Zellic

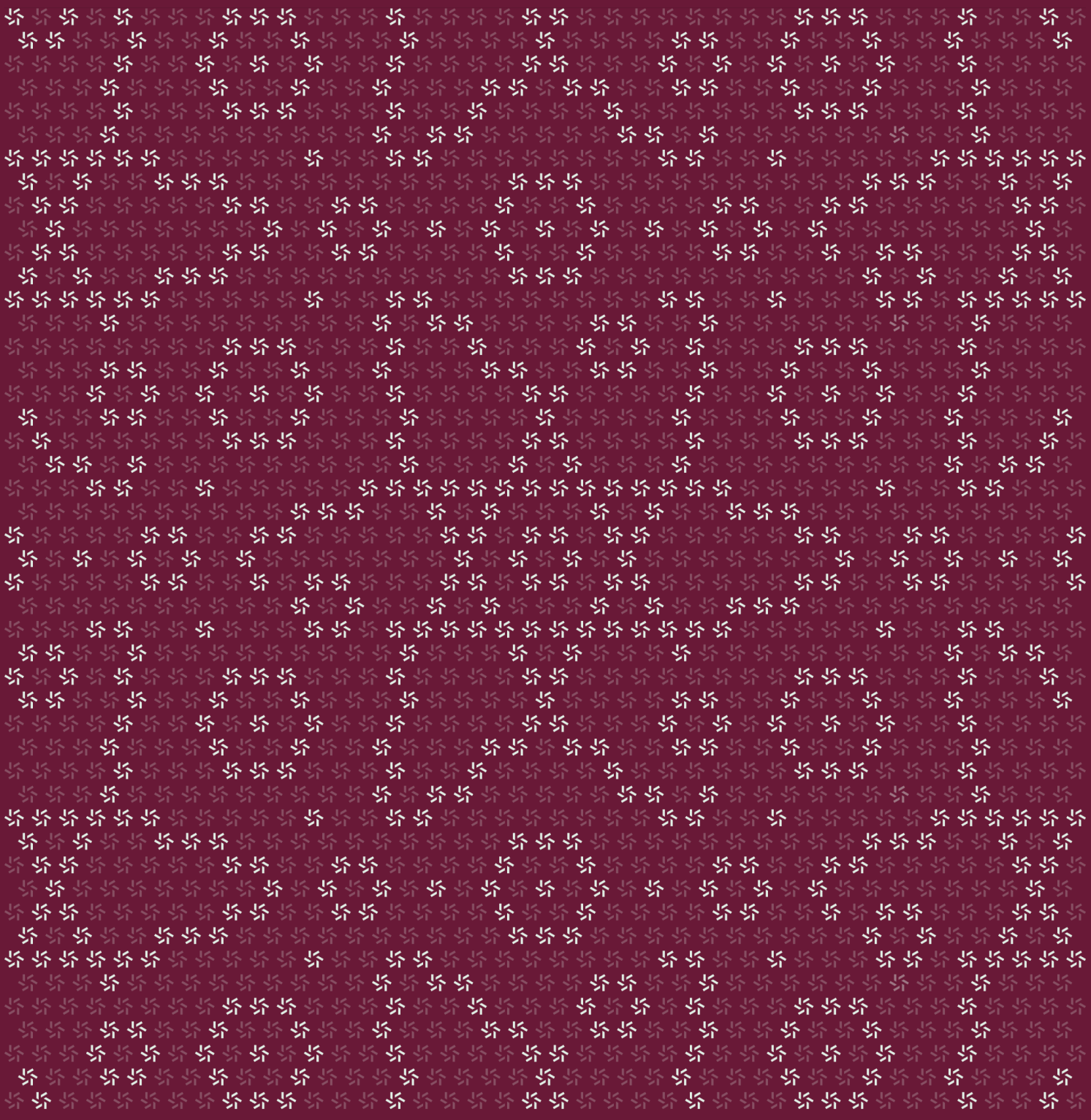
Prepared for
Udit Vira
Timewave

Prepared by
Frank Bachman
Nipun Gupta
Zellic

January 9, 2025

Valence Protocol

Smart Contract Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Timewave from December 2nd, 2024, to January 3rd, 2025. During this engagement, Zellic reviewed Valence Protocol's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Are there any issues concerning asynchronous execution (e.g., error handling, timing failures)?
- Are there any issues causing the contracts to run out of gas?
- Are there issues that allow malicious actors to grief the system by blocking the execution queue?
- Is there any improper handling when part of the system is unavailable (e.g., one blockchain is unreachable)?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

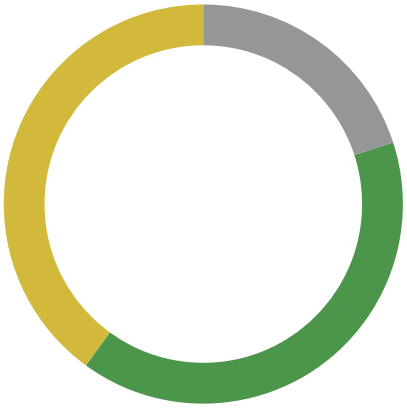
1.4. Results

During our assessment on the scoped Valence Protocol contracts, we discovered five findings. No critical issues were found. Two findings were of medium impact, two were of low impact, and the remaining finding was informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of Timewave in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	2
Low	2
Informational	1



2. Introduction

2.1. About Valence Protocol

Timewave contributed the following description of Valence Protocol:

> We have built a framework and set of libraries that allow application developers to create programs that execute a sequence of operations across multiple different blockchains.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Nondeterminism. Nondeterminism is a leading class of security issues on Cosmos. It can lead to consensus failure and blockchain halts. This includes but is not limited to vectors like wall-clock times, map iteration, and other sources of undefined behavior (UB) in Go.

Arithmetic issues. This includes but is not limited to integer overflows and underflows, floating-point associativity issues, loss of precision, and unfavorable integer rounding.

Complex integration risks. Several high-profile exploits have been the result of unintended consequences when interacting with the broader ecosystem, such as via IBC (Inter-Blockchain Communication Protocol). Zellic will review the project's potential external interactions and summarize the associated risks. If applicable, we will also examine any IBC interactions against the ICS Specification Standard to look for inconsistencies, flaws, and vulnerabilities.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational"

finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

Valence Protocol Contracts

Type	Rust
Platform	CosmWasm
Target	Valence Protocol
Repository	https://github.com/timewave-computer/valence-protocol ↗
Version	`550393b373028148e0a45e227393e58abe2b6ce9`

Programs	`authorization`
	`libraries`
	`processor`
	`program-registry`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 3.6 person-weeks. The assessment was conducted by two consultants over the course of 1.8 calendar weeks.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io

Chad McDonald
Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Frank Bachman
Engineer
frank@zellic.io

Nipun Gupta
Engineer
nipun@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

December 2, 2024	Kick-off call
December 2, 2024	Start of primary review period
January 3, 2025	End of primary review period

3. Detailed Findings

3.1. Pool ratio may change during the queued message

Target	Astroport-withdrawer, Osmosis-gamm-withdrawer		
Category	Business Logic	**Severity**	Medium
Likelihood	Medium	**Impact**	Medium

Description

When assets are withdrawn, the minimum assets to be received by the receiver is calculated in the same function, but this could be different from what users have been expecting. This is because while the message to withdraw the assets is queued, other users could directly swap from the pool and thus change the pool ratio.

For example, in the case of the Astroport-withdrawer library, the `min_assets_to_receive` is calculated in the `create_withdraw_liquidity_msgs` function, as shown below:

```
pub fn create_withdraw_liquidity_msgs(
    deps: &DepsMut,
    cfg: &Config,
) -> Result<Vec<CosmosMsg>, LibraryError> {

    // Calculate how much we are going to get when we withdraw
    let withdrawn_assets: Vec<Asset> = deps.querier.query_wasm_smart(
        cfg.pool_addr.clone(),

    &valence_astroport_utils::astroport_native_lp_token::PoolQueryMsg::Share {
            amount: balance.amount,
        },
    )?;

    // Create the withdraw and send messages
    let withdraw_msg = CosmosMsg::Wasm(cosmwasm_std::WasmMsg::Execute {
        contract_addr: cfg.pool_addr.to_string(),
        msg: to_json_binary(
            &valence_astroport_utils::astroport_native_lp_token::ExecuteMsg::
            WithdrawLiquidity {
                assets: vec![],
                min_assets_to_receive: Some(withdrawn_assets.clone()),
            },
        )?,
        funds: vec![balance],
    });
```

The Osmosis-gamm-withdrawer follows a similar pattern as well, where the expected coins out is calculated in the same transaction.

Impact

If a legitimate user expects that the assets they would receive is of some ratio, the assumption could be violated as soon as the message is queued because the pool ratio could be changed by the time the message is actually executed.

Recommendations

The expected output of coins could be taken as an input in the message itself.

Remediation

This issue has been acknowledged by Timewave, and a fix was implemented in commit `28e8584b` ↗.

3.2. Incorrect ratio calculation

Target	Astroport-lper		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	High	**Impact**	Medium

Description

The Astroport-lper library allows providing liquidity into an Astroport liquidity pool. The double-sided liquidity option allows users to get both the assets from the input account and provide those tokens as liquidity. The function `provide_double_sided_liquidity` first gets the balance of the input account for both the tokens then calculates the actual amount needed based on the ratio/balance of the pool, as shown below:

```
fn calculate_provide_amounts(
    balance1: u128,
    balance2: u128,
    pool_asset1_balance: u128,
    pool_asset2_balance: u128,
    pool_asset_ratio: cosmwasm_std::Decimal,
) -> Result<(u128, u128), LibraryError> {
    // Let's get the maximum amount of assets that we can provide liquidity
    let required_asset1_amount = pool_asset_ratio
        .checked_mul_uint128(balance2.into())
        .map_err(|error| LibraryError::ExecutionError(error.to_string()))?;

    // We can provide all asset2 tokens along with the corresponding maximum of
    asset1 tokens
    if balance1 >= required_asset1_amount.u128() {
        Ok((required_asset1_amount.u128(), balance2))
    } else {
        // We can't provide all asset2 tokens so we need to determine how many
    we can provide according to our available asset1
        let ratio = cosmwasm_std::Decimal::from_ratio(pool_asset1_balance,
    pool_asset2_balance);

        Ok((
            balance1,
            ratio
                .checked_mul_uint128(balance1.into())
                .map_err(|error|
    LibraryError::ExecutionError(error.to_string()))?
```

```
                .u128(),
        ))
    }
}
```

Here, in the case that the balance of asset1 is less than the required amount, the ratio of the pool balance is used to calculate the amount of asset2 needed. The balance of asset2 required is currently calculated as `(pool_asset1_balance/pool_asset2_balance)*balance1`, but it should be `(pool_asset2_balance/pool_asset1_balance)*balance1`.

Impact

The incorrect calculation will lead to an incorrect amount of assets provided to the pool.

Recommendations

We recommend the following changes:

```
let ratio = cosmwasm_std::Decimal::from_ratio(pool_asset1_balance, pool_
    asset2_balance);
let ratio = cosmwasm_std::Decimal::from_ratio(pool_asset2_balance, pool_
    asset1_balance);
```

Remediation

This issue has been acknowledged by Timewave, and a fix was implemented in commit 20988f48 ↗.

3.3. Tokens refunded even if message is partially executed

Target	processor		
Category	Business Logic	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

A message could be forcefully removed by the owner from the queue, which would result in the execution result of the message being set to `ExecutionResult::RemovedByOwner`. If the message is of `Permissioned` type, then the function `process_processor_callback` either burns the tokens if the execution result is `ExecutionResult::Success` or `ExecutionResult::PartiallyExecuted`, or it returns the tokens back in other cases, as shown below:

```
if let OperationInitiator::User(initiator_addr) = &callback.initiator {
    if let AuthorizationMode::Permissioned(PermissionType::WithCallLimit(_)) =
        authorization.mode
    {
        let denom =
            build_tokenfactory_denom(env.contract.address.as_str(),
    &authorization.label);

        let msg = match callback.execution_result {
            ExecutionResult::Success | ExecutionResult::PartiallyExecuted(_,
    _) => {
                // If the operation was executed or partially executed, the
    token will be burned
                burn_msg(env.contract.address.to_string(), 1, denom)
            }
            _ => {
                // Otherwise, the tokens will be sent back
                CosmosMsg::Bank(BankMsg::Send {
                    to_address: initiator_addr.to_string(),
                    amount: coins(1, denom),
                })
            }
        };

        messages.push(msg);
    }
}
```

Impact

In the case the execution result is of the type `ExecutionResult::RemovedByOwner`, the tokens will be returned back to the users, even if the message is partially executed.

Recommendations

We recommend burning the tokens if the execution result is `ExecutionResult::RemovedByOwner`.

Remediation

This issue has been acknowledged by Timewave, and a fix was implemented in commit `23a5f41e` ↗.

3.4. Users may block executions if they initiate callbacks

Target	processor		
Category	Business Logic	**Severity**	Medium
Likelihood	Low	**Impact**	Low

Description

Nonatomic functions could have callbacks associated with them. These callbacks can only be executed with the `pending_callback.address` stored in the configuration. When a message with an atomic/nonatomic function is sent via the authorization module, it increases the `CURRENT_EXECUTIONS` value. This value is only decreased when the message is fully executed on the processor and the processor sends the execution result.

In the case of nonatomic functions with callbacks, the processor sends the execution result after the callback address sends a valid callback (or invalid callback, in which case the execution result is `Rejected` or `PartiallyExecuted`) to the processor module. Therefore, if the callback address does not initiate the callback, the `CURRENT_EXECUTIONS` value will not be decreased for that message, and users may be able to block the authorization's queue as there is a max value for concurrent executions.

Impact

In cases where the user initiates the callback via a call to the callback address, malicious actors may be able to intentionally block the message queue for some authorization module. However, this is not an issue with the valence protocol itself and could only happen due to a misconfiguration by the program creator. For example, if the program is designed such that it relies on an untrustworthy callback originator.

Recommendations

While adding new libraries, consider verifying that the callbacks should not be initiated by the user, which may give them the control to block the executions.

Remediation

This issue has been acknowledged by Timewave.

Timewave contributed the following response:

Our point of view here is that this is a configuration error by the Program creator: they have designed the Program such that it relies on an untrustworthy callback originator. This would fall into the category of user error (or rather error by Program creator), the same as if a Program creator introduced an untrustworthy Library that drains the user's funds or accidentally misconfigured the Program in any other way.

3.5. Unsaved state changes

Target	authorization		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

In the function `retry_bridge_creation`, when the state of a connector is updated, the updated state is not updated in the storage, as shown below:

```
fn retry_bridge_creation(
    deps: DepsMut,
    env: Env,
    domain_name: String,
) -> Result<Response, ContractError> {
    let mut external_domain = EXTERNAL_DOMAINS.load(deps.storage,
    domain_name.clone())?;

    let msg = match external_domain.connector {
        Connector::PolytoneNote {
            address,
            timeout_seconds,
            state,
        } => {
            if state.ne(&PolytoneProxyState::TimedOut) {
                return Err(ContractError::Unauthorized(
                    UnauthorizedReason::BridgeCreationNotTimedOut {},
                ));
            }

            // Update the state
            external_domain.connector = Connector::PolytoneNote {
                address: address.clone(),
                timeout_seconds,
                state: PolytoneProxyState::PendingResponse,
            };
            // It doesn't call `EXTERNAL_DOMAINS.save` or
    `EXTERNAL_DOMAINS.update`
```

Impact

The state changes will not be saved in the storage.

Recommendations

Save the updated state via calling `EXTERNAL_DOMAINS.save` or `EXTERNAL_DOMAINS.update`.

Remediation

This issue has been acknowledged by Timewave, and a fix was implemented in commit `3d1b841e` ↗.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Missing validation checks

As per the pattern in the libraries, the `update_config` function expects an unchecked libconfig and converts it into a checked one (e.g., the addresses, denom and other values) similar to the function validated in each library/../msg.rs. However, in the case of Generic-ibc-transfer, the same checks are not done for `remote_chain_info` in the `update_config` function. More specifically, the `channel_id.is_empty()` and `remote_chain_info.ibc_transfer_timeout` checks are not done in `update_config`.

This issue has been acknowledged by Timewave, and a fix was implemented in commit `4ddd6eb7` ↗.

4.2. Griefing attack vectors

Users can add messages to the queue with/without needing authorization tokens, depending on the type of `AuthorizationMode`. In the case of the `Permissionless` mode, anyone can add new messages. Therefore, anyone could spam the queue if it is permissionless, and if someone wants to execute their message, they would have to process the ticks before their message in the queue, which would require gas fees.

Remediation

The Valence team has acknowledged the issue with the following comment:

> Ticking is permissionless, anyone can do it. The program creator should ideally create the program in a way that someone can just not spam the queue, he has the tools to do it by assigning permissions, time restrictions, execution limits... to each authorization entry. What you describe is possible if the program is configured in such a way that someone can permissionlessly spam the queue. If someone needs to execute a subroutine and there are some subroutines waiting in the queue, then the processor needs to be ticked enough times for that particular subroutine to be executed.

4.3. Unideal swaps while providing single-sided liquidity

When single-sided liquidity is provided via the Astroport-lper, the contract swaps half of the assets provided to the other asset via the pool and then provides liquidity to the pool using both of the assets. As the pool ratio will be changed when assets are swapped, the swapping of exactly half of the provided assets is not ideal. Since the pool ratio is known, consider calculating the exact amount of tokens to swap to maximize the acquired pool share.

Remediation

The Valence team has acknowledged the issue with the following comment:

> The `SingleSidedLiquidity` message is used for small amounts/leftovers after doing double sided liquidity. If for some reason it's triggered for a big amount, we have the limit parameter to limit it and the `max_spread` (called now `slippage_tolerance`) that would prevent a big impact in the pool. In summary, since we are protected by limit and `max_spread` here we don't think it's worth doing these extra calculations for a small move of the pool ratio.

5. System Design

This provides a description of the high-level components of the system and how they interact, including details like a function's externally controllable inputs and how an attacker could leverage each input to cause harm or which invariants or constraints of the system are critical and must always be upheld.

Not all components in the audit scope may have been modeled. The absence of a component in this section does not necessarily suggest that it is safe.

5.1. Authorization

The authorization contract is a single contract deployed on the main domain and defines the authorizations of the top-level application, which can include libraries in different domains (chains). For each domain, there is one processor (with its corresponding execution queues). The authorization contract will connect to all of the processors using a connector (e.g., Polytone, Hyperlane) and will route the functions to be executed to the right domain. At the same time, for each other domain, there is a proxy contract in the main domain that will be forwarding the callbacks received from the processor on the other domain.

The authorization contract is the only address allowed to add a list of functions to the execution queues. It will also be allowed to pause/resume the processor, to arbitrarily remove functions from the queues, or to add certain messages at a specific position.

Invariants

The following conditions must be true while interacting with this module:

1. Only the owner should be allowed to update the ownership or add/remove subowners.
2. Only the owner or subowners should be allowed to execute `PermissionedAction` messages.
3. During the creation of an external domain, another domain of the same name should not already exist.
4. During the creation of an authorization, another authorization of the same name should not already exist.
5. An authorization with the mode `AuthorizationMode::Permissionless` should not have high priority.
6. The domains of the functions for an authorization to be registered should already exist.
7. All functions in an authorization must be executed in the same domain.
8. The maximum number of current executions for an authorization should not exceed its `max_concurrent_executions` value.
9. When inserting or enqueuing messages, the following invariants must hold true:

- The length of the messages and the functions of the authorization module should be the same.
- The message type of the functions and the messages must be true.
- When the messages are converted to JSON, the JSON should only have one top-level key.
- The name of the top-level key should be the same as the name of the message stored in the functions for the authorization.
- The parameter restrictions should be met.

10. Only messages that are in Timeout(retriable) state can be retried.
11. The processor callback should reduce the current executions for the label and update the `execution_result`. If the operation was executed or partially executed, the token will be burned; otherwise, the tokens will be sent back to the initiator.

Test coverage

Cases covered

- The contract is instantiated.
- Only the owner should be able to transfer the ownership.
- Only the owner should be able to add/remove the subowners.
- Only the owner/subowner should be able to create a new authorization.
- Creation of a new authorization with an empty label should revert.
- Creation of a new authorization with a domain that does not exist should revert.
- Creation of a new permissionless authorization with high priority should revert.
- Only the owner/subowner should be able to modify an authorization.
- Modification of an authorization that does not exist should revert.
- Disabling and enabling of authorization should only be allowed for the owner/subowner.
- Minting authorizations for a permissionless type should fail.
- Pausing/resuming of processors should only be allowed to owners/subowners.
- Sending messages to a disabled authorization or an authorization that does not exist should fail.
- Trying to execute an authorization that will be activated in the future should fail because the start time has not been reached yet.
- Sending messages to an expired authorization should fail.
- For `Permissioned` authorization, the user should send the tokens required to send messages.
- Message validation should pass, and the message should contain a valid JSON.
- Enqueueing messages should add the messages to the right queue.
- New messages should not be added if the current executions exceed `max_concurrent_executions`.

Cases not covered

- N/A.

Attack surface

For the `PermissionedAction` type of messages, the attack surface is small, as to perform any malicious action, the keys from the owner/subowner must be stolen by some malicious user.

In the case of `PermissionlessAction` type of messages, users could add messages to the queue with/without needing the authorization tokens depending on the type of `AuthorizationMode`. Adding many messages could potentially delay the executions (as discussed in section 4.2. ↗), but the `Permissionless` ones are always medium priority and thus would not affect the high-priority queue, which is executed first. The `validate_messages` function validates the messages to make sure that the provided message is a JSON string with one top-level key. While reviewing the JSON verification, we noticed that a JSON with more than one of the same keys will be parsed to a single key-value pair (the last one will be considered as the valid one); therefore, the verification will be successful, even if the key-value pairs — except the last one — are all invalid.

For example, if the message passed is `{"key":"value1","key":"value2"}`, then the parsed JSON on which the validations are performed in `validate_messages_generic` will be `{"key":"value2"}`, but the message sent to the `EnqueueMsgs` will still be `{"key":"value1","key":"value2"}`. While we do not consider this a security issue as the conversion of this message to a Cosmos message will fail, we suggest reconstructing the JSON from the parsed result.

The `RetryMsgs` and `RetryBridgeCreation` could be called by anyone only when the original callbacks are not completed and have been timed out. The `ProcessorCallback` could only be called via the `processor_callback_address`, updates the execution result, and reduces the current executions.

5.2. Astroport-lper

Description

The Valence Astroport-lper library allows providing liquidity into an Astroport liquidity pool from an input account and depositing the LP tokens into an output account. The `process_function` describes two messages to do the same:

1. `FunctionMsgs::ProvideDoubleSidedLiquidity` — This provides double-sided liquidity to the preconfigured Astroport pool from the input account and deposits the LP tokens into the output account.
2. `FunctionMsgs::ProvideSingleSidedLiquidity` — This provides single-sided liquidity for the specified asset to the preconfigured Astroport pool from the input account and deposits the LP tokens into the output account.

Invariants

The following conditions must be true while interacting with this module:

1. The caller should be the processor module. It is enforced using the `assert_processor` call in the `execute` function.
2. The call should revert if the pool ratio is not within the `expected_pool_ratio` range.
3. In the case of single-sided liquidity, the asset provided should match one of the assets in the pool.
4. Only the owner should be able to update the processor or transfer ownership.

Test coverage

Cases covered

- Only the owner should be able to update the processor or transfer ownership.
- The library instantiates with incorrect assets.
- The library instantiates with an incorrect pool type.
- Only the processor module should be allowed to execute functions.
- The library provides double-sided liquidity for both native and CW20 LP tokens.
- The library provides single-sided liquidity for both native and CW20 LP tokens.
- The library tests the amount limit for single-sided liquidity.

Cases not covered

- The call should revert if the pool ratio is not within the `expected_pool_ratio` range.

Attack surface

As the pool address, input address, output address, and the pool configuration are already set and cannot be changed except by the owner, the attack surface in this library is quite small. The controlled inputs are just the `asset` (in the case of single-sided liquidity, which is verified to be either one of the assets in the pool), `limit`, and the `expected_pool_ratio_range`.

The `expected_pool_ratio_range` further protects against any kind of sandwich attacks.

5.3. Astroport-withdrawer

The Valence Astroport-withdrawer library allows withdrawing liquidity from an Astroport liquidity pool from an input account and depositing the withdrawn tokens into an output account.

Invariants

The following conditions must be true while interacting with this module:

1. The caller should be the processor module. It is enforced using the `assert_processor` call in the `execute` function.
2. Only the owner should be able to update the processor or transfer ownership.
3. The output tokens should be sent to the output module.

Test coverage

Cases covered

- Only the owner should be able to update the processor or transfer ownership.
- Only the processor module should be allowed to execute functions.
- The library withdraws liquidity for both native and CW20 tokens.

Cases not covered

- N/A.

Attack surface

As the pool address, input address, output address, and the pool configuration are already set and cannot be changed except by the owner, the attack surface in this library is quite small. During the removal of liquidity, the `min_assets_to_receive` value is calculated in the function itself, which might not be the expected value, as discussed in Finding 3.1. ↗.

5.4. Generic-ibc-transfer

The Valence Generic-ibc-transfer library allows transferring funds over IBC from an input account on a source chain to an output account on a destination chain. It is typically used as part of a Valence program. In that context, a processor contract will be the main contract interacting with the Forwarder library. The message could either be a direct transfer to the output account on the destination chain or via a hop mechanism in which the memo field is updated with the `PacketMetadata` that contains the `receiver`, `port`, and the `channel` for the final chain.

Invariants

The following conditions must be true while interacting with this module:

1. The caller should be the processor module. It is enforced using the `assert_processor` call in the `execute` function.

2. Only the owner should be able to update the processor or transfer ownership.
3. The `FixedAmount` value set in the config should not be zero.
4. The time-out for the IBC transfer should be nonzero.
5. The `channel_id` for the remote chain should not be empty.

Test coverage

Cases covered

- The library instantiates with a valid config.
- Instantiation fails if the fixed amount is zero.
- Instantiation fails if the `channel_id` is empty.
- Instantiation fails if the time-out for the IBC transfer is zero.
- The library verifies that the update config correctly updates all the values.
- The transaction will revert if the transfer is initiated with an amount more than the balance.

Cases not covered

- N/A.

Attack surface

As the input address, output address and the transfer configuration are already set and couldn't be changed except by the owner, the attack surface in this library is quite small. There are no other arguments that could be passed during the function call.

5.5. Neutron-ibc-transfer

The Valence Neutron-ibc-transfer library allows transferring funds over IBC from an input account on Neutron to an output account on a destination chain. It is typically used as part of a Valence program. In that context, a processor contract will be the main contract interacting with the Forwarder library. The IBC fee is deducted from the input account as well and therefore should contain sufficient Neutron tokens to cover the fees.

Invariants

The following conditions must be true while interacting with this module:

1. The caller should be the processor module. It is enforced using the `assert_processor` call in the `execute` function.

2. Only the owner should be able to update the processor or transfer ownership.
3. The `FixedAmount` value set in the config should not be zero.
4. The time-out for the IBC transfer should be nonzero.
5. The `channel_id` for the remote chain should not be empty.
6. The `NTRN_DENOM` balance of the input account should be sufficient to cover the fees.

Test coverage

Cases covered

- The library instantiates with a valid config.
- Instantiation fails if the fixed amount is zero.
- Instantiation fails if the `channel_id` is empty.
- Instantiation fails if the time-out for the IBC transfer is zero.
- The library verifies that updating the config correctly updates all the values.
- The transaction will revert if the transfer is initiated with an amount greater than the balance.
- The transaction will revert if there are not sufficient tokens in the input account to cover the fees.

Cases not covered

- N/A.

Attack surface

As the input address, output address, and the transfer configuration are already set and cannot be changed except by the owner, the attack surface in this library is quite small. There are no other arguments that could be passed during the function call.

5.6. Forwarder

The Valence Forwarder library allows continuously forwarding funds from an input account to an output account, following some time constraints. It is typically used as part of a Valence program. In that context, a processor contract will be the main contract interacting with the Forwarder library.

Invariants

The following conditions must be true while interacting with this module:

1. The caller should be the processor module. It is enforced using the `assert_processor` call in the `execute` function.
2. Only the owner should be able to update the processor or transfer ownership.
3. There should not be repeated coins in the forwarding config.
4. If there is some minimum interval set in the forwarding config, it ensures that the duration between each forward is at least the minimum interval.

Test coverage

Cases covered

- The library instantiates with a valid config.
- Instantiation fails for a duplicate denom.
- Instantiation fails for unknown CW20.
- Forwarding the entire balance works as expected.
- Forwarding a partial amount of tokens works as expected.
- Tests with the minimum time-interval constraints work as expected.

Cases not covered

- N/A.

Attack surface

As the pool address, input address, output address, and the forwarding configuration are already set and cannot be changed except by the owner, the attack surface in this library is quite small. There are no other arguments that could be passed during the function call.

Processor

An instance of the Processor contract is present on each of the domains (chains). It handles the execution queues which contain message batches. The source of these message batches is the Authorization contract. The Processor can be ticked permissionlessly, which executes the next message batch in the queue if this one is executable or rotates it to the back of the queue if it isn't executable yet. The processor contract also handles the retry logic for each batch (if the batch is atomic) or function (if the batch is non atomic). After a Message Batch has been executed successfully or it reached the maximum amount of retries, it is removed from the execution queue and the Processor sends a callback with the execution information to the Authorization contract.

The Processor contract has two execution queues, with High and Medium priorities.

Invariants

The following conditions must be true while interacting with this module:

1. Only the Authorization contract should be allowed to queue message batches for execution.
2. The processor must not process messages if it is in the Paused state.
3. Only the Authorization contract should be able to toggle the Paused/Resumed state.
4. Message batches can only be processed if their retry cooldown has expired.
5. When processing a tick, batches should be properly distinguished as atomic/non-atomic to be processed sequentially or batch-wise.
6. Atomic batches must be processed by the processor contract itself through `ExecuteMsg::InternalProcessorAction`.
7. `PENDING_CALLBACK` should be removed after processing a successful callback.
8. Retry attempts for callback messages or bridge creation should occur only if the call back state indicates `TimedOut`.
9. The execution queue's structure should be maintained accurately with internal message insertion, properly handling priority.
10. Any messages except `PermissionlessAction` should be enforced to be access controlled.

Test Coverage

- N/A

Cases not covered

- N/A

Attack Surface

For `InternalProcessorMsg::ExecuteAtomic`, the attack surface is very limited as this message can only originate from the Authorization contract which validates the messages before they are enqueued to the execution queue.

The other two messages (`InternalProcessorAction`, `PolytoneCallback`) are also access controlled and can only be called by the processor contract itself.

`InternalProcessorMsg::LibraryCallback` allows library callbacks from verified senders by verifying the senders from `PENDING_CALLBACK`. It also checks the result of the callback with the expected contacts. The callback is rejected if this fails.

The `RetryMsgs` and `RetryBridgeCreation` messages could be called by anyone only when the original callbacks not completed and have been timed out. The `Tick` message is also permissionless and executes an atomic/non-atomic batch already enqueued in the execution queue. The `High` priority message batches are executed first.

Splitter

The Splitter library enables the distribution of funds from a single input account to multiple output accounts based on a predefined configuration. This is achieved by specifying a split configuration that includes denominations, amounts, or dynamic ratios. This library is typically used as part of a Valence Program, interacting primarily with Processor contracts.

Invariants

The following conditions must be met while interacting with this module:

1. The caller should be the processor module. This is enforced by an `assert_processor` call in the `execute` function.
2. Only the contract owner can update the processor or transfer ownership.
3. The `FixedAmount` specified in the split configuration must be greater than 0.
4. Dynamic ratios used for fund distribution must be successfully retrieved from the specified contracts.
5. Total distribution of funds must not exceed the available balance for any denomination in the input account.
6. The contract addresses and parameter strings for any dynamic ratio calculations must not be empty.

Test Coverage

Cases Covered:

- Verify correct initialization and configuration updating, including handling of invalid configurations.
- Ensure fund splitting executes correctly across native, CW20, and mixed tokens using specified amounts.
- Test failure scenarios where token amounts exceed available balances during splits.
- Validate splits performed correctly using static ratios for native and CW20 tokens.
- Check that splits using dynamic ratios are calculated and executed accurately.

Cases not covered

- N/A

Attack Surface

As the input address, output address and the transfer configuration are already set and couldn't be changed except by the owner, the attack surface in this library is quite small. There are no other arguments that could be passed during the function call.

Osmosis-cl-lper

The Osmosis-cl-lPer library facilitates providing liquidity to a concentrated liquidity pool on the Osmosis from an input account, and deposit the LP tokens into an output account.

Invariants

The following conditions must be ensured while interacting with this module:

1. The library must confirm that the input account holds sufficient assets for both pool assets.
2. For custom positions, the target range must respect the pool tick spacing configuration and be contained within the global tick range.
3. For default liquidity provision, the `bucket_count` must derive a valid tick range that extends symmetrically from the active bucket.
4. Only the owner should be able to update the processor or transfer ownership.

Test Coverage

Cases Covered:

- providing liquidity fails if the pool is not found.
- providing liquidity fails if pool assets do not match expected denominations.
- Test successful provision of liquidity with a valid custom tick range.
- Test successful provision of liquidity using default tick ranges.
- providing liquidity fails with a custom tick range that is not a multiple of the pool's tick spacing.
- providing liquidity fails if the custom tick range's lower tick exceeds the upper tick.
- default liquidity provision fails if it would exceed the configured global tick range.

Cases not covered

- N/A

Attack Surface

The attack surface is minimized by validating the input against the configuration:

- Tick ranges must conform to both the CL pool's tick spacing and the global tick range.
- All configuration and state changes are strictly controlled and cannot be directly influenced by external message inputs. Only the owner has the capability to update the configuration parameters.

Osmosis CL Liquidity Withdrawer

The Valence Osmosis-cl-withdrawer library allows to withdraw a concentrated liquidity position off an Osmosis pool from an input account, and transfer the resulting tokens to an output account.

Invariants

The following conditions must be maintained while interacting with the module:

1. The specified `position_id` must correspond to an existing CL position.
2. For partial withdrawals, the specified `liquidity_amount` should not exceed the total liquidity available in the position.
3. If no amount is specified, the library defaults to withdrawing the entire liquidity available.
4. Only the owner should be able to update the processor or transfer ownership.

Test Coverage

Cases Covered:

- liquidating a specified amount of liquidity where half of the position is successfully liquidated and underlying funds are received.
- liquidating the entire position by defaulting to the full liquidity when no amount is specified.
- attempting to liquidate a position not owned by the input account which should panic with a not owner error.
- attempting to liquidate a non-existing position which should panic due to the position not existing.
- attempting to liquidate an amount greater than the available liquidity, which should panic with an insufficient liquidity error.

Cases not covered

- N/A

Attack Surface

As the input address, output address and the transfer configuration are already set and couldn't be changed except by the owner, the attack surface in this library is quite small. Only `position_id` and `liquidity_amount` are passed as message parameters and correctly validated.

Reverse Fund Splitter

The Valence Reverse Fund Splitter library allows for consolidating funds from multiple input accounts into a single output account based on specified configurations. It can manage a mix of fixed amounts, fixed ratios, and dynamic ratio allocations across different denominations.

Invariants

The following invariants must be maintained when using this module:

1. The calling entity must be the processor module. This is enforced using an `assert_processor` check in the `execute` function.
2. Only the contract owner is authorized to update the processor or transfer ownership.
3. If using `FixedAmount`, the specified amount must not be greater than the account's current balance available in the specified denomination.
4. Dynamic ratios for fund consolidation must be correctly retrieved from the designated contracts.
5. Any dynamic ratio key components such as contract addresses and parameter strings should be non-empty.
6. The consolidation process must be coherent, ensuring the total consolidated funds do not exceed the available balances across all specified accounts for any given denomination.

Test Coverage

Cases Covered:

- Instantiation tests ensure the reverse splitter initializes correctly with various valid and invalid configurations.
- Configuration update tests validate the ability to update configurations with valid and invalid parameters.
- Splitting tests check native and CW20 token amount splitting with varying account setups to verify correct fund consolidation.
- Insufficient balance tests validate error handling when input accounts lack sufficient funds for the specified splits.

- Ratio-based splitting tests confirm the correct handling of both static and dynamic token ratios for fund consolidation, validating the transfer of tokens according to predefined ratios.

Cases not covered

- N/A

Attack Surface

As the input address, output address and the transfer configuration are already set and couldn't be changed except by the owner, the attack surface in this library is quite small. There are no other arguments that could be passed during the function call.

Osmosis GAMM LPer

The Valence Osmosis GAMM LPer library allows to join a pool on Osmosis, using the Generalized Automated Market Maker (GAMM) module, from an input account, and deposit the LP tokens into an output account.

Invariants

The following invariants must be maintained when using this module:

- Ensure that input account balances are sufficient before attempting to provide liquidity.
- Validate the expected spot price range, if passed, to ensure it aligns with the queried pool's spot price.
- For single-sided liquidity provision:
  - Ensure the provisioned amount does not exceed the specified limit.
  - Calculate the appropriate share output amount for the swap.
- For double-sided liquidity provision:
  - Calculate and verify provision amounts based on the current pool ratio.
  - Compute the share output amount without any swap if possible.
- Messages for liquidity provision and LP token transfers must be executed on behalf of the account with the correct permissions.

Test Coverage

Cases Covered:

- Providing liquidity fails when pool assets do not match configuration.
- Two-sided liquidity provision fails if the spot price is outside the specified range.
- Two-sided liquidity is successfully provided when no range is specified.

- Two-sided liquidity is successfully provided when the spot price is within the valid range.
- Single-sided liquidity provision fails if the spot price is outside the specified range.
- Single-sided liquidity is successfully provided when no range is specified.
- Single-sided liquidity is successfully provided when the spot price is within the valid range.

Cases not covered

- N/A

Attack Surface

As the input address, output address and the transfer configuration are already set and couldn't be changed except by the owner, the attack surface in this library is quite small. The message parameters passed are properly validated.

Osmosis GAMM Withdrawer

The Valence Osmosis GAMM Withdrawer library allows to exit a pool on Osmosis, using the GAMM module, from an input account, and deposit the withdrawn tokens into an output account.

Invariants

- Verify the LP token balance of the input account to ensure it holds sufficient tokens to initiate a withdrawal.
- Ensure that liquidity can only be withdrawn if the LP token balance is greater than zero.
- The withdrawal execution simulates the removal to predict the expected returned coin amounts.

Test Coverage

Cases Covered:

- Verifies successful liquidity withdrawal and asset transfer when the input account has sufficient LP tokens.
- Confirms that attempting a withdrawal without LP tokens results in an expected panic error.

Cases not covered

- N/A

Attack Surface

As the input address, output address and the transfer configuration are already set and couldn't be changed except by the owner, the attack surface in this library is quite small. There are no other arguments that could be passed during the function call.

6. Assessment Results

At the time of our assessment, the reviewed code was not deployed to the Mainnet

During our assessment on the scoped Valence Protocol contracts, we discovered five findings. No critical issues were found. Two findings were of medium impact, two were of low impact, and the remaining finding was informational in nature.

6.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.