yAudit Euler Hook Target Firewall Review

Review Resources:

- code repository

Auditors:

- HHK
- Panda

Table of Contents

Review Summary

Euler Hook Target Firewall

Euler Hook Target Firewall contract is an advanced security layer for Euler vaults designed to protect against unauthorized and potentially malicious transactions.

The contracts of the Euler Hook Target Firewall Repo were reviewed over one and a half days. The code review was performed by two auditors between 17 and 18 September 2024. The repository was under active development during the review, but the review was limited to the latest commit, 620f9f036018e11186b1b83114204c1801e1bb70 for the Euler Hook Target Firewall repo, at the start.

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
src/HookTarget/HookTargetFirewall.sol
```

After the findings were presented to the Euler Hook Target Firewall team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Euler Hook Target Firewall and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Average	An issue related to access control was found during the review.
Mathematics	Good	No complex mathematical operations. Simple arithmetic and comparisons are used for threshold checks and amount tracking.
Complexity	Good	Well-structured with clear separation of concerns
Libraries	Good	Relies on external libraries like Solady and OpenZeppelin, which are trusted.
Decentralization	Good	Integrates with off-chain validators while maintaining on-chain decision-making.
Code stability	Good	The code remained stable during the review.
Documentation	Good	The contract is well documented.
Monitoring	Good	Events are properly emitted.
Testing and verification	Low	No tests were available for this contract during the review. The Euler team will port the testsuite once the final version of the Forta protocol is made public. We acknowledged that writing comprehensive tests for contracts with significant off-chain components is challenging.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
    - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas savings
    - Findings that can improve the gas efficiency of the contracts.

- Informational
    - Findings including recommendations and best practices.

Critical Findings

None.

High Findings

None.

Medium Findings

1. Medium - A malicious user can increase `operationCounter` for someone else, resulting in DOS

The `operationCounter` is used to validate a transaction. It will revert if it doesn't match the transaction validated by the Forta endpoint. By increasing another user's `operationCounter`, a malicious actor can block that user from interacting with the vault.

Technical Details

The `HookTargetFirewall` contract implements multiple external functions that anyone can call. The vaults are supposed to call these functions as hooks.

When one these functions is called, they call the internal function `executeCheckpoint()`. The function will first increase the `operationCounter` and `accumulatedAmount` for the user then if the `amount` of the transaction is bigger than the set thresholds for the vault it will trigger a call to the Forta validator.

The `operationCounter` will be passed with other parameters. It is in charge of protecting against replay attacks, as each transaction will have to have been confirmed by the Forta endpoint with a specific `operationCounter` prior to the transaction execution. It additionally enforces that transactions are executed in the order they were approved.

The hook will use the last 20 bytes passed inside the calldata to determine the user. This is how the vaults created with the EVK pass the initial caller to the hook.

The hook doesn't enforce that the address calling it is a registered vault created with the EVK with set thresholds. This allows any user to call it with someone else address as the last 20bytes, increasing the `operationCounter` for that user.

If a user is executing a transaction that requires Forta validation, by changing the user's `operationCounter`, the attacker can invalidate the validation from the Forta endpoint and cause the user's transaction to revert.

POC:

```
// SPDX-License-Identifier: GPL-2.0-or-later

pragma solidity ^0.8.23;

import {EVaultTestBase} from "evk-test/unit/evault/EVaultTestBase.t.sol";
import {HookTargetFirewall} from "../../src/HookTarget/HookTargetFirewall.sol";

contract HookTargetGuardianTests is EVaultTestBase {

    HookTargetFirewall firewallHook;

    function setUp() public virtual override {
        super.setUp();
        firewallHook = new HookTargetFirewall(address(evc), address(0), bytes32(0));
    }

    function test_ImpersonateCall() public {
        address caller = address(123);
        bytes memory call = abi.encodeWithSelector(firewallHook.deposit.selector,
uint256(0), address(0));

        //call
        (bool success, bytes memory data) =
address(firewallHook).call(abi.encodePacked(call, caller));
        (uint256 response) = abi.decode(data, (uint256));
        assertEq(success, true); //call succeeded
        assertEq(response, 0); //call succeeded

        //storage was updated
        uint256 counter = uint256(vm.load(address(firewallHook),
0xa6eef7e35abe7026729641147f7915573c7e97b47efa546f5f6e3230263bcb49));
//operationCounters[caller]
        assertEq(counter, 1);
    }
}
```

Impact

Medium.

Recommendation

Two propositions were made to fix the issue:

- Ensure the address calling is a registered vault by checking if it has `attesters`. However, this was an incomplete fix, as someone could make up a fake vault.
- Set a counter per address using the `msg.sender` calling in the mapping. However, this was found to be too hard to integrate as each transaction would likely call multiple vaults, requiring multiple signatures to be registered prior to the call.

The Euler team chose a slightly different version of the first option, checking the EVK factory to make sure the address calling is a registered proxy.

Developer Response

Fixed in https://github.com/euler-xyz/evk-periphery/pull/95/commits/ea581b62e63ca3551e0a654a90d6c6a4a6c69aef.

Low Findings

None.

Gas Saving Findings

None.

Informational Findings

1. Informational - Weak transaction pool spam protection in HookTargetFirewall

The default protection mechanism in the HookTargetFirewall.sol contract is insufficient to prevent transaction pool spamming. An attacker can bypass the intended limits by submitting multiple transactions under the threshold.

Technical Details

The current implementation of transaction limits in the `HookTargetFirewall.sol` contract has a vulnerability that allows potential exploitation through transaction pool spamming. By submitting multiple transactions under the defined limits, an attacker can ensure their transactions are processed, potentially manipulating the system.

```
File: HookTargetFirewall.sol
350:     function updateAccumulatedAmount(TransferType transferType, address vault, uint256 referenceAmount)
351:         internal
352:         returns (uint256)
353:     {
354:         StorageSlot.Uint256SlotType slot = StorageSlot.asUint256(keccak256(abi.encode(transferType, vault)));
355:         uint256 accumulatedAmount = StorageSlot.tload(slot) + referenceAmount;
356:         StorageSlot.tstore(slot, accumulatedAmount);
357:         return accumulatedAmount;
358:     }
```

HookTargetFirewall.sol#L350-L358

Impact

Informational

Recommendation

Add a secondary protection based on the number of transactions.

- **Fixed limit:** Restrict the number of transactions to X over a Y-block window.2.
- **Exponential backoff:** Implement an exponential backoff mechanism where the thresholds become more strict as more transactions occur within the window.

Developer Response

Fixed in https://github.com/euler-xyz/evk-periphery/pull/95/commits/ea581b62e63ca3551e0a654a90d6c6a4a6c69aef.

2. Informational - Uncertainty over Forta FORTRESS protocol core functioning

Technical Details

At the time of review, the Forta protocol is ongoing a security review and has its main components private. The core functioning of the API endpoint and validator contracts are still unclear.

The current implementation may require changes, tests, and additional reviews in the near future.

Impact

Informational.

Recommendation

Ensure that the implementation matches the final architecture of the Forta FORTRESS protocol before deploying to production.

Developer Response

Acknowledged. We agree with the assessment presented.

Final remarks

This new hook from Euler brings off-chain monitoring on-chain through the FORTA FORTRESS protocol. The codebase is well-structured and efficient. However, auditors raised concerns about the Forta integration, as some components were not publicly available during the review, making it difficult to assess the integration fully.