Makina

Security Assessment

October 29th, 2025 — Prepared by OtterSec

Nicholas R. Putra	nicholas@osec.io
Jinwoo Lee	jin@osec.io
Zhenghang Xiao	kiprey@osec.io
Robert Chen	r@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

MakinaHQ engaged OtterSec to assess the `makina-core` and `makina-periphery` programs. This assessment was conducted between September 30th and October 27th, 2025. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 2 findings throughout this audit engagement.

In particular, we made suggestions for improved functionality, ensuring proper configuration and accurate asset validation (OS-MAK-SUG-01). Additionally, we advised implementing input validation checks to prevent the setting of illogical values in the setter functions for the loss parameter and to ensure acceptance of only valid deposit requests (OS-MAK-SUG-00).

Scope

The source code was delivered to us in Git repositories at github.com/MakinaHQ/makina-core and github.com/MakinaHQ/makina-periphery. This audit was performed against commits 0bbdb9e and 7d6ccb3. The final versions of Makina-core and Makina-periphery repositories are c3e1533 and 0ff57e3 respectively.

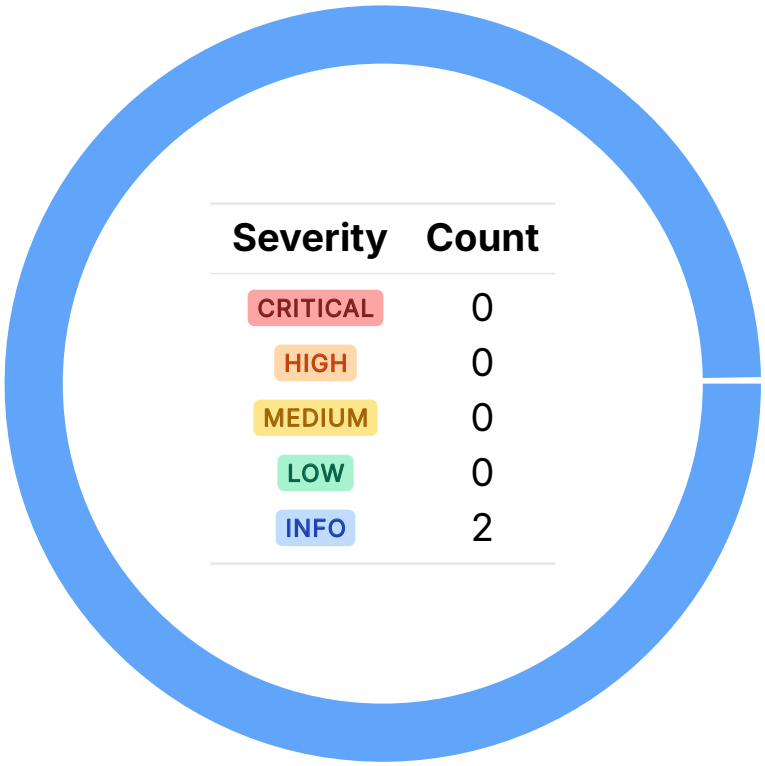
A brief description of the program is as follows:

Name	Description
makina	It enables users to deposit assets into yield-generating Machines, earn returns, and redeem shares asynchronously via NFTs. It supports flashloan-enabled strategies through Calibers, allowing advanced asset management and arbitrage across multiple liquidity providers.

02 — Findings

Overall, we reported 2 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	0
MEDIUM	0
LOW	0
INFO	2

03 — 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-MAK-SUG-00	There are several instances where proper validation is not performed, resulting in potential security issues.
OS-MAK-SUG-01	Recommendation for modifying the codebase for improved functionality, ensuring proper configuration and accurate asset validation.

Missing Validation Logic

OS-MAK-SUG-00

Description

1. Currently `AsyncRedeemer::requestRedeem` allows zero or negligible share amounts, creating meaningless redemption requests. Add a check to reject zero share (or minimum) share amounts to ensure only valid, meaningful requests are created.
2. The setter functions for loss parameters in `Caliber` and `BridgeController` currently allow values exceeding `MAX_BPS`, which is illogical. Add input validation to prevent loss values from exceeding `MAX_BPS`.
3. It is possible for multiple open positions to inadvertently share the same `positionToken`, resulting in conflicts or incorrect valuation. To prevent this, the contract should include an on-chain validation ensuring each `positionToken` is uniquely linked to only one active position.

Remediation

Incorporate the above-mentioned validations into the codebase.

Patch

1. Issue #1 resolved in 41cd7df.
2. Issue #2 was acknowledged by Makina.
3. Issue #3 resolved in c3e1533.

Code Refactoring OS-MAK-SUG-01

Description

1. Add per-bridge cooldowns to limit how frequently a bridge may send outbound transfers. This, combined with a low `maxBridgeLossBps`, prevents rapid successive transfers from creating large cumulative losses.
2. Depositing into `Machine` relies on `_lastTotalAum`, which may be outdated if `_lastGlobalAccountingTime` is stale. This may result in shares being minted based on stale valuations. It will be appropriate to block deposits when `_lastGlobalAccountingTime` is stale to ensure that all share minting is based on up-to-date and accurate asset valuations.
3. Call `_checkFeeSplit` in `WatermarkFeeManager::initialize` to validate the initial management and performance fee splits. This ensures the receivers and split percentages are consistent and sum to `MAX_BPS`, preventing misconfigurations at deployment.

>_ src/fee-managers/WatermarkFeeManager.sol SOLIDITY

```
/// @inheritdoc IMachinePeriphery
function initialize(bytes calldata data) external override initializer {
    WatermarkFeeManagerStorage storage $ = _getWatermarkFeeManagerStorage();
    WatermarkFeeManagerInitParams memory params = abi.decode(data,
        ↪  (WatermarkFeeManagerInitParams));
    if (
            params.initialMgmtFeeRatePerSecond > MAX_FEE_RATE ||
                ↪  params.initialSmFeeRatePerSecond >MAX_FEE_RATE
            || params.initialPerfFeeRate > MAX_FEE_RATE
    ) {
        revert Errors.MaxFeeRateValueExceeded();
    }
    $._mgmtFeeRatePerSecond = params.initialMgmtFeeRatePerSecond;
    $._smFeeRatePerSecond = params.initialSmFeeRatePerSecond;
    $._perfFeeRate = params.initialPerfFeeRate;
    $._mgmtFeeSplitBps = params.initialMgmtFeeSplitBps;
    $._mgmtFeeReceivers = params.initialMgmtFeeReceivers;
    $._perfFeeSplitBps = params.initialPerfFeeSplitBps;
    $._perfFeeReceivers = params.initialPerfFeeReceivers;
}
```

Remediation

Update the codebase with the above refactors.

Patch

1. Issue #1 resolved in bb450bb.
2. Issue #2 was acknowledged by Makina.
3. Issue #3 resolved in 41cd7df.

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 others may have 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.