Code Review of RSTUF
for the Open Source Technology Improvement Fund (OSTIF)

Final Report and Management Summary

2025-03-07

PUBLIC

X41 D-Sec GmbH
Krefelder Str. 123
D-52070 Aachen
Amtsgericht Aachen: HRB19989

https://x41-dsec.de/
info@x41-dsec.de

Organized by the Open Source Technology Improvement Fund

Revision	*Date*	*Change*	*Author(s)*
1	2025-02-14	Final Report and Management Summary	L. Gommans, E. Sesterhenn, A. Basma

Contents

Dashboard

Target	
Customer	Open Source Technology Improvement Fund (OSTIF)
Name	Repository Service for TUF
Type	Server Applications
Version	Commit 9ec018d
Engagement	
Type	White Box
Consultants	3: Ali Basma, Eric Sesterhenn and Luc Gommans
Engagement Effort	22 person-days, 2025-01-30 to 2025-02-14
Total issues found	0

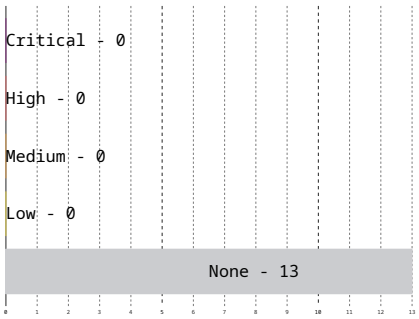

Figure 1: Issue Overview (l: Severity, r: CWE Distribution)

1 Executive Summary

In February 2025, X41 D-Sec GmbH conducted a source code audit of RSTUF to identify vulnerabilities and weaknesses in the software, based on a developed threat model that is included in this report. The work is sponsored by the Open Source Technology Improvement Fund (OSTIF) as part of their ongoing efforts to secure the open source world.

No direct security issues were discovered during this test, but 13 informational notes were identified that can be resolved to further harden the system.

Repository Service for TUF (RSTUF) implements the server components needed to create a repository from which clients that implement TUF (The Update Framework) can download files, safe against tampering between the repository and the client.

In a source code audit, all information about the system is made available to the testers. The test was performed by three experienced security experts between 2025-01-30 and 2025-02-14.

Several outdated components are specified in the deployment guides and dependencies, including services such as Redis and libraries such as Jinja. Most components are not directly exposed to the Internet which limits the exploitability. Still, an attacker who compromised a different part of the infrastructure could abuse any known vulnerabilities.

The deployments could further be hardened by defining which interfaces need to be exposed to other containers. Redis and PostgreSQL are currently open to all other containers (including the public web server) in the Docker setup. This could be restricted to only the necessary exposure by adding network isolation and setting secure passwords. Enabling encrypted communications further hardens the system when the containers are deployed on different physical hosts.

Finally, it was noticed that no time limit is set for tasks. The REST API component submits tasks via a message queue to the Worker component. If such a task gets stuck in an infinite loop or otherwise takes too long, this might lead to a denial of service situation. However, no way was found to exploit this potential issue.

Overall, the systems appear to maintain a strong security posture compared to systems of similar

size and complexity. While it is understood that rate-limiting and authentication have to be added to the RSTUF setup, it is crucial that these functions are implemented securely. X41 furthermore recommends to apply updates and hardening to the system.

2 Introduction

The Update Framework[1] (TUF) is a specification designed to securely deliver software assets to clients. Servers that implement TUF[2] need to cryptographically sign metadata files, certifying what the size and contents of the files should be. Trust is grounded in a set of 'root' keys that can be provided out of band.

Repository Service for TUF (RSTUF) is an implementation for server-side aspects of the framework, providing an API[3] for services that wish to provide assets for clients that implement TUF. It aims to facilitate deployments of any size; for example, for PyPI[4] to be able to provide signatures for its clients per the TUF specification.

X41 performed a source code audit of RSTUF, including setup documentation. Security issues in RSTUF or a deployment environment could enable an attacker to distribute malicious versions of the software or other assets.

2.1 Methodology

X41 reviewed RSTUF as defined in the scope below. The review was mainly based on a source code review alongside analyses of local installations of the software. A manual approach for penetration tests and for code reviews is used by X41. This process is supported by tools such as static code analyzers and industry standard web application security tools[5].

X41 adheres to established standards for source code reviewing and penetration testing. These are in particular the *CERT Secure Coding*[6] standards and the *Study - A Penetration Testing Model*[7]

[1] `https://theupdateframework.org`
[2] The Update Framework
[3] Application Programming Interface
[4] `https://pypi.org`
[5] `https://portswigger.net/burp`
[6] `https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards`
[7] `https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/Penetration/penetration_pdf.pdf?__blob=publicationFile&v=1`

of the German Federal Office for Information Security.

The workflow of source code reviews is shown in figure 2.1. In an initial, informal workshop regarding the design and architecture of the application a basic threat model is created. This is used to explore the source code for interesting attack surface and code paths. These are then audited manually and with the help of tools such as static analyzers. The identified issues are documented and can be used in a GAP analysis to highlight changes to previous audits.

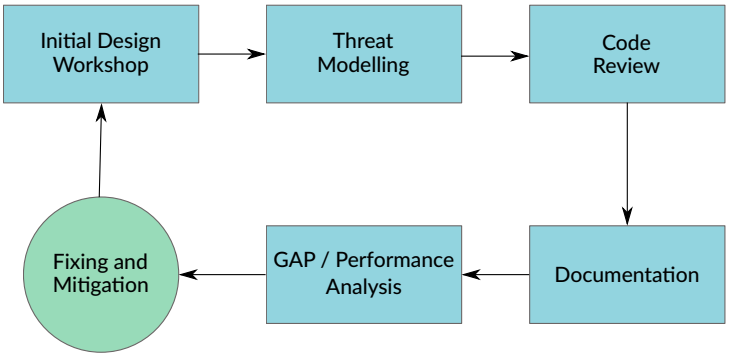

Figure 2.1: Code Review Methodology

2.2 Findings Overview

DESCRIPTION	SEVERITY	ID	REF
Outdated Jinja	NONE	RSTUF-CR-25-100	5.2.1
Outdated Docker Containers	NONE	RSTUF-CR-25-101	5.2.2
Docker Hardening	NONE	RSTUF-CR-25-102	5.2.3
Outdated Documentation	NONE	RSTUF-CR-25-103	5.2.4
Incorrect CSV Parsing	NONE	RSTUF-CR-25-104	5.2.5
Setup Issue on Linux OS	NONE	RSTUF-CR-25-105	5.2.6
No Task Time Limit	NONE	RSTUF-CR-25-106	5.2.7
SSL Not Enabled for Broker	NONE	RSTUF-CR-25-107	5.2.8
RSTUF Ceremony Requests a Password for Unencrypted Keys and Crashes on Incorrect Input	NONE	RSTUF-CR-25-108	5.2.9
Missing Network Segmentation for Docker Example Setup	NONE	RSTUF-CR-25-109	5.2.10
Missing Update Mechanism in Example Deployments	NONE	RSTUF-CR-25-110	5.2.11
Processes Run as Privileged User	NONE	RSTUF-CR-25-111	5.2.12
Internal Server Error on localstack	NONE	RSTUF-CR-25-112	5.2.13

Table 2.1: Security-Relevant Findings

2.3 Scope

The audit was performed against commit *9ec018d*[8] of the Repository Service for TUF source code. Including submodules, this includes around 19 000 lines of Python code according to `cloc`[9], including tests.

This GitHub repository is referred to as the *umbrella repository* because it includes three components as submodules, namely:

- `https://github.com/repository-service-tuf/repository-service-tuf-cli`
  Commit *44a6542cb6*
  A command-line interface to the API, meant as reference implementation.
- `https://github.com/repository-service-tuf/repository-service-tuf-api`
  Commit *18a8135fdd*
  The REST[10] API which queues tasks for the worker.
- `https://github.com/repository-service-tuf/repository-service-tuf-worker`
  Commit *204e0adbbb*
  The TUF implementation which performs the tasks from the queue.

[8] `https://github.com/repository-service-tuf/repository-service-tuf/tree/9ec018dadeb08f7ec8ecd5ba0a2e89c322236eb9`
[9] `https://github.com/AlDanial/cloc`
[10] Representational State Transfer

The repositories also include documentation, and this is also in scope. The rendered version of the ReStructuredText can be viewed on *Read the Docs*:

`https://repository-service-tuf.readthedocs.io`

A Signal chat group was created for efficient communication between the developer and the testers.

2.4 Coverage

A security code audit attempts to find the most important or sometimes as many of the existing problems as possible, though it is practically never possible to rule out the possibility of additional weaknesses being found in the future.

The time allocated to X41 for this assessment was sufficient to yield a good coverage of the given scope.

The source code was audited manually and inspected with the help of Semgrep[11] and Bandit[12]. The codebase was audited for best practices in the use of Celery, following recommendations documented by the Celery project[13].

The RSTUF API was tested for proper input validation against common attack vectors such as injection attacks, parameter tampering, malformed requests and SSRF[14]. All JSON[15] payloads received by the API were checked for proper validation. URL[16] parameters and query strings were tested for length, type, and value constraints. This helps ensure that unexpected or malicious values cannot bypass backend logic. X41 evaluated the API error handling mechanisms to ensure that error messages do not disclose sensitive internal details (such as stack traces or internal configurations) that could aid an attacker. The API requires rate limiting and DoS[17] resilience to ensure that it remains stable and secure under high load or attack conditions. The endpoints were also tested for path traversal vulnerabilities by supplying various manipulated paths and observing if unauthorized directories or files could be accessed. Furthermore, the code was inspected for file and database-based race conditions. The database was thoroughly tested for SQL[18] injection vulnerabilities using sqlmap[19]. No exploitable SQL injection vectors were identified, indicating that the input validation and query parameterization controls in place

[11] `https://semgrep.dev/`
[12] `https://github.com/PyCQA/bandit`
[13] `https://docs.celeryq.dev/en/stable/userguide/security.html`
[14] Server-Side Request Forgery
[15] JavaScript Object Notation
[16] Uniform Resource Locator
[17] Denial of Service
[18] Structured Query Language
[19] `https://github.com/sqlmapproject/sqlmap`

are effective.

2.5 Recommended Further Tests

X41 recommends to audit actual running instances of RSTUF as well, since basic areas of its security rely on the surrounding infrastructure. RSTUF does not implement authentication, rate-limiting, CSP[20] or other defensive security measures. Furthermore, the infrastructure it runs on needs proper network separation and firewalling.

[20] Content Security Policy

3 Rating Methodology

Security vulnerabilities are given a purely technical rating by the testers when they are discovered during a test. Business factors and financial risks for Open Source Technology Improvement Fund (OSTIF) are beyond the scope of a penetration test, which focuses entirely on technical factors. However, technical results from a penetration test may be an integral part of a general risk assessment. A penetration test is based on a limited time frame and only covers vulnerabilities and security issues which have been found in the given time, there is no claim for full coverage.

The CVSS[1] is used to score all findings relevant to security. The resulting CVSS score is mapped to qualitative ratings as shown below.

3.1 CVSS

All findings relevant to security are rated by the testers using the CVSS industry standard version 3.1, revision 1.

Vulnerabilities scored with CVSS get a numeric value based on several metrics ranging from 0.0 (least worst) to 10.0 (worst).

The score captures different factors that express the impact and the ease of exploitation of a vulnerability among other factors. For a detailed description of how the scores are calculated, please see the CVSS version 3.1 specification.[2]

The metrics used to calculate the final score are grouped into three different categories.

[1] Common Vulnerability Scoring System
[2] `https://www.first.org/cvss/v3-1/cvss-v31-specification_r1.pdf`

The *Base Metric Group* represents the intrinsic and fundamental characteristics of a vulnerability that are constant over time and user environments. It captures the following metrics:

- Attack Vector (AV)
- Attack Complexity (AC)
- Privileges Required (PR)
- User Interaction (UI)
- Scope (S)
- Confidentiality Impact (C)
- Integrity Impact (I)
- Availability Impact (A)

The *Temporal Metric Group* represents the characteristics of a vulnerability that change over time but not among user environments. The following metrics are covered by it:

- Exploitability (E)
- Remediation Level (RL)
- Report Confidence (RC)

The *Environmental Metric Group* represents the characteristics of a vulnerability that are relevant and unique to a particular user's environment. It includes the following metrics:

- Attack Vector (MAV)
- Attack Complexity (MAC)
- Privileges Required (MPR)
- User Interaction (MUI)
- Confidentiality Requirement (MCR)
- Integrity Requirement (MIR)
- Availability Requirement (MAR)
- Scope (MS)
- Confidentiality Impact (MC)
- Integrity Impact (MI)
- Availability Impact (MA)

A CVSS vector defines a specific set of metrics and their values, and it can be used to reproduce and assess a given score. It is rendered as a string that exactly reproduces a score.

For example, the vector `CVSS:3.1/AV:N/AC:H/PR:L/UI:R/S:C/C:H/I:L/A:N` defines a base score metric with the following parameters:

- Attack Vector: Network
- Attack Complexity: High
- Privileges Required: Low
- User Interaction: Required
- Scope: Changed
- Confidentiality Impact: High
- Integrity Impact: Low
- Availability Impact: None

In this example, a network-based attacker performs a complex attack after gaining access to some privileges, by tricking a user into performing some actions. This allows the attacker to read confidential data and change some parts of that data.

The detailed scores are the following:

Metric	Score
CVSS Base Score	6.5
Impact Sub-Score	4.7
Exploitability Sub-Score	1.3
CVSS Temporal Score	Not Available
CVSS Environmental Score	Not Available
Modified Impact Sub-Score	Not Available
Overall CVSS Score	6.5

CVSS vectors can be automatically parsed to recreate the score, for example, with the CVSS calculator provided by FIRST, the organization behind CVSS: `https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:N/AC:H/PR:L/UI:R/S:C/C:H/I:L/A:N`.

3.2 Severity Mapping

To help in understanding the results of a test, numeric CVSS scores are mapped to qualitative ratings as follows:

Severity Rating	CVSS Score
NONE	0.0
LOW	0.1–3.9
MEDIUM	4.0–6.9
HIGH	7.0–8.9
CRITICAL	9.0–10.0

3.3 Common Weakness Enumeration

The CWE[3] is a set of software weaknesses that allows vulnerabilities and weaknesses in software to be categorized. If applicable, X41 gives a CWE ID for each vulnerability that is discovered during a test.

CWE is a very powerful method for categorizing a vulnerability. It gives general descriptions and solution advice on recurring vulnerability types. CWE is developed by *MITRE*.[4] More information can be found on the CWE site at `https://cwe.mitre.org/`.

[3] Common Weakness Enumeration
[4] `https://www.mitre.org`

4 Threat Model

A threat model enumerates potential threats against a system in order to systematically analyze them. The model allows reasoning about the security aspects of the system and, in a next step, verifying that safeguards exist against relevant threats. It can be considered an evolving document that can be extended over time.

RSTUF provides an implementation of the server-side aspects of TUF, but does not provide an authentication or authorization mechanism to protect any asset. The administrator deploying RSTUF is responsible for identifying which systems need protecting and implementing this adequately[1].

[1] https://repository-service-tuf.readthedocs.io/en/latest/guide/deployment/planning/deployment.html#rstuf-best-practices

4.1 Asset Overview

The various components outside RSTUF and their interaction are shown in figure 4.1.

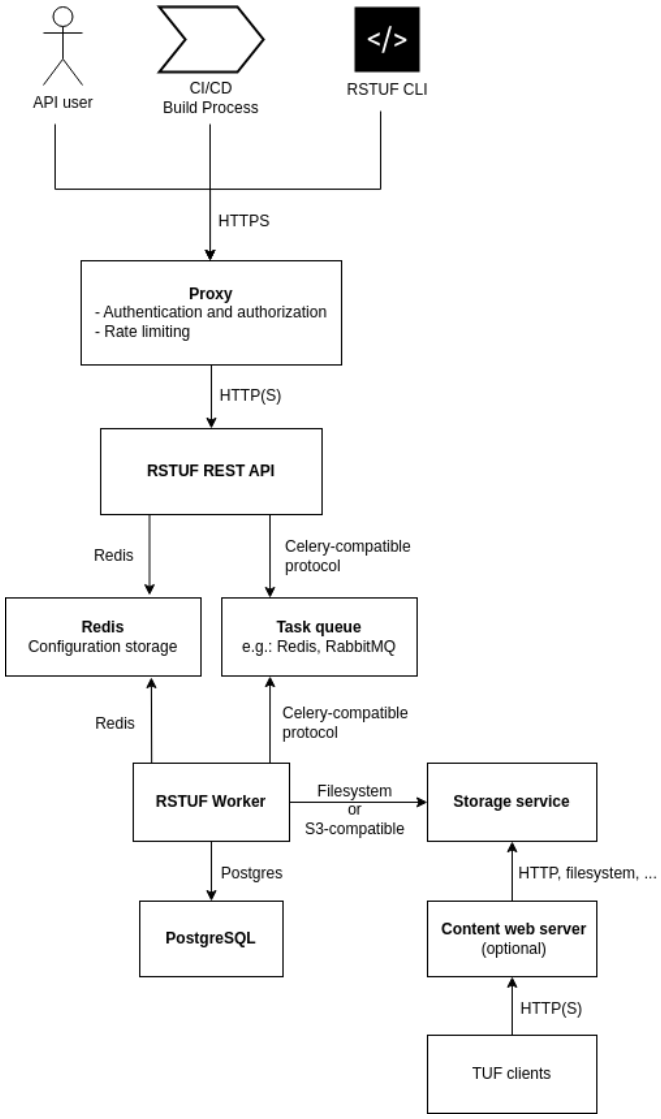

Figure 4.1: Architecture

4.1.1 TUF Clients

RSTUF's purpose is to certify artifacts towards TUF clients. While the clients themselves are not part of RSTUF or this audit, they interface with the service provided by RSTUF.

4.1.2 REST API Users

Users of the REST API service provide commands, data, and signatures. Some of the cryptographic keys used with RSTUF are stored on a system managed by these users.

Automated systems, such as a CI/CD[2] system that submits artifacts, may also make use of the REST API and is therefore considered a REST API user.

4.1.3 Storage System

This system stores the TUF metadata and needs to be exposed to TUF clients[3]. It can store data locally in a container volume or AWS[4] S3[5] buckets.

4.1.4 REST API Endpoint

The API REST service is meant to be the interface with which external systems communicate to trigger actions in the Worker[6].

4.1.5 Worker

The worker manages TUF metadata and updates, signs, and publishes it[7].

4.1.6 Message Queue Redis / RabbitMQ

The API endpoints and Workers communicate through tasks sent into a message broker[8] such as Redis or RabbitMQ.

[2] Continous Integration/Continous Delivery
[3] https://repository-service-tuf.readthedocs.io/en/v0.13.0/guide/deployment/planning/deployment.html#storage-backend-service
[4] Amazon Web Services
[5] Simple Storage Service
[6] https://github.com/repository-service-tuf/repository-service-tuf-worker/
[7] https://repository-service-tuf.readthedocs.io/en/latest/guide/repository-service-tuf-worker/index.html#repository-service-for-tuf-worker
[8] https://repository-service-tuf.readthedocs.io/en/latest/guide/deployment/planning/deployment.html#required-infrastructure-services

4.1.7 Result Backend Redis

This Redis instance should be configured to use persistent storage and saves the configuration and results of RSTUF[9].

4.1.8 Database PostgreSQL Server

The database server is used by the Worker to store and manage metadata[10].

4.1.9 Content Web Server

This is an optional component that provides TUF metadata generated by the Workers to TUF clients[11], such as Python-TUF[12].

4.1.10 Key Service

Key services are an optional component that can manage cryptographic keys, such that an API user does not have to store the keys in another way, such as by using files.

[9] https://repository-service-tuf.readthedocs.io/en/v0.13.0/guide/deployment/planning/deployment.html#result-backend-and-rstuf-settings-redis-server

[10] https://repository-service-tuf.readthedocs.io/en/v0.13.0/guide/deployment/planning/deployment.html#database-postgres-server

[11] https://repository-service-tuf.readthedocs.io/en/v0.13.0/guide/deployment/planning/deployment.html#optional-content-server-webserver

[12] https://github.com/theupdateframework/python-tuf

4.2 Security Boundaries

RSTUF communicates with and consists of several entities with various trust levels. Some of these systems (REST API, Worker, message queue, result backend, database) run in one of the aforementioned containers. In accordance with the documented requirements, it is assumed that they are properly firewalled and not accessible to unauthorized parties.

4.2.1 TUF Clients

The TUF client implementations are generally untrusted and can only read information from the Content Web Server or equivalent. If the information is confidential (for example, internal to a company), these clients may authenticate, but it is not foreseen that they are ever in a privileged position with regards to the RSTUF services.

4.2.2 REST API Users

Users of the REST API are always privileged, but privilege levels can differ depending on the type of system. People that manage the service will be administrative and generally of the highest permission level, but an automated system that pushes updates could be only partially trusted. A trust boundary still exists between having privileged access to the REST API and being an administrator of the server on which the REST API runs.

There can also be a trust boundary between the mechanism that the user uses to store the key and the computer that talks to the REST API, for example: allowing the computer to request a signature when the correct PIN[13] is provided but not to obtain the actual private key.

Connections to the API should make use of transport security (such as TLS[14]) because they are likely to cross an untrusted or less-trusted network.

4.2.3 Storage System

The storage system will receive files from the worker and store them. It may also serve these files to TUF users directly if no other microservice is implemented for that. The files can be served via different protocols and mechanisms such as HTTPS[15], FTP[16] and others.

[13] Personal Identification Number
[14] Transport Layer Security
[15] HyperText Transfer Protocol Secure
[16] File Transfer Protocol

4.2.4 REST API Endpoint

The REST API is trusted to interpret information sent to it and submit it to the task queue for the Worker to act upon asynchronously. Only trusted clients should have access to it.

4.2.5 Worker

The Worker is a key component, since it performs the actual TUF tasks, has access to the storage system, key service, database, and broker. A compromised Worker would affect most parts of the RSTUF infrastructure.

4.2.6 Message Queue Redis / RabbitMQ

The message queue is a central component and can trigger privileged actions in the Worker. It also might have access to key material send via broker messages.

4.2.7 Result Backend Redis

The backend Redis system stores configuration from the RSTUF Worker and provides settings and result data to the REST API system.

4.2.8 Database PostgreSQL Server

The PostgreSQL server is used by the RSTUF Worker to store data by using SQLAlchemy[17].

4.2.9 Content Web Server

Metadata files from the storage system can be pushed to the content web server, or that server may pull the results from the storage system.

4.2.10 Key Service

The Worker fetches the online key from the key service as necessary.

[17] `https://www.sqlalchemy.org/`

4.3 Threats and Mitigations

The following sections describe threats and available or known mitigations related to various aspects of the system.

4.3.1 TUF Clients

Compromised or malicious clients could perform DoS attacks or perform attacks against the web server, exploiting issues in the HTTPS protocol.

4.3.2 REST API Users

Compromised or malicious clients could perform DoS attacks or perform attacks against the web server, exploiting issues in the HTTPS protocol. Furthermore, they could tamper with the configuration to elevate privileges or inject data into the broker communication.

The cryptographic keys that API users employ may be incorrectly or weakly generated. This risk can be reduced by providing guidance on how to safely generate the keys. The same key may also be used multiple times. The RSTUF CLI[18] implementation does not allow using copies of the same key to reach a threshold value (quorum).

Keys may be stolen when they are stored insecurely by clients. A smart card can store keys in a tamper-resistant manner and protect them with a PIN or other mechanism. Because the key is hard to extract from the smart card (or perhaps impossible with current techniques), it is very hard for an attacker to persist access to it without being noticed. However, this cannot prevent a compromised client system that tampers with the metadata before it is sent to the smart card for signing. Another mechanism against the theft of keys is to require signatures from multiple, separately held keys for critical operations, as RSTUF encourages.

Some keys are expected to be stored on the server in order to provide fast updates. See the key service subsection below for threats in this case.

4.3.3 Storage System

The storage system could create DoS situations by not serving files or causing timeouts. Additionally, it could serve outdated or malicious files. Attacks could be performed against the Worker by

[18] Command line interpreter

responding to its writing of files onto the storage system in a malicious manner (e.g., by hanging the write calls).

4.3.4 REST API Endpoint

These clients could perform DoS attacks, exploit issues in the HTTPS or REST implementation or the business logic of the RSTUF API implementation. A compromised API endpoint could lead to malicious signatures on files.

To reduce the risk of DoS, rate limiting per client could be employed on a reverse proxy (sometimes called an API gateway). RSTUF does not need to implement this directly in its code; instead, the project documentation could specify how to configure existing tools for this purpose.

4.3.5 Worker

The Worker is able to attack the key server, storage service, broker and PostgreSQL database by sending malicious data to either system. It is a central component which is trusted by the other systems. Therefore, a compromise would be critical and can, depending on the setup, cause the system to be insecure for a period of time, even if this may be recoverable using offline keys. RSTUF provides guidance on how to recover from a key compromise situation[19].

Audit logging on the Worker can improve the response to a compromise, helping find the root cause.

If too many tasks are submitted via the queue, the Worker could become overloaded and unable to perform all tasks. Limiting the inflow and monitoring the amount of outstanding tasks can mitigate this risk.

4.3.6 Message Queue Redis / RabbitMQ

Access to the message queue is similar to having access to the API, but where the API could filter nonsensical data, this would be submittable to the message queue directly. An attacker compromising the message queue increases the attack surface on the Worker, but otherwise is similar to having full API access.

[19] `https://repository-service-tuf.readthedocs.io/en/stable/guide/general/usage.html#recovering-a-compromised-key`

4.3.7 Result Backend Redis

The backend Redis database can attack the API and Worker system by returning malicious answers to trigger logic issues in these systems. Furthermore, security vulnerabilities in the Redis communication could be abused. Since the system is central, DoS scenarios are a given.

4.3.8 Database PostgreSQL Server

The SQL server could perform attacks against the RSTUF Worker by returning malicious data or abusing the Postgres protocol.

4.3.9 Content Web Server

There is little trust vested in the web server. It serves metadata files to client, but the client verifies its signatures. It may have access to the storage system to retrieve the files which it needs to serve, but there is no confidential information in that system. The most realistic threat is that the service is blocked in a way that updates become unavailable until an administrator intervenes.

4.3.10 Key Service

A compromise of the key service allows an attacker to forge signatures. Depending on the setup, this may amount to less-important metadata files such as the timestamp document, or constitute a complete compromise of signature trustworthiness on any of the protected artifacts.

4.3.11 Attacks from Compromised CI/CD

Artifacts will often be coming from a continuous integration and delivery system. While these talk to the REST API and the risks of a malicious client are discussed above, it is also relevant to realize that this is part of the trust chain. If the build system is compromised, the attacker may not attack the REST API or any other component at all, but could simply build malicious versions of artifacts and let the normal operations run their course with nobody the wiser. Reproducible builds may aid in detecting this risk. The specifics of the build process outside of RSTUF are beyond the scope of this threat model.

4.3.12 Hosting

A compromise of the infrastructure RSTUF is running on will compromise the security of the system and the keys stored thereon. Failing to update the containers with security patches could lead to such a compromise. Installing tainted dependencies that are supplied by a malicious actor via a supply chain attack would lead to compromise as well. An RSTUF developer being compromised could lead to a malicious update.

4.4 Limitations

Since the RSTUF code cannot protect against the types of attacks documented in subsection 4.3.12, they will not be considered during the code audit. Furthermore, keys or clients that are compromised in other ways cannot be protected by RSTUF itself. Since no authorization, authentication or permission handling is implemented in RSTUF at all, this audit cannot investigate security issues that are related to these features.

5 Results

This chapter describes the results of this test. The security-relevant findings are documented in Section 5.1. Additionally, findings without a direct security impact are documented in Section 5.2.

5.1 Findings

No security releated issues were identified during the audit.

5.2 Informational Notes

The following observations do not have a direct security impact, but are related to security hardening, affect functionality, or other topics that are not directly related to security. X41 recommends to mitigate these issues as well, because they often become exploitable in the future. Doing so will strengthen the security of the system and is recommended for defense in depth.

5.2.1 RSTUF-CR-25-100: Outdated Jinja

Affected Component:	requirements.txt

5.2.1.1 Description

Jinja version 3.1.4 is vulnerable to CVE-2024-56201[1], which allows the execution of Python code when attackers are able to control the template filename and contents. This dependency is pulled in via the *requirements.txt* file as shown in listing 5.1.

```
1 jinja2==3.1.4; python_version >= '3.7'
```

Listing 5.1: requirements.txt

This is considered an informational finding, since it seems that RSTUF does not grant users this access.

5.2.1.2 Solution Advice

X41 recommends to update all dependencies to the latest version whenever possible.

[1] https://nvd.nist.gov/vuln/detail/CVE-2024-56201

5.2.2 RSTUF-CR-25-101: Outdated Docker Containers

Affected Component: docker-compose.yml and others

5.2.2.1 Description

The various *Dockerfiles* depend on outdated versions of software. Among them is Redis version 4.0, which is vulnerable to at least CVE-2019-10193[2], CVE-2019-10192[3], CVE-2018-12326[4], and CVE-2018-12453[5], the latter of which allows an attacker to cause a DoS via the `XGROUP` command. Additionally, the dependency on RabbitMQ 3.10.7 introduces vulnerability CVE-2021-32718[6], where the management web interface does not correctly filter *<script>* tags. The Dockerfile for Redis is shown in listing 5.2.

```
redis:
  image: redis:4.0
  volumes:
    - repository-service-tuf-redis-data:/data
  ports:
    - "6379:6379"
```

Listing 5.2: Dockerfile Including Redis

Similarly, all *docker-compose*.yml* files hardcode a container based on Debian Buster (*3.10-slim-buster*), as shown in listing 5.3. Buster was superseded by Bullseye in August 2021. Since RSTUF's initial commit is from April 2022, this might have been copied from an old project without evaluating the suitability. Buster does not receive mainstream security updates anymore[7] since June 2022, and the long-term support security updates provided by volunteers were discontinued in June 2024. When pinning to Python 3.10 is required, a *slim* container is available that does not depend on an OS[8] version: *3.10-slim*.

[2] https://nvd.nist.gov/vuln/detail/CVE-2019-10193
[3] https://nvd.nist.gov/vuln/detail/CVE-2019-10192
[4] https://nvd.nist.gov/vuln/detail/CVE-2018-12326
[5] https://nvd.nist.gov/vuln/detail/CVE-2018-12453
[6] https://nvd.nist.gov/vuln/detail/CVE-2021-32718
[7] https://www.debian.org/releases/buster/
[8] Operating System

```
web:
  image: python:3.10-slim-buster
  command: python -m http.server -d /var/opt/repository-service-tuf/storage 8080
  volumes:
    - repository-service-tuf-storage:/var/opt/repository-service-tuf/storage
  ports:
    - "8080:8080"
```

Listing 5.3: Buster-based Image Used for Public-Facing Web Service

This is considered an informational issue, since these vulnerabilities can not be exploited from outside the system.

5.2.2.2 Solution Advice

X41 recommends upgrading all containers to the latest versions and introduce processes to ensure that this happens on a regular basis.

5.2.3 RSTUF-CR-25-102: Docker Hardening

Affected Component:	docker-compose.yml

5.2.3.1 Description

The *docker-compose.yml* file specifies various services without restricting new privileges or writing to the file system.

Adding the option `no-new-privileges: true` prevents any `setuid` binaries from escalating privileges to root inside the container.

The containers that need persistent storage appear to write to specific bind mounts. Because no container appears to need to write to the root file system, setting `read-only: true` helps restrict what payloads an attacker could run.

This also applies to `docs/source/guide/deployment/guide/docker-compose.yml`, although that file specifies that the PostgreSQL configuration specifically does not follow best practices and should not be used in production.

5.2.3.2 Solution Advice

X41 recommends to use safe defaults in example service configuration files and documentation.

5.2.4 RSTUF-CR-25-103: Outdated Documentation

Affected Component:	docs/source/guide/deployment/setup.rst, docs/source/guide/repository-service-tuf-cli/index.rst, repository-service-tuf-cli/docs/source/guide/index.rst, docs/source/guide/deployment/guide/docker-compose.yml

5.2.4.1 Description

The CLI was changed in July 2024 when deprecated features were removed[9], which was released with version `v0.13.0b1`, but the documentation still refers to the `admin-legacy` subcommand in several files:

- *docs/source/guide/deployment/setup.rst*
- *docs/source/guide/repository-service-tuf-cli/index.rst*
- *repository-service-tuf-cli/docs/source/guide/index.rst*

In the same month, several variables were renamed from containing `_SQL_` to containing `_DB_` (commit `8c8959a0`). This is not yet reflected in many of the source files and documentation examples, such as the Docker setup instructions[10] at the time of writing.

These are not security risks but noticed as part of the audit.

5.2.4.2 Solution Advice

X41 recommends reviewing documentation when changing inputs and (command line) interfaces.

[9] `https://github.com/repository-service-tuf/repository-service-tuf-cli/pull/635`
[10] `https://repository-service-tuf.readthedocs.io/en/latest/guide/deployment/guide/docker.html`

5.2.5 RSTUF-CR-25-104: Incorrect CSV Parsing

Affected Component:	repository-service-tuf-cli/repository_service_tuf/cli/admin/import_artifacts.py:_parse_csv_data()

5.2.5.1 Description

In function ***_parse_csv_data()*** in *import_artifacts.py* a file is treated as CSV[11] and parsed in an open-coded manner. When one of the lines contains a semicolon, the parser will break. These lines are usually escaped by quotes. The parsing code is shown in listing 5.4.

```
1 with open(csv_file, "r") as f:
2     for line in f:
3         path = line.split(";")[0]
4         length = int(line.split(";")[1])
5         hash_algorithm = line.split(";")[2]
6         hash_digest = line.split(";")[3]
```

Listing 5.4: Bad CSV Parsing

5.2.5.2 Solution Advice

X41 recommends to use a robust Python CSV library[12].

[11] Comma Separated Values
[12] `https://docs.python.org/3/library/csv.html`

5.2.6 RSTUF-CR-25-105: Setup Issue on Linux OS

Affected Component:	https://repository-service-tuf.readthedocs.io/en/latest/guide/deployment/guide/quick-start.html

5.2.6.1 Description

During the installation process, X41 encountered an issue where the setup was unable to reach the expected endpoints. Investigation revealed that `Ingress` could not map `localhost` to the correct internal IP[13] address, preventing proper communication. After the development team applied a fix, the issue was resolved, and the endpoints became accessible.

This highlights the possibility of setup-related problems across different operating systems, where specific configurations may be required to ensure proper functionality and avoid unexpected issues.

5.2.6.2 Solution Advice

X41 recommends ensuring proper system configuration and testing to prevent unexpected deployment issues across different operating systems.

[13] Internet Protocol

5.2.7 RSTUF-CR-25-106: No Task Time Limit

Affected Component:	repository-service-tuf-api/repository_service_tuf_api/__init__.py

5.2.7.1 Description

No time limit[14] is set on celery tasks. When these get stuck in an infinite loop or require overly long to perform their task, this might lead to DoS situations.

Since no ways were identified to create a DoS situation, this is considered an informational finding and should be considered an opportunity for hardening the setup.

5.2.7.2 Solution Advice

X41 recommends to set the time limits for all tasks.

[14] `https://docs.celeryq.dev/en/stable/userguide/configuration.html#std-setting-task_time_limit`

5.2.8 RSTUF-CR-25-107: SSL Not Enabled for Broker

Affected Component:	repository-service-tuf-worker/app.py, repository-service-tuf-api/repository_service_tuf_api/__init__.py

5.2.8.1 Description

SSL[15]/TLS is not enabled for connections to the broker service. Depending on the setup, attackers could intercept connections to the broker and add malicious messages into the stream.

Since the network between the broker and the agents connecting to it should be firewalled and not accessible to third parties this is considered an informational issue. Nevertheless, using SSL in these setups would increase security in depth.

5.2.8.2 Solution Advice

X41 recommends to setup SSL for broker connections[16] by default and recommend using them in the best practices documentation[17].

[15] Secure Sockets Layer

[16] `https://docs.celeryq.dev/en/stable/userguide/configuration.html#broker-use-ssl`

[17] `https://repository-service-tuf.readthedocs.io/en/stable/guide/deployment/planning/deployment.html`

5.2.9 RSTUF-CR-25-108: RSTUF Ceremony Requests a Password for Unencrypted Keys and Crashes on Incorrect Input

Affected Component:	repository-service-tuf-cli/repository_service_tuf/cli/admin/helpers.py:_load_signer_from_file_prompt()

5.2.9.1 Description

During the execution of the `rstuf admin ceremony` command, after specifying the path to an unencrypted key, the tool still prompts for a password. Since the key does not require one, this prompt is unexpected. If a random password is entered, the client crashes instead of handling the input gracefully as shown in listing 5.5.

```
Please enter [yellow]path[/] to encrypted private key '[green]
  <securesystemslib.signer._key.SSlibKey object at 0x75ed13f2e1d0>[/]'/home/
  user/Desktop/repository-service-tuf/keys/keypair.pem

Please enter password to encrypted private key 'ali1':
Traceback (most recent call last):
  File "/home/user/.local/bin/rstuf", line 8, in <module>
    sys.exit(rstuf())
             ^^^^^^^
  File "/home/user/.local/pipx/venvs/repository-service-tuf/lib/python3.11/
  site-packages/rich_click/rich_command.py", line 367, in __call__
    return super().__call__(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/user/.local/pipx/venvs/repository-service-tuf/lib/python3.11/
  site-packages/click/core.py", line 1161, in __call__
    return self.main(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/user/.local/pipx/venvs/repository-service-tuf/lib/python3.11/
  site-packages/rich_click/rich_command.py", line 152, in main
    rv = self.invoke(ctx)
         ^^^^^^^^^^^^^^^^
  File "/home/user/.local/pipx/venvs/repository-service-tuf/lib/python3.11/
  site-packages/click/core.py", line 1697, in invoke
    return _process_result(sub_ctx.command.invoke(sub_ctx))
                           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/user/.local/pipx/venvs/repository-service-tuf/lib/python3.11/
  site-packages/click/core.py", line 1697, in invoke
    return _process_result(sub_ctx.command.invoke(sub_ctx))
                           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/user/.local/pipx/venvs/repository-service-tuf/lib/python3.11/
  site-packages/click/core.py", line 1443, in invoke
    return ctx.invoke(self.callback, **ctx.params)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
  File "/home/user/.local/pipx/venvs/repository-service-tuf/lib/python3.11/
  site-packages/click/core.py", line 788, in invoke
    return __callback(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "/home/user/.local/pipx/venvs/repository-service-tuf/lib/python3.11/
  site-packages/click/decorators.py", line 33, in new_func
    return f(get_current_context(), *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "/home/user/.local/pipx/venvs/repository-service-tuf/lib/python3.11/
  site-packages/repository_service_tuf/cli/admin/ceremony.py", line 137, in
  ceremony
    _add_root_signatures_prompt(root_md, None)
  File "/home/user/.local/pipx/venvs/repository-service-tuf/lib/python3.11/
  site-packages/repository_service_tuf/cli/admin/helpers.py", line 633, in
  _add_root_signatures_prompt
    _add_signature_prompt(root_md, key_choices[choice])
  File "/home/user/.local/pipx/venvs/repository-service-tuf/lib/python3.11/
  site-packages/repository_service_tuf/cli/admin/helpers.py", line 589, in
  _add_signature_prompt
    signer = _load_signer_from_file_prompt(key)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "/home/user/.local/pipx/venvs/repository-service-tuf/lib/python3.11/
  site-packages/repository_service_tuf/cli/admin/helpers.py", line 254, in
  _load_signer_from_file_prompt
    private_key = load_pem_private_key(private_pem, password.encode())
                  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

TypeError: Password was given but private key is not encrypted.
```

Listing 5.5: Command Output: rstuf admin ceremony

This behavior may indicate a flaw in the tool key handling logic, potentially affecting the usability and reliability of the setup process. Unexpected crashes could lead to an incomplete or failed administrative setup.

5.2.9.2 Solution Advice

X41 recommends to review and correct the key management logic to ensure that unencrypted keys do not trigger a password prompt and to implement proper error handling to prevent crashes when incorrect input is provided.

5.2.10 RSTUF-CR-25-109: Missing Network Segmentation for Docker Example Setup

Affected Component:	docs/source/guide/deployment/guide/docker.rst

5.2.10.1 Description

The Docker deployment guide[18] does not isolate services from each other. For example, the container housing the REST API needs to be able to submit tasks to Redis for the Worker to pick up, but other containers such as the Content Web Server[19] do not need this access. In the current documentation example, an attacker who compromised the (typically) public-facing web server now gets access to submit tasks that only administrators were supposed to be able to submit via the API.

The PostgreSQL example setup uses an insecure default password (`secret`). If the system is only accessible for the Worker by some isolating mechanism, then this has no impact because authentication is already taken care of by that mechanism. The missing network segmentation allows any container in the network to access the PostgreSQL database with full permissions.

5.2.10.2 Solution Advice

X41 recommends hardening the example setup by segmenting the network such that services only have the access they need.

[18] `https://repository-service-tuf.readthedocs.io/en/latest/guide/deployment/guide/docker.html`
[19] `https://repository-service-tuf.readthedocs.io/en/latest/guide/deployment/planning/deployment.html#optional-content-server-webserver`

5.2.11 RSTUF-CR-25-110: Missing Update Mechanism in Example Deployments

Affected Component:	https://repository-service-tuf.readthedocs.io/en/latest/guide/deployment/index.html

5.2.11.1 Description

The deployment guides[20] provide no mechanism to apply security updates to the different running services. It is also not clear how the availability of security updates will be communicated to system administrators.

It is likely that security vulnerabilities will at some point be found in the components used in the containers after deployment. This may affect custom code (the Worker or API) or third-party components such as Redis, PostgreSQL, or the content web server.

While not all components are directly exposed to the Internet, an attacker could use any vulnerabilities for escalating partial system access.

5.2.11.2 Solution Advice

When providing a deployment guide, X41 recommends to cover security updates. The deployment examples could be configured to automatically take care of updates, or they can specify that such a mechanism still needs to be set up.

For custom services specifically (the REST API and Worker), it helps administrators to define where security advisories will be shared such that they know to expect and apply the update.

[20] `https://repository-service-tuf.readthedocs.io/en/latest/guide/deployment/index.html`

5.2.12 RSTUF-CR-25-111: Processes Run as Privileged User

Affected Component: Helm and Docker Setup

5.2.12.1 Description

Processes in the minikubes containers[21] and Docker setup[22] run with elevated privileges. Among others, `uvicorn` and `celery` run with `root` privileges as shown in listing 5.6.

```
$ kubectl exec  rstuf-rstuf-api-7986c8c57d-ss9rv -n rstuf --  egrep '(U|G)id|Name' /proc/*/status
↪   2> /dev/null
...
/proc/7/status:Name:    uvicorn
/proc/7/status:Uid:    0    0    0    0
/proc/7/status:Gid:    0    0    0    0
...

$ kubectl exec  rstuf-rstuf-worker-85676c8c84-csrgn -n rstuf --  egrep '(U|G)id|Name'
↪  /proc/*/status  2> /dev/null
/proc/15/status:Name:    supervisord
/proc/15/status:Uid:    0    0    0    0
/proc/15/status:Gid:    0    0    0    0
/proc/16/status:Name:    celery
/proc/16/status:Uid:    0    0    0    0
/proc/16/status:Gid:    0    0    0    0
/proc/17/status:Name:    celery
/proc/17/status:Uid:    0    0    0    0
/proc/17/status:Gid:    0    0    0    0
...
```

Listing 5.6: Processes Running as root User

This is considered informational and should be considered as a hardening measure since it is not a security issue in itself, but could increase the impact of another security issue.

5.2.12.2 Solution Advice

X41 recommends to run all processes with the least amount of privileges possible.

[21] `https://repository-service-tuf.readthedocs.io/en/latest/guide/deployment/guide/quick-start.html`

[22] `https://repository-service-tuf.readthedocs.io/en/latest/guide/deployment/guide/docker.html`

5.2.13 RSTUF-CR-25-112: Internal Server Error on localstack

Affected Component:	http://localstack.local/shell

5.2.13.1 Description

When RSTUF is installed via the quickstart guide[23], accessing `http://localstack.local/shell` will result in an internal server error and return a stack trace (shown in listing 5.7). Other paths starting with */shell* are affected as well. This leaks debug data to attackers.


```
{
   "__type" : "InternalError",
   "message" : "exception while calling dynamodb with unknown operation: Traceback (most recent
   ↪  call last):\n  File
   ↪  \"/opt/code/localstack/localstack-core/localstack/aws/protocol/parser.py\", line 173, in
   ↪  wrapper\n    return func(*args, **kwargs)\n           ^^^^^^^^^^^^^^^^^^^^^\n  File
   ↪  \"/opt/code/localstack/localstack-core/localstack/aws/protocol/parser.py\", line 920, in
   ↪  parse\n    target = request.headers[\"X-Amz-Target\"]\n
   ↪  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\n  File \"/opt/code/localstack/.venv/lib/python3.11/site-p
   ↪  ackages/werkzeug/datastructures/headers.py\", line 83, in __getitem__\n    return
   ↪  self._get_key(key)\n           ^^^^^^^^^^^^^^^^^^\n  File \"/opt/code/localstack/.venv/lib
   ↪  /python3.11/site-packages/werkzeug/datastructures/headers.py\", line 97, in _get_key\n
   ↪  raise BadRequestKeyError(key)\nwerkzeug.exceptions.BadRequestKeyError: 400 Bad Request:
   ↪  The browser (or proxy) sent a request that this server could not understand.\n\nThe above
   ↪  exception was the direct cause of the following exception:\n\nTraceback (most recent call
   ↪  last):\n  File
   ↪  \"/opt/code/localstack/.venv/lib/python3.11/site-packages/rolo/gateway/chain.py\", line
   ↪  166, in handle\n    handler(self, self.context, response)\n  File
   ↪  \"/opt/code/localstack/localstack-core/localstack/aws/handlers/service.py\", line 63, in
   ↪  __call__\n    return self.parse_and_enrich(context)\n
   ↪  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\n  File
   ↪  \"/opt/code/localstack/localstack-core/localstack/aws/handlers/service.py\", line 67, in
   ↪  parse_and_enrich\n    operation, instance = parser.parse(context.request)\n
   ↪          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\n  File
   ↪  \"/opt/code/localstack/localstack-core/localstack/aws/protocol/parser.py\", line 177, in
   ↪  wrapper\n    raise UnknownParserError(\nlocalstack.aws.protocol.parser.UnknownParserError:
   ↪  An unknown error occurred when trying to parse the request.\n"
}
```


Listing 5.7: Internal Server Error

[23] https://repository-service-tuf.readthedocs.io/en/stable/guide/deployment/guide/quick-start.html

This is considered informational since production systems might be set up differently and it is not caused by RSTUF code.

5.2.13.2 Solution Advice

X41 recommends to disable all debug settings in RSTUF installations.

6 About X41 D-Sec GmbH

X41 D-Sec GmbH is an expert provider for application security and penetration testing services. Having extensive industry experience and expertise in the area of information security, a strong core security team of world-class security experts enables X41 D-Sec GmbH to perform premium security services.

X41 has the following references that show their experience in the field:

- Source code audit of ISC BIND9 DNS server[1]
- Source code audit of the Git source code version control system[2]
- Review of the Mozilla Firefox updater[3]
- X41 Browser Security White Paper[4]
- Review of Cryptographic Protocols (Wire)[5]
- Identification of flaws in Fax Machines[6,7]
- Smartcard Stack Fuzzing[8]

The testers at X41 have extensive experience with penetration testing and red teaming exercises in complex environments. This includes enterprise environments with thousands of users and vendor infrastructures such as the Mozilla Firefox Updater (Balrog).

Fields of expertise in the area of application security encompass security-centered code reviews, binary reverse-engineering and vulnerability-discovery. Custom research and IT security consulting, as well as support services, are the core competencies of X41. The team has a strong technical background and performs security reviews of complex and high-profile applications such as Google Chrome and Microsoft Edge web browsers.

X41 D-Sec GmbH can be reached via `https://x41-dsec.de` or `mailto:info@x41-dsec.de`.

[1] `https://x41-dsec.de/news/security/research/source-code-audit/2024/02/13/bind9-security-audit/`
[2] `https://x41-dsec.de/security/research/news/2023/01/17/git-security-audit-ostif/`
[3] `https://blog.mozilla.org/security/2018/10/09/trusting-the-delivery-of-firefox-updates/`
[4] `https://browser-security.x41-dsec.de/X41-Browser-Security-White-Paper.pdf`
[5] `https://www.x41-dsec.de/reports/Kudelski-X41-Wire-Report-phase1-20170208.pdf`
[6] `https://www.x41-dsec.de/lab/blog/fax/`
[7] `https://2018.zeronights.ru/en/reports/zero-fax-given/`
[8] `https://www.x41-dsec.de/lab/blog/smartcards/`

Acronyms

API Application Programming Interface . . . 6
AWS Amazon Web Services . . . 17
CI/CD Continous Integration/Continous Delivery . . . 17
CLI Command line interpreter . . . 21
CSP Content Security Policy . . . 10
CSV Comma Separated Values . . . 31
CVSS Common Vulnerability Scoring System . . . 11
CWE Common Weakness Enumeration . . . 14
DoS Denial of Service . . . 9
FTP File Transfer Protocol . . . 19
HTTPS HyperText Transfer Protocol Secure . . . 19
IP Internet Protocol . . . 32
JSON JavaScript Object Notation . . . 9
OS Operating System . . . 27
PIN Personal Identification Number . . . 19
REST Representational State Transfer . . . 8
S3 Simple Storage Service . . . 17
SQL Structured Query Language . . . 9
SSL Secure Sockets Layer . . . 34
SSRF Server-Side Request Forgery . . . 9
TLS Transport Layer Security . . . 19
TUF The Update Framework . . . 6
URL Uniform Resource Locator . . . 9