JUNE.24

SECURITY REVIEW REPORT FOR LAYERZERO

CONTENTS

ABOUT HEXENS

Hexens is a cybersecurity company that strives to elevate the standards of security in Web 3.0, create a safer environment for users, and ensure mass Web 3.0 adoption.

Hexens has multiple top-notch auditing teams specialized in different fields of information security, showing extreme performance in the most challenging and technically complex tasks, including but not limited to: Infrastructure Audits, Zero Knowledge Proofs / Novel Cryptography, DeFi and NFTs. Hexens not only uses widely known methodologies and flows, but focuses on discovering and introducing new ones on a day-to-day basis.

In 2022, our team announced the closure of a $4.2 million seed round led by IOSG Ventures, the leading Web 3.0 venture capital. Other investors include Delta Blockchain Fund, Chapter One, Hash Capital, ImToken Ventures, Tenzor Capital, and angels from Polygon and other blockchain projects.

Since Hexens was founded in 2021, it has had an impressive track record and recognition in the industry: Mudit Gupta - CISO of Polygon Technology - the biggest EVM Ecosystem, joined the company advisory board after completing just a single cooperation iteration. Polygon Technology, 1inch, Lido, Hats Finance, Quickswap, Layerswap, 4K, RociFi, as well as dozens of DeFi protocols and bridges, have already become our customers and taken proactive measures towards protecting their assets.

EXECUTIVE SUMMARY

OVERVIEW

This audit covered LayerZero Network's Omnichain Fungible Token (OFT) Arbitrum native contracts, which integrate the OFT directly into the existing Arbitrum native bridge. This integration allows users to move between L3s using either the Arbitrum native bridge or the native OFT bridge interchangeably, without facing any race conditions or issues of splitting liquidity.

Our security assessment was a full review of the two smart contracts, spanning a total of 2 days.

During our audit we have identified several minor severity vulnerabilities and code optimisations.

Finally, all of our reported issues were fixed by the development team and consequently validated by us.

We can confidently say that the overall security and code quality have increased after completion of our audit.

SCOPE

The analyzed resources are located on:

https://github.com/LayerZero-Labs/orbit-adapter/blob/master/src/L2/OrbitERC20OFTAdapter.sol

https://github.com/LayerZero-Labs/orbit-adapter/blob/master/src/L3/NativeOFTAdapterMsgValueTransfer.sol

Commit: b20f7820794e6e0e79254ed2617cdf2a0090f0d7

The issues described in this report were fixed in the following commit:

https://github.com/LayerZero-Labs/orbit-adapter/tree/87f8f99f7a776df8a176512e10f26c2df156b309

AUDITING DETAILS

STARTED

05.06.2024

DELIVERED

07.06.2024

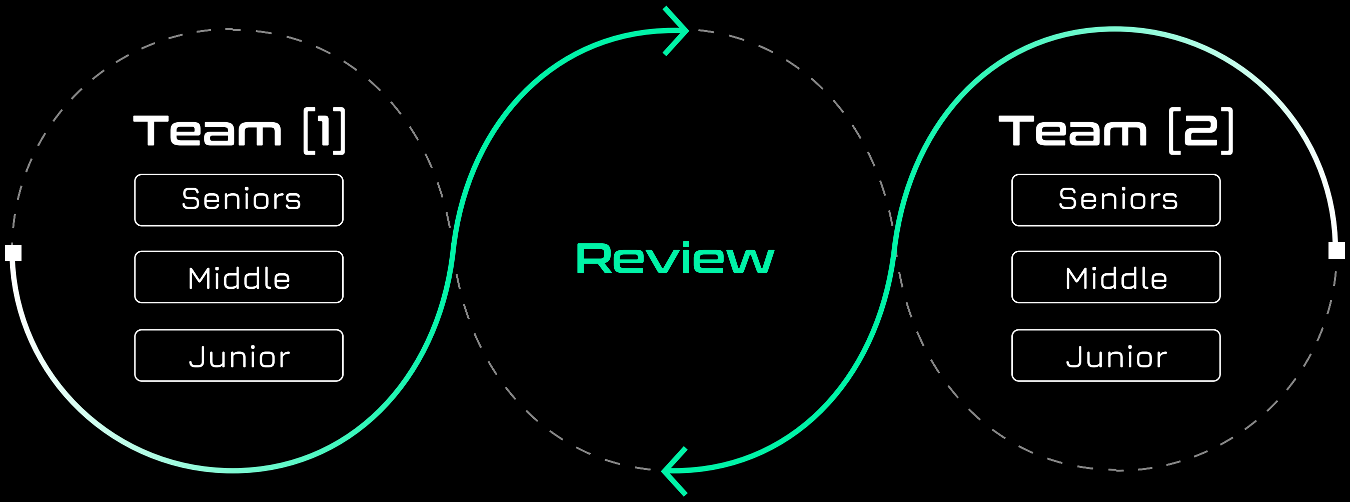
Review Led by

TRUNG DINH

Senior Security Researcher | Hexens

HEXENS METHODOLOGY

Hexens methodology involves 2 teams, including multiple auditors of different seniority, with at least 5 security engineers. This unique cross-checking mechanism helps us provide the best quality in the market.



SEVERITY STRUCTURE

The vulnerability severity is calculated based on two components

- Impact of the vulnerability
- Probability of the vulnerability

Impact	Probability			
	rare	unlikely	likely	very likely
Low/Info	Low/Info	Low/Info	Medium	Medium
Medium	Low/Info	Medium	Medium	High
High	Medium	Medium	High	Critical
Critical	Medium	High	Critical	Critical

SEVERITY CHARACTERISTICS

Smart contract vulnerabilities can range in severity and impact, and it's important to understand their level of severity in order to prioritize their resolution. Here are the different types of severity levels of smart contract vulnerabilities:

Critical

Vulnerabilities with this level of severity can result in significant financial losses or reputational damage. They often allow an attacker to gain complete control of a contract, directly steal or freeze funds from the contract or users, or permanently block the functionality of a protocol. Examples include infinite mints and governance manipulation.

High

Vulnerabilities with this level of severity can result in some financial losses or reputational damage. They often allow an attacker to directly steal yield from the contract or users, or temporarily freeze funds. Examples include inadequate access control integer overflow/underflow, or logic bugs.

Medium

Vulnerabilities with this level of severity can result in some damage to the protocol or users, without profit for the attacker. They often allow an attacker to exploit a contract to cause harm, but the impact may be limited, such as temporarily blocking the functionality of the protocol. Examples include uninitialized storage pointers and failure to check external calls.

Low

Vulnerabilities with this level of severity may not result in financial losses or significant harm. They may, however, impact the usability or reliability of a contract. Examples include slippage and front-running, or minor logic bugs.

Informational

Vulnerabilities with this level of severity are regarding gas optimizations and code style. They often involve issues with documentation, incorrect usage of EIP standards, best practices for saving gas, or the overall design of a contract. Examples include not conforming to ERC20, or disagreement between documentation and code.

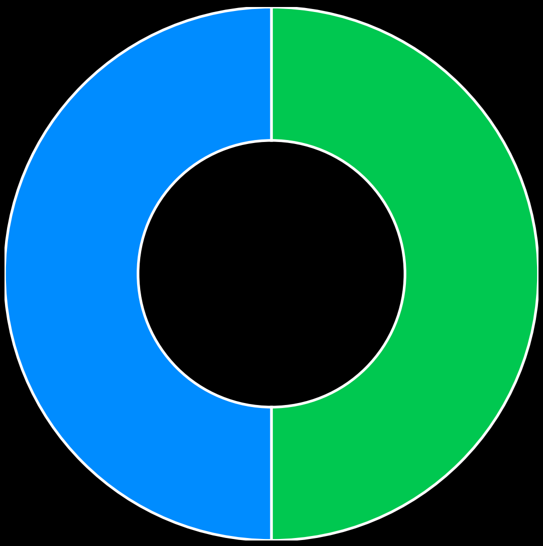
ISSUE SYMBOLIC CODES

Every issue being identified and validated has its unique symbolic code assigned to the issue at the security research stage. Cause of the vulnerability reporting flow design, some of the rejected issues could be missing.

FINDINGS SUMMARY

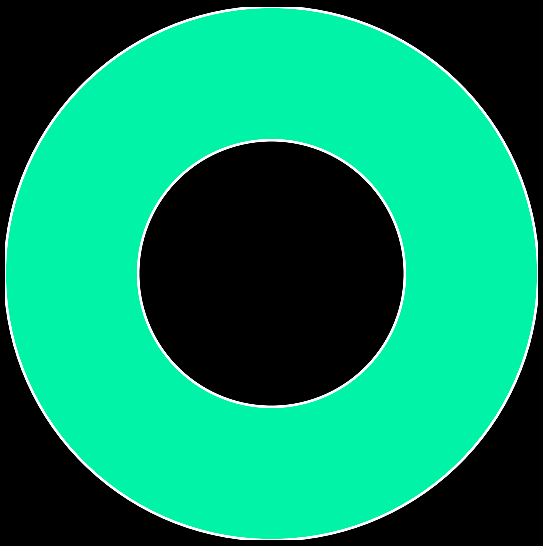
Severity	Number of Findings
Critical	0
High	0
Medium	0
Low	2
Informational	2

Total: 4



● Low

● Informational



● Fixed

WEAKNESSES

This section contains the list of discovered weaknesses.

LZ02-7

VARIABLE BRIDGE CAN BE SET TO IMMUTABLE

SEVERITY: Low

PATH:

src/L2/OrbitERC20OFTAdapter.sol:L13

REMEDIATION:

Consider changing the bridge variable to immutable.

STATUS: Fixed

DESCRIPTION:

The address variable **bridge** in contract **OrbitERC20OFTAdapter** is only set once during the constructor, and there is no setter function to modify its value. Therefore, we can declare **bridge** as an immutable variable to save gas.

```
IBridge private bridge;
```

LZ02-10

SOME NATIVE TOKENS CAN BE LOST WHEN SENDING TOKENS FROM L3

SEVERITY: Low

PATH:

src/L3/NativeOFTAdapterMsgValueTransfer.sol:L150-L153

src/L3/NativeOFTAdapterMsgValueTransfer.sol:L119-L128

REMEDIATION:

See description.

STATUS: Fixed

DESCRIPTION:

In the contract **NativeOFTAdapterMsgValueTransfer**, the function **_payNative()** is overridden to adapt to the new implementation of the function **send()**, which allows users to mint OFT tokens using the **msg.value** when the **msgSenderBalance < _sendParam.amountLD**. The **nativeFee** is then modified to **msg.value - mintAmount** to account for the mint amount.

However, when the **msgSenderBalance > _sendParam.amount**, there is no verification between the **msg.value** and the **feeWithExtraAmount.nativeFee**, other than the requirement that **msg.value** must be greater than **feeWithExtraAmount.nativeFee** in the **_payNative()** function.

This lack of verification will lead to a scenario where:

- msg.value > feeWithExtraAmount.nativeFee AND
- msgSenderBalance > _sendParam.amount

This will result in the amount **msg.value - feeWithExtraAmount.nativeFee** of native tokens being lost in the contract.

```
function _payNative(uint256 _nativeFee) internal override returns (uint256
nativeFee) {
    if (msg.value < _nativeFee) revert NotEnoughNative(msg.value);
    return _nativeFee;
}
```

```
if (msgSenderBalance < _sendParam.amountLD) {
    require(msgSenderBalance + msg.value >= _sendParam.amountLD,
"NativeOFTAdapterMsgValueTransfer: Insufficient msg.value");


    // user can cover difference with additional msg.value ie. wrapping
    uint mintAmount = _sendParam.amountLD - msgSenderBalance;
    _mint(address(msg.sender), mintAmount);


    // update the messageFee to take out mintAmount
    feeWithExtraAmount.nativeFee = msg.value - mintAmount;
}
```

Consider after line 128 adding a else block to verify whether **msg.value == feeWithExtraAmount.nativeFee**

```
if (msgSenderBalance < _sendParam.amountLD) {
    require(msgSenderBalance + msg.value >= _sendParam.amountLD,
"NativeOFTAdapterMsgValueTransfer: Insufficient msg.value");


    // user can cover difference with additional msg.value ie. wrapping
    uint mintAmount = _sendParam.amountLD - msgSenderBalance;
    _mint(address(msg.sender), mintAmount);


    // update the messageFee to take out mintAmount
    feeWithExtraAmount.nativeFee = msg.value - mintAmount;
}
+ else {
+     require(msg.value == feeWithExtraAmount.nativeFee);
+ }
```

LZ02-9

USE CUSTOM ERRORS

SEVERITY: Informational

PATH:

src/L3/NativeOFTAdapterMsgValueTransfer.sol

REMEDIATION:

Replace require statements with Custom Errors for a more streamlined and user-friendly experience. Furthermore, custom errors are much clearer as they allow for parameter values, making debugging much easier.

STATUS: Fixed

DESCRIPTION:

The **NativeOFTAdapterMsgValueTransfer.sol** contract's various errors are currently handled using native Solidity **revert** statements and custom errors, with some exceptions.

For example:

```
require(success, "NativeOFTAdapterMsgValueTransfer: failed to unwrap");
```

src/L3/NativeOFTAdapterMsgValueTransfer.sol:L85

```
require(success, "NativeOFTAdapterMsgValueTransfer: failed to unwrap");
```

src/L3/NativeOFTAdapterMsgValueTransfer.sol:L97

```
require(balanceOf(msg.sender) >= _amount, "NativeOFTAdapterMsgValueTransfer:
Insufficient balance.");
```

src/L3/NativeOFTAdapterMsgValueTransfer.sol:L100

```
require(success, "NativeOFTAdapterMsgValueTransfer: failed to unwrap");
```

src/L3/NativeOFTAdapterMsgValueTransfer.sol:L120

```
require(msgSenderBalance + msg.value >= _sendParam.amountLD,
"NativeOFTAdapterMsgValueTransfer: Insufficient msg.value");
```

LZO2-11

CONSIDER BURNING OFT TOKENS INSTEAD OF TRANSFERRING THEM WHEN SENDING TOKENS CROSS-CHAIN

SEVERITY: Informational

PATH:

src/L3/NativeOFTAdapterMsgValueTransfer.sol:L64

REMEDIATION:

It is recommended to burn the OFT tokens from the sender instead of transferring them to the OFT contract.

STATUS: Fixed

DESCRIPTION:

In the function **NativeOFTAdapterMsgValueTransfer._debit()**, the OFT tokens are transferred to the contract itself. Since the contract has no mechanism to manage these tokens, they are effectively rendered "dead".

```
        _transfer(msg.sender, address(this), amountSentLD);
```

hexens × LayerZero.