OtterSec

USDT OFT

Security Assessment

January 28th, 2025 — Prepared by OtterSec

Robert Chen r@osec.io

Samuel Bétrisey sam@osec.io

Table of Contents

01 — Executive Summary

Overview

LayerZero Labs engaged OtterSec to assess the `usdtOFT` program. This assessment was conducted between October 9th, 2024 and January 20th, 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 made a suggestion regarding a deserialization inconsistency in the transfer ownership functionality (OS-UFT-SUG-00), and highlighted a possible scenario where funds may be locked due to invalid metadata (OS-UFT-SUG-01).

Scope

The source code was delivered to us in a Git repository at https://github.com/LayerZero-Labs/x-usdtoft. This audit was performed against e255830.

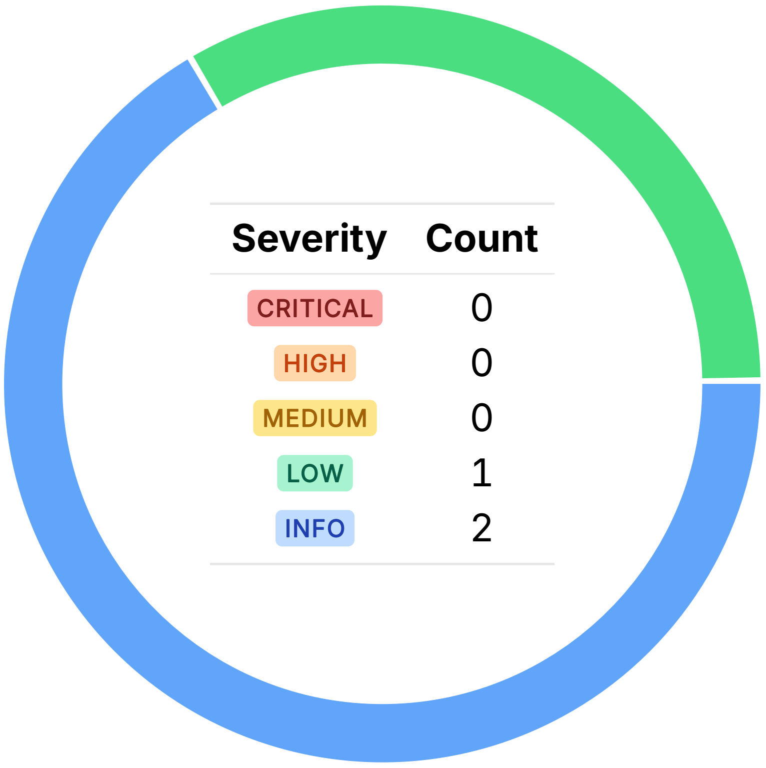
A brief description of the program is as follows:

Name	Description
usdtOFT	The multi-adapter OFT (madOFT) is an OFT that allows an OFT to be deployed with three or more lock/unlock adapter pools.

02 — 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	0
LOW	1
INFO	2

03 — 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-UFT-ADV-00	LOW	RESOLVED ✓	Due to an error in the opcode whitelist, the OApp cannot configure custom settings for its channel.

Incorrect opcode whitelisted LOW OS-UFT-ADV-00

Description

In `setLzConfig`, the wrong opcode is whitelisted. Indeed, `Endpoint::OP::SET_EP_CONFIG_OAPP` is only callable by the controller.

Instead, the OApp should allow sending a message with the `Controller::OP::SET_EP_CONFIG_OAPP` opcode to the controller which will forward it to the endpoint. This issue prevents the configuration of custom settings for its channel.

>_ *x-usdtoft/packages/usdtoft-ton/src/oApp/handlerOApp.fc* FUNC

```
tuple setLzConfig(cell $Config) impure inline method_id {
    [...]
    (
        cell $SendPath,
        int forwardingAddress,
        int opCode,
        cell $innerConfig
    ) = lz::Config::deserialize($Config);

    _assertSendPath($SendPath);

    ifnot (
        (opCode == Endpoint::OP::SET_EP_CONFIG_OAPP) |
        (opCode == MsglibManager::OP::SET_OAPP_MSGLIB_SEND_CONFIG) |
        (opCode == MsglibManager::OP::SET_OAPP_MSGLIB_RECEIVE_CONFIG)
    ) {
        throw(ERROR::InvalidLzConfigOpCode);
    }
    [...]
}
```

Remediation

Replace `Endpoint::OP::SET_EP_CONFIG_OAPP` by `Controller::OP::SET_EP_CONFIG_OAPP`.

Patch

Resolved in a059442.

04 — 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-UFT-SUG-00	Suggestion regarding a deserialization inconsistency in `transferOwnership`.
OS-UFT-SUG-01	If a `Jetton::OP::TRANSFER_NOTIFICATION` is received with invalid metadata, the code unexpectedly reaches a fallback line that returns an empty tuple, locking the funds.

Deserialization Discrepancy

OS-UFT-SUG-00

Description

`transferOwnership` in `usdtoft` stores the new owner's address at a fixed offset utilizing `$Address.Address::getAddress()`. However, when emitting the `TentativeOwnerSet` event, the new owner's address is retrieved via `$Address.Address::sanitize()`, which depends on `cl::get<address>` to parse the cell based on its type information. This discrepancy may result in the address stored as the new tentative owner differing from the address recorded in the event. Utilize the correct method to extract the address.

>_ *x-usdtoft/packages/usdtoft-ton/src/oApp/handlerOApp.fc* FUNC

```
tuple transferOwnership(cell $Address) impure inline {
    (cell $storage, tuple actions) = preamble();
    setContractStorage(
        setOAppStorage(
            $storage,
            getOAppStorage().cl::set(
                OApp::tentativeOwner,
                $Address.Address::getAddress()
            )
        )
    );
    actions~pushAction<event>(
        EVENT::TentativeOwnerSet,
        $Address.Address::sanitize()
    );

    return actions;
}
```

Remediation

Implement the above-mentioned suggestion.

Patch

Resolved in 9df3f47.

Locked Funds due to Invalid Metadata OS-UFT-SUG-01

Description

The `Jetton::OP::TRANSFER_NOTIFICATION` opcode is triggered when a jetton (token) transfer notification is received. When receiving a `Jetton::OP::TRANSFER_NOTIFICATION` with an invalid `$md` (`$md` that is neither `null` nor of type `OFTSend`), an empty tuple is returned and the funds will be locked, even though the comment says this code will never be executed. This should not occur given the SDK is properly implemented. The impact is somewhat mitigated as it is possible for the owner to recover the funds.

Remediation

Ensure the SDK is implemented correctly.

Patch

The LayerZero team has acknowledged this issue.

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.