SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for:	RadicalxChange
Prepared by:	Sherlock
Lead Security Expert:	zzykxx
Dates Audited:	March 19 - March 22, 2024
Prepared on:	May 12, 2024

Introduction

A community of activists, artists, entrepreneurs, and scholars committed to using mechanism design to inspire radical social change.

Scope

Repository: RadicalxChange/pco-art

Branch: main

Commit: 4acd6b06840028ba616b6200439ce0d6aa1e6276

For the detailed scope, see the contest details.

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
2	1

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

zzykxx
Al-Qa-qa
sammy
thisvishalsingh
0xKartikgiri00
tedox
ke1caM
thank_you
Aamirusmani1552

Tricko
cocacola
AMOW
14si2o_Flint
SovaSlava
ge6a
ethernal
zraxx
0xPwned
0xbrivan
FSchmoede
pynschon
0xShitgem
merlin
valentin2304
jah
koreanspicygarlic
CarlosAlbaWork
Krace
0xboriskataa
Marcologonz
DenTonylifer
ljj
FassiSecurity
dipp
404666
Atharv
pseudoArtist
turvec
neocrao
aycozynfada
theOwl
0rpse
psb01
DMoore
theFirstElder
Dots
jasonxiale
cawfree
mrBmbastic
cats
Tendency
cu5t0mPe0
kuprum
fugazzi
neon2835
AgileJune
sandy
FastTiger
offside0011

Issue H-1: Highest bidder can withdraw his collateral due to a missing check in _cancelAllBids

Source: https://github.com/sherlock-audit/2024-02-radicalxchange-judging/issues/14

Found by

0rpse, 0xKartikgiri00, 0xPwned, 0xShitgem, 0xboriskataa, 0xbrivan, 14si2o_Flint, 404666, AMOW, Aamirusmani1552, AgileJune, Al-Qa-qa, Atharv, CarlosAlbaWork, DMoore, DenTonylifer, Dots, FSchmoede, FassiSecurity, FastTiger, Krace, Marcologonz, SovaSlava, Tendency, Tricko, aycozynfada, cats, cawfree, cocacola, cu5t0mPe0, dipp, ethernal, fugazzi, ge6a, jah, jasonxiale, ke1caM, koreanspicygarlic, kuprum, ljj, merlin, mrBmbastic, neocrao, neon2835, offside0011, psb01, pseudoArtist, pynschon, sammy, sandy, tedox, thank_you, theFirstElder, theOwl, thisvishalsingh, turvec, valentin2304, zraxx, zzykxx

Summary

A bidder with the highest bid cannot cancel his bid since this would break the auction. A check to ensure this was implemented in `_cancelBid`.

However, this check was not implemented in `_cancelAllBids`, allowing the highest bidder to withdraw his collateral and win the auction for free.

Vulnerability Detail

The highest bidder should not be able to cancel his bid, since this would break the entire auction mechanism.

In `_cancelBid` we can find a require check that ensures this:

```
require(
    bidder != l.highestBids[tokenId][round].bidder,
    'EnglishPeriodicAuction: Cannot cancel bid if highest bidder'
);
```

Yet in `_cancelAllBids`, this check was not implemented.

```
 * @notice Cancel bids for all rounds
 */
function _cancelAllBids(uint256 tokenId, address bidder) internal {
    EnglishPeriodicAuctionStorage.Layout
        storage l = EnglishPeriodicAuctionStorage.layout();
```

```
    uint256 currentAuctionRound = l.currentAuctionRound[tokenId];

    for (uint256 i = 0; i <= currentAuctionRound; i++) {
        Bid storage bid = l.bids[tokenId][i][bidder];

        if (bid.collateralAmount > 0) {
            // Make collateral available to withdraw
            l.availableCollateral[bidder] += bid.collateralAmount;

            // Reset collateral and bid
            bid.collateralAmount = 0;
            bid.bidAmount = 0;
        }
    }
}
```

Example: User Bob bids 10 eth and takes the highest bidder spot. Bob calls `cancelAllBidsAndWithdrawCollateral`.

The `_cancelAllBids` function is called and this makes all the collateral from all his bids from every round available to Bob. This includes the current round <= and does not check if Bob is the current highest bidder. Nor is `l.highestBids[tokenId][round].bidder` reset, so the system still has Bob as the highest bidder.

Then `_withdrawCollateral` is automatically called and Bob receives his 10 eth back.

The auction ends. If Bob is still the highest bidder, he wins the auction and his bidAmount of 10 eth is added to the availableCollateral of the oldBidder.

If there currently is more than 10 eth in the contract (ongoing auctions, bids that have not withdrawn), then the oldBidder can withdraw 10 eth. But this means that in the future a withdraw will fail due to this missing 10 eth.

Impact

A malicious user can win an auction for free.

Additionally, either the oldBidder or some other user in the future will suffer the loss.

If this is repeated multiple times, it will drain the contract balance and all users will lose their locked collateral.

Code Snippet

https://github.com/sherlock-audit/2024-02-radicalxchange/blob/main/pco-art/contracts/auction/EnglishPeriodicAuctionInternal.sol#L416-L436

https://github.com/sherlock-audit/2024-02-radicalxchange/blob/main/pco-art/contracts/auction/EnglishPeriodicAuctionInternal.sol#L380-L413

https://github.com/sherlock-audit/2024-02-radicalxchange/blob/main/pco-art/contracts/auction/EnglishPeriodicAuctionInternal.sol#L468-L536

Tool used

Manual Review

Recommendation

Implement the require check from _cancelBid to _cancelAllBids.

Discussion

sherlock-admin3

The protocol team fixed this issue in the following PRs/commits:
https://github.com/RadicalxChange/pco-art/pull/9

zzykxx

The highest bidder can't cancel his bid on an active auction round anymore while still being able to cancel his bids on previous rounds.

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-1: Auction fails if the 'Honorarium Rate' is 0%

Source: https://github.com/sherlock-audit/2024-02-radicalxchange-judging/issues/31

Found by

Al-Qa-qa, sammy

Summary

The Honorarium Rate is the required percentage of a winning Auction Pitch bid that the Steward makes to the Creator Circle at the beginning of each Stewardship Cycle.

$$ Winning Bid * Honorarium Rate = Periodic Honorarium $$

To mimic the dynamics of private ownership, the *Creator Circle* may choose a 0% *Honorarium Rate*. However, doing so breaks the functionality of the protocol.

Vulnerability Detail

To place a bid, a user must call the placeBid function in `EnglishPeriodicAuctionFacet.sol` and deposit collateral(`collateralAmount`) equal to `bidAmount + feeAmount`. The `feeAmount` here represents the *Honorarium Rate* mentioned above. The `placeBid` function calls the _placeBid internal function in `EnglishPeriodicAuctionInternal.sol` which calculates the `totalCollateralAmount` as follows :

```
uint256 totalCollateralAmount = bid.collateralAmount + collateralAmount;
```

Here, `bid.collateralAmount` is the cumulative collateral deposited by the bidder in previous bids during the current auction round(i.e, zero if no bids were placed), and `collateralAmount` is the collateral to be deposited to place the bid. However the `_placeBid` function requires that `totalCollateralAmount` is strictly greater than `bidAmount` if the bidder is not the current owner of the *Stewardship License*. This check fails when the `feeAmount` is zero and this causes a *Denial of Service* to users trying to place a bid. Even if the users try to bypass this by depositing a value slightly larger than `bidAmount`, the _checkBidAmount function would still revert with `'Incorrect bid amount'`

POC

The following test demonstrates the above-mentioned scenario :

```
describe('exploit', function () {
   it('POC', async function () {
     // Auction start: Now + 100
     // Auction end: Now + 400
     const instance = await getInstance({
       auctionLengthSeconds: 300,
       initialPeriodStartTime: (await time.latest()) + 100,
       licensePeriod: 1000,
     });
     const licenseMock = await ethers.getContractAt(
       'NativeStewardLicenseMock',
       instance.address,
     );

     // Mint token manually
     const steward = bidder2.address;
     await licenseMock.mintToken(steward, 0);

     // Start auction
     await time.increase(300);

     const bidAmount = ethers.utils.parseEther('1.0');
     const feeAmount = await instance.calculateFeeFromBid(bidAmount);
     const collateralAmount = feeAmount.add(bidAmount);

     // Reverts when a user tries to place a bid
     await expect( instance
       .connect(bidder1)
       .placeBid(0, bidAmount, { value: collateralAmount
↪  })).to.be.revertedWith('EnglishPeriodicAuction: Collateral must be greater
↪  than current bid');



     const extraAmt = ethers.utils.parseEther('0.1');
     const collateralAmount1 = feeAmount.add(bidAmount).add(extraAmt);

     // Also reverts when the user tries to deposit collateral slighty greater
↪  than bid amount
     await expect( instance
       .connect(bidder1)
       .placeBid(0, bidAmount, { value: collateralAmount1
↪  })).to.be.revertedWith('EnglishPeriodicAuction: Incorrect bid amount');

     // Only accepts a bid from the current steward
```

```
    await expect( instance
      .connect(bidder2)
      .placeBid(0, bidAmount, { value: 0 })).to.not.be.reverted;

  });
});
```

To run the test, copy the code above to `EnglishPeriodicAuction.ts` and alter L#68 as follows :

```
-          [await owner.getAddress(), licensePeriod, 1, 10],
+          [await owner.getAddress(), licensePeriod, 0, 10],
```

Run `yarn run hardhat test --grep 'POC'`

Impact

The protocol becomes dysfunctional in such a scenario as users as DOS'd from placing a bid.

Code Snippet

Tool used

Manual Review Hardhat

Recommendation

Alter EnglishPeriodicAuctionInternal.sol::L#330 as follows :

```
- totalCollateralAmount > bidAmount,
+ totalCollateralAmount >= bidAmount,
```

Discussion

sammy-tm

Escalate It states in the docs that the Honorarium rate may be 0 to mimic private ownership. However, as clearly described in the issue above, this functionality fails. https://pco-art-docs.vercel.app/for-artists/pco-settings ***"Lower Honorarium Rates will tend to lead to higher nominal Auction Prices: the capital commitment required of the Steward each cycle is lower given a valuation. A 0% Honorarium Rate effectively would mimic the dynamics of private ownership (with a periodic auction in which the Steward could set an arbitrarily high reserve price). The***

theoretically optimal rate from a Periodic Honorarium maximization perspective is equal to the probability that a new Steward emerges who values the work more than the current Steward (e.g. 1 in every 10 periods implies a 10% optimal rate), but that, of course, might not be the priority."

sherlock-admin2

> Escalate It states in the docs that the Honorarium rate may be 0 to mimic private ownership. However, as clearly described in the issue above, this functionality fails.
> https://pco-art-docs.vercel.app/for-artists/pco-settings ***"Lower Honorarium Rates will tend to lead to higher nominal Auction Prices: the capital commitment required of the Steward each cycle is lower given a valuation. A 0% Honorarium Rate effectively would mimic the dynamics of private ownership (with a periodic auction in which the Steward could set an arbitrarily high reserve price). The theoretically optimal rate from a Periodic Honorarium maximization perspective is equal to the probability that a new Steward emerges who values the work more than the current Steward (e.g. 1 in every 10 periods implies a 10% optimal rate), but that, of course, might not be the priority."***

The escalation could not be created because you are not exceeding the escalation threshold.

You can view the required number of additional valid issues/judging contest payouts in your Profile page, in the Sherlock webapp.

cawfree

Escalate

The amazing supporting documentation for RadicalxChange states that one of the intended logical underpinnings of the fee model should be to mimic private ownership when operating using a 0% honorarium: .

However, the codebase strictly does not permit this due to emergent strict equality checks, which only arise later when actually trying to place a bid. The collection deployer finds out too late.

The counterargument to this finding is it just seems low severity given that this can be resolved by doing something like redeploy to a collection with `1 wei` (i.e. a frontend fix), or even wondering how often communities are going to do this in practicality.

It is just clear that this implementation quirk works directly against the advertised functionality (a literal feature), and that should probably play into the severity. People do like free mints after all.

sherlock-admin2

> Escalate
>
> The amazing supporting documentation for RadicalxChange states that one of the intended logical underpinnings of the fee model should be to mimic private ownership when operating using a 0% honorarium: .
>
> However, the codebase strictly does not permit this due to emergent strict equality checks, which only arise later when actually trying to place a bid. The collection deployer finds out too late.
>
> The counterargument to this finding is it just seems low severity given that this can be resolved by doing something like redeploy to a collection with `1 wei` (i.e. a frontend fix), or even wondering how often communities are going to do this in practicality.
>
> It is just clear that this implementation quirk works directly against the advertised functionality (a literal feature), and that should probably play into the severity. People do like free mints after all.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

Al-Qa-qa

Besides what @cawfree said, I want to add that if the Artist (Collection Admin), deployed the Collection as immutable (i.e. without an owner). No one will be able to change the fees to just make it `1 wei` (And this point I mentioned in my report `107`). This will make all the Auction processes/NFT Collection broken and insolvable.

zzykxx

This is valid. Setting a fee of `0` is a legitimate and expected input and the core functionality of the protocol (auctions) breaks when this is the case.

Hash01011122

It appears valid finding to me with duplicates #12 and #107 @gravenp @St4rgarden would you like to add something here??

sammy-tm

I would also like to urge the lead judge to re-evaluate the severity of this finding. "Breaking core functionality" might be considered High in some cases.

gravenp

The 0% Honorarium example is the theoretical limit, but in practice, it doesn't make sense to configure it that way (the PCO system ends up being a lot of complex

code for the current steward to basically hold the art as long as they want). I agree that as implemented, we allowed it, and it breaks things. Our fix will not allow the artist/admin to configure a 0% honorarium as this specific configuration isn't core to our intended functionality.

Hash01011122

I think this issue is open ended double edge sword where sponsors claiming this cannot be rectified as this is design choice but in my opinion, it suffices medium severity as this breaks protocol's functionality, let me know your thoughts @zzykxx, @gravenp, @Czar102

gravenp

I'm fine with whatever the judges decide. Ultimately, our design choice is not to allow 0% honorarium, so the intended functionality won't include supporting this scenario.

zzykxx

> I think this issue is open ended double edge sword where sponsors claiming this cannot be rectified as this is design choice but in my opinion, it suffices medium severity as this breaks protocol's functionality, let me know your thoughts @zzykxx, @gravenp, @Czar102

In my opinion this should be a valid issue unless it was mentioned somewhere public that the 0% honorarium is not allowed/intended.

Hash01011122

As mentioned by @zzykxx, this is a valid finding with duplicates #107 and #12

Czar102

> A 0% Honorarium Rate effectively **would mimic** the dynamics of private ownership (with a periodic auction in which the Steward could set an arbitrarily high reserve price).

The usage of "would mimic" instead of "mimics" implies that it's either know not to be possible to use such a configuration, or it is undesired for such a configuration to be used.

> The 0% Honorarium example is the theoretical limit, but in practice, it doesn't make sense to configure it that way (the PCO system ends up being a lot of complex code for the current steward to basically hold the art as long as they want).

It seems that using a 0% Honorarium Rate doesn't make sense, hence it doesn't really limit the intended functionality of the implemented codebase.

Am I mistaken somewhere?

sammy-tm

You're correct; however, it doesn't expressly state that the artists are not supposed to do this. Neither does it prevent them from doing so. A naive artist may read the documentation and want to mimic an IRL auction/ private ownership for their art, so they would set the fees to 0%. But doing so visibly breaks the protocol. This needs to be handled either during configuration (as @gravenp mentioned)

> I agree that as implemented, we allowed it, and it breaks things. Our fix will not allow the artist/admin to configure a 0% honorarium as this specific configuration isn't core to our intended functionality.

or mitigated using the recommendation in the Issue.

Al-Qa-qa

The documentation did not deny setting 0 fees totally, it expresses that setting 0% is not typically advisable (Or What means this).

> The 0% Honorarium example is the theoretical limit

This sentence does not exist in the docs, The docs did not explain that 0% is prevented or other, it explained that `it's like the Steward owns the thing they're managing as if it were their own property. They get all the benefits without having to pay anything for it`

So according to the documentation which was the only resource for that issue in the Auditing period, setting 0% Honorarium was not prevented totally, nor said by the Devs in the README, nor that edge case was handled in the code.

I hope the issue gets reviewed according to the information that was with us (Auditors) in the Auditing period, The points that the sponsor said after the Auditing finished (In Escalation) should not be taken as proof to either validate or invalidate issues.

And if the likelihood of the issue is LOW, its impact is so HIGH. And this is illustrated in my report, and escalation engagements.

Hash01011122

This issue, along with its duplicates #12 and #107, should be deemed valid. It highlights the complication arising from a 0% honorarium rate, contradicting the project documentation which states this as a theoretical limit not reflected in the codebase. Moreover, the documentation does not clearly advise against its use. What are your thoughts, @Czar102?

Evert0x

I believe this issue should be valid and the main issue (12) should be assigned Medium severity.

gravenp

It doesn't seem that #12 is the right parent issue to me. #42 and #107 appear to be the proper issues to connect here, @Hash01011122.

sammy-tm

#42 is about setting the fee denominator to 0, which is a completely different issue and is invalid.

Al-Qa-qa

I read issue 12, and the issue is about different Root causes and Impact.

Could you give another look at it @Hash01011122 , and determine your final thought about this issue, whether it is a Dup of 31 and 107 or not?

sammy-tm

Went through #12, agree with @Al-Qa-qa that the concern highlighted in the issue is completely different from #31 and #107. I think the judge has incorrectly grouped these together because the mitigation step is the same.

If you read carefully, #12 is a wrong interpretation of the codebase and focuses on user input validation (invalid according to sherlock rules)

Quoting Issue #12 :

> The root of the issue is the expectation that the fee should be paid on top of the bidAmount, which goes fully against documentation and poses many problems. Since the protocol has the stated requirement of 100% collateralisation, many valid bids will revert with the confusing message that you need overcollateralisation. ***@>"The fee is a percentage of the bid, so there is no logical reason why the fee should be paid on top of the bid."*** Even if users are willing to overcollateralise, for every bid they need to take into account both collateral paid and fee paid and recalculate both perfectly to be within rounding error. This is prone to error. In a heated auction with many competing bidders trying to take the price close to a deadline, the complexity demanded to perfectly calculate every bid is not realistic. This will lead to many valid bids being reverted due to a wei-sized deviation and the auction will close at a lower price than the highest valid bid submitted. This constitutes a loss to the creator's circle and the current holder of the license.

It talks about how it is "confusing" to the user to pay the fee on top of the bid and argues against the intended implementation of the codebase. It is known that the codebase is implemented in such a way that the user is **expected** to pay the fee on top of the bid. The watson has simply misinterpreted the codebase and this is further illustrated by the example scenario described by the watson in the issue.

Issue #12 has nothing to do with the `Honorarium Rate` being set to 0.

I urge the judges @Evert0x @Hash01011122 and the sponsor @gravenp to go through #12 separately and remove duplication from #31 and #107

Hash01011122

@sammy-tm thanks for providing a detailed explanation for every issue of this family, after inspecting issues thoroughly I can say #31 is valid finding with #107 as its dup @Evert0x

Evert0x

Agree with the proposed outcome here. Thanks @sammy-tm

Planning to accept escalation, make #31 a Medium issue with #107 as a duplicate.

Evert0x

Result: Medium Has Duplicates

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- cawfree: accepted

sherlock-admin3

The protocol team fixed this issue in the following PRs/commits: https://github.com/RadicalxChange/pco-art/pull/10

zzykxx

The issue has been fixed as recommended. Plus, if `feeDenominator` is `0` the returned fee is now `0`.

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-2: Currently auctioned NFTs can be transferred to a different address in a specific edge case

Source: https://github.com/sherlock-audit/2024-02-radicalxchange-judging/issues/33

Found by

zzykxx

Summary

Currently auctioned NFTs can be transferred to a different address in a specific edge case, leading to theft of funds.

Vulnerability Detail

The protocol assumes that an NFT cannot change owner while it's being auctioned, this is generally the case but there is an exception, an NFT can change owner via mintToken() while an auction is ongoing when all the following conditions apply:

1. An NFT is added to the collection without being minted (ie. `to` set to `address(0)`).
2. The NFT is added to the collection with the parameter `tokenInitialPeriodStartTime[]` set to a timestamp lower than `l.initialPeriodStartTime` but bigger than `0`(ie. `0 < tokenInitialPeriodStartTime[] < l.initialPeriodStartTime`).
3. The current `block.timestamp` is in-between `tokenInitialPeriodStartTime[]` and `l.initialPeriodStartTime`.

A malicious `initialBidder` can take advantage of this by:

1. Bidding on the new added NFT via placeBid().
2. Calling mintToken() to transfer the NFT to a different address he controls.
3. Closing the auction via closeAuction()

At point `3.`, because the NFT owner changed, the winning bidder (ie. `initialBidder`) is not the current NFT owner anymore. This will trigger the following line of code:

```
l.availableCollateral[oldBidder] +=
↪   l.highestBids[tokenId][currentAuctionRound].bidAmount;
```

Which increases the `availableCollateral` of the `oldBidder` (ie. the address that owns the NFT after point `2.`) by `bidAmount` of the highest bid. But because at the moment the highest bid was placed `initialBidder` was also the NFT owner, he only needed to transfer the `ETH` fee to the protocol instead of the whole bid amount.

The `initialBidder` is now able to extract ETH from the protocol via the address used in point `2.` by calling withdrawCollateral() while also retaining the NFT license.

Impact

Malicious initial bidder can potentially steal ETH from the protocol in an edge case. If the `ADD_TOKEN_TO_COLLECTION_ROLE` is also malicious, it's possible to drain the protocol.

Code Snippet

Tool used

Manual Review

Recommendation

Don't allow `tokenInitialPeriodStartTime[]` to be set at a timestamp before`1.initialPeriodStartTime`.

Discussion

zzykxx

Escalate

Not a duplicate of #9. This issue describes a correct edge case in which a currently auctioned NFT can be transferred.

sherlock-admin2

> Escalate
>
> Not a duplicate of #9. This issue describes a correct edge case in which a currently auctioned NFT can be transferred.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

Hash01011122

Even I considered it unique medium as it was edge case but sponsors viewed it as duplicate of #9 as it has same root cause @gravenp anything which you would like to add?

St4rgarden

Not a duplicate because #9 is a non-issue.

gravenp

@Hash01011122 after @St4rgarden took another look at #33 and #9, we determined that these should NOT be duplicates. I was mistaken. Sorry. We've marked #9 invalid and @St4rgarden is working on a hardhat test to reproduce this one.

Czar102

Planning to consider this a separate issue.

@zzykxx why do you think High severity is appropriate? I'd like to know more about the likelihood of this exploit.

zzykxx

> Planning to consider this a separate issue.
>
> @zzykxx why do you think High severity is appropriate? I'd like to know more about the likelihood of this exploit.

I submitted as high severity because a combination of malicious `initialBidder` and `ADD_TOKEN_TO_COLLECTION_ROLE` can steal funds currently in the contract.

The sponsor is stating somewhere else that `initialBidder` is a trusted role, which would make the issue invalid.

However, the fact that the `initalBidder` or `ADD_TOKEN_TO_COLLECTION_ROLE` is a trusted role was not mentioned anywhere during the time of the audit. One argument on why `initialBidder` should be considered trusted is that it has the power to mint NFTs (and to mint them to himself), and this power is given to them by the owner which is trusted according to the README. I'm not sure this implies that the `initialBidder` (and `ADD_TOKEN_TO_COLLECTION_ROLE`) are trusted to not steal funds in the contract. To me having the power of minting/wrapping NFTs (which will then be auctioned) and the power to steal funds in the contract are two different levels of trust.

This being said, I'll leave this to you and the judges as the decision on the severity mainly revolves around judging technicalities. Feel free to tag me if you need more information.

gravenp

We agree that this is valid. I'm not steeped in the judging technicalities on severity either.

To attempt to clarify a bit, we're not trying to imply elsewhere that the address in `initialBidder` is trusted (just that this field is *by default* set to the artist, who is trusted, at initialization/the first auction). The `ADD_TOKEN_TO_COLLECTION_ROLE` and other admin roles are also trusted with their defined scope, but stealing funds auction funds wouldn't be in that role scope. I don't know where that nets out on likelihood so am ok ceding that determination to the judges and their experience with the contest rules.

Czar102

This issue is certainly valid, and I would place it on the borderline Med/High, but I think Medium is appropriate given that not everyone is able to exploit this vulnerability, only few whitelisted parties.

Evert0x

Planning to remove duplication state and assign Medium severity

Evert0x

Result: Medium Unique

sherlock-admin3

Escalations have been resolved successfully!

Escalation status:

- zzykxx: accepted

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/RadicalxChange/pco-art/pull/11

zzykxx

The issue has been fixed by preventing the auction start time of a newly added NFT to be in the past.

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.