Phi
Security Review

AUGUST, 2024

www.chaindefenders.xyz
https://x.com/ChDefendersEth

Lead Auditors

PeterSR

0x539.eth

Table of Contents

Protocol Summary

Phi Protocol is an open credentialing protocol to help users form, visualize, showcase their onchain identity. It incentivizes individuals to index blockchain transaction data as onchain credential blocks, curate them, host the verification process, and mint onchain credential contents.

Disclaimer

The ChainDefenders team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

Likelihood/Impact	High	Medium	Low
High	H	H/M	M
Medium	H/M	M	M/L
Low	M	M/L	L

Audit Details

Scope

Id	Files in scope
1	Cred.sol
2	PhiFactory.sol
3	Claimable.sol
4	CreatorRoyaltiesControl.sol
5	RewardControl.sol
6	PhiNFT1155.sol
7	BondingCurve.sol
8	CuratorRewardsDistributor.sol
9	PhiRewards.sol

Roles

Id	Roles
1	Owner
2	UserGroups

Executive Summary

Issues found

Severity	Count	Description
High	2	Critical vulnerabilities
Medium	1	Significant risks
Low	0	Minor issues with low impact
Informational	0	Best practices or suggestions
Gas	0	Optimization opportunities

Findings

High

[HIGH-01] Changes to Token Settings Allow Artists to Alter Critical Features

Vulnerability Details

The `updateArtSettings` function in the PhiFactory contract allows artists to modify several key settings of their art at any time. These settings include the URI link, royalty fee and soulBounded (non-transferable) feature.

```
    function updateArtSettings(
        uint256 artId_,
        string memory url_,
        address receiver_,
        uint256 maxSupply_,
        uint256 mintFee_,
        uint256 startTime_,
        uint256 endTime_,
        bool soulBounded_,
        IPhiNFT1155Ownable.RoyaltyConfiguration memory configuration
    )
        external
        onlyArtCreator(artId_)
    {
        ...
        art.receiver = receiver_;
        art.maxSupply = maxSupply_;
        art.mintFee = mintFee_;
        art.startTime = startTime_;
        art.endTime = endTime_;
        art.soulBounded = soulBounded_;
        art.uri = url_;

        uint256 tokenId = IPhiNFT1155Ownable(art.artAddress).
   getTokenIdFromFactoryArtId(artId_);
        IPhiNFT1155Ownable(art.artAddress).updateRoyalties(tokenId,
   configuration);
        emit ArtUpdated(artId_, url_, receiver_, maxSupply_, mintFee_,
    startTime_, endTime_, soulBounded_);
    }
```

The problem is that there are no restrictions or limitations on how these settings can be changed after the art has been created and minted. This flexibility allows artists to alter critical functionalities in ways that could negatively impact existing holders:

1. Changing Soulbound Status: Artists can toggle the soulBounded status to lock transfers, effectively trapping users who purchased NFTs under the assumption they were transferable.
2. Modifying Royalties: Artists can set royalty fees to extremely high values after initial sales, forcing holders to pay unexpected fees upon resale.
3. Updating URLs: Artists can change the linkURI, potentially misleading users or affecting the NFT's perceived value by altering the associated content. In the worst case, the URL could be changed to a malicious link, posing security risks to users who interact with it.

These changes can be made at any time, without prior notice to holders, leaving users vulnerable to unfavourable adjustments.

Impact

Allowing unrestricted changes to critical token settings poses a significant risk to the stability and trustworthiness of the NFTs. Users who have already minted or purchased NFTs could be adversely affected by changes they did not agree to, such as increased fees or transfer restrictions.

Tools Used

Manual review

Recommendation

Implement limits on how and when critical settings can be changed, such as capping royalty rates. This would help protect users while maintaining some flexibility for artists.

[HIGH-02] Signature replay in `createArt` allows to impersonate artist and steal royalties

Vulnerability details

Impact

Loss of funds: anyone can frontrun the `createArt` transaction, reusing the original signature but supplying their own config. As a result the artist, the royalties recipient, as well the the royalty BPS can be set arbitrarily, leading to stealing the royalties from the artist, and achieving other impacts.

Summary

Function PhiFactory::createArt() doesn't limit the signature to either the specific submitter, nor does it include into the signed data the `CreateConfig` parameters, which in particular include the `artist`, the royalties `receiver`, as well as other parameters. Other impacts are possible but here is one specific scenario:

1. The legitimate party creates a valid signature, config, and submits the `createArt` transaction

2. An attacker observes createArt transaction in the mempool, and frontruns it, reusing the signature, but with their own config where they are the royalties recipient
3. Attacker's transaction succeeds, setting them as the royalties recipient
4. The original transaction gets executed, and also succeeds, because createERC1155Internal succeeds both when a new PhiNFT1155 contract is created, as well as when it exists already
5. The legitimate user doesn't notice the difference: both ArtContractCreated and NewArtCreated events are emitted correctly (only NewArtCreated is emitted twice).

As a result, the attacker gets all rewards sent to the PhiRewards contract from PhiNFT1155 when the legitimate party claims an NFT token (see PhiNFT1155::claimFromFa

Other possible impacts (a non-exclusive list):

- The attacker sets themselves as an artist, and gets the possibility to call the PhiNFT1155::updateRoyalties() function, thus setting the royaltyBPS to arbitrary value.
- The attacker sets themselves as an artist, and gets the possibility to call the PhiFactory::updateArtSettings() function, thus setting the parameters to arbitrary values, e.g. maxSupply, endTime, or mintFee.

Proof of Concept

Drop this test to PhiFactory.t.sol and execute via forge test --match-test Kuprum

```
function testKuprum_ImpersonateArtist() public {
    // The owner prepares the signature, the config,
    //  and submits the `createArt` transaction
    string memory artIdURL = "sample-art-id";
    bytes memory credData = abi.encode(1, owner, "SIGNATURE", 31_337,
   bytes32(0));
    bytes memory signCreateData = abi.encode(expiresIn, artIdURL,
   credData);
    bytes32 createMsgHash = keccak256(signCreateData);
    bytes32 createDigest = ECDSA.toEthSignedMessageHash(createMsgHash)
   ;
    (uint8 cv, bytes32 cr, bytes32 cs) = vm.sign(claimSignerPrivateKey
   , createDigest);
    if (cv ≠ 27) cs = cs | bytes32(uint256(1) << 255);
    IPhiFactory.CreateConfig memory config =
        IPhiFactory.CreateConfig(artCreator, receiver, END_TIME, START_TIME, MAX_SUPPLY, MINT_FEE, false);

    // user1 observes `createArt` transaction in the mempool, and frontruns it,
    // reusing the signature, but with their own config where user1 is the receiver
    vm.deal(user1, 1 ether);
    vm.startPrank(user1);
    IPhiFactory.CreateConfig memory user1Config =
        IPhiFactory.CreateConfig(artCreator, user1, END_TIME, START_TIME, MAX_SUPPLY, MINT_FEE, false);
    phiFactory.createArt{ value: NFT_ART_CREATE_FEE }(signCreateData, abi.encodePacked(cr, cs), user1Config);

    // Owner's `createArt` succeeds; there is also no difference in the `ArtContractCreated` event
    vm.startPrank(owner);
    phiFactory.createArt{ value: NFT_ART_CREATE_FEE }(signCreateData, abi.encodePacked(cr, cs), config);

    // Verify that user1 is now the royalties recepient
    uint256 artIdNum = 1;
    IPhiFactory.ArtData memory updatedArt = phiFactory.artData(artIdNum);
    assertEq(updatedArt.receiver, user1, "user1 should be the royalties recepient");
}
```

Tools Used

Manual review

Recommended Mitigation Steps

We recommend the following steps:

- In PhiFactory::createArt():
  - include into the signed data the `CreateConfig` parameters
  - limit transaction execution to a specific submitter
- In PhiFactory::createERC1155Internal() fail the transaction if the contract already exists.

Medium

[MEDIUM-01] Cred creator could stuck funds

Vulnerability details

Impact

Update of cred can be triggered by the cred creator. If there are already issued shares for users, and the cred creator decides to update the `sellShareRoyalty_` this will lead users to pay up to 50% fee depending on the value. And users will be in situation that to withdraw their funds, they need to pay 50% to the cred creator and additionally some fee to the protocol.

Proof of Concept

We have the following situation:

1. 500 issued shares and the current sell fee is 0.05%
2. And the cred creator decides to increase the fee to 50%
3. If any of the users wants to withdraw funds, they need to pay 50% to the creator and also protocol fee

For example let's imagine that the current number of shares issued is 500 of max 999 and one of the users is owner of 5 shares. If we add this test to `Cred.t.sol`

```
function test_value() public {
        uint256 x = bondingCurve.getSellPrice(500, 5);
        assertEq(153_019_801_980_198_020, x);
    }
```

we will see that the sell price is around 1.53e17 which in the current price is equal to 420$, this means that the 50% or 210$ will be paid to the cred creator as fee, which will lead to significant lose of funds for share holders. If the number of shares that are issued the lose is growing.

Tools Used

Manual review

Recommended Mitigation Steps

The protocol should implement a mechanism where this type of updates, will be applied after some period. I will explain my idea in the following lines:

1. If cred creator wants to increase the `sellShareRoyalty_`, it should first create a request for updating which can be applied after 2 days for exapmles.
2. In these two days users who are not okay with the new value of `sellShareRoyalty_` can withdraw their funds from the protocol.

My proposal is to have two functions

1. `updateSellShareRoyaltyRequest` which will create this request and this request can be approved after some period.
2. `approveUpdateSellShareRoyaltyRequest` if the period already passed the cred creator can apply the change.