Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Pentest-Report Firedancer Metrics Sources & APIs 02.2025

Cure53, Dr.-Ing. M. Heiderich, MSc. H. Moesl-Canaval

Index

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Introduction

This report presents the results of a 2025 penetration test and source code audit focusing on the Firedancer Metrics API and underlying source code, which represents the second collaborative project between the Firedancer team and Cure53.

In context, the scope, objectives, and work day allocation (six days) were agreed upon during early discussions held between the two organizations in January 2025. The technical tasks were completed by two experienced pentesters in CW06 2025, focusing on a single Work Package (WP) named *WP1: White-box pen.-tests & source code audits against Firedancer Metrics API*.

Previously, the inaugural Firedancer-Cure53 initiative was performed in November 2024 and documented under the report entitled ASY-01. However, this ASY-02 iteration focused solely on the Firedancer Metrics API and codebase, which were not in scope for the preceding iteration.

Cure53 was provided with sources, test-user credentials, and other miscellaneous assets to facilitate the investigative undertakings, which complied with a white-box pentest methodology. Preparations were completed in late January 2025, namely during CW05, to encourage a hindrance-free examination period.

Slack was the communications platform of choice, as per the opening audit. All team members from both companies that played an active part in this exercise were invited to join the private channel and engage in the ongoing discourse when required. The discussions were generally seamless, with minimal queries needed owing to the thorough scope setup. Cure53 is pleased to note that no delays or blockers were encountered at any stage. Live reporting was offered but ultimately deemed unnecessary.

An extensive degree of coverage was achieved during the allotted time frame, within which the Cure53 testers located a total of four weaknesses affecting the scope under the current implementation. All four were classified as general weaknesses with lower exploitation potential.

The minimal yield of tickets was somewhat expected given the constrained nature of the targeted premise. Nevertheless, this should not detract from the defensive effectiveness of the components at hand, considering that major security ramifications were completely avoided.

In summary, Cure53 can only congratulate the in-house developers at Firedancer for the healthy security foundation established for their products. Albeit, ample time and resource investments, as well as periodic external auditing programs, are required to maintain an exemplary and future-proofed security posture.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Moving forward, the *Scope*, test setup, and available materials are itemized below. Next, the *Test Methodology* clarifies the evaluation techniques applied by Cure53 and all interesting subsequent observations, with the aim of verifying the test team's efforts in spite of the lack of vulnerabilities detected. Subsequently, all *Identified Vulnerabilities* and *Miscellaneous Issues* are provided in chronological order.

The tickets offer an advanced rundown, a Proof-of-Concept (PoC) if required, and advice on fixing the problem in question. Lastly, the *Conclusions* chapter collates all general impressions gained for the in-scope features, giving final thoughts on the wider security posture and how it can be improved.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Scope

- **Pen.-tests & code audits against Firedancer Metrics source & API endpoints**
  - **WP1:** White-box pen.-tests & source code audits against Firedancer Metrics API
    - **Source code:**
      - File:
        - *fdmetrics.zip*
      - SHA256:
        - 47c5952e350ba3fd418f63f7483138b7f914f5bc6831434b56dfdaaa44f6cf7b
    - **Environment:**
      - *./clickhouse server --config-file=./config.xml*
    - **API endpoints:**
      - *GET /health*
      - *POST /insert*
    - **Key focus areas:**
      - Potential misconfiguration of the HTTP server and sophisticated web attacks
      - Metrics data includes only legitimate staked validators, filtered by public key from insert requests submitted by any user
      - Server misconfiguration
      - Liveness issues in HTTP server and/or DB
      - Severe latency issues in HTTP server and/or DB
      - Accuracy of information stored in DB
      - Web 2.0 attack vectors (SQLi, ..)
      - Improper usage of cryptography APIs
      - Sensitive information leakage
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Test Methodology

This section documents the testing methodology applied by Cure53 during this project and discusses the resulting coverage, shedding light on how various components were examined. Further clarification concerning areas of investigation subjected to deep-dive assessment is offered, especially in the absence of significant security vulnerabilities detected.

At an advanced level, the Firedancer metrics server is designed for SOL validators to submit metrics data via the exposed */insert* endpoint, which inserts data into a ClickHouse database upon validation. In summary, this review focused solely on two exposed Firedancer metrics server API endpoints, */insert* (POST) and */health* (GET), which generally offer a minimal degree of attack surface.

Cure53 commenced the engagement by inspecting the provided information and conducting a diligent assessment of the source code. A testing environment was not provided, thus the Cure53 analysts set up a local environment to facilitate dynamic request evaluations.

For optimal pentest alignment and optimization, a list of key focus areas was specified in advance, as follows:

- Potential misconfigurations of the HTTP server and susceptibility to typical web attacks
- Liveness issues affecting HTTP server and/or database
- Severe latency issues within HTTP server and/or database
- Accuracy and integrity of database-stored information
- Web 2.0 attack vectors, including SQL injection (SQLi) and related threats
- Improper use of cryptographic APIs
- Leakage of sensitive information
- Issues related to schema migrations causing system failures

Regarding authentication, no access controls prevent SOL validators from reaching the exposed endpoints. Alternatively, clients must sign the JSON payload using their identity private key and include the signature in the *Authorization* header. All data ingested into ClickHouse is verified against this signature to ensure integrity. While any individual could impersonate a validator, downstream filtering based on public keys ensures that only legitimate staked validators are considered when querying metrics.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

The Firedancer Metrics backend is written in Rust, which was designed in accordance with security principles and hence remains an astute choice for developing secure systems and applications. Core Rust benefits include:

- **Memory safety**: Rust offers a borrow checker that prevents common memory safety issues such as null/dangling pointers, buffer overflows, and data races.
- **Type safety**: Rust's performant type system helps prevent type confusion and other type-related vulnerabilities.
- **Concurrency safety**: Rust prioritizes concurrency safety, with features such as ownership and borrowing that serve to prevent data races and other concurrency-related vulnerabilities.

The in-scope source code was subjected to static analysis, confirming a soundly structured and commendable composition. The development team’s understanding of secure coding practices in Rust quickly became apparent. Cure53 honed in on locating typical programming mistakes that oftentimes occur within languages such as Rust, including remote DoS vectors.

Pertinently, Cure53 observed that the API server listens exclusively for HTTP requests. While this is suboptimal from a security perspective, the transmitted data consists of public metrics data related to SOL primitives solely. Moreover, data integrity is ensured via cryptographic signatures validated by the server application, reducing the immediate necessity of encrypted traffic via HTTPS.

The Rust repository provided source code for two crates, *fdsrv* and *firedancer-metrics-blocks*, which was analyzed for potential panics introduced by absent or incomplete error handling protocols. While error handling is robust during server runtime, the corresponding implementations for application startups and the *firedancer-metrics-blocks* binary are lacking, which could lead to panic situations as discussed in tickets ASY-02-003 and ASY-02-004 respectively. Additionally, the presence of codebase logical errors was estimated, although Cure53 was unable to detect any flaws of this ilk.

During the metrics server application build-time process, Cure53 noticed that Rust types were generated from schema files initially provided as JSON. These JSON files define the data reported by validators to the metrics server. A fundamental component here is the common type containing a timestamp and an identity field (among other properties), the latter of which provides the client’s public key and is a key factor in metrics data validation. Payload signature verification relies on an ED25519 signature check, which is implemented using the *ed25519_dalek* crate. The signature verification process appeared correctly implemented and secure.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

At runtime, the server binary compares the database schema against the built-in counterpart. Upon discrepancy detection, the system triggers a schema migration to align the database with the expected structure. The schema validation and migration logic were reviewed, verifying that the construct is risk averse.

Elsewhere, the Cure53 consultants analyzed the liveness of the HTTP server and searched for potential latency faults. The HTTP server application utilizes the tokio crate with tasks and watch channels, which is a recognized choice in Rust. The testers attempted to create scenarios that could cause unintended behaviors, such as circumventing metric data database storage. However, the implementation successfully withstood all duress measures applied by the review team.

Next, the Python scripts responsible for importing SOL-relevant data were probed for SQLi vulnerabilities and other detrimental circumstances. Here, Cure53 noted that the handling of SOL leader schedule data could be improved due to the potential risk of SQLi, as reported under ticket ASY-02-001. Besides, the testing team was unable to locate any other injection-related shortcomings within the analyzed scripts.

Generally speaking, fuzzing is a software testing technique that involves feeding invalid, unexpected, or random data into a computer program to uncover defects and vulnerabilities that could introduce a program crash, Denial-of-Service (DoS), or otherwise undesirable activities. Firedancer provided access to several fuzz test cases for this exercise, which is impressive and highlights the internal team’s stringent application of these software practices. Cure53’s close scrutiny of the existing fuzzing harnesses confirmed that they are ideally designed and appropriate for their intended purpose, covering the majority of the server's functionality.

Finally, the provided source code was vetted to pinpoint any outdated and vulnerable software dependencies. One such dependency in the Rust project was identified and reported under ticket ASY-02-002. While these findings do not always result in directly exploitable vulnerabilities, the in-house team should strive to keep all third-party libraries up-to-date for software hygiene purposes and in accordance with security best practices.

All in all, Cure53 is pleased to confirm that the Firedancer Metrics API offers first-rate protection from a security standpoint.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, all tickets are given a unique identifier (e.g., *ASY-02-001*) to facilitate any future follow-up correspondence.

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, while a vulnerability is present, an exploit may not always be possible.

ASY-02-001 WP1: Potential SQLi in leader slots fetching code *(Low)*

Cure53's static analysis of the firedancer-metrics repository confirmed the presence of a potential SQL injection vulnerability in the Python code responsible for retrieving SOL leader schedule data. The code directly incorporates data via an API response originating from *https://api.{network}.solana.com* into a SQL query for a ClickHouse database hosted at *http://guse1b-firedancer-ch1.jumpisolated.com*.

To successfully exploit this flaw, an attacker would need to manipulate responses sent by the server hosted at *https://api.{network}.solana.com*, explaining the ticket's categorization as *Miscellaneous* and the reduced severity rating of *Low*.

Affected file:
firedancer-metrics/contrib/fetch-leader-schedule.py

Affected code:

```
    for network in ['testnet', 'mainnet-beta', 'devnet']:
        response = requests.post(
            f'https://api.{network}.solana.com/',
            headers={'Content-Type': 'application/json'},
            data=json.dumps({'jsonrpc': '2.0', 'id': 1, 'method':
'getEpochInfo'}))
        if response.status_code//100 != 2:
            print(f'Error fetching current epoch info on {network}')
            error_responses.append(response)
            continue
        response.raise_for_status()
        response = json.loads(response.text)
        epoch = response['result']['epoch']
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
        [...]

        cluster = 'mainnet' if network == 'mainnet-beta' else network
        response = requests.get(
            'http://guse1b-firedancer-ch1.jumpisolated.com:8123',
            params={'query': f'SELECT COUNT(*) AS count FROM
leader_schedule WHERE cluster=\'{cluster}\' AND epoch={epoch} FORMAT
JSONEachRow'},
            auth=('fd_ci_rw_user', os.environ['CLICKHOUSE_PASSWORD']))
```

To mitigate this issue, Cure53 strongly recommends casting the returned epoch value to an integer, thus negating the potential SQL injection vector.

ASY-02-002 WP1: Outdated & vulnerable dependencies *(Info)*

During the evaluations, Cure53 located a specific software package that is outdated and likely prone to security risks, as displayed below. Notably, the version information provided is based on data collected at the time of testing. Whether these vulnerabilities are exploitable entirely depends on how the relevant functionality is used in the targeted application at present.

Command:

```
% cargo audit

Crate:     openssl
Version:   0.10.68
Title:     ssl::select_next_proto use after free
Date:      2025-02-02
ID:        RUSTSEC-2025-0004
URL:       https://rustsec.org/advisories/RUSTSEC-2025-0004
Solution:  Upgrade to >=0.10.70
Dependency tree:
openssl 0.10.68
└── native-tls 0.2.12
    ├── tokio-native-tls 0.3.1
    │   ├── reqwest 0.11.27
    │   │   └── firedancer-metrics-blocks 0.1.0
    │   └── hyper-tls 0.5.0
    │       └── reqwest 0.11.27
    ├── reqwest 0.11.27
    └── hyper-tls 0.5.0

Crate:     atty
Version:   0.2.14
Warning:   unmaintained
Title:     `atty` is unmaintained
Date:      2024-09-25
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
ID:        RUSTSEC-2024-0375
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0375
Dependency tree:
atty 0.2.14
└── env_logger 0.9.3
    └── firedancer-metrics-blocks 0.1.0

Crate:     atty
Version:   0.2.14
Warning:   unsound
Title:     Potential unaligned read
Date:      2021-07-04
ID:        RUSTSEC-2021-0145
URL:       https://rustsec.org/advisories/RUSTSEC-2021-0145
```

Due to the limited time frame allocated for this assessment, an holistic evaluation of the potential impact of the identified vulnerabilities was not possible. Consequently, the full extent of the implications remains unclear and requires further internal investigation by the development team at their earliest convenience.

Furthermore, achieving robust supply chain security presents a significant challenge. Optimal solutions are often complex and not readily apparent. Additionally, the effectiveness and results of the chosen protection framework can vary considerably depending on the specific libraries implemented and their respective versions.

To mitigate this issue, Cure53 recommends upgrading all affected libraries and establishing a policy to ensure that they remain updated moving forward.

ASY-02-003 WP1: Subpar server startup error handling *(Info)*

While reviewing the Rust-based firedancer-metrics source repository, Cure53 observed that runtime errors are caught and recorded as an event in the ClickHouse database's invalid table. However, errors that occur during server startup result in a panic state, ultimately causing the application to crash.

Depending on the server's configuration, these crashes may prevent the application from restarting or generate excessive core dumps, potentially leading to a DoS condition.

Affected files:

- *firedancer-metrics/srv/src/migrate.rs*
- *firedancer-metrics/srv/src/migrate/describe.rs*

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Affected code:

```
pub async fn migrate(client: &Client, schemas: &HashMap<String, Schema>) {
    let exists: bool = client.query("EXISTS DATABASE?")
       .bind(Identifier(DATABASE))
       .with_option("wait_end_of_query", "1")
       .fetch_one().await.unwrap();

    [...]
}
```

Affected code:

```
pub async fn describe(client: &Client, database: &str, table: &str) ->
ChTable {
    let columns = client
        .query(
            "
            SELECT ?fields
            FROM system.columns
            WHERE database = ?
            AND table = ?
        ",
        )
        .bind(database)
        .bind(table)
        .fetch_all::<RawColumn>()
        .await
        .unwrap();

    [...]
}
```

To mitigate this issue, Cure53 recommends implementing a refined shutdown process similar to typical correlating server operations. Alternatively, a revised approach would be to defer startup until the error conditions are resolved.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

ASY-02-004 WP1: Potential panic in SOL leader schedule fetching code *(Low)*

During the static review of the firedancer-metrics repository, Cure53 verified that the firedancer-metrics-block binary initializes inserter and retriever threads, which are responsible for retrieving SOL blocks and inserting them into ClickHouse. Additionally, a scheduler has been established to receive the current finalized block, compute missing blocks, and send finalized blocks to the inserters for database insertion. Under specific conditions, the scheduler retrieves the SOL leader schedule from the ClickHouse database without verifying the error code. If the database is unavailable, this omission can force the scheduler thread into a panic state.

Affected file:
firedancer-metrics/blocks/src/main.rs

Affected code:

```
async fn database_leader_schedule(&mut self, slot: u64) -> (Vec<Pubkey>,
Vec<usize>) {
    #[derive(clickhouse::Row, serde::Deserialize)]
    struct Row {
        slot: u64,
        leader: String,
    }
    let mut cursor = self.client.query("SELECT slot, leader FROM
leader_schedule WHERE epoch = ? AND cluster = 'mainnet' ORDER BY slot ASC")
        .bind(slot/432000)
        .fetch::<Row>().unwrap();

    let mut leaders = Vec::new();
    let mut schedule = vec![0; 432000];
    while let Some(row) = cursor.next().await.unwrap() {
        let bytes: [u8; 32] =
bs58::decode(&row.leader).into_vec().unwrap().as_slice().try_into().unwrap(
);
        let leader = Pubkey(bytes);
        let slot = row.slot;

        match leaders.iter().position(|x| *x == leader) {
            Some(idx) => schedule[(slot%432000) as usize] = idx,
            None => {
                schedule[(slot%432000) as usize] = leaders.len();
                leaders.push(leader);
            }
        }
    }

    [...]
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

To mitigate this issue, Cure53 recommends implementing robust error handling to manage scenarios whereby the connection to the configured ClickHouse instance is unavailable, hence preventing any resulting panic.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Conclusions

This Q1 2025 Cure53 investigation of the Metrics API developed by Firedancer concludes with a highly favorable verdict, owing to the almost negligible sum of findings and low median impact of the discovered weaknesses.

Prior to commencing proceedings, the internal maintainers supplied all essential information and materials required for the evaluations, including source code access, key focus areas, and additional insights regarding the wider threat model. A test server URL was initially provided to the testing team, although access was blocked by firewall restrictions. Consequently, the testers built a server application and launched a local instance in tandem with a ClickHouse database for examination purposes, for which the dev team provided support and guidance during the local setup process.

The backend is written in Rust, which is an excellent choice for secure system and application development. The source codebase is relatively concise and hence proved straightforward to assess in a meticulous manner. A review of the code confirmed that the Firedancer Metrics API was developed with evident consideration of security and defensive programming practices. This is further demonstrated by the integration of fuzz testing techniques, which are included in the source code repository and were shared by Firedancer.

Cure53 scrutinized the provided fuzzing harnesses and verified that they were ideally composed, with no enhancement opportunities to recommend.

Moreover, the Rust-based backend was primarily subjected to static source code analysis to ascertain its conformance with general cybersecurity paradigms. Additionally, the backend API was assessed for common vulnerabilities that could impact security and reliability, including:

- Potential HTTP server misconfigurations
- Liveness issues affecting the HTTP server and/or database
- Severe latency problems in the HTTP server and/or database
- Web 2.0 attack vectors, including SQLi and related threats
- Improper use of cryptographic APIs
- Leakage of sensitive information, potentially exposing confidential data

These factors were systematically appraised to ensure that the Firedancer Metrics API adheres with security best practices. Here, Cure53 positively noted that the backend application is resilient in relation to the aforementioned risks. Only four hardening vectors were discovered that would benefit from improvement, although these do not pose an immediate security threat.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Firstly, Cure53 located a potential SQL injection pathway in the Python scripts included within the Firedancer source code repository. Since exploitability relies on presumably trusted servers returning malicious input, this shortcoming was classified as *Miscellaneous* only (see ASY-02-001).

Next, Cure53 noted an opportunity to elevate internal third-party library management protocols, as certain software dependencies were outdated and vulnerable (see ASY-02-002). Lastly, the error handling process during server startup can be enhanced (see ASY-02-003) and a potential panic situation affecting the firedancer-metrics-blocks scheduler should be resolved (see ASY-02-004).

Overall, the Cure53 testers achieved satisfactory coverage over the work package and Rust code. All relevant breach scenarios, including those specifically highlighted by Firedancer, were thoroughly explored.

Notably, the backend API does not implement traditional authentication or authorization mechanisms. Instead, clients interacting with the API must sign the JSON payload using their private identity key and include the resulting signature in the *Authorization* header. To guarantee data integrity, all information ingested into the ClickHouse database (not into the public-facing API directly) is validated against this signature prior to acceptance.

Given the scant number of identified issues and low risk implications, Cure53 has opted to include a dedicated *Test Methodology* chapter to underscore the extent of the analytical process.

To offer one final piece of advice, Cure53 believes that committing to repeated security assessments over a sustained period of time will offer myriad advantages. Ideally, penetration tests and source code audits of products such as the Firedancer Metrics API should be performed at least once a year or in the event of substantial platform modifications.

Cure53 would like to thank Anthony Laou Hine Tsuei and Nick Lang from the Firedancer team for their excellent project coordination, support, and assistance, both before and during this assignment.