Interlay Security Audit

Audit of pallets and configuration

Reference	22-03-942-REP
Version	1.1
Date	2022/04/13

Quarkslab SAS
13 rue Saint Ambroise
75011 Paris
France

Contents

1 Project Information

Document history			
Version	**Date**	**Details**	**Authors**
1.1	2022/04/13	Initial Version	Sébastien Rolland, Mahé Tardy and two Quarkslab auditors[1].

Quarkslab		
Contact	**Role**	**Contact Address**
Frédéric Raynal	CEO	fraynal@quarkslab.com
Stavia Salomon	Sales Manager	ssalomon@quarkslab.com
Matthieu Ramtine Tofighi Shirazi	Project Manager	mrtofighishirazi@quarkslab.com
Sébastien Rolland	R&D Engineer	srolland@quarkslab.com
Mahé Tardy	R&D Engineer	mtardy@quarkslab.com
Quarkslab auditor[1]	R&D Engineer	
Quarkslab auditor[1]	R&D Engineer	

Interlay		
Contact	**Role**	**Contact Address**
Dominik Harz	Co-Founder & CTO	dominik@interlay.io
Alexei Zamyatin	Co-Founder & CEO	alexei@interlay.io
Sander Bosma	Software Engineer	sander@interlay.io
Gregory Hill	Research & Software Engineering	gregory@interlay.io

[1]Given the public release of this report, some auditors preferred to remain anonymous.

2 Executive Summary

This report describes the results of the security evaluation made by Quarkslab on multiple components of the parachain developed by Interlay. The audit was focused on the Vault Registry, which is the on-chain part of the vaults management, and the Governance components.

The main components reviewed were the `vault-registry` pallet and the governance pallets, `democracy`, `escrow`, `annuity` and `supply`. Their configuration and integration into the runtimes were also investigated.

The audit aims at verifying that the implementation respects the specification written by Interlay and presents no security issues in terms of availability and resources theft.

Audits have already been performed on some components like the Vault Registry by another audit company [1]. During Quarkslab's assessment, no major vulnerability has been found. The report is composed of many recommendations on minor flaws that were discovered during the audit.

This is version 1.1, which was delivered to Interlay on 2022/04/13, for public release. It follows the initial version, 1.0, which was delivered and discussed with Interlay.

2.1 Disclaimer

This report reflects the work and results obtained within the duration of the audit on the specified scope (see. Section 3.3) as agreed between Interlay and Quarkslab. Tests are not guaranteed to be exhaustive and the report does not ensure the code to be bug-free.

2.2 Findings Summary

The severity classification, informative, low, and medium, reflects a relative hierarchy between the various findings of this report. For example, most panics are unreachable in practice, but in order to reflect the importance of this class of bugs, they were promoted to medium severity. The following table describes in more details the rating for findings severity levels.

Severity	Description
`High`	Exploitable major issues that could result in loss of funds or DDoS attack.
`Medium`	Medium issues that cannot be directly exploited, such as the use of unsafe arithmetic or potential panics. These issues could potentially lead to loss of funds or DDoS attacks in future updates.
`Low`	Low issues that cannot be directly exploited such as mismatch between the specification and implementation on pre/post conditions or incorrect weight. These issues could potentially lead to logic bugs and cheap computation or storage.
`Info`	Diverse informative recommendations on code structure, documentation, TODO annotations, etc.

No critical or exploitable vulnerabilities were found during the audit.

ID	Description	Category	Severity
MEDIUM_1	Hook `on_initialize` from supply pallet will panic in the future	Runtime panic	Medium
MEDIUM_2	Integer overflow and panic in `make_parsable_int` macro	Unsafe arithmetic	Medium
MEDIUM_3	Integer overflow and panic in `U256::parse` implementation	Unsafe arithmetic	Medium
MEDIUM_4	Potential panic with missing checks and `copy_from_slice`	Runtime panic	Medium
LOW_1	Parachain security status is not verified	Unverified precondition	Low
LOW_2	The `SystemCollateralCeiling` is not verified	Unverified submitted value	Low
LOW_3	The Secure Collateral Threshold is not verified	Unverified submitted value	Low
LOW_4	The Liquidation Threshold is not verified	Unverified submitted value	Low
LOW_5	The `replaceCollateral` is decreased by the caller	Not enforced postcondition	Low
LOW_6	The depositStake function is not called	Not enforced postcondition	Low
LOW_7	The Vault status is not verified properly	Wrong precondition verification	Low
LOW_8	The replaceCollateral is not increased	Not enforced postcondition	Low
LOW_9	Usage of unsafe multiplication in the `checkpoint` and `supply_at` function	Unsafe arithmetic	Low
LOW_10	Pre-condition in `withdraw` is not verified	Unverified pre-condition	Low
LOW_11	Pre-condition in `balance_at` is not verified	Unverified pre-condition	Low
LOW_12	Incorrect weights estimation in various extrinsics with length variable arguments	Incorrect weights	Low
LOW_13	Overflow check made after parsing in `bitcoin::BytesParser::parse`	Unsafe arithmetic	Low
LOW_14	Potential panic with `unwrap` in `from_hex_le` and `from_hex_be`	Runtime panic	Low
INFO_1	Parachain security status is indirectly verified	Unverified precondition	Info
INFO_2	Succesfully completed `DepositStake` should be added to post-conditions	Missing postcondition	Info

ID	Description	Category	Severity
INFO_3	Parachain security status is indirectly verified	Unverified precondition	Info
INFO_4	TODO annotations in add_btc_address	TODO annotation	Info
INFO_5	Precondition seems to be inverted	Error in specification	Info
INFO_6	Unspecified accept_new_issues extrinsic	Missing specification	Info
INFO_7	Unspecified report_undercollateralized_vault extrinsic	Missing specification	Info
INFO_8	Preconditions checked by the callees	Indirect verification of precondition	Info
INFO_9	Preconditions checked by the callees	Indirect verification of precondition	Info
INFO_10	Preconditions checked by the callees	Indirect verification of precondition	Info
INFO_11	Preconditions checked by the callees	Indirect verification of precondition	Info
INFO_12	Preconditions checked by the callees	Indirect verification of precondition	Info
INFO_13	Mismatch of amount of collateral between redeem_tokens_liquidation and spec	Specification issue	Info
INFO_14	The name of the function does not match the specification	Specification minor issue	Info
INFO_15	The second extrinsic is not specified	Lack of specification	Info
INFO_16	Call to an unsupported function Currency::repatriate_reserved	Useless computation	Info
INFO_17	Panic in debug because of debug_assert	Runtime panic	Info
INFO_18	Function maturing_referenda_at_inner could use available utility functions	Refactor	Info
INFO_19	Internal function use a different logging mechanism	Debug logging	Info
INFO_20	Wrong pre-condition in increase_unlock_height	Error in pre-condition	Info
INFO_21	TODO annotations in get_free_balance for account restrictions	TODO annotation	Info
INFO_22	Missing benchmark on withdraw weight	TODO annotation and missing benchmark	Info
INFO_23	Base weight and transaction byte fee set at 0 in interbtc	Weight miscomputation and TODO annotations	Info

ID	Description	Category	Severity
INFO_24	Pallet sudo in the interBTC runtime configuration	Privileged pallet	Info
INFO_25	Useless copies in `parse` functions	Useless copies	Info
INFO_26	Integer overflow in `parse_transaction_input`	Unsafe arithmetic	Info
INFO_27	Shift left overflow in `MerkleTree:computer_width`	Unsafe arithmetic	Info
INFO_28	Assumptions on the length of slices in `MerkleProof::traverse_and_build`	Length assumptions	Info

3 Context and Scope

3.1 Context

Interlay aims at bringing Bitcoin liquidity to many networks based on an implementation on Polkadot. Currently, the goal is to create a bridge between the Bitcoin network and the Polkadot network to bring liquidity to other parachains of the network.

To develop its solution, Interlay has a parachain on the Kusama network named Kintsugi. The organization entered the Polkadot auction in the second batch and won the 10th parachain slot. Their interBTC parachain on Polkadot started on the 11th of March 2022.

The purpose of a bridge is to lock an asset on the blockchain of origin and to mint an asset, equivalent in value, on the destination blockchain. One of the difficulties to implement bridges comes from the fact that blockchains have different technologies that present more or less capable interfaces. For example, Polkadot network blockchains are extremely customizable and flexible in comparison to the Bitcoin blockchain, that has very limited programming capabilities. To solve this issue, Interlay has built an off-chain client to perform complex tasks on the Bitcoin side and use its parachain to safely lock on-chain collateral on the Polkadot network. In addition, one specificity of the Interlay solution is that the vaults, which are the components that keep the locked Bitcoin, are fully decentralized and can operate independently. They register themselves via the parachain on the Polkadot network and can manage their actions on-chain via this interface.

3.2 Safety and Security Properties

In the case of a substrate-based network, the traditional security properties of a blockchain are somehow delegated to the relay chain of their network. Indeed, parachains are not fully independent and uses the consensus mechanism of the underlying relay chain. The block time can vary so there are no 1:1 block correspondence between relay chains and parachains. Yet, the parachain mechanism ensures a perfect lineage tracking between relay chain and parachain blocks.

However, specifically for blockchain bridges, one of the main security concern is that the 1:1 peg is not true because of vulnerabilities in the logic or the implementation of the bridge.

The considered security model takes into account a misbehaving user of the interlay ecosystem which attempts to attack the available interface surface of the parachain. The entry point for users is the JSON-RPC API which mainly exposes functions, called extrinsics.

3.3 Scope

The scope of this audit is defined by Rust modules or crates, that compose the features of a substrate-based blockchain. They are called pallets in the Polkadot ecosystem. All the audited source code is available in the main Github repository of Interlay named interbtc[1].

The audit focused on the following pallets and their respective runtime configuration in the parachains implementations:

- `vault-registry`: handles the management (registration, reporting, etc.) of the vaults.
- Governance related pallets:
  - `escrow`: implements a mechanism to lock governance token in exchange of voting power.
  - `democracy`: implements the governance features of the parachain to propose privileged actions and vote for them.
  - `annuity`: distributes various rewards.
  - `supply`: generates new supply to support inflation.

The usage of libraries and frameworks, such as ORM or Substrate, were reviewed but their respective implementations were out of the scope of this audit.

More information about the specific version and the setup of the audit can be found in Section 3.4.

3.4 Audit Settings

As the Interlay codebase is undergoing significant changes with the approach of the launch of interBTC on the Polkadot mainnet, versions used for the audit have been frozen in agreement with the Interlay team. Exact versions and commit ID are shown in Table 3.1. From the specified commit hashes, the parachain runtime binary was compiled on Linux x86-64 as well as macOS arm64.

Project	interbtc
Repository	`https://github.com/interlay/interbtc`
Commit hash	`9fed496a74c9b2b8bc0a3ed15e35804f79c79728`
Commit date	2022/02/07
Runtime	v1.7.1

Table 3.1: interbtc version references

To start a network, the CLI tool `polkadot-launch`[2] was used with a configuration to start a `rococo-local` relay chain with three validators and the `testnet` interBTC runtime with one collators. Polkadot `v0.9.15`[3] was used. A custom build-spec was given to the parachain to alter the initial values used by some pallets. To see more about the JSON configuration used, see in Appendix A.

[1] `https://github.com/interlay/interbtc`
[2] `https://github.com/paritytech/polkadot-launch`
[3] `https://github.com/paritytech/polkadot/releases/tag/v0.9.15`

4 Methodology

4.1 Familiarize with the Interlay ecosystem

The first step of the audit was to dive into the ecosystem, reading the documentation at our disposal and testing the product with the testnet available. Then a local working setup was built to experiment comfortably with the parachain and perform some dynamic tests. In the case of Interlay, it was pretty straightforward using `polkadot-launch` presented in Section 3.4.

The project has multiple documentations. The auditors first discovered the Interlay & Kintsugi Documentation [2], that had a FAQ hosted on notion [3]. Then, the general content also available on video provided a clear introduction to the project [4]. Finally, the auditors spent time reading the specification documentation [5].

4.2 Static code review and analysis

The next step consisted in manually reviewing the code of the pallets in order to find potential issues. Most of the attention was given to the extrinsics, which are the dispatchable functions from the blockchain. A close look was given to the internal and private functions that could be called from extrinsics and thus somehow reachable by a user.

Here is the methodology, designed as a checklist, that was used when auditing extrinsics.

Verify that weights are computed and benchmarked correctly. Weight is the computational cost of calling a dispatchable. The "pricing" of this value has to be correct to avoid cost-less execution on the blockchain which can lead to a denial of service attack. The substrate documentation recommends to write benchmarks and to run them against a specific machine configuration to measure the weight of calling extrinsics. If the weight is a constant, it must be wisely chosen via a benchmark, but a weight can also be variable according to the length or value of arguments of a dispatchable. Indeed, some extrinsics can take arrays, or blob of data, as inputs so the weight has to be adjusted to take its length into account if it can increase the computing cost.

Look for unsafe arithmetic functions. Rust is well-known for providing many protections against memory-related problems. But arithmetic, for performances reasons, is not checked when compiling in release mode. In blockchain environments, arithmetic is often critical because it is applied to assets. More generally, an overflow or an underflow could break the logic of a function or the computation of its weight, thus providing free execution or generation/destruction of assets.

Look for missing storage deposits. Some extrinsics that require storage can use deposits to avoid providing free storage without asking the user to pay for important fees. Such extrinsics should be investigated because free storage means denial of service attacks.

Look for runtime panic. Parachain runtimes must be written in a defensive manner and never panic because the blockchain has to produce blocks. Thus panics must be avoided as much as possible, that is why it is interesting to look for `.unwrap()` and `.expect()`, among others, in the code.

Verify the authorization model. Extrinsics usually start with an authorization check, under the form of `ensure_something(origin)?;` or a configuration filter for example. For each extrinsic, the authorization model must be enforced in order to prevent an unprivileged user to perform a privileged operation.

Verify the usage of the transactional macro. The `#[transactional]` macro was recently introduced in Substrate and is useful in order to make sure that the side effects of an extrinsic can be reverted if it does not succeed. Otherwise, an error during a dispatchable execution could create an undetermined state for the blockchain storages.

Verify the validity of pre and post conditions. The specification of interbtc [5] contains, for most of the extrinsics, the conditions that should be met before and after the execution of the extrinsic, making it easier to verify the correctness of the implementation.

Look for general logic and implementation errors. Look for anything that could produce an error or something that was forgotten at the time of implementation.

4.3 Configuration review

The configuration was also reviewed for the pallets that were audited. The types used by the pallets and called by the extrinsics were investigated. Most of the time, it is difficult to audit a pallet without looking at its typical configuration. At the time of the audit, there were `interlay`, `kintsugi` and `testnet` runtimes.

4.4 Dynamic testing

Given the weak points discovered during the static analysis phase, some test scenarios were performed to ensure that extrinsics behave correctly or that an exploitation was not possible. For that, the auditors used the `@polkadot/api` [6] in TypeScript by generating the TypeScript types of the interbtc runtime and a custom wrapper around the `py-substrate-interface` Python library [7].

4.5 Tools

In addition to manual reviews and dynamic tests, some tools were used to ease the analysis. Rust Analyzer [8] was used to navigate in the code, finding correct types, and also to expand macros.

Also, the auditors used Clippy [9], the reference linter tool for the Rust language. For example, among others, this kind of Clippy command was used, retrieving a lot of false positive that have to be manually reviewed.

```
$ cargo clippy --no-deps -p CRATE_NAME -- -A clippy::all -W
↪  clippy::integer_arithmetic -W clippy::string_slice -W clippy::expect_used -W
↪  clippy::fallible_impl_from -W clippy::get_unwrap -W
↪  clippy::index_refutable_slice -W clippy::indexing_slicing -W
↪  clippy::match_on_vec_items -W clippy::match_wild_err_arm -W
↪  clippy::missing_panics_doc -W clippy::panic -W clippy::panic_in_result_fn -W
↪  clippy::unreachable -W clippy::unwrap_in_result -W clippy::unwrap_used
```

5 Recommendations

The following sections are mostly organized into pallets with the remarks, associated with the code and the configuration, grouped. However, there is a section for general recommendations on the runtime configuration and on pallets content that were out of scope but partially investigated with the approval of the Interlay team.

5.1 Vault Registry

The Vault Registry pallet takes care of vault management. For example, it allows vaults to register themselves, update their collateral, update public keys, and liquidate. It is composed of 7 extrinsics including 2 undocumented ones. It also contains 5 "privileged" extrinsics which require to be called by the root account. Functions of the Vault Registry module are mainly used by the issue, redeem, refund and replace modules which are not in the scope of this audit.

5.1.1 `register_vault` extrinsic and others

`INFO_1`	Parachain security status is indirectly verified		
Category	Unverified precondition		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The `register_vault` method should comply with preconditions and postconditions as per the specification[1], version 5.6.1 at the time the audit has been performed. It seems that the following precondition is not verified: “The BTC Parachain status in the Security component MUST NOT be SHUTDOWN:2”

However, the implemented `BaseCallFilter` of Interlay runtime prevents from calling Vault Registry pallet extrinsics in this case.

```
impl Contains<Call> for BaseCallFilter {
  fn contains(call: &Call) -> bool {
    if matches!(
      call,
      Call::System(_)
        | Call::Authorship(_)
        | Call::Session(_)
        | Call::Timestamp(_)
        | Call::ParachainSystem(_)
```

[1] https://spec.interlay.io/spec/vault-registry.html#register-vault

```
                | Call::Sudo(_)
                | Call::Democracy(_)
                | Call::Escrow(_)
                | Call::TechnicalCommittee(_)
        ) {
            // always allow core calls
            true
        } else if security::Pallet::<Runtime>::is_parachain_shutdown() {
            // in shutdown mode, all non-core calls are disallowed
            false
        } else if let Call::PolkadotXcm(_) = call {
            // For security reasons, disallow usage of the xcm package by users. Sudo
        ↪   and
            // governance are still able to call these (sudo is explicitly
        ↪   white-listed, while
            // governance bypasses this call filter).
            false
        } else {
            true
        }
    }
}
```

As per the precondition and as a good practice, it should be verified within the extrinsic to prevent future updates from breaking the rule.

Warning

The following extrinsics and internal functions also have the exact same issue:

- `deposit_collateral` extrinsic, see Section 5.1.2 for more information.
- `decrease_tokens`.
- `decrease_to_be_issued_tokens`.
- `decrease_to_be_redeemed_tokens`.
- `decrease_tokens`.
- `issue_tokens`.
- `redeem_tokens`.
- `redeem_tokens_liquidation`.

`INFO_2`	Succesfully completed `DepositStake` should be added to post-conditions		
Category	Missing postcondition		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

As this extrinsic is calling the `deposit_collateral` and the `try_deposit_collateral` functions, both of which in turn call `VaultStaking::deposit_stake`, it would make sense to add it as a postcondition.

5.1.2 `deposit_collateral` extrinsic

`INFO_3`	Parachain security status is indirectly verified		
Category	Unverified precondition		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

Like above, the `deposit_collateral` extrinsic does not verify the precondition mentioned in the specification “The BTC Parachain status in the Security component MUST NOT be SHUTDOWN:2”

However, in addition of being blocked by the `BaseCallFilter` runtime configuration, the call wouldn’t succeed thanks to the Oracle crate that calls `ensure_parachain_status_running` while checking for the exchange rate. This would raise an error if the status is not set on `RUNNING:0`.

As per the precondition and as a good practice, it should anyway be verified within the extrinsic to prevent future updates to the filter or the oracle to break this precondition.

5.1.3 `withdraw_collateral` extrinsic and others

`LOW_1`	Parachain security status is not verified		
Category	Unverified precondition		
Status	Present		
Rating	Severity: Low	Impact: None	Exploitability: None

The `withdraw_collateral` method does not verify the precondition mentioned in the specification [5]: “The BTC Parachain status in the Security component MUST be set to RUNNING:0.”

This precondition, unlike the above in Section 5.1.1, is not verified because the `BaseCallFilter` only blocks extrinsics when the status is `SHUTDOWN:2`. The parachain could also be in `ERROR:1` and the functions would be executed.

Warning

The following extrinsics and internal functions also have the exact same issue:

- `replace_tokens` internal function.
- `try_increase_to_be_redeemed_tokens` internal function.
- `try_increase_to_be_issued_tokens` internal function, see Section 5.1.18 for more information.
- `try_increase_to_be_replaced_tokens` internal function, see Section 5.1.19 for more information.

5.1.4 `register_address` extrinsic

`INFO_4`	TODO annotations in `add_btc_address`		
Category	TODO annotation		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

In `add_btc_address` method, reached by `register_vault -> insert_vault_deposit_address -> RichVault::insert_deposit_address`, in *vault-registry/src/types.rs:125* there is the following line:

```
// TODO: add maximum or griefing collateral
```

The exact meaning of this todo is unclear without more context but it should be reviewed. Moreover, a `TODO` in production source code needs to be highlighted.

`INFO_5`	Precondition seems to be inverted		
Category	Error in specification		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The specification[2] mentions several preconditions of which:

- A vault with id vaultId MUST NOT be registered

However it seems it should be:

- A vault with id vaultId MUST be registered

5.1.5 `accept_new_issues` extrinsic

`INFO_6`	Unspecified `accept_new_issues` extrinsic		
Category	Missing specification		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

[2] https://spec.interlay.io/spec/vault-registry.html#registeraddress

The method `accept_new_issues` is an extrinsic of the Vault Registry pallet, however it is missing from the specification. No issue was found and an authorization model is enforced.

5.1.6 `report_undercollateralized_vault` extrinsic

INFO_7	Unspecified `report_undercollateralized_vault` extrinsic		
Category	Missing specification		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The method `report_undercollateralized_vault` is an extrinsic of the Vault Registry pallet, however it is missing from the specification. No issue was found and an authorization model is enforced.

5.1.7 `set_system_collateral_ceiling` privileged extrinsic

LOW_2	The `SystemCollateralCeiling` is not verified		
Category	Unverified submitted value		
Status	Present		
Rating	Severity: Low	Impact: None	Exploitability: None

As written above the definition of `SystemCollateralCeiling`, in a comment, the `SystemCollateralCeiling` should be greater than an other value:

```
/// Determines the over-collateralization rate for collateral locked by Vaults,
↪   necessary for
/// wrapped tokens. This threshold should be greater than the
↪   LiquidationCollateralThreshold.
```

However, this condition is not verified by `set_system_collateral_ceiling`. This information should also be added in the specification [5] as a precondition.

5.1.8 `set_secure_collateral_threshold` privileged extrinsic

LOW_3	The Secure Collateral Threshold is not verified		
Category	Unverified submitted value		
Status	Present		
Rating	Severity: Low	Impact: None	Exploitability: None

This method allows to change the secure collateral threshold. According to the specification [5]:

- The Secure Collateral Threshold MUST be greater than the Liquidation Threshold.
- The Secure Collateral Threshold MUST be greater than the Premium Redeem Threshold.

However, there is no verification on the submitted new threshold.

5.1.9 `set_liquidation_collateral_threshold` privileged extrinsic

LOW_4	The Liquidation Threshold is not verified		
Category	Unverified submitted value		
Status	Present		
Rating	Severity: Low	Impact: None	Exploitability: None

This method allows to change the liquidation collateral threshold for a currency. If a Vault's collateral rate drops below this, automatic liquidation is triggered. According to the specification [3], "The Liquidation Threshold MUST be greater than 100% for any collateral asset". However there is no verification on the submitted new threshold.

5.1.10 `cancel_replace_tokens` internal function

INFO_8	Preconditions checked by the callees		
Category	Indirect verification of precondition		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The following preconditions listed in the specification [5] are checked in an indirect way, using `checked_sub` inside the functions it calls:

[3] `https://spec.interlay.io/spec/vault-registry.html#register-vault`

1. If oldVault is not liquidated, its toBeRedeemedTokens MUST be greater than or equal to tokens.
2. If oldVault is liquidated, the liquidation vault's toBeRedeemedTokens MUST be greater than or equal to tokens.
3. If newVault is not liquidated, its toBeIssuedTokens MUST be greater than or equal to tokens.
4. If newVault is liquidated, the liquidation vault's toBeIssuedTokens MUST be greater than or equal to tokens.

The use of the `ensure!` macro would improve the readability and prevent future updates to the vaults' `decrease_to_be_issued` function from breaking these checks

5.1.11 `decrease_to_be_issued_tokens` internal function

`INFO_9`	Preconditions checked by the callees		
Category	Indirect verification of precondition		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

Similarly to the `cancel_replace` function (5.1.10), the use of the `ensure!` macro would improve the readability of the following preconditions:

1. If the vault is not liquidated, it MUST have at least tokens toBeIssuedTokens.
2. If the vault is liquidated, it MUST have at least tokens toBeIssuedTokens.

5.1.12 `decrease_to_be_redeemed_tokens` internal function

`INFO_10`	Preconditions checked by the callees		
Category	Indirect verification of precondition		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

Same as for the `cancel_replace` function (5.1.10), the use of the `ensure!` macro would improve the readability and stability of the following preconditions:

1. If the vault is not liquidated, its toBeRedeemedTokens MUST be greater than or equal to tokens.
2. If the vault is liquidated, the toBeRedeemedTokens of the liquidation vault MUST be greater than or equal to tokens.

5.1.13 `decrease_to_be_replaced_tokens` internal function

`LOW_5`	The `replaceCollateral` is decreased by the caller		
Category	Not enforced postcondition		
Status	Present		
Rating	Severity: Low	Impact: None	Exploitability: None

The postcondition “The vault’s replaceCollateral MUST be decreased by (min(tokens, toBeReplacedTokens) / toBeReplacedTokens) * replaceCollateral.” is not respected by the function. The computed value is returned but the function is not taking care of decreasing the `replaceCollateral` value of the vault. Nevertheless the calling functions are then using that returned value as an argument of the `transfer_funds` function which in turn takes care of decreasing the value. No exception was found during the audit but relying on the caller to enforce the postcondition of a function is error prone.

5.1.14 `decrease_tokens` internal function

`INFO_11`	Preconditions checked by the callees		
Category	Indirect verification of precondition		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

Same as for the `cancel_replace` function (5.1.10), the use of the `ensure!` macro would improve the readability and stability of the following preconditions:

1. If the vault is not liquidated, its toBeRedeemedTokens and issuedTokens MUST be greater than or equal to tokens.
2. If the vault is liquidated, the toBeRedeemedTokens and issuedTokens of the liquidation vault MUST be greater than or equal to tokens.

5.1.15 `issue_tokens` internal function

`INFO_12`	Preconditions checked by the callees		
Category	Indirect verification of precondition		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

Same as for the `cancel_replace` function (5.1.10), the use of the `ensure!` macro would improve the readability and stability of the following preconditions:

1. If the vault is not liquidated, its toBeIssuedTokens MUST be greater than or equal to tokens.
2. If the vault is liquidated, the toBeIssuedTokens of the liquidation vault MUST be greater than or equal to tokens.

5.1.16 `redeem_tokens` internal function

`LOW_6`	The depositStake function is not called		
Category	Not enforced postcondition		
Status	Present		
Rating	Severity: Low	Impact: None	Exploitability: None

The postcondition "If the vault IS liquidated function depositStake MUST complete successfully - parameterized by vaultId, vaultId, and toBeReleased."" is not respected. The function `depositStake` is never called in that case as shown on the following unrolled snippet:

```
let mut vault = Self::get_rich_vault_from_id(&vault_id)?;

// need to read before we decrease it
let to_be_redeemed_tokens = vault.to_be_redeemed_tokens();

vault.to_be_redeemed_tokens().checked_sub(&tokens)?.amount();
vault.update(|v| {
  v.to_be_redeemed_tokens = new_value;
});

Pallet::<T>::get_rich_liquidation_vault(vault.data.id.currencies)
  .decrease_issued(tokens);

let to_be_released = Self::calculate_collateral(&vault.liquidated_collateral(),
  tokens,
  &to_be_redeemed_tokens
)?;

let new = Self::get_total_user_vault_collateral(vault_id.currencies)?
  .checked_sub(to_be_released)?;
TotalUserVaultCollateral::<T>::insert(vault_id.currencies, new.amount());

vault.update(|v| {
  v.liquidated_collateral = v
    .liquidated_collateral
    .checked_sub(&to_be_released.amount())
    .ok_or(Error::<T>::ArithmeticUnderflow)?;
});

// release the collateral back to the free balance of the vault
ensure!(
  <orml_tokens::Pallet<T>>::unreserve(to_be_released.currency_id,
    vault_id.account_id,
    to_be_released.amount
    ).is_zero(),
  orml_tokens::Error::<T>::BalanceTooLow
);
```

No attack scenario was derived from this, hence the low rating of this bug.

5.1.17 `redeem_tokens_liquidation` internal function

`INFO_13`	Mismatch of amount of collateral between `redeem_tokens_liquidation` and spec		
Category	Specification issue		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The postcondition "The redeemer MUST have received an amount of collateral equal to $(tokens \times liquidationVault.backingCollateral) / liquidationVault.issuedTokens$" seems to be incorrect. The implementation is instead making a transfer of $(tokens \times liquidationVault.backingCollateral) / (liquidationVault.issuedTokens + liquidationVault.to\_be\_issued\_tokens)$, which is the amount documented in other place of the specification, in the "Vault Liquidation" part, in the section "Liquidations (Safety Failures)"[4] for example.

```
// transfer liquidated collateral to redeemer
let to_transfer = Self::calculate_collateral(
  &source_liquidation_vault.current_balance(currency_id)?,
  amount_wrapped,
  &liquidation_vault.backed_tokens()?,
)?;
```

The function `calculate_collateral` takes the collateral, the numerator and the denominator for the calculation. We noted that the denominator is `liquidated_vault.backed_tokens()` and this method returns `self.issued_tokens().checked_add(&self.to_be_issued_tokens())`.

5.1.18 `try_increase_to_be_issued_tokens` internal function

The `try_increase_to_be_issued_tokens` method does not verify the precondition mentioned in the specification [5], "The BTC Parachain status in the Security component MUST be set to

[4]https://spec.interlay.io/security_performance/liquidations.html#liquidations-safety-failures

RUNNING:0.", see Section 5.1.3 for more information.

LOW_7	The Vault status is not verified properly		
Category	Wrong precondition verification		
Status	Present		
Rating	Severity: Low	Impact: None	Exploitability: None

One of the precondition mentioned by the specification [5] tells that:

- `The vault status MUST be Active(true)`

However the vault is retrieved using `get_active_rich_vault_from_id` in *vault-registry/src/lib.rs:974* which retrieves an active Vault without further verification on the internal boolean value.

```rust
let mut vault = Self::get_active_rich_vault_from_id(&vault_id)?;
```

The following method is defined as follow :

```rust
/// Like get_rich_vault_from_id, but only returns active vaults
fn get_active_rich_vault_from_id(vault_id: &DefaultVaultId<T>) ->
↪   Result<RichVault<T>, DispatchError> {
    Ok(Self::get_active_vault_from_id(vault_id)?.into())
}
```

In turn `get_active_vault_from_id` evaluates to:

```rust
/// Like get_vault_from_id, but additionally checks that the vault is active
pub fn get_active_vault_from_id(vault_id: &DefaultVaultId<T>) ->
↪   Result<DefaultVault<T>, DispatchError> {
    let vault = Self::get_vault_from_id(vault_id)?;
    ensure!(
        matches!(vault.status, VaultStatus::Active(_)),
        Error::<T>::VaultNotFound
    );
    Ok(vault)
}
```

The method verifies that the vault is `Active(_)` but not if it is `Active(true)`. The gathered vault is active but may not accept new issue requests.

An additional check should be added to ensure the retrieved vault is accepting new issues, for example using an `ensure!` block like so:

```
ensure!(
    vault.status == VaultStatus::Active(true),
    Error::<T>::VaultNotAcceptingNewIssues
);
```

5.1.19 `try_increase_to_be_replaced_tokens` internal function

INFO_14	The name of the function does not match the specification		
Category	Specification minor issue		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

It seems this method is documented as `increaseToBeReplacedToken` in the specification[5] instead of `tryIncreaseToBeReplacedTokens`.

The `try_increase_to_be_replaced_tokens` method does not verify the precondition mentioned in the specification [5], "The BTC Parachain status in the Security component MUST be set to RUNNING:0.", see Section 5.1.3 for more information.

LOW_8	The replaceCollateral is not increased		
Category	Not enforced postcondition		
Status	Present		
Rating	Severity: Low	Impact: None	Exploitability: None

The `try_increase_to_be_replaced_tokens` method does not fulfill the postcondition mentioned in the specification:

- `The vault's replaceCollateral MUST be increased by collateral.`

The `new_collateral` is computed and used as a return value, but the vault's `replaceCollateral` in not increased by the function. It seems the caller is not using the value either to increase the vault's `replaceCollateral`.

5.2 Democracy

The democracy pallet implements proposals and vote mechanisms to perform a privileged action on the blockchain. Voting power is provided by the escrow pallet that is investigated in Section 5.3.

[5] `https://spec.interlay.io/spec/vault-registry.html#increasetobereplacedtokens`

To get a better understanding of the working of the democracy in the Polkadot ecosystem, the article “Participate in Democracy” [10] provides good summary.

Technically, this pallet is mostly a fork of the FRAME democracy pallet made on November 17th 2021[6]. Interlay first removed code from it, to see what was exactly removed, when in *interbtc/crates/democracy/src*, type:

```
$ git diff 2c55e2cce 9fed496a -- lib.rs
```

The file is approximately 1150 lines long and the modifications of interlay since the fork are 100 lines, this estimation was made with the following command:

```
$ git blame lib.rs | grep -v 2c55e2cce | wc -l
```

However since November 17th 2021, the official democracy FRAME implementation changed a little bit: many raw arithmetic operations were replaced by their corresponding safe `saturating_{add|mul}`. When in *substrate/frame/democracy/src*, to see the changes, type:

```
$ git diff c087bbedbde16711450c186518314903a2949cb3 master -- lib.rs
```

5.2.1 Differences with upstream substrate democracy pallet

Here is an overview of the changes on the upstream substrate repository[7] since the fork that could be cherrypicked. The ~~crossed-out~~ elements are those not concerned because the extrinsics or functions were removed from the pallet.

Events

FRAME developers added two new events: `voted` and `seconded`.

Extrinsics

- `second` emits the new seconded event.
- `fast_track` uses `saturating_add` instead of `+` in the `end` parameter of the `inject_referendum` call.
- ~~`veto_external` uses `saturating_add` to compute the `until` variable.~~
- `reap_preimage` uses `saturating_add` to check for the `ensure!` that throws a `TooEarly` error.

Internal functions

- `backing_for` uses `saturating_mul` to compute a deposit.

[6] https://github.com/interlay/interbtc/commit/2c55e2cce9e02c8bbf31217f4b777f03fdcb530a
[7] https://github.com/paritytech/substrate

- `internal_start_referendum` uses `saturating_add` for the `end` parameter of `inject_referendum`.
- `try_vote` deposits the new voted event.
- `try_remove_vote` uses `saturating_mul` to compute `unlock_at`.
- ~~`try_delegate` adds new logic, see the diff for more information~~
- ~~`try_undelegate` uses `saturating_add` to compute `unlock_block` used in `prior.accumulate` call.~~
- ~~`launch_external` uses `saturating_add` to compute the `end` parameter to `inject_referendum`.~~
- `launch_public` uses `defensive_unwrap_or_else` instead of `unwrap_or_else` and uses `saturating_add` to compute the `end` parameter to `inject_referendum`.
- `bake_referendum` uses `saturating_add` to compute the `when` variable.

5.2.2 General remarks

Here are a few general remarks on the democracy pallet, more information can be found in the following subsection about the points addressed.

Note

Proposals are stored as a tuple of `(index, hash, who)`. Deposits are tracked also as a tuple of `(who, value)`. It would be recommended to use specific types to encapsulate these tuples to ease reading the code and leverage Rust's type system.

Note

The use of the `transactional` macro would allow to revert modifications to the storage in case of failure instead of relying only on the developers to check every preconditions prior to making any change to the storage.

Note

It was noted that this crate has a different behavior than the others regarding logging. Several internal functions are printing debug information, even in a production build, using `sp_runtime::print`.

5.2.3 `second` extrinsic

`INFO_15`	The `second` extrinsic is not specified		
Category	Lack of specification		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

In the absence of specifications, the role of the `seconds_upper_bound` parameter of the `second` extrinsic is unclear.

The documentation states that it is only used to compute the weight for the extrinsic.

```
/// - `seconds_upper_bound`: an upper bound on the current number of seconds on
↪  this proposal. Extrinsic is
///   weighted according to this value with no refund
```

The auditors suggest to have the weight computed under a worst case scenario and to remove this parameter from the extrinsic.

5.2.4 `reap_preimage` extrinsic

INFO_16	Call to an unsupported function `Currency::repatriate_reserved`		
Category	Useless computation		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

INFO_17	Panic in debug because of `debug_assert`		
Category	Runtime panic		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The `democracy` pallet is configured to use the `escrow` pallet as its currency. This mean that the following call inside `reap_image` will always return an error:

```
let res = T::Currency::repatriate_reserved(&provider, &who, deposit, BalanceStatus::
↪  Free);
debug_assert!(res.is_ok());
```

As the implementation of `escrow::repatriate_reserved` is the following:

```
  // NOT SUPPORTED
  fn repatriate_reserved(
     _slashed: &T::AccountId,
     _beneficiary: &T::AccountId,
     _value: Self::Balance,
     _status: BalanceStatus,
  ) -> sp_std::result::Result<Self::Balance, DispatchError> {
    Err(Error::<T>::InvalidAction.into())
  }
```

Knowing that, it seems odd to use `debug_assert`, which isn't recommended as it panics at runtime, even if these are removed from default release builds.

5.2.5 `maturing_referenda_at_inner` internal function

INFO_18	Function `maturing_referenda_at_inner` could use available utility functions		
Category	Refactor		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The role of the function is to get ongoing referendum which will end at a given time. This function could be refactored to use the `referendum_status` and `ensure_ongoing` functions, as well as simplifying the logic and types used.

```
fn maturing_referenda_at_inner(
  n: T::BlockNumber,
  range: core::ops::Range<PropIndex>,
) -> Vec<(ReferendumIndex, ReferendumStatus<T::BlockNumber, T::Hash, BalanceOf<T>>)
↪   > {
  range
    .into_iter()
    .map(|i| (i, Self::referendum_info(i)))
    .filter_map(|(i, maybe_info)| match maybe_info {
      Some(ReferendumInfo::Ongoing(status)) => Some((i, status)),
      _ => None,
    })
    .filter(|(_, status)| status.end == n)
    .collect()
}
```

The auditors believe that encapsulating the return type in its own type will improve the overall readability. Using a slice complicates the code and triggers a `clippy` warning.

```
very complex type used. Consider factoring parts into `type` definitions
`#[deny(clippy::type_complexity)]` implied by `#[deny(warnings)]`
for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#type_complexity
```

5.2.6 `check_pre_image_is_missing`, `pre_image_data_len`, `decode_compact_u32_at` and `on_initialize` internal function

`INFO_19`	Internal function use a different logging mechanism		
Category	Debug logging		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The internal functions `check_pre_image_is_missing`, `pre_image_data_len`, `decode_compact_u32_at` and `on_initialize` are using `sp_runtime::print` which is used to print debug information, this should be removed from a production build to use the same logging mechanisms as the other crates.

5.3 Escrow

The escrow pallet allows locking governance tokens, for example KINT for Kintsugi or INTR for interBTC, in exchange for voting power, implemented by the use of the reward pallet. The voting power, or stake in the term of the reward pallet, decreases linearly following a curve. The minimal duration, or minimal granularity on the x-axis of the curve is called the span.

This pallet is of moderate size and most of its logic complexity resides in the `checkpoint` internal function. All the crate's internal functions are only used locally and are not called by other pallets. Most of the findings are small issues regarding the specification and unsafe arithmetic.

Note

It was noted that the `default_weights`, of the extrinsics `increase_unlock_height` and `withdraw` were significantly higher, by an order of 20 considering `RocksDbWeight`, to those of `create_lock` and `increase_amount`. Those benchmarks were apparently made considering the most complex curve scenario for the two extrinsics.

5.3.1 unsafe arithmetic

`LOW_9`	Usage of unsafe multiplication in the `checkpoint` and `supply_at` function		
Category	Unsafe arithmetic		
Status	Present		
Rating	Severity: Low	Impact: None	Exploitability: None

The `checkpoint` (on line 404[8]) and `supply_at` (on line 546[9]) functions perform raw multiplications which could be replaced by an overflow-safe operation, `saturating_mul` for example.

The arithmetic rounding of the height by the span "$(height/span) * span$" performed in the `round_height` internal function should be safe as it cannot overflow.

5.3.2 `increase_unlock_height` extrinsic

INFO_20	Wrong pre-condition in `increase_unlock_height`		
Category	Error in pre-condition		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The specification for this extrinsic[10] might be wrong. It seems that the pre-condition "The amount MUST be non-zero." is a typo from a copy-paste from the previous extrinsic. Also, the name of the implementation `increase_unlock_height` is not the same as the one in the specification `extend_unlock_height`.

5.3.3 `withdraw` extrinsic

LOW_10	Pre-condition in `withdraw` is not verified		
Category	Unverified pre-condition		
Status	Present		
Rating	Severity: Low	Impact: None	Exploitability: None

For this extrinsic, it seems that the specification pre-condition: "The account's old_locked.amount MUST be non-zero." condition is not checked. It does not seem to be a security issue.

[8] https://github.com/interlay/interbtc/blob/1.7.1/crates/escrow/src/lib.rs#L404
[9] https://github.com/interlay/interbtc/blob/1.7.1/crates/escrow/src/lib.rs#L546
[10] https://spec.interlay.io/spec/escrow.html#extend-unlock-height

5.3.4 `set_account_limit` extrinsic

INFO_21	TODO annotations in `get_free_balance` for account restrictions		
Category	TODO annotation		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

This extrinsic has low security risk considering that only the root origin is authorized but it is worth noticing that it is a direct access to insert into the `Limits` storage that limits how much some accounts can mint. This storage is used in the `get_free_balance` internal function which should be modified in the future to remove the specific restrictions enforced on these limited account.

```
fn get_free_balance(who: &T::AccountId) -> BalanceOf<T> {
  let free_balance = T::Currency::free_balance(who);
  // prevent blocked accounts from minting
  if <Blocks<T>>::get(who) {
    Zero::zero()
  }
  // limit total deposit of restricted accounts
  else if let Some((start, end)) = <Limits<T>>::get(who) {
    // TODO: remove these restrictions in the future when the token distribution
↪   is complete
    let current_height = Self::current_height();
    let point = Point::new::<T::BlockNumberToBalance>(free_balance, start, end, end
↪   .saturating_sub(start));
    point.reverse_balance_at::<T::BlockNumberToBalance>(end, current_height)
  } else {
    free_balance
  }
}
```

5.3.5 `balance_at` internal function

LOW_11	Pre-condition in `balance_at` is not verified		
Category	Unverified pre-condition		
Status	Present		
Rating	Severity: Low	Impact: None	Exploitability: None

The specified pre-condition “The height MUST be $>=$ point.height.” is not verified in the code of the internal function.

5.4 Annuity

The annuity pallet is dedicated to distributing rewards. It is composed of only one extrinsic and it is mostly used for its `on_initialize` hook.

This pallet is instantiated two times in the runtimes for the rewards related to escrow and the vaults.

```
type EscrowAnnuityInstance = annuity::Instance1;
... snip
type VaultAnnuityInstance = annuity::Instance2;
```

This pallet is, as stated, composed of a call on the `on_initialize` hook which is called at each block creation. This hook will call `distribute_block_reward` from the type implementing the `BlockRewardProvider` trait provided by the configuration. These types are slighty different between the escrow and vault instances.

INFO_22	Missing benchmark on `withdraw` weight		
Category	TODO annotation and missing benchmark		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

This pallet is small in size and no benchmarks or extensive tests are performed on it. The only extrinsic, `withdraw`'s weight is arbitrarily set to 100_000_000 with a `TODO` comment on top of it. It would be recommended to implement the corresponding benchmark to ensure the function's cost is adapted to the amount of accesses it makes to the storage. However the `withdraw` extrinsic is pretty simple, it only ensures that the origin is signed, withdraw the reward using the `withdraw_-reward` of the type implementing of the `BlockRewardProvider` and transferring the asset to the signed origin.

Warning

The annuity pallet is missing from the specification.

5.5 Supply

The supply pallet handles the generation of governance tokens, for example KINT on Kintsugi and INTR on InterBTC. These tokens, via a locking mechanism in the escrow pallet, give voting power for the governance and are separate from the interBTC tokens emitted when sending BTC to a vault.

Some tokens are generated every year (more precisely every 2_628_000 blocks) as part of the inflation mechanism. The intended evolution stated in the documentation is that 10 million governance tokens will be generated on day one and distributed over the course of the next four years as rewards when generating blocks, being active as a Vault, etc. Next, “2% annual inflation afterward, indefinitely.” means that at the start of year five 200_000 tokens will be generated and then distributed during year five, then 204_000 to distribute during year six, etc. There should be no other way to generate those tokens than the annual inflation. However, democracy is able to change that inflation rate, using this pallet’s only extrinsic, through a proposal and voted by the people owning governance tokens.

More technically, on each block creation, in the `on_initialize` hook, some tests are performed against the total supply and the inflation rate to choose to call the `Currency::deposit_creating` and deposit the generated supply into the pallet’s account. Then `OnInflation::on_inflation` is called from the configuration with the pallet’s account as the `from` argument to transfer the generated supply to different accounts. In the current configuration, some are distributed to the vault and escrow reward pallets according to `INFLATION_REWARDS` constants and the rest is given to the treasury.

The only extrinsic available is `set_start_height_and_inflation` which is a direct root access to put the `StartHeight` and `Inflation` storage.

Note

This pallet’s implementation is really similar to annuity, that we review in Section 5.4. It also lacks benchmarks for weights or complexe tests. But the only extrinsic is a root privileged one which makes it *de facto* more secure.

Warning

The supply pallet is missing from the specification.

5.5.1 Inflation mechanism will panic in the future

`MEDIUM_1`	Hook `on_initialize` from supply pallet will panic in the future		
Category	Runtime panic		
Status	Present		
Rating	Severity: Medium	Impact: None	Exploitability: None

During the audit, it was noted that the internal function `begin_block`, used in the `on_initialize` hook, performs a call to the Rust `unwrap` function, which is designed to panic under certain conditions. After some investigations and exchanges with the team, it was assessed that it has almost no security impact as the rate at which governance tokens are generated is fixed (10 million over the first four years then, starting year five, 2% every year, as explained in the previous paragraphs) and it should cause a panic in more than 1500 years.

The panic can be triggered by modifying the `should_inflate_supply_from_start_height` test as follows:

```
fn should_inflate_supply_from_start_height() {
  run_test(|| {
    Supply::begin_block(0);
    let mut start_height = 100;
    assert_eq!(Supply::start_height(), Some(start_height));
    assert_eq!(Supply::last_emission(), 0);

    for _emission in 1..1576 {
      Supply::begin_block(start_height);
      start_height += YEARS;
    }
  })
}
```

Which leads to the panic message:

```
"tests::should_inflate_supply_from_start_height" panicked at "called
↪  'Option::unwrap()' on a 'None' value", crates/supply/src/lib.rs:163:99
```

5.5.2 Configuration

No problem was found in the configuration. The distribution percentages are the ones defined in the token economy whitepaper [11] and the `on_inflation` function is using a chain of `saturating_sub` calls to ensure no extra tokens are distributed due to rounding errors with the percentages.

5.6 Runtime Configuration

5.6.1 Weight general settings at 0 in interBTC runtime configuration

INFO_23	Base weight and transaction byte fee set at 0 in interbtc		
Category	Weight miscomputation and TODO annotations		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The Interlay runtime configuration contains some comments marked as TODO. It concerns the runtime settings `weights.base_extrinsic` and `TransactionByteFee`, they are both set to 0. It means that in the fee calculation process [12], only the call weight is added, without the base and length based weight.

At the time of the audit, these settings are still applied on the master branch of interBTC, and running in production. Quarkslab contacted Interlay and they explained that it was known and should be resolved soon.

```rust
weights.base_extrinsic = 0; // TODO: this is 0 so that we can do runtime upgrade
↪  without fees. Restore value afterwards!
```

```rust
// TODO: this is 0 so that we can do runtime upgrade without fees. Restore value
↪  afterwards!
pub const TransactionByteFee: Balance = 0;
```

```rust
// TODO: update this once we have the crowdloan data in
// Require 1 vINTR for now
pub MinimumDeposit: Balance = 1 * UNITS;
```

5.6.2 Pallet sudo in interBTC runtime configuration

INFO_24	Pallet sudo in the interBTC runtime configuration		
Category	Privileged pallet		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The sudo pallet is still present in interBTC runtime configuration[11]. It has been removed from the Kintsugi runtime but not from the interBTC one. It would be preferable to remove this pallet as it allows a privileged access for an arbitrary account to perform an extrinsic call with the root origin. Such call should be made via democracy instead of sudo, for example.

5.7 Weights

5.7.1 Incorrect weights estimation in `issue::execute_issue`, `redeem::execute_redeem`, `refund::execute_refund`, `relay::report_vault_theft` and `replace::execute_replace`

LOW_12	Incorrect weights estimation in various extrinsics with length variable arguments		
Category	Incorrect weights		
Status	Present		
Rating	Severity: Low	Impact: None	Exploitability: None

While investigating how the internal functions of `vault_registry` were used, it was noticed that

[11] https://github.com/interlay/interbtc/blob/1.7.1/parachain/runtime/interlay/src/lib.rs#L1232

several extrinsics from different pallets were configured without having a proper weight. It is a good practice to have the computed weight taking into account the worst case scenario when it comes to variable-length arrays.

For example, if we take a closer look to the `issue::execute_issue` extrinsic, both `merkle_proof` and `raw_tx` parameters are not used in the weight computation.

```rust
#[pallet::weight(<T as Config>::WeightInfo::execute_issue())]
#[transactional]
pub fn execute_issue(
    origin: OriginFor<T>,
    issue_id: H256,
    merkle_proof: Vec<u8>,
    raw_tx: Vec<u8>,
) -> DispatchResultWithPostInfo {
    let executor = ensure_signed(origin)?;
    Self::_execute_issue(executor, issue_id, merkle_proof, raw_tx)?;
    Ok(().into())
}
```

The same problem exists in `redeem::execute_redeem`, `refund::execute_refund`, `relay::report_vault_theft` and `replace::execute_replace`.

5.8 Bitcoin

In accordance with Interlay, the auditors spent a few days to verify that functions used by the `bitcoin` crate were panic-free. It is to be noted that these functions were previously audited and not in the scope of this audit. Even so, the auditors were interested to take a quick look at them because they were flagged with several warnings by Clippy.

Nothing critical was found but refactoring parts of the code should be done.

5.8.1 Integer overflow and panic `make_parsable_int`

MEDIUM_2	Integer overflow and panic in `make_parsable_int` macro		
Category	Unsafe arithmetic		
Status	Present		
Rating	Severity: Medium	Impact: None	Exploitability: None

After investigating the incorrect weights in Section 5.7.1, it was noticed that these extrinsics are using the functions `btc_relay::parse_transaction` and `btc_relay::parse_merkfle_proof`, with inputs controlled by an attacker. The implementations of these functions can be found in the bitcoin crate files[12].

[12] https://github.com/interlay/interbtc/blob/1.7.1/crates/bitcoin/src/merkle.rs#L222

In the macro that will implement `Parsable` for all Rust integer, `make_parsable_int`, see Listing 1, there is a potential overflow that leads to a panic. It can't be triggered in practice since it requires to overflow an `usize`, which is a huge array input for an extrinsic, but the check for the integer overflow needs to be rewritten.

```rust
/// Macro to generate `Parsable` implementation of uint types
macro_rules! make_parsable_int {
  ($type:ty, $bytes:expr) => {
    impl Parsable for $type {
      fn parse(raw_bytes: &[u8], position: usize) -> Result<($type, usize), Error> {
        if position + $bytes > raw_bytes.len() {
          return Err(Error::EndOfFile);
        }
        let mut value_bytes: [u8; $bytes] = Default::default();
        value_bytes.copy_from_slice(&raw_bytes[position..position + $bytes]);
        Ok((<$type>::from_le_bytes(value_bytes), $bytes))
      }
    }
  };
}
```

Listing 1: `make_parsable_int` macro

Here is a test to trigger the overflow that will fail in release mode and the output result.

```rust
#[test]
fn test_overflow() {
  let raw_bytes = vec![0u8; 100];
  let position = usize::MAX - 1;

  let (result, bytes_consumed) = u64::parse(&raw_bytes, position).unwrap();
}
```

```
---- parser::tests::test_overflow stdout ----
thread 'parser::tests::test_overflow' panicked at 'slice index starts at
↪  18446744073709551614 but ends at 6', crates/bitcoin/src/parser.rs:55:1
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

5.8.2 Integer overflow in `U256::parse`

`MEDIUM_3`	Integer overflow and panic in `U256::parse` implementation		
Category	Unsafe arithmetic		
Status	Present		
Rating	Severity: Medium	Impact: None	Exploitability: None

The same problem as the one explained in Section 5.8.1 is present in `U256::parse`.

```rust
impl Parsable for U256 {
  fn parse(raw_bytes: &[u8], position: usize) -> Result<(U256, usize), Error> {
    if position + 4 > raw_bytes.len() {
      return Err(Error::EndOfFile);
    }

    let mut bytes: [u8; 4] = Default::default();
    bytes.copy_from_slice(&raw_bytes[position..position + 4]);

    let bits = u32::from_le_bytes(bytes);
    let compact = U256::set_compact(bits).ok_or(Error::InvalidCompact)?;
    Ok((compact, 4))
  }
}
```

The following test fails in release mode:

```rust
#[test]
fn test_overflow_U256() {
  let raw_bytes = vec![0u8; 100];
  let position = usize::MAX - 3;

  let (result, bytes_consumed) = U256::parse(&raw_bytes, position).unwrap();
}
```

```
---- parser::tests::test_overflow_U256 stdout ----
thread 'parser::tests::test_overflow_U256' panicked at 'slice index starts at
↪  18446744073709551612 but ends at 0', crates/bitcoin/src/parser.rs:148:32
```

5.8.3 Logical error in `bitcoin::BytesParser::parse`

`LOW_13`	Overflow check made after parsing in `bitcoin::BytesParser::parse`		
Category	Unsafe arithmetic		
Status	Present		
Rating	Severity: Low	Impact: None	Exploitability: None

The overflow check in `bitcoin::BytesParser::parse` is made after parsing, see Listing 2. It does not prevent the potential panic in `parse` implementation if `self.position` is close to `usize::MAX`, which may not possible to reach with a real world example.

```
pub(crate) fn parse<T: Parsable>(&mut self) -> Result<T, Error> {
    let (result, bytes_consumed) = T::parse(&self.raw_bytes, self.position)?;
    self.position = self
        .position
        .checked_add(bytes_consumed)
        .ok_or(Error::ArithmeticOverflow)?;
    Ok(result)
}
```

Listing 2: `bitcoin::BytesParser::parse`

5.8.4 Useless copies in `parse` functions

`INFO_25`	Useless copies in `parse` functions		
Category	Useless copies		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

It is to be noted that the parsing creates a lot of useless copies. Slices are converted to a `Vec` when a `ByteParser` is created.

```
let mut parser = BytesParser::new(input_script);
... snip
impl BytesParser {
    /// Creates a new `BytesParser` to parse the given raw bytes
    pub(crate) fn new(bytes: &[u8]) -> BytesParser {
        BytesParser {
            raw_bytes: Vec::from(bytes),
            position: 0,
        }
    }
}
```

5.8.5 Integer overflow in `parse_transaction_input`

`INFO_26`	Integer overflow in `parse_transaction_input`		
Category	Unsafe arithmetic		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

An integer overflow is present in the `parse_transaction_input` function. It does not really matter since the `height_size` needed to overflow will cause the next call to read to fail.

```
script_size = script_size.checked_sub(height_size + 1).ok_or(Error::EndOfFile)?;
```

5.8.6 Shift left overflow in `MerkleTree::compute_width`

INFO_27	Shift left overflow in `MerkleTree:computer_width`		
Category	Unsafe arithmetic		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

The preconditions for the shift left to overflow cannot currently be met, but it should be best to verify.

```
impl MerkleTree {
  pub fn compute_width(transactions_count: u32, height: u32) -> u32 {
    (transactions_count + (1 << height) - 1) >> height
  }
```

5.8.7 Slice length assumptions issues in `MerkleProof::traverse_and_build`

INFO_28	Assumptions on the length of slices in `MerkleProof::traverse_and_build`		
Category	Length assumptions		
Status	Present		
Rating	Severity: Info	Impact: None	Exploitability: None

This function makes heavy assumptions on the size of `tx_ids` and `matches`. A check should be added to verify that the sizes are correct and identical before proceeding.

```
pub(crate) fn traverse_and_build(
  &mut self,
  height: u32,
  pos: u32,
  tx_ids: &[H256Le],
  matches: &[bool],
) -> Result<(), Error> {
    let mut parent_of_match = false;
    let mut p = pos << height;
    while p < (pos + 1) << height && p < self.transactions_count {
        parent_of_match |= matches[p as usize];
        p += 1;
    }

    self.flag_bits.push(parent_of_match);

    if height == 0 || !parent_of_match {
        let hash = self.compute_merkle_root(pos, height, tx_ids)?;
        self.hashes.push(hash);
    } else {
        let next_height = height.checked_sub(1).ok_or(Error::ArithmeticUnderflow)?;
        let left_index = pos.checked_mul(2).ok_or(Error::ArithmeticOverflow)?;
        let right_index = left_index.checked_add(1).ok_or(Error::ArithmeticOverflow)?;

        self.traverse_and_build(next_height, left_index, tx_ids, matches)?;
        if right_index < self.compute_partial_tree_width(next_height) {
            self.traverse_and_build(next_height, right_index, tx_ids, matches)?;
        }
    }

    Ok(())
}
```

5.8.8 `types::H256Le` various panics

MEDIUM_4	Potential panic with missing checks and `copy_from_slice`		
Category	Runtime panic		
Status	Present		
Rating	Severity: Medium	Impact: None	Exploitability: None

The input slice used in `from_bytes_le` and `from_bytes_be` functions is not checked and the call to `copy_from_slice` can panic[13].

```
impl H256Le {
    /// Creates a H256Le from little endian bytes
    pub fn from_bytes_le(bytes: &[u8]) -> H256Le {
        let mut content: [u8; 32] = Default::default();
        content.copy_from_slice(&bytes);
        H256Le { content }
    }
```

[13] https://doc.rust-lang.org/std/primitive.slice.html#panics-29

```
    /// Creates a H256Le from big endian bytes
    pub fn from_bytes_be(bytes: &[u8]) -> H256Le {
        let bytes_le = reverse_endianness(bytes);
        let mut content: [u8; 32] = Default::default();
        content.copy_from_slice(&bytes_le);
        H256Le { content }
    }
```

LOW_14	Potential panic with unwrap in from_hex_le and from_hex_be		
Category	Runtime panic		
Status	Present		
Rating	Severity: Low	Impact: None	Exploitability: None

The call to `hex::decode` in `from_hex_le` and `from_hex_be` functions is not checked properly, the `unwrap` can fail and cause a panic. Nevertheless, please note that these functions are only enabled in `std` environment, see the `#[cfg(feature = "std")]` annotation. Thus, these functions will only be available if the +runtime is built and run as a native binary and not a WASM binary.

```
#[cfg(feature = "std")]
pub fn from_hex_le(hex: &str) -> H256Le {
    H256Le::from_bytes_le(&hex::decode(hex).unwrap())
}

#[cfg(feature = "std")]
pub fn from_hex_be(hex: &str) -> H256Le {
    H256Le::from_bytes_be(&hex::decode(hex).unwrap())
}
```

6 Conclusion

First of all, the quality of the interBTC code base managed by Interlay should be highlighted. The audit was facilitated by the organization of the code and the global help provided by Interlay. On top of that, the specification and the effort made to write preconditions and postconditions for each specified function were really useful for static code review. Quarkslab encourages Interlay to keep the quality consistency that was noted on most of the pallets for the whole project and to continue having their code audited by external companies.

The audit unveiled mostly informative and low recommendations, that should not be exploitable in practice but might be interesting for the Interlay team, to keep a consistent quality across the project and prevent future issues. A lot of the recommendations are informative, related to the preconditions and the postconditions of the specification, the lacking of specification for some part of the code, and "TODO" annotations in the code. Some of these recommendations were made possible because of the extensive specification and documentation of the project which must be hard to maintain. Nevertheless, some issues were found, related to runtime panic, incorrect weight computation or race conditions.

Glossary

clippy A collection of lints to catch common mistakes and improve your Rust code. See `https://github.com/rust-lang/rust-clippy`.

collator A node that maintains a parachain by collecting parachain transactions and producing state transition proofs for the validators.

extrinsic State changes that come from the outside world, i.e. they are not part of the system itself. Extrinsics can take two forms, "inherents" and "transactions".

interBTC is Interlay's flagship product - Bitcoin on any blockchain. A 1:1 Bitcoin-backed asset, fully collateralized, interoperable, and censorship-resistant. interBTC will be hosted as a Polkadot parachain and connected to Cosmos, Ethereum and other major DeFi networks.

pallet Substrate modules exposing various extrinsics, events, errors and storage items that will be compiled in the runtime and usable by users or other components. It is implemented as Rust crates.

validator A node that secures the Relay Chain by staking DOT, validating proofs from collators on parachains and voting on consensus along with other validators.

Bibliography

[1] Informal Systems. *Security Audits*. 2021. URL: `https://github.com/informalsystems/audits` (visited on Mar. 10, 2022) (cit. on p. 2).

[2] *Interlay & Kintsugi Documentation*. Interlay. URL: `https://docs.interlay.io/` (visited on Mar. 14, 2022) (cit. on p. 8).

[3] *Interlay FAQ*. Interlay. URL: `https://interlay.notion.site/interlay/Interlay-FAQ-5e3019b1cfd94f6693dc186e9640e607` (visited on Mar. 14, 2022) (cit. on p. 8).

[4] Alexei Zamyatin. "Kintsugi - Radically open Bitcoin for Kusama". In: Kusama Demo Day Sep 2021. Sept. 1, 2021. URL: `https://www.youtube.com/watch?v=ErZBxmZY-_Y` (visited on Mar. 14, 2022) (cit. on p. 8).

[5] *interBTC Technical Specification*. v5.6.1. Interlay. URL: `https://spec.interlay.io/` (visited on Mar. 14, 2022) (cit. on pp. 8, 9, 13, 15, 16, 20–22).

[6] Parity. *polkadot/api*. URL: `https://github.com/polkadot-js/api` (visited on Mar. 14, 2022) (cit. on p. 9).

[7] Polkascan. *Python Substrate Interface*. URL: `https://github.com/polkascan/py-substrate-interface` (visited on Mar. 14, 2022) (cit. on p. 9).

[8] Rust-analyzer. *Rust-analyzer*. URL: `https://github.com/rust-analyzer/rust-analyzer` (visited on Mar. 14, 2022) (cit. on p. 9).

[9] rust-lang. *rust-clippy*. URL: `https://github.com/rust-lang/rust-clippy` (visited on Mar. 14, 2022) (cit. on p. 10).

[10] Polkadot. *Participate in Democracy*. URL: `https://wiki.polkadot.network/docs/maintain-guides-democracy` (visited on Mar. 17, 2022) (cit. on p. 23).

[11] Kintsugi Labs. *INT Token Economy*. Dec. 7, 2021. URL: `hhttps://github.com/interlay/whitepapers/blob/master/Interlay_Token_Economy.pdf` (visited on Mar. 14, 2022) (cit. on p. 32).

[12] Polkadot. *Fee Calculation*. URL: `https://wiki.polkadot.network/docs/learn-transaction-fees#fee-calculation` (visited on Mar. 21, 2022) (cit. on p. 32).

Appendix A

Polkadot-launch Configuration

```json
{
    "relaychain": {
        "bin": "/home/vagrant/binaries/polkadot-v0.9.15",
        "chain": "rococo-local",
        "nodes": [
            {
                "name": "alice",
                "wsPort": 9944,
                "port": 30444
            },
            {
                "name": "bob",
                "wsPort": 9955,
                "port": 30555
            },
            {
                "name": "charlie",
                "wsPort": 9966,
                "port": 30666
            }
        ],
        "genesis": {
            "runtime": {
                "runtime_genesis_config": {
                    "configuration": {
                        "config": {
                            "validation_upgrade_frequency": 1,
                            "validation_upgrade_delay": 20
                        }
                    }
                }
            }
        }
    },
    "parachains": [
        {
            "bin": "/home/vagrant/binaries/interbtc-parachain-9fed496a",
            "id": "2121",
            "balance": "1000000000000000000000",
            "nodes": [
                {
                    "wsPort": 9988,
                    "port": 31200,
                    "name": "alice",
                    "flags": [
                        "--",
                        "--execution=wasm"
                    ]
                }
            ],
            "chain": "/home/vagrant/configs/interbtc-spec-raw.json"
        }
    ],
    "simpleParachains": [],
    "hrmpChannels": [],
    "types": {},
    "finalization": false
}
```