Portal Defi Network Audit

Grey Swan

23 June - 21 July 2025

Executive Summary

The Portal Network is a protocol that matches asset traders and facilitates their swapping assets on their respective chains atomically. This audit reviewed many of the core network contracts as well as trade-settling contracts on Ethereum.

- **Total Findings:** 15 issues identified
- **Severity Breakdown:**
  - Critical: 1
  - High: 1
  - Medium: 3
  - Low: 10
  - Considerations: 12
- **Recommendations:** All critical and high-severity issues must be addressed before deployment. Medium and low-severity issues should be resolved to further mitigate risks.

Audit Scope

For this audit, 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 bridging and settling contracts on Ethereum as well as many of the the core contracts of the Portal Network (sans order book) . 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 network itself is a Cosmos zone, forked from the Cronos chain (itself a fork of Ethermint). This means that the network runs an EVM implementation, the contracts are written in Solidity, and users can port dapps from EVM chains to Portal with negligible difficulties. At launch the supported assets will be Ethereum, Bitcoin (via the Lightning Network), SOL, and the network's native token, Portal. The network also has several important actors. Anyone can engage with the execution layer, or even become a node operator, to participate in the proof-of-stake consensus. But there is, at present, only one Validator. This Validator is in charge of the swap workflow and needs to perform several key functions, such as registering assets, tallying/assigning bridged assets, granting privileges to contracts in the system, and most importantly releasing funds from the bridge. This role is held by the Portal team itself, and in future iterations of the protocol there are plans to decentralize this role to at most twenty Validators. Another key role in the system is the Portal Foundation, which will have the power to register assets and give contracts power to move bridged assets. Their role is being developed with the protocol.

Portal Network

The Portal Network's asset-matching workflow consists of a few key contracts. The `ChainManager` and `AssetManager` contracts record what chains and assets are allowed in the marketplace, with the Validator having control over enabling/disabling all assets. The `LiquidityManager` contract is where bridged assets are accounted for and is controlled by the Validator. The Portal team is in the process of developing an order book where market traders can register their intent to trade at certain prices and be matched with counterparties for the trade. This order book will register these counterparties in the `SwapManager` contract, which decides who is the first mover in the asset swapping and provides a function for the parties to register the steps they are taking in the swap process. Swapping occurs on the respective chains of the assets and follows a model similar to a Hash Time Lock Contract (HTLC). The first person to pay (called the secret holder in this system) creates a secret that when revealed will finalize the movement of both traders' assets to each other's wallets. The second trader to pay is called the secret seeker. Each trader needs to take specific steps in order to protect themselves. If the secret holder gives their assets to the escrow contract and reveals the secret, the secret seeker could in theory remove those funds by revealing the secret without having had to pay the secret holder. The Portal Network's contracts understand and facilitate these steps in order.

Ethereum Network

The escrow account on Ethereum is the `InvoiceManager` contract and is where swap assets will pass through on their way to a swap's counterparty. The contract interacts with `NativeAssetManager`, which is where the Validator can register assets on Ethereum that are registered on Portal. The `PortalToken` contract is the Ethereum ERC20 token of the Portal Network's native token. As such it can be minted and burned when bridged by `PortalGateway` when users want to send this token to and from the Portal Network. The `PortalGateway` is the Ethereuem record of the validators and provides ways for them to perform actions in Ethereum.

Audit Assumptions

For this audit we have assumed that the Validator will perform their role competently, correctly, and in good faith. This includes an assumption that the penalty mechanisms, reward incentives, and a reputation-based framework the team will implement when the role is opened up will work correctly and as intended to curtail malicious validator behavior.

There are several libraries that the scope files import and utilize such as OpenZeppelin Contracts and Solmate. We assume that all such imported files will be the files anticipated and that they will work correctly within the system. We also have assumed that the Portal Network infrastructure (their node client and the Cosmos network) will work as expected and without error.

Critical Severity Issues

Overly Permissive Delegation Functions

In `CustomERC6909`, the `setOperator` function allows anyone to set themselves as an operator for anyone else. Furthermore, the `approve` function allows anyone to set an allowance for anyone else. Both of these mean that anyone can call these functions to give themselves the ability to call `transferFrom` with an unlimited allowance and remove all of anyone's assets. The ERC-6909 specification states that only the `msg.sender` may set operators for their account and indeed has no `owner` parameter in either of these functions. We recommend removing the `owner` parameter and limiting the function to allow `msg.sender` to set only their own parameters and no one else's.

UPDATE: This issue was fixed in commit `4524d7b`.

High Severity Issues

InvoiceManager can lock funds

The `InvoiceManager` contract facilitates the user-to-user asset swap where one trader will pay their part of the swap on the Ethereum chain. Since neither trader wants to pay the other first in a trustless environment, the scheme to ensure payment is to have the first trader lock their funds into a contract which only releases the locked funds to the second trader but only if the first trader gives them the secret key to open it. The second trader then deposits their funds into a similar but separate contract using the same key. Once the first trader reveals the key, all the funds are unlocked and moved to the other party. The obvious weakness to this approach is if one person refuses to continue the process, the funds are locked with no way to retrieve them. This is mitigated by timing out the contracts such that the funds are deposited, earmarked for the counterparty, and waiting to unlocked up until a specified time when the transaction is considered stalled and the funds will go back to the depositor. At each step, choosing the timelock carefully protects all parties. We see no such timelock in `InvoiceManager`. The secret holder (a.k.a. the first depositor) can deposit their funds in the contract, but if the secret seeker (the second depositor) doesn't deposit their funds on the other chain, we see no way for the secret holder to recover their funds. We recommend allowing the secret holder to recover their funds if the swap never reaches the `SEEKER_PAID` status after a specific time. Care will have to be taken to ensure that the timelocks are appropriately spaced between chains.

UPDATE: This issue has been partially resolved at time of publication. The Portal team added a timelock but has not yet ensured that it closes at the correct times. They stated:

> We will update this in the near future. The enhanced design will include:
>
> - Dynamic timelock adjustments based on chain specific block times, network congestion levels, and potentially swap amount
> - Enforcing timelock ordering, making sure that the holder's refund deadline is greater than the seeker's and keeping a gap between the two parties.

Medium Severity Issues

Assets on unregistered chain

ChainManager registers and enables chains, but this is never checked or called by any other contract in the system. When AssetManager registers or enables assets, we recommend that it check the ChainManager to see if the chain is enabled. Consider also whether ChainManager should push an update to AssetManager if a chain becomes disabled to then disable all the associated assets.

UPDATE: This issue is fixed in commit 485cbb0.

receive function doesn't deposit

In both NativeLiquidityManager and InvoiceManager, the receive function can receive ether that is sent to the contract, but it does not credit the msg.value to the transaction sender. This could lead to a scenario where users send ether to the contract and have no way of accessing it. We recommend crediting the caller's account with the msg.value or removing the receive functions entirely.

UPDATE: This issue was fixed in commit 8c3f403.

Power Accumulator Overflow

Solidity allows users to explicitly convert types but does so without checking or warning. This could give the illusion that such conversions are safe when in fact they can be very problematic. Within the PortalGateway contract, there are several instances of a voting power total being accumulated and then used to calculate a threshold of the voting power that must be met to pass a vote. The total power, however, is never checked to see if it is larger than can be represented by the uint64 type that will hold the voting threshold value. The following example can be found in addValidator , removeValidator, changeValidatorPower, and _updateValidatorSet where totalPowerAccumulator has type uint64 and threshold has type uint8:

```
unchecked {
      uint256 tmp = Math.ceilDiv(totalPowerAccumulator * threshold,
         100);

      if (tmp > type(uint64).max) revert PowerOverflow();

      _validatorState = ValidatorState({
        valsetNonce: newNonce,
        powerThreshold: uint64(tmp),
        totalPower: uint64(totalPowerAccumulator)
        });
    }
```

Placed in an unchecked block, the calculation `totalPowerAccumulator * threshold` will wrap the resulting value if it exceeds the bounds of the `uint64` type, potentially yielding a significantly smaller number. The value of `tmp` is bounded above by this result (and in the extreme is only a hundredth as large), rendering the maximum check unnecessary. The impact is that power values can be carefully chosen to severely reduce the power threshold necessary for a (possibly singleton) set of validators to make arbitrary changes to the validator set. We recommend (1) testing `totalPowerAccumulator` for overflow whenever it is changed, and (2) upcasting the factors of the `threshold` multiplication to avoid wrapping.

UPDATE: This issue was fixed in commit `9d17fdf`.

Low

Missing Event Firings

In `AssetManager`, assets can be registered via the `registerAsset` function. They can be registered as either `enabled` or `disabled`. When an asset is enabled or disabled in the `enable`/`disableAsset` functions, an event is fired, but this event is missing in the `registerAsset` function. We recommend emitting this event, as it is an important state change.

UPDATE: This issue was fixed in commit `9d62ea5`.

Events for redundant state changes

There are places in the codebase where events are emitted even though the state of the contract itself has not changed.

- In `AssetManager`, the `enable`/`disableAsset` functions emit events as long as the `assetId` parameter is found in the system, even if a caller is setting an asset's `enabled` state to what it already is.
- In `CustomERC6909.setGlobalOperator` an event is fired even if the state isn't changed.

In order to keep the event log accurate and informative, we recommend reverting if the caller is setting a storage value to its current value.

UPDATE: This issue was fixed in commit `161267a`.

Non-descriptive Errors

There are many places in the codebase where errors are thrown but may not help users know where the error occurred.

- In `CustomERC6909`'s `transfer` and `transferFrom` function, if a caller attempts to transfer a balance greater than they have, it will revert on an underflow, which may not be informative to the caller. Add an explicit error such as `ERC6909InsufficientBalance`.
- `CustomERC6909.transferFrom` also errors with insufficient allowance. Consider throwing an `ERC6909InsufficientAllowance` error.
- In `AssetManager.retrieveAssetDecimals` if an asset is not found, the `index-1` will trigger an out-of-bounds revert rather than the `AssetNotFound` error that other functions throw

- `NativeLiquidityManager.withdraw` will revert if the address cannot receive ETH and should have a descriptive error for that case
- `SwapManager.registerInvoice` will throw `NotAuthorized` if called with a `id` of zero. It should throw `SwapNotFound` instead

We recommend adding informative errors throughout the codebase to make intent clear and keep users informed.

UPDATE: These issues were fixed in commit `673879c`.

Floating Pragma

There are several places where floating pragma directives are used. These are generally not used for system deployments, where it can cause confusion as to what compiler was used throughout the system and what compiler bugs may be present.

- `ImmutableCreate2Factory.sol` has the `solidity ^0.8.25`
- `RoleManager.sol` has the `solidity ^0.8.20`

We recommend fixing all pragma directives to a specific version.

UPDATE: This issue was fixed in commits `673879c` and `1626592`.

Security Contact

Adding an email or address to the "header" comments of a smart contract allows users who want to disclose issues to do so quickly and correctly. We recommend adding a security contact to the contract documentation in each file. See this example for a best practice on how to do this.

UPDATE: This issue was fixed in commit `15ba1c7`.

Identical Overwrite

In `PortalToken`, the `decimals` function overwrites the `decimals` function of `ERC20Upgradeable` but implements it identically. We recommend removing the identical implementation.

UPDATE: This issue was fixed in commit `ca923ca`.

isEqual returns false if equal

In the `Shared` library, the `isEqual` function over `Swap` structures omits a necessary negation operation on the results of several field-by-field checks, resulting in a function that does not compute structural equality as intended. As the `isEqual` function is not called, we recommend removing the definition altogether.

UPDATE: This issue was fixed in commit `39dc645`.

Negation of int256

There are several places in the codebase where an `int256` (which can be negative) is negated. This will fail if the value being negated is `type(int256).min` because this negation is greater than `type(int256).max`. This occurs in

- `NativeLiquidityManager`'s `withdraw` function
- `LiquidityManager`'s `burnAsset` function on lines 92, 93, and 117

We recommend explicitly handling the case where `type(int256).min` is the value you wish to operate on.

UPDATE: This issue was not fixed at time of publication. The Portal team said:

> We will address this issue in the next iteration

Unnecessarily Complicated Execution Paths

There are several places in the codebase where control flow or looping can be simplified. The complexity can make it difficult to reason about and introduce risks to the contract's logic. We recommend fixing the following issues:

- The `ChainManager.registerAssetChain` method calls `_registerAssetChain` and the `AssetChainUpdated` is emitted in both sides of the if else statement and thus can both be removed from the check.
- The `createSwap` function in `SwapManager` relies on several lines of ternary operators to set the secret holder and seeker data. Its coherence and clarity would be much better served if an **`if`**/**`else`** block checking for the `isLightning` condition were used instead

UPDATE: This issue was not fixed at time of publication. The Portal team said:

> We will address this issue in the next iteration

Manual Import

The `PortalGateway` contract has an `IPortalToken` interface in the file that is identical to the interface provided by the `IPortalToken.sol` file. The possibility of differences between these two interfaces adds complexity and risk to the contract. We recommend importing from the file and removing the manual version.

UPDATE: This issue was not fixed at time of publication. The Portal team said:

> We will address this issue in the next iteration

Considerations

Code Styling

There are several instances in the codebase where a consistent style is not used.

- Public variables are sometimes prefixed with an underscore, which is the convention for denoting private variables:
  - `_assetIdByProps` in `AssetManager`
  - `_deposits` in `NativeLiquidityManager`
  - `_deposits` in `NativeAssetManager`

- It is good practice to give keys and values names when declaring a mapping. Consider the `isOperator` mapping in `CustomERC6909`. It could be given names such as `mapping(address User => mapping(address Operator => bool)`, which make the mapping much easier to understand. There are several such mappings in the codebase to consider such updating.
- The `burnAsset` method `LiquidityManager` checks for negative zero.

To make the codebase more readable and consistent, consider fixing these issues.

UPDATE: This consideration was not accepted at time of publication. The Portal Team stated:

> We anticipate accepting this in our next iteration

Event Improvements

Events are a key way that off-chain systems can build around an application and understand its state. Indexing makes it easy for observers to quickly find the information they need. Consider implementing indexing for parameters that quickly stratify the events into relevant groups and reworking events so that indexing can be used. For example, the following events emit structs that preclude useful indexing:

- The `AssetMinted` and `AssetBurned` events of `ILiquidityManager.sol`.
- The `SwapMatched`, `SwapHolderInvoiced` , and `SwapSeekerInvoiced`.
- The `AssetRegistered` and `SwapInvoiceCreated` events of `IInvoiceManager.sol`.

Consider converting these events to emit values instead of single structs. Consider also indexing the following events' parameters in `IInvoiceManager.sol`: `SwapHolderPaid`, `SwapSeekerPaid`, `SwapHolderSettled`, and `SwapSeekerSettled` .

UPDATE: This consideration was not accepted at time of publication. The Portal Team stated:

> We anticipate accepting this in our next iteration

Unused Local Variables

The `LIGHTNING_HASH` constant in `InvoiceManager.sol` is unused. Consider whether it needs to be included in the contract.

UPDATE: This consideration was not accepted at time of publication. The Portal Team stated:

> We anticipate accepting this in our next iteration

Implicit Variable Visibility

There are several variables that have no explicit visibility declared:

- Within `SwapManager`: `LIGHTNING_HASH`
- Within `InvoiceManager`: `LIGHTNING_HASH`
- Within `NativeAssetManager`: `ETHEREUM_HASH`, `ETH_SYMBOL_HASH`, and `ETH_NAME_HASH`
- Within `NativeLiquidityManager`: `_depositNonce`

Consider adding visibility identifiers, as they make intention clear for users and future developers.

UPDATE: This consideration was not accepted at time of publication. The Portal Team stated:

> We anticipate accepting this in our next iteration

TODO comments

There were several comments in the code that demarcate future work with TODO statements:

- Line 69 of `AssetManager.sol`
- Line 104 of `Shared.sol`
- Line 95 of `SwapManager.sol`
- Line 21 of of `InvoiceManager.sol`

Consider removing these before deployment and publication.

UPDATE: This consideration was not accepted at time of publication. The Portal Team stated:

> We anticipate accepting this in our next iteration

Duplicated Data

The `AssetChainData` struct utilized in `ChainManager` stores the name of the chain and whether it's active. This is then stored in the `nameToAssetChainData` mapping, which is indexed by the name of the very data that it stores. This is redundant because in order to retrieve the name from the mapping, you would already have to know it. Considering storing only whether it's active. This would also entail storing a separate boolean value denoting whether the chain has been registered (for the validation `enable`/`disableChain`).

UPDATE: This consideration was not accepted at time of publication. The Portal Team stated:

> We anticipate accepting this in our next iteration

Upgradeable contracts and gap variables

Both `PortalToken` and `RoleManager` use gap variables to allow future upgrades to these contracts to add more variables without overwriting storage of inheriting contracts. EIP-7201 describes an alternative method for assigning storage slots called "namespace storage." Such an approach is much more resistant to storage collisions on upgrade. Consider implementing namespace storage instead of using gap variables.

UPDATE: This consideration was not accepted at time of publication. The Portal Team stated:

> We anticipate accepting this in our next iteration

Disable All Initializers

The `PortalToken` contract has no constructor. Instead it has an `initialize` function, as it is intended to be the implementation for a proxy contract. It is nonetheless a best practice to disable the initializers on an implementation contract even if there is no security risk of someone interacting with it. Consider adding a constructor that solely calls `_disableinitializers()`.

UPDATE: This consideration was not accepted at time of publication. The Portal Team stated:

> We anticipate accepting this in our next iteration

Use of `transfer`

Both `transfer` and `send` send an ETH amount to the address they operate on as well as 2300 gas in order to ensure the transfer. However, EVM opcodes can and often do change their cost, meaning that 2300 may not be enough in the future for receiving contracts to actually receive the funds. Furthermore, smart contracts may need more than 2300 gas to `receive` the ETH sent. However, the use of its alternative `address.call{value: amount}("")` is dangerous due to it forwarding all available gas, opening the door for re-entrancy attacks. To mitigate this, best practice is to use `call` with a re-entrancy guard. Consider adopting the `call` method for ETH transfers and implementing it with re-entrancy guards in the contract.

`transfer` is used on

- line 245 of `InvoiceManager.sol`
- line 165 of `NativeLiquidityManager.sol`

UPDATE: This consideration was partially accepted at time of publication, removing the `transfer` call in the `NativeLiquidityManager` contract. The Portal Team stated:

> We anticipate accepting the rest of this consideration in our next iteration

Spurious Dependency

`SwapManager` has an unnecessary dependency on `OrderbookMarket`, as the sole reference of its type is not used beyond mere identity. Removing this dependency would make the `SwapManager` abstract to the type of the swap creator and reduce the surface area of the interface.

UPDATE: This consideration was not accepted at time of publication. The Portal Team stated:

> We anticipate accepting this in our next iteration

Inconsistent Context Usage

`ChainManager` and `PortalGateway` inherit the `Context` contract, which provides `_msgSender()` and `_msgData` methods for accessing these variables under different circumstances. Consider if these methods should be used in the code instead of `msg.sender` as found in the contracts.

UPDATE: This consideration was not accepted at time of publication. The Portal Team stated:

> We anticipate accepting this in our next iteration

Use of uint type

It is good security practice to be explicit in your code. Consider using the fully specified `uint256` /`int256` type in place of `uint`/`int`. We note its use in the following locations:

- Lines 103, 119, 135, and 149 of `AssetManager.sol`
- Lines 142 and 166 of `Shared.sol`
- Line 32 of `SwapManager.sol`

UPDATE: This consideration was not accepted at time of publication. The Portal Team stated:

> We anticipate accepting this in our next iteration

Appendix A - Files in Scope

Repo: https://github.com/Portaldefi/mono/tree/816e32aa225addefc7935b1c691ff616e8ad4b0c Commit: `816e32aa225addefc7935b1c691ff616e8ad4b0c`

`js/ethereum/contracts/`

- ImmutableCreate2Factory.sol
- InvoiceManager.sol
- NativeAssetManager.sol
- NativeLiquidityManager.sol
- PortalGateway.sol
- PortalToken.sol
- RoleManager.sol

`js/ptb/contracts/`

- AssetManager.sol
- ChainManager.sol
- CustomERC6909.sol
- LiquidityManager.sol
- Market.sol
- Shared.sol
- SwapManager.sol
- ValidatorManager.sol
- markets/OrderbookMarket.sol

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.