Covalent contest
Findings & Analysis Report

2021-11-19

Table of contents

Overview

About C4

Code4rena (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 Covalent smart contract system written in Solidity. The code contest took place between October 19—October 21 2021.

Wardens

14 Wardens contributed reports to the Covalent code contest:

1. cmichel
2. WatchPug (jtp and ming)
3. jonah1005
4. xYrYuYx
5. pants
6. hickuphh3
7. gpersoon
8. ye0lde
9. pauliax
10. harleythedog
11. pmerkleplant

12. defsec
13. csanuragjain

This contest was judged by Alex the Entreprenerd.

Final report assembled by CloudEllie and moneylegobatman.

Summary

The C4 analysis yielded an aggregated total of 9 unique vulnerabilities and 38 total findings. 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 4 received a risk rating in the category of LOW severity.

C4 analysis also identified 11 non-critical recommendations and 18 gas optimizations.

Scope

The code under review can be found within the C4 Covalent contest repository, and is composed of 17 smart contracts written in the Solidity programming language and includes 442 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] Usage of an incorrect version of `Ownbale` library can potentially malfunction all `onlyOwner` functions

Submitted by WatchPug

`DelegatedStaking.sol` L62-L63

```
    // this is used to have the contract upgradeable
    function initialize(uint128 minStakedRequired) public initialize
```

Based on the context and comments in the code, the `DelegatedStaking.sol` contract is designed to be deployed as an upgradeable proxy contract.

However, the current implementation is using an non-upgradeable version of the `Ownbale` library: `@openzeppelin/contracts/access/Ownable.sol` instead of the upgradeable version: `@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol`.

A regular, non-upgradeable `Ownbale` library will make the deployer the default owner in the constructor. Due to a requirement of the proxy-based upgradeability system, no constructors can be used in upgradeable contracts. Therefore, there will be no owner when the contract is deployed as a proxy contract.

As a result, all the `onlyOwner` functions will be inaccessible.

Recommendation

Use `@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol` and `@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol` instead.

And change the `initialize()` function to:

```
function initialize(uint128 minStakedRequired) public initialize
    __Ownable_init();
    ...
}
```

[kitti-katy (Covalent) confirmed](#)

[GalloDaSballo (judge) commented](#):

> Agree with the finding, when using Upgradeable Proxies it's important to use the adequate libraries that will be compatible with initializable contracts

[GalloDaSballo (judge) commented](#):

> The sponsor has mitigated the issue

[kitti-katy (Covalent) patched](#)

[H-02] `unstake` should update exchange rates first

Submitted by cmichel

The `unstake` function does not immediately update the exchange rates. It first computes the `validatorSharesRemove = tokensToShares(amount, v.exchangeRate)` **with the old exchange rate.**

Only afterwards, it updates the exchange rates (if the validator is not disabled):

```
// @audit shares are computed here with old rate
uint128 validatorSharesRemove = tokensToShares(amount, v.exchang
require(validatorSharesRemove > 0, "Unstake amount is too small'

if (v.disabledEpoch == 0) {
    // @audit rates are updated here
    updateGlobalExchangeRate();
    updateValidator(v);
```

```
        // ...
    }
```

Impact

More shares for the amount are burned than required and users will lose rewards in the end.

POC

Demonstrating that users will lose rewards:

1. Assume someone staked `1000 amount` and received `1000 shares`, and `v.exchangeRate = 1.0`. (This user is the single staker)
2. Several epochs pass, interest accrues, and `1000 tokens` accrue for the validator, `tokensGivenToValidator = 1000`. User should be entitled to 1000 in principal + 1000 in rewards = 2000 tokens.
3. But user calls `unstake(1000)`, which sets `validatorSharesRemove = tokensToShares(amount, v.exchangeRate) = 1000 / 1.0 = 1000`. **Afterwards**, the exchange rate is updated: `v.exchangeRate += tokensGivenToValidator / totalShares = 1.0 + 1.0 = 2.0`. The staker is updated with `s.shares -= validatorSharesRemove = 0` and `s.staked -= amount = 0`. And the user receives their 1000 tokens but notice how the user's shares are now at zero as well.
4. User tries to claim rewards calling `redeemAllRewards` which fails as the `rewards` are 0.

If the user had first called `redeemAllRewards` and `unstake` afterwards they'd have received their 2000 tokens.

Recommended Mitigation Steps

The exchange rates always need to be updated first before doing anything. Move the `updateGlobalExchangeRate()` and `updateValidator(v)` calls to the beginning of the function.

kitti-katy (Covalent) confirmed

GalloDaSballo (judge) commented:

> Agree with the finding, using the old exchange rate ends up burning more shares than what would be correct The sponsor has mitigated the issue

Medium Risk Findings (3)

[M-01] reward tokens could get lost due to rounding down

Submitted by gpersoon, also found by hickuphh3, xYrYuYx, and jonah1005

Impact

The function `depositRewardTokens` divides the “amount” of tokens by `allocatedTokensPerEpoch` to calculate the `endEpoch`. When “amount” isn’t a multiple of `allocatedTokensPerEpoch` the result of the division will be rounded down, effectively losing a number of tokens for the rewards.

For example if `allocatedTokensPerEpoch` is set to 3e18 and “amount” is 100e18 then `endEpoch` will be increased with 33e18 and the last 1e18 tokens are lost.

A similar problem occurs here:

- in `setAllocatedTokensPerEpoch()`, with the recalculation of `endEpoch`
- in `takeOutRewardTokens()`, with the retrieval of tokens
- in _stake(), when initializing `endEpoch` (e.g. when `endEpoch` ==0)

Proof of Concept

- `DelegatedStaking.sol` L90-L98
- `DelegatedStaking.sol` L368-L383

Recommended Mitigation Steps

In `depositRewardTokens()` add, in the beginning of function, before the if statement:

```
    require(amount % allocatedTokensPerEpoch == 0,"Not multiple");
```

In `takeOutRewardTokens()` add:

```
    require(amount % allocatedTokensPerEpoch == 0,"Not multiple");
```

Update `setAllocatedTokensPerEpoch()` to something like:

```
    if (`endEpoch` != 0) {
    ...
    uint128 futureRewards = ...
    require(futureRewards % amount ==0,"Not multiple");
    ...\
    } else { // to prevent issues with \_stake()
    require(rewardsLocked % allocatedTokensPerEpoch==0,"Not multiple
    }
```

[kitti-katy (Covalent) confirmed](#):

> Agreed, the original assumption was that the owner would always make sure the take out and deposit amount is multiple of emission rate. But yes, this is good to add the check. Also it is not that risky since the emission rate wouldn't be that high per epoch and the loss will always be less than the emission rate.

[GalloDaSballo (judge) commented](#):

> Agree with the finding, since it's a rounding error the max loss in rewards can at most be 1 less than the denominator

> That said, this is a Medium Severity Finding as per the doc: `2 — Med: Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.`

> Where in this case the rounding is a way to leak value (loss of yield)

[M-02] Incorrect `updateGlobalExchangeRate` implementation

Submitted by xYrYuYx

Impact

`UpdateGlobalExchangeRate` has incorrect implementation when `totalGlobalShares` is zero.

If any user didn't start stake, `totalGlobalShares` is 0, and every stake it will increase. but there is possibility that `totalGlobalShares` can be 0 amount later by unstake or disable validator.

Proof of Concept

This is my test case to proof this issue: C4_issues.js L76

In my test case, I disabled validator to make `totalGlobalShares` to zero. And in this case, some reward amount will be forever locked in the contract. After disable validator, I mined 10 blocks, and 4 more blocks mined due to other function calls, So total 14 CQT is forever locked in the contract.

Tools Used

Hardhat test

Recommended Mitigation Steps

Please think again when `totalGlobalShares` is zero.

kitti-katy (Covalent) acknowledged:

> That is right, and I think the best solution would be to add a validator instance who is the owner and stake some low amount of tokens in it. This way we can make sure there is no such situation when `totalGlobalShares` becomes `0` and if everyone unstaked, the owner could take out reward tokens and then unstake / redeem rewards.

> Not sure. That could even be marked as “high risk”. if the situation happens and not handled right away (taking out reward tokens), then there could be more significant financial loss.

[kitti-katy (Covalent) commented](#):

> marked resolved as it will be manually handled

[GalloDaSballo (judge) commented](#):

> The issue found by the warden is straightforward: Through mix of unstaking and the use of `disableValidator` the warden was able to lock funds, making them irredemeable

> It seems to me that this is caused by the fact that `unstake` as well as `disableValidator` will reduce the shares: https://github.com/code-423n4/2021-10-covalent/blob/a8368e7982d336a4b464a53cfe221b2395da801f/contracts/DelegatedStaking.sol#L348`

> I would recommend separating the shares accounting from the activation of validator, simply removing the subtraction of global shares in `disableValidator` would allow them to claim those shares.

> The function `disableValidator` can be called by either the validator or the owner, while onlyOwner can add a new validator

> The owner has the ability to perform this type of griefing, as well as a group of validators if they so chose

> Due to the specifics of the grief I will rate it of Medium Severity, as per the docs: `2 — Med: Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.`

> In this case we have a way to leak value (lock funds) with specific condition (malicious owner or multiple griefing validators)

[M-03] Validator can fail to receive commission reward in `redeemAllRewards`

Submitted by jonah1005

Impact

Validator can fail to receive commission reward by calling `redeemAllRewards`.
There's a check in `redeemAllRewards`

```
    uint128 rewards = sharesToTokens(s.shares, v.exchangeRate) - s.s
    require(rewards > 0, "Nothing to redeem");
```

The validator's tx might be reverted here even if he got some commission reward to receive.

Proof of Concept

We can trigger the bug by setting `commisionRate` to `1e18 - 1` (DelegatedStaking.sol L275-L276)

Recommended Mitigation Steps

Though this may rarely happen and the validator can redeem the reward through `redeemRewards`, this may cause some issues when the validator is handled by a contract.

I consider calling `redeemRewards` in `redeemAllReawards` as a more succinct way to do this.

kitti-katy (Covalent) acknowledged:

> I don't think there will ever be a commission rate set to almost 100%. Since it is changed by the owner we will make sure the input is correct.

GalloDaSballo (judge) commented:

> Agree with the finding and understand the sponsors take.

> As per the docs for contests: `2 — Med: Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.`

> This fall as a medium severity

> A simple mitigation would be to limit the `commisionRate` to less than the value that causes issues

Low Risk Findings (4)

- [L-01] `addValidator()` : Validator's commission rate should be checked to not exceed divider *Submitted by hickuphh3, also found by cmichel, jonah1005, and pauliax*
- [L-02] Line 127 lack of precision *Submitted by pants*
- [L-03] addValidatior doesn't check new validator address != 0 *Submitted by pants*
- [L-04] Unbounded iteration over validators array *Submitted by cmichel*

Non-Critical Findings (11)

- [N-01] Typos *Submitted by WatchPug, also found by hickuphh3 and yeOlde*
- [N-02] Unnecessary require checker *Submitted by xYrYuYx*
- [N-03] Inconsistent definition of integer sizes in function `getDelegatorDetails()` *Submitted by pmerkleplant*
- [N-04] Unclear definition of `validatorId` 's integer size *Submitted by pmerkleplant*
- [N-05] Misleading parameter name *Submitted by WatchPug*
- [N-06] Code Style: private/internal function names should be prefixed with `_` *Submitted by WatchPug*
- [N-07] Code duplication *Submitted by WatchPug*
- [N-08] Make more data accessible *Submitted by hickuphh3*

- [N-09] `getValidatorsDetails` is getting disabled validators as well *Submitted by csanuragjain*
- [N-10] emit staked should be at stake function and not _stake. *Submitted by pants*
- [N-11] emit initialize *Submitted by pants*

Gas Optimizations (18)

- [G-01] Move Function `_stake` Validator Declaration *Submitted by yeOlde*
- [G-02] Adding unchecked directive can save gas *Submitted by WatchPug, also found by pauliax and yeOlde*
- [G-03] Long Revert Strings *Submitted by yeOlde*
- [G-04] Update function access *Submitted by xYrYuYx, also found by WatchPug, defsec, harleythedog, pants, and pauliax*
- [G-05] Recommend to use OZ SafeERC20 library *Submitted by xYrYuYx, also found by cmichel, defsec, and pants*
- [G-06] Declare variable `CQT` as constant *Submitted by pmerkleplant, also found by harleythedog*
- [G-07] Change lines to save gas *Submitted by pants*
- [G-08] Change order of lines to save gas in `setAllocatedTokensPerEpoch` *Submitted by pants*
- [G-09] `getDelegatorDetails` declaration inside a loop *Submitted by pants*
- [G-10] Cache storage variables in the stack can save gas *Submitted by WatchPug, also found by harleythedog and pants*
- [G-11] ++i is more gas efficient than i++ in loops forwarding *Submitted by pants*
- [G-12] `delegatorCoolDown` *Submitted by pants*
- [G-13] state variable divider could be set immutable. *Submitted by pants, also found by WatchPug and jonah1005*
- [G-14] `takeOutRewardTokens()` : Optimise epochs calculation and comparison *Submitted by hickuphh3, also found by WatchPug*

- [G-15] reset `rewardsLocked` to 0 when no longer used *Submitted by gpersoon*
- [G-16] Check `validatorId < validatorsN` can be done earlier *Submitted by WatchPug*
- [G-17] Avoid unnecessary storage read can save gas *Submitted by WatchPug*
- [G-18] unnecessary assert when dealing with CQT *Submitted by jonah1005*

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