Persistence Security audit

Author: Vu Duc Hieu, Robin Tunley

08 / 13 / 2023

contact@notional.ventures

notional.ventures

github.com/notional-labs

Table of contents

Disclaimer

This audit has been performed by

Notional Labs

https://notional.ventures/
contact@notional.ventures

Introduction

Notional Labs has been engaged by Persistence to perform a security audit on (1) the codebase for the Persistence One blockchain and (2) the Version 8 upgrade for this chain. We have been instructed to include the following components:

1. A comprehensive report outlining the findings of the code review and auditing process, including a summary of any vulnerabilities, weaknesses or other material issues identified.
2. A list of recommendations for remediation and improvement of any identified issues.
3. A detailed list of the testing procedures used, including any test cases and results.
4. A list of any additional tools or resources that may be required

These will be presented within this audit.

Overview of Persistence One

The complete documentation for the Persistence One software was reviewed by our team, to ensure our full understanding of the functionality of the codebase. A brief overview is presented here.

Persistence is an appChain focused on Liquid Staking Finance (LSTFi). Using the various dapps on the Persistence chain, starting with pStake and Dexter, users can access various DeFi opportunities using liquid staked tokens. To facilitate these use cases, the team is using forks of the cosmos-SDK, ibc-go, and wasmd repositories. These forks are all compatible with version 47 on the Cosmos SDK, including the use of the Liquid Staking Module (LSM).

One thing that our team noticed immediately upon reviewing the repositories is that unlike other Cosmos projects, the Persistence team has broken down the chain's functionality into multiple small repositories to accommodate development teams in different parts of the world. This organization makes it easier for teams to manage their work, and avoid possible merge conflicts. While we worry that this organization of the code may make it difficult to manage version control in the future, we can appreciate the efficiencies that the Persistence team has derived from it.

Codebase

The repositories that we reviewed in the above section are listed here for reference:

- https://github.com/persistenceOne/cosmos-sdk/releases/tag/v0.47.3-lsm2
- https://github.com/persistenceOne/ibc-go/releases/tag/v7.2.0-lsm2
- https://github.com/persistenceOne/wasmd/tree/v0.40.2-lsm2
- https://github.com/persistenceOne/persistence-sdk (Commit hash: 0dc2eba7b381a7a4aa7c4f6fc60ea8041f900883)
- https://github.com/persistenceOne/pstake-native (Commit hash: 4537d4f134d26e3e6cebf12e82aec5805bf3ae1d)
- https://github.com/persistenceOne/persistenceCore/releases/tag/v7.0.2
- https://github.com/persistenceOne/persistenceCore/blob/main/app/upgrades/v8/upgrades.go

Methodology and Tools

The audit was performed in three primary stages. Stage 1 and 2 were focused on the current codebase, while Stage 3 was focused on the version 8 upgrade.

Stage 1 utilized the automated testing provided by the Persistence Team. First, the unit tests of each repository were analyzed to ensure that all critical edge-cases were covered. These tests were performed on source code to ensure that all pass without error. Second, all integration tests provided by Persistence were run to verify that no errors occurred and that our results match theirs.

Stage 2 involved manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines across all repositories. The single critical and most major issues discovered by our team were revealed during this process.

Finally, in Stage 3, we needed to verify that the upgrade to version 8 would be seamless. This upgrade will include integration with SDK version 47, the addition of new modules like the LSM, and some edits to the config parameters in the Staking module.

A successful upgrade amounts to ensuring that the blockchain data will migrate from the previous version 7 to version 8, which is done via the upgrade handler. These tests were performed manually. First, the ExportGenesis function was used to export the state of the current Persistence mainnet from our own node. Then, the InitGenesis function was used to simulate a new mainnet with that seed data. This upgrade was performed successfully without errors.

Summary of Findings

Overview of Codebase

Upon completion of our audit, we have had the opportunity to reflect on the progress that has been made by the Persistence team. Before covering the issues uncovered by our audit, we present a short summary of the major repositories reviewed - their features, and thoughts on their development.

The pstake-native repository includes the primary application logic. Besides the calculation issue mentioned below, we feel it is very well-written. We also noticed that over the course of this audit, the testing for this repository has been greatly improved, particularly for the keeper and the message server.

The persistence-SDK repository contains several key modules and supports a collection of on-chain timers, which allow for actions to be executed on-chain at predetermined points in the future. It also allows for interchain queries and manages the inflation of the native token.

Finally, persistence-core is the primary appChain repository. It contains 'app.go', config files, and upgrade handlers. Overall, we found these repositories easy to read, and we believe their on-going maintenance should prove extremely manageable, even in the process of onboarding new developers.

Classification of Issues

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following:

- **Pending** - the root of this issue still being determined
- **Acknowledged** - this issue will be fixed in a future version
- **Resolved** - this issue has already been fixed

Overview of Issues

Overall, we detected one critical issue and three major issues which required immediate attention. We also found two minor and three informational issues of much lower severity. A brief summary of our findings has been tabulated using the legend defined in the Classification of Findings section above. This is intended as a surface level overview only, with more detailed information and recommendations to follow in the next section.

No	Repo found	Description	Severity	Status
1	pstake-native	Wrong calculation in msg Redeem can be exploited to change protocol	**Critical**	**Resolved**
2	persistence-sdk	Missing IBC Router for IBC Packet Forward Middleware	**Major**	**Resolved**
3	pstake-native	Wrong comparison between totalDelegations and totalUnbondings	**Major**	**Resolved**
4	persistence-sdk	MakeRequest is missing connectionId & chainId valid check	**Minor**	**Acknowledged**
5	pstake-native	Declare variable multiple times	**Minor**	**Acknowledged**
6	pstake-native	Duplicate coin variable	**Informational**	**Resolved**
7	pstake-native	Lack test on msgServer in liquidstakeibc module	**Informational**	**Resolved**
8	persistence-sdk	Missing specific type of ICQ events for cosmos/relayer	**Informational**	**Acknowledged**

Issue Details and Recommendations

Here we discuss each finding in more detail, and present possible solutions to the issues discovered. We begin with the critical issue found, finishing with the informational ones.

Critical Issues

1. An incorrect calculation in the Redeem message can be exploited

This issue is found in the `pstake-native` repository within the `liquidstakeibc` module. There is a msgServer function called `Redeem` which defines a method for instantly redeeming liquid staked tokens. Where, `msg.Amount` is the parameter relating to the amount of staked tokens the user wants to use to redeem the IBC token.

In the function, the variable `stkAmount` indicates the actual amount of tokens used to redeem after deducting the fee. The amount of IBC tokens the user receives is calculated from this variable according to the formula:

```
redeemAmount := sdktypes.NewDecCoinFromCoin(stkAmount).Amount.Mul(hc.CValue)
```

In which, hc.CValue is calculated by:

```
hc.CValue := (total stk tokens minted) / (total ibc tokens amount staked)
```

From the formula, we can see that `redeemAmount` is being miscalculated. It should have been `Quo` calculation instead of `Mul`. This error causes the amount of IBC tokens to be redeemed by users to be less than expected. In addition, changing the balance of `DepositModuleAccount` incorrectly leads to an unexpected change in `CValue`. An attacker could potentially profit from this error.

Recommendation

Change the above calculation to:

```
redeemAmount := sdktypes.NewDecCoinFromCoin(stkAmount).Amount.Quo(hc.CValue)
```

Status: Resolved

This issue was resolved in the commit:
https://github.com/persistenceOne/pstake-native/commit/676b6e7ccc097d0505fd110cee66ca4c7d85ba5d

Major Issues

2. Missing IBC Router for IBC Packet Forward Middleware

We found this issue in commit:
https://github.com/persistenceOne/persistence-sdk/blob/0dc2eba7b381a7a4aa7c4f6fc60ea8041f900883/simapp/keepers/keepers.go#L411-L423

Within it, appKeepers.RouterKeeper is defined but the IBC Packet Forward Middleware module was not added to IBC Router. As a result, this module will not work as intended.

```
	hooksICS4Wrapper := ibchooks.NewICS4Middleware(
		appKeepers.IBCKeeper.ChannelKeeper,
		appKeepers.Ics20WasmHooks,
	)

	appKeepers.HooksICS4Wrapper = &hooksICS4Wrapper

	// Hooks Middleware
	transferIBCModule := ibctransfer.NewIBCModule(*appKeepers.TransferKeeper)

	hooksTransferStack := ibchooks.NewIBCMiddleware(&transferIBCModule,
appKeepers.HooksICS4Wrapper)
```

Recommendation

To resolve this, packetForwardMiddleware needs to be added to the IBC Transfer stack hooksTransferStack.

```
// Hooks Middleware
	transferIBCModule := ibctransfer.NewIBCModule(*appKeepers.TransferKeeper)

	packetForwardMiddleware := router.NewIBCMiddleware(
		transferIBCModule,
		appKeepers.RouterKeeper,
		0,
		routerkeeper.DefaultForwardTransferPacketTimeoutTimestamp,
		routerkeeper.DefaultRefundTransferPacketTimeoutTimestamp,
	)

	hooksTransferStack := ibchooks.NewIBCMiddleware(&packetForwardMiddleware,
appKeepers.HooksICS4Wrapper)

	appKeepers.TransferStack = &hooksTransferStack
```

The variable routerKeeper was moved such that it is defined before IBC transfer stack.

```
routerKeeper := routerkeeper.NewKeeper(
		appCodec,
		appKeepers.keys[routertypes.StoreKey],
                    appKeepers.GetSubspace(routertypes.ModuleName),
                    appKeepers.TransferKeeper,
                    appKeepers.IBCKeeper.ChannelKeeper,
                    appKeepers.DistributionKeeper,
                    appKeepers.BankKeeper,
                    appKeepers.IBCKeeper.ChannelKeeper,
          )
appKeepers.RouterKeeper = routerKeeper
```

Status: Resolved

Issue resolved by commit:
https://github.com/persistenceOne/persistence-sdk/pull/405/commits/0f0f484e01a46d8cb27ae1346c0a1cb3a2f1af23

3. Wrong comparison between totalDelegations and totalUnbondings

We found this issue in commit:
https://github.com/persistenceOne/pstake-native/blob/4537d4f134d26e3e6cebf12e82aec5805bf3ae1d/x/liquidstakeibc/keeper/msg_server.go#L489-L500

In the code, we have a comparison:

```
if totalDelegations.LT(unbondAmount.Amount) {
                    return nil, errorsmod.Wrapf(
                              types.ErrNotEnoughDelegations,
                              "delegated amount %s is less than the total undelegation %s
for epoch %d",
                              totalDelegations,
                              totalUnbondings,
                              unbondingEpoch,
                    )
          }
```

Which totalDelegations is total amount of tokens staked with host zone and unbondAmount is amount of single undelegation. This comparison can lead to single msg.LiquidUnstake easily ignoring this condition. However, the total amount of epoch undelegation exceeds the staked amount of the host zone. So this will cause an error in UndelegationWorkflow.

Recommendation

After analyzing the above case, we should compare totalDelegations with the total undelegation of the epoch (after adding the current undelegation). So the correct comparison should be:

```
if totalDelegations.LTE(totalUnbondings.Amount) {
```

totalUnbondings is the total expected unstake of the epoch after the msg is processed.

Status: **Resolved**

Issue is resolved by commit:
https://github.com/persistenceOne/pstake-native/commit/676b6e7ccc097d0505fd110cee66ca4c7d85ba5d

Minor Issues

4. MakeRequest is missing connectionId & chainId valid check

We found this issue in commit:
https://github.com/persistenceOne/persistence-sdk/blob/master/x/interchainquery/keeper/keeper.go#L122

This function can be improved by adding some other chainId and connectionId checking conditions for empty string and the connectionId must follow the pattern connection-number.

Recommendation

The addition of conditions such as:

```
if connectionId == ""

if !strings.HasPrefix(connectionId, "connection")

if chainId == ""
```

will improve the quality of this function.

Status: **Acknowledged**

This issue will be fixed when the Persistence team eventually replaces the Interchain Query Module with the Async Interchain Query Module, and so does not require immediate attention.

5. Declare variable multiple times

We found this issue in commit:
https://github.com/persistenceOne/pstake-native/blob/29ef8d2f1f813acb9dc384fb4317dfbde9c042fa/x/liquidstakeibc/keeper/delegation_strategy.go#L34-L35

The variables currentDelegation & futureDelegation are declared to be used once per loop through the entire host zone's validator set. This is not necessary.

```
    for _, validator := range hc.Validators {
                    // calculate the new total delegated amount for the host chain

                    currentDelegation := hc.GetHostChainTotalDelegations()
                    futureDelegation := currentDelegation.Add(actionableAmount)
```

In addition, currentDelegation is calculated by hc.GetHostChainTotalDelegations which also runs through the same loop to calculate DelegatedAmount:

```
          for _, validator := range hc.Validators {

                    totalDelegations = totalDelegations.Add(validator.DelegatedAmount)
```

This results in a current functional complexity is n^2 (where n is the number of validators of the host zone).

Recommendation

The removal of currentDelegation and futureDelegation defintions out of the loop will reduce the complexity from n^2 to n.

Status: Acknowledged

The Persistence team has recognized this issue and it will be resolved in the next upgrade.

Informational

6. Duplicate coin variable

We found this issue in commit:
https://github.com/persistenceOne/pstake-native/blob/4537d4f134d26e3e6cebf12e82aec5805bf3ae1d/x/liquidstakeibc/keeper/msg_server.go#L477-L479

The values of tokenAmount and decTokenAmount are the same. This means that the definition of tokenAmount is not necessary, and we can assign decTokenAmount directly to calculate unbondAmount.

```
          decTokenAmount :=
sdktypes.NewDecCoinFromCoin(unstakeAmount).Amount.Mul(sdktypes.OneDec().Quo(hc.CValue))

          tokenAmount, _ := sdktypes.NewDecCoinFromDec(hc.HostDenom,
decTokenAmount).TruncateDecimal()

          unbondAmount := sdktypes.NewCoin(hc.HostDenom, tokenAmount.Amount)
```

Recommendation

Remove tokenAmount variable:

```
	decTokenAmount :=
sdktypes.NewDecCoinFromCoin(unstakeAmount).Amount.Mul(sdktypes.OneDec().Quo(h
c.CValue))

	unbondAmount, _ := sdktypes.NewDecCoinFromDec(hc.HostDenom,
decTokenAmount).TruncateDecimal()
```

Status: Resolved

Issue is resolved by commit:
https://github.com/persistenceOne/pstake-native/commit/676b6e7ccc097d0505fd110cee66ca4c7d85ba5d

7. Lack test on msgServer in liquidstakeibc module

We found this issue in commit:
https://github.com/persistenceOne/pstake-native/blob/4537d4f134d26e3e6cebf12e82aec5805bf3ae1d/x/liquidstakeibc/keeper

There is no test file for msgServer functions. This could potentially lead to many errors which are not easily diagnosable.

Recommendation

More tests are required to cover all keeper, msgServer, and Query functions. In addition, tests must cover all edge cases.

Status: Resolved

This issue was resolved by the commit:
https://github.com/persistenceOne/pstake-native/commit/f4173353730 93f59ae914e8da3f9057404153996

8. Missing specific type of ICQ events for cosmos/relayer

IBC Relayers could be using the cosmos/relayer, and within it there are declarations of two different event variables as follows:
https://github.com/cosmos/relayer/blob/main/relayer/processor/types_internal.go#L293-L294

```
  ClientICQTypeRequest  ClientICQType = "query_request"

  ClientICQTypeResponse ClientICQType = "query_response"
```

In the Persistence Interchain Query Module, only the default emit event is included. This means that relayers using cosmos/relayer will not be able to find any ICQ msgs.

Because most relayers are not currently running cosmos/relayer, this issue has been classified as informational, and not immediately urgent.

```
    k.IterateQueries(ctx, func(_ int64, queryInfo types.Query) (stop bool) {

                         if queryInfo.LastEmission.IsNil() || queryInfo.LastEmission.IsZero() ||
queryInfo.LastEmission.Add(queryInfo.Period).Equal(sdk.NewInt(ctx.BlockHeight())) {

                                     k.Logger(ctx).Info("Interchainquery event emitted", "id",
queryInfo.Id)

                                     event := sdk.NewEvent(
                                                 sdk.EventTypeMessage,
```

Recommendation

The addition of

- thequery_request event in the EndBlocker
- query_response event in MsgSubmitQueryResponse

will allow relayers to detect all event types and process all interchain queries.

Status: Acknowledged

The ICQ module is scheduled to be replaced by the Async ICQ module as soon as it is ready. This change should render the current issue resolved.

Conclusion

Notional has enjoyed working with the Persistence One team throughout the course of this audit. We have appreciated their communication and professionalism at every stage, and we look forward to seeing version 8 in production. The analysis performed by our team indicates that we should not expect additional issues with the upgrade.

Our discovery of a number of issues within their codebase has been met with prompt action by the team, and all of the issues uncovered have either been resolved or have clear plans in place for their eventual resolution. We find the novel structure of their codebase easy to understand and well-suited for the structure and arrangement of their development team.

While this code organization is a departure from other liquid staking projects like Stride and Quicksilver, Persistence's Version 8 will be the first liquid staking provider to implement the LSM in production, which is very exciting. We will be continuing on-going work with the Persistence code and mainnet upgrade, and look forward to our continued collaboration.