hexens × LayerZero.

OCT.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 focused on Upgradeable EVM OApp/OFT LayerZero contracts, which represent the upgradeable version of the OFT contract. The implementation derivatives are similar to the already deployed Frax OFT tokens, but with a preference for OpenZeppelin V5 contracts.

Our security assessment was a full review of the two smart contracts, spanning a total of 3 days.

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

Finally, all of our reported issues were acknowledged or 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/devtools/pull/934/commits/91f18a76bdabc5ff4092686fcfa47b16d87e82b1

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

https://github.com/LayerZero-Labs/devtools/pull/934/commits/b509b39f72e1c841a3217096278952e4bb4f19ab

AUDITING DETAILS

STARTED
28.10.2024

DELIVERED
01.11.2024

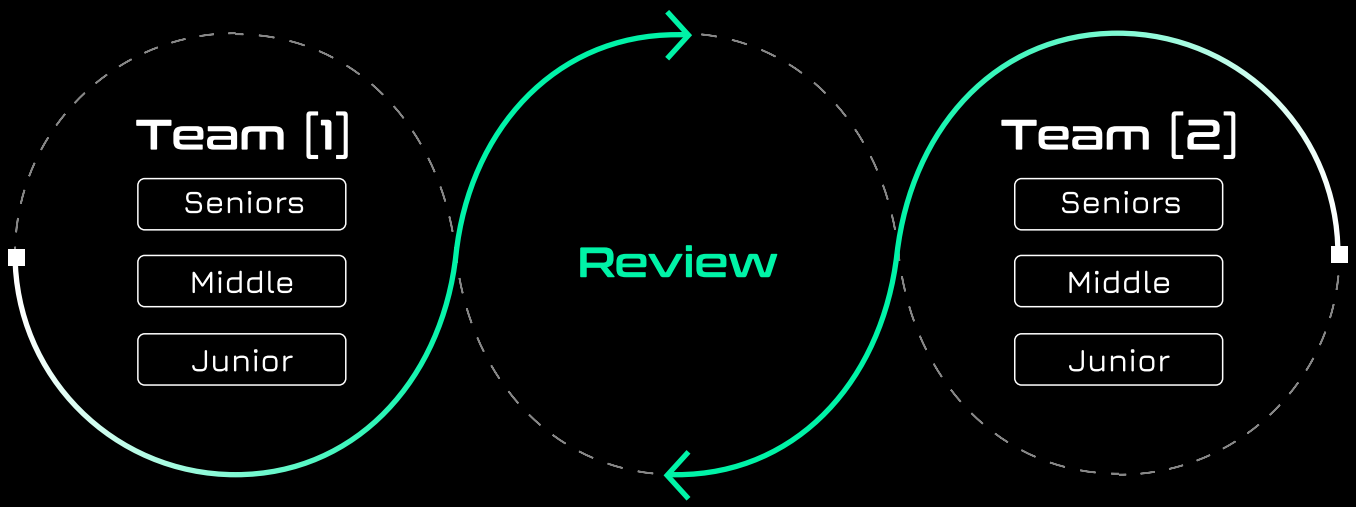
Review Led by

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.

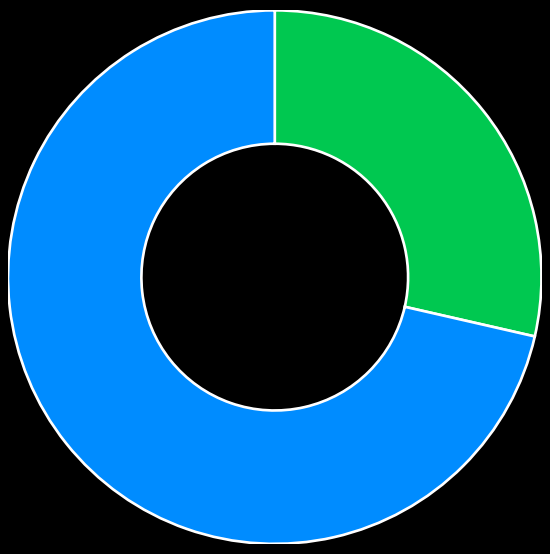
ISSUE SYMBOLIC CODES

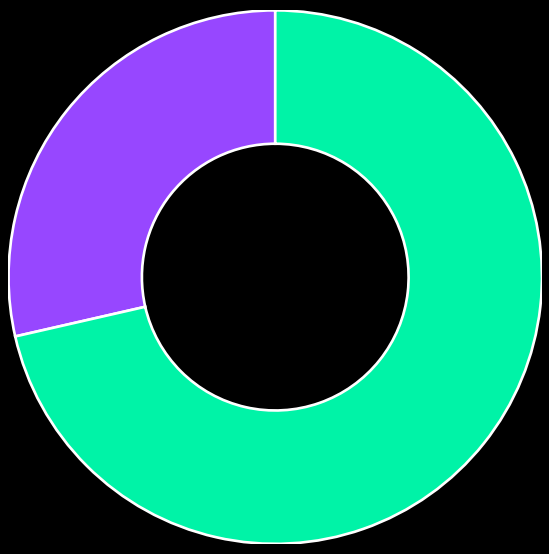
Every issue being identified and validated has its unique symbolic code assigned to the issue at the security research stage. Cause of the vulnerability reporting flow design, some of the rejected issues could be missing.

FINDINGS SUMMARY

Severity	Number of Findings
Critical	0
High	0
Medium	0
Low	2
Informational	5

Total: 7





WEAKNESSES

This section contains the list of discovered weaknesses.

LZOUPG-3

REDUNDANT CALL TO __OAPPCORE_INIT() IN THE OFTCOREUPGRADEABLE INITIALIZER

SEVERITY: Low

PATH:

packages/oft-evm-upgradeable/contracts/oft/OFTCoreUpgradeable.sol#L98-L103

REMEDIATION:

We recommend to remove the call __OAppCore_init(_delegate); in __OFTCore_init().

STATUS: Fixed

DESCRIPTION:

The **__OAppCore_init()** initializer will be called twice in **__OFTCore_init()** as **__OApp_init()** also contains a call to it.

```
function __OFTCore_init(address _delegate) internal onlyInitializing {
    __OAppCore_init(_delegate); // 1
    __OApp_init(_delegate); // <--
    __OAppPreCrimeSimulator_init();
    __OAppOptionsType3_init();
}
```

```
function __OApp_init(address _delegate) internal onlyInitializing {
    __OAppCore_init(_delegate); // 2
    __OAppReceiver_init();
    __OAppSender_init();
}
```

TODO'S IN CODE

SEVERITY: Low

PATH:

packages/oapp-evm-upgradeable/contracts/precrime/PreCrimeUpgradeable.sol#194-198

REMEDIATION:

Consider resolving the TODO's.

STATUS: Acknowledged

DESCRIPTION:

There are open TODO's in code where a simulation **PreCrimeUpgradeable::simulate()** may revert due to the enforced nonce order on line 198 in PreCrimeUpgradeable.sol.

The same TODO also exists in PreCrime.sol.

TODO's may refer to open issues that should be resolved.

```
// TODO ??
// Wont the nonce order not matter and enforced at the OApp level? the simulation will revert?

// the following packet's nonce add 1 in order
if (packet.origin.nonce != ++nonce) revert PacketUnsorted();
```

LZOUPG-2

THE INHERITANCE PATTERN IN OAPPUPGRADEABLE CAN BE IMPROVED

SEVERITY: Informational

PATH:

packages/oapp-evm-upgradeable/contracts/oapp/OAppUpgradeable.sol#L32-L37

REMEDIATION:

It is recommended to:

- replace __OAppReceiver_init()/__OAppSender_init() with __OAppReceiver_init_unchained()/__OAppSender_init_unchained() in __OApp_init()
- add the call to __OAppCore_init() in __OAppReceiver_init()/__OAppSender_init()

STATUS: Fixed

DESCRIPTION:

Since **OAppUpgradeable** doesn't directly inherit **OAppCoreUpgradeable**, the **OAppUpgradeable::__OApp_init()** initializer shouldn't call **__OAppCore_init()** and **__OAppReceiver_init()/__OAppSender_init()** together, accordingly with the best practices from the following resources:

- Using with Upgrades - OpenZeppelin Docs
- Why keep "__Ownable_init" beside "__Ownable_init_unchained"?

```
function __OApp_init(address _delegate) internal onlyInitializing {
    __OAppCore_init(_delegate);
    __OAppReceiver_init();
    __OAppSender_init();
}
```

```
function __OAppReceiver_init() internal onlyInitializing {}
function __OAppReceiver_init_unchained() internal onlyInitializing {}
```

```
function __OAppSender_init() internal onlyInitializing {}
function __OAppSender_init_unchained() internal onlyInitializing {}
```

REDUNDANT FUNCTION PRECRIMEPEERS()

SEVERITY: Informational

PATH:

packages/oapp-evm-upgradeable/contracts/precrime/PreCrimeUpgradeable.sol#L79-L82
packages/oapp-evm-upgradeable/contracts/precrime/PreCrimeUpgradeable.sol#L141-L144

REMEDIATION:

See description.

STATUS: Fixed

DESCRIPTION:

In the contract **PreCrimeUpgradeable.sol**, the implementations of **preCrimePeers()** and **getPreCrimePeers()** are identical, with the only difference being that **preCrimePeers()** is an internal function, while **getPreCrimePeers()** is **external**.

By consolidating these functions into a single **public** function, we can eliminate redundancy and reduce deployment gas costs.

```
function getPreCrimePeers() external view returns (PreCrimePeer[] memory) {
    PreCrimeStorage storage $ = _getPreCrimeStorage();
    return $.preCrimePeers;
}
```

```
function preCrimePeers() internal view returns (PreCrimePeer[] storage) {
    PreCrimeStorage storage $ = _getPreCrimeStorage();
    return $.preCrimePeers;
}
```

Consider removing the function **preCrimePeers()** and changing the visibility of **getPreCrimePeers()** to **public**.

```
- function getPreCrimePeers() external view returns (PreCrimePeer[] memory)
+ function getPreCrimePeers() public view returns (PreCrimePeer[] memory){
    PreCrimeStorage storage $ = _getPreCrimeStorage();
    return $.preCrimePeers;
}

- function preCrimePeers() internal view returns (PreCrimePeer[] storage) {
-     PreCrimeStorage storage $ = _getPreCrimeStorage();
-     return $.preCrimePeers;
- }
```

CONSTANT VARIABLE NAMING SHOULD FOLLOW UPPER_CASE_WITH_UNDERSCORES CONVENTION

SEVERITY: Informational

PATH:

packages/oapp-evm-upgradeable/contracts/oapp/OAppCoreUpgradeable.sol#L18-L19
packages/oapp-evm-upgradeable/contracts/oapp/libs/OAppOptionsType3Upgradeable.sol#L19-L20
packages/oapp-evm-upgradeable/contracts/precrime/OAppPreCrimeSimulatorUpgradeable.sol#L20-L21
packages/oapp-evm-upgradeable/contracts/precrime/PreCrimeUpgradeable.sol#L23-L24
packages/oft-evm-upgradeable/contracts/oft/OFTCoreUpgradeable.sol#L34-L35

REMEDIATION:

Rename these constants to use all uppercase letters with underscores for better alignment with the recommended naming convention.

STATUS: Fixed

DESCRIPTION:

According to the Solidity style guide:

Constants should be named using all uppercase letters with underscores separating words (e.g., MAX_BLOCKS, TOKEN_NAME, TOKEN_TICKER, CONTRACT_VERSION).

Currently, the constants OAppCoreStorageLocation, OAppOptionsType3StorageLocation, etc., use mixed case, which may cause confusion when reading the code.

```
bytes32 private constant OAppCoreStorageLocation =
    0x72ab1bc1039b79dc4724ffca13de82c96834302d3c7e0d4252232d4b2dd8f900;
```

```
bytes32 private constant OAppOptionsType3StorageLocation =
    0x8d2bda5d9f6ffb5796910376005392955773acee5548d0fcdb10e7c264ea0000;
```

```
bytes32 private constant OAppPreCrimeSimulatorStorageLocation =
    0xefb041d771d6daaa55702fff6eb740d63ba559a75d2d1d3e151c78ff2480b600;
```

```
bytes32 private constant PreCrimeStorageLocation =
    0x56af11d8938063b7ae95a808f88d40c589be3bbf4cb6facdbc642a7b38e01f00;
```

```
bytes32 private constant OFTCoreStorageLocation =
    0x41db8a78b0206aba5c54bcbfc2bda0d84082a84eb88e680379a57b9e9f653c00;
```

LZOUPG-6

CONFUSING PLACEMENT OF THE EXPLANATORY COMMENT

SEVERITY: Informational

PATH:

packages/oft-evm-upgradeable/contracts/oft/OFTCoreUpgradeable.sol#L310

REMEDIATION:

Consider modifying line 310 of the contract OFTCoreUpgradeable to:

- endpoint.sendCompose(toAddress, _guid, 0, /* the index of the composed message*/ composeMsg);
+ endpoint.sendCompose(toAddress, _guid, 0 /* the index of the composed message*/, composeMsg);

STATUS: Fixed

DESCRIPTION:

In line 310 of the **OFTCoreUpgradeable._lzReceive()** function, the comment **/* the index of the composed message */** is intended to clarify the purpose of the third input parameter **0**. However, since the comment is positioned after the third comma, it may mistakenly appear to reference the fourth parameter, **composeMsg**, potentially confusing the reader.

```
endpoint.sendCompose(toAddress, _guid, 0, /* the index of the composed message*/ composeMsg);
```

DECIMALS() IS NO ERC20 STANDARD

SEVERITY: Informational

PATH:

packages/oft-evm-upgradeable/contracts/oft/
OFTAdapterUpgradeable.sol#33

REMEDIATION:

Let the token decimals be passed in as a parameter or consider using some function safeDecimals() like this:

https://github.com/boringcrypto/BoringSolidity/blob/c73ed73afa9273fbce93095ef177513191782254/contracts/libraries/BoringERC20.sol#L49-L55

STATUS: Acknowledged

DESCRIPTION:

The **decimals()** function is not a part of the ERC-20 standard, and was added later as an optional extension:

"OPTIONAL - This method can be used to improve usability, but interfaces and other contracts MUST NOT expect these values to be present."

Some ERC20 tokens may not support the **decimals()** function, therefore it's unsafe to cast all tokens to this interface.

```
constructor(
    address _token,
    address _lzEndpoint
) OFTCoreUpgradeable(IERC20Metadata(_token).decimals(), _lzEndpoint) {
    innerToken = IERC20(_token);
}
```


hexens
×
LayerZero.