// Smart Contract Security Assessment

11.04.2024 - 11.19.2024

Voucher Contracts

iExec

HALBORN

Report a Bug

Voucher Contracts - iExec

Prepared by: HALBORN Last Updated 01/27/2025 Date of Engagement by: November 4th, 2024 - November 19th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
4	0	0	0	0	4

1. Introduction

iExec engaged Halborn to conduct a security assessment on their smart contracts beginning on November 4th and ending on November 19th, 2024. The security assessment was scoped to the smart contracts provided to the Halborn team.

Commit hashes and further details can be found in the Scope section of this report.

TABLE OF CONTENTS

2. Assessment Summary

The team at Halborn assigned a full-time security engineer to assess the security of the smart contracts. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were partially addressed by the iExec team :

- Disincentivize users to user low trust levels.
- Update voucher properties to all existing vouchers.
- Tidy the codebase to remove duplicated code.
- Optimize the failedWork logic by removing unnecessary transfers.

3. Test Approach And Methodology

Halborn performed a combination of manual review of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the assessment:

- Research into the architecture, purpose, and use of the platform.
- Smart contract manual code review and walkthrough to identify any logic issue.
- Thorough assessment of safety and usage of critical Solidity variables and functions in scope that could lead to arithmetic related vulnerabilities.
- Manual testing by custom scripts.
- Graphing out functionality and contract logic/connectivity/functions (`solgraph`).
- Static Analysis of security for scoped contract, and imported functions. (`Slither`).
- Local or public testnet deployment (`Foundry`, `Remix IDE`).

The PoCo assessment also emphasized in the stakes lifecycle, and the machine state of tasks and contributions.

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: iexec-voucher-contracts

(b) Assessed Commit ID: ef4b1cb

(c) Items in scope:

- iexec-voucher-contracts/IVoucher.sol
- iexec-voucher-contracts/VoucherProxy.sol
- iexec-voucher-contracts/Voucher.sol
- iexec-voucher-contracts/IVoucherHub.sol
- iexec-voucher-contracts/NonTransferableERC20Upgradeable.sol
- iexec-voucher-contracts/VoucherHub.sol

Out-of-Scope: Third party dependencies and economic attacks.

FILES AND REPOSITORY

(a) Repository: PoCo

(b) Assessed Commit ID: 4821374

(c) Items in scope:

- PoCo/contracts/libs/IexecLibCore_v5.sol
- PoCo/contracts/modules/delegates/IexecPoco1Delegate.sol
- PoCo/contracts/modules/delegates/IexecPoco2Delegate.sol
- PoCo/contracts/modules/delegates/IexecPocoAccessorsDelegate.sol
- PoCo/contracts/modules/delegates/IexecPocoBoostDelegate.sol

- PoCo/contracts/modules/delegates/IexecPocoCommonDelegate.sol

Out-of-Scope: Third party dependencies and economic attacks.

REMEDIATION COMMIT ID:

- https://github.com/iExecBlockchainComputing/PoCo/pull/168/commits/7b3cba56384e4535a23f3d3f64a6e2a9660fe4c3
- https://github.com/iExecBlockchainComputing/PoCo/pull/167/commits/0ac30f0d9a6fc0ee71d0fd61bfa9b6a3fed2228a

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	0

INFORMATIONAL
4

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
TRUST MINIMUM ALLOWS FOR IMMEDIATE CONSENSUS	INFORMATIONAL	ACKNOWLEDGED - 01/02/2025
UPDATING VOUCHER TYPES DOES NOT APPLY TO ALL EXITING VOUCHERS INSTANTLY	INFORMATIONAL	ACKNOWLEDGED - 01/02/2025
REUSED CODE	INFORMATIONAL	SOLVED - 12/19/2024
UNECESSARY TRANSFER IN FAILEDWORK CAN BE OPTIMIZED	INFORMATIONAL	SOLVED - 12/17/2024

7. FINDINGS & TECH DETAILS

7.1 TRUST MINIMUM ALLOWS FOR IMMEDIATE CONSENSUS

// INFORMATIONAL

Description

The consensus behind the proof of contribution is based on multiple contributions to increase the trust in a computation. It was found that it was possible to ask for a task with a trust level of 1, which leads to a direct acceptation of the task result, without needing consensus to validate it.

This would allow a worker, which is priorly authorized by the worker pool owner by signature, to return any dishonest result and be accepted by the Poco2 contract.

This condition is also made possible via the `contributeAndFinalize` function, that only works for trusts of 1 and immediately reaches consensus with only one submission, as well as the whole `IexecPocoBoostDelegate` logic that facilitates such a behavior.

This issue is not a vulnerability in itself but allows users to choose low trust scenarios without restriction.

Code Location

In the `IexecPoco2Delegate` contract:

- The `checkConsensus` function guard is easily passed when `trust` is 1:

```
if (consensus.group[_consensus] * trust > consensus.total * (trust - 1)) {
    // Preliminary checks done in "contribute()"
    uint256 winnerCounter = 0;
```

```
    for (uint256 i = 0; i < task.contributors.length; ++i) {
        address w = task.contributors[i];
        if (
            m_contributions[_taskid][w].resultHash == _consensus &&
            m_contributions[_taskid][w].status ==
            IexecLibCore_v5.ContributionStatusEnum.CONTRIBUTED // REJECTED contri
        ) {
            winnerCounter = winnerCounter + 1;
        }
    }
    // _msgSender() is a contributor: no need to check
    task.status = IexecLibCore_v5.TaskStatusEnum.REVEALING;
    task.consensusValue = _consensus;
    task.revealDeadline = block.timestamp + task.timeref * REVEAL_DEADLINE_RATIO;
    task.revealCounter = 0;
    task.winnerCounter = winnerCounter;

    emit TaskConsensus(_taskid, _consensus);
}
```

- The `contributeAndFinalize` function reaches consensus for trusts of 1:

```
function contributeAndFinalize(
    bytes32 _taskid,
    bytes32 _resultDigest,
    bytes calldata _results,
    bytes calldata _resultsCallback, // Expansion - result separation
    address _enclaveChallenge,
    bytes calldata _enclaveSign,
    bytes calldata _authorizationSign
) external override {
```

```
IexecLibCore_v5.Task storage task = m_tasks[_taskid];
IexecLibCore_v5.Contribution storage contribution = m_contributions[_taskid][
IexecLibCore_v5.Deal memory deal = m_deals[task.dealid];

require(task.status == IexecLibCore_v5.TaskStatusEnum.ACTIVE);
require(task.contributionDeadline > block.timestamp);
require(task.contributors.length == 0);
require(deal.trust == 1); // TODO / FUTURE FEATURE: consider sender's score ?

bytes32 resultHash = keccak256(abi.encodePacked(_taskid, _resultDigest));
bytes32 resultSeal = keccak256(abi.encodePacked(_msgSender(), _taskid, _resul
```

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:H/D:N/Y:N (1.5)

Recommendation

It is recommended to disincentivize users to user low trust levels.

Remediation

ACKNOWLEDGED: The **iExec team** acknowledged this finding.

7.2 UPDATING VOUCHER TYPES DOES NOT APPLY TO ALL EXITING VOUCHERS INSTANTLY

// INFORMATIONAL

Description

Voucher properties can be updated by the MANAGER_ROLE owner, such as in updateVoucherTypeDuration. It was found that updating the voucher duration did not apply instantly to all active vouchers. This can cause unwanted behavior if the manager believes that the duration will apply to every voucher, for example setting a longer expiring time but seeing vouchers expiring before expectations.

Code Location

The updateVoucherTypeDuration only updates the duration setting and not the active vouchers:

```
function updateVoucherTypeDuration(
    uint256 id,
    uint256 duration
) external onlyRole(MANAGER_ROLE) whenVoucherTypeExists(id) { // ok checks index
    VoucherHubStorage storage $ = _getVoucherHubStorage();
    $._voucherTypes[id].duration = duration;
    emit VoucherTypeDurationUpdated(id, duration);
}
```

Instead, the modification will be applied only when calling topUpVoucher:

```
function topUpVoucher(address voucher, uint256 value) external onlyRole(MINTER_RO
    require(value > 0, "VoucherHub: no value");
    VoucherHubStorage storage $ = _getVoucherHubStorage();
```

```
    require($._isVoucher[voucher], "VoucherHub: unknown voucher");
    _mint(voucher, value); // VCHR
    _transferFundsToVoucherOnPoco(voucher, value); // SRLC
    uint256 expiration = block.timestamp + $._voucherTypes[Voucher(voucher).getTy
    Voucher(voucher).setExpiration(expiration);
    emit VoucherToppedUp(voucher, expiration, value);
}
```

Score

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

It is recommended to review the intended use of `updateVoucherTypeDuration` and assess whether all existing vouchers should be affected by the change.

Remediation

ACKNOWLEDGED: The **iExec team** acknowledged this finding, also mentioning that:

> The manager of vouchers wants to give guaranties about expiration of a voucher. When a voucher is delivered, credits inside the voucher should not expire before original expiration

7.3 REUSED CODE

// INFORMATIONAL

Description

A state variable in `IexecPocoDelegate.sol` is being recalculated even though a specifically precomputed local variable is available for its value. Although this does not present a security risk, it could lead to potential bugs in future updates. Specifically, if the `group` variable is modified under the assumption that it directly affects `consensus.group`, unintended behavior may occur.

Code Location

- In `IexecPocoDelegate.sol`:

```
IexecLibCore_v5.Consensus storage consensus = m_consensus[_taskid];

uint256 weight = Math.max(m_workerScores[_msgSender()] / 3, 3) - 1;
uint256 group = consensus.group[_resultHash];
uint256 delta = Math.max(group, 1) * weight - group;

contribution.weight = Math.log2(weight);
consensus.group[_resultHash] = consensus.group[_resultHash] + delta;
consensus.total = consensus.total + delta;
```

Score

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

It is recommended to use the precomputed local `group` variable directly in the assignment for `consensus.group[_resultHash]` instead of recalculating its value.

Remediation

SOLVED: The issue was fixed in commit `7b3cba5` by reusing the precomputed `group` variable.

Remediation Hash

https://github.com/iExecBlockchainComputing/PoCo/pull/168/commits/7b3cba56384e4535a23f3d3f64a6e2a9660fe4c3

7.4 UNECESSARY TRANSFER IN FAILEDWORK CAN BE OPTIMIZED

// INFORMATIONAL

Description

The `failedWork` function seizes the stake from the worker pool owner and transfers that amount to a special address (named `kitty`). It was found that the function transfers from `IexecPoco2Delegate` to `kitty`, then transfers from `kitty` to `IexecPoco2Delegate` the same amount before accounting the balance in the `m_frozen` state variable.

It can be optimized by removing the transfers and only recording the state modification in the `m_frozen` balances.

Code Location

- In `IexecPoco2Delegate`:

```
function failedWork(bytes32 _dealid, bytes32 _taskid) internal {
    IexecLibCore_v5.Deal memory deal = m_deals[_dealid];

    uint256 taskPrice = deal.app.price + deal.dataset.price + deal.workerpool.pri
    uint256 poolstake = (deal.workerpool.price * WORKERPOOL_STAKE_RATIO) / 100;

    unlock(deal.sponsor, taskPrice); // Refund the payer of the task
    seize(deal.workerpool.owner, poolstake, _taskid);
    reward(KITTY_ADDRESS, poolstake, _taskid); // → Kitty / Burn
    lock(KITTY_ADDRESS, poolstake); // → Kitty / Burn
}
```

Score

[AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N](AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N) (0.0)

Recommendation

It is recommended to remove the unnecessary transfers of the `failedWork` function.

Remediation

SOLVED: The issue was fixed in commit 0ac30f0 by removing unnecessary transfers.

Remediation Hash

https://github.com/iExecBlockchainComputing/PoCo/pull/167/commits/0ac30f0d9a6fc0ee71d0fd61bfa9b6a3fed2228a

8. AUTOMATED TESTING

Static Analysis Report

Description

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

The security team assessed all findings identified by the Slither software and everything was categorised as false positives.

Results

Voucher Contracts

- `VoucherHub.sol`:

```
'solc --version' running
'solc @=node_modules/@ ./contracts/VoucherHub.sol --combined-json abi,ast,bin,bin-runtime,srcmap,srcmap-runtime,u
serdoc,devdoc,hashes --optimize --via-ir --allow-paths .,/Users/alexisfabre/Desktop/Halborn/20240511-IExec-Vouche
r/iexec-voucher-contracts/contracts' running
INFO:Slither:./contracts/VoucherHub.sol analyzed (39 contracts with 58 detectors), 0 result(s) found
```

- `Voucher.sol`:

```
'solc --version' running
'solc @=node_modules/@ ./contracts/beacon/Voucher.sol --combined-json abi,ast,bin,bin-runtime,srcmap,srcmap-runti
me,userdoc,devdoc,hashes --optimize --via-ir --allow-paths .,/Users/alexisfabre/Desktop/Halborn/20240511-IExec-Vo
ucher/iexec-voucher-contracts/contracts/beacon' running
INFO:Slither:./contracts/beacon/Voucher.sol analyzed (12 contracts with 58 detectors), 0 result(s) found
```

PoCo Contracts

- `IexecPoco1Delegate.sol`:

```
INFO:Detectors:
Store.m_appregistry (contracts/Store.v8.sol#56) is never initialized. It is used in:
	- IexecPoco1Delegate._matchOrders(IexecLibOrders_v5.AppOrder,IexecLibOrders_v5.DatasetOrder,IexecLibOrder
s_v5.WorkerpoolOrder,IexecLibOrders_v5.RequestOrder,address) (contracts/modules/delegates/IexecPoco1Delegate.sol#
141-388)
Store.m_datasetregistry (contracts/Store.v8.sol#58) is never initialized. It is used in:
	- IexecPoco1Delegate._matchOrders(IexecLibOrders_v5.AppOrder,IexecLibOrders_v5.DatasetOrder,IexecLibOrder
s_v5.WorkerpoolOrder,IexecLibOrders_v5.RequestOrder,address) (contracts/modules/delegates/IexecPoco1Delegate.sol#
141-388)
Store.m_workerpoolregistry (contracts/Store.v8.sol#60) is never initialized. It is used in:
	- IexecPoco1Delegate._matchOrders(IexecLibOrders_v5.AppOrder,IexecLibOrders_v5.DatasetOrder,IexecLibOrder
s_v5.WorkerpoolOrder,IexecLibOrders_v5.RequestOrder,address) (contracts/modules/delegates/IexecPoco1Delegate.sol#
141-388)
Store.EIP712DOMAIN_SEPARATOR (contracts/Store.v8.sol#116) is never initialized. It is used in:
	- SignatureVerifier._toTypedDataHash(bytes32) (contracts/modules/delegates/SignatureVerifier.v8.sol#19-21
)
Store.m_presigned (contracts/Store.v8.sol#124) is never initialized. It is used in:
	- SignatureVerifier._verifyPresignature(address,bytes32) (contracts/modules/delegates/SignatureVerifier.v
8.sol#88-93)
Store.m_categories (contracts/Store.v8.sol#161) is never initialized. It is used in:
	- IexecPoco1Delegate._matchOrders(IexecLibOrders_v5.AppOrder,IexecLibOrders_v5.DatasetOrder,IexecLibOrder
s_v5.WorkerpoolOrder,IexecLibOrders_v5.RequestOrder,address) (contracts/modules/delegates/IexecPoco1Delegate.sol#
141-388)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables
INFO:Detectors:
```

- `IexecPoco2Delegate.sol`:

```
Store.EIP712DOMAIN_SEPARATOR (contracts/Store.v8.sol#116) is never initialized. It is used in:
        - SignatureVerifier._toTypedDataHash(bytes32) (contracts/modules/delegates/SignatureVerifier.v8.sol#19-21
)
Store.m_presigned (contracts/Store.v8.sol#124) is never initialized. It is used in:
        - SignatureVerifier._verifyPresignature(address,bytes32) (contracts/modules/delegates/SignatureVerifier.v
8.sol#88-93)
Store.m_deals (contracts/Store.v8.sol#135) is never initialized. It is used in:
        - IexecPoco2Delegate.successWork(bytes32,bytes32) (contracts/modules/delegates/IexecPoco2Delegate.sol#26-
54)
        - IexecPoco2Delegate.failedWork(bytes32,bytes32) (contracts/modules/delegates/IexecPoco2Delegate.sol#56-6
6)
        - IexecPoco2Delegate.initialize(bytes32,uint256) (contracts/modules/delegates/IexecPoco2Delegate.sol#71-9
4)
        - IexecPoco2Delegate.contribute(bytes32,bytes32,bytes32,address,bytes,bytes) (contracts/modules/delegates
/IexecPoco2Delegate.sol#97-169)
        - IexecPoco2Delegate.contributeAndFinalize(bytes32,bytes32,bytes,bytes,address,bytes,bytes) (contracts/mo
dules/delegates/IexecPoco2Delegate.sol#172-246)
        - IexecPoco2Delegate.finalize(bytes32,bytes,bytes) (contracts/modules/delegates/IexecPoco2Delegate.sol#29
5-331)
        - IexecPoco2Delegate.claim(bytes32) (contracts/modules/delegates/IexecPoco2Delegate.sol#333-354)
        - IexecPoco2Delegate.checkConsensus(bytes32,bytes32) (contracts/modules/delegates/IexecPoco2Delegate.sol#
362-396)
        - IexecPoco2Delegate.distributeRewards(bytes32) (contracts/modules/delegates/IexecPoco2Delegate.sol#401-4
74)
        - IexecPoco2Delegate.distributeRewardsFast(bytes32) (contracts/modules/delegates/IexecPoco2Delegate.sol#4
79-490)
        - IexecPoco2Delegate.executeCallback(bytes32,bytes) (contracts/modules/delegates/IexecPoco2Delegate.sol#4
95-526)
Store.m_teebroker (contracts/Store.v8.sol#149) is never initialized. It is used in:
        - IexecPoco2Delegate.contribute(bytes32,bytes32,bytes32,address,bytes,bytes) (contracts/modules/delegates
/IexecPoco2Delegate.sol#97-169)
        - IexecPoco2Delegate.contributeAndFinalize(bytes32,bytes32,bytes,bytes,address,bytes,bytes) (contracts/mo
dules/delegates/IexecPoco2Delegate.sol#172-246)
Store.m_callbackgas (contracts/Store.v8.sol#156) is never initialized. It is used in:
        - IexecPoco2Delegate.executeCallback(bytes32,bytes) (contracts/modules/delegates/IexecPoco2Delegate.sol#4
95-526)
Store.m_categories (contracts/Store.v8.sol#161) is never initialized. It is used in:
        - IexecPoco2Delegate.initialize(bytes32,uint256) (contracts/modules/delegates/IexecPoco2Delegate.sol#71-9
4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables
```

The same issue was detected on the `IexecPocoAccessorsDelegate.sol`, `IexecPocoBoostDelegate.sol`, `IexecPocoCommonDelegate.sol`.

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project’s integrity and addressing potential vulnerabilities introduced by code modifications.