PROPS Rewards Engine Contracts Audit

SEPTEMBER 2, 2019 | IN SECURITY AUDITS | BY OPENZEPPELIN SECURITY

The YouNow team asked us to review and audit their Rewards Engine contracts that distribute their PROPS token. We looked at the code and now publish our results.

The audited commit is e5ce0b2df1fbe108458d86820da578db56ac28d1 and the files included in scope were ERC865Token.sol, IERC865.sol, PropsRewards.sol, PropsRewardsLib.sol, PropsTimeBasedTransfers.sol, and PropsToken.sol.

This audit was a full audit covering the entire project. This audit was performed after the completion of the previous top-level audit. This audit encompassed the entire system as a whole, as well as a deep inspection of each contract individually.

Here are our assessment and recommendations, in order of importance.

Critical Severity

[C01] Any Selected Validator can Permanently DoS Reward Payouts

For a one-time upfront cost, any selected validator is capable of putting the contract into a permanently "stuck" state, where no further rewards can be paid out.

For any given `_rewardsDay`, anyone can precompute arbitrarily many "valid" `_rewardsHash`s (that is, hashes that will make the `_rewardsHashIsValid` function return true). In particular, one can keep the `_rewardsDay`, `_applications.length`, `_amounts.length`, and `_amounts` inputs fixed and vary the members of the `_applications` parameter. Each variation of the members of the `_applications` parameter will result in a new (valid) hash. Note that to produce a valid hash, the addresses in the `_applications` parameter do not need to be actual application addresses, and so the number of valid hashes that can be created this way is not bounded by the number of permutations of applications that have been set.

A malicious selected validator may call the `submitDailyRewards` function several times, passing in a distinct (valid) `_rewardsHash` each time, making sure the `_rewardsDay` value is always a valid one (that is, one that would not revert when passed to the `onlyValidRewardsDay` modifier).

During each of these calls, the `submitDailyRewards` function passes the `calculateApplicationRewards` function a new, distinct `_rewardsHash`. (Note that the `onlyOneRewardsHashPerValidator` modifier does not prevent this because it does not actually enforce that there is only one `_rewardsHash` per validator. It instead enforces that there is only one confirmation per validator per `_rewardsHash`.) The `calculateApplicationRewards` function, in turn, pushes the new `_rewardsHash` to the `rewardsLibData.dailyRewards.submittedRewardsHashes` array on line 218 of PropsRewardsLib.sol.

Since each of these calls uses a distinct `_rewardsHash`, and since the number of selected validators is expected to be at least 2, the conditional statement on line 224 of PropsRewardsLib.sol will never be true, and so the `calculateApplicationRewards` function will return `0`. This means that the conditional on line 181 of PropsRewards.sol will always be false, and so `_mintDailyRewardsForApps` will not be called.

Similarly, since the malicious selected validator is passing a unique `_rewardsHash` during each call of the `submitDailyRewards` function (and since the number of selected validators is assumed to be at least 2), the `calculateValidatorRewards` function will always return `0`, so the conditional on line 192 of PropsRewards.sol will always be false and `_mintDailyRewardsForValidators` will not be run.

The result is that the malicious selected validator can cause the `rewardsLibData.dailyRewards.submittedRewardsHashes` array to grow arbitrarily large by calling `submitDailyRewards` with distinct `_rewardsHash` values.

This is problematic because paying out rewards requires a call to the `_resetDailyRewards` function, which both iterates over and attempts to delete the `rewardsLibData.dailyRewards.submittedRewardsHashes` array. So if the malicious selected validator makes the `submittedRewardsHashes` array large enough, any future attempt to payout rewards will fail due to hitting the block gas limit.

One possible solution would be to follow a pattern like the one shown in this StackExchange answer, where the array is not deleted but instead has its values overwritten.

Update: Fixed in 9ab4ec4. The `_resetDailyRewards` function is no longer iterating the `submittedRewardsHashes` array which was at risk of being inflated by a malicious attacker.

High Severity

No issues of high severity were found

Medium Severity

[M01] The selectedApplications array is not enforced

The `currentList` and `previousList` arrays in the `selectedApplications` struct are controlled by the `controller` and are used to set the list of applications that are able to receive rewards. However, these arrays are never checked when rewards are being distributed. It is possible for validators to distribute rewards to applications that are not included in the `selectedApplications` arrays. Consider requiring that all applications are included in the appropriate array in the `selectedApplications` struct for a `rewardsHash` to be valid.

Update: Fixed in 9ab4ec4. A check was added to the `_rewardsHashIsValid` function enforcing that an application is in the appropriate array in the `selectedApplications` struct.

Low Severity

[L01] Improper use of uint256 type

The `uint256` type is misused in the following locations:

In the `Submission` struct located in the `PropsRewardsLib` library, the `uint256` type is used for the `finalized` parameter. This parameter is only ever set to 0 or 1 and is intended to represent whether a submission has been finalized or not. Consider using the `bool` type to increase readability and prevent confusion regarding the intended use of the parameter.

The `_getSelectedRewardedEntityListType` function in the `PropsRewardsLib` library returns a `0` if the `currentList` should be used and a `1` if the `previousList` should be used. For increased clarity, consider creating a `RewardedEntityListType` enum that can be returned instead or renaming the function to something like `_usePreviousSelectedRewardedEntityList` and returning a `bool`.

Update: Partial fix in 9ab4ec4. In the `Submission` struct, the YouNow team renamed `finalized` to `finalizedStatus` but chose not change the type because other statuses may be added in the future. The YouNow team pointed out that using a `bool` type until more statuses are added in a contract upgrade would require the `bool` `finalizedStatus` to be deprecated and a new `uint256` parameter to be added. The `_getSelectedRewardedEntityListType` function was renamed `_usePreviousSelectedRewardsEntityList` and now returns a `bool`.

[L02] updateController Accepts the Zero Address

The `updateController` function in `PropsRewards.sol` accepts the zero address. If the controller mistakenly calls this function without passing an address, the zero address will become the new controller.

To protect the controller from mistakenly burning their privileges, considering using a `require` statement to ensure that the `_controller` parameter is non-zero.

Resolved in commit 9ab4ec4.

Notes & Additional Information

- There are some minor differences between the intended behavior as described in the specification and the actual behavior of the contract. We list those here. The specification says:

  > "Every day, Validators submit data to the Rewards Engine, representing how much each of the active Applications should receive for the previous day [...] Data must be submitted before the end of the next day"

  This is not enforced in the contract. For example, validators could (collectively) decide not to call the `submitDailyRewards` function for any number of days (say, 6 months) and then, in a single day, call `submitDailyRewards` once for each `rewardsDay` in that 6 month period.

  If they are careful (ie: they make sure to reach consensus about day `N` before reaching consensus about day `N+1`), then they'll be able to collect rewards for every day of those 6 months.

  The specification says:

  > "Once all Validators have submitted, rewards are paid to the Validators"

  This is only true if consensus was reached. If (for example) each validator submitted a different hash, then the validators will not be paid — even though all of them have submitted. More than half of all validators must submit the same hash for any of the validators to get paid.

The specification says:

> "If one or more Validators does not submit, rewards are paid when the first Validator submits on the following day"

This may not occur on the following day if, for example, no validator calls the `submitDailyRewards` function.

If the above contract behavior is correct, consider updating the specification documentation to match.

The PROPS team noted that "the spec will be updated."

- The Dependencies section of the README does not explicitly state which `ganache-cli` version to install. `ganache-cli` versions 6.4.2, 6.4.3, and 6.4.4 have a gas estimation bug that causes the tests to fail. Consider being explicit when stating dependencies, and use `ganache-cli` versions <=6.4.1 or >=6.4.5.*Resolved in commit 9ab4ec4.*
- The Test section of the README does not provide accurate instructions for the current state of the project, as `./test/propstoken.test.js` is not a valid file path. Update the manual test line in the README to match the `npm run test` script, which is `NODE_ENV=test truffle test ./test/*.test.js –network test`.*Resolved in commit 9ab4ec4.*
- The specification says:

  > "Every day, Validators submit data to the Rewards Engine, representing how much each of the active Applications should receive for the previous day [...] The data is hashed, to compare between validators"

  It is possible that the validators may agree on the set of applications and the amount of their rewards, but may disagree on which hash to submit when calling the `submitDailyRewards` function.

  In particular, all validators may agree on the set of applications and amounts, but each of them may list the applications and amounts in a different order when calling the `submitDailyRewards` function. In this case, each submitted hash will be different and consensus will not be reached at the contract level.

  It is important that validators agree not just on the applications and amounts, but also on how to order them when computing the corresponding hash. Consider having some canonical ordering of application addresses (for example, alphabetical order). This does not need to be enforced at the contract level, and would not require any contract changes.

  The PROPS team noted that "the spec will be updated and instructions for validators will be given with their software."
- The `onlyOneRewardsHashPerValidator` modifier does not prevent a given validator from submitting more than one distinct `_rewardsHash`. The modifier prevents a given validator from confirming the same `_rewardsHash` more than once. Consider renaming this modifier to `onlyOneConfirmationPerValidatorPerRewardsHash`.*Resolved in commit 9ab4ec4.*
- Good naming is one of the keys for readable code, and to make the intention of the code clear for future changes. There are some names in the code that make it confusing or hard to understand.Consider the following suggestions:
  - Rename `Data.DailyRewards.lastRewardsDay` to `Data.DailyRewards.lastApplicationsRewardsDay`
  - Rename `Data.lastRewardsDay` to `Data.lastValidatorsRewardsDay`
  - Rename `onlyOneRewardsHashPerValidator` to `onlyOneConfirmationPerValidatorPerRewardsHash` (See note above)
  - Rename `calculateApplicationRewards` to `calculateAndFinalizeApplicationRewards`

  Resolved in commit 9ab4ec4.
- On Line 161 of `PropsRewards.sol` the conditional statement checks that both `_rewardsDay > 0` and `_rewardsDay > rewardsLibData.dailyRewards.lastRewardsDay`. Note that if the second half of the conditional is true (that is, if `_rewardsDay > rewardsLibData.dailyRewards.lastRewardsDay`) then the first half is also true. Also note that if the second half of the conditional is false, then the entire conditional will fail whether or not the first half is true. This implies that the first half of the conditional is redundant.Consider simplifying the conditional to `if (_rewardsDay > rewardsLibData.dailyRewards.lastRewardsDay)`.*Resolved in commit 9ab4ec4.*

- In the `PropsRewards` contract the natspec comment for the `_applications` parameter of the `setApplications` function mistakenly says "address[] array of validators". Consider updating this comment to say "address[] array of applications".*Resolved in commit 9ab4ec4*.
- In the Props Token tests, there are many interactions between tests. Interactions between tests make it difficult to reason about the current state of the contracts, make changes to individual tests, run a single test, and may lead to false negatives and/or positives in the test suite. Consider setting up the contracts for each individual test rather than for each test file.*The PROPS team noted that this "will be updated for the next iteration of the contract."*
- Multiple functions in the `PropsRewards` contract and the `PropsRewardsLib` library return a boolean value unnecessarily. For example, the `_finalizeDailyApplicationRewards` function, will always return true and the return value is never checked. Consider removing all unnecessary return values.*Resolved in commit 06c5cc1*.
- The `_decimals` parameter passed into `_initializePostRewardsUpgrade1()` in `PropsRewards.sol` is of type `uint256`, yet it is recasted to type `uint256` on line 308. Consider removing this unnecessary typecast to `uint256`.*Resolved in commit 9ab4ec4*.
- The `ganache-cli` npm script creates only 10 accounts, but the tests require at least 45 accounts to be created. Add the `-a` flag to the `npm run ganache-cli` script with the appropriate number of accounts required to run the tests successfully.*Resolved in commit 9ab4ec4*.

Conclusion

1 critical and 0 high severity issues were found. Several changes were proposed to follow best practices and reduce the potential attack surface.

> *Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the PROPS Token's contracts. The above should not be construed as investment advice. For general information about smart contract security, check out our thoughts here*

Security Audits

- If you are interested in smart contract security, you can continue the discussion in our forum, or even better, join the team 🚀
- If you are building a project of your own and would like to request a security audit, please do so here.

RELATED POSTS

SECURITY AUDITS

E
x
p
l

START THE DISCUSSION AT FORUM.OPENZEPPELIN.COM