Moca Security Audit

: Moccaverse - Mocatoken

Apr 17, 2024

Revision 1.1

ChainLight@Theori

Table of Contents

Moca Security Audit 1
Table of Contents 2
Executive Summary 3
Audit Overview 4
- Scope 4
- Code Revision 5
- Severity Categories 5
- Status Categories 6
- Finding Breakdown by Severity 7

Findings 8
- Summary 8
- #1 MOCA-001 _DEPLOYMENT_CHAINID should be set in the constructor 9
- #2 MOCA-002 MocaOFT should have a hard mint cap 11
- #3 MOCA-003 OFT send can succeed in the source chain but fail in the destination chain 13
- #4 MOCA-004 Defensive programming and other minor suggestions 15
- #5 MOCA-005 Conservative configuration (UlnConfig) should be used for LayerZero 17
- #6 MOCA-006 EnforcedOptionParam in SetGasLimitsHome must use remoteChainId 19
- Revision History 21

Executive Summary

Starting on March 14th, 2024, ChainLight of Theori conducted a security audit of Moccaverse's smart contracts, including MocaToken, MocaOFT, and MocaTokenAdapter, for one week. The Mocca token supports multi-chain (i.e. Polygon and Ethereum) through LayerZero to address the deployment strategy where functionalities requiring the Moca token exist primarily on the Polygon network, even though the token is issued on Ethereum. This approach necessitates the use of LayerZero's Omnichain Fungible Token (OFT v2). Therefore, we focused on mitigating the Single Point of Failure (SPOF) scenarios of LayerZero while maintaining the user experience.

The audit focused on several threat/mitigation scenarios listed below:

- Preventing theft of tokens
- Verifying the correct implementation of OFT, EIP-3009
- Addressing discrepancies between Ethereum and Polygon networks
- Mitigating potential Denial of Service (DoS) attacks
- Proper initialization and implications of rate limiting
- The trade-off between security and user experience due to rate limiting features
- Access Control List (ACL) configurations

The audit discovered the scenarios where rate limiting could potentially be abused, as detailed in MOCA-003. However, Moccaverse team has acknowledged to accept slight compromises to the user experience to prioritize security, and has also mentioned that the scenario could occur in a very rare case. Key recommendations to mitigate the risk of a single point of failure (SPOF) associated with LayerZero include the introduction of a hard mint cap (MOCA-002), conservative adjustments to the required number of confirmations and diverse oracles of LayerZero's Decentralized Verifier Networks (MOCA-005). It mitigates the current dependency risks on LayerZero and Google Cloud Platform (GCP) for critical operations.

Audit Overview

Scope

Name	Moca Security Audit
Target / Version	• Git Repository (mocaverse/MocaToken): 461ceebf546e78a628239b6d5831161894f656a9 - 847784ff633c9a3c1dd7940cb763d36bb7003779 ◦ MocaToken, MocaOFT and MocaTokenAdapter
Application Type	Smart contracts
Lang. / Platforms	Smart contracts [Solidity]

Code Revision

N/A

Severity Categories

Severity	Description
Critical	The attack cost is low (not requiring much time or effort to succeed in the actual attack), and the vulnerability causes a high-impact issue. (e.g., Effect on service availability, Attacker taking financial gain)
High	An attacker can succeed in an attack which clearly causes problems in the service's operation. Even when the attack cost is high, the severity of the issue is considered "high" if the impact of the attack is remarkably high.
Medium	An attacker may perform an unintended action in the service, and the action may impact service operation. However, there are some restrictions for the actual attack to succeed.
Low	An attacker can perform an unintended action in the service, but the action does not cause significant impact or the success rate of the attack is remarkably low.
Informational	Any informational findings that do not directly impact the user or the protocol.
Note	Neutral information about the target that is not directly related to the project's safety and security.

Status Categories

Status	Description
Reported	ChainLight reported the issue to the client.
WIP	The client is working on the patch.
Patched	The client fully resolved the issue by patching the root cause.
Mitigated	The client resolved the issue by reducing the risk to an acceptable level by introducing mitigations.
Acknowledged	The client acknowledged the potential risk, but they will resolve it later.
Won't Fix	The client acknowledged the potential risk, but they decided to accept the risk.

Finding Breakdown by Severity

Category	Count	Findings
Critical	**0**	• N/A
High	**0**	• N/A
Medium	**1**	• `MOCA-003`
Low	**1**	• `MOCA-002`
Informational	**4**	• `MOCA-001` • `MOCA-004` • `MOCA-005` • `MOCA-006`
Note	**0**	• N/A

Findings

Summary

#	ID	Title	Severity	Status
1	`MOCA-001`	`_DEPLOYMENT_CHAINID` should be set in the constructor	Informational	Patched
2	`MOCA-002`	`MocaOFT` should have a hard mint cap	Low	Patched
3	`MOCA-003`	OFT send can succeed in the source chain but fail in the destination chain	Medium	Patched
4	`MOCA-004`	Defensive programming and other minor suggestions	Informational	Patched
5	`MOCA-005`	Conservative configuration (`UlnConfig`) should be used for LayerZero	Informational	Acknowledged
6	`MOCA-006`	`EnforcedOptionParam` in `SetGasLimitsHome` must use `remoteChainId`	Informational	Patched

#1 `MOCA-001` `_DEPLOYMENT_CHAINID` should be set in the constructor

ID	Summary	Severity
`MOCA-001`	`constructor()` in `MocaToken` and `MocaOFT` should initialize `_DEPLOYMENT_CHAINID`.	Informational

Description

The `_domainSeparator()` of `MocaToken` and `MocaOFT` are as follows:

```
function _domainSeparator() internal override view returns (bytes32) {
    return block.chainid == _DEPLOYMENT_CHAINID ? _DOMAIN_SEPARATOR : EIP712.makeDomainSeparator(name(), _version);
}
```

If `_DEPLOYMENT_CHAINID` and `block.chainid` differ, it returns the `EIP712.makeDomainSeparator(name(), _version)`. If they are equal, it returns `_DOMAIN_SEPARATOR`, the value initialized (i.e., as a caching purpose) in `constructor()` with `EIP712.makeDomainSeparator(name(), _version)`. This is to recalculate the domain separator on a hard fork because a different `block.chainid` must result in a different domain separator. However, the domain separator will always be recalculated because `_DEPLOYMENT_CHAINID` is not initialized in the `constructor()`.

Impact

Informational

Recommendation

`constructor()` in `MocaToken` and `MocaOFT` should initialize `_DEPLOYMENT_CHAINID` to `block.chainid`.

Remediation

Patched

It is patched as recommended.

#2 `MOCA-002` `MocaOFT` should have a hard mint cap

ID	Summary	Severity
`MOCA-002`	The total supply of `MocaOFT` should be limited.	Low

Description

`MocaOFT` serves as a bridge for `MocaToken` between the Ethereum and Polygon networks, utilizing the LayerZeroV2 (LZ) protocol to facilitate cross-chain transactions. The `MocaToken` has a fixed total supply of `8_888_888_888 * 1e18` when deployed on the Ethereum network, a figure that is immutable due to the contract's non-upgradeable nature. Accordingly, the supply of `MocaOFT` must be capped by the total supply of `MocaToken`, ensuring that it cannot surpass this predefined limit (`8_888_888_888 * 1e18`).

Impact

Low

If LZ is compromised, the attacker can arbitrarily mint the OFT tokens on the destination chain (Polygon).

Recommendation

Override the `_update()` function in the `MocaOFT` contract to ensure that the OFT's total supply does not exceed the L1's fixed total supply (`8_888_888_888 * 1e18`).

When implementing OFT for a non-EVM chain, the minting cap should be `8_888_888_888 * sharedDecimals()`. (where `sharedDecimals()` is 6)

Remediation

Patched

It is patched as recommended.

#3 `MOCA-003` OFT send can succeed in the source chain but fail in the destination chain

ID	Summary	Severity
`MOCA-003`	When a user tries to send `MocaOFT` to another chain, the receiving transaction may fail on the destination chain, freezing the user's token.	Medium

Description

When a user tries to send `MocaOFT` to another chain, the rate limit validation variables, `sentTokenAmounts` and `receivedTokenAmounts` in the `MocaOFT` contracts are incremented on the source chain and the destination chain contracts, respectively. (i.e., In the `MocaOFT` contract of the source chain, `sentTokenAmounts` is incremented, and in the `MocaOFT` of the destination chain, `receivedTokenAmounts` is incremented.) Suppose the `receivedTokenAmounts` in the destination chain is greater than the `sentTokenAmounts` in the source chain. In that case, the sending transaction succeeds in the source chain, but the receiving transaction fails in the destination chain, and the user's tokens are frozen.

The following are three scenarios where the `receivedTokenAmounts` of the destination chain `MocaOFT` is greater than the `sentTokenAmounts` of the source chain `MocaOFT`, and the user's tokens may be frozen.

Scenario I. If a `whitelist` address of the source chain sends tokens to an address that is not on the `whitelist` of the destination chain, the `sentTokenAmounts` of the source chain will not increase, but the `receivedTokenAmounts` of the destination chain `MocaOFT` will increase. If another user sends tokens from the same source chain to the same destination chain before the rate limit is reset, the sending transaction on the source chain succeeds, but the receiving transaction on the destination chain may fail.

Scenario II. If `MocaOFT` is sent to another chain when the rate limit is about to be reset, the sending transaction on the source chain may be executed before the rate limit is reset, and the receiving transaction on the destination chain may be executed after the rate limit is reset. If another user subsequently sends `MocaOFT` from the same source chain to the same destination chain, the rate limit will be reset on the source chain. However, in the destination chain, the rate limit has already been reset, and the `receivedTokenAmounts` have been incremented.

Therefore, the sending transaction may succeed on the source chain, and the receiving transaction may fail on the destination chain due to exceeding the rate limit.

Scenario III. Suppose there is a discrepancy between the chain configurations. For example, the `outboundLimits` of the source chain is set higher than the `inboundLimits` of the destination chain. In that case, the sending transaction from the source chain succeeds, but the receiving transaction from the destination chain may fail because it exceeds the rate limit.

Impact

Medium

If a receiving transaction fails in the destination chain, a user can retry the failed receiving transaction by calling `EndpointV2.lzReceive()` after the rate limit is reset. However, the user has to wait up to a day for the rate limit to be reset, which can be a bad user experience as the token is frozen for up to a day. (Assuming that they are normal/innocent users.)

Recommendation

Owners/operators should respond to the situation where the `receivedTokenAmounts` of the destination chain is greater than the `sentTokenAmounts` of the source chain and the innocent user's tokens are frozen, by implementing a function to reset the `MocaOFT.receivedTokenAmounts` so that the user can immediately retry the receiving transaction on the destination chain.

Remediation

Patched

It is patched as recommended.

#4 `MOCA-004` Defensive programming and other minor suggestions

ID	Summary	Severity
`MOCA-004`	The description includes multiple suggestions for preventing incorrect settings caused by operational mistakes, mitigating potential issues, improving code maturity and readability, and other minor issues.	Informational

Description

1. `setOutboundLimit()`, `setInboundLimit()`: `chainId` is the parameter's name, but the actual value passed is `eid`. `eid` should be used instead of `chainId` to avoid confusion.
   - LZ also suggests the same since `eid` can be more than one for a chain.

2. `_debit()`, `_credit()`: The `sentTokenAmounts` / `receivedTokenAmount` variable names could be interpreted as the total amount of tokens given or received by the other party. We suggest changing these names to `sentTokenAmountsInThisEpoch` / `receivedTokenAmountInThisEpoch` to better express that the rate limit is reset daily.
3. `lastSentTimestamps[dstEid] = currTimestamp;` only needs to be done when the rate limit is reset. Moving this code inside `if ((currTimestamp / (1 days)) > (lastSentTimestamp / (1 days)))` block is recommended to optimize gas by reducing storage access.
4. `_debit()`, `_credit()`: `// If these two values are different, it means that at least one full day has passed since the last transaction.` The comment differs from the actual code. The comment should be revised.

Impact

Informational

Recommendation

Consider applying the suggestions in the description above.

Remediation

Patched

It is patched as recommended.

#5 `MOCA-005` Conservative configuration (`UlnConfig`) should be used for LayerZero

ID	Summary	Severity
MOCA-005	`MocaTokenAdapter` and `MocaOFT` use the default `UlnConfig` without configuring additional DVNs. `UlnConfig` should be customized to reduce the risk from chain reorg and DVN compromises.	Informational

Description

In the message verification phase of LayerZero V2, the integrity of the message is checked by referring to the `UlnConfig` set in the OApp. This value is set to default when not configured. (LayerZero Labs, Google Cloud)

The default configuration values for relevant routes are as follows. (Each route's required confirmation count differs, and `requiredDVNs` is set to 2.)

Ethereum → Polygon

```
confirmations: 15
requiredDVNCount: 2
optionalDVNCount: 0
optionalDVNThreshold: 0
requiredDVNs: [0x589dEDbD617e0CBcB916A9223F4d1300c294236b,0xD56e4eAb23cb81
f43168F9F45211Eb027b9aC7cc]
optionalDVNs: []
```

Polygon → Ethereum

```
confirmations: 512
requiredDVNCount: 2
optionalDVNCount: 0
optionalDVNThreshold: 0
requiredDVNs: [0x23DE2FE932d9043291f870324B74F820e11dc81A,0xD56e4eAb23cb81
f43168F9F45211Eb027b9aC7cc]
optionalDVNs: []
```

Impact

Informational

Recommendation

1. `confirmations` should be set to a higher value considering the typical block finalization time of the home chain. For example, Ethereum → Polygon: `>= 65`, Polygon → Ethereum: `>= 1200`.
2. More DVNs should be added to `requiredDVNs` to reduce the risk from DVN compromises. We recommend DVN operators Nethermind and Animoca-Blockdaemon. (Nethermind's DVN would be resilient due to deployment on multiple GCP availability zones. And, Animoca-Blockdaemon's DVN can serve as a last line of defense by dropping messages since it is affiliated with the Moccaverse project.)

Remediation

Acknowledged

Moccaverse team applied recommendation 2 as is, and recommendation 1 was partially applied due to UX concerns. (Confirmation count for Ethereum is kept to 15 (the default), and 768 (3x of the default) is used for Polygon.)

#6 `MOCA-006` `EnforcedOptionParam` in `SetGasLimitsHome` must use `remoteChainId`

ID	Summary	Severity
`MOCA-006`	In the deploy script (i.e., `Others.s.sol`), `EnforcedOptionParam` of `SetGasLimitsHome` uses wrong variable (`homeChainID`) instead of `remoteChainID`.	Informational

Description

The deploy script (i.e. `Others.s.sol`) configures the gas limit of the external call (`sendAndCall`) on the home and remote chain. However, there is a problem with setting the gas limit to the wrong chain id (i.e. should be fixed as `remoteChainId` instead of `homeChainId`).

Impact

Informational

Recommendation

Change `homeChainID` to `remoteChainId`.

```
- enforcedOptionParams[1] = EnforcedOptionParam(homeChainID, 2, hex"000301
001101000000000000000000000000000f4240");
+ enforcedOptionParams[1] = EnforcedOptionParam(remoteChainId, 2, hex"0003
01001101000000000000000000000000000f4240");
```

Remediation

Patched

It is patched as recommended.

Revision History

Version	Date	Description
1.0	Mar 24, 2024	Initial version
1.1	Apr 17, 2024	Remediation status update, Other minor revisions

ChainLight