Exactly

Smart Contract Audit

coinspect

Exactly

Exactly RewardsController

February 2023

Smart Contract Audit

V230404 Prepared for Exactly • April 2023

1. Executive Summary

In **February 2023, Exactly** engaged Coinspect to perform a source code review of changes to the **Exactly Protocol**. The objective of the project was to evaluate the security of these modifications.

This report joins a timeline of correlative security audits along with issue IDs, highlighting multiple distinct bugs within the same scope.

One low-risk issue was identified during the assessment. Two informational notes are added to the report which pose no immediate risk.

High Risk	Medium Risk	Low Risk
Open **0**	Open **0**	Open **0**
Fixed 0	Fixed 0	Fixed 2
Reported 0	Reported 0	Reported 2

EXR-8 describes how admins can inadvertently break the reward system by setting some variables to invalid values; even though other variables are bound by the smart contract. **EXR-11** shows how admins are unable to restart some distributions.

EXR-12 is an informational comment that explains the risk of the reward rate minimum value not being adequate for certain tokens. **EXR-9** and **EXR-10** are also informational comments regarding code quality and consequences of the changes with no immediate security risk but that Coinspect believes Exactly's team should be aware of.

2. Assessment and Scope

In February 2023, **Exactly Protocol** engaged Coinspect to perform a review on the changes introduced to the **RewardsController** contract. The objective of the project was to evaluate the security and the correct integration of the recent changes made to the rewards distribution system.

The audit was done in two stages. Both phases dealt specifically with the file `RewardsController.sol` of the git repository at https://github.com/exactly/protocol on the branch `next`.

The first part of the audit started on Feb 13, 2023 and was conducted at commit `76a605faf12faa4a18c3ce5dde0d5d6bbd676a27.` The second part started on Feb 23, 2023 and reviewed changes up until commit `d398c1414fba5f8bfc522797588c6e87b95b0650.`

The changes were introduced after Coinspect's latest audit on the rewards system and aim to provide enhancements, optimizations, and bug fixes. The Exactly Team provided a clear brief of each change and organized them in separate commits easing the review process.

A detailed list of the commits inspected both in the first and second stage follows. Commits not mentioned deal with changes out of scope, such as deployment configurations and changes to contracts other than `RewardsController.sol`.

First stage

The commit `120163937a5d6e28954c01fd3e6b645e8f933886` fixes **EXR-1** reverting if the indexes retrieved overflow. Also, introduces new changes that modify and rearrange data types on structs to optimize their slot packing.

On `4758613fdf613251c2a925a39dee055e2b81b59c`, Exactly modified the interface of `claim()` allowing users to specify which reward asset they are willing to claim. Previously, users were only allowed to call `claimAll().` This change aims to increase the flexibility and to improve the user experience of the rewards system. However, Coinspect identified that the change on this interface exposes the `update()` function to be external, **EXR-9**.

The commit `14a1ebbb5c52b1b854d052edb4a65258571909c4` makes the different operations (borrow and deposit) a boolean variable instead of using the old structs. Coinspect recommends keeping the structs as they are more descriptive and understandable than boolean variables that could be difficult to understand depending the context.

On commit `c3e00bfb34106a13aa1b05fba301659abcd8cd94` several view functions were refactored rearranging the return values of each one of them. The commit `6319be2899bd0e701faef74fab15c1f87bbf14dd` modifies the interface of the core function of rewards system, `previewAllocation()`, adding the `deltaTime` parameter. This newly added parameter allows externals to retrieve the rewards allocation at a specific point of time. It is worth mentioning that the remaining parts of the code that call `previewAllocation()` to retrieve the global indexes, calculate and pass the `deltaTime` considering the last update and the current timestamp.

Commit `dae0e0fb7878e34a8c63ea38cd77e8b3921c8eff` attempts to fix a setup bug related to the `config()` flow that required for modifications to be called with a set of unchanged + changed parameters, now allowing to specify an `end` without relying on calculated values. Also, this update adds in the system an additional nested level as each reward token of a distribution can have their own `start` and `end` dates independently. The accrual is now made for each different token of a distribution instead of targeting the market of a distribution.

The commit `500186d9a9c1cff17fb832d13ce7e9c208759a51` removes decimals for calculations of distribution rewards in favor of base units. On commit `2a8a97a157c495227835436910b310c8f3fbe8f3` the utilization rate was bounded to prevent a revert due to an underflow when calculating the sigmoid. For the changes made on that commit, Coinspect identified that the same treatment as the deposit allocation weight factor could be made for the transition factor to prevent breaking the reward calculation temporarily, **EXR-8**.

Second stage

Commit `a1136b980994eebb6b5a574b713964cb7ab0f2c9` changes the rewards calculations.

Commit `bd4bb24fa58586c21c80d7f50a25b2d7b2a3eca0` introduces support to update total distribution and the distribution period of a reward.

Commit `9bedec4abe1787460dd4f32684d720f592a827be` adds a safeguard to prevent issue **EXR-8**, reported in the first half of the assessment.

Commit `18e95f0ab9be5cb330def3d93b2934f39efd6b74` allows updating the release rate of rewards several times for the same distribution.

Coinspect identified two low risk issues related to the configuration process managed by the administrator. It is not possible to restart previous distributions for the same market and reward token because the `config()` function does not properly update the start variable under this condition, **EXR-11**. Also, because the rewards rate has a minimum value of 1, it won't be possible to start reasonable distributions for tokens with low decimals but high value (like WBTC), **EXR-12**.

Fixes review

The fixes were reviewed at commit `b84d181a98a89324cad02595c1e6af29a904379a` which fixed **EXR-11**. **EXR-8** had already been fixed on the second stage of the audit.

During this stage, **EXR-12** was reassessed as informational. It was previously considered `LOW` severity.

3. Summary of Findings

Id	Title	Total Risk	Fixed
EXR-8	Admin can inadvertently break rewards calculations	Low	✔
EXR-9	Update method is now public	Info	-
EXR-10	Inconsistent return types	Info	-
EXR-11	Wrong rewards when restarting distributions	Low	✔
EXR-12	Impossible to set reasonable reward rates for some tokens	Info	-

4. Detailed Findings

EXR-8 Admin can inadvertently break rewards calculations

Total Risk	Impact	Location
Low	Low	`RewardsController.sol`
Fixed ✔	Likelihood Low	

Description

Admin can accidentally break the reward calculations by setting invalid values on distributions. Exactly's team can restore functionality by using setters or upgrading the contracts.

Even though commit `2a8a97a157c495227835436910b310c8f3fbe8f3` introduces measures to prevent calculation on the sigmoid to revert because of division by zero or underflow, other values such as `transitionFactor` did not get the same treatment.

To continue the example of `transitionFactor`: if it is set to a number equal or bigger than `10**18`, the following line will cause a revert:

```
previewAllocation():
  ...
 int256(v.transitionFactor.divWadDown(1e18 - v.transitionFactor)).lnWad())) / 1e18).expWad()
 ...
```

Even though this would cause the system to be non-operational, the team can salvage the situation, as `config()` is available to admins. It is worth mentioning that if the admin is behind a timelock, the time to make this up will be restricted to the current execution waiting time.

Recommendation

Restrict the values of the transition factor so division by zero or an underflow are not possible.

Status

Fixed at `9bedec4abe1787460dd4f32684d720f592a827be`.

EXR-9	Update method is now public		
Total Risk **Info**	Impact -	Location -	
Fixed -	Likelihood -		

Description

Users might abuse the modified `claim()` method to call `update()` externally. This has no direct impact, but should be taken into account in future changes.

The `claim()` interface was modified to allow users to specify the rewards and actions performed that will be claimed. Because of this, it is now possible to call `claim()` passing an `ERC20` array with inconsistent addresses. This will not claim any rewards as the amount accrued for those tokens will be zero but will update the states of the `msg.sender` for the given operations and existing rewards.

Recommendation

Make sure calling `update()` with arbitrary data keeps being safe in the future.

Status

-

EXR-10 Inconsistent return types

Total Risk	Impact	Location
Info	-	-
Fixed	**Likelihood**	
-	-	

Description

The recent modifications made on data types to pack variables in fewer slots modified the accrued and index types of the Account struct: they are now `uint128`. However, the external `accountOperation()` function still returns `uint256`:

```
function accountOperation(
  address account,
  Market market,
  bool operation,
  ERC20 reward
) external view returns (uint256, uint256) {
  return (
    distribution[market].rewards[reward].accounts[account][operation].accrued,
    distribution[market].rewards[reward].accounts[account][operation].index
  );
}
```

Recommendation

Use consistent data types when returning, either change the interface or cast the variables.

Status

Exactly has stated that the current code leads to gas savings and thus will be kept.

EXR-11	Wrong rewards when restarting distributions		
Total Risk **Low**	Impact Low	Location RewardsController.sol	
Fixed ✔	Likelihood Medium		

Description

It is impossible to restart a distribution with the same period as the one that was active. When attempting to do so, the contract will silently update the previous distribution instead of starting a new one.

Contract admins are able to update configurations for existing distributions. They are also able to start new distributions for the same market and reward token.

Nevertheless, the contract will not restart a distribution that has already ended if the same distribution period is set for the new distribution. This can be seen in the following code block:

```
        } else if (configs[i].distributionPeriod != end - start) {
          rewardData.start = start = uint32(block.timestamp);
          rewardData.releaseRate = configs[i].totalDistribution.mulWadDown(1e18 /
configs[i].distributionPeriod);
          rewardData.lastConfigReleased = 0;
        }
```

At this point, `start` and `end` are the values of the previous distribution. If the distribution period of the new one is equal to the previous one, the comparison will be false, and `start`, `releaseRate` and `lastConfigReleased` will not be updated.

As `start` is not updated, `end` is also not updated:

```
      rewardData.end = start + uint32(configs[i].distributionPeriod);
```

All other values except for start, `releaseRate` and `lastConfigRelease` are updated. This means that now the older distribution is in an unexpected state, with some values updated but some not. **The most impactful of these discrepancies is that the total distribution is updated but the release rate is still set to the previous value.**

Recommendation

The admin should be able to explicitly differentiate the scenario when they want to update a previously ended distribution and to restart one with the same period.

A possible way to do this would be by adding a new value to the configurations which force the admin to explicitly state whether they want to restart a distribution with the same period or modify the previous one when updating a distribution that has already ended.

Status

Fixed at commit `b84d181a98a89324cad02595c1e6af29a904379a`. The configuration now has an explicit `start` parameter that the admin can set.

EXR-12	Impossible to set reasonable reward rates for some tokens		
Total Risk **Info**	Impact -	Location RewardsController.sol	
Fixed -	Likelihood -		

Description

The reward rate of distributions that involve certain tokens as rewards can not be set to a reasonable value. The tokens affected are those with a high monetary value but a low amount of decimals, such as WBTC.

This means that some distributions might be set to a reward rate of zero inadvertently, leaving users unable to claim from distributions that should be ongoing.

The reward rate will be calculated taking into account the distribution period and the amount to distribute:

```
    rewardData.releaseRate = configs[i].totalDistribution.mulWadDown(1e18 /
configs[i].distributionPeriod);
```

This calculation does not consider that its minimum value of 1 can be extremely valuable for some tokens. It makes it impossible to configure distributions that reward some amounts of highly-valued tokens.

WBTC is used as an example because Exactly already has a market for it. The issue nevertheless affects other popular tokens.

Proof of Concept

The following case shows a reward distribution setup using WBTC as reward token. For this proof of concept a getter for the release rate was added to `RewardsContoller`.

PoC

```
function test_DistWithValuableLowDecToken() external {
```

```
    vm.warp(0);
    RewardsController.Config[] memory configs = new RewardsController.Config[](1);
    configs[0] = RewardsController.Config({
      market: marketUSDC,
      reward: wbtc,
      priceFeed: MockPriceFeed(address(0)),
      targetDebt: 20_000e6,
      totalDistribution: 10e6, // 10 WBTC per year
      distributionPeriod: 52 weeks,
      undistributedFactor: 0.5e18,
      flipSpeed: 2e18,
      compensationFactor: 0.85e18,
      transitionFactor: 0.64e18,
      borrowAllocationWeightFactor: 0,
      depositAllocationWeightAddend: 0.02e18,
      depositAllocationWeightFactor: 0.01e18
    });
    rewardsController.config(configs);
    console.log('Rewards Rate: %s', rewardsController.getRate(marketUSDC, wbtc));
    wbtc.mint(address(rewardsController), 33e6);

    marketUSDC.deposit(1_000e6, address(this));
    marketUSDC.borrow(500e6, address(this), address(this));

    vm.warp(53 weeks);
    console.log("Claimable: %s", rewardsController.allClaimable(address(this), wbtc));
  }
```

Output

```
Rewards Rate: 0
Claimable: 0
```

For a period of 52 weeks (1 year), the approximate minimum distribution amount of WBTC needed to get a **rate of 1** (minimum rate that accumulates rewards) is `31.54 WBTC` per year ≈ `757,000 USD` with a WBTC @ 24,000 USD.

Recommendation

Consider using base units when calculating the rewards rate.

Status

The monetary risks of this issue were overstated due to a slight error in the test used as proof of concept. While the issue exists, it has been recategorized as informational. Exactly should exercise caution when using tokens with a low amount of decimals but high monetary value.

5. Disclaimer

The information presented in this document is provided "as is" and without warranty. The present security audit does not cover any off-chain systems or frontends that communicate with the contracts, nor the general operational security of the organization that developed the code.