SOLIDIFIED

Audit Report for Passage Protocol Memberships - December 10, 2022

Summary

Audit Report prepared by Solidified covering Passage Protocol's membership contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on November 18, 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/passport-contracts-2
Commit hash: `7dc0c7dd0193a15b5fd9f245addd843a79033097`

```
contracts
├── LoyaltyLedger2.sol
├── PassageRegistry2.sol
├── PassageUtils.sol
├── Passport2.sol
├── factory
│   └── PassagePresetFactory.sol
├── interfaces
│   ├── ILoyaltyLedger2.sol
│   ├── IPassageAccess.sol
│   ├── IPassageRegistry2.sol
│   ├── IPassport2.sol
│   └── modules
│       ├── beforeTransfer
│       │   ├── I1155BeforeTransfersModule.sol
│       │   └── I721BeforeTransfersModule.sol
│       ├── minting
│       │   └── IMintingModule.sol
│       └── render
│           └── IRenderModule.sol
├── lib
```

```
│   ├── ERC2771Recipient.sol
│   ├── Ownable.sol
│   └── PassageAccess.sol
├── modules
│   ├── beforeTransfer
│   │   ├── NonTransferable1155Module.sol
│   │   └── NonTransferable721Module.sol
│   └── minting
│       ├── BaseMintModule.sol
│       ├── BaseMintModuleCloneable.sol
│       ├── ClaimList.sol
│       ├── MintsPerAddress.sol
│       ├── MintsPerClaim.sol
│       ├── MintsPerSingleClaim.sol
│       ├── PermissionedMint.sol
│       └── PricedMint.sol
└── test
    ├── Test1155BeforeTransferModule.sol
    ├── TestBadMintingModule.sol
    ├── TestBeforeTransferModule.sol
    ├── TestCloneFactory.sol
    ├── TestERC721.sol
    ├── TestLoyaltyLedger2Upgraded.sol
    ├── TestPassageRegistry2Upgraded.sol
    ├── TestPassport2Upgraded.sol
    └── TestRenderModule.sol
```

Intended Behavior

The code base implements generic, extensible membership NFTs for the Passage 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 Membership contracts contain no critical issues, 1 major issue, 3 minor issues, and 3 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	PassageRegistry2.sol: Increasing version numbers not enforced	Major	Resolved
2	ClaimList.sol: User can only claim once	Minor	Acknowledged
3	transfer used for sending ETH	Minor	Resolved
4	LoyaltyLedger2.createToken: Name can be empty	Minor	Resolved
5	PassageRegistry2.sol: Duplicated code in modifier onlyFromManagedPassport	Note	Resolved
6	Incorrect comments throughout the code	Note	Resolved
7	PassagePresetFactory.sol: Minting module name can be empty	Note	Resolved

Critical Issues

No critical issues have been found.

Major Issues

1. `PassageRegistry2.sol`: Increasing version numbers not enforced

The position of an implementation within the `passportImplementations` or `loyaltyImplementations` array is used as the version number when upgrading the passport or loyalty implementation. However, when a new implementation is added to the array, it is not checked that `passportVersion` / `loyaltyLedgerVersion` also return this version. If this is not the case, upgrades become impossible, because `upgradePassport` / `upgradeLoyalty` validate the returned version.
For instance, a passport implementation that returns 0 for `passportVersion` is added as the second element (with index 1) to `passportImplementations`. The call `upgrade(1, address(passport))` will succeed because `1 == passport.passportVersion() + 1`. However, if another passport implementation is added to `passportImplementations` and this one returns 2 for `passportVersion`, the call to `upgrade(2, address(passport))` will fail because `1 == passport.passportVersion() + 1`. It is not possible to recover from such a scenario, adding another implementation can not resolve the problem.

Recommendation
When adding new implementations, enforce that the returned version (`passportVersion` / `loyaltyLedgerVersion`) matches the array index.

Minor Issues

2. `ClaimList.sol`: User can only claim once

In `ClaimList.canMint()`, `claimlistClaimed[minter]` is set to true after the first mint of a user. However, it can happen that a user does not claim the maximum allowed amount (maxAmount) in one transaction and tries to claim the rest later on. This will fail with the current implementation.

Recommendation
Consider storing the amount that was claimed and allowing claims when the requested amount is smaller than the maximum amount minus the already claimed amount.

Status
Acknowledged: “The functionality of only allowing a user to claim once is intentional. A user can claim up to the maximum amount, but only has 1 transaction to do so.”

3. `transfer` used for sending ETH

In various places (`LoyaltyLedger2.withdraw`, `Passport2.withdraw`, `PermissionedMint.withdraw`), the `transfer` function is used to transfer ETH. Because `transfer` only comes with a 2300 gas stipend, its use is discouraged. If the receiver is for instance a multi-sig wallet with some business logic in its `receive` function, the transaction will revert.

Recommendation
Use `call` instead of `transfer`.

4. LoyaltyLedger2.createToken: Name can be empty

It is possible to create a token with an empty name in `LoyaltyLedger2.createToken`. However, because the length of the name is used to check for the existence of a token (in `exists`), this will lead to undesirable behavior: A token with an empty name gets an ID, but no actions (setting the maximum supply, minting, querying the base URI, …) are possible for this token. The ID is therefore wasted and the valid IDs are no longer consecutive.

Recommendation
Enforce that the length of the name is not zero.

Informational Notes

5. PassageRegistry2.sol: Duplicated code in modifier onlyFromManagedPassport

The modifier `onlyFromManagedPassport` performs the same check twice:
`require(managedPassports[_msgSender()] || managedPassports[_msgSender()], "R1");`

Recommendation
Remove the duplicate check to save gas.

6. Incorrect comments throughout the code

The tokenParameters comment about createLoyalty in PassagePresetFactory states that there is an array of tokens when there is not.

The comment above `mintPassports` in `Passport2` is missing a comment for the `_amounts` parameter.

Recommendation

Update the comments to reflect the current state of the code.

7. `PassagePresetFactory.sol`: Minting module name can be empty

It is possible to add a minting module to the array in `PassagePresetFactory.sol`**.** Although it does not interfere with the functionality, requiring that it is not an empty string might still make sense, since an empty string sounds odd.

Recommendation

Enforce that the name of the minting module is not zero.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Passport Protocol or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Oak Security GmbH