Portal Defi Solana HTLC Audit

Grey Swan

28-31 July 2025

Contents

Executive Summary

The Solana HTLC component of the Portal Network comprises both an implementation of hashed timelock contracts native to the Solana network. The Solana-native implementations of the component are invoked by the Portal Network's web2 infrastructure which facilitates atomic asset swaps across networks. This audit recognizes that the codebase is not yet feature-complete. Nevertheless, it was able to assess many important aspects of the system and identify facilities which currently exist in the code which must be reviewed upon feature completion.

Total Findings: 7 issues identified

Severity Breakdown:

- Critical: 0
- High: 0
- Medium: 4
- Low: 3
- Informational: 4

Overall Security Posture: No critical- or high-severity issues were identified, meaning that it is possible to iterate this codebase to feature completion without uncovering or introducing such issues.

Recommendations: Medium- and low-severity issues should be resolved to improve robustness. Upon feature-completion, the issues should be revisited and the additions and changes examined.

Audit Scope

Grey Swan was given the PortalDefi/mono repository at commit `816e32a`, which was committed to the `master` branch on 9 July 2025. We were asked to review the Solana program `hashedtimelock`. This program is to be used in their atomic asset swapping workflow. A precise file list can be found in Appendix A.

System Overview

The Portal to Bitcoin network is a new protocol that aims to facilitate cross-chain swaps of assets. It does this by allowing users to bridge in assets into a margin account on the Portal Network where they are credited with the assets from their chains and can put these assets into an order book to find users to match and swap with. Once two users are matched, the network helps facilitate the users' swap by registering their swap and letting them register invoices with each other to swap the assets on the respective chains where the assets are held. In this way, users only need to bridge a small amount of collateral in order to engage in a larger market and swap directly with other users.

The `hashedtimelock` program on Solana is the program traders will use to swap their Solana tokens to their swap counterparty. It does this by allowing a trader to lock funds into a "purse" token account that can be claimed by their counterparty once a secret to unlock has been revealed. The `hashedtimelock` also has the ability for users to deposit funds generally with a one day wait to be able to withdraw their funds once the withdraw process has occurred. In a future iteration ostensibly, this will allow users to bridge Solana tokens into the Portal network.

Medium Severity Issues

Data Accounts Not Protected

There are data accounts that are not protected from being overwritten.

- `init` can be called by anyone at any time to set the `fee` data in the config account
- `create_invoice` can be called by a signer multiple times to overwrite the data of their invoice

At this point in the development of the system, the impact is small, as, e.g., the fee is unused. This issue has significantly more impact at the stage where the system is feature-complete. We recommend that access control be implemented to ensure that data is protected from unauthorized and unintended overwriting (possibly using the `#[access_control(...)]` constraint annotation).

Update: This issue was fixed in commit `81a25a1`

`lock` caller is allowed to set arbitrary timeout period

The `lock` instruction allows the invoker (e.g. the secret holder) to specify an arbitrary `timeout`, which specifies the number of seconds until they are allowed to retrieve the funds. While this value is public on the Solana network, it is not broadcast in the subsequent log event. The counterparty should ensure that the timeout is sufficient to complete the swap protocol, specifically taking into account the transaction/confirmation times on their chain.

We assume that the web2 Portal infrastructure, which facilitates the swap protocol, invokes the instruction with an appropriate timeout. If less trust in this infrastructure is desired, we recommend that the timeout be guarded against values too low or high or, further, that it be hard-coded to an appropriate value.

Update: This issue has not been fixed. The Portal Defi team stated:

> We are planning to address this issue on the client side entirely (i.e. the web extension or app will verify the counterparty's timeouts before creating their own htlc).

Timeout bypass

If a user deposits funds in this contract they can begin the withdrawal process at any time by calling `withdraw`. This intends to make them wait a day before being able to realize the withdrawal with a call to `timeout_withdrawal`. However, users can bypass this by calling `withdraw_immediately` and

removing their funds immediately. And should this function be removed, users can easily bypass any future waiting period simply by calling `withdraw` right after they deposit their funds, allowing them to withdraw their funds at any time after the day after their deposit. We recommend requiring the user to call `withdraw` again if they have not removed their funds within a grace period after their funds are available. We also recommend applying a fee to those who want to withdraw immediately.

Update: This issue was fixed in commit `476fc53`.

Use of unchecked math

Rust does not check mathematical operations for over/underflow by default. Therefore all such operations should use their checked version to prevent these catastrophic types of errors. In the `save_fund` function, unchecked addition is used to account for the amount that has been deposited into the `Fund` account. This could diverge from the actual amounts if this amount were to ever reach the maximum of the `u64` type that is used for accounting. This maximum is roughly 2e19, a number that could very easily be reached if the token decimals were large enough. We recommend using `checked_add` for this operation.

Update: This issue was fixed in commit `32ac795`

Low Severity Issues

CPI code should use defined abstractions

The function `unlock::withdraw_and_close_purse` explicitly constructs a CPI context to perform a checked transfer. For a routine CPI, expanding the multi-step construction inline inhibits high-level reasoning about the code. It also violates the software engineering principle of having an authoritative implementation of each logical operation. We recommend using the `shared::transfer_tokens` function as `deposit::send_offered_tokens_to_vault` does or documenting with the construction why it must occur and the CPI be carried out explicitly.

Update: This issue was fixed in commit `b432c18`.

Invoice account cannot be closed

The user that creates invoices has to pay the fee to create the `Invoice` data accounts. This cost can never be returned to them because these invoices can never be closed. We recommend closing these accounts when the swap workflow they represent is finalized.

Update: This issue was fixed in commit `16765e7`.

Lack of comments

Throughout the codebase, functions are missing comments that describe their intended purpose. We recommending adding such comments to make future iterations easier to develop and secure.

Update: This issue was fixed in commit `b432c18`.

Considerations

`htl` Data Account Doesn't Use Associated Tokens

In the `lock` function, the caller can specify the account the funds will eventually be sent to, however, the token account they specify need not be a token account that the unlocker has access to. Consider storing the unlocker's address in the htl account so that their associated token account can receive the locked funds. Doing so would make offchain systems integrate more easily with the program and reduce the risk of signing a transaction that sends the funds somewhere not wanted.

Update: This consideration has not been accepted.

FIXME comments

In lib.rs!, lines 33-57 are commented out, representing a development idea that clutters the code. We recommend that the idea be documented and the code finalized or removed when the codebase is feature complete.

Update: This consideration has not been accepted.

Inconsistent use of set_inner

The `set_inner` facility allows one to explicitly and atomically overwrite the state of an account. Where applicable, its use is preferred over piecemeal state update, which can increase cognitive load about state update and inhibit reasoning. We recommend that `set_inner` be used wherever applicable, including in the `make_invoice::make_invoice` function.

Update: This consideration was accepted in commit `a7931e2`.

Unused Errors

Both `Unauthorized` and `InvalidSignatures` errors are unused. We recommend that their use is verified when the codebase is feature-complete, whether they need to be implemented or removed from the system.

Update: This consideration was partially accepted with `Unauthorized` becoming utilized in commit `32ac795`.

Appendix A - Files in Scope

Repo: https://github.com/Portaldefi/mono

Commit: `816e32aa225addefc7935b1c691ff616e8ad4b0c`

js/solana/programs/hashedtimelock/src

- constants.rs
- error.rs
- lib.rs
- instructions/
  - deposit.rs
  - expire.rs
  - init.rs
  - lock.rs
  - make_invoice.rs
  - mod.rs
  - shared.rs
  - timeout_withdrawal.rs
  - unlock.rs
  - withdraw_immediately.rs
  - withdraw.rs
- state/
  - config.rs
  - fund.rs
  - htl.rs
  - invoice.rs
  - mod.rs
  - swap.rs

Appendix B - Disclosures

No employee at Grey Swan has a financial stake or vested interest in, or a relationship outside of security consulting with, Portal Defi or its associated entities. Fees for this engagement were fixed and not contingent upon the findings or conclusions of the audit.