Quantstamp

Bucket Protocol V2

Executive Summary

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

Type	CDP-backed Stablecoin
Timeline	2025-08-11 through 2025-08-22
Language	Move
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	README.md
Source Code	• https://github.com/Bucket-Protocol/v2-move-contracts #32c4c41
Auditors	• Adrian Koegl Auditing Engineer • Hamed Mohammadi Auditing Engineer • Rabib Islam Senior Auditing Engineer

Documentation quality	Medium
Test quality	High
Total Findings	13 Fixed: 5 Acknowledged: 7 Mitigated: 1
High severity findings	1 Fixed: 1
Medium severity findings	0
Low severity findings	6 Fixed: 3 Acknowledged: 2 Mitigated: 1
Undetermined severity findings	0
Informational findings	6 Fixed: 1 Acknowledged: 5

Summary of Findings

Quantstamp has audited Bucket's v2 protocol, a collateralized debt position-based stablecoin on Sui, implementing several mechanisms to peg `USDB` to the US Dollar. The audit scope included the following packages:

- **bucket_v2_framework**: Provides foundational utilities including fixed-point arithmetic libraries, account abstraction system, and core data structures used throughout the protocol.
- **bucket_v2_usd**: Implements the `USDB` stablecoin with a centralized treasury that manages minting/burning permissions, enforces module-specific supply limits, and collects protocol fees.
- **bucket_v2_oracle**: Delivers a weighted price aggregation system that combines multiple price feed sources with configurable weights.
- **bucket_v2_cdp**: Manages collateralized debt positions through isolated vaults for different collateral types.
- **bucket_v2_psm**: Operates the Peg Stability Module enabling 1:1 swaps between `USDB` and other stablecoins and enforces price tolerance checks to maintain peg stability during market volatility.

The contracts are generally well-designed and demonstrate excellent coding practices. While the protocol is well-equipped to operate during normal market conditions, we are concerned about the incentive mechanisms to support the protocol during extreme market conditions. Particularly, we recommend implementing mechanisms for handling bad debt and improving on the incentive mechanism for liquidations.

During the audit, we have identified 1 high, 6 low, and 6 informational findings, as well as a few suggestions to follow best practices.

Fix Review Summary: The client has addressed all issues by fixing, acknowledging, or mitigating them. Particularly, the high severity issue was fixed and different design decisions were extensively discussed with the client. We can confirm that the design decisions listed as issues were deliberate choices with extensive consideration of trade-offs. We appreciate the communication and awareness of the client in this respect.

Furthermore, the fix review commit contains newly introduced packages and code that were *not* in scope for the audit. Only fixes related to the issues were inspected during the fix review.

ID	DESCRIPTION	SEVERITY	STATUS
BKU-1	Security Level Constraint Can Be Circumvented	● High	Fixed
BKU-2	Non-Deterministic Oracle Selection	● Low	Fixed
BKU-3	Missing Input Validation	● Low	Mitigated
BKU-4	Division Overflow in Float and Double Arithmetic	● Low	Fixed
BKU-5	Protocol only Supports Hard Liquidations	● Low	Acknowledged
BKU-6	Security Levels Lack Flexibility	● Low	Acknowledged
BKU-7	No Protocol-Controlled Oracle Staleness Checks	● Low	Fixed
BKU-8	Collateral Depreciation Can Result in Protocol Debt Spiral	● Informational	Acknowledged
BKU-9	Functions Silently Fail	● Informational	Acknowledged
BKU-10	Consider Freezing `USDB` Metadata Object	● Informational	Acknowledged
BKU-11	Absence of Protocol Design Documentation	● Informational	Fixed
BKU-12	Mean-Sensitive Outlier Detection in Oracle Aggregation	● Informational	Acknowledged
BKU-13	PSM Price Tolerance Creates Trade-Off Between Reserve Protection and Peg Stability	● 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

We have audited the `.move` files in the included directories, although two files were explicitly out of scope. While the `price_rules` directory was out of scope, we nonetheless performed a quick review since it was needed for context, identifying issues with the implementation.

Files Included

Repo: https://github.com/Bucket-Protocol/v2-move-contracts(add460fea2b41f7197daf001dd4a2e241e00c2f3)

Files:
- `bucket_framework/*` (two files excluded)
- `bucket_usd/*`
- `bucket_oracle/*`
- `bucket_cdp/*`
- `bucket_psm/*`

Files Excluded

Repo: https://github.com/Bucket-Protocol/v2-move-contracts(add460fea2b41f7197daf001dd4a2e241e00c2f3)

Files:
- `bucket_framework/sources/liability.move`
- `bucket_framework/sources/sheet.move`
- `testnet_only/*`
- `price_rules/*`

Operational Considerations

- The `AdminCap` holder can mint unlimited tokens by adding new modules with arbitrary supply limits.
- We assume any protocol upgrades undergo thorough security audits before deployment.
- We assume critical parameters are configured appropriately to ensure reliable protocol operation:
  - Price tolerance in PSM module
  - Interest rates in vaults
  - Collateral ratios in vaults
  - Outlier tolerance in oracles
- Oracles and their weights must be chosen carefully, as oracles that depend on spot prices from single sources, like a DEX, are highly vulnerable to manipulation – flash loans can be used to manipulate the price and exploit the difference via e.g. liquidations of positions that become undercollateralized due to momentary price movements
- The PSM is designed to perform swaps within a price range. When the price is outside this range, swaps via the PSM are no longer possible.
- We trust the security level mechanism will not be misused to freeze legitimate user operations during non-emergency situations.

Key Actors And Their Capabilities

bucket_usd

- `AdminCap` **Holder**
    - Set module supply limits and add/remove module versions
    - Change treasury beneficiary address
- **Treasury Beneficiary**
    - Claim all collected protocol fees including interest, liquidation fees, and PSM fees

bucket_oracle

- `ListingCap` **Holder**
    - Create and register new price aggregators for assets
    - Set oracle weights and adjust weight thresholds
    - Configure outlier tolerance for price manipulation resistance

bucket_cdp

- `AdminCap` **Holder**
    - Create new vaults with custom parameters
    - Set interest rates and modify collateral ratios
    - Change liquidation rules and manage vault supply limits
    - Control module interactions through request/response checklists
- **Vault Managers (ACL)**
    - Set vault security levels based on assigned manager level
    - Freeze vault operations during emergencies (level 1 blocks all, level 2 allows deposits/liquidations)
- **Liquidation Rule Modules**
    - Execute liquidations on unhealthy positions
    - Must match vault's configured liquidation rule type

bucket_psm

- `AdminCap` **Holder**
    - Create PSM pools for different stablecoins
    - Set swap fees including partner-specific rates
    - Adjust price tolerance thresholds for 1:1 swap eligibility

Findings

BKU-1 Security Level Constraint Can Be Circumvented

● High ⓘ Fixed

Update

Fixed by the client as per recommendation.
Addressed in: `49f5916fb915743b929b5c5d28d2647a0e24d14e`.

File(s) affected: `bucket_cdp/sources/vault.move`

Description: The `update_position()` function throws an error depending on the user's operation and the vault's security level. It is *intended* that if the user wants to deposit collateral, the user is allowed if the security level is 0 or 2; if the user wants to withdraw collateral, repay a debt, or borrow, the security level must be 0. This behavior is based on the following code block:

```
// check security by actions
    if(request.deposit_amount() > 0) {
        // deposit actions will only be blocked when security level equals to 1
        vault.check_secutiry_level(1);
    }else{
        // borrow; repay; witdraw;
        vault.check_secutiry_level(2);
    };
```

However, it is possible to withdraw collateral, repay a debt, or borrow even if the security level is 2: the user simply needs to include a deposit amount with their call of `update_position()`. This way, the security level access control is circumvented.

Recommendation: Instead of an `if - else` condition, implement independent `if` statements that depend on inputs to `update_position()`, since several operations are supported at once in `update_position()`.

BKU-2 Non-Deterministic Oracle Selection

● Low ⓘ Fixed

Update

Fixed by the client by requiring that users need to collect the prices of *all* price feeds. This prevents users from "cherry-picking" favorable prices.
Addressed in: `d8badbbeea0e666415187a47f065c9c0f3b2c6c0`.

File(s) affected: `bucket_oracle/sources/aggregator.move`

Description: The oracle system allows users to select different subsets of price feeds to reach the weight threshold, enabling them to obtain multiple valid prices within the same transaction. While we did not identify a direct exploitation path in the current CDP and PSM modules (which only use prices for health checks and divergence thresholds), this non-deterministic design violates oracle best practices and could become exploitable if the protocol evolves. Furthermore, sophisticated users can currently pick a subset of oracles that is most advantageous to them.

The current implementation enables scenarios where a user with oracles weighted 1/1/1 and a threshold of 2 can choose between any two, potentially obtaining different aggregated prices. This flexibility becomes particularly concerning during periods of high volatility when oracle prices naturally diverge due to update delays or network conditions.

Recommendation: Implement a deterministic oracle selection model following industry standards. A robust approach uses primary, secondary, and fallback oracles where the primary and secondary prices must agree within a threshold, with the fallback used only when they diverge. This ensures price consistency by requiring two out of three oracles to provide similar values.

BKU-3 Missing Input Validation

● Low ⓘ Mitigated

Update

Mitigated by the client. Only a check for `min_collateral_ratio_bps` was applied in `bucket_v2_cdp::vault`.
Addressed in: `365d283bdda523b5fa5bc3996182085cf8ab3113`.

File(s) affected: `bucket_cdp/sources/vault.move`, `bucket_oracle/sources/aggregator.move`, `bucket_psm/sources/pool.move`, `bucket_framework/sources/account.move`

Description: Human error may lead to erroneous inputs that drive unexpected protocol behavior. As such, input validation is important to include in the code to prevent incorrect inputs from influencing the protocol.

Many of the inputs are integers, some of which are formatted as basis points (bps). In most cases, basis points should not exceed 10000 (i.e. 100%). However, inputs above this number are possible in the following instances:

1. `vault`
   1. `create()`
      1. `interest_rate_bps`
      2. `min_collateral_ratio_bps`
   2. `set_interest_rate()`
      1. `interest_rate_bps`
2. `aggregator`
   1. `new()`
      1. `outlier_tolerance_bps`
3. `pool`
   1. `new()`
      1. `swap_in_fee_rate_bps`
      2. `swap_out_fee_rate_bps`
      3. `price_tolerance_bps`
   2. `set_fee_config()`
      1. `swap_in_fee_rate_bps`
      2. `swap_out_fee_rate_bps`
   3. `set_price_tolerance()`
      1. `tolerance_bps`
4. `account`
   1. `new()`
      1. `alias` (trim & sanitize)

Recommendation: Add the corresponding checks.

BKU-4 Division Overflow in Float and Double Arithmetic

• Low ⓘ Fixed

Update

Fixed by adding pre-checks: Both `div()` functions will revert if the value of `a` is too large.
Addressed in commits `8d0e536e169268b84dadca576905b1c9adb8fc3c`.

File(s) affected: `bucket_framework/sources/float.move`, `bucket_framework/sources/double.move`

Description: The `div()` functions in both `float` and `double` modules can trigger runtime panics when dividing large values. The division performs `(a.value * WAD) / b.value`, which overflows when `a.value * WAD` exceeds the type's maximum value (u128 for float, u256 for double). This creates an overflow threshold of approximately `u128::MAX / 10^9` for float operations.

This inconsistent error handling could cause transaction failures in critical protocol operations involving large values.

Exploit Scenario:
Here is a test case for `float.move` that expect the arithmetic error to occur in the `div()` function:

```
#[test, expected_failure(arithmetic_error, location = Self)]
  fun test_div_overflow_demonstration() {
    let overflow_threshold = std::u128::max_value!() / WAD + 1;
    let large_float = from_scaled_val(overflow_threshold);
    let _result = large_float.div(one());
  }
```

Recommendation: Implement explicit overflow protection by either casting intermediate calculations to larger types (u256 for float) or adding pre-checks that return descriptive errors before overflow occurs. This ensures predictable behavior and prevents unexpected transaction failures during critical protocol operations.

BKU-5 Protocol only Supports Hard Liquidations

• Low ⓘ Acknowledged

Update

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

```
On purpose. Will upgrade to soft liquidation if needed using the LiquidationRule witness.
```

File(s) affected: `bucket_cdp/sources/vault.move`

Description: The liquidation mechanism forces complete position closure regardless of how slightly a position breaches the minimum collateralization ratio. For example, a position at 149% collateralization faces total liquidation despite needing only minimal deleveraging to restore a 150% threshold. This all-or-nothing approach causes unnecessarily severe user losses and discourages efficient capital utilization near the collateralization boundaries.

The implementation allows liquidators to specify repayment amounts but always liquidates proportionally across the entire position rather than targeting a healthy collateralization ratio. This design may hold back users who are looking for optimized capital efficiency, since their position would be extremely risky.

Recommendation: Implement partial liquidation logic that calculates and liquidates only the minimum collateral needed to restore a position to a safe collateralization level.

BKU-6 Security Levels Lack Flexibility

• Low ⓘ Acknowledged

Update

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

```
Keep it simple for manager to pause according to different severity level.
```

File(s) affected: `bucket_cdp/sources/vault.move`

Description: The vault implements a three-level security system that provides insufficient granularity for emergency responses. Level 0 allows all operations, level 1 blocks everything, and level 2 blocks borrowing, repaying, and withdrawing while still allowing deposits and liquidations. This design may force administrators to disable critical functions unnecessarily.

The current implementation prevents nuanced responses to specific vulnerabilities. At level 1, even beneficial actions like debt repayments and liquidations are blocked, which could worsen a crisis by preventing deleveraging and position cleanup. At level 2, repayments are blocked alongside withdrawals and borrows, preventing users from voluntarily reducing their debt during emergencies. A withdrawal vulnerability should not require stopping repayments, and a borrow exploit shouldn't prevent users from improving their positions.

Recommendation: We suggest implementing a bitmap-based permission system where each operation can be individually enabled or disabled. For example, use bit flags for deposit (0×1), withdraw (0×2), borrow (0×4), repay (0×8), and liquidate (0×10). This allows precise control during emergencies, such as disabling only withdrawals while maintaining liquidations and repayments to preserve protocol health.

BKU-7 No Protocol-Controlled Oracle Staleness Checks

• Low ⓘ Fixed

Update

Fixed by enforcing freshness in `pyth.move`. It should be noted, however, that these freshness checks depend on the implementation of the respective price feed module added in the future.
Addressed in: `31552fce89681883823157ac001f85ead6abf662`.

File(s) affected: `bucket_oracle/sources/collector.move`, `price_rules/pyth_rule/sources/pyth_rule.move`

Description: The protocol delegates price freshness validation entirely to external price providers rather than enforcing its own staleness requirements. Currently, only Pyth is implemented as a price source, and the protocol relies on Pyth's internal staleness checks without the ability to configure acceptable price age limits. This dependency prevents the protocol from adjusting freshness requirements based on market conditions or risk parameters. Furthermore, price feed providers, such as Pyth, will panic if the price is outdated, leaving the protocol unable to revert to other oracle providers. The lack of protocol-level control also complicates adding new price sources with different staleness guarantees.

Recommendation: Implement protocol-controlled staleness checks using `pyth::get_price_no_older_than()` with configurable maximum age parameters during `collector::collect()`. This allows the protocol to enforce consistent freshness requirements across all price sources and gracefully handle stale prices by falling back to alternative oracles rather than reverting transactions.

BKU-8
Collateral Depreciation Can Result in Protocol Debt Spiral

• Informational Acknowledged

Update

The client provided the following explanation:

```
Centralization method to buy back the bad debt.
```

Update

This issue was extensively discussed with the client. The client takes all measures in preventing bad debt and is confident in their abilities to quickly liquidate positions. In case bad debt should accumulate despite all measures, it will be covered by the client through liquidation of underwater positions at additional cost.

File(s) affected: `bucket_cdp/sources/vault.move`

Description: The protocol lacks mechanisms to handle positions becoming undercollateralized. During normal market conditions, positions are typically liquidated before reaching this state. However, when collateral prices rapidly decline (which is fairly common in the cryptocurrency market), liquidations may not execute quickly enough. The `liquidate()` function only provides proportional collateral, meaning liquidators are only incentivized while positions carry more collateral relative to their debt.

Once positions go underwater, the protocol depends entirely on altruistic liquidators willing to absorb losses. Even with centralized liquidators incentivized to maintain protocol health, this design creates systemic risk. Furthermore, the user whose position is now undercollateralized has no incentive to pay back the debt. Users aware of this vulnerability may trigger bank runs during market stress, causing cascading withdrawals that further destabilize the stablecoin's collateralization.

Exploit Scenario:

In the following, we discuss the potential impact of swift market movements on protocol health, illustrating the protocol's fragility in the absence of an embedded mechanism to alleviate bad debt.

Consider Alice who opens a position with $1,500 collateral to mint 1,000 `USDB`. A flash crash drops her collateral value to $600. At this point, neither Alice nor any rational liquidator has incentive to close the position - Alice won't repay $1,000 debt to recover $600 collateral, and liquidators won't pay $1,000 to receive $600 worth of assets. The position remains open indefinitely, continuously accruing interest that further inflates the protocol's unbacked debt burden.

As more positions become underwater during the crash, bad debt accumulates across the protocol. Sophisticated traders recognize the growing insolvency and begin buying `USDB` at a discount on exchanges (e.g., at $0.95) and immediately redeeming it through the `PSM` for $1.00 worth of stablecoins, extracting value from the PSM modules. This arbitrage continues until `PSM` reserves are exhausted or the oracle price of `USDB` becomes so low that the PSM no longer facilitates swapping. In either case, `USDB` loses its redemption backstop and enters free fall, potentially declining to the actual collateral backing ratio.

Recommendation: Design and implement a bad debt recovery system. In the following, we offer best practices to address this issue long-term:

1. **Liquidation Incentive**: Create a mechanism to incentivize clearing underwater positions by minting additional `USDB` as "protocol debt" to compensate liquidators.
2. **Protocol Reserve**: Redirect a portion of all protocol fees (interest accrual, future liquidation penalties, PSM swap fees) into a dedicated reserve buffer that can absorb bad debt. If "protocol debt" is ever created, a dedicated function can cover it using the protocol reserves.
3. **Liquidation Fees**: Take an additional fee from liquidations that goes towards the protocol reserve buffer.
4. **Emergency Recapitalization**: Implement debt auctions where governance tokens (if introduced) can be minted and sold to cover bad debt, and/or allow direct recapitalization contributions from the treasury or foundation during extreme events.

These mechanisms are essential for maintaining user confidence. Historical CDP failures demonstrate that protocols without proper bad debt handling may experience trust erosion, leading to bank runs that become self-fulfilling prophecies of protocol failure.

BKU-9 Functions Silently Fail

• Informational Acknowledged

Update

Acknowledged by the client.
The client provided the following explanation:

```
Abort for unchanged state seems unnecessary. And in PTB if a moveCall abort the whole
transaction fails. For example, we want to add checks to all CDP vaults and put all CDP vaults
in transaction but one of the vault is already get added then we don't want the transaction to
fail.
```

File(s) affected: `bucket_framework/sources/sheet.move`, `bucket_cdp/sources/vault.move`, `bucket_usd/sources/usdb.move`, `bucket_cdp/sources/acl.move`, `bucket_oracle/sources/collector.move`

Description: There are a number of `public` functions in the codebase that may be called with the intention of effecting some change in state, but that provide no feedback when there is no change. This can surprise users who expect some sort of feedback when a function does not do what it is supposed to do in the mind of the user.

The following is a list of instances:

1. `sheet`
   1. `add_creditor()`
   2. `add_debtor()`
   3. `ban()`
   4. `unban()`
2. `vault`
   1. `add_request_check()`
   2. `remove_request_check()`
   3. `add_response_check()`
   4. `remove_response_check()`
3. `usdb`
   1. `remove_module()`
   2. `remove_version()`
4. `acl`
   1. `remove_role()`
5. `collector`
   1. `remove()`

Recommendation: Have each function either return a boolean indicating whether state was changed or throw an error if no state was changed.

BKU-10 Consider Freezing `USDB` Metadata Object

● Informational ⓘ Acknowledged

Update

Acknowledged by the client.
The client provided the following explanation:

```
On purpose. In case we want to update it in the future.
```

File(s) affected: `bucket_usd/sources/usdb.move`

Description: The `bucket_v2_usd::usdb::init()` function uses `coin::create_currency()` to initialize `USDB`. The function then uses `transfer::share_object()` to share the object instead of freezing it. A better approach would be to freeze the object.

Recommendation: Unless changes to the coin metadata are intended, consider freezing this object.

BKU-11 Absence of Protocol Design Documentation

● Informational ⓘ Fixed

Update

The client added a design documentation explaining intentions. This could be extended with graphics and diagrams.
Addressed in: `75886a65030e81687b58c743e394668a16508d3f`.

Description: The protocol lacks documentation explaining its design decisions, making it difficult to evaluate whether the chosen mechanisms will maintain peg stability. We found no rationale for critical choices like using instant liquidations instead of auctions, allowing users to select oracle subsets, implementing PSM price tolerance thresholds, or excluding liquidation penalties entirely. Without understanding the team's intentions and trade-offs, we cannot properly assess whether our recommendations align with the protocol's goals or if certain "issues" are actually deliberate design choices.

This gap is particularly concerning for a CDP protocol where seemingly minor parameter choices can trigger cascading failures during market stress. The absence of economic modeling documentation or stress test scenarios leaves both auditors and future operators guessing about expected behavior under extreme conditions.

Recommendation: Document the reasoning behind key design decisions, including why certain industry-standard mechanisms were excluded and what alternatives were considered. This should cover oracle architecture choices, liquidation mechanism selection, PSM parameters, and the economic models used to validate these decisions. Clear documentation helps distinguish bugs from features and ensures the protocol operates as intended.

BKU-12
Mean-Sensitive Outlier Detection in Oracle Aggregation

● Informational ⓘ Acknowledged

Update
Acknowledged by the client.

File(s) affected: `bucket_oracle/sources/aggregator.move`

Description: The outlier detection mechanism uses average-based comparison, which is inherently unpredictable. When calculating outliers based on the weighted average of all submitted prices, the reference point for determining outliers shifts based on the distribution of prices themselves, creating mean-sensitivity that makes the final price non-deterministic. This approach can lead to unexpected oracle rejections or acceptances based on the specific price values submitted.

During periods of high volatility when oracle prices naturally diverge, a borderline outlier oracle might be included or excluded depending on other oracles' values. For example, with three oracles reporting $100, $101, and $120 with a 10% tolerance threshold: the mean of $107 causes the $120 oracle to be filtered (11.2% deviation), resulting in a final price of $100.5. However, if the third oracle reported $115 instead, the mean of $105.3 keeps all oracles (9.5% deviation), resulting in a final price of $105.3. This 5% difference in aggregated price stems from mean-shift sensitivity rather than actual market conditions.

Recommendation: For outlier detection, replace the average-based approach with pairwise compatibility checks that determine if any two oracles agree within acceptable bounds, eliminating the mean-sensitivity issues of average-based filtering.

Alternatively, use median-based filtering which is more robust against extreme values and provides more predictable outlier detection behavior.

BKU-13
PSM Price Tolerance Creates Trade-Off Between Reserve Protection and Peg Stability

● Informational ⓘ Acknowledged

Update
Acknowledged by the client.
The client provided the following explanation:

```
Will set it around 1% to deal with only extraordinary circumstances.
```

File(s) affected: `bucket_psm/sources/pool.move`

Description: The `PSM`'s price tolerance mechanism, which disables swaps when the oracle price deviates beyond a threshold from $1, presents a fundamental design trade-off that lacks clear resolution. This feature has both protective and destabilizing effects that become particularly significant during market stress.

From a reserve protection perspective, the threshold serves as a critical safeguard. When `USDB` depegs significantly, unrestricted `PSM` swaps would allow arbitrageurs to drain the entire reserve by exchanging devalued `USDB` for fully-valued stablecoins at a 1:1 rate. Given the protocol's lack of bad debt recovery mechanisms, preserving `PSM` reserves becomes essential for maintaining any semblance of stability. The threshold effectively acts as a circuit breaker, preventing complete reserve depletion during severe depegging events.

However, this same mechanism may accelerate the very crisis it aims to prevent. When `USDB` begins depegging and crosses the threshold, the PSM becomes completely non-functional, eliminating the primary arbitrage mechanism that would normally restore the peg. Without `PSM` support, even minor depegs could spiral out of control. Users observing the disabled `PSM` might panic, rushing to close their `CDP` s before conditions worsen, creating selling pressure that drives `USDB` further from its peg. This self-reinforcing cycle could transform a manageable deviation into a downwards spiral.

Recommendation: The optimal approach depends on broader protocol design decisions. If bad debt recovery mechanisms are implemented, consider removing or significantly increasing the price tolerance to prioritize peg stability. The `PSM` could then fulfill its role as a stability mechanism even during stress, with bad debt handled through separate systems. Alternatively, if maintaining the threshold, implement graduated responses rather than binary on/off states - for example, increasing fees progressively as prices deviate, or limiting swap sizes rather than disabling entirely. This would preserve some stabilizing effect while still protecting reserves during extreme events.

Auditor Suggestions

S1 Improve Code Documentation

Mitigated

Update

Mitigated by the client by adding README documentation for some modules. However, in-line documentation, especially for extensive structs, could be extended.
Addressed in: `a93f90bd3aae75990d23c8a6b2295be7cbd0e416`, `f9e0c72cd66fad660cca763b6d2f2a8125feaa32`, and `51ade01e481694a981f3273b5c384c063f868f43`.

Description: In-line code documentation could be improved across the codebase. The following are some deficiencies:

1. Not every set of modules comes with a `README`. These should be included in order to provide high-level context regarding different codebase components.
2. Structs should be thoroughly commented. For instance, each struct should:
   1. Carry a comment that defines its purpose;
   2. Carry comments describing each field in the struct.

Recommendation: Improve the documentation accordingly.

S2 Unused Code

Acknowledged

Update

The client provided the following explanation:

```
For future upgrade.
```

Description: `sheet::Request` has a `checklist` field of type `Option<vector<Entity>>` that is never used for any notable feature. This is an instance of incomplete code that may be done away with. Furthermore, the `pool::Pool` struct contains a `sheet: Sheet<T, BucketV2PSM>` field that is never used in any pool operations.

Both imported modules were not in scope for this audit.

Recommendation: Incomplete code should be either removed or completed depending on the desired implementation.

S3 Typos

Fixed

Update

Typos were fixed by the client.
Addressed in: `2a707ef0a97225e78f9168f8f75ecaec87676fcb`.

File(s) affected: `bucket_cdp/sources/vault.move`, `bucket_framework/sources/sheet.move`

Description: There are a number of typos throughout the codebase:

1. The function `check_secutiry_level()` should be renamed `check_security_level()`.
2. The argument `_liqudation_rule` in `vault::liquidate()` should be renamed `_liquidation_rule`.

3. The field `payor_debts` in `sheet::Request` should be renamed `payer_debts`.
4. Replace instances of `payor` with `payer`.

Recommendation: Correct the variable and function names accordingly.

S4 Lack of Event Emission for Several State-Changing Functions

Mitigated

Update

Mitigated by the client. Events were added for `set_security_by_admin()` and `set_security_by_manager_level()` in `vault.move`.
Addressed in: `589939faa004c8e6e8eb25866dd3c51ee5eeb336`.

File(s) affected: `bucket_psm/sources/pool.move`, `bucket_cdp/sources/vault.move`, `bucket_usd/sources/usdb.move`

Description: Throughout the code base, there are many administrative functions that update important state variables in shared objects. However, not all of them emit the proper events. A non-exhaustive list of such instances is provided below:

1. In the `bucket_v2_usd::usdb` module, the following functions lack event emission:
   - `set_supply_limit()`
   - `set_beneficiary_address()`
   - `add_version()`
   - `remove_version()`
   - `remove_module()`
2. In the `bucket_v2_cdp::vault` module, the following functions lack event emission:
   - `set_interest_rate()`
   - `set_manager_role()`
   - `remove_manager_role()`
   - `set_security_by_admin()`
   - `set_security_by_manager_level()`
3. In the `bucket_v2_psm::pool` module, the following functions lack event emission:
   - `new()`
   - `set_fee_config()`
   - `set_price_tolerance()`

Recommendation: Consider emitting more events throughout the code base to improve off-chain monitoring.

S5 Redundant Balance Tracking

Acknowledged

Update

Acknowledged by the client.
The client provided the following explanation:

```
On purpose. For example if we want to add flash loan on PSM pool in the future, the balance
will changes during the flash loan but the balance_amount still holds the correct value.
```

File(s) affected: `bucket_psm/sources/pool.move`

Description: The PSM pool maintains both a `Balance<T>` object and a separate `balance_amount: u64` field that duplicates the same information. This inflicts redundant updates on every balance update.

Recommendation: Remove the `balance_amount` field and directly query `balance.value()` when needed. This eliminates redundancy.

S6 Misleading Wad Constant Name

Fixed

Update

Fixed by the client by changing the variable name to `PRECISION`.
Addressed in: `344ffa0aa021f0da452e3488be14911fd5e9abc9`.

File(s) affected: `bucket_framework/sources/float.move`

Description: The constant `WAD` in `float.move` uses 9 decimal places, contradicting the established Ethereum convention where WAD represents 18 decimals. This misuse of standard terminology could confuse developers familiar with DeFi conventions.

Recommendation: Rename the constant to `PRECISION` or `SCALE` to accurately reflect its 9-decimal precision and avoid confusion with Ethereum's WAD standard.

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 pose 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).

Test Suite Results

We successfully ran all 73 tests across the 5 packages.

Fix Review Update: Across all packages, 3 test cases were added, with negligible impact on test coverage.

Fix Review Update 2: A total of 111 tests were successfully run across all packages.

```
**bucket_framework**
[ PASS    ] bucket_v2_framework::liability::test_destroy_non_zero_credit
[ PASS    ] bucket_v2_framework::liability::test_destroy_non_zero_debt
[ PASS    ] bucket_v2_framework::linked_table::borrow_missing
[ PASS    ] bucket_v2_framework::linked_table::borrow_mut_missing
[ PASS    ] bucket_v2_framework::float::test_advenced
[ PASS    ] bucket_v2_framework::liability::test_settle
[ PASS    ] bucket_v2_framework::float::test_basic
[ PASS    ] bucket_v2_framework::linked_table::destroy_non_empty
[ PASS    ] bucket_v2_framework::float::test_div_by_zero
[ PASS    ] bucket_v2_framework::float::test_div_u64_by_zero
[ PASS    ] bucket_v2_framework::float::test_fraction_by_zero
[ PASS    ] bucket_v2_framework::double::test_advenced
[ PASS    ] bucket_v2_framework::double::test_basic
[ PASS    ] bucket_v2_framework::linked_table::front_back_empty
[ PASS    ] bucket_v2_framework::float::test_number_too_large_when_add
[ PASS    ] bucket_v2_framework::double::test_div_by_zero
[ PASS    ] bucket_v2_framework::sheet_tests::test_banned_creditor
[ PASS    ] bucket_v2_framework::double::test_div_u64_by_zero
[ PASS    ] bucket_v2_framework::double::test_fraction_by_zero
[ PASS    ] bucket_v2_framework::float::test_number_too_large_when_ceil
[ PASS    ] bucket_v2_framework::double::test_number_too_large_when_add
[ PASS    ] bucket_v2_framework::sheet_tests::test_banned_debtor
[ PASS    ] bucket_v2_framework::double::test_number_too_large_when_ceil
[ PASS    ] bucket_v2_framework::float::test_number_too_large_when_div
[ PASS    ] bucket_v2_framework::linked_table::pop_back_empty
[ PASS    ] bucket_v2_framework::float::test_number_too_large_when_mul
[ PASS    ] bucket_v2_framework::double::test_number_too_large_when_div
[ PASS    ] bucket_v2_framework::float::test_number_too_large_when_mul_u64
[ PASS    ] bucket_v2_framework::double::test_number_too_large_when_mul
[ PASS    ] bucket_v2_framework::float::test_number_too_large_when_round
[ PASS    ] bucket_v2_framework::float::test_pow
[ PASS    ] bucket_v2_framework::linked_table::pop_front_empty
[ PASS    ] bucket_v2_framework::float::test_sub_too_much
[ PASS    ] bucket_v2_framework::double::test_number_too_large_when_mul_u64
[ PASS    ] bucket_v2_framework::double::test_number_too_large_when_round
[ PASS    ] bucket_v2_framework::account::test_alias_length_too_long_when_create
[ PASS    ] bucket_v2_framework::double::test_pow
[ PASS    ] bucket_v2_framework::double::test_sub_too_much
[ PASS    ] bucket_v2_framework::linked_table::push_back_duplicate
[ PASS    ] bucket_v2_framework::sheet_tests::test_checklist_not_fulfill
[ PASS    ] bucket_v2_framework::sheet_tests::test_not_creditor
[ PASS    ] bucket_v2_framework::linked_table::push_back_singleton
[ PASS    ] bucket_v2_framework::sheet_tests::test_not_debtor
[ PASS    ] bucket_v2_framework::linked_table::push_front_duplicate
[ PASS    ] bucket_v2_framework::sheet_tests::test_pay_too_much
[ PASS    ] bucket_v2_framework::account::test_alias_length_too_long_when_update
[ PASS    ] bucket_v2_framework::linked_table::push_front_singleton
[ PASS    ] bucket_v2_framework::linked_table::push_mixed_duplicate
[ PASS    ] bucket_v2_framework::linked_table::remove_missing
[ PASS    ] bucket_v2_framework::linked_table::sanity_check_contains
[ PASS    ] bucket_v2_framework::linked_table::sanity_check_drop
[ PASS    ] bucket_v2_framework::linked_table::sanity_check_size
[ PASS    ] bucket_v2_framework::sheet_tests::test_sheet
[ PASS    ] bucket_v2_framework::account::test_init
[ PASS    ] bucket_v2_framework::linked_table::simple_all_functions
[ PASS    ] bucket_v2_framework::linked_table::test_all_orderings
[ PASS    ] bucket_v2_framework::linked_table::test_insertable_linked_table
Test result: OK. Total tests: 57; passed: 57; failed: 0

**bucket_usd**
[ PASS    ] bucket_v2_usd::limited_supply::test_supply_not_enough
[ PASS    ] bucket_v2_usd::bucket_v2_usd_tests::test_bucket_v2_usd
[ PASS    ] bucket_v2_usd::bucket_v2_usd_tests::test_coin_type_not_found
[ PASS    ] bucket_v2_usd::bucket_v2_usd_tests::test_destroy_non_empty_supply
[ PASS    ] bucket_v2_usd::bucket_v2_usd_tests::test_invald_module
[ PASS    ] bucket_v2_usd::bucket_v2_usd_tests::test_invald_module_version
[ PASS    ] bucket_v2_usd::bucket_v2_usd_tests::test_not_beneficiary
[ PASS    ] bucket_v2_usd::bucket_v2_usd_tests::test_supply_exceed_limit
Test result: OK. Total tests: 8; passed: 8; failed: 0

**bucket_oracle**
[ PASS    ] bucket_v2_oracle::bucket_v2_oracle_tests::test_bucket_v2_oracle
[ PASS    ] bucket_v2_oracle::bucket_v2_oracle_tests::test_coin_type_already_listed
[ PASS    ] bucket_v2_oracle::bucket_v2_oracle_tests::test_create_with_invalid_threshold
[ PASS    ] bucket_v2_oracle::bucket_v2_oracle_tests::test_total_invalid_weight
[ PASS    ] bucket_v2_oracle::bucket_v2_oracle_tests::test_total_weight_not_enough
[ PASS    ] bucket_v2_oracle::bucket_v2_oracle_tests::test_update_with_invalid_threshold
[ PASS    ] bucket_v2_oracle::bucket_v2_oracle_tests::test_zero_total_weight
Test result: OK. Total tests: 7; passed: 7; failed: 0

**bucket_cdp**
```

```
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_borrow_without_oracle_price
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_debtor_not_found
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_debtor_not_found_raw
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_invalid_liquidation
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_long_term_position
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_minor_security_for_available_liquidation
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_minor_security_for_deposit
[ PASS    ]
bucket_v2_cdp::bucket_v2_cdp_tests::test_minor_security_for_deposit_with_positive_borrow
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_minor_security_for_liquidation
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_minor_security_for_other_actions
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_missing_request_witness
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_missing_response_witness
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_overall
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_position_is_healthy
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_position_is_not_healthy
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_position_order
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_remove_manager_role_not_manager_err
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_repay_too_much
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_set_manager_role_err_invalid_level
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_set_manager_role_with_minor_level
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_set_manager_role_with_revoke_security_
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_set_manager_role_with_strictest_level
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_strictest_security_for_deposit_action
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_strictest_security_for_liquidation
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_strictest_security_for_other_action
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_update_manager_role
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_withdraw_too_much
[ PASS    ] bucket_v2_cdp::bucket_v2_cdp_tests::test_withdraw_without_oracle_price
Test result: OK. Total tests: 28; passed: 28; failed: 0

**bucket_psm**
[ PASS    ] bucket_v2_psm::bucket_v2_psm_tests::test_exceed_limit_err
[ PASS    ]
bucket_v2_psm::bucket_v2_psm_tests::test_fluctuating_downward_price_out_of_tolerance_err
[ PASS    ] bucket_v2_psm::bucket_v2_psm_tests::test_fluctuating_price_with_downward_direction
[ PASS    ] bucket_v2_psm::bucket_v2_psm_tests::test_fluctuating_price_with_upward_direction
[ PASS    ]
bucket_v2_psm::bucket_v2_psm_tests::test_fluctuating_upward_price_out_of_tolerance_err
[ PASS    ] bucket_v2_psm::bucket_v2_psm_tests::test_pool_main_
[ PASS    ] bucket_v2_psm::bucket_v2_psm_tests::test_pool_main_update_rate
[ PASS    ] bucket_v2_psm::bucket_v2_psm_tests::test_pool_main_with_decimal_3
[ PASS    ] bucket_v2_psm::bucket_v2_psm_tests::test_pool_main_with_decimal_9
[ PASS    ] bucket_v2_psm::bucket_v2_psm_tests::test_pool_with_partner_main
[ PASS    ] bucket_v2_psm::bucket_v2_psm_tests::test_swap_out_with_insufficient_pool
Test result: OK. Total tests: 11; passed: 11; failed: 0
```

Code Coverage

Most packages achieved a coverage result of > 80%, while the `bucket_v2_framework` package only achieved a coverage of ~70%. We recommend increasing the test coverage for all packages to 90%+, and ideally 100%.

Fix Review Update 2: The coverage was increased to >90% across all packages, improving the overall test suite quality.

---

bucket_framework

Module	Coverage
bucket_framework::account	87.65%
bucket_framework::float	100.00%
bucket_framework::double	100.00%
bucket_framework::liability	100.00%
bucket_framework::linked_table	100.00%
bucket_framework::sheet	100.00%
Total	99.50%

bucket_usd

Module	Coverage
bucket_usd::admin	100.00%
bucket_usd::treasury	100.00%
bucket_usd::limited_supply	75.26%
bucket_usd::usdb	95.55%
Total	92.14%

bucket_oracle

Module	Coverage
bucket_oracle::result	100.00%
bucket_oracle::listing	100.00%
bucket_oracle::collector	91.30%
bucket_oracle::aggregator	95.98%
Total	95.88%

bucket_cdp

Module	Coverage

bucket_cdp::acl	96.30%
bucket_cdp::events	91.01%
bucket_cdp::memo	100.00%
bucket_cdp::request	88.33%
bucket_cdp::response	86.00%
bucket_cdp::witness	100.00%
bucket_cdp::version	100.00%
bucket_cdp::vault	90.20%
Total	90.36%

bucket_psm

Module	**Coverage**
bucket_psm::events	100.00%
bucket_psm::memo	100.00%
bucket_psm::witness	100.00%
bucket_psm::version	100.00%
bucket_psm::pool	92.54%
Total	93.16%

Changelog

- 2025-08-25 - Initial report
- 2025-09-03 - Final Report
- 2025-09-12 - Final Report v2

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

Notice of confidentiality

Links to other websites

Disclaimer

Quantstamp