Quantstamp

Hipo Finance

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Liquid Staking
Timeline	2025-03-03 through 2025-03-25
Language	FunC
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Official Docs
Source Code	• https://github.com/HipoFinance/contract #da12344
Auditors	• Michael Boyle Auditing Engineer • Jonathan Mevs Auditing Engineer • Darren Jensen Auditing Engineer

Documentation quality	High
Test quality	High
Total Findings	4 Acknowledged: 4
High severity findings	0
Medium severity findings	1 Acknowledged: 1
Low severity findings	1 Acknowledged: 1
Undetermined severity findings	0
Informational findings	2 Acknowledged: 2

Summary of Findings

Hipo Finance is a liquid staking protocol built on the TON ecosystem. It allows users to stake TON without needing the full 300,000 TON typically required to run a validator. On the other side, technical users can borrow staked TON to operate validators, also without meeting the full requirement. This dual-sided model expands validator participation while enabling DeFi engagement through the hTON receipt token. If a validator is slashed, their rewards are used first to cover the loss, offering protection to stakers. Hipo also employs a bill NFT system, allowing users to stake and unstake during a round, with automatic redemption at the round's end.

During the audit, we identified one medium-severity, one low-severity, and two informational issues. The test suite is thorough, covering both successful and failure scenarios. We recommend that all issues be either resolved or explicitly acknowledged.

Fix-Review Update 2025-03-31:
After reviewing the client's response to our findings, it is clear that they have thoroughly explained their current approach. While some enhancements are not implemented immediately, the client has acknowledged the issues and outlined plans for future improvements. Overall, the protocol remains secure and is considered production-ready.

ID	DESCRIPTION	SEVERITY	STATUS
HIPO-1	Unhandled Stake Recovery Failure Leads to Potential Accounting Inconsistencies	Medium	Acknowledged
HIPO-2	Upgradeable Contracts Do Not Use a Time Lock	Low	Acknowledged
HIPO-3	`gift_tokens()` Can Skew TON-to-hTON Exchange Rate Under Low-Liquidity Conditions	Informational	Acknowledged
HIPO-4	Halter or Governor May Not Receive Excess Gas	Informational	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

> **Disclaimer**
> Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

Files Included

Repo: https://github.com/HipoFinance/contract(91291c5) Files: `contracts/*`

Files Excluded

Repo: https://github.com/HipoFinance/contract(91291c5) Files: `contracts/imports/stdlib.fc`

Operational Considerations

- The treasury contract depends on timely external messages to trigger time-sensitive state transitions (e.g., via `participate_in_election()`, `vset_changed()`, and `finish_participation()`). Borrowers and protocol participants are incentivized to do this.
- Governor and halter are expected to act honestly in accordance with the expected protocol behavior.
- Borrowers should be aware that they may only have one loan request open at a time. Making a new request will delete the previous request, regardless of whether the previous request was more favorable than the new one. Each new request incurs the same request loan fee, and the new stake amount provided on the new request will increment the existing stake amount for the request.
- The Treasury and Parent contracts support contract code and data updates through the `upgrade_code` and `upgrade_data` functions, which may introduce unforeseen risks in future versions.

Key Actors And Their Capabilities

- Governor
  - Update critical protocol parameters:
    - Set or adjust the governance fee
    - Modify the instant mint flag
    - Change the rounds imbalance parameter
  - Manage governance and upgrades:
    - Propose a new governor via propose_governor
    - Accept governance changes via accept_governance
    - Initiate system upgrades (using upgrade_code and proxy_upgrade_code)
  - Manage library dependencies:
    - Add libraries with proxy_add_library
    - Remove libraries with proxy_remove_library
  - Oversee administrative actions:
    - Mint bills for deferred unstaking
    - Route or trigger privileged state transitions
- Halter
  - Provide emergency controls and backup governance:
    - Set the protocol to a stopped state via set_stopped
    - Adjust shared parameters like instant mint flag and rounds imbalance
    - Participate in key administrative actions alongside the Governor (e.g., content updates via proxy_set_content)
  - Serve as an alternative authority for urgent actions
- Parent Contract
  - Act as the central gateway for user interactions:
    - Route deposits from user wallets to the Treasury
    - Validate that operations (e.g., deposits, unstaking) come from up-to-date and authorized wallets
  - Serve as the master/minter for hTON jettons
- Stakers
  - Deposit TON to participate in staking:
    - Receive hTON jettons as a receipt for their deposit
  - Initiate unstaking:
    - Burn hTON tokens to redeem TON from the protocol
    - Interact with the unstaking process which may involve bill NFT minting for deferred operations
- Borrowers
  - Request loans to meet staking requirements:
    - Submit loan requests that include staking collateral and specify a minimum payment
  - Interact with the loan system:
    - Use the received loans strictly for staking to validate blocks
    - Accept that any penalties or misbehaviors reduce their collateral rather than affecting stakers

Findings

HIPO-1

Unhandled Stake Recovery Failure Leads to Potential Accounting Inconsistencies

Medium | Acknowledged

Update

While the client's statements are accurate for the current state of the elector contract, there is potential for the elector contract to be upgraded. This issue is not problematic now, but could be in the future.

Update

Marked as "Unresolved" by the client.
The client provided the following explanation:

```
I reject it.
  It's crucial to process the error response the same as a success response.
  The case of an error response from elector smart contract is only when the signature is invalid. If we don't slash the borrower, it can lead to a potential attack where borrowers can request loans with the best return rate, but sign their request with an invalid signature.
  This can then lead to a situation where attackers prevent honest borrowers from requesting a loan, stopping the protocol from generating rewards for stakers.
  Also, the ok? variable is intentionally named and not used, to show the return field in the incoming message.
```

File(s) affected: `contracts/treasury.fc`

Description: The `treasury` contract processes `op::recover_stake_result` messages from the `loan` contract, which includes calculating any reward or slashing reductions to the borrower `stake_amount`. However, when a stake recovery message is sent from the `elector` contract due to an error, the `recover_stake_result()` function still attempts to execute reward and slashing logic. Unless the `reward` is exactly zero, this can result in unfair stake slashing and inaccurate treasury accounting. Further, the calculated fees for loan requests and staking rely on constant gas calculations for the elector contract, which could change in the future if the behavior of the elector contract were to be updated by the network, resulting in an underestimation that could cause a borrower to lose their stake.

Recommendation:

- Modify the `recover_stake_result()` function in the `treasury` contract to explicitly check for error conditions, perhaps by using the `ok?` variable (which is currently unused).
- Skip reward and slashing calculations (currently on lines L1181 - L1203 of the `treasury` contract) when a stake recovery operation has failed (`ok?` is `false`).
- Ensure the full `stake_amount` is returned to the borrower.

HIPO-2 Upgradeable Contracts Do Not Use a Time Lock

• Low ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
I accept it, and may include a time lock in future versions.
```

File(s) affected: `contracts/treasury.fc`, `contracts/parent.fc`

Description: The `Treasury` and `Parent` contracts are upgradeable by the Governor without a time lock delay. A time lock would allow users to exit the project before an upgrade goes into effect. It also allows for a fail-safe in the case of compromised private keys. If an attacker obtained the private key for the Governor, he could upgrade the contract to steal the funds, and the team would not have time to react.

Recommendation: Consider splitting the upgrade process into a two-step process in which new code is proposed and then can be set after a predetermined amount of time has passed.

HIPO-3 `gift_tokens()` Can Skew TON-to-hTON Exchange Rate Under Low-Liquidity Conditions

• Informational ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
I accept it, and may include a minimum liquidity check in future versions.
```

Description: The `gift_tokens()` function allows TON to be added to the staking pool without minting corresponding hTON receipt tokens. While this mechanism is intended to benefit existing hTON holders by increasing the backing of each token, it can distort the TON-to-hTON exchange rate under certain conditions.

If the protocol holds a very low amount of staked TON (e.g., early in the lifecycle or after a mass unstaking event), gifting a disproportionately large amount of TON can significantly inflate the value of hTON. This may create opportunities for manipulation or unintended arbitrage, especially if hTON is traded on secondary markets.

Recommendation: Consider implementing a minimum pool liquidity threshold below which `gift_tokens()` is disabled or volume-limited.

HIPO-4 Halter or Governor May Not Receive Excess Gas

• Informational ⓘ Acknowledged

Update

Marked as "Unresolved" by the client.
The client provided the following explanation:

```
With the current TVM, it's not possible to find the address of the original sender, in a bounce
message.
  Bounced messages are very limited, in total 256 bits are returned, 32 bits of it will be the bounce
code, so only 224 bits remain, 32 bits being the original opcode, 64 bits being the query_id. This only
allows for 128 more bits, which can only host a Coins amount, and it can not contain any address field.
```

So, as the comment in the code states, it always returns the excess gas to the owner, which is *usually* the original sender. Governor should be careful in this case.

File(s) affected: `contracts/wallet.fc`

Description: In the `on_bounce()` function in `wallet.fc`, the contract assumes that any failure should return the excess gas to the owner of the wallet. However, it is possible that the original caller was either the Halter or the Governor address in the case where the wallet upgrade flow fails. Any extra gas would be sent to the owner instead of the administrative accounts.

Recommendation: Consider determining who to send the excess gas to based on the operation that failed.

Auditor Suggestions

S1 Missing Input Validation

Acknowledged

Update

Marked as "Unresolved" by the client.
The client provided the following explanation:

1. There is no need to check against max_stake. More than max_stake amounts are ignored in elector. If a borrower can still provide a better rate for that amount, let it help the stakers of Hipo.
2. recv_internal functions always receive an in_msg_full non-empty message. No need to check for it.
3. No worries, since governor can reset it to another address if needed.
4. That's possible and I'll consider it in future versions.

File(s) affected: `treasury.fc`

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error.

- In `request_loan()`, there is no check against the `max_stake` value of the network.
- All `recv_internal()` functions can benefit from checking that `in_msg_full()` is non-empty.
- In `treasury.fc`, the `set_halter()` function allows the `halter` address to be updated to `new_halter`; however, it does not validate for a zero address. Consider adding validation that the `new_halter` address is not zero.
- In the `treasury.fc` contract, the `accept_governance()` function checks whether the function is called after the `accept_after` timestamp. However, there is no deadline specified, which means a proposed governor could remain in the contract indefinitely and be accepted at any point in the future. To address this, consider implementing a deadline by adding a fixed duration to the `accept_after` timestamp. This would ensure the `governor` is accepted within a reasonable timeframe after the waiting period.

Recommendation: We recommend adding the relevant checks.

S2 Unused Variable

Acknowledged

Update

Marked as "Unresolved" by the client.
The client provided the following explanation:

It's used as a form of documentation, otherwise I had to use a comment in front of it to tell the reader what this field is.

File(s) affected: `treasury.fc`

Description: The `ok?` field in `recover_stake_result()` is currently unused. If it is not needed, it can be removed from future versions of the code.

Recommendation: If this variable is no longer needed, it can be removed.

S3 Gas Savings Through Early Failure

Acknowledged

Update

Marked as "Unresolved" by the client.
The client provided the following explanation:

```
As a style of code, all checks are grouped together in the middle of the handlers. These can be moved
to earlier lines in the handler, but it will only help save fees for those attacking the protocol,
which is not a good reason to break the code style.
```

File(s) affected: `treasury.fc`

Description: In `request_loan()`, the following checks can be done before replacing a previous borrower's request or enforcing the bound of 7 participations at a time:

- `throw_unless(err::access_denied, borrower_wc == chain::base);`
- `throw_unless(err::not_accepting_loan_requests, round_since == next_round_since);`
- `throw_unless(err::not_accepting_loan_requests, now() < participate_since);`
- `throw_if(err::stopped, stopped?);`

This would offer gas savings before looping or modifying storage.

Recommendation: Consider performing the mentioned checks earlier in the function

S4 Magic Numbers

Acknowledged

Update

Marked as "Unresolved" by the client.
The client provided the following explanation:

```
I do accept that using a named constant for some of these can help with the readability of the code,
and I'll consider it in future versions.
```

File(s) affected: `contracts/utils.fc`, `contracts/treasury.fc`

Description: Magic numbers—literal numeric values without explanatory context—are used throughout the codebase. These values can obscure the intention behind certain calculations and make the code harder to maintain or audit. It is generally considered best practice to replace magic numbers with named constants to improve readability and make future updates safer.

The following instances were identified:

- `utils.fc:15`: The `store_body()` function uses `513`.
- `utils.fc:21`: The `store_log()` function uses `654`.
- `treasury.fc:1186, 1198`: The `recover_stake_result()` function uses `65535` (maximum of a 16-bit integer) as the base for fee calculation.
- `treasury.fc:788, 896, 1092, 1182, 1183`: `255` is used repeatedly (maximum of an 8-bit integer).
- `treasury.fc:603`: Throws if `participations_count > 7`. Consider defining the max allowed participations as a constant or configurable parameter.

Recommendation: Consider adding aptly named constants for these values to `imports/constants.fc`.

S5 Code Improvements

Acknowledged

Update

Marked as "Unresolved" by the client.
The client provided the following explanation:

```
1. It doesn't matter in TVM.
2. It doesn't matter in TVM.
3. It will not have any gas saving, and this way it's more similar to other codes, and shows the fields
   purpose better.
4. Global variables in TVM are only usable in the context of processing an internal message, so nothing
   can go wrong with updating it in this function.
     5.1. Typo will be fixed in future versions.
     5.2. The max_recommended_punishment_for_validator_misbehaviour is copies verbatim from another
   smart contract written by Ton Core, and I believe it's better to keep it as an exact copy.
     5.3. The storage of a bill is currently at 1022 bits. If I had to store this field as a 256-bit
   integer, then I had to store an additional cell which makes the bill contract more costly. As it can
   hold a very big number of NFT indexes by 64-bit, I compromised here, so that users can save more on
   gas fees.
```

File(s) affected: `contracts/treasury.fc`, `contracts/bill.fc`, `contracts/imports/utils.fc`

Description:

- In the `decide_loan_requests()` function of the `treasury` contract, the `op::process_loan_requests` message is sent out before updating the participations data. Consider updating the participations before sending out the message.
- In the `recover_stakes()` function of the `treasury` contract, the `op::recover_stake` message is sent out before updating the participations data. Consider updating the participations before sending out the message.
- In the `treasury` contract, the `vset_changed()` function loads the `query_id` but does not use it. Consider removing the `query_id` local variable and skipping those bits instead.
- In the `treasury` contract, the `calculate_min_coins()` directly updates the `total_coins` global variable. To avoid any unexpected accounting issues following contract upgrades, avoid directly updating the global variable and consider copying `total_coins` to a local variable and using that instead.
- In `utils.fc`, the `request_sort_key()` defines a local variable, `efficieny`, which has a typo and should be changed to `efficiency`. In `utils.fc`, the `max_recommended_punishment_for_validator_misbehaviour()` contains unused local variables `prefix` and `unpunishable_interval`, which can be removed. In `bill.fc`, the `get_static_data()` function stores the index as a 256-bit `int`; however, the contract's global int index is stored as a 64-bit int. Consider correcting the size of the index data field stored in `get_static_data()`.

Recommendation: Consider applying the code improvements as described.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Files

- `411...a41 ./contracts/librarian.fc`
- `b31...5cf ./contracts/parent.fc`
- `ea9...668 ./contracts/bill.fc`
- `a05...c4f ./contracts/treasury.fc`
- `bf0...5d4 ./contracts/wallet.fc`
- `7a6...294 ./contracts/loan.fc`
- `6d6...47c ./contracts/collection.fc`
- `f77...4c9 ./contracts/imports/constants.fc`
- `cd9...29d ./contracts/imports/stdlib.fc`
- `2c2...902 ./contracts/imports/utils.fc`

Tests

- `f47...c40 ./tests/Access.spec.ts`
- `6fc...c04 ./tests/Librarian.spec.ts`
- `0fb...f92 ./tests/Large.spec.ts`
- `40f...b94 ./tests/Loan.spec.ts`

- `f48...c71 ./tests/setup-jest.ts`
- `741...be5 ./tests/Getters.spec.ts`
- `c37...a2a ./tests/helper.ts`
- `1ea...d55 ./tests/MaxGas.spec.ts`
- `f01...4e2 ./tests/Wallet.spec.ts`
- `cd2...a58 ./tests/Governance.spec.ts`
- `fc8...891 ./tests/MinGas.spec.ts`

Test Suite Results

There are 9 test suites with a total of 82 passing tests. The tests cover both "happy" and "unhappy" paths.

```
PASS  tests/Librarian.spec.ts
 Librarian
   ✓ should add a library (7813 ms)
   ✓ should remove a library (1776 ms)
   ✓ should withdraw surplus (2085 ms)

 console.info
   Code Storage Cost:
                 | Bytes | Cells | 1 Year Cost
      Treasury   |  7198 |   141 |  0.061633690
      Parent     |  1240 |    24 |  0.010547449
      Wallet     |  1722 |    37 | 15.528361817
      Collection |   915 |    26 |  9.777045411
      Bill       |  1107 |    25 | 10.276051026
      Loan       |   515 |    10 |  4.387111084
      Librarian  |   233 |     5 |  2.099961915

          Total  | 12929 |   268 | 42.140712392

     at logCodeCost (tests/helper.ts:171:17)

console.info
  Average gas usage:
           slow stake 1: 0.044839558
           slow stake 2: 0.040905202
         instant stake 1: 0.017832758
         instant stake 2: 0.013843600
          slow unstake 1: 0.040705602
          slow unstake 2: 0.040705604
       instant unstake 1: 0.020468002
       instant unstake 2: 0.020406000

      at Object.<anonymous> (tests/MinGas.spec.ts:564:17)

PASS  tests/MinGas.spec.ts
 Min Gas
   ✓ should require min gas fee in treasury (7134 ms)
   ✓ should require min gas fee in wallet (2059 ms)
   ✓ should print average gas usage for stake and unstake (5387 ms)

 console.info
   Total Fees: 9.993329273

     at logTotalFees (tests/helper.ts:152:17)

PASS  tests/Large.spec.ts
 Large
   ✓ should send a big batch of messages to recover stakes (9688 ms)
   ✓ should handle large number of loan requests (11267 ms)

 console.info
   Treasury Fees:
       request loan:       0.808723772
       deposit coins:      0.052548358
       unstake all tokens: 0.083542
```

```
      at logTreasuryFees (tests/helper.ts:221:17)

  console.info
    Wallet Fees:
        send tokens:        0.014612358
        unstake tokens:     0.0711156
        upgrade wallet:     0.0213988

      at logWalletFees (tests/helper.ts:233:17)

  console.info
    Total Fees: 0.185190173

      at logTotalFees (tests/helper.ts:152:17)

PASS  tests/Governance.spec.ts (60.651 s)
  Governance
    ✓ should propose governor (3306 ms)
    ✓ should accept governance (1791 ms)
    ✓ should set halter (1467 ms)
    ✓ should set stopped (2740 ms)
    ✓ should proxy set content (1958 ms)
    ✓ should set reward share (1475 ms)
    ✓ should set rounds imbalance (1311 ms)
    ✓ should send message to loan (1390 ms)
    ✓ should send process loan requests (1495 ms)
    ✓ should upgrade code (2595 ms)
    ✓ should withdraw surplus (1406 ms)
    ✓ should gift coins (3555 ms)

PASS  tests/Access.spec.ts (65.037 s)
  Access
    ✓ should check access in treasury (10760 ms)
    ✓ should check access in parent (2493 ms)
    ✓ should check access in wallet (2341 ms)
    ✓ should check access in collection (2378 ms)
    ✓ should check access in bill (2651 ms)
    ✓ should check access in loan (6731 ms)
    ✓ should check access in librarian (1613 ms)

PASS  tests/Getters.spec.ts (66.748 s)
  Getters
    ✓ should calculate code sizes (4036 ms)
    ✓ should return max punishment value (1725 ms)
    ✓ should return jetton data (2235 ms)
    ✓ should return loan data (5085 ms)
    ✓ should return wallet data (1762 ms)
    ✓ should return treasury state (2007 ms)
    ✓ should return wallet state (2481 ms)
    ✓ should return treasury fees (1695 ms)
    ✓ should return wallet fees (2404 ms)
    ✓ should return max burnable tokens (1904 ms)
    ✓ should return surplus (1707 ms)
    ✓ should return metadata for SBTs (2961 ms)

  console.info
    Total Fees: 0.592110699

      at logTotalFees (tests/helper.ts:152:17)

PASS  tests/Wallet.spec.ts (87.142 s)
  Wallet
    ✓ should deposit and mint tokens when there is no active round (5199 ms)
    ✓ should deposit and save coins when there is an active round (2562 ms)
    ✓ should deposit and mint tokens when instant mint flag is set (1685 ms)
    ✓ should deposit coins for a different owner (2324 ms)
    ✓ should unstake and withdraw coins when there is no active round (2426 ms)
    ✓ should unstake and reserve tokens when there is an active round (2295 ms)
    ✓ should unstake with different modes where there is no active round (2748 ms)
    ✓ should unstake with different modes when there is an active round (4147 ms)
    ✓ should deposit coins for comment d (1584 ms)
```

```
    ✓ should unstake all tokens for comment w sent to treasury (2511 ms)
    ✓ should unstake all tokens for comment w sent to wallet (1833 ms)
    ✓ should handle multiple deposits, unstakes, and sends (6433 ms)
    ✓ should handle invalid sends (1866 ms)
    ✓ should send tokens with minimum fee (1465 ms)
    ✓ should send tokens to another new wallet (1561 ms)
    ✓ should send tokens to another existing wallet (1575 ms)
    ✓ should send tokens with a payload similar to dedust (1335 ms)
    ✓ should respond with wallet address (1314 ms)
    ✓ should provide current quote (1380 ms)
    ✓ should withdraw surplus (1476 ms)
    ✓ should withdraw wrongly sent jettons (1408 ms)
    ✓ should upgrade wallet to itself when there is no new version (1496 ms)
    ✓ should upgrade wallet to new version when there is a new version (1900 ms)
    ✓ should upgrade wallet to new version when halter decides (1813 ms)

  console.info
    Total Fees: 1.089795388

      at logTotalFees (tests/helper.ts:152:17)

PASS  tests/Loan.spec.ts (94.685 s)
  Loan
    ✓ should save a loan request (4626 ms)
    ✓ should participate in election (6615 ms)
    ✓ should handle external participate message only once (6403 ms)
    ✓ should change vset (7462 ms)
    ✓ should finish participation (5858 ms)
    ✓ should handle external finish message only once (5777 ms)
    ✓ should remove participation when all loan requests are rejected (4840 ms)
    ✓ should remove participation when there is no funds available to give loans (4909 ms)
    ✓ should participate in election with balanced rounds (5139 ms)
    ✓ should handle loan request edge cases (2560 ms)
    ✓ should handle dense election spans (2249 ms)
    ✓ should correctly set sort keys (2683 ms)

  console.info
    Total Fees: 58.86769057

      at logTotalFees (tests/helper.ts:152:17)

  console.info
    Compute Gas:
        const int gas::send_tokens = 10678;
        const int gas::receive_tokens = 11691;
        const int gas::deposit_coins = 18741;
        const int gas::proxy_save_coins = 6175;
        const int gas::save_coins = 7091;
        const int gas::mint_bill = 7757;
        const int gas::assign_bill = 5960;
        const int gas::burn_bill = 6558;
        const int gas::bill_burned = 12316;
        const int gas::mint_tokens = 12230;
        const int gas::proxy_tokens_minted = 6841;
        const int gas::tokens_minted = 13453;
        const int gas::unstake_tokens = 9040;
        const int gas::proxy_reserve_tokens = 6538;
        const int gas::reserve_tokens = 15521;
        const int gas::burn_tokens = 16627;
        const int gas::proxy_tokens_burned = 7307;
        const int gas::tokens_burned = 7179;
        const int gas::send_unstake_all = 12967;
        const int gas::proxy_unstake_all = 6553;
        const int gas::unstake_all = 7423;
        const int gas::upgrade_wallet = 7618;
        const int gas::proxy_migrate_wallet = 7978;
        const int gas::migrate_wallet = 12802;
        const int gas::proxy_merge_wallet = 7841;
        const int gas::merge_wallet = 7443;

      at logComputeGas (tests/MaxGas.spec.ts:1728:17)
```

```
 console.info
   Compute Gas:
       const int gas::request_loan = 45000;
       const int gas::participate_in_election = 29000;
       const int gas::decide_loan_requests = 21000;
       const int gas::process_loan_requests = 26000;
       const int gas::proxy_new_stake = 8000;
       const int gas::vset_changed = 11000;
       const int gas::finish_participation = 12000;
       const int gas::recover_stakes = 22000;
       const int gas::proxy_recover_stake = 5000;
       const int gas::recover_stake_result = 39000;
       const int gas::burn_all = 8000;
       const int gas::last_bill_burned = 19000;
       const int gas::new_stake = 18000;
       const int gas::new_stake_error = 6000;
       const int gas::new_stake_ok = 2000;
       const int gas::recover_stake = 11000;
       const int gas::recover_stake_ok = 6000;

     at logComputeGas (tests/MaxGas.spec.ts:1728:17)

PASS  tests/MaxGas.spec.ts (189.591 s)
 Max Gas
   ✓ should find max gas for wallet (8694 ms)
   ✓ should handle a single loan request to log compute gas (6309 ms)
   ✓ should find max gas for loan when all requests are in the same bucket (18114 ms)
   ✓ should find max gas for loan when all requests are in different buckets (14613 ms)
   ✓ should find max gas for loan when all requests are rejected (6728 ms)
   ✓ should find max gas for loan when recovering stakes (98195 ms)
   ✓ should increase gas fees of loans by 10% (1397 ms)

Test Suites: 9 passed, 9 total
Tests:       82 passed, 82 total
Snapshots:   0 total
Time:        191.916 s, estimated 229 s
Ran all test suites.
```

Changelog

- 2025-03-25 - Initial report
- 2025-03-28 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and and may not be represented as such. No third party is entitled to rely on the report in any any way, including for the purpose of making any decisions to buy or sell a product, product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or or any open source or third-party software, code, libraries, materials, or information to, to, called by, referenced by or accessible through the report, its content, or any related related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.