Universal Reward Distributor

Security Review

Cantina Managed review by:
Saw-mon-and-Natalie, Lead Security Researcher **Jonah1005**, Lead Security Researcher **StErMi**, Security Researcher

November 14, 2023

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

[PROJECT DESCRIPTION HERE]

From Sep 28th - Oct 16th the Cantina team conducted a review of Universal Reward Distributor on commit hash 26388e...18e3bd. The team identified a total of **14** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 4
- Gas Optimizations: 2
- Informational: 8

3 Findings

3.1 Low Risk

3.1.1 The `URD` owner can always front run the `acceptRoot` execution

Severity: Low Risk

Context: UniversalRewardsDistributor.sol#L149-L158

Description: The `owner` of the `URD` contract can freely update the `timelock` state variable if the new value is greater (or equal) to the current one.

Given that the `timelock` has no lower or upper bound, this allows the `owner` to be able to frontrun the `acceptRoot` execution by setting the new `timelock` value equal to `timelock+1` or in general to a value that would make the `block.timestamp >= pendingRoot.submittedAt + timelock` requirement revert.

Recommendation: If such behavior is intended (given that the `owner` can anyway revoke the pending root by executing `revokeRoot`, even if the pending root could be accepted), Morpho should document extensively these edge cases.

Cantina:

The PR `https://github.com/morpho-org/universal-rewards-distributor/pull/94` addresses the issue. Now the timelock timestamp (when the pending proposal can be accepted) is embedded directly into the proposal struct and the `admin` can't influence the active proposal timelock duration.

3.1.2 `UniversalRewardsDistributor._setOwner` allows the owner to renounce the ownership, making the URD possibly useless

Severity: Low Risk

Context: UniversalRewardsDistributor.sol#L201-L205

Description: The current implementation of `_setOwner` allows the current owner to renounce to the ownership of the contract (passing `newOwner` equal to `address(0)`).

By allowing such logic, the URD could become "useless" (no rewards can be claimed):

- If no `updater` has been configured and the `root` is empty, no users would be allowed to claim rewards and the `root` cannot be updated in the future
- If no `updater` has been configured and the `root` is not empty, no one will be able to update the `root` to allow more rewards to be claimed

Recommendation: Morpho should consider disallowing the `owner` to renounce the ownership if at least one `updater` has not been configured.

Cantina:

Morpho has decided not to implement the recommendations but document the behavior in PR `https://github.com/morpho-org/universal-rewards-distributor/pull/85`

3.1.3 `UniversalRewardsDistributor.constructor` allows the deployment of a useless distributor

Severity: Low Risk

Context: UniversalRewardsDistributor.sol#L62-L74

Description: The current implementation of the `UniversalRewardsDistributor` contract does not perform any sanity checks during the execution of the `constructor`.

The `address initialOwner`, `uint256 initialTimelock`, `bytes32 initialRoot`, `bytes32 initialIpfsHash` input parameters of the `constructor` are not validated in either the `constructor` or in the respective setters function.

This allows the deployer to deploy a distributor that could have the following configuration

- `owner = address(0)`
- `root = bytes32(0)`

with such config

1) user can't claim anything because the `root` is empty
2) the `root` can't be updated because there's no owner that can call `setRoot(newRoot)`
3) no external actors can propose a new `root` via `submitRoot` because
    1) There's not an owner that can execute such function
    2) There's no `updater` that can execute such function (those are not configured during `constructor` time) and because there's no `owner`, it won't be possible to add new `updater`

Recommendation: Morpho should not allow the deployment of a `UniversalRewardsDistributor` construct that has the following configuration:

- `owner = address(0)`
- `root = bytes32(0)`

Morpho:

Cantina:

Morpho has decided not to implement the recommendations but document the behavior in the PR https://github.com/morpho-org/universal-rewards-distributor/pull/85.

3.1.4 The `URD` could introduce unexpected behaviors if deployed on chains that do not support `PUSH0` opcode

Severity: Low Risk

Context: UrdFactory.sol#L2, UniversalRewardsDistributor.sol#L2, foundry.toml

Description: The `PUSH0` opcode that has been introduced with `Solidity 0.8.20` is not currently supported by some of the major Layer 2 chains. Deploying contracts that have been built with such version could introduce unexpected behaviors.

Unlike other projects (included in the Morpho Blue ecosystem), the URD (Universal Permissionless Rewards Distributor) project does not enforce any solidity version or EVM version in the `foundry.toml` or `hardhthardhat.config.ts` (that is not present in the codebase).

Recommendation: If the project will be deployed on chains that do not support the features offered by Solidity `0.8.21`, Morpho should apply the following changes:

- Use Solidity `0.8.19` (or below)
- Set the EVM Version to `paris`

Cantina:

Morpho has decided to "downgrade" the solidity version of `UniversalRewardsDistributor` and `UrdFactory` to Solidity v0.8.19 in the PR https://github.com/morpho-org/universal-rewards-distributor/pull/86

3.2 Gas Optimization

3.2.1 `setTimelock(...)` can be optimised

Severity: Gas Optimization

Context:

- UniversalRewardsDistributor.sol#L152

Description/Recommendation:

One can cache `pendingRoot.submittedAt` and `timelock` (`timelock` can stay as is depending on which path you would want to optimise) to avoid reading from storage twice.

Morpho:

Cantina:

3.2.2 Consider refactoring `acceptRoot` to save gas and have a cleaner code

Severity: Gas Optimization

Context: UniversalRewardsDistributor.sol#L98-L103

Description: The `acceptRoot` function can be refactored with the result of having a cleaner code and avoiding performing 2 `SLOAD`

Recommendation: Morpho should consider the following changes to the `acceptRoot` function's code:

```
    function acceptRoot() external {
        require(pendingRoot.submittedAt > 0, ErrorsLib.NO_PENDING_ROOT);
        require(block.timestamp >= pendingRoot.submittedAt + timelock, ErrorsLib.TIMELOCK_NOT_EXPIRED);

-        root = pendingRoot.root;
-        ipfsHash = pendingRoot.ipfsHash;

-        emit EventsLib.RootSet(pendingRoot.root, pendingRoot.ipfsHash);

-        delete pendingRoot;
+        _setRoot(pendingRoot.root, pendingRoot.ipfsHash)
    }
```

Cantina:

The recommendations have been implemented in PR https://github.com/morpho-org/universal-rewards-distributor/pull/81

3.3 Informational

3.3.1 `onlyUpdater` modifier naming can be made more explicit

Severity: Informational

Context:

- UniversalRewardsDistributor.sol#L55

Description: `onlyUpdater` is defined as:

```
/// @notice Reverts if the caller is not the owner nor an updater.
modifier onlyUpdater() {
    require(isUpdater[msg.sender] || msg.sender == owner, ErrorsLib.CALLER_NOT_OWNER_OR_UPDATER);
    _;
}
```

Here both the `owner` and also any assigned `updater` can be allowed.

Recommendation: Rename this modifier to `onlyOwnerOrUpdater()` so that the naming would be more explicit.

Cantina:

The PR https://github.com/morpho-org/universal-rewards-distributor/pull/93 addresses the issue.

3.3.2 The maximum possible claimable amount for an account and reward token should be documented fully

Severity: Informational

Context:

- UniversalRewardsDistributor.sol#L113

Description: Fixing `account` a and `reward` token r. During the whole lifetime of the contract there might be multiple trees and each with multiple leaves such that the leaf corresponds to (a, r), the the maximum claimable amount for a of this r token would be:

$$C_{max}^{a,r} = \max c \mid L(a, r, c) \to T_i, i \in I$$

parameter	description
I	set of tree indices
T_i	tree with index i
$L(a, r, c)$	a leaf with correspond `account` a, `reward` token r and `claimable` c
$L \to T$	The leaf L belongs to the tree T
a	`account`
r	`reward` token
c	`claimable`

Note that:

- it's possible that a tree T might have two leaves (or more) $L_1(a, r, c_1)$ and $L_2(a, r, c_2)$.
- A user might not be able to claim $C_{max}^{a,r}$. It would depend how fast the user reacts to the changes in the `root`.

Recommendation: It might be useful to document the above peculiarities.

Morpho:

This limitation should be mentioned in the readme since the tree builder probably wants the user to claim the sum.

Cantina:

3.3.3 Define `_isPendingRootExpired()` to refactor internal logic

Severity: Informational

Context:

- UniversalRewardsDistributor.sol#L96
- UniversalRewardsDistributor.sol#L152

Description: In both `acceptRoot()` and `setTimelock(...)` the following condition is checked:

```
pendingRoot.submittedAt + timelock <= block.timestamp
```

Recommendation: It might make sense to define an internal function:

```
function _isPendingRootExpired() internal view returns (bool) {
   return pendingRoot.submittedAt + timelock <= block.timestamp;
}
```

to refactor the above mentioned logic in this context.

Cantina:

The PR https://github.com/morpho-org/universal-rewards-distributor/pull/94 addresses the issue. Now the timelock timestamp (when the pending proposal can be accepted) is embedded directly into the proposal struct and the `admin` can't influence the active proposal timelock duration.

3.3.4 Morpho should consider reverting the `claim` process with more specific revert messages to cover edge cases

Severity: Informational

Context: UniversalRewardsDistributor.sol#L113-L134

Description: The `claim` function could internally revert some explicit reason already covered by the `require` statements. Other than those reasons, there are also some implicit one that could happen and could be better monitored (via dApps/monitoring tools)

Let's assume that both the `require` present in the code are passed. This means that there's a configured `root` and that the `root` has verified that the user is entitled to receive at max `claimable` amount of the `reward` token (the final amount to receive depends on how many of that token they have already claimed).

- if `claimable < claimed[account][reward]` it would mean that the current configured `root` (that has been updated in the past) is miss configured. The process will revert to an underflow error when `claimable - claimed[account][reward]` is executed. Morpho could consider performing an explicit `require` statement and revert with a more meaningful `RootMissconfigured` error.
- If `ERC20(reward).balanceOf(address(this)) < claimable` it would mean that the `owner` (or who is responsible for the distribution of the rewards) have not correctly sent the needed amount of `reward` to the `URD`. The process will revert when `safeTransfer` is executed (and the URD has not enough balance to cover the operation). Morpho could consider performing an explicit `require` statement and revert with a more meaningful `NotEnoughFundsInUrd` error.

Recommendation: Morpho should consider performing more explicit `require` statement to cover the described edge cases and be able to revert with more meaningful error messages.

Cantina:

With the PR https://github.com/morpho-org/universal-rewards-distributor/pull/84 Morpho has implemented the first recommendation. Note that the now the `ErrorsLib.CLAIMABLE_TOO_LOW` will be thrown also when the user tries to claim a second time from the same `root`.

3.3.5 Consider documenting the scenario where it's allowed to set an empty distribution `root`

Severity: Informational

Context: UniversalRewardsDistributor.sol#L188-L198

Description: The `_setRoot` (called by `submitRoot` and `setRoot`) allows the caller to set the `root` state variable to an empty root. When the `root` is empty, no users will be able to claim any rewards.

Recommendation: If such behavior is allowed, Morpho should consider documenting it and document in which use case it should be allowed. If such behavior is not allowed, both `submitRoot` and `setRoot` should revert if the `newRoot` parameter is equal to `bytes32(0)`

Cantina:

Morpho has decided to document the behavior. The documentation can be found in the PR https://github.com/morpho-org/universal-rewards-distributor/pull/85.

An additional documentation to explain the use case of an empty root has been added in the commit https://github.com/morpho-org/universal-rewards-distributor/pull/85/commits/c9151dbfde7f04eb22f8fb65134ba56e5a368307

3.3.6 Consider renaming the function `revokeRoot` to `revokePendingRoot` and the event `RootRevoked` to `PendingRootRevoked`

Severity: Informational

Context: UniversalRewardsDistributor.sol#L173

Description: The `revokeRoot` function is not revoking the current `root` but instead is deleting (revoking) the `pendingRoot`.

Recommendation: The function `revokeRoot` should be renamed `revokePendingRoot` to be aligned with the function's logic. For the same reason, the event `RootRevoked` should be renamed to `PendingRootRevoked`.

Cantina:

The recommendations have been implemented in the PR `https://github.com/morpho-org/universal-rewards-distributor/pull/83`

3.3.7 `UniversalRewardsDistributor` setters should revert if the new value is equal to the current one

Severity: Informational

Context: UniversalRewardsDistributor.sol#L82-L89, UniversalRewardsDistributor.sol#L141-L143, UniversalRewardsDistributor.sol#L149-L157, UniversalRewardsDistributor.sol#L163-L167, UniversalRewardsDistributor.sol#L182-L184

Description: Morpho should follow the same approach already adopted in the other projects of the Morpho Blue ecosystem and revert if the caller is trying to submit (pending approval) or set a state variable with a value that is equal to the current one.

The check and revert should be added to the following functions:

- `submitRoot`
- `acceptRoot`
- `setRoot`
- `setTimelock`
- `setRootUpdater`
- `setOwner`

Recommendation: Morpho should follow the same approach already adopted in the other projects of the Morpho Blue ecosystem and revert if the caller is trying to submit (pending approval) or set a state variable with a value that is equal to the current one.

Cantina:

The recommendations have been implemented in the PR `https://github.com/morpho-org/universal-rewards-distributor/pull/92`

3.3.8 `UniversalRewardsDistributor.constructor` is not emitting all the possible events

Severity: Informational

Context: UniversalRewardsDistributor.sol#L71-L73

Description: During the execution of the `constructor` the `EventsLib.TimelockSet` and `EventsLib.RootSet` events are not emitted if the `initialTimelock` and `initialRoot` input parameters are equal to `0` and `bytes32(0)` respectively.

Morpho should consider emitting those events even if the values are equal to the default one to allow external entities (dApps, monitoring tools, ...) to correctly track the contract's lifecycle and state.

Recommendation: Morpho should always emit the `EventsLib.TimelockSet` and `EventsLib.RootSet` in the `constructor`.

Cantina:

The recommendations have been implemented in the PR https://github.com/morpho-org/universal-rewards-distributor/pull/82