# Source Code Audit on The Update Framework for Open Source Technology Improvement Fund (OSTIF)

**Final Report and Management Summary**

2022-09-09

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

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

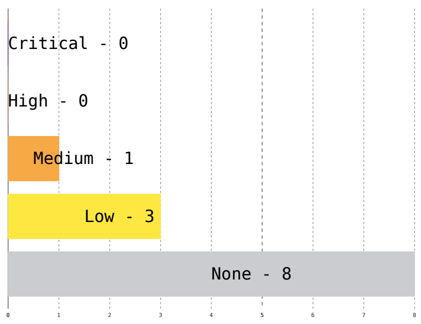
Organized by the Open Source Technology Improvement Fund

| *Revision* | *Date* | *Change* | *Author(s)* |
|---|---|---|---|
| 1 | 2022-08-25 | Final Report | E. Sesterhenn, L. Gommans, N. Abel, Y. Klawohn |
| 2 | 2022-09-09 | Set status to public | L. Gommans |

# Contents

# Dashboard

| | |
|---|---|
| **Target** | |
| Customer | Open Source Technology Improvement Fund (OSTIF) |
| Name | The Update Framework |
| Type | Source Code and Specification |
| Version | v2.0.0 (7ada2af384442f1c3775a4bcbd9f33270c1c8fa0) |
| | |
| **Engagement** | |
| Type | Source Code Audit |
| Consultants | 4: Eric Sesterhenn, Luc Gommans, Niklas Abel and Yasar Klawohn |
| Engagement Effort | 18 person-days, 2022-08-08 to 2022-08-19 |
| | |
| Total issues found | 4 |



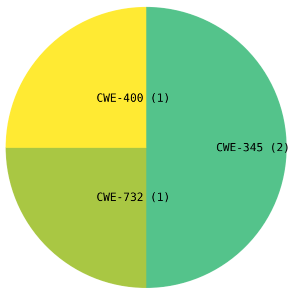


**Figure 1:** Issue Overview (l: Severity, r: CWE Distribution)

# 1 Executive Summary

In August 2022, X41 D-Sec GmbH performed a source code audit against The Update Framework (TUF) to identify vulnerabilities and weaknesses in the source code and specification. The test was organized by the Open Source Technology Improvement Fund[1].

A total of four vulnerabilities were discovered during the audit by X41. None were rated as having a critical or high severity, one as medium, and three as low. Additionally, eight issues without a direct security impact were identified.

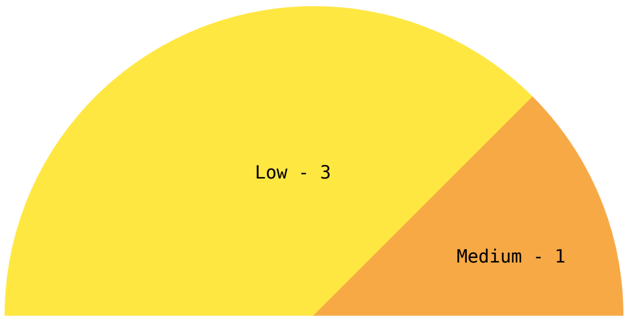


**Figure 1.1:** Issues and Severity

The Update Framework is a specification with a reference implementation designed to provide developers with a framework for delivering updates to their applications securely. Vulnerabilities in the specification or an implementation could allow an attacker to take over the update mechanism of many other projects, deliver malicious updates, and gain control of the targeted systems.

[1] `https://ostif.org`

In a source code audit, the testers receive all available information about the target. The test was performed by four experienced security experts between 2022-08-08 and 2022-08-19.

The most severe issue discovered pertains to file permissions set on private key files when using the basic usage instructions. An attacker who already has access to the local system as another user could read these files, for example as a tenant on a shared server system, and use the keys to sign false updates.

Another weakness exists in Python's JSON parser, where an attacker could supply a document at the size limit and trigger the parser to hang for a certain amount of time. The time depends on the size limit and CPU speed; by default, the time is on the order of a few minutes. This might frustrate users and cause them to abort the update process, leaving them on an old release.

Several other improvements were identified to improve defense-in-depth and reduce ambiguity in the specification.

X41 recommends to apply further hardening to the code and to evaluate supply chain attacks risks in further tests. Such attacks could, for example, occur due to a compromise of the GitHub repository, which is considered fully trusted by the project team.

Overall, the project shows a high maturity in terms of security. Having a specification as a human-readable description of what the code should do makes it possible to better reason about and more easily verify the code.

# 2 Introduction

X41 reviewed The Update Framework, consisting of a specification[1] and a Python reference implementation. The system allows developers to securely manage updates of software and to minimize the impact on users in the event of a supply chain security event.

It is intended to be widely used, which would make it an important target for any supply chain attacks on other projects.

## 2.1 Scope

The main scope and focus of this work is:

- The implementation of the detailed client workflow of the TUF specification[2],
- the safety of the ngclient library[3]
- and the Metadata API[4].

Excluded from the scope are dependencies. These consist of the well-known *`requests`* library, as well as a package called *`securesystemslib`* which is developed by an overlapping set of developers.

A particular concern expressed by the developers is the JSON[5] parsing, since this is a relatively complex operation and operates on untrusted input from the network.

[1] `https://theupdateframework.github.io/specification/v1.0.30/index.html`
[2] `https://theupdateframework.github.io/specification/v1.0.30/index.html`
[3] `https://theupdateframework.readthedocs.io/en/latest/api/tuf.ngclient.html`
[4] `https://theupdateframework.readthedocs.io/en/latest/api/tuf.api.html`
[5] JavaScript Object Notation

## 2.2 Findings Overview

| DESCRIPTION | SEVERITY | ID | REF |
|---|---|---|---|
| Private Key World-Readable | MEDIUM | TUF-CR-22-01 | 4.1.1 |
| Shallow Build Artifact Verification | LOW | TUF-CR-22-02 | 4.1.2 |
| Quadratic Complexity in JSON Number Parsing | LOW | TUF-CR-22-03 | 4.1.3 |
| Release Signatures Add No Protection | LOW | TUF-CR-22-04 | 4.1.4 |
| File Not Closed Or Flushed | NONE | TUF-CR-22-100 | 4.2.1 |
| GitHub 2FA Guidelines | NONE | TUF-CR-22-101 | 4.2.2 |
| PyPI 2FA Guidelines | NONE | TUF-CR-22-102 | 4.2.3 |
| Permissive Verification | NONE | TUF-CR-22-103 | 4.2.4 |
| Update Cycle Ambiguity in Specification | NONE | TUF-CR-22-104 | 4.2.5 |
| Cleanup Procedure Not Specified | NONE | TUF-CR-22-105 | 4.2.6 |
| Unversioned Cryptographic Primitives | NONE | TUF-CR-22-106 | 4.2.7 |
| Branch Protection Security | NONE | TUF-CR-22-107 | 4.2.8 |

**Table 2.1:** Security-Relevant Findings

## 2.3 Coverage

A security assessment 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.

A manual approach for code review is used by X41. This process is supported by tools such as static code analyzers and industry standard security tools.

The time allocated to X41 for this code review was sufficient to yield a reasonable coverage of the given scope. Covered topics include:

- JSON parsing (identified by the developers as a point of concern)
- General Python coding problems, using various static analysis tools
- ‘Getting started’ documentation
- File system permissions
- Third parties which the project depends on (both software and services) and how these risks can be reduced
- Release verification
- Setting invalid threshold values to manipulate the signature validation

- Cryptographic primitives used
- Practical testing of computational complexity, download size limits, and download time limits

Suggestions for next steps in securing this scope can be found in section 2.4.

## 2.4 Recommended Further Tests

It is recommended to do a source code audit on the parts that TUF uses from the dependency `securesystemslib`. Supply chain vulnerabilities here may cascade to TUF and hardening advice provided in this report could also be applied to the dependency's project settings.

# 3 Rating Methodology for Security Vulnerabilities

Security vulnerabilities are given a purely technical rating by the testers as they are discovered during the 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. Yet 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.

In total, five different ratings exist, which are as follows:

| Severity Rating |
|---|
| None |
| Low |
| Medium |
| High |
| Critical |

A low rating indicates that the vulnerability is either very hard for an attacker to exploit due to special circumstances, or that the impact of exploitation is limited, whereas findings with a medium rating are more likely to be exploited or have a higher impact. High and critical ratings are assigned when the testers deem the prerequisites realistic or trivial and the impact significant or very significant.

Findings with the rating 'none' are called side findings and 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.

## 3.1 Common Weakness Enumeration

The CWE[1] is a set of software weaknesses that allows the categorization of vulnerabilities and weaknesses in software. If applicable, X41 provides the CWE-ID for each vulnerability that is discovered during a test.

CWE is a very powerful method to categorize a vulnerability and to give general descriptions and solution advice on recurring vulnerability types. CWE is developed by *MITRE*[2]. More information can be found on the CWE website at `https://cwe.mitre.org/`.

[1] Common Weakness Enumeration
[2] `https://www.mitre.org`

# 4 Results

This chapter describes the results of this test. The security-relevant findings are documented in Section 4.1. Additionally, findings without a direct security impact are documented in Section 4.2.

## 4.1 Findings

The following subsections describe findings with a direct security impact that were discovered during the test.

### 4.1.1 TUF-CR-22-01: Private Key World-Readable

| | |
|---|---|
| *Severity:* | MEDIUM |
| *CWE:* | 732 – Incorrect Permission Assignment for Critical Resource |
| *Affected Component:* | https://gist.github.com/lukpueh/40e19dd54ab0c020954ad8236ec4e953 |

#### 4.1.1.1 Description

The usage example at `https://gist.github.com/lukpueh/40e19dd54ab0c020954ad8236ec4e953` generates the private key file without setting its file permissions.

By default, the ***securesystemslib.interface.generate_and_write_unencrypted_ed25519_keypair()*** function creates files which are readable for every user on the system, so that all users and services can read the private key information of the local system. The code is shown in listing 4.1.

The default file permissions depend on the *umask* which is set to "world readable" on most Linux distributions.

```
# Create one key pair per role
# NOTE: For some roles, e.g. root, usually a threshold of multiple signing keys is used
for role_name in roles.keys():
    path_pub = (KEY_DIR / role_name).as_posix()
    path_priv = generate_and_write_unencrypted_ed25519_keypair(path_pub)
    keys[role_name] = import_ed25519_privatekey_from_file(path_priv)
```

**Listing 4.1:** Private Key Generation Without Setting File Permissions

#### 4.1.1.2 Solution Advice

The ***generate_and_write_unencrypted_ed25519_keypair()*** function does not allow to set file permissions for the generated files. X41 recommends to create empty files for the private keys and set their file permissions prior to generating the keys, setting only read and write permissions for the file owner.

### 4.1.2 TUF-CR-22-02: Shallow Build Artifact Verification

| | |
|---|---|
| *Severity:* | LOW |
| *CWE:* | 345 – Insufficient Verification of Data Authenticity |
| *Affected Component:* | verify_release |

#### 4.1.2.1 Description

In the *verify_release* script, in the root of the repository, the functions ***verify_github_release()*** and ***verify_pypi_release()*** only perform a shallow comparison of the directories by using ***filecmp.dircmp()***[1].

> “[. . . ] The dircmp class compares files by doing shallow comparisons as described for filecmp.cmp(). [. . . ]”[2]

The documentation for ***filecmp.cmp()*** is as follows:

> “ [. . . ] If shallow is `true` and the ***os.stat()*** signatures (file type, size, and modification time) of both files are identical, the files are taken to be equal.
>
> Otherwise, the files are treated as different if their sizes or contents differ. [. . . ]”

The ***verify_github_release()*** function is shown in listing 4.2. The relevant parts of ***verify_github_release()*** and ***verify_pypi_release()*** do not differ, which is why the listing of the latter is omitted.

Since only file metadata is compared, inequality of file contents is not detected.

```python
def verify_github_release(version: str, compare_dir: str) -> bool:
    """Verify that given GitHub version artifacts match expected artifacts"""
    base_url = (
        f"https://github.com/{GITHUB_ORG}/{GITHUB_PROJECT}/releases/download"
    )
    tar = f"{PYPI_PROJECT}-{version}.tar.gz"
    wheel = f"{PYPI_PROJECT}-{version}-py3-none-any.whl"
    with TemporaryDirectory() as github_dir:
        for filename in [tar, wheel]:
            url = f"{base_url}/v{version}/{filename}"
            response = requests.get(url, stream=True)
            with open(os.path.join(github_dir, filename), "wb") as f:
                for data in response.iter_content():
```

[1] https://docs.python.org/3/library/filecmp.html
[2] https://docs.python.org/3/library/filecmp.html#filecmp.dircmp

```
                f.write(data)

    same = dircmp(github_dir, compare_dir).same_files
    return sorted(same) == [wheel, tar]
```

**Listing 4.2:** Shallow File Verification

#### 4.1.2.2 Solution Advice

X41 recommends to use ***filecmp.cmp()*** for file comparisons instead, with the optional *shallow* parameter set to *False*.

### 4.1.3 TUF-CR-22-03: Quadratic Complexity in JSON Number Parsing

| | |
|---|---|
| *Severity:* | LOW |
| *CWE:* | 400 – Uncontrolled Resource Consumption ('Resource Exhaustion') |
| *Affected Component:* | tuf/api/serialization/json.py |

#### 4.1.3.1 Description

Parsing numbers in JSON using Python's built-in *json* library takes a quadratic amount of time (the time taken doubles while the input size increases linearly):

| Number Size | Time taken |
|---|---|
| 1 MiB | 3.6 seconds |
| 2 MiB | 14.1 seconds |
| 3 MiB | 32.7 seconds |
| 4 MiB | 57.9 seconds |

**Table 4.1:** Number Size vs. Parsing Time

Note that the same value as string finishes parsing instantly.

An attacker could abuse this property by modifying the update files to include a huge number. Because the JSON is parsed before signature verification can happen, the attacker needs no signing keys.

The standard limit for a metadata file is 5 million bytes, coming out at about 2 minutes of CPU[3] time on a modern system. Such a value can be included in every JSON file that the client will download. Memory seems to be constant during this time.

While the project does not aim to run on embedded systems, older systems might experience a freeze for a good while. The user might try to interrupt the process due to locking up the system with no apparent progress or clear error message and forego updating. Because the huge number can be included in an unsigned part of the JSON object, this would not cause any error and the user (even if they patiently let it run) does not know to sound any alarm.

To reproduce, a JSON file could be modified to include a huge number or this command could be used:

```
$ time python3 -c 'import json; json.loads("["+("9"*1000*1000*5)+"]")'
$ time pypy -c 'import json; json.loads("["+("9"*1000*1000*5)+"]")'
```

[3] Central Processing Unit

**Listing 4.3:** Python Number Parsing

There is no option for the built-in Python deserializer to ignore numbers longer than a reasonable number of bytes (given that the maximum integer would be a few hundred digits). The GIL is presumably locked (even a regular Ctrl+C does not stop the progress) so a separate Python-native thread cannot monitor the parsing duration. Perhaps a regular expression could be constructed to detect the problem (note that these are also susceptible to denial of service when exponential backtracking is required), though that seems error-prone. Lowering the maximum document size comes with inherent trade-offs. Showing a warning to the user before starting the parsing progress is beyond the scope of the project.

#### 4.1.3.2 Solution Advice

If possible, X41 recommends to investigate why this problem occurs and resolve it in the interpreter.

### 4.1.4 TUF-CR-22-04: Release Signatures Add No Protection

| | |
|---|---|
| *Severity:* | LOW |
| *CWE:* | 345 – Insufficient Verification of Data Authenticity |
| *Affected Component:* | verify_release |

#### 4.1.4.1 Description

The *verify_release* script does a local build and, if the build metadata is equal to that of GitHub's build, creates signature files which can be attached to a release. In principle, verifying the reproducible build locally makes sense. However, the script builds from a fresh `git clone` without any verification. If the GitHub repository were to be compromised by any vector, the attacker would also just change the code and the verification would appear successful. The signatures do not add any protection.

In a discussion with the developers, GitHub is considered fully trusted (i.e., the TUF project does not attempt to protect from a scenario in which GitHub was compromised).

PyPI can contain signature files, but `pip` does not have a way to verify this by default[4]. If commit signing were to be implemented, users would have to use an alternative client like `twine` or verify the releases from GitHub to be sure they got the code as released by the developers.

#### 4.1.4.2 Solution Advice

X41 recommends to either verify releases using commit signing or to not add signatures to releases to prevent a false sense of end-to-end verification.

[4] `https://security.stackexchange.com/questions/232855/does-pythons-pip-provide-cryptographic-authentication-and-integrity-validation`

## 4.2 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.

### 4.2.1 TUF-CR-22-100: File Not Closed Or Flushed

| *Affected Component:* | tuf/ngclient/updater.py:_persist_metadata() |
|---|---|

#### 4.2.1.1 Description

In the function ***_persist_metadata()***, the value of *temp_file.name* is read before the file is closed or flushed. This could cause an error because the file may not exist when *temp_file.name* is used[5]. The code is shown in listing 4.4.

```
def _persist_metadata(self, rolename: str, data: bytes) -> None:
        """Write metadata to disk atomically to avoid data loss."""
        temp_file_name: Optional[str] = None
        try:
            # encode the rolename to avoid issues with e.g. path separators
            encoded_name = parse.quote(rolename, "")
            filename = os.path.join(self._dir, f"{encoded_name}.json")
            with tempfile.NamedTemporaryFile(
                dir=self._dir, delete=False
            ) as temp_file:
                temp_file_name = temp_file.name
                temp_file.write(data)
            os.replace(temp_file.name, filename)
        except OSError as e:
            # remove tempfile if we managed to create one,
            # then let the exception happen
            if temp_file_name is not None:
                try:
```

[5] https://registry.semgrep.dev/rule/python.lang.correctness.tempfile.flush.tempfile-without-flush

```
                    os.remove(temp_file_name)
                except FileNotFoundError:
                    pass
            raise e
```

**Listing 4.4:** Missing ***flush()*** Could Cause Errors

#### 4.2.1.2 Solution Advice

X41 recommends to use ***flush()*** before reading $temp\_file.name$.

### 4.2.2 TUF-CR-22-101: GitHub 2FA Guidelines

|  |  |
|---|---|
| *Affected Component:* | Developer Guidelines |

#### 4.2.2.1 Description

There seem to be no guidelines provided for what kind of 2FA[6] methods are to be used for GitHub accounts with privileged access to the repository. GitHub requires the first registered 2FA method to be either SMS[7] or TOTP[8]; only then, a phishing-resistant WebAuthn hardware tokens can be added (called "Security Keys" in GitHub). These include, for example, YubiKeys, Apple devices starting with their upcoming operating system updates, or Windows devices supporting "Windows Hello". One can then habitually log in with WebAuthn instead of the initially added SMS or TOTP factors.

What makes WebAuthn resistant against phishing is that a solution to a website's challenge is scoped to that website's domain. Say a user is led to e.g. `attacker.example.com`, which displays a `github.com` login form and forwards the real `github.com` WebAuthn challenge to the user. Now, even if the user authorizes their WebAuthn device to solve the challenge, the solution won't be accepted by `github.com`, because it would be bound to `attacker.example.com`[9].

#### 4.2.2.2 Solution Advice

Particularly for projects that might be used as part of a supply chain attack, X41 recommends to provide guidelines to developers for how to secure their GitHub accounts with phishing-resistant 2FA. Hardware tokens should be used at all times and TOTP as a fallback when all hardware tokens are lost. Notably, the fallback should not be used if a (potential phishing) page asks for it.

[6] Two Factor Authentication
[7] Short Message Service
[8] Time-based One-Time Password
[9] `https://community.ibm.com/community/user/security/blogs/shane-weeden1/2021/12/08/what-makes-fido-and-webauthn-phishing-resistent`

### 4.2.3 TUF-CR-22-102: PyPI 2FA Guidelines

| | |
|---|---|
| *Affected Component:* | Developer Guidelines |

#### 4.2.3.1 Description

There seem to be no guidelines provided for what kind of 2FA methods are to be used for PyPI. PyPI allows users to have phishing-resistant WebAuthn hardware tokens including, for example, YubiKeys, Apple devices starting with their upcoming operating system updates, or Windows devices supporting “Windows Hello”.

What makes WebAuthn resistant against phishing is that a solution to a website’s challenge is scoped to that website’s domain. Say a user is led to e.g. `attacker.example.com`, which displays a `github.com` login form and forwards the real `github.com` WebAuthn challenge to the user. Now, even if the user authorizes their WebAuthn device to solve the challenge, the solution won’t be accepted by `github.com`, because it would be bound to `attacker.example.com`[10].

#### 4.2.3.2 Solution Advice

Particularly for projects that might be used as part of a supply chain attack, X41 recommends to provide guidelines to developers for how to secure their PyPI accounts with phishing-resistant 2FA.

[10] `https://community.ibm.com/community/user/security/blogs/shane-weeden1/2021/12/08/what-makes-fido-and-webauthn-phishing-resistent`

### 4.2.4 TUF-CR-22-103: Permissive Verification

| *Affected Component:* | tuf/api/metadata.py |
|---|---|

#### 4.2.4.1 Description

The JSON data is canonicalized before signing or verifying. This means that various formats are accepted. The values shown in listing 4.5 are all considered equivalent and will verify correctly without having to re-sign the data.

```
{"example\r":"A"}
{"example\r":"B","example\u000d":"A"}
{"example\u000d":"A"}
```

**Listing 4.5:** Equivalent Formats

At *tuf/api/serialization/json.py:JSONSerializer:serialize()*, there is a parameter available, `validate`, which checks for this problem, but this is not enabled.

The testers did not find a way to exploit this in the project because there are no values in use where special characters would occur. Because no language other than Python is used in the scope, there are also no parsing differences from other libraries or languages. One possible scenario would be where a project is adding custom fields to the signed section of an appropriate metadata file and relying on the string value rather than the parsed value, or treating illegal UTF-8 encoding or Unicode code points differently. Allowing differences in parsing could be risky and the development team identified the JSON parsing as a particular concern.

#### 4.2.4.2 Solution Advice

X41 recommends to validate whether serialization and subsequent deserialization changes the values.

### 4.2.5 TUF-CR-22-104: Update Cycle Ambiguity in Specification

| *Affected Component:* | https://theupdateframework.github.io/specification/ |
|---|---|

#### 4.2.5.1 Description

Step 5 at `https://theupdateframework.github.io/specification/latest/#update-root` specifies (emphasis added):

> The version number of the trusted root metadata file (version N) MUST be less than the version number of the new root metadata file (version N+1). Effectively, this means checking that the version number signed in the new root metadata file is indeed N+1. If the version of the new root metadata file is less than the version of the trusted metadata file, discard it, abort the update cycle, and report the rollback attack. In case they are equal, again discard the new root metadata, but **proceed the update cycle** with the already trusted root metadata.

Proceeding with the update cycle could mean going to the next step (5.3.6), proceeding with the next metadata file since the root data is now considered trusted (e.g. proceed to 5.4), or jumping to another step in the root update procedure (e.g. 5.3.10, since step 5.3.3 indicates that this is the end of the loop).

The correct option is the latter one, otherwise some checks would be skipped (5.3.10–12) or an infinite loop would be possible by an unauthenticated attacker. The reference implementation appears to use the correct next step.

#### 4.2.5.2 Solution Advice

X41 recommends to clarify this point of the specification.

### 4.2.6 TUF-CR-22-105: Cleanup Procedure Not Specified

| | |
|---|---|
| *Affected Component:* | Specification |

#### 4.2.6.1 Description

The specification does not define at what point which files can be deleted. While metadata files are not going to be huge and require cleanup soon after release, users of the specification or reference implementation might eventually want to clear up space but it is not defined how this is to be achieved. Deleting all files older than X days might have security implications, as (for example) the root metadata files should form an unbroken chain.

#### 4.2.6.2 Solution Advice

X41 recommends to specify how long created files are relevant for.

### 4.2.7 TUF-CR-22-106: Unversioned Cryptographic Primitives

| *Affected Component:* | Specification |
|---|---|

#### 4.2.7.1 Description

Several places in the specification require using a specific hashing function without an available upgrade path. While this is not a problem for the foreseeable future from X41's perspective because the primitives are well-chosen, this might place TUF in a similar position as git today where an upgrade path might be desirable but hard to achieve.

Examples include SHA-256 use for key identifiers[11] and *path_hash_prefixes*[12].

Files do include a *SPEC_VERSION*[13] which allows changing every aspect of the format completely, but this seems less flexible than allowing to swap out an individual aspect. Some users may also wish to enable more experimental algorithms (for example for post-quantum safety) or algorithms encouraged by bodies other than NIST[14] (perhaps for legal compliance in their jurisdictions).

#### 4.2.7.2 Solution Advice

X41 recommends to include versioning, for example by prefixing hashes with an *1:* for the current (first) version.

[11] `https://theupdateframework.github.io/specification/latest/#role-keyid`
[12] `https://theupdateframework.github.io/specification/latest/#path_hash_prefixes`
[13] `https://theupdateframework.github.io/specification/latest/#spec_version`
[14] National Institute of Standards and Technology

### 4.2.8 TUF-CR-22-107: Branch Protection Security

| | |
|---|---|
| *Affected Component:* | GitHub Repository Settings |

#### 4.2.8.1 Description

GitHub allows configuring Branch Protection rules. These may seem to be a security feature that helps protect against an account compromise because it can be configured to require multiple reviews before being allowed to change the code in the repository. However, these rules can be bypassed by a repository administrator (also when ‘do not allow bypassing [by admins]’ is enabled) because the administrator can disable this setting without any confirmation from other repository collaborators. One can also invite dummy accounts as collaborators and get sign-offs on a pull request that way, though that would be more visible.

Based on the bypasses known to the testers at the time, it would be possible to use branch protection rules as security against a single account compromise if the repository administrator accounts are not in regular use. This way, the setting could not be temporarily disabled and no dummy collaborators could be invited without gaining access to this locked-away administrator account.

The TUF repository[15] currently has pull requests with at least one review required via branch protection settings, but not the setting that administrators cannot bypass it.

#### 4.2.8.2 Solution Advice

X41 recommends enabling the ‘Do not allow bypassing the above settings’ setting in the branch protection settings. Administrators can still bypass this in an emergency by disabling the setting again. No 2FA prompt seems to be needed for this, making the disabling even faster. The advantage is that collaborators with administrator permissions cannot accidentally commit something and forget to pass it through the usual review process, as this seems to be the default radio button selected when changing files using the GitHub UI[16].

X41 recommends to furthermore evaluate the merit of having a separate administrator account for the repository in order to better protect against individual account takeovers.

[15] `https://github.com/theupdateframework/python-tuf/`
[16] User Interface

# 5 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:

- Review of the Mozilla Firefox updater[1]
- X41 Browser Security White Paper[2]
- Review of Cryptographic Protocols (Wire)[3]
- Identification of flaws in Fax Machines[4,5]
- Smartcard Stack Fuzzing[6]

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://blog.mozilla.org/security/2018/10/09/trusting-the-delivery-of-firefox-updates/`
[2] `https://browser-security.x41-dsec.de/X41-Browser-Security-White-Paper.pdf`
[3] `https://www.x41-dsec.de/reports/Kudelski-X41-Wire-Report-phase1-20170208.pdf`
[4] `https://www.x41-dsec.de/lab/blog/fax/`
[5] `https://2018.zeronights.ru/en/reports/zero-fax-given/`
[6] `https://www.x41-dsec.de/lab/blog/smartcards/`

# Acronyms

| | | |
|---|---|---|
| **2FA** | Two Factor Authentication | 21 |
| **CPU** | Central Processing Unit | 16 |
| **CWE** | Common Weakness Enumeration | 10 |
| **JSON** | JavaScript Object Notation | 6 |
| **NIST** | National Institute of Standards and Technology | 26 |
| **SMS** | Short Message Service | 21 |
| **TOTP** | Time-based One-Time Password | 21 |
| **UI** | User Interface | 27 |