    }
```

Client: Fixed as per recommendation in https://github.com/ZaK3939/phi-protocol/pull/43/commits/5ff888ba97b5786d02fbeca311bae6088e5a25f5 and https://github.com/ZaK3939/phi-protocol/pull/43/commits/4120b1cb1f45323990501e786f259522f907f340.

Renascence: Issue has been resolved as per recommendation.

[I-3] `nonReentrant` modifier is not initialized

Context: Cred.sol#L100 ContributeRewards.sol#L45

Description: The contracts `Cred.sol` and `ContributeRewards.sol` have the `nonReentrant` modifier in their code, but do not set `locked` variable to `1` when created. As a result, functions that use this modifier will always fail bacause of the `if (locked != 1) revert Reentrancy();` check.

Recommendation: It is recommended to set `locked = 1` on creation and use `nonReentrant` modifier in all functions that transfer ETH to external addresses.

Client: Fixed in https://github.com/ZaK3939/phi-protocol/pull/43/commits/b530b4148f44fd17f7ac386c808f60c23e185b6a

Renascence: Verified, `nonReentrant` modifier is initialized properly.

[I-4] Duplicate `credType` check

Context: Cred.sol#L227 Cred.sol#L518

Description: The `credType` check is already performed in the `createCred` function:

```
    function _createCredInternal(
        address creator_,
        string memory credURL_,
        string memory credType_,
        string memory verificationType_,
        address bondingCurve_,
        uint16 signalRoyalty_,
        uint16 unSignalRoyalty_
    )
        internal
        whenNotPaused
        returns (uint256)
    {
»       if (!credType_.eq("BASIC") && !credType_.eq("ADVANCED")) {
            revert InvalidCredType();
        }
```

Recommendation: Remove duplicate check from `_createCredInternal`.

Client: Fixed in https://github.com/ZaK3939/phi-protocol/pull/43/commits/ae0c39813a804b576a90d676e014679fbbd3dc32

Renascence: Verified, duplicate check is removed.