Security Audit Report

Anoma 2024 Q3

Authors: Andrija Mitrovic, Ivan Golubovic, Manuel Bravo

Last revised 16 August, 2024

Table of Contents

Audit Overview

Scope

In July 2024, Informal Systems has conducted a security audit for IBC integration into Namada. The audit aimed at inspecting the correctness and security properties of the solution. The components under scope are:

- IBC integration (integration of ibc-rs into Namada's validity predicate execution model)
    - /crates/ibc
    - crates/namada/src/ledger/native_vp/ibc/mod.rs
    - crates/namada/src/ledger/native_vp/ibc/context.rs
- MASP integration (integration of the MASP with IBC)
    - crates/namada/src/ledger/native_vp/masp.rs
    - crates/shielded_token

The audit was performed from July 16th, 2024 to August 9th, 2024 by the following personnel:

- Andrija Mitrovic
- Ivan Golubovic
- Manuel Bravo

Relevant Code Commits

The audited code was from:

- commit hash 012cd88b3c441ebe4a9d938a44b7fe33fc0e7ee9.

Conclusion

After conducting a thorough review of the project, we found it to be carefully designed and generally well implemented. The audit was divided into two primary areas of focus: the integration of IBC-rs and the MASP (Multi-Asset Shielded Pool) Validity Predicate (VP). Given that the MASP VP had been previously audited, our attention was primarily on the new changes related to IBC integration.

In the IBC integration portion of the audit, our goal was to assess whether everything necessary for the integration with IBC-rs was in place, to evaluate how the integration was carried out, and to identify any potential vulnerabilities. The absence of significant findings in this area indicates that the integration was well-executed, with no critical vulnerabilities identified. The implementation was robust, clear, and easy to understand.

The MASP component, however, presented more challenges due to its complexity. We are especially grateful to the Namada implementation team, who provided invaluable assistance in helping us navigate and fully comprehend the the MASP design. The findings in this area stemmed not from poor implementation but from the inherently complex design, which involves numerous moving parts, assumptions, and dependencies on external code beyond the MASP itself. Despite these challenges, the collaboration with the Namada team allowed us to leverage our expertise to identify areas for potential improvement.

Overall, the project is solid, with a well-thought-out design and a high-quality implementation. The cooperation and expertise of the Namada team greatly contributed to the success of the audit, and we believe the project is on a strong foundation moving forward.

Audit Dashboard

Target Summary

- **Type**: Protocol and Implementation
- **Platform**: Rust
- **Artifacts**: IBC and MASP IBC integration

Engagement Summary

- **Dates**: 16.07.2024. - 9.08.2024.
- **Method**: Manual code review, protocol analysis

Severity Summary

Finding Severity	**#**
Critical	2
High	0
Medium	0
Low	1
Informational	4
Total	**7**

Threat Analysis

In our threat analysis, we start by defining a set of properties required for the correctness of the IBC integration in Namada. We separate them into safety and liveness. For each property, we define one or more threats. We then analyzed them individually to see if they could be violated, resulting in the findings presented in the Findings section.

Additionally, we have inspected a set of non-functional threats related to integer-type conversions, integer overflow, and DoS attacks. We have listed them at the bottom of this section.

Safety Properties

1. Namada correctly implements all ibc-rs core, client and client upgradability context APIs

Threat 1.1: Namada does not implement one or more context methods.

Threat 1.2: Namada's implementation of one or more context methods does not attain the IBC specification.

Threat 1.3: The data passed to one or more context methods is not serialized/deserialized correctly upon storing/retrieving.

Threat 1.4: The implementation of one or more context methods does not handle "storage not responding" scenarios correctly.

Threat 1.5: The implementation of one or more context methods may cause panic, which would halt the node.

Conclusion: The threats do not hold. Implementation of the following ibc-rs core, client and client upgradability context APIs is checked against:

- **Core context APIs**: These connect all the implemented IBC core modules to the host blockchain:
  - ValidationContext
  - ExecutionContext
- **Client context APIs**: These integrate supported light clients to the host:
  - ClientValidationContext
  - ClientExecutionContext
- **Client Upgradability Context APIs**: Enables light clients to undergo an upgrade process:
  - UpgradeValidationContext
  - UpgradeExecutionContext

The first two are completely implemented, with a few functions from those traits that use default implementations. The third, Client Upgradability Context APIs are not implemented at all but this is not a problem because Namada stores the upgraded client state and consensus state by a Namada governance transaction. Chain upgrade is handled with `MsgUpgradeClient` . It's implemented in https://github.com/cosmos/ibc-rs/blob/86a0ee0d7c245592307b9bd7659bab05526cc136/ibc-core/ics02-client/src/handler/upgrade_client.rs. All APIs are implemented correctly that no of the mentioned threats are possible.

2. Namada's correctly implements all ibc-rs token transfer (ICS 20) and NFT token transfer (ICS 721) APIs

Threat 2.1: Namada does not implement one or more core context methods.

Threat 2.2: Namada's implementation of one or more context methods is not implemented correctly, e.g., the token escrow method does not transfer tokens from the appropriate accounts to the escrow account.

Threat 2.3: The data passed to one or more context methods is not serialized/deserialized correctly upon storing/retrieving.
Threat 2.4: The implementation of one or more context methods does not handle "storage not responding"

scenarios correctly.
Threat 2.5: The implementation of one or more context methods may cause panic, which would halt the node.

Conclusion: The threats do not hold. Implementation of the following ibc-rs token transfer (ICS 20) and NFT token transfer (ICS 721) APIs is checked against:

- **Token transfer APIs (ICS 20)**: Implementation of escrow/unescrow and mint/burn APIs on the host chain:
  - TokenTransferValidationContext
  - TokenTransferExecutionContext
- **NFT token transfer APIs (ICS 721)**: These integrate supported light clients to the host:
  - NftTransferValidationContext
  - NftTransferExecutionContext

All APIs are implemented correctly that no of the mentioned threats are possible.

3. Namada's IBC validity predicate only accepts a transaction if it includes a valid IBC operation

Threat 3.1: The IBC VP accepts a transaction that does not include an IBC operation and modifies IBC keys.

Threat 3.2: The IBC VP accepts a transaction with an IBC operation without validating via its corresponding ibc-rs validation method.

Threat 3.3: The IBC VP accepts a transaction with an IBC operation whose state changes differ from the state changes that the associated ibc-rs execution method would generate.

Threat 3.4: The IBC VP accepts a transaction with an IBC operation that generates a set of events different from the events that the associated ibc-rs execution method would generate.

Conclusion: The threats do not hold. IBC VP accept only transactions that have an internal address and only checks the validity of state modification under IBC keys, accepting only IBC operation whose state changes are the same as the associated ibc-rs execution method would generate. IBC VP accepts only transactions with an IBC operation that generates a set of events same as the events generated through ibc-rs execution method.

4. Namada's IBC validity predicate ensures that the IBC traces are stored correctly

Threat 4.1: The checks in the validity predicate do not guarantee that if a transaction is accepted and mints some tokens, then the state changes would mimic those done by the maybe_store_ibc_denom function.

Threat 4.2: The IBC VP accepts transactions that include an IBC operation that changes an IBC trace key and that does not store any value with the key.

Threat 4.3: The IBC VP accepts transactions that include an IBC operation that changes an IBC trace key and is associated value does not match the trace hash.

Conclusion: The threats do not hold. The validation of IBC trace follows the trace hash calculation done in maybe_store_ibc_denom function on storing trace hash. It rejects transactions that include an IBC operation that changes an IBC trace key and that does not store any value with the key and the transactions that stored an invalid trace.

5. Namada's IBC validity predicate ensures that IBC operations register rate limiter deposit and withdraws correctly

Threat 5.1: The IBC VP accepts transactions that include an IBC operation that increases a token's per-epoch deposit and does not register it properly, i.e., according to the add_deposit function.

Threat 5.2: The IBC VP accepts transactions that include an IBC operation that increases a token's per-epoch withdraw and does not register it properly, i.e., according to the add_withdraw function.

6. Namada’s IBC rate limiter prevents deposits into Namada that increase the supply to more than the allowed mint limit

Threat 6.1: The IBC VP accepts transactions that include an IBC operation that mints tokens and does not register it properly, i.e., according to the update_mint_amount function.

Threat 6.2: The IBC VP accepts transactions that include an IBC operation that increases the supply over the allowed mint limit.

Conclusion: The threats do not hold. On transaction execute update_mint_amount function updates the minted value that is later used in limit checking, no possibility of passing a transaction that does not register properly and thus passing the mint limit check. Token minting due deposits is not possible over mint limit.

7. Namada’s IBC rate limiter correctly implements the per-epoch net throughput limit

Threat 7.1: The `check_limits` function does not compute the current net throughput correctly.

Threat 7.2: The net throughput is not reset when a new epoch starts.

Threat 7.3: The IBC VP does not reject transactions once the net throughput limit is reached or that would make the net throughput go over the limit.

Conclusion: The threats do not hold. Throughput is calculated as a absolute difference between the deposit and withdraw value. Due to an unpredictable deposit and withdrawal occurrences it could happen that the throughput becomes larger then the throughput limit at one moment thus preventing any transfers to happen thus also preventing changes that will decrease difference between the deposit and withdraw values. In order to overcome this, on the end of each epoch the net throughput is reset.

8. Balances changes due to IBC messages are recorded correctly.

Threat 8.1: Due to a miscalculation of the pre and post MASP balance due to an IBC operation, the MASP validity predicate rejects valid transactions.

Threat 8.2: Malicious users can exploit a miscalculation of the pre and post MASP balance due to an IBC operation to get invalid transactions to pass the MASP validity predicate: burn, mint, or divert assets.

Conclusion: The inspection identified the critical areas in the code where accounts balances change due to IBC messages are handled. The primary function responsible for this is `is_valid_masp_transfer`, which uses the `changed_balances` structure to determine state changes resulting from a transaction and to validate these changes accordingly.

Integration with IBC introduced significant changes, notably within the `validate_state_and_get_transfer_data` function. Initially, the keys of the changed account balances are obtained and recorded in `changed_balances` via the `apply_balance_change` function. Here, the pre-state and post-state balances are adjusted based on actual storage values.

The next phase involves updating balances based on the transaction itself. This is handled in `apply_ibc_packet`, which directs balance changes to specific functions depending on the transfer type:

- `IBCMessage::Transfer` or `IBCMessage::NFTTransfer` -> `apply_transfer_msg`
- `IBCMessage::Envelope` -> `apply_recv_message`

For `apply_transfer_msg`, the post-state of the IBC internal address and the transfer receiving address are updated.

In terms of balance changes within `changed_balances` we observed the changes in the two most important accounts:

- IBC Internal Address:
  - `apply_balance_change` : Updates pre-state and post-state balances.
  - `apply_transfer_msg` : Adjusts post-state by the transaction amount.
  - `apply_recv_msg` : Updates pre-state for received packets.
- MASP Address:
  - `apply_balance_change` : Updates pre-state and post-state balances.
  - `apply_transfer_msg` : Adjusts post-state by the transaction amount if it involves MASP.

The inspection included checking the following transactions and their implications regarding MASP and IBC interactions:

1. Transaction with Envelope IBC Message (Recv, Ack, Timeout) with Shielding Transaction in the memo field.
2. Transaction with `MsgTransfer` or `NFTTransfer` IBC Message Interacting with MASP with Shielding Transaction via reference in `::transfer::shielded_section_hash`.
3. Transaction with `MsgTransfer` or `NFTTransfer` IBC message without interacting with MASP but with an unrelated shielding transaction with shielding transaction via reference in the data section, unrelated to `::transfer::shielded_section_hash`.

Key observations:

- The `validate_state_and_get_transfer_data` function incorporates changes introduced for IBC integration. It retrieves balance changes and applies them to the `changed_balances` structure.
- Balance changes due to IBC transfers (using `IBCMessage::Transfer` or `IBCMessage::NFTTransfer`) are handled in `apply_transfer_msg` . This function updates the post-state of the relevant accounts, including the IBC internal address and the transfer receiving address. Similar balance changes occur for receiving transfers via `apply_recv_msg` in case of receiving `IBCMessage::Envelope` .
- For sending source transfers, the IBC internal address post-state is initially increased and then decreased by the transaction amount to prevent double-counting.
- When MASP transaction is incorporated within an IBC transfer (sending or receiving), the balances of the MASP address, the IBC internal address, and the remote receiver address are updated accordingly.
- The inspection considered various shielding transaction scenarios (refunding, receiving, unshielding, shielding, shielded) within IBC messages.

The inspection of these threats resulted in the following findings:

Incorrect Balance Updates in IBC Transfer Handling
Potential Issues with Handling Multiple MASP Actions
Redundant field in shielding data structure

9. If the MASP validity predicated accepts a transaction that modifies balances, then users of the accounts whose balances decrease have authorized the transfer.

Threat 9.1: Malicious users can replay transfers without the source account owner's authorization because the MASP VP does not check all required signatures.

Threat 9.2: Malicious users can replay transfers without the source account owner's authorization because the MASP VP does not verify signatures correctly.

Conclusion: The inspection focused on ensuring that transactions are authorized by the correct signatories and that replay protection mechanisms are correctly implemented to prevent transactions from being applied multiple times. This included verifying how signers for the transaction are collected and how their signatures are validated.

Key observations:

- The system identifies accounts requiring signatures based on balance changes and transparent inputs.
- The code iterates over required signers, decodes their addresses, and verifies signatures against provided public keys.
- A specific check is in place to prevent simultaneous credit and debit of the IBC account within a single transaction.
- The signature verification mechanism plays a crucial role in preventing transaction replay. By requiring signatures from all accounts experiencing a balance decrease, the system ensures that a transaction cannot be embedded within another without the authorization of all affected parties. This protects against front-running attacks and unauthorized asset movement.

The requirement for signatures from accounts with decreased balances directly addresses the risk of replay attacks without authorization.

The signature verification process, including public key retrieval, threshold checks, and signature validation, helps mitigate the risk of incorrect signature verification.

The threats do not hold, apart from the fact that previously mentioned finding is in relation to signatures.

Liveness Properties

10. Assume a transaction with a valid IBC operation is included in a committed block whose execution won't exceed the limits. The IBC validity predicate accepts the transaction.

Threat 10.1: The IBC VP rejects a transaction with a valid IBC operation, i.e., one generated via the client library in `wasm/tx_ibc/src/lib.rs`.

Threat 10.2: The IBC rate limiter rejects a transaction that increases the supply of a token but its execution would not exceed the mint limit.

Threat 10.3: The IBC rate limiter rejects a transaction that changes the per-epoch net throughput but does not make it exceed the limit.

Threat 10.4: Malicious users can halt the chain (crash nodes) by submitting an invalid IBC operation that crashes the IBC VP.

Conclusion: The threats do not hold. IBC VP deterministically checks the state changes on transaction execution thus valid operations will not be rejected. Rate limiter can not reject transactions that do not exceed the mint and throughput limits. No possibility of crashing the IBC VP has been found.

11. If a valid MASP transaction with an IBC message that has not been executed before is validated by the validity predicate, then it passes validation.

Threat 11.1: Honest users cannot execute valid MASP transactions with an IBC message because the VP requires a signature from a user whose account balance did not decrease due to the execution of the transaction.

Threat 11.2: Malicious users can halt the chain (crash nodes) by submitting an invalid IBC operation that crashes the MASP VP.

Conclusion: The inspection determined that Threat 11.1 does not hold. There is no way to trick the MASP validity predicate into requiring an unnecessary signature. The only conditions under which a signature is required are when the address provides a transparent input or when the account's balance in the post-state is lower than in the pre-state due to the transaction application. Therefore, the validity predicate correctly identifies and includes only the necessary signatures in the set of signers, ensuring that valid MASP transactions with IBC messages are not improperly rejected.

Regarding Threat 11.2, the inspection confirmed that the MASP validity predicate is robust against attempts to crash the system via invalid IBC operations. All necessary checks, including overflow and underflow protections, are properly implemented. The code maintains determinism and correctly handles data structures and MASP/IBC interactions, preventing any potential crashes. Consequently, this threat also does not hold.

Findings

Title	Type	Severity	Status
Incorrect Balance Updates in IBC Transfer Handling	IMPLEMENTATION	4 CRITICAL	ACKNOWLEDGED
Potential Token Loss Due to Refund Transaction Failures	IMPLEMENTATION	4 CRITICAL	ACKNOWLEDGED
Potential Issues with Handling Multiple MASP Actions	IMPLEMENTATION	1 LOW	ACKNOWLEDGED
Redundant Field in Shielding Data Structure	IMPLEMENTATION	0 INFORMATIONAL	ACKNOWLEDGED
Non Automatic Updating Mint Limit Could Lead to Rejecting Valid Transfers	IMPLEMENTATION	0 INFORMATIONAL	ACKNOWLEDGED
Duplicated Checks in MASP VP and IBC VP	IMPLEMENTATION	0 INFORMATIONAL	ACKNOWLEDGED
Minor Code Improvements	IMPLEMENTATION	0 INFORMATIONAL	ACKNOWLEDGED

Incorrect Balance Updates in IBC Transfer Handling

Project	**Namada 2024 Q3**
Type	IMPLEMENTATION
Severity	4 CRITICAL
Impact	3 HIGH
Exploitability	3 HIGH
Status	ACKNOWLEDGED
Issue	https://github.com/anoma/namada/pull/3611

Involved artifacts

- namada/crates/namada/src/ledger/native_vp/masp.rs

Description

The MASP VP computes how MASP transactions change balances in `changed_balances` and based on those changes performs the validation. When a transaction includes a `Recv` IBC message, the `changed_balances` variable is updated by adding the transferred amount to the IBC internal address balance before the execution of the transaction; see the following code:

```
// Apply the given write acknowledge to the changed balances structure
   fn apply_recv_msg(
       &self,
       mut acc: ChangedBalances,
       msg: &IbcMsgRecvPacket,
       ibc_traces: Vec<String>,
       amount: Amount,
       keys_changed: &BTreeSet<Key>,
   ) -> Result<ChangedBalances> {
       self.check_packet_receiving(msg, keys_changed)?;

       // If the transfer was a failure, then enable funds to
       // be withdrawn from the IBC internal address
       if self.is_receiving_success(
           &msg.packet.port_id_on_b,
           &msg.packet.chan_id_on_b,
           msg.packet.seq_on_a,
       )? {
           for ibc_trace in ibc_traces {
               // Get the received token
```

```
                let token = namada_ibc::received_ibc_token(
                    ibc_trace,
                    &msg.packet.port_id_on_a,
                    &msg.packet.chan_id_on_a,
                    &msg.packet.port_id_on_b,
                    &msg.packet.chan_id_on_b,
                )
                .into_storage_result()
                .map_err(Error::NativeVpError)?;
                let delta = ValueSum::from_pair(token.clone(), amount);
                // Enable funds to be taken from the IBC internal
                // address and be deposited elsewhere
                // Required for the IBC internal Address to release
                // funds
                let ibc_taddr = addr_taddr(IBC);
                let pre_entry = acc
                    .pre
                    .get(&ibc_taddr)
                    .cloned()
                    .unwrap_or(ValueSum::zero());
                acc.pre.insert(
                    ibc_taddr,
                    checked!(pre_entry + &delta)
                        .map_err(native_vp::Error::new)?,
                );
            }
        }
        Ok(acc)
    }
```

The execution of a valid `Recv` IBC message may lead to minting or unescrowing tokens depending on whether Namada is the native chain of the denom being transferred. In the case of minting, the MASP VP handling of the `Recv` IBC message is correct: it “artificially” adds the transferred amount to the IBC internal address balance before the execution of the transaction to account for the fact that the tokens are minted. Nevertheless, in the case when the tokens are unescrowed, the tokens are indeed transferred from the IBC internal address to the destination Namada address: the tokens are already in the IBC internal address balance before the execution of the transaction. Hence, adding them “artificially” is not needed and could create an unbalance.

Problem Scenarios

The above problem may lead to incorrect balance calculations: the pre-state balance of the IBC internal address is inflated, leading to incorrect accounting. Intuitively, this could lead to some valid transactions being rejected by the MASP VP (liveness violation) or allow malicious users to commit invalid MASP transactions (safety violation).

The incorrect balance calculation might affect the determination of who should sign the transaction, as this is based on balance changes where every account that has the post state lower than pre state has to sign the transaction, thus some of the accounts would be missing from the signers set as their pre balance could be greater than it really is.

Recommendation

The MASP VP `apply_recv_msg` function should only add the transferred tokens to the IBC internal account balance before the execution of the transaction when Namada is not the native chain of the transferred denom.

Potential Token Loss Due to Refund Transaction Failures

Project	**Namada 2024 Q3**
Type	IMPLEMENTATION
Severity	4 CRITICAL
Impact	3 HIGH
Exploitability	3 HIGH
Status	ACKNOWLEDGED
Issue	https://github.com/anoma/namada/issues/3620

Involved artifacts

- namada/crates/namada/src/ledger/native_vp/masp.rs

Description

Users can transfer tokens from Namada's MASP to a remote chain via IBC. If the transfer fails, e.g., timeout, the tokens should be refunded back to MASP. To do so, users must include a shielding transaction in the packet's memo field that is executed in case of a refund.

Unfortunately, there is no guarantee that the user-constructed refunding shielding transaction will be accepted by the MASP VP in case a transfer fails even when the refunding transaction is well-constructed. This is a problem as the transferred tokens will remain permanently blocked: the user cannot regenerate the refunding shielding transaction to circumvent the problem.

Note: It's important to note that the Namada team has identified this issue in parallel to our inspection and has plans to resolve it by disallowing refunds to the MASP. The intended solution is to restrict refunds to transparent addresses, effectively eliminating this problem.

Problem Scenarios

Consider a scenario where a token transfer from Namada's MASP to a remote chain fails, triggering the execution of a refund transaction. The refund transaction is expected to return the blocked funds by unescrowing or minting them back to the MASP. However, the validity predicate might reject this refund transaction for several reasons:

- If the assets being shielded, i.e., refunded, are not epoched correctly. This may happen because the refunding shielding transaction is constructed in the epoch at which the user creates the transfer and the epoch at which the refunding transaction is executed may be greater. This is aggravated by the fact that the whole operation involves cross-chain communication, which increases the likelihood of this happening. This is the most concerning scenario because the refunding transaction is well-constructed but still gets rejected due to something outside of the users control.

- If the refunding shielding transaction is simply not well-formed. The tokens would still be blocked permanently, but one could argue that it is the user’s responsibility to guarantee the well-formedness of the refunding transaction.

A second negative consequence of the issue is that relayers may get disincentive to submit timeout or failure ack packets given that these may get rejected for reasons outside of their control.

Recommendation

The Namada team plans to disallow refunds to MASP addresses entirely and restrict refunds to transparent addresses instead, which eliminates the issue.

Potential Issues with Handling Multiple MASP Actions

Project	**Namada 2024 Q3**
Type	IMPLEMENTATION
Severity	1 LOW
Impact	1 LOW
Exploitability	1 LOW
Status	ACKNOWLEDGED
Issue	

Involved artifacts

- namada/crates/namada/src/ledger/native_vp/masp.rs

Description

The `get_masp_section_ref` function within the codebase uncovers a potential vulnerability due to its handling of multiple MASP actions within a transaction:

```
/// Helper function to get the optional masp section reference from the
/// [`Actions`]. If more than one [`MaspAction`] has been found we return the
/// first one
pub fn get_masp_section_ref<T: Read>(
    reader: &T,
) -> Result<Option<TxId>, <T as Read>::Err> {
    Ok(reader.read_actions()?.into_iter().find_map(|action| {
        // In case of multiple masp actions we get the first one
        if let Action::Masp(MaspAction { masp_section_ref }) = action {
            Some(masp_section_ref)
        } else {
            None
        }
    }))
}
```

The current design decision allows multiple MASP actions within a transaction. The validity predicate is ignoring all MASP actions but the first one. After discussing with the implementation team, it is concluded that it would be more useful to restrict the number of MASP actions to one.

Problem Scenarios

The function currently returns the first encountered MASP action, effectively ignoring subsequent ones. While the existing validation process is likely to reject transactions with multiple MASP actions from being processed, there's a risk of unexpected behavior or errors if this assumption is invalidated.

Recommendation

Enforce a strict limit of one MASP action per transaction by modifying the `get_masp_section_ref` function to return an error if multiple MASP actions are found. This will prevent potential issues and increase code robustness.

Redundant Field in Shielding Data Structure

Project	**Namada 2024 Q3**
Type	**IMPLEMENTATION**
Severity	**0 INFORMATIONAL**
Impact	**0 NONE**
Exploitability	**0 NONE**
Status	**ACKNOWLEDGED**
Issue	

Involved artifacts

- namada/crates/ibc/src/msg.rs

Description

The `IBCShieldingData` struct contains two fields: `refund` and `shielding`. The current code logic dictates that only one of these fields will be populated for a given transaction, indicating potential redundancy in the data structure.

Problem Scenarios

The presence of both `refund` and `shielding` fields within the `shielding_data` struct can lead to increased code complexity and potential for errors. Additionally, it might consume unnecessary storage space if both fields are consistently populated with default values.

Recommendation

To optimize the data structure and improve code clarity, consider refactoring the `IBCShieldingData` struct to contain a single field representing either a refund or a shielding operation. This could be achieved by using an enum or option type to encapsulate the different transaction types.

By streamlining the data structure, the code becomes more readable, maintainable, and efficient. It also reduces the potential for errors arising from incorrect field usage.

Non Automatic Updating Mint Limit Could Lead to Rejecting Valid Transfers

Project	**Namada 2024 Q3**
Type	PROTOCOL
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Status	ACKNOWLEDGED
Issue	

Important note

After the discussion with Anoma team, the finding described below was marked as informational and according to their request, it is going to stay in the report as it may be beneficial for some readers.

The response from the team explained the decision which lead to the current design of mint limits:

> *There are a few reasons we want this kind of mint limit:*
> - Bound the assets secured by Namada PoS (and thereby provide credible guarantees of PoS security)
> - Bound the assets secured by the protocol (and thereby provide a more feasible recovery early on if there are bugs + the ability to increase limits gradually)
> - Bound the assets transferred from another chain, particularly in cases where there could be activity which the Namada validators want to limit their exposure to (e.g. Lazarus Group activity on Ethereum)
>
> - Throughput limits help here too, but we want both options, as throughput limits themselves don't establish an absolute bound

Involved artifacts

- /native_vp/ibc/mod.rs

Description

Mint and throughput limits are introduced to prevent malicious users from making transfers that drain large amounts of assets quickly. These interrupt transfers to prevent large malicious transfers thus giving time to the community to act appropriately.

The `mint_limit` is intended to bound the total amount of assets from other chains on Namada. If the mint limit is reached for a given asset, incoming transfers are rejected until the minted amount drops due to outcoming transfers, i.e., burning.

The limit of the difference between deposits and withdrawals during one epoch is called throughput limit. If this limit is reached the transfers are rejected until the difference between deposits and withdrawals becomes smaller than the limit or a new epoch has been reached thus resetting the deposit and withdrawal values to zero, i.e., resetting the throughput.

It can be seen that no reset/update for minted tokens limit check is done. This could lead that valid transfers are rejected if the mint limit is reached.

Problem Scenarios

Assume that there are more transfers to Namada that lead to more tokens being minted than burned. As the chain popularity and usage increases this is to be expected. This will lead to reaching the `mint_limit` this preventing valid transfers. These transfers will be blocked until the `mint_limit` is updated via governance.

Recommendation

An automatic update of mint limit should be introduced on each epoch that will follow the trends of minting and burning tokens during previous epochs.

Duplicated Checks in MASP VP and IBC VP

Project	**Namada 2024 Q3**
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Status	ACKNOWLEDGED
Issue	

Involved artifacts

- namada/crates/namada/src/ledger/native_vp/masp.rs

Description

The MASP VP performs several checks related to IBC message processing, including:

- `check_ibc_transfer`
- `check_packet_receiving`
- `is_receiving_success`

These checks appear to duplicate functionality that might be expected from the IBC VP itself.

The provided explanation from the Anoma team suggests these redundant checks exist for two reasons:

1. To avoid assumptions about how the IBC VP interprets IBC data within the MASP VP's `Data` section.
2. To avoid creating a circular dependency between `namada_ibc` and `namada_vp_env` crates.

However, the first one raised concerns about what and how validation procedures are duplicated between the two. Intuitively, if there is a need to duplicate validation, then it should be done completely, not just some parts of it. Otherwise, there should not be redundancies, and the IBC VP should be responsible for checking everything in relation to IBC.

Problem Scenarios

The redundancy raises concerns regarding the assumption that the IBC VP might not function correctly for certain transactions, leading to duplicated checks in the MASP VP. This assumption is problematic as both VPs are expected to run as intended without needing redundant validations. If validation needs to be duplicated, the consistency of such duplication becomes questionable - partial duplication of IBC checks, such as transfer validations, could lead to inconsistencies or missed security checks.

Recommendation

Consolidate IBC-related checks within the IBC VP, ensuring that the MASP VP does not include redundant validations unless necessary.

Minor Code Improvements

Project	**Namada 2024 Q3**
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Status	ACKNOWLEDGED
Issue	https://github.com/anoma/namada/pull/3611

Description

During the code inspection, a several places for small improvements in the code were identified:

- A small optimization is possible in this part of code:
  Refactor the code to perform an early exit by evaluating the `non_allowed_changes` condition before creating `masp_token_map_changed` and `masp_transfer_changes`. This will streamline the logic and potentially improve performance by avoiding unnecessary computations when non-allowed changes are detected. The refactored code should look like this:

```
let masp_keys_changed: Vec<&Key> =
    keys_changed.iter().filter(|key| is_masp_key(key)).collect();

let non_allowed_changes = masp_keys_changed.iter().any(|key| {
    !is_masp_transfer_key(key) && !is_masp_token_map_key(key)
});

if non_allowed_changes {
    return Err(...); // handle the error or early exit here
}

let masp_token_map_changed = masp_keys_changed
    .iter()
    .any(|key| is_masp_token_map_key(key));

let masp_transfer_changes = masp_keys_changed
    .iter()
    .any(|key| is_masp_transfer_key(key));
```

- The comment should say `Underflow` instead of `overflow`.
- The comment should say `Underflow` instead of `overflow`.

- The comment should say `Underflow` instead of `overflow`.
- The comment should say `Underflow` instead of `overflow`.

Appendix: Vulnerability Classification

For classifying vulnerabilities identified in the findings of this report, we employ the simplified version of Common Vulnerability Scoring System (CVSS) v3.1, which is an industry standard vulnerability metric. For each identified vulnerability we assess the scores from the *Base Metric Group*, the Impact score, and the Exploitability score. The *Exploitability score* reflects the ease and technical means by which the vulnerability can be exploited. That is, it represents characteristics of the *thing that is vulnerable*, which we refer to formally as the *vulnerable component*. The *Impact score* reflects the direct consequence of a successful exploit, and represents the consequence to the *thing that suffers the impact*, which we refer to formally as the *impacted component*. In order to ease score understanding, we employ CVSS Qualitative Severity Rating Scale, and abstract numerical scores into the textual representation; we construct the final *Severity score* based on the combination of the Impact and Exploitability sub-scores.

As blockchains are a fast evolving field, we evaluate the scores not only for the present state of the system, but also for the state that deems achievable within 1 year of projected system evolution. E.g., if at present the system interacts with 1-2 other blockchains, but plans to expand interaction to 10-20 within the next year, we evaluate the impact, exploitability, and severity scores wrt. the latter state, in order to give the system designers better understanding of the vulnerabilities that need to be addressed in the near future.

Impact Score

The Impact score captures the effects of a successfully exploited vulnerability on the component that suffers the worst outcome that is most directly and predictably associated with the attack.

Impact Score	**Examples**
🟠 **High**	Halting of the chain; loss, locking, or unauthorized withdrawal of funds of many users; arbitrary transaction execution; forging of user messages / circumvention of authorization logic
🟡 **Medium**	Temporary denial of service / substantial unexpected delays in processing user requests (e.g. many hours/days); loss, locking, or unauthorized withdrawal of funds of a single user / few users; failures during transaction execution (e.g. out of gas errors); substantial increase in node computational requirements (e.g. 10x)
🟢 **Low**	Transient unexpected delays in processing user requests (e.g. minutes/a few hours); Medium increase in node computational requirements (e.g. 2x); any kind of problem that affects end users, but can be repaired by manual intervention (e.g. a special transaction)
🔵 **None**	Small increase in node computational requirements (e.g. 20%); code inefficiencies; bad code practices; lack/incompleteness of tests; lack/incompleteness of documentation

Exploitability Score

The Exploitability score reflects the ease and technical means by which the vulnerability can be exploited; it represents the characteristics of the vulnerable component. In the below table we list, for each category, examples of actions by actors that are enough to trigger the exploit. In the examples below:

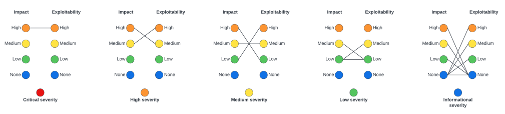
- *Actors* can be any entity that interacts with the system: other blockchains, system users, validators, relayers, but also uncontrollable phenomena (e.g. network delays or partitions).
- *Actions* can be

    - *legitimate*, e.g. submission of a transaction that follows protocol rules by a user; delegation/redelegation/bonding/unbonding; validator downtime; validator voting on a single, but alternative block; delays in relaying certain messages, or speeding up relaying other messages;
    - *illegitimate*, e.g. submission of a specially crafted transaction (not following the protocol, or e.g. with large/incorrect values); voting on two different alternative blocks; alteration of relayed messages.
- We employ also a *qualitative measure* representing the amount of certain class of power (e.g. possessed tokens, validator power, relayed messages): *small* for < 3%; *medium* for 3-10%; *large* for 10-33%, *all* for >33%. We further quantify this qualitative measure as relative to the largest of the system components. (e.g. when two blockchains are interacting, one with a large capitalization, and another with a small capitalization, we employ *small* wrt. the number of tokens held, if it is small wrt. the large blockchain, even if it is large wrt. the small blockchain)

Exploitability Score	**Examples**
🟠 **High**	illegitimate actions taken by a small group of actors; possibly coordinated with legitimate actions taken by a medium group of actors
🟡 **Medium**	illegitimate actions taken by a medium group of actors; possibly coordinated with legitimate actions taken by a large group of actors
🟢 **Low**	illegitimate actions taken by a large group of actors; possibly coordinated with legitimate actions taken by all actors
🔵 **None**	illegitimate actions taken in a coordinated fashion by all actors

Severity Score

The severity score combines the above two sub-scores into a single value, and roughly represents the probability of the system suffering a severe impact with time; thus it also represents the measure of the urgency or order in which vulnerabilities need to be addressed. We assess the severity according to the combination scheme represented graphically below.



As can be seen from the image above, only a combination of high impact with high exploitability results in a Critical severity score; such vulnerabilities need to be addressed ASAP. Accordingly, High severity score receive vulnerabilities with the combination of high impact and medium exploitability, or medium impact, but high exploitability.

Severity Score	**Examples**
🔴 **Critical**	Halting of chain via a submission of a specially crafted transaction
🟠 **High**	Permanent loss of user funds via a combination of submitting a specially crafted transaction with delaying of certain messages by a large portion of relayers
🟡 **Medium**	Substantial unexpected delays in processing user requests via a combination of delaying of certain messages by a large group of relayers with coordinated withdrawal of funds by a large group of users

Severity Score	Examples
🟢 **Low**	2x increase in node computational requirements via coordinated withdrawal of all user tokens
🔵 **Informational**	Code inefficiencies; bad code practices; lack/incompleteness of tests; lack/ incompleteness of documentation; any exploit for which a coordinated illegitimate action of all actors is necessary

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability, etc.) set forth in the associated Services Agreement. This report provided in connection with the Services set forth in the Services Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement.

This audit report is provided on an “as is” basis, with no guarantee of the completeness, accuracy, timeliness or of the results obtained by use of the information provided. Informal has relied upon information and data provided by the client, and is not responsible for any errors or omissions in such information and data or results obtained from the use of that information or conclusions in this report. Informal makes no warranty of any kind, express or implied, regarding the accuracy, adequacy, validity, reliability, availability or completeness of this report. This report should not be considered or utilized as a complete assessment of the overall utility, security or bugfree status of the code.

This audit report contains confidential information and is only intended for use by the client. Reuse or republication of the audit report other than as authorized by the client is prohibited.

This report is not, nor should it be considered, an “endorsement”, “approval” or “disapproval” of any particular project or team. This report is not, nor should it be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts with Informal to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor does it provide any indication of the client’s business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should it be leveraged as investment advice of any sort.

Blockchain technology and cryptographic assets in general and by definition present a high level of ongoing risk. Client is responsible for its own due diligence and continuing security in this regard.