

# Wormhole Terra

# Audit

---



Presented by:

**OtterSec**

**James Wang**

**Robert Chen**

[contact@osec.io](mailto:contact@osec.io)

[james.wang@osec.io](mailto:james.wang@osec.io)

[r@osec.io](mailto:r@osec.io)



# Contents

- 01 Executive Summary** **2**
  - Overview . . . . . 2
  - Key Findings . . . . . 2
  - Scope . . . . . 2
- 02 Findings** **3**
- 03 General Findings** **4**
  - OS-TRA-SUG-00 | Possible Address Collision . . . . . 5
  - OS-TRA-SUG-01 | Incorrect Address Normalization . . . . . 6

**Appendices**

- A Vulnerability Rating Scale** **7**
- B Procedure** **8**

# 01 | Executive Summary

## Overview

Wormhole Foundation engaged OtterSec to assess an upgrade to the terra program. This assessment was conducted between January 13th and January 16th, 2024. For more information on our auditing methodology, refer to [Appendix B](#).

## Key Findings

We produced 2 findings throughout this audit engagement.

In particular, we made recommendations around preventing potential address collisions with the native format ([OS-TRA-SUG-00](#)). Additionally, we highlighted an incongruity in the address normalization process that, although rare, may result in the loss of user funds ([OS-TRA-SUG-01](#)).

## Scope

The source code was delivered to us in a Git repository at [github.com/wormhole-foundation/wormhole/tree/main/terra](https://github.com/wormhole-foundation/wormhole/tree/main/terra), where we reviewed the pull request [#3138](#) which is responsible for updating the contract for Terra classic CosmWasm 1.0+ changes.

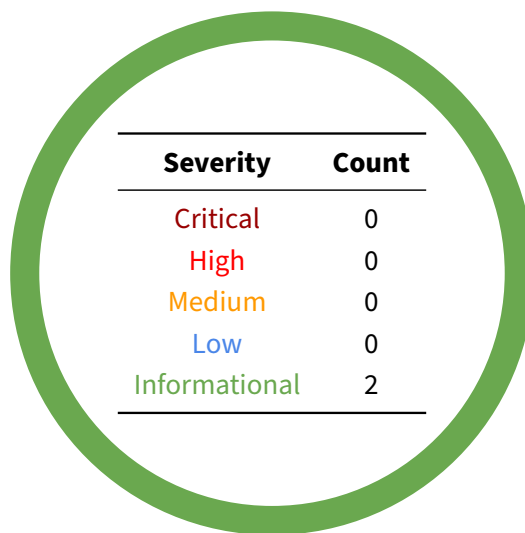
A brief description of the programs is as follows:

Name	Description
terra	Program which contains the Terra smart contracts for the Wormhole wrapped token and message / token bridge. This update focuses on handling changes in CosmWasm customs query API and change in contract address length.

## 02 | Findings

Overall, we reported 2 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
Critical	0
High	0
Medium	0
Low	0
Informational	2

## 03 | General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-TRA-SUG-00	Prevention of potential address collisions with the native format by ensuring the Terra CW20 token contract addresses do not start with [1, 0, 0, ...].
OS-TRA-SUG-01	If the target receiver is a 32-byte address starting with twelve zeros, the address normalization process may incorrectly strip leading zeros.

## OS-TRA-SUG-00 | Possible Address Collision

### Description

In `handle_initiate_transfer_token`, a check exists while bridging Terra CW20 tokens, which requires the asset's address not to resemble a specific native address format, represented by an array starting with `[1, 0, 0, ...]` (eleven zeros). However, if there were a new CW20 address format that is 32 bytes long, there is a theoretical possibility that it starts with `[1, 0, 0, ...]`. The probability of this happening is extremely low ( $(\frac{1}{2})^{96}$ ). However, if such a collision occurs, it may yield unexpected behavior or conflicts.

*s-token-bridge/src/contract.rs*

RUST

```
fn handle_initiate_transfer_token(
    [...],
) -> StdResult<Response> {
    [...]
    // Ensure that the asset's address does not collide with the native
    // address format. This is impossible for legacy CW20 addresses as they are
    // 20 bytes long left padded with 0s, so their first byte can't be 1.
    // However, it's theoretically possible for a new 32 byte CW20 address to have
    // this format. The probability of this happening is  $1 / 2^{96} \approx 1.2 * 10^{-29}$ ,
    // so it is negligible. Regardless, we block such addresses here
    // for the sake of completeness and documentation.
    assert!(!is_native_id(&asset_address));
    [...]
}
```

### Remediation

Implement logic to wrap tokens before bridging.

### Patch

Wormhole's team decided not to patch it since the collision probability was extremely low.

## OS-TRA-SUG-01 | Incorrect Address Normalization

### Description

Within `handle_complete_transfer_token`, an issue arises when the target receiver is a 32-byte address (CosmWasm contract) that starts with `[0, 0, 0, ...]` (twelve zeros). In such a case, there is a potential for the address to be falsely identified as a legacy address, given the initial 16 bytes being zeros, skipping the subsequent 12 bytes as they are removed in `get_address`. It is important to note that the likelihood of such a scenario occurring is exceedingly rare, on the order of  $(\frac{1}{2})^{96}$ . However, in the event of such an edge case, it may result in fund loss if the transfer occurs to an incorrect address.

*wormhole/src/byte\_utils.rs*

RUST

```
fn get_address(&self, mut index: usize) -> CanonicalAddr {
    // Legacy terra addresses are 20 bytes, but new addresses (after the cosmwasmd
    //   ↪ 1.0 upgrade)
    // are 32 bytes. In the Wormhole wire format, addresses are always encoded as
    //   ↪ 32 bytes,
    // so in order to determine which type of address we're dealing with, we check
    //   ↪ the first
    // 12 bytes of the address. If they are all 0, then we have a legacy address,
    //   ↪ otherwise
    // we have a new address.
    // grab the first 16 bytes (u128) then shift right 4 bytes (32 bits) to get the
    //   ↪ first 12bytes.
    // If the number is 0, then we have a legacy address.
    let legacy_address: bool = self.get_u128_be(index) >> 32 == 0;
    [...]
}
```

### Remediation

Review `get_address` and the normalization logic to ensure that it accurately identifies and handles legacy and new 32-byte addresses, especially when the address starts with leading zeros.

### Patch

Wormhole's team decided to accept the risk and not patch it since the collision probability was extremely low.

# A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the [General Findings](#) section.

---

<b>Critical</b>	<p>Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.</p> <p>Examples:</p> <ul style="list-style-type: none"><li>• Misconfigured authority or access control validation.</li><li>• Improperly designed economic incentives leading to loss of funds.</li></ul>
<b>High</b>	<p>Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.</p> <p>Examples:</p> <ul style="list-style-type: none"><li>• Loss of funds requiring specific victim interactions.</li><li>• Exploitation involving high capital requirement with respect to payout.</li></ul>
<b>Medium</b>	<p>Vulnerabilities that may result in denial of service scenarios or degraded usability.</p> <p>Examples:</p> <ul style="list-style-type: none"><li>• Computational limit exhaustion through malicious input.</li><li>• Forced exceptions in the normal user flow.</li></ul>
<b>Low</b>	<p>Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.</p> <p>Examples:</p> <ul style="list-style-type: none"><li>• Oracle manipulation with large capital requirements and multiple transactions.</li></ul>
<b>Informational</b>	<p>Best practices to mitigate future security risks. These are classified as general findings.</p> <p>Examples:</p> <ul style="list-style-type: none"><li>• Explicit assertion of critical internal invariants.</li><li>• Improved input validation.</li></ul>

---



## B | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.