Ether.Fi SyncPools

Security Assessment

April 17th, 2024 — Prepared by OtterSec

Robert Chen r@osec.io

James Wang james.wang@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

LayerZero Labs engaged OtterSec to assess the `syncPools` program. This assessment was conducted between April 1st and April 7th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 3 findings throughout this audit engagement.

We identified a high-risk vulnerability involving the use of the wrong destination peer address, resulting in messages being sent to the incorrect contract (OS-ESP-ADV-00). Additionally, we highlighted a scenario where excess funds are stored in the balance of the contract instead of being automatically transferred (OS-ESP-ADV-01).

We also made recommendations around the need for adherence to coding best practices and ensuring efficiency (OS-ESP-SUG-00).

Scope

The source code was delivered to us in a Git repository at https://github.com/LayerZero-Labs/Etherfi-SyncPools. This audit was performed against commit a5ce170.

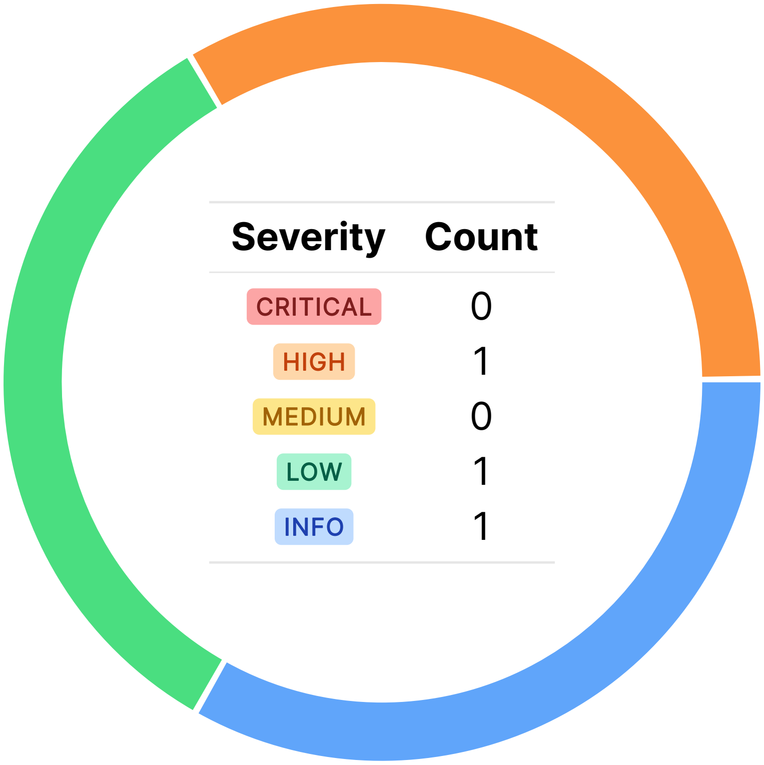
A brief description of the programs is as follows:

Name	Description
syncPools	An upgradable version of the L2SyncMint framework which facilitates efficient native minting on Layer 2 chains by enabling batched syncs for low-cost transactions, fast updates via LayerZero, and cost-effective ETH bridging through native L2 bridge.

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	1
MEDIUM	0
LOW	1
INFO	1

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-ESP-ADV-00	HIGH	RESOLVED ✓	Messages are sent to the incorrect contract due to the use of the wrong destination peer address in `_sync`.
OS-ESP-ADV-01	LOW	RESOLVED ✓	Excess `msg.value` will be parked in `L1BaseSyncPoolUpgradeable` instead of being automatically transferred, if `_finalizeDeposit` is executed before `_anticipatedDeposit`.

Incorrect Destination Peer HIGH

OS-ESP-ADV-00

Description

The destination peer contract is incorrectly determined within `_sync` in `L2LineaSyncPoolETHUpgradeable` and `L2ModeSyncPoolETHUpgradeable`. `_sync` determines the peer address using `_getPeerOrRevert` based on the destination endpoint id (`dstEid`). However, the intended recipient of the synchronization message should be the receiver contract (`L1ModeReceiverETH`) rather than the `LZ` peer (`L1BaseSyncPool`).

>_ *contracts/L2/syncPools/L2LineaSyncPoolETHUpgradeable.sol* rust

```rust
function _sync(
    uint32 dstEid,
    address l2TokenIn,
    address l1TokenIn,
    uint256 amountIn,
    uint256 amountOut,
    bytes calldata extraOptions,
    MessagingFee calldata fee
) internal virtual override returns (MessagingReceipt memory) {
    [...]

    address peer = address(uint160(uint256(_getPeerOrRevert(dstEid))));

    [...]

    IMessageService(getMessenger()).sendMessage{value: amountIn}(peer, 0, message);
    return receipt;
}
```

Sending synchronization messages to the wrong destination compromises the integrity and security of the synchronization process. It may also result in inconsistencies between the `L2` and `L1` states, potentially disrupting the operation of the application.

Remediation

Update `_sync` to ensure that synchronization messages are sent to the correct destination peer contract (`L1ModeReceiverETH`).

Patch

Fixed in b52373f.

Inconsistency In Handling Of Excess Funds LOW OS-ESP-ADV-01

Description

If `_finalizeDeposit` is called before `_anticipatedDeposit`, `msg.value` determines the deposit amount, and any excess `msg.value` remains in the balance of `L1BaseSyncPoolUpgradeable`. Parking excess `msg.value` in the contract's balance may not be desirable. While the funds may be recovered manually, it adds complexity and may result in inefficient fund management.

Remediation

Replace `msg.value` with `self.balance` in `_finalizeDeposit`. This guarantees that any surplus funds in the contract's balance are accounted for in the deposit process, preventing the accumulation of unexpected funds.

Patch

Fixed in b4ead0f.

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-ESP-SUG-00	Suggestions regarding ensuring adherence to coding best practices and code optimization.

Code Maturity

OS-ESP-SUG-00

Description

1. In `_sync` within `L2ModeSyncPoolETHUpgradeable`, ensure to set a non-zero value for `minGasLimit` to prevent potential relay failures when transmitting a message from Layer two to Layer one using `sendMessage`.
2. `L1BaseSyncPoolUpgradeable` does not invoke `__ReentrancyGuard_init`. Although the current absence of `ReentrancyGuard` initialization in the contract may not manifest any discernible effects on its functionality, it is advisable to initialize it to maintain consistency and comply with security standards.

Remediation

Implement the above-mentioned suggestions.

Patch

Fixed in cea4056 and 8ec8d5d.

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 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.