SOLIDIFIED

Audit Report for Passage Labs, Inc. - October 7, 2022

Summary

Audit Report prepared by Solidified covering the Passage Protocol.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on September 19, 2022, and the results are presented here.

Audited Files

The source code has been supplied in the following source code repository:

Repo: https://github.com/passage-protocol/toll-booth/
Commit hash: `9eb89f53f2feceea69aa7227986062f278fd0757`

Fixes received in commit `e4bafd7a4b8e79213c9042afcd1cbbb85b73757e`

```
contracts
├── ChainedResolver.sol
├── ERC721Plugin.sol
├── Membership.sol
├── SumResolver.sol
├── SuperfluidPlugin.sol
├── TextMetadata.sol
├── ThresholdResolver.sol
├── Tollbooth.sol
├── TrialPlugin.sol
├── UriMetadata.sol
├── factory
│   ├── MembershipFactory.sol
│   ├── SubscriptionFactory.sol
│   ├── SuperfluidPluginFactory.sol
│   └── TrialPluginFactory.sol
├── interfaces
│   ├── IEvents.sol
│   ├── IMembership.sol
│   ├── IMembershipFactory.sol
│    ├── IMetadata.sol
│    ├── IOwnable.sol
│    ├── IPlugin.sol
│    ├── IResolver.sol
│    ├── ISuperfluidPluginFactory.sol
│    ├── ITollbooth.sol
│    └── ITrialPluginFactory.sol
└── test
     ├── TestERC1155.sol
     ├── TestERC20.sol
     └── TestERC721.sol
```

Intended Behavior

The audited codebase implements a generic framework for web3 memberships that can be used with multiple plugins (e.g., a Superfluid plugin to integrate with the payment streaming protocol).

Findings

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	Low	-
Code readability and clarity	High	-
Level of Documentation	Medium	-
Test Coverage	High	-

Issues Found

Solidified found that the Passage protocol contracts contain no critical issues, one major issue, 7 minor issues, and 0 informational notes.

We recommend issues are amended, while informational notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	SuperfluidPlugin.sol: afterAgreementTerminated may revert or use too much gas	Major	Resolved
2	TrialPlugin.sol: redeem can shorten trial duration	Minor	Acknowledged
3	SuperfluidPlugin.sol: Duplicate entries in tokens possible	Minor	Resolved
4	SuperfluidPlugin.sol: MAX_TOKENS in constructor not enforced	Minor	Resolved
5	TrialPlugin.sol: Untyped data signing in redeem	Minor	Resolved
6	TextMetadata.sol: attributes instead of properties used	Minor	Resolved
7	SuperfluidPlugin.sol: feeReceiver updates only reflected on next flow update	Minor	Acknowledged
8	TrialPlugin.sol: Trial runs until Membership.update(user) is called	Minor	Acknowledged

Critical Issues

No critical issues have been found.

Major Issues

1. SuperfluidPlugin.sol: afterAgreementTerminated may revert or use too much gas

According to the Superfluid documentation, Super Apps are not allowed to revert in the `afterAgreementTerminated` callback. However, `_update` is called in the callback, which calls `Membership.update`. This function iterates over all classes and calls `shares` on all plugins. It can happen that one of those plugins reverts, which would cause the super app to be jailed. Similarly, there is an upper gas limit for the callback, which could also be exceeded when too many classes or plugins are used.

Recommendation

Either catch reversions and ensure that the gas limit will never be exceeded or introduce some other mechanism for updating in these cases (e.g., marking the account as dirty and performing the update in the `Membership._membership` function).

Minor Issues

2. TrialPlugin.sol: redeem can shorten trial duration

It is possible to redeem a signature for a free trial, even if the user has already called `enroll` and is currently in a trial period (as long as the user has a signature where the end time matches his current end time). This may be itself undesirable, but it can also lead to situations where the `redeem` call actually shortens the trial duration. Let's say that the usual `duration` (when calling `redeem`) is 1 week, but an operator has decided to grant the user an additional day. When the user calls `redeem` with this signature in the beginning of his trial period, it will be shortened, as the new end is now `block.timestamp + 1 day`.

Recommendation
Handle the case when redeem is called for a user that is currently in a trial period. Depending on the specifications, this should not be allowed, or the trial period should be extended by the provided `duration_`.

Status
Acknowledged: "This behavior is as designed. The `redeem` method is intended to be used for flexible trial signatures, where users can redeem whenever ready. If the controller needs to assign an explicit end time, the `credit` method can be used directly. Overall, we believe any potential confusion on trial mechanics is best addressed at the UX level"

3. `SuperfluidPlugin.sol`: Duplicate entries in `tokens` possible

While it is verified in `setToken` that the token does not exist already, this check is not performed in the constructor and it is therefore possible to instantiate the plugin with duplicate entries in `tokens`.

Recommendation
Check in the constructor that `pclasses[tokens_[i]] == 0` before adding a token to the list.

4. `SuperfluidPlugin.sol`: `MAX_TOKENS` in constructor not enforced

The upper limit for the tokens is only enforced in `setToken`, but not in the constructor, meaning that it is possible to have more than `MAX_TOKENS` tokens.

Recommendation

Enforce the limit in the constructor.

5. TrialPlugin.sol: Untyped data signing in redeem

The signatures that are passed to redeem are untyped, i.e. raw data is directly hashed. This can cause multiple vulnerabilities:

- All signatures are replayable on chains with different chain IDs.
- A signature that was never determined for the application can still be used, as long as the data matches. For instance, it could happen that the controller signed the data `(msg.sender, uint, uint, 0)` for some other applications, but the signature would still be redeemable within the Passage Protocol.

Recommendation

Implement EIP-712.

6. TextMetadata.sol: attributes instead of properties used

According to EIP-1155, additional properties and metadata should be in the `properties` array, but the array is currently indexed with `attributes`.

Recommendation

Use `properties` instead of `attributes` as key.

7. SuperfluidPlugin.sol: feeReceiver updates only reflected on next flow update

When the fee receiver is updated (`passage.tollbooth.plugin.superfluid.feeReceiver`), the flow to the fee receiver will only be changed when a user updates a flow. This contrasts with updates of `receiver`, which are immediately.

Recommendation

If this is not intended, consider introducing an authorized function to redirect the fee flow after an update.

Status

Acknowledged: “This is as designed. We prefer the passive update of fee settings for simplicity and also for scalable updates across many membership programs.”

8. TrialPlugin.sol: Trial runs until Membership.update(user) is called

The balances of a user are only updated when `Membership.update(user)` is called. For the Superfluid plugin this works well because the function is called whenever a flow is changed. However, when the trial plugin is used, this has to be called explicitly, otherwise the user will continue to have the shares of the trial, even if it is expired. In a situation where a user is in a trial period, but also has a signature that he can redeem for another free trial, this also introduces a front-running opportunity: The user waits until `update` is called with his address. He then front-runs this call with a call to redeem. This procedure allows him to stretch the duration of the trial.

Recommendation

Consider incorporating the fact that certain plugins are time dependent (e.g., by flagging them) and query the shares for them on every call to `membership(user, class)`.

Status

Acknowledged: “This is as designed. The membership class actually does already query the underlying plugin data on every call. The only difference between these “active” vs “passive” plugin types is that minting and burning events will not be emitted immediately. In the future, we plan to have a keeper system for lively updates.”

Informational Notes

No additional notes

Disclaimer