Zellic

Prepared for
Larry Lyu
Left Curve Software Limited

Prepared by
Nipun Gupta
Avraham Weinstock
Zellic

April 4, 2025

Dango Hyperlane

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 Left Curve Software Limited from March 31st to April 3rd, 2025. During this engagement, Zellic reviewed Dango Hyperlane'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 cross-chain messages correctly sent and received?
- Are received messages correctly authenticated?
- Is the accounting of value transferred, including fees, correct?
- Is there any way to bypass the rate-limiting mechanism?
- What denial-of-service vectors are possible?

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

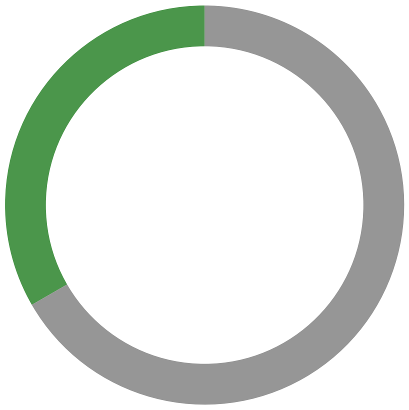
Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped Dango Hyperlane modules, we discovered three findings. No critical issues were found. One finding was of low impact and the other findings were informational in nature.

Breakdown of Finding Impacts

Impact Level	Count
🟥 Critical	0
🟧 High	0
🟨 Medium	0
🟩 Low	1
⬜ Informational	2



2. Introduction

2.1. About Dango Hyperlane

Left Curve Software Limited contributed the following description of Dango Hyperlane:

> Dango is an upcoming decentralized exchange, a novel limit order book, margin system, and user experience. The scope of this audit is Dango's implementation of Hyperlane, a cross-chain interoperability protocol.

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 modules.

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.

2.3. Scope

The engagement involved a review of the following targets:

Dango Hyperlane Modules

Type	Rust
Platform	Cosmos

Target	left-curve
Repository	https://github.com/left-curve/left-curve ↗
Version	`17d6b0e71c2a990887a1d612933e7fbf606e7254`

Programs	`hyperlane/*`
	`dango/genesis/src/lib.rs`
	`dango/warp/*`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 1.2 person-weeks. The assessment was conducted by two consultants over the course of four calendar days.

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:

Nipun Gupta
Engineer
nipun@zellic.io ↗

Avraham Weinstock
Engineer
avi@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

March 31, 2025	Kick-off call
March 31, 2025	Start of primary review period
April 3, 2025	End of primary review period

3. Detailed Findings

3.1. Multisig ISM–verification denial of service

Target	hyperlane/isms/multisig/src/query.rs		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Medium	**Impact**	Low

Description

The Multisig ISM's `Verify` query attempts to check that every signature for the message is valid, even if there are more signatures than configured validators (in this situation, verification will always fail due to the signers not being a subset of the validators).

Impact

Since signature verification is a fairly CPU-intensive operation, and queries do not have a fee to invoke, an attacker can perform a denial-of-service attack by requesting verification of messages with an arbitrary number of signatures.

The following unit test measures how long the ISM's `verify` method takes for 10,000 signatures (approximately 20 seconds on an M3 Macbook):

```
#[test]
fn more_signatures_than_validators() {
    use rand::rngs::OsRng;
    use sha3::{Digest, Keccak256};
    use std::time::Instant;
    let keys = (0..10000)
        .map(|_| k256::ecdsa::SigningKey::random(&mut OsRng))
        .collect::<Vec<_>>();
    let validators: BTreeSet<_> = keys
        .iter()
        .map(|k| {
            HexByteArray::from_inner(

    Keccak256::digest(&k.verifying_key().to_encoded_point(false).to_bytes()[1..])
                [12..]
                .try_into()
                .unwrap(),
            )
        })
        .collect();
```

```
let message = Message {
    version: MAILBOX_VERSION,
    nonce: 36,
    origin_domain: 80001,
    sender:
addr32!("0000000000000000000000004980c17e2ce26578c82f81207e706e4505fae3b"),
    destination_domain: 43113,
    recipient:
addr32!("0000000000000000000000004980c17e2ce26578c82f81207e706e4505fae3b"),
    body: hex!("48656c6c6f21").to_vec().into(),
};

let mut metadata = Metadata {
    origin_merkle_tree: addr32!(
        "0000000000000000000000009af85731edd41e2e50f81ef8a0a69d2fb836edf9"
    ),
    merkle_root:
hash!("a84430f822e0e9b5942faace72bd5b97f0b59a58a9b8281231d9e5c393b5859c"),
    merkle_index: 36,
    signatures: btree_set! {},
};
let multisig_hash = eip191_hash(multisig_hash(
    domain_hash(
        message.origin_domain,
        metadata.origin_merkle_tree,
        HYPERLANE_DOMAIN_KEY,
    ),
    metadata.merkle_root,
    metadata.merkle_index,
    message.encode().keccak256(),
));

let signatures: BTreeSet<_> = keys
    .iter()
    .map(|k| {
        use sha3::digest::generic_array::sequence::Lengthen;
        let (sig, id)
= k.sign_prehash_recoverable(&multisig_hash[..]).unwrap();
        HexByteArray::from_inner(
            sig.to_bytes().append(id.to_byte() + 27)[..]
                .try_into()
                .unwrap(),
        )
    })
    .collect();
    metadata.signatures = signatures;

    let mut ctx = MockContext::new();

    VALIDATOR_SETS
        .save(
            &mut ctx.storage,
            message.origin_domain,
            &ValidatorSet {
                threshold: 1,
                validators: btree_set! { validators.first().cloned().unwrap()
    },
            },
        )
        .unwrap();

    let pre = Instant::now();
    verify(ctx.as_immutable(), &message.encode(),
    &metadata.encode()).should_fail();
    let post = Instant::now();
    println!("Verify time: {:?}", post - pre);
}
```

Recommendations

Require that there are at most as many signatures as validators before verifying any signatures:

```
    let validator_set = VALIDATOR_SETS.load(ctx.storage, message.origin_
        domain)?;

    ensure!(metadata.signatures.len() <= validator_set.validators.len(), "
        more signatures than validators");

     // Loop through the signatures and recover the addresses.
    let validators = metadata
        .signatures
        .into_iter()
        .map(|signature| {
            let pk = ctx.api.secp256k1_pubkey_recover(
                &multisig_hash,
                &signature[..64],
                signature[64] - 27, // Ethereum uses recovery IDs 27, 28
    instead of 0, 1.
```

```
                  false,              // We need the _uncompressed_ public key
      for deriving address!
              )?;
              let pk_hash = ctx.api.keccak256(&pk[1..]);
              let address = &pk_hash[12..];

              Ok(HexByteArray::from_inner(address.try_into().unwrap()))
          })
          .collect::<StdResult<BTreeSet<_>>>()?;

      let validator_set = VALIDATOR_SETS.load(ctx.storage, message.origin_
          domain)?;
```

Remediation

This issue has been acknowledged by Left Curve Software Limited, and a fix was implemented in `PR #616` ↗.

3.2. Centralization risk of ownership mechanism

Target	dango/genesis/src/lib.rs		
Category	Protocol Risks	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

Grug's chain state contains an owner, which has permission to set various configuration settings, including outbound rate limits and alloyed token mappings. The `build_genesis` function initializes the owner to the address of a spot account managed by a single key. The comments indicate an intent to transfer control to a multi-sig account after chain creation, which would be an improvement, but it would still be centralized relative to a governance-proposal mechanism that allows holders of a governance token to vote on governance proposals.

Impact

The capability to set the rate limits and alloyed token mappings allows the owner account to drain escrow accounts on counterparties by adding an attacker-controlled chain to the alloyed token mapping of the token to drain, setting the outbound rate limit sufficiently high to cover the existing escrow balances and initiating outbound transfers.

Recommendations

Initialize the owner directly to a multi-sig account in genesis. If a mechanism for governance proposals is added (perhaps as an account type that allows other account types to vote on proposals), initialize the owner account to that mechanism in genesis.

Remediation

This issue has been acknowledged by Left Curve Software Limited and they provided the following response:

> Noted, not addressing for now. The `build_genesis` [function] is not intended for production. For mainnet, we will use a more decentralized solution for the owner account.

3.3. Counterparty risk of alloyed tokens

Target	dango/warp/src/execute.rs		
Category	Protocol Risks	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The alloyed-token mechanism treats tokens from distinct counterparty chains as fungible (e.g., if chains A and B have `hyp/a/usdc` and `hyp/b/usdc` as their representations of USDC, the alloying mechanism allows treating both of them interchangeably as `hyp/all/usdc`). This simplifies the user experience by only requiring users to manage one representative of a token, but it introduces counterparty risk.

Impact

If the value of chain A's representation of some token that is alloyed on Grug decreases relative to chain B's representation (such as due to a bug allowing minting of tokens on A or an attacker trying to exfiltrate funds from a hack through A), holders of that token on A can swap for the corresponding token on B by transferring through Grug, up to the outbound limit towards B. Users holding the alloyed token on Grug will not be able to withdraw the relatively valuable token on B.

Recommendations

If having a single representation of some token at the interface level is desired, accomplish that by automatically swapping the held representative of the token for the withdrawn version of the token via the DEX on withdrawal if the swap can be accomplished at a 1:1 ratio.

Remediation

This issue has been acknowledged by Left Curve Software Limited and they provided the following response:

> Noted, not addressing for now. We consider the counterparty risk an acceptable tradeoff for the benefit in user experience.

4. 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.

4.1. Component: Hyperlane

Multisig ISM

The Multisig ISM (Interchain Security module) is called to verify the validity of the message via validator signatures. The `verify` function is called from the mailbox, and it returns `Ok(())` if the message is signed by enough validators. It has the following three entry points:

1. `instantiate` — used to initialize the contract (it sets the validator sets for the provided domains)
2. `execute` — used to update/add new validator sets
3. `query` — used to query the state or to verify that the message is signed by enough validators

Invariants

- Only an admin can update/add new validator sets.
- A call to verify should revert if the number of validators that signed the message is less than the threshold.
- A call to verify should revert if the signed validators are not a subset of a validator set stored in the storage.
- Repeated signatures should be rejected.

Test coverage

Cases covered

- Repeated signatures are rejected.
- The threshold for signed validators is not met.

Cases not covered

- Only an admin can update/add new validator sets.

Attack surface

The validator sets should only be allowed to be updated by a trusted account/multi-sig. If the owner account is compromised, then an attacker could update the `VALIDATOR_SETS`, leading to verification of malicious messages. This therefore introduces a centralization issue.

Mailbox

This contract is used to send and receive the cross-chain messages. It has the following three entry points:

1. `instantiate` — used to initialize the contract and set the `CONFIG` and `MERKLE_TREE` values
2. `execute` — used to send/receive the multi-sig message
    - Two types of messages are allowed — `ExecuteMsg::Dispatch` (used to dispatch the message) and `ExecuteMsg::Process` (used to process an incoming message).
3. `query` — used to query the state

Invariants

- The nonce should be incremented during every outgoing message.
- The destination domain cannot be the local domain.
- A specific message can only be delivered once.
- The message version should be `MAILBOX_VERSION`.
- The ISM verification should pass for a message to be processed.

Test coverage

Cases covered

- The Merkle tree is updated as intended (integration tested by Warp).

Cases not covered

- Sending a dispatch message on a destination domain that is the same as the local domain should fail.
- The message version should be `MAILBOX_VERSION`.
- A message cannot be delivered twice.
- If the ISM verification fails, the message should not be processed.

Attack surface

The ISM verification is an important part of the processing of an incoming message. If an attacker could bypass the verification, they might be able to process fraudulent messages. This is protected against by making sure that the ISM verification does not return an error. A message, if it could be delivered twice, could also lead to critical issues; it is protected against by making sure that the `DELIVERIES` set does not contain the `message_id` prior to processing it.

Validator Announcer

The Validator Announcer (VA) contract is used to announce a validator-signature storage location that is used by the relayer. It has the following three entry points:

1. `instantiate` — used to initialize the contract and set the `MAILBOX` and `ANNOUNCE_FEE_PER_BYTE` values
2. `execute` — accepts the `ExecuteMsg::Announce` message type, which announces the location
3. `query` — used to query the state

Invariants

- The announcer should provide the fee during the `announce` call.
- The address recovered should be equal to the validator's address.

Test coverage

Cases covered

- Announcing a validator without sufficient fee fails.
- Announcing a validator with an incorrect/multiple denom fails.
- Announcing the same validator at the same location fails.

Cases not covered

- N/A.

Attack surface

If an attacker could update the storage of another validator, they could change the storage location. The check that the recovered public key is the same as the validator ensures that this does not happen. The fee check also ensures that validators cannot announce without providing sufficient fees.

4.2. Component: Warp

The Warp contract handles fungible-token transfers over Hyperlane, in a manner similar to ICS-20 token transfers but with the messages being delivered through Hyperlane instead of through ICS-4 packets.

It has the following entry points:

1. `instantiate` — used to initialize the contract (the instantiate message contains the address of the Mailbox contract, which the Warp contract uses to send/receive cross-chain messages)
2. `execute` — the Warp contract has five types of messages to execute, three of which (`SetRoute`, `SetAlloy`, and `SetRateLimits`) are used to set configuration by the chain owner, and the other two of which (`TransferRemote` and `Recipient`) are used to initiate outgoing transfers and receive incoming transfers
3. `query` — used to query the contract's state, including which mailbox address it uses, what routes to send outgoing transfers to for each denom, which alloyed token (if any) a denom maps to, and which ISM to use for verifying incoming messages
4. `cron_execute` — automatically executed daily to reset the quota used to limit outgoing transfers

Invariants

- The mailbox configured during instantiation cannot be modified.
- Only the chain owner can execute the `SetRoute`, `SetAlloy`, and `SetRateLimits` messages.
- Nonsynthetic tokens (whose denoms do not start with `hyp/`) are held by the Warp contract in escrow when sent to remote chains.

Test coverage

Cases covered

- Attempting an outbound transfer of a denom with no configured route fails.
- A correct outbound transfer succeeds with the expected mailbox state, fee payment, and transfer events.
- An outbound transfer with a synthetic token results in the token being deducted from the sender but not held by the Warp contract.
- An outbound transfer followed by an incoming transfer with a nonsynthetic token has the expected user and contract balances.
- An incoming transfer with a synthetic token has the expected user balance.
- Incoming transfers for distinct alloyed tokens are received as the alloyed token by the recipient and are held as the non-alloyed representative by the Warp contract.

- Outbound transfers of alloyed tokens deduct the non-alloyed representative from the Warp contract.
- The outbound quota limits outbound transfers and is reset with a simulated passage of time.

Cases not covered

- Non-owners cannot execute `SetRoute`, `SetAlloy`, and `SetRateLimits`.
- Outbound transfers with an insufficient sender balance should fail.
- Incoming transfers for nonsynthetic tokens without sufficient escrowed tokens held by the Warp contract should fail.
- Non-mailbox addresses canot execute `Recipient` to emulate an incoming transfer.

Attack surface

Message: `TransferRemote`

The `TransferRemote` message initiates an outbound transfer.

The caller must provide funds of a single token type. If the token is alloyed, the alloyed tokens are burned, and the underlying token is used for the transfer. A route must be set for the token, and the amount must cover that route's egress fee. The amount (after deduction of the egress fee) must exceed the remaining quota for outbound transfers, and the quota is decreased by the amount. If the token is synthetic (e.g., the underlying token of an alloyed token), it is burned; otherwise, it is escrowed by the Warp contract. The egress fee is paid to the taxman contract, and the message is dispatched to the mailbox to send the funds to the remote chain.

Message: `Recipient`

The `Recipient` message, with its `RecipientMsg::Handle` variant (currently the only variant), processes an incoming transfer.

It is invoked by the mailbox when receiving messages from a remote chain; the mailbox's verification of the sender authenticates this process. The route is looked up to determine which token to use for the sender's domain. If the token is an alloyed token, its underlying token is minted, and its representative is used for the transfer. The outbound quota is increased by the amount. If the token is synthetic, it is minted to the recipient; otherwise, it is transferred from the Warp contract's escrow.

4.3. Component: Genesis

The genesis component is used at the time of chain initialization to upload and instantiate some contracts. The following contracts are instantiated at genesis:

- account_factory

- bank
- dex
- hyperlane, which includes the mailbox, ism, and va
- lending
- oracle
- taxman
- vesting
- warp

A few genesis users are also initialized, along with the contracts mentioned above. There is no attack surface as an arbitrary user could not modify/call this directly.

The genesis component configures the Warp contract's cron job, which resets the quota for outbound transfers, to execute daily.

5. Assessment Results

At the time of our assessment, the reviewed code was not deployed to the Dango mainnet.

During our assessment on the scoped Dango Hyperlane modules, we discovered three findings. No critical issues were found. One finding was of low impact and the other findings were informational in nature.

5.1. 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.