Source Code Audit on the in-toto Framework for Open Source Technology Improvement Fund (OSTIF)

Final Report and Management Summary

2023-05-16

PUBLIC

X41 D-SEC GmbH
Krefelderstr. 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	2023-02-24	Final Report	Eric Sesterhenn, J. M., and Luc Gommans
2	2023-03-16	Clarifications	Luc Gommans
3	2023-05-16	Public Report	Luc Gommans

Contents

Dashboard

Target	
Customer	Open Source Technology Improvement Fund (OSTIF)
Name	in-toto Framework
Type	Specification and Reference Implementations
Version	1.3.1 (Python implementation) and 0.6.0 (Go implementation)
Engagement	
Type	Source Code Audit
Consultants	3: Eric Sesterhenn, J. M. and Luc Gommans
Engagement Effort	24 person-days, 2023-02-06 to 2023-02-24
Total issues found	8

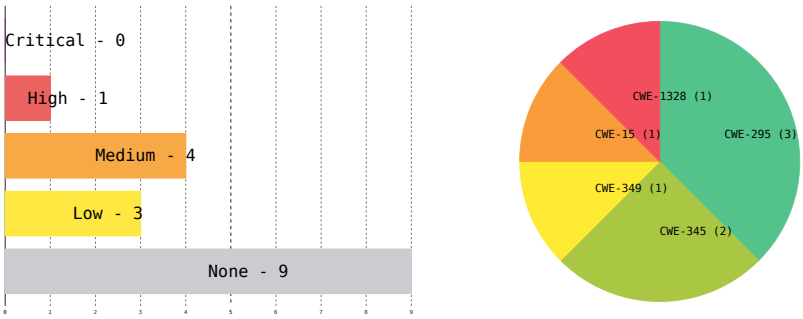

Figure 1: Issue Overview (l: Severity, r: CWE Distribution)

1 Executive Summary

In February 2023, X41 D-Sec GmbH performed a source code security audit against the in-toto Python and Go implementations along with the architectural specifications to identify vulnerabilities and weaknesses in the framework.

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

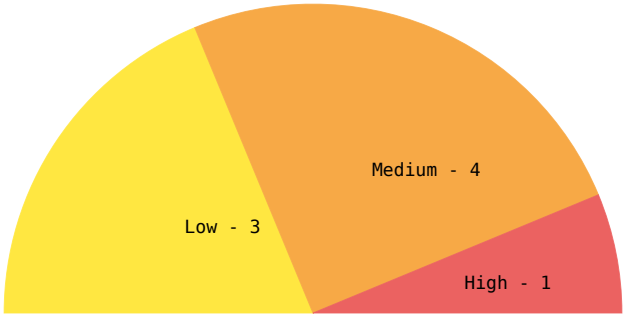

Figure 1.1: Issues and Severity

in-toto is a framework that helps to secure the software supply chain from initiation to end-user installation. Vulnerabilities in the framework or specification would allow an attacker to compromise the software supply chain and therefore the product relying on it.

In a source code audit, the testers receive all available information about the target, including source code and specification.

The test was performed by three experienced security experts between 2023-02-06 and 2023-02-24.

The code quality for the source code analyzed was very good and most issues identified are related to the design or architectural limits of in-toto.

The most severe issue discovered allows the supply chain to be compromised until the final verification, since individual functionaries cannot verify the intermediate data supplied using in-toto. Additionally, file metadata is not verified which might lead to security issues unless a container format is used.

Several issues were found in the Python implementation which are related to the verification of data signed with PGP keys that have been imported from GnuPG to the layout file.

Although no direct security implication was identified, the ambiguity of JSON parsing is a concern to the auditors and might lead to issues in specific setups or when additional implementations of the specifications are created.

X41 recommends to make use of GnuPG for verification of signatures, and to move from JSON-based formats that are prone to different interpretations to a strictly defined format for all metadata. Additionally, X41 recommends to clearly describe the attack scenario and goals of in-toto as well as the limitations of what the framework provides. Furthermore, an audit of the securesystemslib dependency should be performed.

In conclusion, in-toto offers a robust codebase but is able to improve on the general design architecture.

2 Introduction

X41 reviewed the in-toto Python and Go implementations as well as the specification. These ensure that no untrusted third parties added malicious code or data as part of the software supply chain.

Attackers could try to attack the supply chain in various stages, ranging from altering commits of individual developers in a code repository to compromising the CI/CD[1] pipeline.

2.1 Methodology

The audit was based on a source code and specification review.

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

X41 adheres to established standards for source code reviewing and penetration testing. These are in particular the *CERT Secure Coding*[2] standards and the *Study - A Penetration Testing Model*[3] of the German Federal Office for Information Security.

[1] Continuous Integration/Continuous Delivery
[2] https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards
[3] https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/Penetration/penetration_pdf.pdf?__blob=publicationFile&v=1

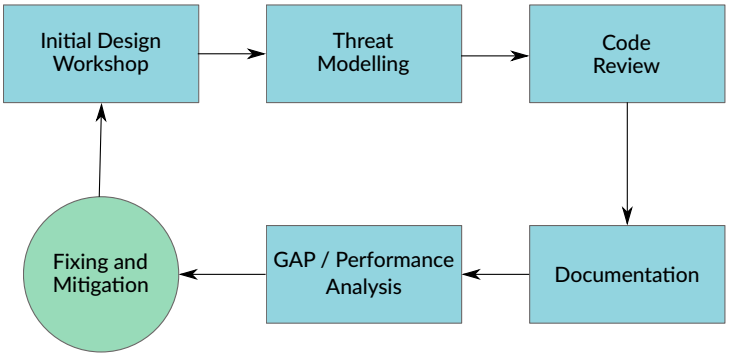

Figure 2.1: Code Review Methodology

2.2 Findings Overview

DESCRIPTION	SEVERITY	ID	REF
File Metadata Ignored	MEDIUM	TOTO-PT-23-01	4.1.1
Configuration Read From Local Directory	MEDIUM	TOTO-PT-23-02	4.1.2
Layout Replay	LOW	TOTO-PT-23-03	4.1.3
Link File Reuse	MEDIUM	TOTO-PT-23-04	4.1.4
Functionaries Do Not Perform Verification	HIGH	TOTO-PT-23-05	4.1.5
PGP Key Creation Time Not Validated	LOW	TOTO-PT-23-06	4.1.6
PGP Key Revocation Not Considered	MEDIUM	TOTO-PT-23-07	4.1.7
PGP Key Usage Flags Not Considered	LOW	TOTO-PT-23-08	4.1.8
Broken Filename Mangling	NONE	TOTO-PT-23-100	4.2.1
No User ID for Vulnerability Report PGP Key	NONE	TOTO-PT-23-101	4.2.2
Argument Silently Ignored	NONE	TOTO-PT-23-102	4.2.3
Key Management	NONE	TOTO-PT-23-103	4.2.4
Inconsistent Use of GnuPG	NONE	TOTO-PT-23-104	4.2.5
Undefined Pattern Matching	NONE	TOTO-PT-23-105	4.2.6
Expiration Date Might Prevent Installation	NONE	TOTO-PT-23-106	4.2.7
JSON Parsing Differences	NONE	TOTO-PT-23-107	4.2.8
Complicated Signature Count	NONE	TOTO-PT-23-108	4.2.9

Table 2.1: Security-Relevant Findings

2.3 Scope

The scope of this test were the Python 1.3.1 (5.5 KLOC[4])[5] and Go 0.6.0 (9 KLOC)[6] implementations. Additionally, the review covered the specification[7] and attestation definition[8].

The threat model of in-toto is described in a paper[9]. Since in-toto provides the framework, the actual threat model might be different for the project using it and needs to be reflected in the layout rules. For this audit, it is assumed that attackers are able to modify data, layout and link files that are exchanged between functionaries or before being passed to the user.

2.4 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.

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

Bandit[10] and Semgrep[11] were used to perform static analysis on the Python code and its dependencies. Semgrep and staticcheck[12] were used for Go code. Additionally, the specification and attestation definition were audited for design flaws and additional attack surface. The Python and Go implementations were audited for deviations from the specifications and implementation-specific security issues.

Suggestions for next steps in securing this scope can be found in section 2.5.

[4] Thousand Lines of Code
[5] https://github.com/in-toto/in-toto/releases/tag/v1.3.1
[6] https://github.com/in-toto/in-toto-golang/releases/tag/v0.6.0
[7] https://github.com/in-toto/docs/blob/e2501be6d0bc82e9a3a46f12ef1e84d65af584fb/in-toto-spec.md
[8] https://github.com/in-toto/attestation/tree/903f02ab40f7b4823eed502528c598a7ba617bd3
[9] https://www.usenix.org/system/files/sec19-torres-arias.pdf
[10] https://github.com/PyCQA/bandit
[11] https://semgrep.dev/
[12] https://staticcheck.io/

2.5 Recommended Further Tests

X41 recommends to mitigate the issues described in this report. Afterwards, CVE[13] IDs[14] should be requested and customers be informed (e.g., via a changelog or a special note for issues with higher severity) to ensure that they can make an informed decision about upgrading or other possible mitigations.

Dependencies of the project that have not yet undergone security testing are recommended to also investigate, in particular as they relate to cryptographic operations such as the Python and Go implementations of `securesystemslib`.

[13] Common Vulnerabilities and Exposures
[14] Identifiers

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 informational 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 TOTO-PT-23-01: File Metadata Ignored

Severity:	MEDIUM
CWE:	349 – Acceptance of Extraneous Untrusted Data With Trusted Data
Affected Component:	Architecture

4.1.1.1 Description

in-toto verifies the integrity of all files by using cryptographically secure hashes, but the integrity of file metadata is not verified and therefore not ensured by the process. This includes file permissions, timestamps and additional associated data such as NTFS[1] alternate data streams[2].

The impact of this depends on the individual project and files associated. File permissions might be changed to very permissive, which allows local privilege escalations after installing the final product. Additionally, removing the executable permission from a file might cause the software

[1] New Technology File System
[2] `https://owasp.org/www-community/attacks/Windows_alternate_data_stream`

to behave in insecure ways (one might imagine a `SecurityHelper.exe` could be prevented from running).

4.1.1.2 Solution Advice

X41 recommends to verify file metadata as well or at least provide tools to do so. If this is not possible, it should be advised to always place the files in a container format that specifies permissions.

4.1.2 TOTO-PT-23-02: Configuration Read From Local Directory

Severity:	MEDIUM
CWE:	15 – External Control of System or Configuration Setting
Affected Component:	in-toto-1.3.1/in_toto/user_settings.py

4.1.2.1 Description

The in-toto configuration is read from various directories and allows to configure the behavior of the framework. Among the files read is `.in_totorc` which is a hidden file[3] in the directory in which in-toto is run. When this file is bundled among the development files, this might allow attackers to influence in-between steps (e.g., by using exclude patterns or setting a different artifact base path).

The actual impact depends on how in-toto is used for a certain project and if an attacker is able to add files between functionaries.

4.1.2.2 Solution Advice

X41 recommends to only use configuration files from paths which an attacker (between functionaries) has no influence over. If setups rely on this feature, it should be opt-in via a global configuration file.

[3] `https://en.wikipedia.org/wiki/Hidden_file_and_hidden_directory#Unix_and_Unix-like_environments`

4.1.3 TOTO-PT-23-03: Layout Replay

Severity:	LOW
CWE:	1328 – Security Version Number Mutable to Older Versions
Affected Component:	Architecture

4.1.3.1 Description

The layout file contains an expiration date that is intended to prevent attackers (to a certain extent) from replaying older versions that might contain security vulnerabilities to users. Nevertheless, it might be possible for attackers to block the roll-out of a certain version and perform replays during the expiration period.

```
1  "signed": {
2    "_type": "layout",
3    "expires": "2023-03-10T10:02:43Z",
```

Listing 4.1: Layout Expiration

4.1.3.2 Solution Advice

X41 recommends to add a version number or counter into the layout that ensures that users can verify whether they are missing an in-between version. Additionally, users are able to use the version number to detect layout replay.

4.1.4 TOTO-PT-23-04: Link File Reuse

Severity:	MEDIUM
CWE:	345 – Insufficient Verification of Data Authenticity
Affected Component:	Architecture

4.1.4.1 Description

The link files created by in-toto do not reference a layout file nor do they contain timestamps. This allows attackers to replay the first steps by replacing link files with ones from an earlier version or even replay single steps where the hashes of the expected materials match an earlier version. This might allow them to introduce bugs or prevent fixes for bugs that are not visible via the hashes. One such scenario would be the update of an compiler that introduces mitigations similar to Meltdown and Spectre[4].

4.1.4.2 Solution Advice

X41 recommends to add a globally unique ID to the layout file to be referenced by link files. A weaker mitigation might be to add timestamps into the link files that display when they are created and a timestamp that specifies when the layout file starts being valid (similar to the expiration field) to reduce the time frame in which a link file can be replayed.

[4] `https://meltdownattack.com/`

4.1.5 TOTO-PT-23-05: Functionaries Do Not Perform Verification

Severity:	HIGH
CWE:	345 – Insufficient Verification of Data Authenticity
Affected Component:	Architecture

4.1.5.1 Description

There is no verification step defined by in-toto to be performed by the functionaries before they use the provided data. An attacker able to modify files that are shared between the functionaries (as specified by the threat model) is able to add malicious content and therefore compromise the whole chain of functionaries. This is only detected when the product's supply chain is verified at the end. At this point, the whole supply chain might already be compromised and other products been tampered with and keys compromised without this being detectable by in-toto.

4.1.5.2 Solution Advice

X41 recommends to have each functionary verify the steps before to ensure all actions are performed on trusted data. This should be done for the layout file, the link files, and the actual data operated on. This seems to be discussed in a GitHub issue[5] as well.

[5] `https://github.com/in-toto/in-toto/issues/116`

4.1.6 TOTO-PT-23-06: PGP Key Creation Time Not Validated

Severity:	LOW
CWE:	295 – Improper Certificate Validation
Affected Component:	securesystemslib-0.26.0/securesystemslib/gpg/functions.py:252

4.1.6.1 Description

The Python implementation of in-toto correctly raises an exception during the verification process when a PGP[6] key's validity period is in the past, but does not do so when the validity period (starting with the key creation time) is in the future.

A validity period in the future is usually a sign of a wrong system clock, meaning it can't be trusted for verifying the validity period.

A MITM[7] attacker who is able to manipulate delivered software products might also be able to control the system time by manipulating NTP[8]. In a scenario where an attacker gained control over two expired subkeys with no overlapping validity period, the attacker could set the system time to a time before the validity period of either key, resulting in both keys being accepted.

4.1.6.2 Solution Advice

X41 recommends to additionally assert that the key has been created before the current system time. Please refer to section 4.2.5 for further discussion of issues related to PGP.

[6] Pretty Good Privacy
[7] Manipulator-in-the-middle Attack
[8] Network Time Protocol

4.1.7 TOTO-PT-23-07: PGP Key Revocation Not Considered

Severity:	MEDIUM
CWE:	295 – Improper Certificate Validation
Affected Component:	Python implementation

4.1.7.1 Description

The in-toto Python implementation uses GnuPG[9] to support the use of PGP keys. Keys are fetched from GnuPG and added to the layout, but GnuPG is not involved in the validation process, meaning that keys that have been revoked (e.g., due to a compromised supply chain) are still accepted in signatures.

4.1.7.2 Solution Advice

X41 recommends to verify that no keys have been revoked at the time of verification. Please refer to section 4.2.5 for further discussion of issues related to PGP.

[9] GNU Privacy Guard

4.1.8 TOTO-PT-23-08: PGP Key Usage Flags Not Considered

Severity:	LOW
CWE:	295 – Improper Certificate Validation
Affected Component:	Python implementation

4.1.8.1 Description

The in-toto Python implementation uses GnuPG to support the use of PGP keys. When specifying a key to be added to the layout, the key's metadata is fetched from GnuPG. The key and its subkeys are added to the layout and equally allowed to sign data, but the key's usage flags[10] are not included in the key's metadata and not used during verification.

This potentially allows to use a PGP subkey for signing that is not permitted to sign data as per its key usage flags.

4.1.8.2 Solution Advice

X41 recommends to verify key usage flags at the time of verification. Please refer to section 4.2.5 for further discussion of issues related to PGP.

[10] `https://datatracker.ietf.org/doc/html/rfc4880#section-5.2.3.21`

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 TOTO-PT-23-100: Broken Filename Mangling

Affected Component:	in-toto-1.3.1/in_toto/runlib.py

4.2.1.1 Description

Filenames get mangled to provide consistency between different operating systems. Replacing a backslash with a slash might break valid Unix file paths.

```
1 if os.path.isfile(artifact):
2   # FIXME: this is necessary to provide consisency between windows
3   # filepaths and *nix filepaths. A better solution may be in order
4   # though...
5   artifact = artifact.replace('\\', '/')
6   key = _apply_left_strip(artifact, artifacts_dict, lstrip_paths)
7   artifacts_dict[key] = _hash_artifact(artifact,
8       normalize_line_endings=normalize_line_endings)
```

Listing 4.2: Filename Mangling

4.2.1.2 Solution Advice

X41 recommends to normalize file paths by using *file://* URLs[11] that are handled the same on all supported operating systems.

[11] Uniform Resource Locators

4.2.2 TOTO-PT-23-101: No User ID for Vulnerability Report PGP Key

Affected Component: in-toto-1.3.1/SECURITY.md

4.2.2.1 Description

The `SECURITY.md` file specifies:

"Optionally, reports that are emailed can be encrypted with PGP. You should use PGP key fingerprint 903B AB73 640E B6D6 5533 EFF3 468F 122C E816 2295."

This PGP key is hard to use because it contains no user ID:

```
$ gpg --recv-keys 903BAB73640EB6D65533EFF3468F122CE8162295
gpg: key 468F122CE8162295: new key but contains no user ID - skipped

$ gpg --encrypt --recipient 903BAB73640EB6D65533EFF3468F122CE8162295
gpg: 903BAB73640EB6D65533EFF3468F122CE8162295: skipped: No public key
gpg: [stdin]: encryption failed: No public key
```

Listing 4.3: Downloading the PGP Key

Due to the unstable situation of key servers, some have chosen not to distribute user IDs anymore. This can be remedied with local workarounds such as appending some binary string containing a user ID specification to the public key file[12], but this raises the bar for using secure communication via a third-party email address.

4.2.2.2 Solution Advice

If the key has a user ID, it is recommended to include the public key instead of the fingerprint. This avoids needing a third party to obtain the key as well as avoiding the stripped user ID problem.

If the key has no user ID in its original form, the easiest solution might be to give it a generic name without email address such as "in-toto Security Key".

[12] `https://unix.stackexchange.com/a/683251`

4.2.3 TOTO-PT-23-102: Argument Silently Ignored

Affected Component:	in-toto-1.3.1/in_toto/in-toto-keygen.py

4.2.3.1 Description

When running the in-toto key generation helper, it silently ignores the key size option when creating ECDSA[13] or ED25519[14] keys. This might lead to confusion regarding the strength of the actual key generated.

```
1 $ in-toto-keygen -t ecdsa -b 120000000 keyfile
2 $
```

Listing 4.4: Argument Silently Ignored

4.2.3.2 Solution Advice

X41 recommends to generate a warning message.

[13] Elliptic Curve Digital Signature Algorithm
[14] Edwards-curve 25519

4.2.4 TOTO-PT-23-103: Key Management

Affected Component: Architecture

4.2.4.1 Description

in-toto does not deal with key management and relies on additional channels to distribute and verify the generated keys in a secure manner. Therefore, key expiration and revocation is outside of the project's responsibility. However, the keys generated by `in-toto-keygen` do not provide fields that can support this. Although this is not security relevant in regards to the threat model defined, this could be improved upon.

4.2.4.2 Solution Advice

X41 recommends to use keys and certificates that have key management infrastructure in place instead of generating a new key format. These might contain PGP keys and S/MIME[15] certificates.

[15] Secure/Multipurpose Internet Mail Extensions

4.2.5 TOTO-PT-23-104: Inconsistent Use of GnuPG

Affected Component:	Python implementation

4.2.5.1 Description

The Python implementation of in-toto supports the use of PGP keys via GnuPG.

When specifying a PGP key (via its ID) to be added to the layout, the key's metadata is fetched from GnuPG and added to the key object in the layout. The metadata includes some (but not all) attributes of the key. It also includes subkeys of the specified key, which are accepted in place of the specified key during verification.

in-toto re-implements some (but not all) of GnuPG's validations when adding the key and during verification, however only the key metadata from the layout is used during verification and GnuPG is not invoked in the process.

X41 understands that in-toto's goal is not to provide a re-implementation of GnuPG and therefore considers it an unnecessary risk – and avoidable complexity – to evade GnuPG's validations in the verification process.

Please refer to findings 4.1.6, 4.1.7, 4.1.8, and 4.2.9 that were found related to this issue.

4.2.5.2 Solution Advice

X41 recommends to only specify the key ID in the layout, and to leave all other steps involved with key management, signing, and verification to GnuPG.

4.2.6 TOTO-PT-23-105: Undefined Pattern Matching

Affected Component:	Specification, Python implementation, Go implementation

4.2.6.1 Description

Section 4.3.3 of the in-toto specification specifies a "pattern" for the artifact rules, but only describes them as "bash-style wildcards" and does not further define the pattern matching syntax.

The Python implementation of in-toto uses the ***fnmatch***[16] module for pattern matching, while the Go implementation uses a customized version[17] of the ***filepath.Match***[18] function.

The Python and Go functions differ in the way patterns are applied, for example regarding escaping and negated sequence matching.

4.2.6.2 Solution Advice

X41 recommends to describe the pattern syntax in the specification, or to refer to a specific version of a third-party pattern syntax definition, such as IEEE Std 1003.1-2017, 2.13.1[19].

The Python and Go implementations should implement the same pattern matching syntax.

[16] https://docs.python.org/3.10/library/fnmatch.html
[17] https://github.com/in-toto/in-toto-golang/blob/82bb771212f4cfb71da8a325278f3cce6f042205/in_toto/match.go
[18] https://pkg.go.dev/path/filepath@go1.17.13#Match
[19] https://pubs.opengroup.org/onlinepubs/9699919799/utilities/V3_chap02.html#tag_18_13_01

4.2.7 TOTO-PT-23-106: Expiration Date Might Prevent Installation

Affected Component: Architecture

4.2.7.1 Description

The expiration date in the layout files might prevent users from properly verifying and installing a product after that date. This might force the users of in-toto to create additional releases that offer no functional changes or use overly long expiration dates.

4.2.7.2 Solution Advice

X41 recommends to add an option to enforce a certain version counter value as mentioned in finding 4.1.3.

4.2.8 TOTO-PT-23-107: JSON Parsing Differences

Affected Component:	Architecture, Python implementation, Go implementation

4.2.8.1 Description

in-toto uses JSON[20] for many of the security-relevant parts, which poses a risk due to the relatively lax definition of JSON and the factual differences of JSON parsers and serializers[21].

The Go implementation ignores the case of key names when parsing, but does not accept extraneous keys whereas the Python implementation is case sensitive, but ignores extraneous keys.

In either case, the JSON document can be tampered with while the cryptographic signature is still considered valid.

When the contents are the same, for instance if the uppercased `Run` key also contains a bare `whoami` command, then the document is currently accepted as valid in both languages. This means that the following JSON documents are all considered valid *without re-signing* in both languages:

```
$ cat variant1.json
{
 "run": ["whoami"]
}

$ cat variant2.json
{
 "run": ["whoami"],
 "Run": ["whoami"]
}

$ cat variant3.json
{
 "Run": ["whoami"],
 "ru\u006E": ["whoami"]
}
```

Listing 4.5: Identical JSON Objects after Canonicalization

[20] JavaScript Object Notation
[21] `http://seriot.ch/projects/parsing_json.html`

Each of the three variants verifies successfully using an identical signature in both the Python and Go implementations of in-toto. The Go implementation produces errors when there are unexpected (extraneous) keys, but since it considers differently-cased variants to be identical, the alterations are accepted. The Python implementation does not error out when an extraneous key is added, such as a differently-cased one.

One might imagine a scenario where in-toto verifies the signature of a file and then passes that file to a different program, which then parses different data (e.g., a command to execute) than the one verified by in-toto before.

```
"signed": {
 "_type": "layout",
 "expires": "2023-03-15T09:18:03Z",
 "inspect": [
  {
   "run": [
    "whoami"
   ],
   "_type": "inspection",
   "expected_materials": [],
   "expected_products": [],
   "name": "runcmd",
   "Run": [
    "rm",
    "/"
   ]
  }
 ],
 "keys": {},
 "readme": "",
 "steps": []
}
```

Listing 4.6: JSON "run" versus "Run"

The payload shown in listing 4.6 would execute `whoami` in Python but `rm /` in Go because the case insensitive handling of keys in Go results in it overwriting the formerly benign command. Both languages have a policy of considering the last key with the same name as the valid one, but Go ignores case differences.

In another attack scenario, the signatory might be tricked into accepting the *layout* file contents because they look good, while in reality a secondary key located further down in the document specifies what is truly being executed, as also demonstrated in listing 4.6. The overwriting can be made less obvious by adding more information in between, obfuscating the *Run* key such as

by using hex encoding, and obfuscating the malicious command's contents.

ITE[22]-5[23] proposes the use the Dead Simple Signing Envelope (DSSE)[24], where data is serialized before it is signed. This allows to verify the signature before deserialization and prevents modifications to the serialized data after signing. However, DSSE uses JSON as a container format and is thus vulnerable to some extent when parsing the container – A DSSE file might contain two pairs of payloads and signatures that are read inconsistently depending on the implementation. If the DSSE payload also consists of serialized JSON (as currently proposed), DSSE would not prevent the second attack scenario described where an attacker tricks the signatory.

4.2.8.2 Solution Advice

To minimize the attack vector, X41 recommends to use a serialization format that is more strictly defined, has less implementation differences and relies on the data structure to be defined on the serializing and deserializing ends rather than within the serialization. Alternatively, X41 recommends to implement stricter JSON document verification in both languages in addition to DSSE, such as rejecting documents with duplicate keys, ill-cased keys, and extraneous keys. However, it is important to point out that such safety measures are outside the usual handling of JSON documents and may not be followed by (or even be reasonably available to) all implementers of the in-toto specification.

[22] in-toto Enhancement
[23] `https://github.com/in-toto/ITE/blob/master/ITE/5/README.adoc`
[24] `https://github.com/secure-systems-lab/dsse`

4.2.9 TOTO-PT-23-108: Complicated Signature Count

Affected Component: Python implementation

4.2.9.1 Description

The in-toto Python implementation accepts signatures from PGP subkeys (if they are specified in the layout) in place of their master keys. To prevent the use of multiple subkeys of the same master key to count multiple times against the threshold, only the number of unique master key's signatures are to be counted against the threshold specified in the layout.

in-toto uses a complicated method involving a dictionary, a list, a unique set from that list, and several subtractions of their lengths to achieve this. The complexity poses a potential risk, as it is easy to make a mistake when making changes to this part of the code.

4.2.9.2 Solution Advice

X41 recommends to simplify the counting of signatures. Please refer to section 4.2.5 for further discussion of issues related to PGP.

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:

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

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

Acronyms

CI/CD Continuous Integration/Continuous Delivery 6
CVE Common Vulnerabilities and Exposures 9
CWE Common Weakness Enumeration 11
ECDSA Elliptic Curve Digital Signature Algorithm 23
ED25519 Edwards-curve 25519 23
GnuPG GNU Privacy Guard 19
ID Identifier 9
ITE in-toto Enhancement 30
JSON JavaScript Object Notation 28
KLOC Thousand Lines of Code 8
MITM Manipulator-in-the-middle Attack 18
NTFS New Technology File System 12
NTP Network Time Protocol 18
PGP Pretty Good Privacy 18
S/MIME Secure/Multipurpose Internet Mail Extensions 24
URL Uniform Resource Locator 21