# Switchboard EVM

# Audit

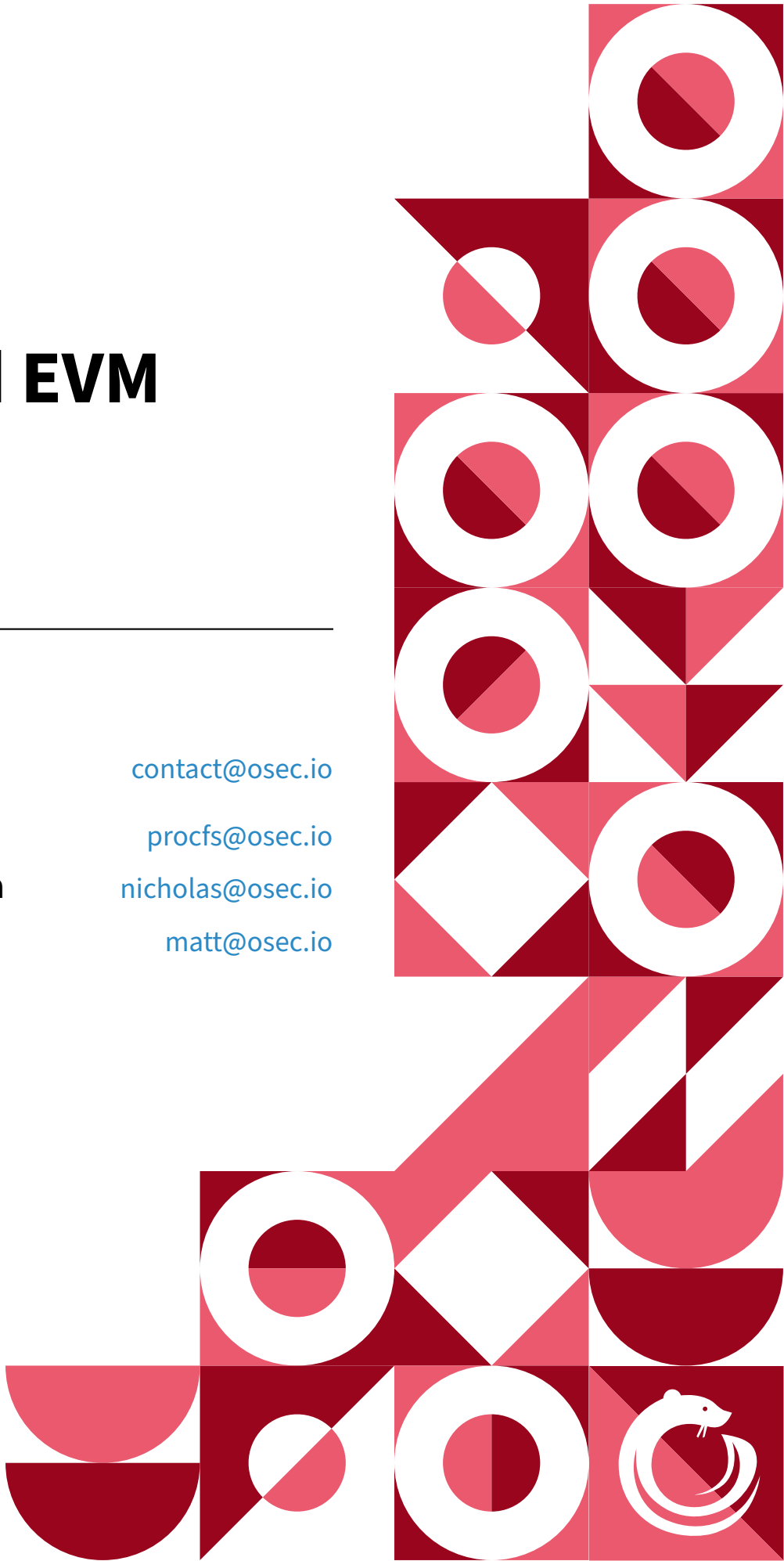

Presented by:

**OtterSec** contact@osec.io

**Woosun Song** procfs@osec.io

**Nicholas Putra** nicholas@osec.io

**Matteo Oliva** matt@osec.io

# Contents


**01 Executive Summary** **2**

Overview . . . . . 2

Key Findings . . . . . 2

**02 Scope** **3**

**03 Findings** **4**

**04 Vulnerabilities** **5**

OS-SWB-ADV-00 [crit] | Reward Manipulation . . . . . 6

OS-SWB-ADV-01 [crit] | Unchecked Fund Provision . . . . . 8

OS-SWB-ADV-02 [crit] | Incorrect Oracle Garbage Collection . . . . . 9

OS-SWB-ADV-03 [high] | Unbounded Gas Rebate . . . . . 11

OS-SWB-ADV-04 [low] | Neglected Balance Update . . . . . 12

OS-SWB-ADV-05 [low] | Incorrect Transaction Expiration Checks . . . . . 14

OS-SWB-ADV-06 [low] | Incorrect Reentrancy Guard Implementation . . . . . 15

OS-SWB-ADV-07 [low] | Incorrect Array Removal Logic . . . . . 17

**05 General Findings** **18**

OS-SWB-SUG-00 | Potentially Dangerous Self Call Pattern . . . . . 19

OS-SWB-SUG-01 | Constant Expiry Threshold . . . . . 21

OS-SWB-SUG-02 | Gas Optimization . . . . . 22


## Appendices


**A Vulnerability Rating Scale** **23**

**B Procedure** **24**

# 01 | Executive Summary

## Overview

Switchboard engaged OtterSec to perform an assessment of the `switchboard-evm` program. This assessment was conducted between June 16th and June 30th, 2023. For more information on our auditing methodology, see Appendix B.

## Key Findings

Over the course of this audit engagement, we produced 11 findings in total.

In particular, we discovered vulnerabilities that allow an unlimited amount of funds to be stolen (OS-SWB-ADV-00, OS-SWB-ADV-01) alongside a vulnerability that allows for a bounded amount of funds to be stolen under an adversarial block builder assumption (OS-SWB-ADV-03). We also discovered an incorrect garbage collection implementation (OS-SWB-ADV-02). These vulnerabilities may compromise critical protocol assumptions.

Additionally, we advised removing a potentially unsafe delegate call which may escalate into a security threat with future code additions (OS-SWB-SUG-00). We also made recommendations around making a constant configurable (OS-SWB-SUG-01), removing unnecessary modifiers (**??**), and gas optimization (OS-SWB-SUG-02).

# 02 | Scope

The source code was delivered to us in a git repository at github.com/switchboard-xyz/switchboard-evm. This audit was performed against commit 8c4f247.

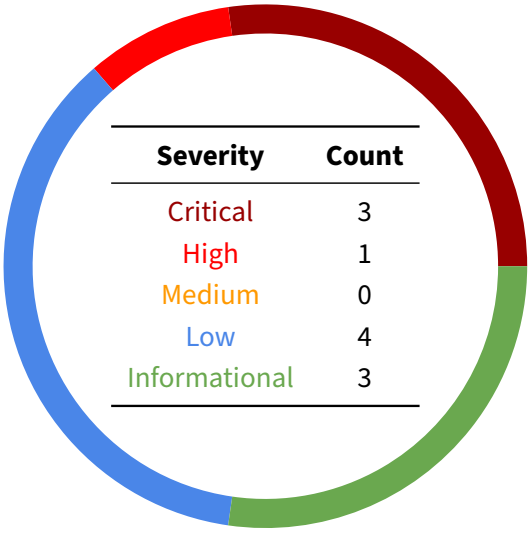
A brief description of the programs is as follows.

| Name | Description |
| --- | --- |
| switchboard-evm | A smart contract that supports a trust-minimized, decentralized oracle network. Its core components are:<br>• Oracle: Posts off-chain updates in a round-robin fashion, which periodically provides a heartbeat to signal readiness.<br>• AttestationQueue: Verifies attestation quotes on-chain, granting or revoking oracle permissions based on enclave contents. |

# 03 | Findings

Overall, we reported 11 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 help mitigate future vulnerabilities.



| Severity | Count |
|---|---|
| Critical | 3 |
| High | 1 |
| Medium | 0 |
| Low | 4 |
| Informational | 3 |

# 04 | 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-SWB-ADV-00 | Critical | Resolved | An attacker may steal funds by increasing rewards after quote updates. |
| OS-SWB-ADV-01 | Critical | Resolved | `SbFunction.forward` has no checks on `msg.value`, allowing unlimited theft of funds. |
| OS-SWB-ADV-02 | Critical | Resolved | Garbage collection occurs on valid oracles. |
| OS-SWB-ADV-03 | High | Resolved | An attacker may profit by utilizing unbounded gas rebates. |
| OS-SWB-ADV-04 | Low | Resolved | In `openInterval`, only the interval balance gets updated while the aggregator balance remains unmodified. |
| OS-SWB-ADV-05 | Low | Resolved | The condition for reverting in `runTransactions` and `forward` is incorrect. |
| OS-SWB-ADV-06 | Low | Resolved | The `guarded` modifier implemented in `Receiver` allows reentrant calls. |
| OS-SWB-ADV-07 | Low | Resolved | `removeMrEnclaveToOracleQueue` performs removals even when the specified `mrEnclave` is not present in the array. |

## OS-SWB-ADV-00 [crit] | Reward Manipulation

### Description

An attacker may steal funds from the diamond by increasing rewards after updating an enclave before using it for validation. The reward is taken in `updateEnclave`, which posts the enclave's `cid`.

*contracts/src/enclave/Enclave.sol* SOLIDITY

```
function updateEnclave(
    address enclaveId, // enclave id
    bytes calldata cid
) external payable guarded(1) {
   /* ... */

    // we pay the verifier queue's reward to verify the enclave
    uint256 verifierReward = queue.reward;
    if (msg.value < verifierReward) {
        revert ErrorLib.InsufficientBalance();
    }

    // Add balance to enclave
    EnclaveLib.fundEnclave(enclaveId, msg.value);

    /* ... */
}
```

The reward provided in `updateEnclave` is distributed upon failing or verifying an enclave. For instance, in `failEnclave`, a verifier enclave marks the subject enclave as failed and gives rewards to the verifier enclave's owner.

*contracts/src/enclave/Enclave.sol* SOLIDITY

```
function failEnclave(
    address verifierId, // caller's enclave address
    address enclaveId, // enclave address to be verified
    uint256 enclaveIdx
) external guarded(1) {
    /* ... */

    // write enclave failure
    EnclaveLib.setEnclaveFailure(enclaveId);

    // do payout
    if (queue.reward > 0) {
        emit EnclavePayoutEvent(verifierId, enclaveId, queue.reward);
        payable(msg.sender).transfer(queue.reward);
    }
}
```

The vulnerability arises from the possibility of increasing the reward amount between reward provision and reward distribution.

### Proof of Concept

For the proof of concept, consider a scenario where an attacker executes the following steps:

1. The attacker creates an `AttestationQueue` with zero rewards.
2. The attacker creates an enclave and invokes `updateEnclave` and `forceOverrideVerify`, making it available as a verifier enclave. Because the attestation queue reward is zero, no funds are required to call `updateEnclave`.
3. The attacker increases the reward amount by calling `setAttestationQueueConfig`. The new reward amount is equal to the Ether balance of the diamond.
4. The attacker utilizes the verifier enclave to call `failEnclave` on itself. The attacker, who owns the verifier enclave, will be able to withdraw the `queue.reward` amount from the diamond, equal to its entire Ether balance.

This method allows the attacker to extract Ether from the contract.

### Remediation

Prevent reward payouts in `failEnclave` and `validateEnclave` if the reward amount is larger than the enclave balance.

### Patch

Fixed in a58f237 and a937a28.

## OS-SWB-ADV-01 [crit] | Unchecked Fund Provision

### Description

`SbFunction.forward` allows external calls with user-provided `msg.values` without checking if sufficient funds are provided. An attacker may steal all Ether from the contract by forging a transaction with a `value` equal to the diamond balance.

*contracts/src/Switchboard/sbFunction/SbFunction.sol* SOLIDITY

```
function forward(
    TransactionLib.Transaction[] memory transactions,
    bytes[] memory signatures
) external guarded(257) {
    // Just run the functions and verify the signatures
    for (uint256 i = 0; i < transactions.length; i++) {
        TransactionLib.Transaction memory transaction = transactions[i];

        /* ... */

        // EIP2771: call the function with the encoded data and the from address
        (bool success, bytes memory returnData) = transaction.to.call{
            value: transaction.value
        }(abi.encodePacked(transaction.data, transaction.from));
    }
}
```

### Remediation

Disallow transfer of funds through `forward` by setting the call value to zero.

### Patch

Fixed in e833e48.

## OS-SWB-ADV-02 [crit] | Incorrect Oracle Garbage Collection

### Description

`oracleHeartbeat` performs garbage collection on a valid oracle instead of an expired one. Each oracle has a field named numRows, which indicates the number of queues associated with it. An oracle may only be added to a queue if its numRows value is zero. Therefore, setting numRows of an oracle back to zero should always be accompanied by queue removal to prevent the oracle from existing in multiple queues simultaneously.

However, `oracleHeartbeat` erroneously performs `setNumRows` on `oracleId` instead of `gcOracleId` during garbage collection.

*contracts/src/Switchboard/oracle/Oracle.sol* SOLIDITY

```
function oracleHeartbeat(address oracleId) external {
    /* ... */

    // get gcIdx - guaranteed to have at least 1 element here
    uint256 gcIdx = queue.gcIdx;
    address gcOracleId = queue.oracles[gcIdx];

    // increment gcIdx
    OracleQueueLib.incrementGC(oracle.queueId);

    // handle expired oracles if gcIdx is expired
    if (
        (OracleLib.oracles(gcOracleId).lastHeartbeat +
            queue.oracleTimeout) < block.timestamp
    ) {
        // log the garbage collection
        emit OracleGC(gcOracleId, oracle.queueId);

        // swap remove queue.oracles[gcIdx]
        OracleLib.setNumRows(oracleId, 0);
        OracleQueueLib.swapRemove(oracle.queueId, gcIdx);
    }
}
```

By exploiting this vulnerability, an attacker may create an oracle that exists in two queues simultaneously, violating crucial assumptions made by other parts of the code. For example, `saveResults` assumes that an oracle exists in only one queue and that an oracle's effective heartbeat indicates it is not expired. However, if an oracle is present in multiple queues, it may continue to perform heartbeats even when it is already expired.

### Remediation

Change `setNumRows(oracleId, 0)` to `setNumRows(gcOracleId, 0)`.

### Patch

Fixed in e833e48.

## OS-SWB-ADV-03 [high] | Unbounded Gas Rebate

### Description

Block builders may exploit gas rebates, as they do not pay for gas costs. This enables them to set an unusually high gas price, potentially resulting in the theft of Ether from the diamond.

`saveResult` provides gas rebates equal to the gas spent in the function multiplied by `tx.gasprice`. Regular users gain no benefits from setting a high `tx.gasprice` as they are responsible for paying the increased gas cost. However, this does not apply to block builders. An oracle authority may collude with a block builder to set an abnormally high gas price to receive inflated rebates.

*contracts/src/aggregator/Aggregator.sol* SOLIDITY

```
function saveResult(
    address aggregatorId,
    address oracleId,
    int256 result
) internal {
    // snapshot gas left
    uint256 remainingGas = gasleft();

    /* computation intensive work */

    // handle payment for oracle results - don't err out for now, just don't pay
    uint256 fullReward = reward + (remainingGas - gasleft()) * tx.gasprice;

    // remove funds from interval balance if we can (and if reward is not 0)
    if (aggregator.balanceLeftForInterval >= fullReward && reward != 0) {
        /* ... */

        // pay oracle for gas spend + reward - handing off trust to the oracle
    ↪  operator address
        payable(msg.sender).transfer(fullReward);
    }
}
```

The severity of this issue is mitigated by the fact that `balanceLeftForInterval` bounds the amount of rewards extractable.

### Remediation

Place a constant boundary on spent gas.

### Patch

Fixed in ee13d95 and 168ec96.

## OS-SWB-ADV-04 [low] | Neglected Balance Update

### Description

`openInterval` does not update the `aggregator.balance` field. Consequently, subsequent withdrawals may fail due to an inaccurate `aggregator.balance` value, even if there are sufficient funds available within the aggregator.

*contracts/src/Switchboard/aggregator/Aggregator.sol* SOLIDITY

```
function openInterval(address aggregatorId) external payable guarded(4) {
    AggregatorLib.Aggregator storage aggregator = AggregatorLib.aggregators(
        aggregatorId
    );
    uint256 reward = OracleQueueLib.oracleQueues(aggregator.queueId).reward;

    if (msg.value < reward * (aggregator.config.minOracleResults + 1)) {
        revert ErrorLib.InsufficientBalance();
    }

    AggregatorLib.refreshInterval(
        aggregatorId,
        reward * (aggregator.config.minOracleResults + 1)
    );
}
```

For instance, the vulnerability may result in an integer overflow during the execution of `withdrawIntervalAndBalance`, potentially resulting in failures of legitimate oracle result post transactions.

*contracts/src/Switchboard/aggregator/AggregatorLib.sol* SOLIDITY

```
function withdrawIntervalAndBalance(
    address aggregatorId,
    uint256 amount
) internal {
    Aggregator storage aggregator = diamondStorage().aggregators[
        aggregatorId
    ];
    aggregator.balanceLeftForInterval -= amount;
    aggregator.balance -= amount;
}
```

### Remediation

Update `aggregator.balance` in `openInterval`.

### Patch

Fixed in fdaed57.

## OS-SWB-ADV-05 [low] | Incorrect Transaction Expiration Checks

### Description

The implementation of the deadline expiration checks in `runTransactions` and `forward` is incorrect, resulting in these function invocations reverting unconditionally.

SOLIDITY

```
function runTransactions(
    address functionId,
    uint256 reward,
    TransactionLib.Transaction[] memory transactions,
    bytes[] memory signatures
) internal {
    FunctionLib.SbFunction storage fn = FunctionLib.funcs(functionId);
    for (uint256 i = 0; i < transactions.length; i++) {
        /* ... */

        if (block.timestamp < transaction.expirationTimeSeconds) {
            revert ErrorLib.TransactionExpired();
        }

        /* ... */
    }
}
```

The condition for expiration checks if the block timestamp is greater than the expiration deadline. Therefore, the direction of the inequality should be inverted to accurately reflect this condition.

### Remediation

Invert the inequality.

### Patch

Fixed in a58f237 and 4e1d94c.

## OS-SWB-ADV-06 [low] | Incorrect Reentrancy Guard Implementation

### Description

The implementation of the `guard` modifier is incorrect and may be bypassed through specific call paths.

*contracts/src/Switchboard/util/Recipient.sol* SOLIDITY

```
function hasEntryCode(uint256 code) internal view returns (bool) {
    return diamondStorage().code & code != 0;
}

modifier guarded(uint256 code) {
    uint256 startCode = UtilLib.getCode();

    // if this is the first call to a guarded function, set the code (permissions
    ↪  level for the entry point)
    if (startCode == 0) {
        UtilLib.setCode(code);
    }

    // if the code is the same as the code - this represents a re-entry, gotta
    ↪  revert
    if (startCode != 0 && startCode == code) {
        revert ErrorLib.InvalidEntry();
    }

    // if the code is not a subset of the code - entry is not allowed
    if (startCode != 0 && !UtilLib.hasEntryCode(code)) {
        revert ErrorLib.InvalidEntry();
    }

    _;

    // reset the code to the previous value - will end up being 0
    UtilLib.setCode(startCode);
}
```

The `guard` modifier allows entry if bits intersect between the callee's `code` and the caller's `code`. However, this implementation allows for a nested call path such as `guarded(258)` → `guarded(257)` → `guarded(1)` to bypass the intended restriction. This is a violation as `258 & 1 = 0`, but calling `guarded(257)` in the middle enables this bypass.

### Remediation

Re-implement reentrancy guards to prevent reentering unless self-forwarding behavior is intended.

### Patch

Fixed in d57d785 and d9ab8a0.

## OS-SWB-ADV-07 [low] | Incorrect Array Removal Logic

### Description

`removeMrEnclaveToOracleQueue` fails to revert when an element is missing, leading to the unintended removal of an entry. The function utilizes a linear search on the `mrEnclaves` array to locate the index intended for removal. However, it mistakenly proceeds with the removal process regardless of the presence of `mrEnclave`, removing the default index.

*contracts/src/Switchboard/util/Recipient.sol* SOLIDITY

```
    function removeMrEnclaveFromOracleQueue(
        address queueId,
        bytes32 mrEnclave
    ) internal {
        DiamondStorage storage ds = diamondStorage();
        AttestationConfig storage config = ds.queueAttestationConfigs[queueId];
        uint256 idx = 0;
        for (uint256 i = 0; i < config.mrEnclaves.length; i++) {
            if (config.mrEnclaves[i] == mrEnclave) {
                idx = i;
                break;
            }
        }
        uint256 lastIdx = config.mrEnclaves.length - 1;
        config.mrEnclaves[idx] = config.mrEnclaves[lastIdx];
        config.mrEnclaves.pop();
    }
```

### Remediation

Perform element removal only when the element is present.

### Patch

Fixed in 7dc4922.

# 05 | 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 lead to security issues in the future.

| ID | Description |
|---|---|
| OS-SWB-SUG-00 | `saveResults` uses a delegate call for invoking `oracleHeartbeat`, which is susceptible to `msgSender` forgery. |
| OS-SWB-SUG-01 | `verifyEnclave` has a fixed expiration threshold of 20 seconds, which may necessitate modifications depending on chain properties or conditions. |
| OS-SWB-SUG-02 | `saveResults` calculates a function selector in a gas-costly manner by invoking `keccak256` on the function signature string. |

## OS-SWB-SUG-00 | Potentially Dangerous Self Call Pattern

Using a delegate call in `saveResults` for invoking `oracleHeartbeat` raises concerns due to the potential interaction with `EIP2771`. Introducing `EIP2771` alters the behavior of determining the sender of the message in a self-call context, where the sender is determined by inspecting the call data rather than relying on the `msg.sender` value, as implemented in `getMsgSender`.

SOLIDITY

```
function getMsgSender() internal view returns (address payable signer) {
    signer = payable(msg.sender);
    if (msg.data.length >= 20 && signer == address(this)) {
        assembly {
            signer := shr(96, calldataload(sub(calldatasize(), 20)))
        }
    }
}
```

Thus, if `saveResults` is called in a self-call context, `getMsgSender` will return `oracleId`. By setting `oracleId` to the address of an oracle queue authority, an attacker may bypass authority checks.

*contracts/src/Switchboard/aggregator/Aggregator.sol* SOLIDITY

```
function saveResults(
    address[] calldata ids, // Aggregator ids
    int256[] calldata results, // Results from the oracle for each feed listed in
    ↪  ids
    address queueId, // Oracle Queue Id
    uint256 oracleIdx // Oracle Index in the queue
) external guarded(4) {
    /* ... */
    bytes4 heartbeatSelector = bytes4(
        keccak256("oracleHeartbeat(address)")
    );
    address facet = LibDiamond
        .diamondStorage()
        .selectorToFacetAndPosition[heartbeatSelector]
        .facetAddress;
    (bool success, bytes memory returnData) = address(facet).delegatecall(
        abi.encodeWithSelector(heartbeatSelector, oracleId)
    );
    /* ... */
}
```

Currently, reentering `saveResults` is infeasible due to the absence of external address calls in functions marked with `guarded(4)`. However, we advise addressing this issue nonetheless due to future code additions.

### Remediation

Handle `EIP2771` in the self-call to `oracleHeartbeat`.

SOLIDITY

```
function saveResults(
    address[] calldata ids, // Aggregator ids
    int256[] calldata results, // Results from the oracle for each feed listed in
    ↪  ids
    address queueId, // Oracle Queue Id
    uint256 oracleIdx // Oracle Index in the queue
) external guarded(4) {
    /* ... */
    (bool success, bytes memory returnData) = address(this).call(
        abi.encodeWithSelector(heartbeatSelector, oracleId, msg.sender)
    );
    /* ... */
}
```

### Patch

Fixed in d9ab8a0.

## OS-SWB-SUG-01 | Constant Expiry Threshold

### Description

`verifyEnclave` relies on a fixed expiration threshold of 20 seconds, which may prove inadequate in scenarios where the chain is congested or if the contract is deployed on an EVM sidechain with a different block generation time. The function currently reverts if the time reported by the verifier enclave authority and the execution time (`block.timestamp`) differ by more than 20 seconds. While this measure is commendable, it is advisable to make the threshold configurable as this contract may be deployed on chains with varying properties or under changing chain conditions.

*contracts/src/Switchboard/enclave/Enclave.sol* SOLIDITY

```
function verifyEnclave(
    address verifierId, // caller's enclave/enclave address
    address enclaveId, // enclave address to be verified
    uint256 enclaveIdx, // enclave idx on verifier queue
    uint256 timestamp, // timestamp of enclave (to be validated against block
    ↪  timestamp)
    bytes32 mrEnclave // enclave measurement of enclave
) external guarded(1) {
    /* ... */

    uint256 timestampdiff = UtilLib.abs(
        int256(block.timestamp) - int256(timestamp)
    );

    if (timestampdiff > 20) {
        revert ErrorLib.IncorrectReportedTime();
    }

    /* ... */
}
```

### Remediation

Implement setter functions to set this threshold.

### Patch

Fixed in ca211e0.

## OS-SWB-SUG-02 | Gas Optimization

### Description

The invocation of `keccak256` for selector computations in `saveResults` may be optimized. Using `keccak256` for selector calculations involves a computationally expensive operation, resulting in higher gas consumption. An alternative approach to improve efficiency is to replace the invocation of `keccak256` with the `.selector` syntax available in Solidity.

*contracts/src/Switchboard/aggregator/Aggregator.sol* SOLIDITY

```
function saveResults(
    address[] calldata ids, // Aggregator ids
    int256[] calldata results, // Results from the oracle for each feed listed in
    ↪  ids
    address queueId, // Oracle Queue Id
    uint256 oracleIdx // Oracle Index in the queue
) external guarded(4) {
    /* ... */
    bytes4 heartbeatSelector = bytes4(
        keccak256("oracleHeartbeat(address)")
    );
    address facet = LibDiamond
        .diamondStorage()
        .selectorToFacetAndPosition[heartbeatSelector]
        .facetAddress;
    (bool success, bytes memory returnData) = address(facet).delegatecall(
        abi.encodeWithSelector(heartbeatSelector, oracleId)
    );
    /* ... */
}
```

### Remediation

Replace the call with `Oracle.oracleHeartbeat.selector` or validate the oracle by observing state variables.

### Patch

Fixed in 55ac77b.

# A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the General Findings section.

| | |
|---|---|
| **Critical** | Vulnerabilities that immediately lead to loss of user funds with minimal preconditions<br>Examples:<br>• Misconfigured authority or access control validation<br>• Improperly designed economic incentives leading to loss of funds |
| **High** | Vulnerabilities that could lead to loss of user funds but are potentially difficult to exploit.<br>Examples:<br>• Loss of funds requiring specific victim interactions<br>• Exploitation involving high capital requirement with respect to payout |
| **Medium** | Vulnerabilities that could lead to denial of service scenarios or degraded usability.<br>Examples:<br>• Malicious input that causes computational limit exhaustion<br>• Forced exceptions in normal user flow |
| **Low** | Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk.<br>Examples:<br>• Oracle manipulation with large capital requirements and multiple transactions |
| **Informational** | Best practices to mitigate future security risks. These are classified as general findings.<br>Examples:<br>• Explicit assertion of critical internal invariants<br>• 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 implementation of the program 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 sum, 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 get a comprehensive understanding of 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.