F Y E O

Security Code Review
FLARE Data Availability

FLARE

November 2024
Version 1.0

Presented by:
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Public

Table of Contents

Executive Summary

Overview

FLARE engaged FYEO Inc. to perform a Security Code Review of FLARE Data Availability.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on October 26 - November 04, 2024, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following issues have been identified during the testing period. These should be prioritized for remediation to reduce the risk they pose:

- FYEO-FLARE-01 – Multiple vulnerabilities in outdated version of Django
- FYEO-FLARE-02 – Package djangorestframework before 3.15.2 is vulnerable to Cross-Site Scripting
- FYEO-FLARE-03 – Incomplete code
- FYEO-FLARE-04 – Inconsistent logging
- FYEO-FLARE-05 – Setup defaults to None
- FYEO-FLARE-06 – Missing documentation of unit tests

Based on our review process, we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

The FYEO Review Team performed a Security Code Review FLARE Data Availability. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a private repository at https://gitlab.com/flarenetwork/FSP/data-availability with the commit hash 9199fb40d10a860b32a55753d351270e9be65df5.

Remediations were submitted with the commit hash 6e5c7da422d1a07bba01dd2db5f5e078bbe89906.

Files included in the code review

```
data-availability/
├── configuration/
│   ├── configs/
│   │   ├── __init__.py
│   │   ├── coston.py
│   │   ├── coston2.py
│   │   ├── flare.py
│   │   └── songbird.py
│   ├── __init__.py
│   ├── config.py
│   ├── contract_types.py
│   └── types.py
├── fdc/
│   ├── management/
│   │   ├── commands/
│   │   │   ├── __init__.py
│   │   │   └── process_fdc_data.py
│   │   └── __init__.py
│   ├── migrations/
│   │   ├── 0001_initial.py
│   │   └── __init__.py
│   ├── serializers/
│   │   ├── data.py
│   │   ├── query.py
│   │   └── request.py
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── models.py
│   ├── urls.py
│   └── views.py
```

Files included in the code review

```
├── fsp/
│   ├── migrations/
│   │   ├── 0001_initial.py
│   │   └── __init__.py
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── epoch.py
│   ├── models.py
│   ├── serializers.py
│   ├── tests.py
│   ├── urls.py
│   └── views.py
├── ftso/
│   ├── management/
│   │   ├── commands/
│   │   │   ├── __init__.py
│   │   │   └── process_ftso_data.py
│   │   └── __init__.py
│   ├── migrations/
│   │   ├── 0001_initial.py
│   │   └── __init__.py
│   ├── serializers/
│   │   ├── data.py
│   │   ├── query.py
│   │   └── request.py
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── models.py
│   ├── urls.py
│   └── views.py
├── processing/
│   ├── client/
│   │   ├── main.py
│   │   └── types.py
│   ├── __init__.py
│   ├── fdc_processing.py
│   ├── ftso_processing.py
│   ├── main.py
│   ├── processing.py
│   └── utils.py
├── project/
```

Files included in the code review

```
│   ├── settings/
│   │   ├── __init__.py
│   │   ├── ci_testing.py
│   │   ├── common.py
│   │   ├── local.py
│   │   └── remote.py
│   ├── __init__.py
│   ├── asgi.py
│   ├── urls.py
│   └── wsgi.py
└── manage.py
```

Table 1: Scope

Technical Analyses and Findings

During the Security Code Review FLARE Data Availability, we discovered:

- 2 findings with MEDIUM severity rating.
- 3 findings with LOW severity rating.
- 1 finding with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

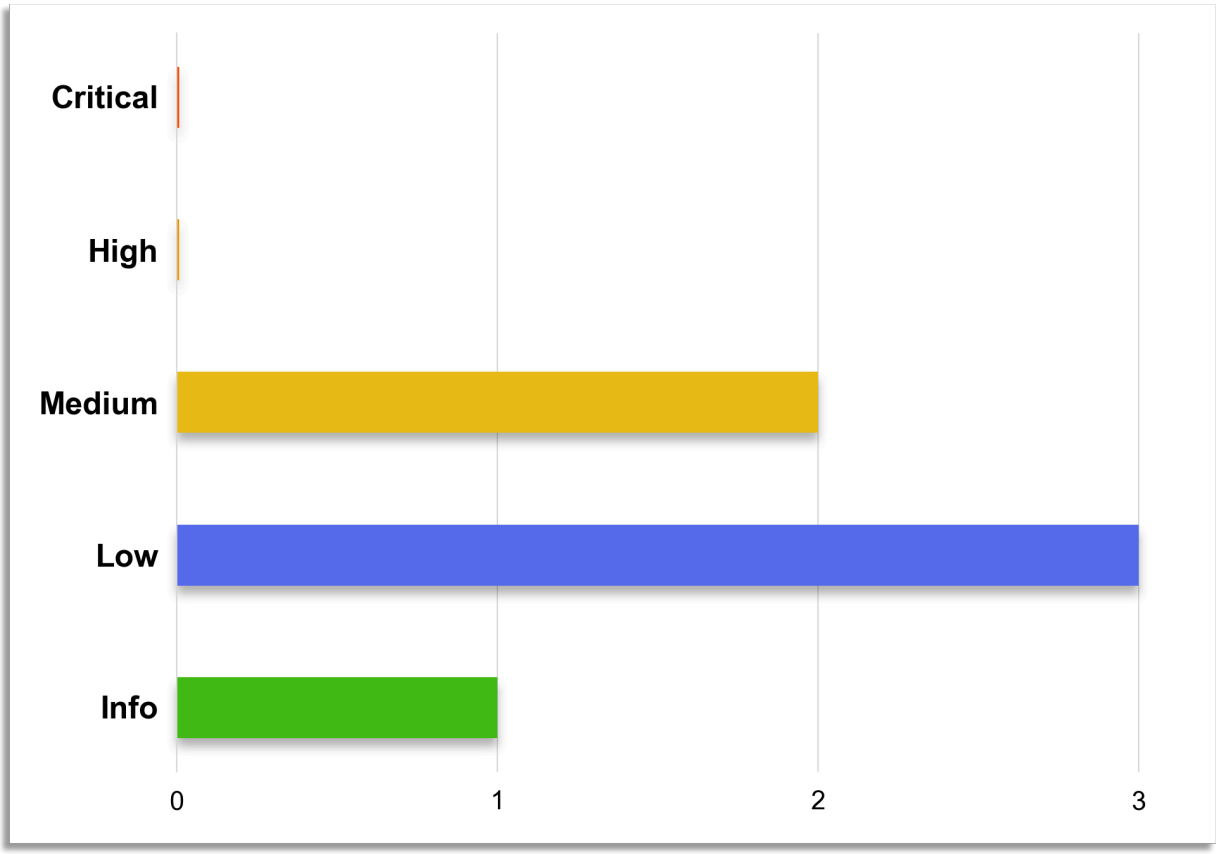

Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Severity	Description
FYEO-FLARE-01	**Medium**	Multiple vulnerabilities in outdated version of Django
FYEO-FLARE-02	**Medium**	Package djangorestframework before 3.15.2 is vulnerable to Cross-Site Scripting
FYEO-FLARE-03	**Low**	Incomplete code
FYEO-FLARE-04	**Low**	Inconsistent logging
FYEO-FLARE-05	**Low**	Setup defaults to None
FYEO-FLARE-06	**Informational**	Missing documentation of unit tests

Table 2: Findings Overview

Technical Analysis

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

Conclusion

Based on our review process, we conclude that the code implements the documented functionality to the extent of the reviewed code.

Technical Findings

General Observations

The Data Availability service is a Django application for handling and validating attestations and feed data tied to the Flare blockchain, focusing on cryptographic validation using Merkle trees. The service interfaces with external data sources through API clients, fetches data related to specific voting rounds, and uses Merkle tree structures to ensure the data's integrity before persisting it in a relational database.

The main logic revolves around the AttestationResult and FeedResult models, which store data about blockchain voting rounds, feeds, and associated attestation results. Each record in these models represents a verified attestation, containing fields like voting_round_id, cryptographic proofs, and serialized data about voting results. The AttestationResult model supports cryptographic verification by calculating and storing keccak hashes of each attestation, making it possible to verify data authenticity through Merkle proofs. The FeedResult model manages feeds associated with voting rounds, includes voting outcomes, turnout, and other metadata. For every feed, the model computes its hash based on Ethereum's ABI encoding scheme.

The service's API layer exposes data endpoints via Django REST Framework viewsets, allowing clients to retrieve attestation and feed results through GET and POST endpoints. Users can request feeds by voting round or obtain Merkle proofs for specified feeds. Each view validates incoming requests, which enforces schema integrity for requests and responses.

The Merkle tree structure is critical for data integrity. When processing attestations or feed data, the service organizes relevant records into a Merkle tree and compares its root hash with known values from external data sources or previous rounds. This comparison ensures that the data stored and served remains consistent with the data on-chain. If mismatches occur, such as differences in Merkle root hashes, the service raises errors to indicate potential data corruption.

Overall, this Django service provides a robust backend for blockchain data attestation, verifying data integrity with Merkle tree structures and offering an API layer for efficient access and integration into broader decentralized systems. It was noted however, that the service seems to have been rushed to the finish line. Error handling as well as logging, testing and overall documentation are lacking and more effort invested in these would greatly contribute to a more maintainable and secure codebase.

Multiple vulnerabilities in outdated version of Django

Finding ID: FYEO-FLARE-01
Severity: **Medium**
Status: **Remediated**

Description

The package `django` before version 5.0.9 is subject to multiple vulnerabilities.

Proof of Issue

File name: requirements.txt
Line number: 2

```
Found 11 known vulnerabilities in 2 packages Name                Version ID
Fix Versions ------------------- ------- ------------------- ------------------ django
5.0.6   PYSEC-2024-58       4.2.14,5.0.7 django              5.0.6   PYSEC-2024-57
4.2.14,5.0.7 django              5.0.6   PYSEC-2024-56       4.2.14,5.0.7 django
5.0.6   PYSEC-2024-59       4.2.14,5.0.7 django              5.0.6   PYSEC-2024-69
4.2.15,5.0.8 django              5.0.6   PYSEC-2024-70       4.2.15,5.0.8 django
5.0.6   PYSEC-2024-68       4.2.15,5.0.8 django              5.0.6   PYSEC-2024-67
4.2.15,5.0.8 django              5.0.6   PYSEC-2024-102      4.2.16,5.0.9,5.1.1 django
5.0.6   GHSA-rrqc-c2jx-6jgv 4.2.16,5.0.9,5.1.1
```

Severity and Impact Summary

Even though most of these vulnerabilities are in the web rendering and template stack of Django some of them could still be exploited leading to unexpected behavior and potentially lead to a denial of service situation for the web service.

Recommendation

Please upgrade the package to the latest version of the Django framework.

Package djangorestframework before 3.15.2 is vulnerable to Cross-Site Scripting

Finding ID: FYEO-FLARE-02
Severity: **Medium**
Status: **Remediated**

Description

The package `djangorestframework` before `3.15.2` is vulnerable to Cross-site Scripting (XSS) via the break_long_headers template filter due to improper input sanitation before splitting and joining with tags.

Proof of Issue

File name: requirements.txt
Line number: 12

Severity and Impact Summary

This could lead to cross site scripting and reflective injection of code in the responses.

Recommendation

Please upgrade the package to at least `3.15.2`.

Incomplete code

Finding ID: FYEO-FLARE-03
Severity: **Low**
Status: **Remediated**

Description

The codebase appears unfinished in many places and the handling of errors and edge cases is lacking, which could lead to unexpected behavior.

Proof of Issue

File name: processing/main.py
Line number: 29

```
def __init__(self, rpc_url: str, sync_config: SyncingConfig, relay: Contract):
    relay = relay
    assert relay is not None and relay.address is not None
```

The assignment does nothing.

File name: fsp/models.py
Line number: 40

```
def from_decoded_dict(cls, event_data: EventData, state):
    (
        _protocol_id,
        _voting_round_id,
        _is_secure_random,
        _merkle_root,
    ) = event_data_extract_args(event_data, "protocolId", "votingRoundId",
"isSecureRandom", "merkleRoot")
    protocol_id = int(_protocol_id)
    voting_round_id = int(_voting_round_id)
    is_secure_random = bool(_is_secure_random)
    merkle_root = un_prefix_0x(_merkle_root.hex().lower())
```

The state parameter remains unused.

File name: fdc/views.py
Line number: 37

```
def get_proof_round_id_bytes(self, request, *args, **kwargs):
    self.serializer_class = AttestationMinimalProofSerializer

    _body = AttestationTypeGetByRoundIdBytesRequest(data=self.request.data)
    print("HERE")
    _body.is_valid(raise_exception=True)
    print("HERE2")
    body = _body.validated_data
```

The parameters are unused and the print code indicates an attempt to debug something.

File name: configuration/contract_types.py
Line number: 13

```
def abi_from_file_location(file_location):
    return json.load(open(file_location))["abi"]
```

Make sure to close the file handle.

File name: processing/main.py
Line number: 79

```
try:
    logger.debug(f"Processing round {ev}")
    protocol_config.processor.process(ev)
except Exception as e:
    processing_qeue.put(ev)
```

This queue may grow in an uncontrolled fashion as there are no safeguards in place before the queue is drained again.

```
# TODO: make sure this timestamp is correct
# TODO:(luka) Should we validate?
# TODO:(matej) validate both at the same time
# TODO:(luka) Also handle too early rounds
# TODO:(luka) we have no data, error/none
# TODO:(luka) We can handle this differently
# TODO:(luka) WIP
```

Several comments indicate the work is unfinished.

Severity and Impact Summary

A lack of error handling and resource management could lead to availability problems.

Recommendation

Make sure to implement safeguards throughout the code base.

Inconsistent logging

Finding ID: FYEO-FLARE-04
Severity: **Low**
Status: **Remediated**

Description

Some parts of the code use a logger, others do print. There is also a log of `self` which could eventually log sensitive information.

Proof of Issue

File name: processing/main.py
Line number: 68

```
print(f"Processing from {from_block_exc} to {to_block_inc}, found {len(events)} events")
```

File name: ftso/views.py
Line number: 42

```
print(f"Querying for available feeds for round: {voting_round_id}")
```

File name: processing/processing.py
Line number: 21

```
logging.error(
    "Protocol ID mismatch %s: \nExpected: %s \nReceived: %s",
    self,
    self.protocol_id,
    root.protocol_id,
)
```

Severity and Impact Summary

Logging `self` could potentially include sensitive data if the object were to be modified to store such data.

Recommendation

Make sure to not log sensitive data or objects that could be modified to include such data.

Setup defaults to None

Finding ID: FYEO-FLARE-05
Severity: **Low**
Status: **Remediated**

Description

For each name the setup function attempts to find a contract. The initial value is assigned to be `None`. If a contract can not be found, it is most likely an error however.

Proof of Issue

File name: configuration/contract_types.py
Line number: 98

```
@classmethod
def default(cls) -> Self:
    attr_names = [a.name for a in cls.__attrs_attrs__]  # type: ignore
    with open(f"configuration/chain/{settings.CONFIG_MODULE}/contracts.json") as f:
        contracts = {c["name"]: c["address"] for c in json.load(f)}

        kwargs = {}

        for name in attr_names:
            kwargs[name] = None
            if name in contracts:
                kwargs[name] = Contract(
                    name,
                    contracts[name],

f"configuration/chain/{settings.CONFIG_MODULE}/artifacts/{name}.json",
                )

        return cls(**kwargs)
```

Severity and Impact Summary

The relay contract is checked, but missing contracts may lead to error if the code is changed later on.

Recommendation

While the code asserts that `relay is not None and relay.address is not None` for the relay contract, this setup could have `None` values for any other contract that may be used. If the config expects these to be set, a value of `None` should be treated as an error.

Missing documentation of unit tests

Finding ID: FYEO-FLARE-06
Severity: **Informational**
Status: **Remediated**

Description

While there are unit tests there is no instruction on how to run them.

The documentation states how to build the docker image but there does not seem to be any way to run the unit tests through the built docker image.

Severity and Impact Summary

Informational

Recommendation

Improve the instructions on how to run the unit test either by a local command or through a command on the docker image.

Our Process

Methodology

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

Figure 2: Methodology Flow

Kickoff

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.

The Classification of vulnerabilities

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations