TUF and target discovery

Oct 7, 2022
Jussi Kukkonen

Problem statement

The Update Framework provides secure content delivery by offering a few interesting features:

1. Target downloads protected by signatures
2. Delegation of trust
3. Key management
4. zero client configuration – trust delegation and key management require no action from the client

There is another feature that many users expect to find:

5. Artifact discovery/search

Unfortunately **Artifact/target discovery is not part of the TUF design**: the only way to "find" a specific artifact in a generic TUF repository is to know its unique identifier, the *targetpath*. TUF is a way to download artifacts, not to discover which artifacts are available.

A common example of the expected search functionality is versions: if the repository contains targets with targetpaths *artifact-v1.0.0.zip* and *artifact-v2.0.0.zip* TUF offers no answer to the questions

- What is the newest version of *artifact-v*.zip*?
- What versions of *artifact-v*.zip* are available*?*

The TUF specification design means that there is no generic way to provide these answers. In addition, even though the metadata may allow some forms of target discovery (with e.g. custom metadata), the delegation features of TUF make it very difficult to do so safely.

That said, there are solutions that are applicable to specific cases: A repository may provide a custom search index or the repository's delegations may be structured in a way that allows a client to use the metadata as a search index.

Possible solutions

Only use well known targetpaths

Aka “Do not expect TUF to handle target discovery”. It may be useful to re-evaluate the repository design: Sometimes it is possible to design target paths that clients can always predict. If this is the case, discovery is not really needed.

Application specific search index as a target file

The most powerful search index is the application-specific one: A target file with a well-known targetpath that contains a searchable index of the actual artifacts in whatever form is most useful for the application.
This solution will be compatible with any spec-compliant client library but requires the client application to implement the search itself.
This is the solution PyPIs PEP-458 uses: each project has an index file in a well known location (something like *“simple/<PROJECT>.html”*). The index file contains a list of all artifacts provided by that project. Clients always download that index file first, and then decides which target they are actually interested in.
Note that

- If the “actual artifacts” are delegated to the same role as the index file, the artifacts do not necessarily even need to be targets listed in TUF metadata: the index file can include the information required to verify the artifacts
- The index file cancontain the artifact metadata in the format that TUF uses: this way the final verification is trivial as the TUF client can be used to download and verify artifact validity.

Metadata role as search index

It is possible to use a targets roles metadata itself as a search index – client could use e.g. the custom metadata attached to the specific targets within a delegated role to decide which targets it is interested in. This approach has problems however:

- Doing this for delegated (non top-level) roles has safety issues: *The fact that a target is listed in a roles metadata, does not mean the role is delegated to provide that metadata*. Using a roles target list for target discovery essentially circumvents TUF delegation security guarantees
- Even without the security problem, hashed bin delegation and succinct delegation (TAP-15) are incompatible with this approach: selecting the correct bin requires knowing the full targetpath

Because of this safety issue, TUF client libraries should *not* simply expose the metadata to the client application. The issue can be solved, but this requires new features from the TUF client library. The algorithm would look like this:

```python
# pseudocode
def get_all_targetinfos(targetpath: str) -> List:
    result = []
    # find the delegating role
    targets_role = get_delegated_role_for(targetpath)
    # make sure each targetpath really is delegated to role
    for other_targetpath, target in targets_role.targets.items():
        if role == get_delegated_role_for(other_targetpath)
            result.append(target)
    return result
```

Things to note here:

- The correct role is selected with a targetpath argument
- Each target in the role is verified to actually be delegated to this role

Next steps

Potential action items include

- Define some of these discovery methods in the client workflow or supporting documentation – the purpose would be to make client developers aware of the use cases so they expose the required functionality to applications
- Make sure the discovery methods are supported in existing client libraries
    - python-tuf does not currently expose metadata to downloader clients (issue 1995)
- Document the design restrictions and best practices for repository maintainers

These are independent actions that have no strict dependencies on each other.

Links

Sigstore TUF Client

python-tuf issue #1995 "RFE: expose delegating metadata to client application"