Zellic

Prepared for
Luka Avbreht
Filip Koprivec
Flare Network

Prepared by
Sunwoo Hwang
Weipeng Lai
Zellic

September 4, 2025

FAsset OFTAdapter

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 Flare Network from August 28th to August 29th, 2025. During this engagement, Zellic reviewed FAsset OFTAdapter'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:

- Is the token contract compatible with LayerZero?
- Is there any issue in the inheritance of the OFT contract?

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 FAsset OFTAdapter contracts, we discovered one finding, which was of low impact.

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

Breakdown of Finding Impacts

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

2. Introduction

2.1. About FAsset OFTAdapter

Flare Network contributed the following description of FAsset OFTAdapter:

> The OTF Adapter implementation for FAssets. Used to configure and enable bridging of the FAssets using L0 OFT 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 contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

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:

FAsset OFTAdapter Contracts

Type	Solidity
Platform	EVM-compatible

Target	fassets
Repository	https://gitlab.com/flarenetwork/fassets ↗
Version	`e35b73b3d06cdaa81bf20746d22dfde5c352fb6b`
Programs	`/fasset-oftadapter/contracts/*.sol`

2.4. Project Overview

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

Pedro Moura
Engagement Manager
pedro@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Sunwoo Hwang
Engineer
sunwoo@zellic.io ↗

Weipeng Lai
Engineer
weipeng.lai@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

August 28, 2025	Kick-off call
August 28, 2025	Start of primary review period
August 29, 2025	End of primary review period

3. Detailed Findings

3.1. Owner can set OFT sending fee to 100%

Target	FAssetOFT, FAssetOFTAdapter		
Category	Business Logic	**Severity**	Medium
Likelihood	Low	**Impact**	Low

Description

Both FAssetOFT and FAssetOFTAdapter inherit fee control from `FeeUpgradeable`, which allows the owner to set the fee up to 10,000 basis points (bps), that is, 100%.

```
uint16 public constant BPS_DENOMINATOR = 10_000;

function setDefaultFeeBps(uint16 _feeBps) external onlyOwner {
    if (_feeBps > BPS_DENOMINATOR) revert IFee.InvalidBps();
    FeeStorage storage $ = _getFeeStorage();
    $.defaultFeeBps = _feeBps;
    emit DefaultFeeBpsSet(_feeBps);
}

function setFeeBps(uint32 _dstEid, uint16 _feeBps, bool _enabled)
    external onlyOwner {
    if (_feeBps > BPS_DENOMINATOR) revert IFee.InvalidBps();
    FeeStorage storage $ = _getFeeStorage();
    $.feeBps[_dstEid] = FeeConfig(_feeBps, _enabled);
    emit FeeBpsSet(_dstEid, _feeBps, _enabled);
}
```

Fees are subtracted from the send amount before slippage enforcement. If a user submits `minAmountLD = 0`, a 100% fee results in `amountReceivedLD` being `0`, and the transaction does not revert.

```
function _debitView(
    uint256 _amountLD,
    uint256 _minAmountLD,
    uint32 _dstEid
) internal view virtual override returns (uint256 amountSentLD,
    uint256 amountReceivedLD) {
    amountSentLD = _amountLD;

    // @dev Apply the fee, then de-dust the amount afterwards.
```

```
        // This means the fee is taken from the amount before the dust is removed.
        uint256 fee = getFee(_dstEid, _amountLD);
        unchecked {
            amountReceivedLD = _removeDust(_amountLD - fee);
        }

        // @dev Check for slippage.
        if (amountReceivedLD < _minAmountLD) {
            revert SlippageExceeded(amountReceivedLD, _minAmountLD);
        }
    }
```

Impact

This configuration creates a centralization risk. If the owner's private key is compromised, an attacker can set the fee to 100%. Cross-chain transactions that specify `minAmountLD = 0` would transfer the entire amount as fees without reverting, and the attacker could withdraw the accrued fees via `withdrawFees`.

Recommendations

We recommend enforcing a protocol-level fee cap below 10,000 bps (for example, 100–500 bps) in the fee logic. Specifically, bind the computed value in `getFee` so the effective fee never exceeds the cap.

Remediation

This issue has been acknowledged by Flare Network, and a fix was implemented in commit `6895fb18` ↗.

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. FAsset pause could cause inbound `lzReceive` to fail

In emergencies, FAsset transfers can be paused. While paused, the FAssetOFTAdapter cannot credit incoming tokens because its `_credit` path calls `innerToken.safeTransfer`, which reverts under a pause:

```
function _credit(
    address _to,
    uint256 _amountLD,
    uint32 /*_srcEid*/
) internal virtual override returns (uint256 amountReceivedLD) {
    // @dev Unlock the tokens and transfer to the recipient.
    innerToken.safeTransfer(_to, _amountLD);
    // @dev In the case of NON-default OFTAdapter, the amountLD MIGHT not be ==
    amountReceivedLD.
    return _amountLD;
}
```

As a result, inbound LayerZero `lzReceive` executions on that chain will revert.

If FAsset is paused, users should be notified to stop sending FAssetOFT to the chain hosting the FAssetOFTAdapter until the token is unpaused.

4.2. Dust amounts are collected as fees

In FAssetOFT and FAssetOFTAdapter, sent amounts undergo de-dusting to ensure that `amountReceivedLD` fits within six shared decimals.

```
function _debitView(
    uint256 _amountLD,
    uint256 _minAmountLD,
    uint32 _dstEid
) internal view virtual override returns (uint256 amountSentLD,
    uint256 amountReceivedLD) {
    amountSentLD = _amountLD;
```

```
        // @dev Apply the fee, then de-dust the amount afterwards.
        // This means the fee is taken from the amount before the dust is removed.
        uint256 fee = getFee(_dstEid, _amountLD);
        unchecked {
            amountReceivedLD = _removeDust(_amountLD - fee);
        }

        // @dev Check for slippage.
        if (amountReceivedLD < _minAmountLD) {
            revert SlippageExceeded(amountReceivedLD, _minAmountLD);
        }
    }
```

The FXRPOFT token uses six decimals, so no dust is produced when sending it cross-chain. However, if a future FAsset uses more than six decimals, the following dust-related considerations apply.

When users send amounts that do not align with six-decimal precision after applying the fee, the excess dust is retained by the contract as an additional fee. This mechanism has implications.

For contract owners, even when explicit fees are set to zero, the contract still accumulates dust fees, so regular use of `withdrawFees()` is necessary to retrieve them.

For users, the optimal strategy is to send amounts that align with six-decimal precision after fee deduction. Sending imprecise amounts may result in paying more while receiving the same net amount. Users should calculate their send amounts to avoid unnecessary dust generation.

5. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

5.1. Module: FAssetOFTAdapter.sol

Function: `constructor(address _token, address _lzEndpoint)`

This function sets the token and LayerZero endpoint addresses when the contract is deployed.

Inputs

- `_token`
  - **Validation**: Must be a valid token address.
  - **Impact**: The token address.
- `_lzEndpoint`
  - **Validation**: Must be a valid LayerZero endpoint address.
  - **Impact**: The LayerZero endpoint address.

Branches and code coverage

Intended branches

- [x] Successfully sets the token and LayerZero endpoint addresses.

Negative behavior

- [ ] Reverts if the `_token` is the zero address.
- [ ] Reverts if the `_lzEndpoint` is the zero address.

Function: `initialize(address _owner)`

This function initializes the FAssetOFTAdapter contract.

Inputs

- `_owner`

- **Validation**: Must be a valid owner address.
- **Impact**: The owner address for setting the fee of the OFTAdapter contract.

Branches and code coverage

Intended branches

- [x] Successfully sets the token owner.

Negative behavior

- [ ] Reverts if the `_owner` is the zero address.

5.2. Module: FAssetOFT.sol

Function: `constructor(address _lzEndpoint)`

This function sets the LayerZero endpoint address when the contract is deployed.

Inputs

- `_lzEndpoint`
  - **Validation**: Must be a valid LayerZero endpoint address.
  - **Impact**: The LayerZero endpoint address.

Branches and code coverage

Intended branches

- [x] Successfully sets the `_lzEndpoint` address.

Negative behavior

- [ ] Reverts if the `_lzEndpoint` address is the zero address.

Function: `initialize(string _name, string _symbol, address _owner)`

This function initializes the FAssetOFT contract.

Inputs

- `_name`

    - **Validation**: Must be a valid token name.
    - **Impact**: The token name.
- `_symbol`
    - **Validation**: Must be a valid token symbol.
    - **Impact**: The token symbol.
- `_owner`
    - **Validation**: Must be a valid owner address.
    - **Impact**: The owner address for setting the fee of the OFT contract.

Branches and code coverage

Intended branches

- [x] Successfully sets the token name, symbol, and owner.

Negative behavior

- [ ] Reverts if the `_owner` is the zero address.

6. Assessment Results

During our assessment on the scoped FAsset OFTAdapter contracts, we discovered one finding, which was of low impact.

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.