OtterSec

Wormhole Solana Token22

Security Assessment

September 26th, 2025 — Prepared by OtterSec

Tamta Topuria tamta@osec.io

Robert Chen r@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

Wormhole foundation engaged OtterSec to assess the `solana-token-22` program. This assessment was conducted between September 15th and September 25th, 2025. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 3 findings throughout this audit engagement.

In particular, we identified a vulnerability where the parser incorrectly accepts mint accounts sized between 82 and 165 bytes, which are invalid, risking incorrect metadata deserialization (OS-WST-ADV-00). Additionally, the length check during account deserialization always fails, as it enforces fixed account sizes, resulting in Token-2022's variable-length accounts being rejected, rendering the Token-2022 integration insufficient to actually accept t22 tokens (OS-WST-ADV-01).

We also recommended modifying the current transfer logic to work with these Token-2022 extensions, or to disallow attestation of mints that rely on incompatible extensions (OS-WST-SUG-00).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/wormhole-foundation/wormhole. This audit was performed against PR#4482.

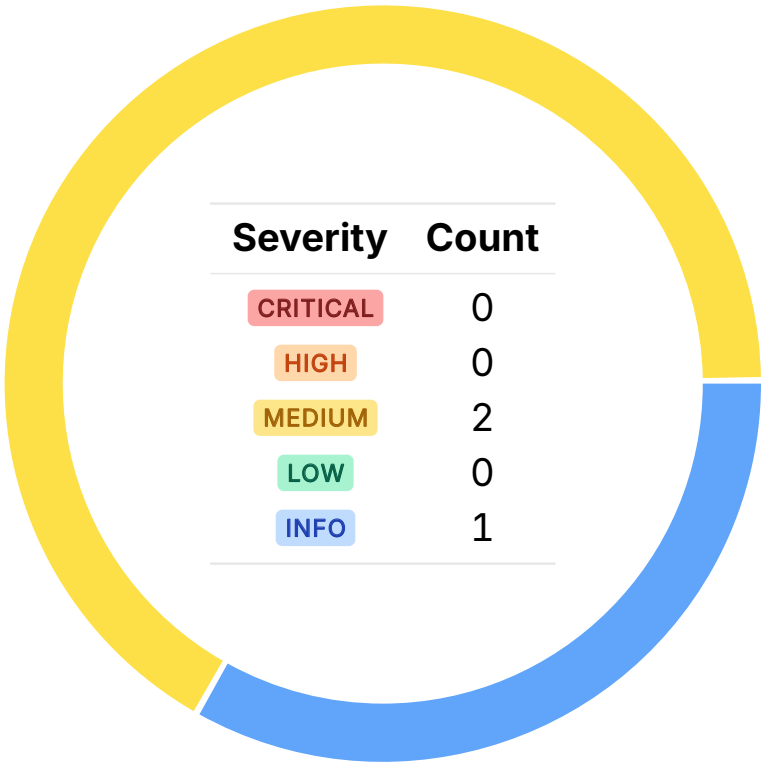
A brief description of the program is as follows:

Name	Description
solana-token-22 (PR#4482)	The PR fixes metadata handling and adds support for Token2022 tokens, including metadata pointers.

03 — Findings

Overall, we reported 3 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	2
LOW	0
INFO	1

04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-WST-ADV-00	MEDIUM	RESOLVED	The parser wrongly accepts mint accounts sized between 82 and 165 bytes, which are invalid, risking incorrect metadata deserialization.
OS-WST-ADV-01	MEDIUM	RESOLVED	`unpack_unchecked` in legacy `SPL` enforces fixed account sizes, resulting in Token-2022's variable-length accounts being rejected.

Invalid Mint Account Size Handling MEDIUM OS-WST-ADV-00

Description

`parse_token2022_metadata` incorrectly treats any mint account larger than 82 bytes but smaller than 165 bytes as a valid older layout. However, no legitimate `Mint` account may have a size in this range. It is either exactly 82 bytes (no extensions) or greater than 165 bytes (with extensions). As a result, the parser may deserialize malformed accounts, resulting in incorrect metadata interpretation.

>_ *wormhole/solana/modules/token_bridge/token_metadata_parser/src/lib.rs* RUST

```rust
pub fn parse_token2022_metadata(
    mint_account: Pubkey,
    mint_account_data: &[u8],
) -> Result<MintMetadata, ParseError> {
    [...]
    let (account_type_offset, start_offset) = if mint_account_data.len() > BASE_MINT_LEN_V2 {
        (ACCOUNT_TYPE_OFFSET_V2, ACCOUNT_TYPE_OFFSET_V2 + 1)
    } else {
        // Older layout
        (ACCOUNT_TYPE_OFFSET, ACCOUNT_TYPE_OFFSET + 1)
    };
    [...]
```

Remediation

Update the condition to reject accounts with sizes between 82 and 165 bytes.

Patch

Resolved in be2f4fe.

Code Breaking Changes MEDIUM OS-WST-ADV-01

Description

Currently, the length check during account deserialization always fails, rendering the Token-2022 integration insufficient to actually accept `t22` tokens. The legacy `SPL` function, `unpack_unchecked` enforces a strict `len == LEN` check, which works for legacy `SPL` accounts with fixed sizes (82-byte `Mints`). However, this is not the case for `t22` mints. Token-2022 accounts are variable-length due to TLV extensions, so their size exceeds these fixed values, and they always fail deserialization.

>_ *solana-sdk/program-pack/src/lib.rs* RUST

```rust
/// Unpack from slice without checking if initialized
fn unpack_unchecked(input: &[u8]) -> Result<Self, ProgramError> {
    if input.len() != Self::LEN {
        return Err(ProgramError::InvalidAccountData);
    }
    Self::unpack_from_slice(input)
}
```

Remediation

Remove the account size check, but implement alternative checks and sufficient measures to prevent account confusion.

Patch

Resolved in 4fada95.

05 — 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-WST-SUG-00	In the current implementation, attested mints with certain Token-2022 extensions become non-transferable because the basic transfer instruction does not support them

Failed Transfers Due to Unsupported Extensions OS-WST-SUG-00

Description

In the current transfer model, mints with extensions such as `TransferHook`, `TransferFee`, or `PausableAccount` may be attested; however, the basic `transfer` instruction does not support these features (unlike `transfer_checked`). This results in all transfers of such tokens failing after attestation.

Remediation

Replace `transfer` with `transfer_checked`, or disallow attestation of mints with incompatible extensions.

Patch

The Wormhole team has decided to block attestation of unsupported extensions on the client side.

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.

CRITICAL

Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
- Improperly designed economic incentives leading to loss of funds.

HIGH

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
- Exploitation involving high capital requirement with respect to payout.

MEDIUM

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

LOW

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

INFO

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
- Improved input validation.

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 others may have 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.