Reality Cards (round 2) Findings & Analysis Report

2021-09-21

Table of contents

Overview

About C4

Code 432n4 (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 code contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the code contest outlined in this document, C4 conducted an analysis of the Reality Cards smart contract system written in Solidity. The code contest took place between August 18—August 25 2021.

Wardens

11 Wardens contributed reports to the Reality Cards (round 2) code contest:

1. gpersoon
2. tensors
3. cmichel
4. leastwood
5. Jmukesh
6. hickuphh3
7. shw
8. 0xImpostor
9. 0xsanson
10. qedk
11. PierrickGT

This contest was judged by 0xean.

Final report assembled by moneylegobatman and ninek.

Summary

The C4 analysis yielded an aggregated total of 30 unique vulnerabilities. All of the issues presented here are linked back to their original finding

Of these vulnerabilities, 2 received a risk rating in the category of HIGH severity, 3 received a risk rating in the category of MEDIUM severity, and 25 received a risk rating in the category of LOW severity.

C4 analysis also identified 11 non-critical recommendations and 9 gas optimizations.

Scope

The code under review can be found within the C4 Reality Cards (Round 2) code contest repository is comprised of 26 smart contracts written in the Solidity programming language and includes 7,770 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

High Risk Findings (2)

[H-01] `findNewOwner` edgecase

Submitted by gpersoon

In the function `findNewOwner` of `RCOrderbook`, as loop is done which included the check `_loopCounter < maxDeletions` Afterwards, a check is done for "(*loopCounter != maxDeletions)" to determine if the processing is finished. If loopCounter == maxDeletions then the conclusion is that it isn't finished yet.

However, there is the edgecase that the processing might just be finished at the same time as `_loopCounter == maxDeletions`.

You can see this the best if you assume `maxDeletions==1`, in that case it will never draw the conclusion it is finished. Of course having `maxDeletions==1` is very unlikely in practice.

```
549
550  function findNewOwner(uint256 _card, uint256 _timeOwnership
551 ...
552      // delete current owner
553      do {
554          _newPrice = _removeBidFromOrderbookIgnoreOwner( _hea
555          _loopCounter++;              // delete next bid if fo
556      } while (    treasury.foreclosureTimeUser( _head.next, _
557              _loopCounter < maxDeletions );
558
559      if (_loopCounter != maxDeletions) {   // the old owner i
560          _newOwner = ....
561          ...
562      } else {
563          // we hit the limit, save the old owner, we'll try a
564          ...
565      }
566 }
```

Recommend using a different way to determine that the processing is done. This could save some gas. Note: the additional check also costs gas, so you have to verify the end result.

Perhaps in `setDeletionLimit`, doublecheck that `_deletionLimit` > 1.

Splidge (Reality Cards) confirmed and disagreed with severity:

> oh wow, this is actually a really big problem. It's easier to see it if `maxDeletions` is 1 but it exists with any size of `maxDeletions`. Whenever we find a valid owner on the final iteration of the loop the if statement will simply check if it was the final loop. That valid owner is then assumed to be invalid and saved for the next transaction to try and find a new owner. When that next transaction happens the valid owner is immediately deleted and not given any ownership of the card at all. I think this just falls short of 3 (High risk) because I don't think it'd be possible for an attacker to engineer the situation to have a particular user deleted without ownership. But I believe this would count as 2 (Med risk) because the protocol "availability could be impacted" for the user that is deleted.

Splidge (Reality Cards) commented:

> I have since thought of an attack that could have used this and might raise it to 3 (High risk).

> Due to the difficultly of monitoring which cards you own all the time a valid strategy which some users employ is to bid high enough to scare off other users (usually bidding significantly beyond the 10% minimum increase). Suppose Alice employs this strategy by bidding $100 on a card that was previously only $10. Mal (our attacker) wishes to rent the card but wants to pay less than $100. Mal could use Sybil accounts to place `maxDeletions - 1` bids all for the minimum rental duration (only funding the accounts for the minimum duration). Mal would then need to wait for the minimum duration of all these bids to expire, `(maxDeletions - 1 ) * minimumRentalDuration` Once this has completed Mal can place a bid at $11, this will trigger a rent collection which will attempt to `findNewOwner`, Alice being the user that was found on the last iteration of the loop would be considered as invalid. There will not be a change of ownership or any events emitted about this until the next rent collection is triggered. This means that the UI would still consider Alice to be the owner of card (Mals' Sybil bids having had `LogRemoveFromOrderbook` and `LogUserForeclosed` events emitted) and other users might not consider trying to outbid this, whereas actually Mal is accruing time at a significantly cheaper rate.

> Thinking about it, this doesn't really even need Alice at all, Mal could have placed all the higher bids to simultaneously scare off other users while renting at a lower price.

> I think the fix is relatively simple, by checking if we found a valid user OR hit the deletion limit we can make it so that we don't skip any bids. This would then leave Alice (or Mal in the other version) correctly having to pay for the time at the higher price.

Oxean (judge) commented:

> upgrading based on sponsors analysis

Splidge (Reality Cards) patched:

> Fixed here

[H-02] `UberOwner` has too much power

Submitted by tensors

The Uber Owner has too much power within the system. This makes the protocol closer to a centralized prediction market whose rules are determined by the Uber Owner. See issue page for referenced code

The above functions can be used by the Uber Owner to completely change the functionality of the system. This goes well beyond simple setting new constants and fees, the Uber Owner can basically reprogram how the entire protocol works. Not to mention if the address falls into the wrong hands.

Recommend limiting the permission of the Uber Owner to something more manageable and trustable. If upgrades to underlying contracts are required they can be done through a proxy instead, in the standard way.

mcplums (Reality Cards) disputed:

> This is a subjective opinion- there is always going to be a compromise between decentralisation and the ability to respond to potential problems. The latter is especially important with a protocol that is so new.

> There is no correct answer here, but the current abilities of `uberOwner` were decided after a lot of thought and are in line with other DeFi protocols.

Splidge (Reality Cards) commented:

> I'd just like to add that we did recognize the power of the UberOwner which is why it is separated from the Owner specifically so that we can add additional security to it (in the form of a multisig) and so that we can relinquish this control at the appropriate time. This was covered in the readme. And also commented in the code.

Oxean (judge) commented:

> I think the warden(s) have a valid point here. This is an incredible amount of power for a single address to yield over the protocol, even if backed by a multi-sig.

> Is it no an option to 1) pause all activity, and unlock all funds allowing users to withdraw their own funds or 2) pause all activity besides withdraws and implement a time delay between that and the "rug pull" function being called.

> The readme also states

> ```
> Alternatively we may wish for this to be a multisig but the normal owner to not be, for convenience.
> ```

> Without a multisig, I believe this absolutely qualifies as a high severity issue as a compromise of a single end user address compromises the entire system, with a multisig it potentially lowers the severity down to a medium, but its still a risk that is worth highlighting in the system and for the sponsor to scrutinize if there are indeed other mitigation paths that could be taken.

Medium Risk Findings (3)

[M-01] Uninitialized Variable `marketWhitelist` in `RCTreasury.sol`

Submitted by leastwood, also found by 0xsanson, gpersoon, hickuphh3 and JMukesh

The variable, `marketWhitelist`, is never initialized in the contract `RCTreasury.sol`. As a result, the function `marketWhitelistCheck()` does not perform a proper check on whitelisted users for a restricted market. Additionally, the function will always return `true`, even if a market wishes to restrict its users to a specific role.

The initial state variable is defined in `RCTreasury.sol` **L75**.

The state variable `marketWhitelist` is accessed in the function `RCTreasury.marketWhitelistCheck()` at `RCTreasury.sol` **L269-L281**.

The function `RCTreasury.marketWhitelistCheck()` is called in `RCMarket.newRental()` at `RCMarket.sol` **L758-L761**. The comment indicates that there should be some ability to restrict certain markets to specific whitelists, however, there are no methods in `RCTreasury` that allow a market creator to enable this functionality.

Recommend ensuring this behavior is intended. If this is not the case, consider adding a function that enables a market creator to restrict their market to a specific role by whitelisting users.

Splidge (Reality Cards) confirmed and disagreed with severity:

> I think the severity could be double-checked on this one. It's a close one but I'd be tempted to put it under 1 (low risk) as a "Function incorrect to spec". Regardless, this will be fixed.

> Edit: I notice the duplicates were both marked as 1 (low risk).

0xean (judge) commented:

> based on " but the function of the protocol or its availability could be impacted" in the code4 docs, I am going to agree with warden and leave this as a 2. The function of the protocol is certainly impacted in a case where the whitelist if not working correctly.

Splidge (Reality Cards) patched:

> Fixed here

[M-02] Parameter updates not propagated

Submitted by gpersoon, also found by cmichel

There are several functions to update parameters. However these parameters are only updated on the top level and not propagated to the other contracts. This could lead to various unpredictable results. Examples are:

- `setNftHubAddress` of `RCFactory`
- `setOrderbookAddress` of `RCFactory`
- `setLeaderboardAddress` of `RCFactory`
- `setMinRental` of `RCTreasury`

```
586
587 function setNftHubAddress(IRCNftHubL2 _newAddress) external
588     require(address(_newAddress) != address(0), "Must set Ad
589     nfthub = _newAddress;
590 }
591
592 function setOrderbookAddress(IRCOrderbook _newOrderbook) ext
593     require( treasury.checkPermission(TREASURY, msgSender())
594     orderbook = _newOrderbook;
595 }
596
597 function setLeaderboardAddress(IRCLeaderboard _newLeaderboar
598     require( treasury.checkPermission(TREASURY, msgSender())
599     leaderboard = _newLeaderboard;
600 }
601
188
189 function setMinRental(uint256 _newDivisor) public override o
190     minRentalDayDivisor = _newDivisor;
191 }
```

Recommend implementing a way to notify the underlying contracts of the updates.

Splidge (Reality Cards) acknowledged:

> We have come to realise that it is very unlikely we will be able to change certain contracts once they are in-use, the exception being the market where a new reference could be deployed. In practice we do use `setNftHubAddress` shortly after deploying new contracts, this is so that we can continue to use an existing NFT hub that has already been put through Matic Mintable Asset mapping, but changing this while a market is active would cause problems. While we accept that changing these parameters on active contracts may be troublesome we will not be making changes at this time, partly because it's useful to be able to change these before the contracts are in use but also due to the potential risk of introducing new problems at this stage in the project.

[M-03] Deposits don't work with fee-on transfer tokens

Submitted by cmichel

There are ERC20 tokens that may make certain customizations to their ERC20 contracts. One type of these tokens is deflationary tokens that charge a certain fee for every `transfer()` or `transferFrom()` . Others are rebasing tokens that increase in value over time like Aave's aTokens (`balanceOf` changes over time).

The `RCTreasury.deposit()` function will credit more deposits than the contract actually received:

```
erc20.safeTransferFrom(msgSender(), address(this), _amount);
user[_user].deposit += SafeCast.toUint128(_amount);
```

Recommend ensuring that the `erc20` token does not implement any customizations. Alternatively, a mitigation is to measure the asset change right before and after the asset-transferring routines

Splidge (Reality Cards) acknowledged:

> The issue that **keeps on giving..**

Low Risk Findings (25)

[L-01] Can't retrieve all data with `getMarketInfo`

Submitted by gpersoon, also found by hickuphh3 and cmichel

The function `getMarketInfo` of `RCFactory` only can give results back in the range 0... `marketInfoResults` Supplying `_skipResults` doesn't help, it then just skips the first `_skipResults` records.

Assume `marketInfoResults` == 10 and `_skipResults` == 20: Then no result will be given back because " `_resultNumber` < `marketInfoResults` " will never allow `_resultNumber` to be bigger than 10

Note: this is low risk because `getMarketInfo` is a backup function (although you maybe want the backup to function as expected)

```
227
228 function getMarketInfo( IRCMarket.Mode _mode, uint256 _state
229     returns ( address[] memory, string[] memory, string[] me
230     ..
231     uint256 _resultNumber = 0;
```

```
232     ..
233     while (_resultNumber < marketInfoResults && _marketIndex
234         ...
235             if (_resultNumber < _skipResults) {
236                 _resultNumber++;
237             } else {
238                 _marketAddresses[_resultNumber] = _market;
239                 ....
240                 _resultNumber++;
241             }
242         }
243     }
244     return (_marketAddresses, _ipfsHashes, _slugs, _potSizes
```

Recommend updating the code to something like the following:

```
uint idx;
while (idx < marketInfoResults && _marketIndex > 1) {
    _marketIndex--;
    address _market = marketAddresses[_mode][_marketIndex];
    if (IRCMarket(_market).state() == IRCMarket.States(_state))
        if (_resultNumber < _skipResults) {
            _resultNumber++;
        } else {
            _marketAddresses[idx] = _market;
            _ipfsHashes[idx] = ipfsHash[_market];
            _slugs[idx] = addressToSlug[_market];
            _potSizes[idx] = IRCMarket(_market).totalRentCollect
            idx++;
        }
    }
}
```

[Splidge (Reality Cards) confirmed]:

> Yep, this was a last minute untested addition. In implementing your recommended fix I hit “Stack too deep” problems, so have split the function in two with one being internal. Firstly `getMarketInfo` will get the market addresses required and then `_getMarketInfo` will get the additional info required. This change has allowed me to remove the separate `marketResults` variable and pass the required number of results directly to `getMarketInfo`.

Splidge (Reality Cards) patched:

> Fixed here

[L-02] Return value of `erc20.approve` is unchecked

Submitted by shw

The `SafeERC20` library is used in the `RCTreasury` contract to handle the transfer of tokens that are not compliant with the ERC20 specification. However, in line 347, the `approve` function is used instead of the `safeApprove` function. Tokens not compliant with the ERC20 specification could return `false` from the `approve` function call to indicate the approval fails, while the calling contract would not notice the failure if the return value is not checked.

Recommend using the `safeApprove` function instead, which reverts the transaction with a proper error message when the return value of `approve` is `false`. A better approach is to use the `safeIncreaseAllowance` function, which mitigates the multiple withdrawal attack on ERC20 tokens.

Splidge (Reality Cards) disputed:

> I do not think using `safeApprove` would help here. Given the premise that the ERC20 in use is non-compliant and the approval fails, does it matter if the contracts are made aware of this or not? In either case transfers to the bridge contract would not be possible. The solution therefore isn't a different approval but using a different ERC20 which is compliant. Using a different ERC20 isn't necessary as the one we have chosen, USDC, is compliant.

> `safeIncreaseAllowance` wouldn't make any difference either, we are approving the transfer of more tokens than exist, it'd be pretty impressive to see a multiple withdrawal attack manage to withdraw enough tokens to warrant using the additional approval. A rough calculation shows that if you moved the entire USDC supply on Polygon in a single transaction (assuming the lower end gas 50k) it'd still take `2.89*10^23` blocks filled with transfers before you get to the limit of a `type(uin256).max` approval, even if we sped Polygon up to 10 blocks/second

> that'd still take 67,000 x the age of the universe to process 😱, but certainly after that we wouldn't want anybody using the extra approval.

Oxean (judge) commented:

> Agree with warden that it would be better to use safeApprove and revert the setBridge call with a reasonable error message than fail silently. The safeIncreaseAllowance doesn't *need* to be implemented based on the use case. Using safeERC20 calls in a contract that are imports the library seems like a no brainer even if the expected use case doesn't require it. These contracts will potentially persist much longer past everyone remembering all the nuances of which ERC20 tokens work and which might not.

Splidge (Reality Cards) acknowldeged:

> Understood, changing to Acknowledged as for this use case there is no impact and any further changes will delay other audits and our time to launch.

[L-03] Direct usage of `ecrecover` allows signature malleability

Submitted by shw

The `verify` function of `NativeMetaTransaction` calls the Solidity `ecrecover` function directly to verify the given signature. However, the `ecrecover` EVM opcode allows for malleable (non-unique) signatures and thus is susceptible to replay attacks. Although a replay attack on this contract is not possible since each user's nonce is used only once, rejecting malleable signatures is considered a best practice.

- NativeMetaTransaction.sol#L97
- SWC-117: Signature Malleability
- SWC-121: Missing Protection against Signature Replay Attacks

Recommend using the `recover` function from OpenZeppelin's ECDSA library for signature verification.

Splidge (Reality Cards) disputed:

> This is a **duplicate** of an issue found in the last contest.

> It was not solved last time because we didn't want to risk introducing problems by completely rewriting our own version of the MetaTx contract (the version we are using is from a trusted source and working in many production environments that have been audited), so instead of writing our own version I attempted to use the OpenZeppelin version. We need to use the upgradable versions of the OpenZeppelin contracts because we clone RCMarket.sol, however there was an incompatibility between ERC2771Context.sol which uses an `immutable` variable in the constructor and AccessControl.sol which calls _msgSender() in the constructor. Details on this can be found in **this thread.**.

> As you can see in that thread the incompatibility has apparently been fixed, however this was done only 5 hours before the start of this contest (and after we had submitted the code for the contest), so it wasn't practical to implement this solution for this contest. Regardless as you say this attack is "not possible since each user's nonce is used only once".

> I'll mark this as 'disputed' for now, if the judges decide that we should have re-written our own version (or discovered time-travel and implemented the OpenZeppelin version that's now availiable) then I'd be happy to change it to 'acknowledged' as we will not be making changes to such a sensitive area at this stage in the project.

0xean (judge) commented:

> Regardless of the solution and the mitigation steps the team is willing or not willing to take, I think highlighting the malleability issue is a valid 1 (Low Risk) issue that may not be worth fixing based on other risks as mentioned by @Splidge (Reality Cards)

[L-04] Return Value is Not Validated

Submitted by leastwood

The `circuitBreaker()` function in `RCMarket.sol` is utilised in the event an oracle never provides a response to a RealityCards question. The function makes an external call to the `RCOrderbook.sol` contract through the `closeMarket()`

function. If for some reason the orderbook was unable to be closed, this would never be checked in the `circuitBreaker()` function. See `RCMarket.sol` L1215-L1223.

Recommend ensuring this is intended behavior, or otherwise validate the response of `orderbook.closeMarket()`. Another option would be to emit the result of the external call in the `LogStateChange` event, alongside the state change.

Splidge (Reality Cards) confirmed:

> We have decided that given the extra complexity added to the contracts since the `circuitBreaker` was first implemented it will no longer be able to perform all the required functions and so it will be removed. If this call to the orderbook fails (or is removed) then users would not have their rentalRates reduced and so their funds would slowly be drained until the foreclose. We also have `setAmicableResolution` which can be used in the event of an oracle failure.

Splidge (Reality Cards) patched:

> `circuitBreaker` removed here

[L-05] External Call Made Before State Change

Submitted by leastwood

There are a number of functions in `RCTreasury.sol` which make external calls to another contract before updating the underlying market balances. More specifically, these affected functions are `deposit()`, `sponsor()`, and `topupMarketBalance()`. As a result, these functions would be prone to reentrancy exploits. However, as `safeTransferFrom()` operates on a trusted ERC20 token (RealityCard's token), this issue is of low severity. See issue page for reference code.

Recommend modifying the aforementioned functions such that all state changes are made before a call to the ERC20 token using the `safeTransferFrom()` function.

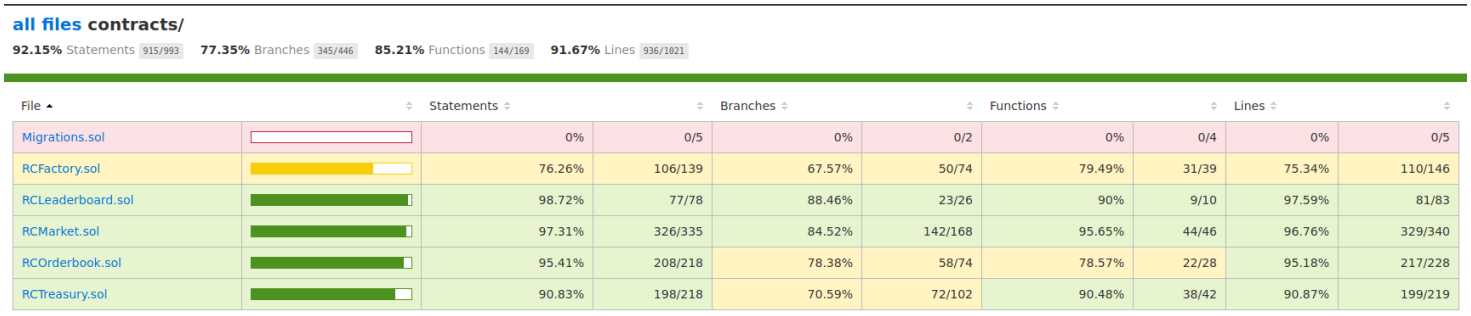
Splidge (Reality Cards) confirmed and patched:

> Fixed here

[L-06] Test Coverage Improvements

Submitted by leastwood

Adequate test coverage and regular reporting is an essential process in ensuring the codebase works as intended. Insufficient code coverage may lead to unexpected issues and regressions arising due to changes in the underlying smart contract implementation.



all files contracts/

92.15% Statements 915/993 **77.35%** Branches 345/446 **85.21%** Functions 144/169 **91.67%** Lines 936/1021

File	Statements		Branches		Functions		Lines	
Migrations.sol	0%	0/5	0%	0/2	0%	0/4	0%	0/5
RCFactory.sol	76.26%	106/139	67.57%	50/74	79.49%	31/39	75.34%	110/146
RCLeaderboard.sol	98.72%	77/78	88.46%	23/26	90%	9/10	97.59%	81/83
RCMarket.sol	97.31%	326/335	84.52%	142/168	95.65%	44/46	96.76%	329/340
RCOrderbook.sol	95.41%	208/218	78.38%	58/74	78.57%	22/28	95.18%	217/228
RCTreasury.sol	90.83%	198/218	70.59%	72/102	90.48%	38/42	90.87%	199/219

Image above showcases total test coverage of the target contracts.

Recommend ensuring the coverage report produced via `npx hardhat coverage` covers all functions within Reality Card's smart contract suite.

Splidge (Reality Cards) acknowledged:

> We acknowledge further testing is important and will continue to improve the test coverage going forward.

[L-07] `RCFactory` : Solve stack too deep for `getMarketInfo()`

Submitted by hickuphh3

The `marketInfoResults` is a parameter used by `getMarketInfo()` to determine the length of results to return. As the `setMarketInfoResults()` comments state, "(it) would be better to pass this as a parameter in `getMarketInfo` .. however we are limited because of stack too deep errors".

This limitation can be overcome by defining the return array variables as the function output, as suggested below.

The need for `marketInfoResults` and its setter function is then made redundant, whilst making querying results of possibly varying lengths more convenient.

Recommend the following:

```
function getMarketInfo(
  IRCMarket.Mode _mode,
  uint256 _state,
  uint256 _skipResults,
  uint256 _numResults // equivalent of marketInfoResults
)
  external
  view
  returns (
    address[] memory _marketAddresses,
    string[] memory _ipfsHashes,
    string[] memory _slugs,
    uint256[] memory _potSizes
        )
 {
          uint256 _marketIndex = marketAddresses[_mode].length;

          _marketAddresses = new address[](_numResults);
          _ipfsHashes = new string[](_numResults);
          _slugs = new string[](_numResults);
          _potSizes = new uint256[](_numResults);
                ...
}
```

Splidge (Reality Cards) confirmed:

> I wish I saw this before I did the fix to issue #14 , in implementing that fix I split `getMarketInfo()` in half making part of it internal which has managed to get around the “Stack too deep” problem and so I was able to get rid of `marketInfoResults` at the same time. Still this is useful to keep in mind for the future. 👍

Splidge (Reality Cards) patched:

> Fixed alongside issue #14 here

[L-08] `RCFactory` : Do multiplication instead of division for length checks

Submitted by hickuphh3, also found by leastwood

Solidity division rounds down, so doing `M / 2 <= N` checks mean that `M` can be at most `2N + 1`.

This affects the following checks:

```
    require(
            (_tokenURIs.length / 2) <= cardLimit,
            "Too many tokens to mint"
    );

    require(
      _cardAffiliateAddresses.length == 0 ||
              _cardAffiliateAddresses.length == (_tokenURIs.length /
      "Card Affiliate Length Error"
    )
```

Note that with the current implementation, if `_tokenURIs` is of odd length, its last element will be redundant, but market creation will not revert.

The stricter checks will partially mitigate `_tokenURIs` having odd length because `_cardAffiliateAddresses` is now required to be exactly twice that of `_tokenURIs`.

These checks should be modified to:

```
    require(
            _tokenURIs.length <= cardLimit * 2,
            "Too many tokens to mint"
    );

    require(
      _cardAffiliateAddresses.length == 0 ||
              _cardAffiliateAddresses.length * 2 == _tokenURIs.lengt
      "Card Affiliate Length Error"
```

```
    );
```

In addition, consider adding a check for `_tokenURIs` to strictly be of even length.

```
require(_tokenURIs.length % 2 == 0, "TokenURI Length Error");
```

Splidge (Reality Cards) confirmed and patched:

> Fixed **here**

[L-09] safer implementation of `tokenExists`

Submitted by gpersoon

The function `tokenExists` does only limited checks on the existence of cards. It doesn't doublecheck that `tokenIds[_card]` != 0 This is relevant because 0 is the default value of empty array elements. Although this isn't a problem in the current code, future changes might accidentally introduce vulnerabilities.

Also cards are only valid if they are below `numberOfCards` . This has led to vulnerabilities in previous versions of the contract (e.g. previous contest)

```
1139
1140 function tokenExists(uint256 _card) internal view returns (
1141         return tokenIds[_card] != type(uint256).max;
1142 }
```

Recommend changing the function to something like the following:

```
function tokenExists(uint256 _card) internal view returns (bool)
    if (_cardId >= numberOfCards) return false;
    if (tokenIds[_card] == 0) return false;
    return tokenIds[_card] != type(uint256).max;
}
```

Splidge (Reality Cards) confirmed:

> Obviously I didn't learn my lesson from the last contest, I've added in the check that the card Id is less than the `numberOfCards`.

> However, I'm reluctant to disallow the use of tokenId 0. This would mean that the first NFT we mint wouldn't be usable in a market, so we would need to create a dead market just to get rid of that first NFT. All things going well the first NFT minted could end up being a valuable one, who knows..? 🚀 An alternative would be for the factory to start minting from index 1 but this would mean instead of using the `totalSupply()` as the next token to mint, we would need to use `totalSupply() + 1`, I'm unsure how this would affect integration with services such as opensea where they may expect the first NFT to have index 0, I feel like at this late stage it's too much of a change that could introduce other issues later, and so for now I'll not be adding `if (tokenIds[_card] == 0) return false;`

[Splidge (Reality Cards) patched]:

> Fix to the check that the `_card` is a valid card number added in [this] commit

[L-10] `uint32` conversion doesn't work as expected.

Submitted by gpersoon, also found by 0xsanson

The `uint32` conversion in `setWinner` of the `RCMarket` doesn't work as expected. The first statement: "`uint32(block.timestamp)`" already first the `block.timestamp` in a `uint32`. If it is larger than `type(uint32).max` it wraps around and starts with 0 again The testcode below shows this.

Check for "`<= type(uint32).max`" in the second statement is useless because `_blockTimestamp` is always <= `type(uint32).max`

```
507
508  function setWinner(uint256 _winningOutcome) internal {
509 ...
510      uint256 _blockTimestamp = uint32(block.timestamp);
511      require(_blockTimestamp <= type(uint32).max, "Overflow")
512
513
514 //Testcode:
```

```
515 pragma solidity 0.8.7;
516 contract Convert {
517     uint256 public a = uint256( type(uint32).max )+1; // a==4
518     uint32  public b = uint32(a); // b==0
519     uint256 public c = uint32(a); // c==0
520 }
```

Recommend doing the `require` first (without a typecast to `uint32`):

```
require( block.timestamp <= type(uint32).max, "Overflow");
uint256 _blockTimestamp = uint32(block.timestamp);
```

Splidge (Reality Cards) confirmed:

> Good to know about this. My assumption would have been that this overflow should have been guarded against with the inbuilt SafeMath with Solidity ^0.8. I suppose I put it right there in comments that this should have used SafeCast and not SafeMath 🤦

Splidge (Reality Cards) patched:

> Fixed here

[L-11] `msgSender()` or `_msgSender()`

Submitted by gpersoon

The code has two implementations of `msgSender`:

- `msgSender()` => uses meta transaction signer
- `_msgSender()` => maps to `msg.sender`

`_msgSender()` is used in a few locations

- when using `_setupRole`, this seems legitimate
- in function `withdraw` (whereas the similar function `withdrawWithMetadata` uses `msgSender()`)

It is confusing to have multiple functions with almost the same name, this could easily lead to mistakes.

```
105
106 function msgSender() internal view returns (address payable
107     if (msg.sender == address(this)) {
108         assembly {   sender := shr(96, calldataload(sub(call
109     } else {
110             sender = payable(msg.sender);
111     }
112     return sender;
113 }
```

```
// https://github.com/OpenZeppelin/openzeppelin-contracts/blob/m
function _msgSender() internal view virtual returns (address) {
    return msg.sender;
}
```

```
164
165 function withdraw(uint256 tokenId) external override {
166     require(  _msgSender() == ownerOf(tokenId), "ChildMintab
167     withdrawnTokens[tokenId] = true;
168     _burn(tokenId);
169 }
```

```
function withdrawWithMetadata(uint256 tokenId) external overrid
    require( msgSender() == ownerOf(tokenId), "ChildMintableERC
    withdrawnTokens[tokenId] = true;
    // Encoding metadata associated with tokenId & emitting ever
    emit TransferWithMetadata( ownerOf(tokenId), address(0), tok
    _burn(tokenId);
}
```

```
RCNftHubL1.sol:       _setupRole(DEFAULT_ADMIN_ROLE, _msgSender()
RCNftHubL2.sol:       _setupRole(DEFAULT_ADMIN_ROLE, _msgSender()
RCTreasury.sol:         _setupRole(DEFAULT_ADMIN_ROLE, _msgSender
RCTreasury.sol:         _setupRole(UBER_OWNER,                 _msg
RCTreasury.sol:         _setupRole(OWNER,
```

```
RCTreasury.sol:        _setupRole(GOVERNOR,                    _m
RCTreasury.sol:        _setupRole(WHITELIST,
```

Recommend double-checking the use of `_msgSender()` in withdraw and adjust if necessary. Also, adding comments when using `_msgSender()`. And finally consider overriding `_msgSender()`, as is done in the example here:

Splidge (Reality Cards) confirmed:

> I'm not sure this would have caused problems because we never intend to use `withdraw`, it's simply there as a requirement for Matic Mintable Asset Mapping. But as we stated in the readme all functions should use `msgSender()` I have removed all instances of `_msgSender()`

Splidge (Reality Cards) patched:

> Fixed here

[L-12] `rentAllCards` : don't have to pay for card you already own

Submitted by gpersoon

The function `rentAllCards` of `RCMarket` checks for `_maxSumOfPrices` to see you are not paying more that you want.

However, the first part of the calculations (which calculate `_actualSumOfPrices`), do not take in account the fact that you might already own a card. (while the second part of the code does). If you already own the card you don't have to pay for it and you certainly don't have to pay the extra `minimumPriceIncreasePercent`.

The code at "Proof of Concept" shows a refactored version of the code (see other issue "make code of `rentAllCards` easier to read"). This immediately shows the issue.

```
// https://github.com/code-423n4/2021-08-realitycards/blob/main/
function calc(uint256 currentPrice) returns(uint256) {
```

```
        if (currentPrice == 0)
            return MIN_RENTAL_VALUE;
        return (currentPrice *(minimumPriceIncreasePercent + 100)) /
    }

    function rentAllCards(uint256 _maxSumOfPrices) external override
        ..
        uint256 _actualSumOfPrices = 0;
        for (uint256 i = 0; i < numberOfCards; i++) {
            _actualSumOfPrices += calc(card[i].cardPrice);    // no c
        }
        require(_actualSumOfPrices <= _maxSumOfPrices, "Prices too h

        for (uint256 i = 0; i < numberOfCards; i++) {
            if (ownerOf(i) != msgSender()) {
                uint256 _newPrice=calc(card[i].cardPrice);
                newRental(_newPrice, 0, address(0), i);
            }
        }
    }
```

Add " `if (ownerOf(i) != msgSender()) {` " also in the first part of the code of `rentAllCards`

```
    uint256 _actualSumOfPrices = 0;
    for (uint256 i = 0; i < numberOfCards; i++) {
        if (ownerOf(i) != msgSender()) {                // extra if st
            _actualSumOfPrices += calc(card[i].cardPrice);
        }
    }
```

[Splidge (Reality Cards) confirmed and patched]:

> Fixed in the same commits as issue #20 [Here] and [here]

[L-13] `getMostRecentMarket` can revert

Submitted by gpersoon

The function `getMostRecentMarket` of `RCFactory.sol` will revert if no markets of the specific mode are created yet.

```
171
172 function getMostRecentMarket(IRCMarket.Mode _mode)  external
173 {
174     return marketAddresses[_mode][marketAddresses[_mode].len
175 }
```

Recommend changing the function `getMostRecentMarket` to something like:

```
function getMostRecentMarket(IRCMarket.Mode _mode)  external vie
    if ( marketAddresses[_mode].length ==0) return address(0);
    return marketAddresses[_mode][marketAddresses[_mode].length
}
```

Splidge (Reality Cards) confirmed and patched:

> Fixed **here**

[L-14] `updateTokenURI` doesn't call `setTokenURI`

Submitted by gpersoon

The function `updateTokenURI` of `RCFactory.sol` doesn't update the uris of `RCNftHubL2`. E.g. it doesn't call `setTokenURI` to try and update the already created NFT's. This way the URIs of already minted tokens are not updated.

See issue page for referenced code.

Recommend also calling `setTokenURI` of `RCNftHubL2`. Or, restricting `updateTokenURI` to the phase where no NFT's are minted yet. Or, at least add comments to `updateTokenURI`.

Splidge (Reality Cards) confirmed:

> The reason behind `updateTokenURI` is because in some early markets we found that the artwork used had copyright or trademark infringements. It was recommended from a legal point of view that we should have a mechanism for removing/amending the artwork. The end goal would be to decentralize the whole project and at that stage this legal issue disappears along with the need to change the tokenURI.

> Also call setTokenURI of RCNftHubL2

> This could prove difficult as we haven't settled on a maximum number of NFTs we can award from the leaderboard and so there could be gas limits around updating all the tokenURIs for already minted tokens. There also exists the potential that the user has upgraded the NFT to the mainnet. Given how infrequently (ideally never) we plan to use this feature it seemed reasonable to be able to call `updateTokenURI` so all new NFTs are minted with the updated metadata and then call `setTokenURI` (on either layer 1 or layer 2) to update all the already minted tokens.

> Or restrict updateTokenURI to the phase where no NFT's are minted yet.

> This would only work if we knew before we minted anything that there was a legal infringement, in which case we wouldn't use the art to begin with.

> Or at least add comments to updateTokenURI

> I will add some comments to clarify the reasons for `updateTokenURI` and why we don't update already minted tokens.

> I also notice while writing this out that if we intend to relinquish the ability to `updateTokenURI` then this power should be given to the UBER_OWNER and not the OWNER. I will change this.

[Splidge (Reality Cards) patched]:

> Fixed [here]

[L-15] `RCTreasury : AccessControl` diagram contains Leaderboard, but it has no role

Submitted by hickuphh3, also found by leastwood and cmichel

See issue page for diagram and discussion.

[L-16] `RCLeaderboard.market` storage variable is not used

Submitted by cmichel

The `RCLeaderboard.market` storage variable is never used. Instead, the `MARKET` role seems to be used to implement authentication.

Unused code can hint at programming or architectural errors.

Recommend using it or remove it.

Splidge (Reality Cards) confirmed:

> Removed it.

Splidge (Reality Cards) patched:

> Fixed here

[L-17] Markets can start in the past

Submitted by cmichel

The `RCFactory._checkTimestamps` function only checks that the start timestamp (`_timestamps[0]`) is not in the past if `advancedWarning != 0`.

Markets can be created that already started in the past. I'm not sure if this is intended.

Recommend always performing the `require(_timestamps[0] >= block.timestamp, "Market opening time not set");` check, not only in the

`advancedWarning != 0` case.

Splidge (Reality Cards) disputed:

> In the case where we want to open a market immediately it would be difficult to do so without allowing timestamps in the past because we can't say what time the transaction will be mined and so what the `block.timestamp` would be. By not checking this we can open markets immediately with ease.

0xean (judge) commented:

> The check could allow for some expected amount of time slippage due to mining to have transpired and potentially still be useful to avoid errors. I think it's a valid point by the warden.

Splidge (Reality Cards) acknowledged:

> Changing to acknowledged based on the judges assessment.

[L-18] add zero address validation in constructor

Submitted by JMukesh, also found by cmichel

since the parameter in the constructor are used to initialize the state variable , proper check up should be done , other wise error in these state variable can lead to redeployment of contract. See issue page for referenced code.

Recommend adding zero address validation.

Splidge (Reality Cards) confirmed:

> RCOrderbook and RCTreasury both have setter functions where an incorrectly initialized variable could be set after deployment. It's unlikely for these to be set incorrectly because of the deployment script we use, however I'll add a setter to the RCLeaderboard so we can set the address afterwards.

Splidge (Reality Cards) patched:

> Treasury setter in RCLeaderboard added here

[L-19] use of array without checking its length

Submitted by JMukesh

Since no limit is mentioned in `batchWhitelist()` for the input of `_users` array , it may run out of gas if array length become large. See `RCTreasury.sol` L249.

Recommend adding a limitation for which , this number of address can be whitelisted at a time.

Splidge (Reality Cards) acknowledged:

> This whitelist is a temporary measure to be used in the Beta and the run-up to launch, after launch it will be disabled. As such we will not be making changes to a feature that will not be used going forward.

[L-20] No check for the `referenceContractAddress` in `createMarket()`

Submitted by JMukesh

`referenceContractAddress` is used in `createMarket()` to create `newAddress` for the market , a necessary check should be there that `referenceContractAddress` exist or not, because if `createMarket()` is called before `setReferenceContractAddress()` , `address(0)` will be passed as `referenceContractAddress` , since `addMarket()` of `treasury` and `nfthub` does not have address validation for the market. See `RCFactory.sol` L714.

Recommend adding a condition to check the `referenceContractAddress`

Splidge (Reality Cards) confirmed and patched:

> Fixed here

[L-21] no time restriction in `setMarketTimeRestrictions ()`

Submitted by JMukesh

As mentioned in the comment, time must be at least this many seconds, but it lack a check that given time is `atleast >= someTime` , as a result `minimumDuration` and `maximumDuration` are directly initialized without any check. See `RCFactory.sol` L431.

Recommend adding require condition to check those value before setting it.

Splidge (Reality Cards) confirmed:

> This is just poor commenting from when `advancedWarning` , `minimumDuration` and `maximumDuration` each had their own setter functions. They were combined to avoid the Factory going over the contract size limit. However the comments weren't combined correctly. 0 seconds is a valid `advancedWarning` if you want the market to open immediately. 0 seconds for `maximumDuration` may also be used if we don't want to set a maximum.

> I'll not be adding in the checks but I will update the comments.

Splidge (Reality Cards) acknowledged:

> On looking back through this I may have misunderstood what the warden was proposing here. I thought the warden wanted a zero value check, but they could have been asking for the `minimumDuration < maximumDuration` check. This would make more sense as if the maximum was less than the minimum it wouldn't be possible to create a new market. The same problem would happen if `advancedWarning` was set to be greater than the `maximumDuration` .
> Unfortunately the Factory is right on the size limit and adding these checks would require other changes to the contract which pose the risk of introducing more problems. Given how infrequently we expect to change these values, and the fact they can be changed back again should a mistake be made, I'll be marking this issue as acknowledged. Initial fix to add more comments was implemented here

[L-22] `transferCard` should be done after treasury is updated.

Submitted by 0xImpostor

When the current owner of the card is still the new owner of the card, `transferCard` is called before the treasury is updated. While this does not currently pose a risk, it is not aligned with best practices of [check-effect-interations] and opens your code to a potential re-entrancy attack in the future.

```
// line 381
treasury.updateRentalRate(
    _oldOwner,
    _user,
    user[_oldOwner][index[_oldOwner][_market][_card]].price,
    _price,
    block.timestamp
);
transferCard(_market, _card, _oldOwner, _user, _price);
...
// line 449
treasury.updateRentalRate(
    _user,
    _user,
    _price,
    _currUser.price,
    block.timestamp
);
transferCard(_market, _card, _user, _user, _currUser.price);
```

Splidge (Reality Cards) commented:

> Fixed here

[L-23] Inaccurate Comment

Submitted by leastwood

This issue has no direct security implications, however, there may be some confusion when understanding what the `RCFactory.createMarket()` function actually does.

See `RCFactory.sol` **L625**.

Recommend updating the line (linked above) to include the `SAFE_MODE` option outline in the `enum` type in `IRCMarket.sol`. For example, the line `/// @param _mode 0 = normal, 1 = winner takes all` could be updated to `/// @param _mode 0 = normal, 1 = winner takes all, 2 = SAFE_MODE`

Oxean (judge) confirmed:

> Based on C4's docs - Comment issues are 1, bumping to 1.

> "1 — Low: Low: Assets are not at risk. State handling, function incorrect as to spec, issues with comments."

Splidge (Reality Cards patched):

> Fixed here

[L-24] `RCLeaderboard` : Erroneous comment

Submitted by hickuphh3

The comments above the event declarations were probably copied over from `RCOrderbook`. They should be modified to refer to the leaderboard.

```
    /// @dev emitted every time a user is added to the leaderboard
    event LogAddToLeaderboard(address _user, address _market, uint25
    /// @dev emitted every time a user is removed from the leaderboa
    event LogRemoveFromLeaderboard(
        address _user,
        address _market,
        uint256 _card
    );
```

Splidge (Reality Cards) confirmed:

> probably copied over from RCOrderbook

> Exactly. I'll correct this.

Oxean (judge) commented:

```
1 — Low: Low: Assets are not at risk. State handling, function i
```

Splidge (Reality Cards) patched:

> Fixed here

[L-25] Use `_safeTransfer` when transferring NFTs

Submitted by shw

The `transferNft` function of `RCNftHubL2` is called when transferring the card to the final winner. However, this function does not check whether the recipient is aware of the ERC721 protocol and calls `_transfer` directly. If the recipient is a contract not aware of incoming NFTs, then the transferred NFT would be locked in the recipient forever. See `RCNftHubL2.sol` **L135**.

Recommend using the `_safeTransfer` function instead, which checks if the recipient contract implements the `onERC721Received` interface to avoid loss of NFTs.

Splidge (Reality Cards) disputed:

> This transfer function is just for the market to move the NFTs while the market is operational, during this phase it doesn't matter if the NFT is given to a non-compliant contract as the market is in control of moving the NFT and will simply give it to the next user that makes a rental. Using `_safeTransfer` would be dangerous as it would give an attacker the opportunity to take part in a market via a contract that is not ERC721 compliant, when the market attempts to pass ownership to this contract the transfer would revert and the market would become locked.

Oxean (judge) commented:

> `claimCard` in RCMarket.sol also calls the transfer method to claim the card, which could result in the situation outlined by the warden.

Splidge (Reality Cards) commented:

> Ahh sorry, I missed the key part in the wardens impact "when transferring the card to the final winner".

> I still think this isn't an issue as if the destination is a non-compliant contract, when using `_safeTransfer` the `claimCard` call would revert. The NFT is permanently stuck on either the market or on the non-compliant contract. At least the person who wrote the contract, funded it, used it to place bids, win the NFT and make the `claimCard` call would still get it transferred to their contract to use as a trophy cabinet of sorts. Maybe that person didn't implement `onERC721Received` ? This check would be too little too late.

> Maybe we could check when the first rental is made that the sender is eligible? Although that would deviate slightly from the standard as `onERC721Received` is supposed to be called after the transfer.

Non-Critical Findings (11)

- [N-01] Unable to Recover Improperly Transferred Tokens
- [N-02] User can deposit more than `maxContractBalance`
- [N-03] Divide Before Multiply
- [N-04] Remove hardhat console import
- [N-05] `RCTreasury` : Spelling Errors
- [N-06] `RCTreasury` : new `hasRole()` function with string role
- [N-07] `RCFactory` : Unnecessary casting in `updateTokenURI()`
- [N-08] Access Control Constants
- [N-09] `tokenExists` ==> `cardExists`
- [N-10] remove `addMarket` from `RCNftHubL2`
- [N-11] remove unused modifiers

Gas Optimizations (9)

- [G-01] Overflow in `Mode` Type
- [G-02] gas saving in `_processRentCollection`
- [G-03] `RCMarket.sol` - Gas optimization in `_payoutWinnings`
- [G-04] `RCMarket.sol` - Gas optimization in `claimCard`
- [G-05] optimize `_beforeTokenTransfer`
- [G-06] make code of `rentAllCards` easier to read and maintain
- [G-07] Tight Variable Packing
- [G-08] Allowance checks are not required
- [G-09] Gas optimization in `RCMarket.sol` and other files

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top