Llama Locker

Security Assessment

May 13th, 2024 — Prepared by OtterSec

Nicholas R. Putra nicholas@osec.io

Alpha Toure shxdow@osec.io

Robert Chen notdeghost@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

Llama Locker engaged OtterSec to assess the `LlamaLocker` program. This assessment was conducted between May 7th and May 9th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 3 findings throughout this audit engagement.

In particular, we identified a vulnerability where the unlock function fails to clear the non-fungible token, which may be exploited by an attacker to create a denial of service by repeatedly unlocking the same one (OS-LML-ADV-00). We also identified a potential re-entrancy risk in the claim function, which might be exploited to claim more rewards than intended if the reward token supports a callback feature (OS-LML-ADV-01).

We also recommended the removal of an unutilized function (OS-LML-SUG-00).

Scope

The source code was delivered to us in a Git repository at https://github.com/pyk/LlamaLocker. This audit was performed against commit a0c1394.

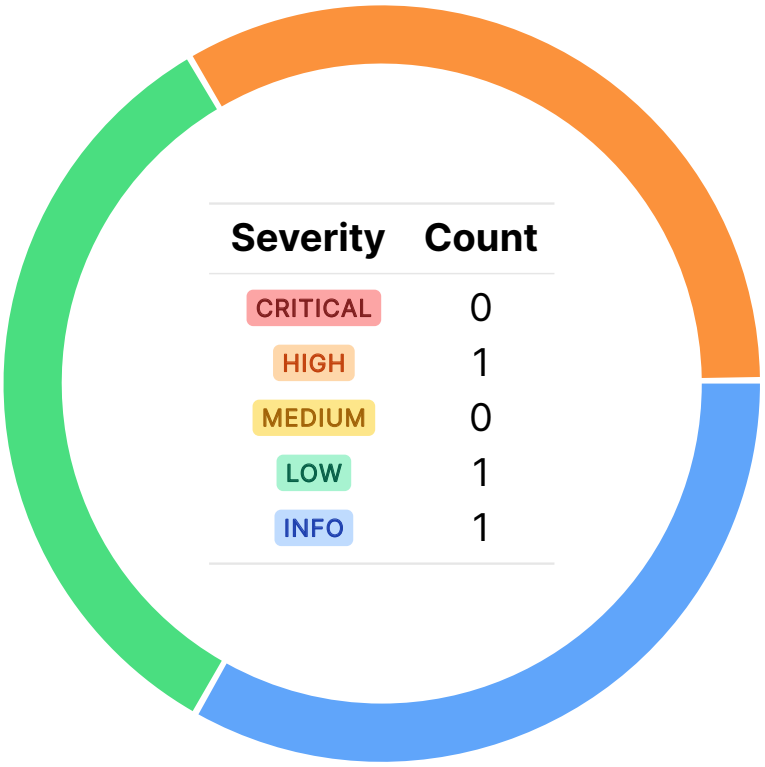
A brief description of the programs is as follows:

Name	Description
LlamaLocker	Provides a mechanism for users to lock their LLAMA tokens and earn a share of the yield generated by the treasury over time. It manages epochs, reward token distribution, and token locking/unlocking, ensuring fair and transparent reward distribution to users.

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-LML-ADV-00	HIGH	RESOLVED	`unlock` does not clear the `locks[tokenId]` value, which attackers may exploit to create a cause a denial of service by repeatedly unlocking the same non-ungible token.
OS-LML-ADV-01	LOW	RESOLVED	The `claim` function does not follow the Checks-Effect-Interaction (CEI) pattern, which attackers may exploit to claim more rewards if the reward token is a special `ERC20` Token that supports callbacks.

Denial Of Service HIGH OS-LML-ADV-00

Description

`LlamaLocker::unlock` fails to clear the `locks[tokenId]` value after successfully unlocking a non fungible token. After a non-fungible token is unlocked, its corresponding entry in the locks mapping (`locks[tokenId]`) should be cleared to reflect that the non-fungible token is no longer locked. However, if this step is omitted, the contract will consider the non-fungible token locked, even after it is unlocked.

>_ *LlamaLocker.sol* solidity

```solidity
 /// @dev Unlock non fungible tokens; it will revert if tokenId have invalid unlock window
function unlock(uint256[] calldata _tokenIds) external {
    uint256 tokenCount = _tokenIds.length;
    if (tokenCount == 0) revert Empty();

    // Backfill epochs
    _backfillEpochs();
    uint256 currentEpochIndex = epochs.length - 1;
    emit DebugCurrentEpochIndex(currentEpochIndex);

    // Decrease total locked non fungible tokens
    totalLockednon fungible tokens -= tokenCount;
    for (uint256 i = 0; i < tokenCount; ++i) {
        _unlock(msg.sender, _tokenIds[i], currentEpochIndex);
    }
}
```

This allows a user to orchestrate a denial of service attack by reducing the `totalLockedNFT` count to zero, even though one non-fungible token (`tokenIdB`) remains locked in the contract, restricting legitimate users from unlocking `tokenIdB` as the contract erroneously believes that all non-fungible tokens are already unlocked.

Proof of Concept

1. Two non-fungible tokens, `tokenIdA` and `tokenIdB` , are locked in the contract, and the `totalLockednon fungible tokens` value is initially set to two, reflecting the total number of locked tokens.
2. The attacker calls `unlock`, passing an array containing the same `tokenIdA` twice: [`tokenIdA`, `tokenIdA`].

3. During the first unlock, `unlock` is called for `tokenIdA`. As part of the unlocking process, the non-fungible token associated with `tokenIdA` is transferred back to the owner (`msg.sender`) utilizing the `safeTransferFrom` function.
4. Since the non-fungible token transfer triggers `onERC721Received` in the contract, the attacker may intercept this call and transfer the non-fungible tokens back to the locker contract manually before the transaction completes.
5. During the second unlock attempt for `tokenIdA` , `unlock` is called again. However, since the `locks[tokenIdA]` value was not cleared after the first unlock, the contract still considers `tokenIdA` as locked. Therefore, it proceeds to transfer the non fungible token back to the owner, unaware that it has already been unlocked.
6. As a result, the `totalLockednon fungible tokens` value decreases from two to zero, and the user is unable to unlock `tokenIdB` due to it.

Remediation

Ensure that the `locks[tokenId]` value is cleared after each successful unlock operation.

Patch

Fixed in 0b2c4d1.

Re-entrancy Attack for Special ERC20 Token LOW OS-LML-ADV-01

Description

The `LlamaLocker::claim` function does not adhere to the Checks-Effects-Interaction (CEI) pattern. Specifically, it attempts to transfer the reward before updating the `claimedRewards` mapping. This sequence can be exploited if the reward token is a specialized `ERC20` token, such as an `ERC777`, which supports callbacks during transfers. In such cases, an attacker could launch a re-entrancy attack, enabling them to claim more rewards than intended.

Remediation

Ensure that the `claimedRewards` value is increased before calling `safeTransfer` and add `nonreentrant` protection to the function.

Patch

Fixed in 0b2c4d1.

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-LML-SUG-00	`_updateAccountReward` is never utilized in the code base.

Unused Function

OS-LML-SUG-00

Description

`_updateAccountReward` is responsible for updating the rewards for a specific account based on the locked NFTs associated with that account and the reward rates for each token. However, it is never utilized in the current code, and no logic is implemented for claiming or distributing rewards to users.

>_ *LlamaLocker.sol* solidity

```
/// @dev Locked NFT cannot be withdrawn for LOCK_DURATION_IN_EPOCH and are eligible to receive
    ↪  share of yields
function lock(LockInput[] calldata _inputs) external {
    uint256 inputCount = _inputs.length;
    if (inputCount == 0) revert Empty();

    _backfillEpochs();
    uint256 currentEpochIndex = epochs.length - 1;
    emit DebugCurrentEpochIndex(currentEpochIndex);

    for (uint8 i = 0; i < inputCount; ++i) {
        _lock(_inputs[i], msg.sender, currentEpochIndex);
    }
}
```

Remediation

Remove the function.

Patch

Fixed in fd4c1ce.

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.