cOinspect

You build, we defend.

flare

**Source Code Audit**

Web2 JSON Attestation V2

October 2025

# JQ Filter V2

## Source Code Audit

Version: v251208 Prepared for: Flare October 2025

# Security Assessment

# 1. Executive Summary

In **October 2025**, **Flare** engaged Coinspect to perform the second Source Code Audit of the `Web2Json` Attestation. The objective of the project was to evaluate recent improvements made to the `Web2Json` requests processing.

| Solved | Caution Advised | Resolution Pending |
|---|---|---|
| High<br>0 | High<br>0 | High<br>0 |
| Medium<br>2 | Medium<br>0 | Medium<br>0 |
| Low<br>2 | Low<br>0 | Low<br>0 |
| No Risk<br>0 | No Risk<br>0 | No Risk<br>0 |
| Total<br>**4** | Total<br>**0** | Total<br>**0** |

The assessment revealed two medium-risk problems, one of which allows adversaries to override `Web2Json` responses via malicious JQ filters. Even though the code underwent significant improvements around JQ filters sanitization, Coinspect discovered that the malicious input reported as `FWJQ-01` in the initial review of the present scope was still valid. The remaining issue could allow adversaries to crash the API verifier server by generating thousands of `Web2Json` requests.

Finally, the report includes two low-risk findings describing potentially problematics scenarios involving GitHub actions configuration files.

# 2. Summary of Findings

This section provides a concise overview of all the findings in the report grouped by remediation status and sorted by estimated total risk.

## 2.3 Solved issues & recommendations

These issues have been fully fixed or represent recommendations that could improve the long-term security posture of the project.

| Id | Title | Risk |
|---|---|---|
| FWJQ-10 | Attackers can generate fake attestations via crafted JSON queries (v2) | Medium |
| FWJQ-11 | Unbounded request queue in ProcessPoolService may lead to Denial-of-Service (DoS) | Medium |
| FWJQ-12 | Unpinned third-party GitHub actions | Low |
| FWJQ-13 | Unverified external resources downloaded in CI pipeline | Low |

# 3. Scope

The scope was set to be the following repositories:

- https://gitlab.com/flarenetwork/fdc/verifier-indexer-api between commits **aabc258a3806828f221cd5d923280cc01aace898** and **5e036b364f839092c089ea106e6ec147055fa9e8**, with a specific focus on the improvements in the processing of `Web2Json` attestation requests.
- https://gitlab.com/flarenetwork/FSP/flare-smart-contracts-v2 between commits **ecfefacd5677836974bbc193d60031796fb0bd18** and **1f538d7225a9b5218eb447a2e8a44bef67aaa7dd**.

# 4. Assessment

The `Web2Json` Attestation feature is designed to bridge Web2 data into on-chain environments. It allows users to request data from a target server, process the JSON response with a custom JQ filter, and receive verifiable attestation for use in smart contracts. This review specifically focused on the enhancements made since the previous audit in June 2025.

The service now implements a process pool to handle data processing tasks in a limited number of isolated child processes, preventing the main application from being blocked by long-running or malicious requests. It maintains a queue of tasks meant to apply JQ filters and perform the ABI encoding, and distributes them to a pool of available worker processes. This architecture isolates each task in a sandboxed environment with strict memory (256mb) and execution time limits (1 second as default). If a worker process crashes, times out, or exceeds its memory allocation, it is automatically terminated and replaced without affecting the main server or other requests.

Also, the version reviewed prevents DNS rebinding attacks by performing two-step verification process. First, the `validateUrl` function resolves the target URL's hostname to a list of IP addresses, ensuring they are not private or malicious. These resolved IPs are then stored and subsequently enforced during the actual HTTP request within the `executeRequest` function. By locking the request to these pre-validated IP addresses, the system ensures that even if the DNS record is later changed by an attacker, the request will still be sent to the original, legitimate server.

A potentially problematic behavior was identified where the system parses JSON input in `src/verification/web-2-json/validate-json.ts` before validating its size, which could expose the application to out-of-memory errors when processing large payloads. However, this risk is mitigated by the `express.json` middleware, configured in `src/verifierServer.ts:39`, which enforces a default request body size limit of 100kb. This initial check ensures that excessively large JSON payloads are rejected before they can be parsed, effectively preventing the potential memory exhaustion issue.

## 4.1 Security assumptions

- The list of blocked/allowed hostnames and settings around `Web2Json` requests' restrictions are correctly configured and actively maintained.

- The underlying Node.js environment and all its dependencies (including `@jq-tools/jq`, `ethers`, `axios`) are free from known vulnerabilities.
- JQ queries, `Web2Json` requests and external API servers (web service that the verifier calls on behalf of the user) are assumed adversarial.
- Consumers of attestation requests and responses perform additional verification on the attestation requests' parameters (mainly JQ filters and target URLs) to ensure that server responses are not tampered with.
- The verifier indexer API server is assumed only directly reachable by the FDC client. Note however that it is assumed an adversary can generate any number of `Web2Json` requests at a negligible cost.

## 4.2 Testing

Coinspect observed that the project includes a suite of unit and end-to-end tests that cover the new JQ filter validation and process sandboxing functionalities. The tests validate security configurations, such as URL sanitization, hostname blocking, and HTTP method restrictions. Futhermore, the test suite includes cases to ensure that malicious JQ filters are correctly identified and rejected, and that resource intensive operations trigger timeouts as expected. However, the test suite lacks extensive negative test cases around JQ queries. This is why the data fabrication vulnerability `FWJQ-10` was missed, as there were no tests covering the example input from `FWJQ-01`.

## 4.3 Code quality

Coinspect observed that the new code added is well-structure, modular with a robust error handling, which enhances it overall maintainability.

# 5. Detailed Findings

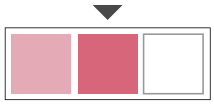
## FWJQ-10

## Attackers can generate fake attestations via crafted JSON queries (v2)

Status
**Solved**

Resolution
**Fixed**

Risk
**Medium**



Impact
**High**

Likelihood
**Low**

Location

`src/verification/web-2-json/validate-jq.ts`

## Description

The `validateJqFilter` function fails to properly sanitize user-supplied JQ filters, allowing queries that can modify or completely overwrite the server's JSON response. This issue was previously reported as `FWJQ-01` but the implemented fix was incomplete. An adversary can abuse this problem to manipulate the data returned by the verifier, breaking business logic or misleading downstream consumers of the attestation data.

The current implementation relies on a denylist of dangerous keywords (e.g., `system`, `eval`, `while`). However, this approach is insufficient as it does not account for queries that are syntactically safe but violate the intended threat model and business logic.

For example, the filter `.title = "AAAA"` uses a binary assignment expression to modify a field, while a filter like `{"userId":1,"id":1,"title":"AAAA","completed":false}` replaces the entire JSON object. Both are currently permitted and allow an attacker to control the structure and content of the final response.

```
// src/verification/web-2-json/validate-jq.ts:82
case 'binary':
  stack.push(expr.right);
  stack.push(expr.left);
  break;
```

The validation logic for binary expressions and object construction does not check for these types of manipulations, focusing only on preventing the use of explicitly blocked keywords.

Given that it is assumed that consumers of attestation requests and responses are in charge of making sure that only valid parameters are used, the likelihood of this being exploited was determined to be low.

## Recommendation

Document the expected responsibilities of components handling attestation requests and responses, including a clear warning about the risks of accepting unverified responses.

Given the difficulty of covering all malicious or unintended JQ filter attack vectors with a denylist approach, consider implementing a strict, limited allowlist of JQ filter expressions.

## Status

Fixed in commit **0c022215e3e9fdfe4b1aad0b779cfa089dd0c91b** and commit **03fbf8ea8f2646411aed716696facd52ebf8d877** from https://github.com/flare-foundation/flare-specs. The JQ queries filtering was improved to prevent assignment operators and the logic was modified to use an allowlist instead of a denylist. Note however that the malicious input mentioned in this issue, which allows an adversary to completely override the server response, is still accepted by the filter.

On the other hand, the warning was properly added to the attestation type in Flare's specification documents.

## Proof-of-Concept (PoC)

To test the new filtering functionality, Coinspect started a local API verifier instance. Using the following request:

```
{
  "attestationType":
"0x494a736f6e4170690000000000000000000000000000000000000000000000000000",
  "sourceId":
"0x5745423200000000000000000000000000000000000000000000000000000000",
  "requestBody": {
      "url": "https://jsonplaceholder.typicode.com/todos/1",
      "httpMethod":"GET",
      "headers":"",
      "body":"",
      "queryParams":"",
      "postProcessJq":
'{"userId":1,"id":1,"title":"AAAA","completed":false}',
      "abiSignature": (
          '{"components":['
          '{"internalType":"uint8","name":"userId","type":"uint8"},'
          '{"internalType":"uint8","name":"id","type":"uint8"},'
          '{"internalType":"string","name":"title","type":"string"},'
          '{"internalType":"bool","name":"completed","type":"bool"}'
          '],"name":"task","type":"tuple"}'
      ),
  },
}
```

The server returned the following response, like the one obtained in FWJQ-01.

```
"responseBody": {
      "abiEncodedData":
"0x0000000000000000000000000000000000000000000000000000000000000020000
0000000000000000000000000000000000000000000000000000000000000010000000000
0000000000000000000000000000000000000000000000000000000100000000000000000
0000000000000000000000000000000000000000800000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000441414141000000000000000000000000000000000
00000000000000000000"
    }
```

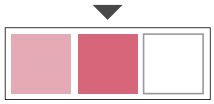
# FWJQ-11

# Unbounded request queue in ProcessPoolService may lead to Denial-of-Service (DoS)

Status
**Solved**

Resolution
**Fixed**

Risk
**Medium**



Impact
**High**

Likelihood
**Medium**

Location

`src/verification/web-2-json/process-pool.service.ts`

## Description

The `ProcessPoolService` utilizes an unbounded in-memory queue for processing `Web2Json` verification requests, which can be exploited to cause a Denial of Service (DoS). An adversary can abuse this by submitting thousands of on-chain requests likely pointing to a malicious or buggy server that intentionally delays responses and sends large data payloads, causing the verifier API to run out of memory and crash.

The service is designed to handle intensive tasks by queueing incoming requests and processing them in a pool of workers. However, the queue itself has no size limit. A malicious server can delay its response up to the configured `requestTimeoutMs` (defaulting to 5000 ms) and return a response body up to `maxResponseSize` (defaulting to 1 MB).

```
// src/config/defaults/web2Json-config.ts
maxResponseSize: 1024 * 1024,
requestTimeoutMs: 5_000,
```

If an attacker can generate on-chain requests at a rate faster than the service can process them, the `requestQueue` will grow indefinitely.

```
// src/verification/web-2-json/process-pool.service.ts:30
private requestQueue: QueuedRequest[] = [];
```

Each queued item holds the response body (`jsonData`), which can be up to 1 MB in size. With a standard Node.js memory limit (~1.5 GB available), the queue only needs to hold around 1,500 such requests to exhaust all available memory and crash the application, making the verifier unavailable.

The issue's likelihood was rated medium, as exploiting it requires an API server that returns unusually large responses and introduces maximum delays. However, Coinspect identified no significant barriers—such as rate limiting, queue constraints, or high transaction fees—that would effectively discourage attackers.

## Recommendation

Implement a mechanism to prevent the rapid accumulation of `Web2Json` requests. Consider implementing an exponential time-based fee selection mechanism.

## Status

Fixed in commit **0c022215e3e9fdfe4b1aad0b779cfa089dd0c91b**. The process pool service now defines a hard limit (`maxQueueSize`) of 1000 enqueued requests, which if, full, will start to rejected new requests.

The Flare team noted that the FDC client (outside the current scope) will handle re-attempting rejected requests. They also mentioned that, given the current 15M block gas limit, a maximum of 150 requests per second could be created under ideal conditions, assuming one block per second. Considering the request timeout (5s), it is considered safe under these assumptions.

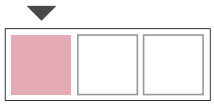
# FWJQ-12

## Unpinned third-party GitHub actions

Status
**Solved**

Resolution
**Fixed**

Risk
**Low**



Impact
**Medium**

Likelihood
**Low**

Location

`.github/workflows/build-container.yml`

## Description

The CI/CD workflow defined in `build-container.yml` utilizes several third-party GitHub actions pinned to mutable version tags (e.g., actions/checkout@v4, docker/login-action@v3) instead of immutable commit hashes. An adversary who compromises a third-party action's repository could push a malicious update to an existing tag. This would cause the workflow to unknowingly execute malicious code, which could lead to compromised builds or exfiltration of secrets.

The following actions are affected:

```
actions/checkout@v4
docker/setup-qemu-action@v3
docker/setup-buildx-action@v3
docker/login-action@v3
docker/build-push-action@v5
```

Also, the same workflow specifies `runs-on: ubuntu-latest`, which creates an uncontrolled dependency on the latest version of the Ubuntu runner image provided by the platform, allowing unvetted changes or vulnerabilities in the runner image to be automatically incorporated into the build environment.

## Recommendation

Pin all third-party GitHub Actions to their full-length commit SHA.

Pin the Ubuntu image version.

## Status

Fixed in commit **0c022215e3e9fdfe4b1aad0b779cfa089dd0c91b**. The Ubuntu runner and workflow actions are now pinned to a specific version and commit respectively.

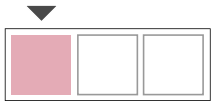
# FWJQ-13

## Unverified external resources downloaded in CI pipeline

Status
**Solved**

Resolution
**Fixed**

Risk
**Low**



Impact
**Low**

Likelihood
**Low**

Location

`https://gitlab.com/flarenetwork/fdc/verifier-indexer-api/.gitlab-ci.yml`

## Description

The `.gitlab-ci.yml` configuration for the `e2e_test` job downloads several database snapshot files from an external URL (`https://githubstatic.flare.center`) using `curl`. These downloaded resources are then used in the end-to-end tests. However, the integrity of these files is not verified after download.

This could lead to compromised test environments, skewed test results, or potentially the execution of malicious code within the CI/CD pipeline, depending on how the files are used.

```
curl -o test/e2e_tests/db/db_btc_testnet
https://githubstatic.flare.center/db_btc_testnet
curl -o test/e2e_tests/db/db_btc2_testnet
```

```
https://githubstatic.flare.center/db_btc2_testnet
curl -o test/e2e_tests/db/db_doge_testnet
https://githubstatic.flare.center/db_doge_testnet
curl -o test/e2e_tests/db/db_xrp_testnet
https://githubstatic.flare.center/db_xrp_testnet
```

## Recommendation

Implement integrity verification for the downloaded files.

## Status

Fixed in commit **d2a10a614396be4d45d03fc361cddc471980d7f1**. The workflow definition now verifies that the SHA sum of the database dumps match the expected ones.

# 6. Disclaimer

The contents of this report are provided "as is" without warranty of any kind. Coinspect is not responsible for any consequences of using the information contained herein.

This report represents a point-in-time and time-boxed evaluation conducted within a specific timeframe and scope agreed upon with the client. The assessment's findings and recommendations are based on the information, source code, and systems access provided by the client during the review period.

The assessment's findings should not be considered an exhaustive list of all potential security issues. This report does not cover out-of-scope components that may interact with the analyzed system, nor does it assess the operational security of the organization that developed and deployed the system.

This report does not imply ongoing security monitoring or guaranteeing the current security status of the assessed system. Due to the dynamic nature of information security threats, new vulnerabilities may emerge after the assessment period.

This report should not be considered an endorsement or disapproval of any project or team. It does not provide investment advice and should not be used to make investment decisions.