Quantstamp Security Assessment Certificate

July 15th 2022 — Quantstamp Verified

Casper Labs (Phase 1)

This smart contract audit was prepared by Quantstamp, the protocol for securing smart contracts.

Executive Summary

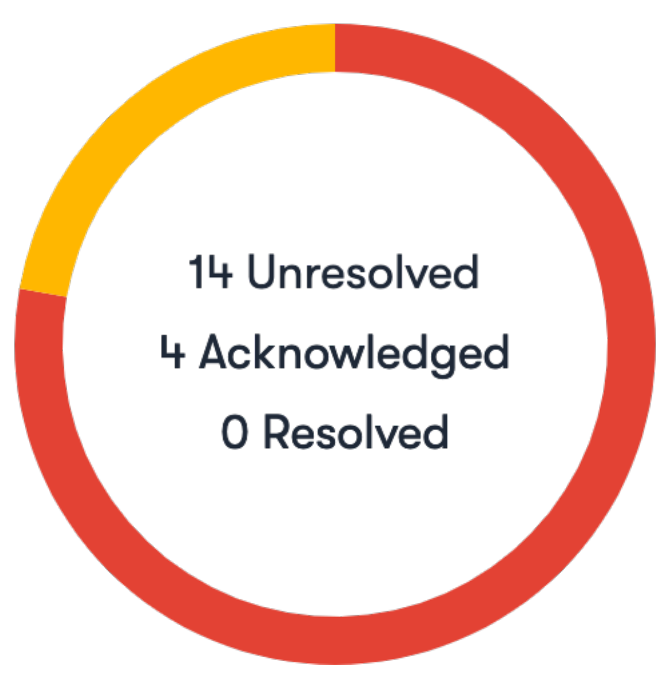
Type	Blockchain
Auditors	Leonardo Passos, Senior Research Engineer Jake Goh Si Yuan, Senior Security Researcher Luís Fernando Schultz Xavier da Silveira, Security Consultant
Timeline	2021-03-01 through 2021-04-01
Languages	Rust
Methods	Manual Review
Specification	Online docs
Documentation Quality	Medium
Test Quality	Undetermined
Source Code	

Repository	Commit
casper-node	cb1d20a

Goals

- Can funds be lost or be locked?
- Does the code adhere to the online documentation?

Total Issues	18	(0 Resolved)
High Risk Issues	4	(0 Resolved)
Medium Risk Issues	1	(0 Resolved)
Low Risk Issues	8	(0 Resolved)
Informational Risk Issues	0	(0 Resolved)
Undetermined Risk Issues	5	(0 Resolved)



High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	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.

Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
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).
Fixed	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

This report contains the audit findings of the system contracts of the Casper Labs node repository. The audit code includes all the system contracts under `types/src/system/*` and `smart_contracts/contracts/client/*` (except `counter-define`). Any code outside these folders is not in the scope of this audit (e.g., consensus, network layer, execution environment, etc).

Altogether, we found 18 issues of varying severity, four of which are of high concern. Those include authentication, integer overflow, performance degradation (with potential network stall), and softer slashing policy in comparison to what the provided specification states. All other issues (14) are medium severity (1) , low severity (8), and the remaining five are undetermined (we could not state their consequences). In addition to the reported issues, we report many best practices to increase overall code and documentation quality.

On the tests side of things, the project currently fails when one attempts to fully execute the test suite. Hence, we could not assess the test results, nor coverage data. We recommend fixing this as quickly as possible.

Disclaimer: This audit assumes correctness of the execution engine, which the system contracts heavily rely on. For the sake of the audit, we took the execution engine as a black box, while consulting the Casper Labs team and underlying documentation whenever necessary to understand things at that layer. The audit of the execution engine itself is left for a second phase.

ID	Description	Severity	Status
QSP-1	Authentication Relies on Deploy Launcher	High	Acknowledged
QSP-2	Delegated Tokens Are Not Slashed	High	Unresolved
QSP-3	Unhandled Integer Overflow	High	Unresolved
QSP-4	Performance Degradation and Denial of Service	High	Unresolved
QSP-5	Incorrect Refund Formula	Medium	Unresolved
QSP-6	Balance Probes Do Not Enforce Read Rights	Low	Acknowledged
QSP-7	`transfer` Does Not Check Read Access of `source`	Low	Unresolved
QSP-8	Malicious Validators Could Impersonate the System Account Role	Low	Unresolved
QSP-9	Many Purses Could Be Created With Zero Amount (Exhaustion Attack)	Low	Acknowledged
QSP-10	`MAX_PAYMENT` is in Motes Rather than Gas	Low	Unresolved
QSP-11	`get_refund_purse` Does Not Grant Read Rights	Low	Unresolved
QSP-12	Unexpected Arguments in `transfer-to-account-stored`	Low	Unresolved
QSP-13	Delegation Rate Reset May Not Benefit Delegators	Low	Unresolved
QSP-14	Payment Purse Invariant May Not Hold	? Undetermined	Unresolved
QSP-15	`add_bid` Can Be Called By Non-Genesis Validators	? Undetermined	Acknowledged
QSP-16	Unclear Vesting Initialization	? Undetermined	Unresolved
QSP-17	Unbonding Does Not Lock Tokens For 24h	? Undetermined	Unresolved
QSP-18	Potential Excess of Validators	? Undetermined	Unresolved

Quantstamp Audit Breakdown

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

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

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. 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:
   i. 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.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, 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.

Findings

QSP-1 Authentication Relies on Deploy Launcher

Severity: *High Risk*

Status: Acknowledged

File(s) affected: `types/src/system/auction/mod.rs`

Description: The `Auction::add_bid`, `Auction::withdraw_bid`, `Auction::delegate`, `Auction::undelegate`, and `Auction::activate_bid` functions authenticate based on the user account/contract that originates the deploy, not on the contract or account that makes the contract call. Hence, if a user deploy interacts with an untrusted contract, the latter can call the system contracts on behalf of the user that initiated the deploy.

Recommendation: At the very least, document the current authentication limitations, providing clear semantics for `get_caller`, something that is not currently present in the online documentation. Ideally, introduce a new construct to perform the authentication based on the immediate caller of the contract and not on the deploy origin. With the latter, authenticate using the immediate caller, taking funds from its main purse instead of the deploy initiator.

QSP-2 Delegated Tokens Are Not Slashed

Severity: *High Risk*

Status: Unresolved

File(s) affected: `types/src/system/auction/mod.rs`

Description: Following the online documentation: "*Casper does not treat delegated stake differently from validator stake. If the validator is slashed, all tokens delegated to the validator will also be slashed*". However, the `slash` function does not slash delegated stake; rather, it only slashes the validator's. This opens the room for validators to delegate to themselves, reducing their slashed amount if they misbehave (intentionally or not).

Exploit Scenario: Since delegated stake is not currently slashed validators can bid a single mote and then (possibly through a different account), delegate themselves the remaining value. They can then win elections and misbehave as they like. Slashings won't affect them because the delegated stake is not slashed.

Recommendation: Slash delegated stake.

QSP-3 Unhandled Integer Overflow

Severity: High Risk

Status: Unresolved

File(s) affected: `types/src/system/*`

Description: In many parts of the audited system contracts, potential integer overflows could occur; for instance, in (this list may not be exhaustive; we suggest a thorough search):

- `types/src/system/mint/mod.rs` (L76);
- `types/src/system/mint/mod.rs` (L140);
- `types/src/system/auction/bid/vesting.rs` (L75);
- `types/src/system/auction/bid/vesting.rs` (L77);
- `types/src/system/auction/seigniorage_recipient.rs` (L38);
- `types/src/system/auction/seigniorage_recipient.rs` (L47);
- `types/src/system/auction/mod.rs` (L424).
- `types/src/system/auction/mod.rs` (L367, 422, 424, 488, 499, 500, 511);
- `types/src/system/auction/mod.rs` (L501).

Also note that in `system/handle_payment/mod.rs` there is a possible overflow on L127. Although `REFUND_PERCENTAGE` is zero now and therefore poses no threat, there's no guarantee this will not change in the future.

Recommendation: Either enable the compiler overflow check in release mode, or rely on checked arithmetic operations (e.g., `check_add`) at the code level.

QSP-4 Performance Degradation and Denial of Service

Severity: High Risk

Status: Unresolved

File(s) affected: `types/src/system/auction/*`,

Description: The performance of the following functions is negatively impacted as more users delegate stake and/or make bids. As more storage is required, serialization costs and loop iterations increase:

- `auction::bid::process`;
- `auction::Auction::get_era_validators`;
- `auction::Auction::read_seigniorage_recipients`;
- `auction::Auction::add_bid`;
- `auction::Auction::withdraw_bid`;
- `auction::Auction::delegate`;
- `auction::Auction::run_auction`;
- `auction::Auction::distribute`;
- `auction::detail::get_bids`;
- `auction::detail::set_bids`;
- `auction::detail::get_unbonding_purses`;
- `auction::detail::set_unbonding_purses`;
- `auction::detail::process_unbond_requests`;
- `auction::detail::create_unbonding_purse`;
- `auction::detail::reinvest_delegator_rewards`;
- `auction::detail::reinvest_validator_reward`.

If too many delegators join and make an increasingly high number of bids, the overall system performance degrades, potentially impacting the usability as the Casper network grows. Moreover, one should not ignore the possibility of denial-of-service attacks, either from attackers making many bids and delegations or from them exploiting the fact that many of these functions have fixed gas costs.

Recommendation: One way to handle this issue is to require a large minimum bond in order to participate in the auction. However, while this mitigates the denial of service problem, it does not solve the performance degradation case. For the latter, a major code overhaul would be required. A redesign of the auction contract storage would be needed so only fixed-size data structures are stored under each key. The main idea is to have each validator have a single purse where all bonded funds (delegated or not) are stored. Whenever someone (validator or delegator) bonds, the auction contract would emit that party a promissory note containing a share value representing the equity of the party over the stake and rewards. By burning a promissory note, a user could retrieve the funds to which he is entitled. The total amount of shares is variable to accommodate bonds and unbonds without changing the share amounts in the promissory notes. This way, simply minting into the bond purse would automatically increase everyone's value and, by burning from the purse, one would automatically punish the stake-holders. Note that promissory notes are fungible, so they can be stored as a map from the validator and delegator account hashes to the amount of shares. When implementing this, one needs to be extremely mindful and careful with the share arithmetic, which is a major source of rounding related issues in smart contracts.

QSP-5 Incorrect Refund Formula

Severity: Medium Risk

Status: Unresolved

Description: The formula in `types/src/system/handle_payment/mod.rs` (L124) is missing a division by `100`.

Exploit Scenario: Once the Casper Labs team updates `L16` with a percentage larger than `1`, users can carefully craft payment code that would allow free execution as follows (for illustration purposes, say the percentage is set to `2`):

- The attacker estimates the gas cost of the deploy;

- The attacking deploy deposits roughly twice that cost into the payment purse (usually a little more for a safety margin);
- The refund is computed as `(total - amount_spent) * 2`, which, in this case, would be roughly `total`;
- The execution, no matter how expensive, will only be charged very little.

While it doesn't look like funds can be minted or stolen, at this point we can not discard this possibility.

Recommendation: Divide the expression by `100`. Note this fix has to be performed in conjunction with QSP-3 (*Unhanded Integer Overflow*).

QSP-6 Balance Probes Do Not Enforce Read Rights

Severity: Low Risk

Status: Acknowledged

File(s) affected: `types/src/system/mint/mod.rs`

Description: Following the spec, a URef should only be read if it has been given read rights. However, in `Mint::balance`, this check is not performed; any contract could read one's balances knowing a purse's URef.

Recommendation: Add a condition statement checking whether the URef grants read permission; if not, revert.

Update: The team has clarified that for the balance read, this is an exception to the general rule, as there is a case for users checking other users' balances.

QSP-7 `transfer` Does Not Check Read Access of `source`

Severity: Low Risk

Status: Unresolved

File(s) affected: `types/src/system/mint/mod.rs`

Description: The `Mint::transfer` function does not check whether the `source` URef has read rights, nor does the call to `read_balance` (L104). This should be the case; otherwise, a `source` without read rights would still be read.

Recommendation: Add a condition statement checking whether the `source` URef has read rights; otherwise, revert.

QSP-8 Malicious Validators Could Impersonate the System Account Role

Severity: Low Risk

Status: Unresolved

Description: A malicious node could impersonate the system role by changing the underlying source code; for it to have a meaningful impact on the network, the attacker would have to spin up many nodes and stake on all of them (not necessarily economically viable). If that happens, consensus could be compromised and tokens could be minted out of thin air.

Recommendation: From a technical perspective, the issue could be largely mitigated by making the code close-sourced; one would have to rely on signed and pre-compiled statically linked executables with obfuscated code. However, such an approach defeats the decentralized philosophy of the project, not to mention the ability to have the code audited by the community at large. We DO NOT recommend anything in that direction. The current economic incentives in place are likely to discourage such an attack. Nonetheles, make sure to monitor such incentives and network behavior over time.

QSP-9 Many Purses Could Be Created With Zero Amount (Exhaustion Attack)

Severity: Low Risk

Status: Acknowledged

File(s) affected: `types/src/system/mint/mod.rs`

Description: The `Mint::mint` of zero motes creates a purse, which increases storage.

Exploit Scenario: An attacker creates a large amount of purses as a means to exhaust storage resources.

Recommendation: Make sure the cost of creating an empty purse disincentivizes malicious users in creating a large number of purses.

QSP-10 `MAX_PAYMENT` is in Motes Rather than Gas

Severity: Low Risk

Status: Unresolved

Description: If gas prices increase, the viability of more sophisticated payment options (or, in extreme cases, even the standard payment contract), may be compromised.

Exploit Scenario: Measure `MAX_PAYMENT` in units of gas instead.

QSP-11 `get_refund_purse` Does Not Grant Read Rights

Severity: Low Risk

Status: Unresolved

File(s) affected: `types/src/system/handle_payment/mod.rs`

Description: The `HandlePayment::get_refund_purse` strips all access rights prior to returning the refund purse. This seems too strict, as one will not even be able to read the purse.

Recommendation: Give read access rights to the refund purse being returned.

QSP-12 Unexpected Arguments in `transfer-to-account-stored`

Severity: Low Risk

Status: Unresolved

Description: `transfer-to-account-u512-stored`, at L48, calls a `new_contract` in its final step of the `store()` function, and uses parameters `Some(...)` for `hash_name` and `uref_name`. Yet in the nearly identical `transfer-to-account-stored` (L44), `None` is used instead.

Similarly in the former declaration of the `EntryPoint`, the `CLType::Unit` is used (L39) as a parameter, but in the latter, `CLType::URef` (L35) is used instead.

Recommendation: Clarify the matter with better documentation in the code; furthermore, we suspect that `transfer-to-account-stored` is deprecated and could be safely removed.

QSP-13 Delegation Rate Reset May Not Benefit Delegators

Severity: Low Risk

Status: Unresolved

Description: Validators can change the delegation rate whenever they change their bid; delegators who staked on the validator prior to the delegation rate change may now receive a lower cut if the delegation rate increases.

Recommendation:

- Make sure the issue is properly communicated to users; and/or
- Disable delegate rate changes within an era.

QSP-14 Payment Purse Invariant May Not Hold

Severity: Undetermined

Status: Unresolved

File(s) affected: `types/src/system/handle_payment/mod.rs`

Description: According to the comment in the code, the `finalize` function "*maintains the invariant that the balance of the payment purse is zero at the beginning and end of each deploy and that the refund purse is unset at the beginning and end of each deploy*". However, once the constant `REFUND_PERCENTAGE` is eventually set to a positive value (currently hardcoded as zero), the invariant may not always hold, as the refund or the reward purse could be be the same as the payment purse.

Recommendation: Return an error if either the refund purse or the reward purse are the same as the payment purse.

QSP-15 `add_bid` Can Be Called By Non-Genesis Validators

Severity: Undetermined

Status: Acknowledged

File(s) affected: `types/src/system/auction/mod.rs`

Description: The header comments for the `add_bid` function states the following:

```
"For a non-founder validator, this adds, or modifies, an entry in the `bids` collection and
calls `bond` in the Mint contract to create (or top off) a bid purse. It also adjusts the
delegation rate."
```

Note that nothing is stated for founder validators. Hence, it is unclear what the expected behavior should be. Our interpretation is that the function should revert with an error if called by a founding validator. As that is not the case, we cannot determine what happens as a consequence.

Recommendation: If not an issue, add better documentation to the code as a means to clarify the intended behavior. If an issue, then restrict the `add_bid` function s.t. it can only be called by non-founding validators.

Update: The team improved the header comment in the `add_bid` function to clarify that it can be called by both founder and non-founder validators.

QSP-16 Unclear Vesting Initialization

Severity: Undetermined

Status: Unresolved

File(s) affected: `types/src/system/auction/bid/mod.rs`

Description: Currently, the `process` function returns `true` if the bid vesting is initialized or if at least one delegation vesting is initialized. This is not, however, what the function's header comment suggests: "*Returns true if the provided bid's vesting schedule was initialized.*". We cannot determine if the code reflects the intended behavior, nor the consequences in the case of a mismatch.

Recommendation: Verify that the implemented code does reflect the intended behavior; if so, adjust the header comment accordingly. Otherwise, adjust the code to reflect the current comment.

QSP-17 Unbonding Does Not Lock Tokens For 24h

Severity: Undetermined

Status: Unresolved

File(s) affected: `types/src/system/auction/detail.rs`

Description: Following the specification, "*for security purposes, whenever a token is un-staked or un-delegated, the protocol will continue to keep the token locked for 1 day*". When processing unbonding requests (see `process_unbond_requests`), the logic is to proceed with the transfer iff `current_era_id >= unbonding_purse.era_of_creation() + unbonding_delay` However, `unbonding_delay` is set to 14 eras in production, as given in `resources/production/chainspec.toml`. Since an era lasts 30 minutes, that equates to 14 x 30 = 420 minutes = 7 hours. Hence, the 24h lock period is not enforced in the code.

Recommendation: Make sure the implemented code has the intended behavior; if so, change the docs accordingly. Otherwise, if the code behavior is to enforce the 24h lock period, change the chainspec `unbonding_delay` to 48 eras (48 x 30 = 1440 min = 24h).

QSP-18 Potential Excess of Validators

Severity: *Undetermined*

Status: Unresolved

File(s) affected: `types/src/system/auction/mod.rs`

Description: If the number of validator slots is less than the number of founding validators, function `auction::Auction::run_auction` will return more validators than the number of validator slots (it will return the list of founding validators). The consequences of the latter are unclear to us.

Recommendation: Clarify what would happen if the issue happens; for instance, by adding better documentation at the code level.

Code Documentation

Code documentation is almost non-existent.

There are dangling TODOs in the code lacking any description.

In the code, balances are not tied to account hashes, which is a counterintuitive design that is not properly justified. Likewise, purses are not necessarily tied to accounts, which is also counterintuive. Consider documenting your design decisions and their underlying rationale in public facing documents or in the code itself.

The system contracts are essentially part of the casper node; opposed to what the online documentation states, they do not exist as self-contained wasm module contracts. Hence, updates require a network hard-fork, as well as updated releases of the node software. This must be reflected in the current online documentation, which is out-dated.

The header comment for `Auction::get_era_validators` states the following: "*Publicly accessible, but intended for periodic use by the Handle Payment contract to update its own internal data structures recording current and past winners*". The `get_era_validators`, however, is not called by the handle payment contract. Hence, it seems the comment is outdated and does not reflect the latest implementation. Similar issue occurs with the `Auction::read_seigniorage_recipients` function.

The header comment for `Auction::add_bid` states the following: "*For a non-founder validator, this adds, or modifies, an entry in the bids collection and calls bond in the Mint contract to create (or top off) a bid purse. It also adjusts the delegation rate*". However, there is no `bond` function in the mint contract. Hence, it seems the comment is outdated and does not reflect the latest implementation. Similar issue with `delegate` (call to inexistent `bond` function) and `undelegate` (call to inexistent `unbond` function).

In `add_bid` (same with `withdraw_bid` and `delegate`), the caller is checked against the account hash parameter as a means to make sure whoever invokes the function is indeed the validator account. Note, however, that the implemented check only guarantees that the deploy origin stems from the validator. If the validator calls a contract, which in turns calls `add_bid`, the latter can place a bid on behalf of the validator. Add better documentation in `add_bid`/`withdraw_bid`/`delegate` in the code to clarify the matter. In the specification docs, clarify the semantics of `get_caller`.

In `run_auction` (`types/src/system/auction/mod.rs`), the function header documentation states that the validators are ordered from largest to smallest, but it seems that the inclusion of unsorted founders at the reserved slots means that ordering may only be true for non-founding validators.

Many comments were found to be misleading or poor, and could be improved, namely:

- `types/src/system/auction/mod.rs` (L164): it should be unvested stake;
- `types/src/system/handle_payment/constants.rs` (L25): it's the deploy launcher who pays for computation;
- `types/src/system/auction/mod.rs` (L198-199): the validator does not have to be in the founder validator set;
- `types/src/system/auction/mod.rs` (L201-202): the funds are not actually transferred to the validator's purse; in particular, the validator doesn't have access to the funds;
- `types/src/system/auction/mod.rs` (L202, 264): the function returns neither a tuple nor the purse;
- `types/src/system/handle_payment/error.rs`: `SystemFunctionCalledByUserAccount`, `InsufficientPaymentForAmountSpent` and `FailedTransferToAccountPurse` are user errors, not internal (system) errors;
- `types/src/system/mint/error.rs` (L35): as discussed in our calls, there is no local storage;
- `types/src/system/mint/error.rs` (L174): isn't that what `InvalidAccessRights` is for?;
- `smart_contracts/contracts/client/withdraw-bid/src/main.rs` (L24): unclear comment;
- `types/src/system/handle_payment/error.rs` (L80): the word “key” is missing;
- `types/src/system/mint/mod.rs` (L21), `types/src/system/auction/providers.rs` (L84): “new token” should be “new purse”;
- `types/src/system/handle_payment/constants.rs` (L5): shouldn't it be “account”?;
- `types/src/system/standard_payment/mod.rs` (L18): it should be “the payment purse”;
- `types/src/system/auction/constants.rs` (L26): shouldn't it be “`public_key`”?;
- `types/src/system/standard_payment/mod.rs` (L1): pleonasm;
- `types/src/system/auction/error.rs` (L74): double “not”;
- `types/src/system/auction/error.rs`(L88)` : “Validators” is in the plural.

Adherence to Best Practices

The function `internal::finalize_payment` (`types/src/system/handle_payment/mod.rs`) does not check if the given phase is `Phase::FinalizePayment`, but it should (e.g., as done in `set_refund`). Essentially, this performs defensive programming. Otherwise, one could potentially call the function in an incorrect context, which could cause unexpected side effects.

In `types/src/system/auction/detail.rs` (L280), a truncating operation is carried out on `delegator_reward`. This means that there might be dusty remains from this operation that could possibly accumulate over time. We suggest documenting and letting users know about it.

Different from non-synthetic methods, host contracts do not implement synthetic functions (e.g., `create()`); their implementation is given by the execution engine. It is unclear why that choice was made, as synthetic methods are essentially alias to specific calls with specific arguments (e.g., `create()` = `mint(0)`). Such implementation could reside in the host contract side, making the latter self-contained.

`SYSTEM_ACCOUNT` is defined in multiple places; instead, we suggest to place its declaration & definition in a reusable utility function or constant.

The following code appears duplicated in different parts of the system contracts (e.g., `mint/mod.rs` and `handle_payment/mod.rs`):

```
let <some var> = match self.get_key(<SOME KEY>) {
    Some(Key::URef(uref)) => uref,
    Some(_) => return Err(Error::SOME_ERROR>),
    None => return Err(Error::<SOME ERROR>),
}
```

Consider refactoring the shown code to a function as a means to eliminate duplicated code.

The error in L138 in `system/mint/mod.rs` seems incorrect:

```
136. let round_seigniorage_rate: Ratio<U512> = self
137.     .read(round_seigniorage_rate_uref)?
138.     .ok_or(Error::TotalSupplyNotFound)?; <====
```

The error should report something related to the round seigniorage rate, not total supply. Furthermore, would it not be the case that a panic is better suited here, as it stems developers' mistake?

`Error::BondTooSmall` is triggered only when the bond value is zero; it is never trigger by a small positive value. Hence, renaming the error to `BondEqualsZero` conveys better meaning.

In `Auction::add_bid`, the calculation of the account hash in L115 is unnecessary, as it was already calculated in L100. As both rely on `blake2b`, the removal is safe.

Statements such as `AccountHash::from_public_key(&public_key, |x| self.blake2b(x))` can be simplified to `AccountHash::from(&public_key)` We recommend the simplification across all files where `from_public_key` appears.

In `types/src/system/auction/bid/mod.rs`, `Bid::activate` and `Bid::deactivate` have a single and fixed return Boolean value (`false` and `true`, respectively). While unconditionally returning a single value makes sense in the case of `self` (e.g., to chain calls), it makes little sense in the case of Booleans. Hence, consider removing the return value.

In `run_auction` (`system/auction/mod.rs`), consider changing the following snippet

```
 if evicted_validators.contains(validator_public_key) {
    bids_modified = bid.deactivate();
}
```

to

```
if evicted_validators.contains(validator_public_key) {
    bid.deactivate();
    bids_modified = true;
}
```

The latter explicitly states that `bids_modified` will only receive the `true` value within the loop containing the shown code; otherwise, one is forced to look into `bid_deactivate()` to reach the same conclusion.

In `types/src/system/auction/mod.rs`, `Auction::run_action`, add a comment that the sorting of non-founder weights is in descending order, as that could easily be missed when looking at the comparison function call (`rhs.cmp(lhs)`).

Typo in word "delgation"; found in `smart_contracts/contracts/client/add-bid/src/main.rs`, `smart_contracts/contracts/client/delegate/src/main.rs`.

In `types/src/system/auction/mod.rs`, L427-445 contains a block statement whose content could have been put directly in the block of the function. Seems unnecessary.

Typos in `types/src/system/auction/bid/mod.rs`: "schemars" -> "schemas", "vesting_sechdule" -> "vesting_schedule".

Typo in `types/src/system/auction/seigniorage_recipient.rs` (L41): "Caculates" -> "Calculates".

Typo in `system/auction/unbonding_purse.rs` (L62): should be "when" not "and".

Typo in `types/src/system/mint/mod.rs` (L41): "returns a the new amount" -> "returns the new amount".

`types/src/system/mint/mod.rs` (L138): bad error enumerate. Use a meaningful name instead.

`types/src/system/mint/mod.rs` (L43, 68): `Error::MissingKey` is not an appropriate error type for these situations.

In `types/src/system/auction/detail.rs` (L197-204) illustrate confusing nomenclatures used throughout the auction codebase: the `UnbondingPurse` structure is not actually a purse and accounts are not unbonding purses. Adjust nomenclature accordingly.

There is an overall confusion regarding the usage of the term “key”. The constants in `system::mint::constants` such as `TOTAL_SUPPLY_KEY` are keys in the contract context were we to regard it as a key-value mapping. However, the type `Key` actually represents values in this mapping. Either use different terms, or clarify the matter with better code comments.

As there are many typos in the code, consider running a spellchecker against the code base.

`types/src/system/auction/error.rs` (L204, 205): duplicate lines. Remove one.

Bad variable names: `types/src/system/auction/bid/vesting.rs` (L47); `types/src/system/auction/delegator.rs` (L91); `types/src/system/auction/mod.rs` (L68); `types/src/system/auction/detail.rs` (L48). Consider using more descriptive names.

Rather than duplicating argument names in client contracts (`smart_contracts/contracts/client/`), we recommend importing the corresponding constants from the `types` crate.

`types/src/system/auction/mod.rs` (L286-295): arguably, the check in L295 should be performed before the bonding purse is created.

The `Copy` trait is meant for small, register-fitting, types and may be unsuitable for the following types: `auction::bid::VestingSchedule`; `auction::Delegator`; `auction::UnbondingPurse`.

`types/src/system/handle_payment/error.rs` (L5): looks like an unnecessary `use` statement; just use `Result` in L136.

The expensive conversion `Ratio::from` in `types/src/system/mint/mod.rs` (L140) is not necessary.

Giving the following types names would make the code more readable: `[U512; LOCKED_AMOUNTS_LENGTH]`.

`smart_contracts/contracts/client/counter-define/src/main.rs` (L15, 16): unnecessary `::{self}`.

Clarifications (to be considered by the team)

- `types/src/system/handle_payment/error.rs`: since internal errors are supposed to never happen, wouldn't it make more sense for systems errors to cause a panic?
- `types/src/system/auction/mod.rs` (L435), `types/src/system/auction/mod.rs` (L482), `types/src/system/auction/detail.rs` (L49, 53): since this is never supposed to happen, wouldn't a panic be a better behavior?
- `types/src/system/mint/mod.rs`, function `balance`: `Ok(None)` is never returned, so why the `Option<U512>` in the return type?
- `types/src/system/auction/bid/vesting.rs` (L102-121): is there a good reason for this code to be written using unsafe Rust? The performance penalty seems very minor (if present at all) and the same functionality admits a much simpler implementation in safe Rust.
- `types/src/system/auction/detail.rs` (L96): `EraId` seems preferable to `u64`.
- `types/src/system/auction/era_info.rs` (L196): why not `|&allocation|` rather than `move |allocation|`?
- `types/src/system/handle_payment/mod.rs` (L53-59): why not `use super::*` instead?

Test Results

Test Suite Results

Building the project currently fails; hence, we cannot assess the test suite execution result.

Code Coverage

Building the project currently fails; hence, we cannot assess test coverage.

Changelog

- 2021-04-01 - Initial report

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.