Secure Update Framework Repository Key Management and Trust Delegation

Justin Cappos Geremy Condra

September 15, 2010

1 Introduction

The goal of a TUF repository is to store a collection of targets (the updated files to be served to the client) and a set of metadata with which the client can ensure that the updates they recieve are complete, timely, and authentic. Meeting those goals requires that the respository be able to leverage digital signatures, and so TUF provides a set of tools that helps manage the complexity of generating metadata and improves the security posture of the repository. These tools (including mechanisms for key management, delegation of trust, and key revocation) form the subject of this document.

1.1 Scope

This document specifies the required trust delegation and key management routines for TUF repositories.

2 Overview

The Update Framework is a Python library designed to assist software developers in the task of safely and securely updating their software after its deployment with an emphasis on resiliance in the face of key compromise. Towards that end, the issues of key storage, revocation of keys and their accompanying trust, and the delegation of that trust have been given careful attention. This document exists to provide guidance on both mandatory behaviors and best practices with regard to those.

2.1 Relationship to Other Documents

Much of the behavior specified in this document is partially laid out in the core TUF document. The system's behavior with regard to freeze and replay attacks is covered in the document entitled "Software Update Security Framework: Repository Library Replay and Freeze Attack Protection". The client-side counterpart to this document contains a large amount of information on the

response that subsystem will demonstrate in many of the same circumstances. In particular, it contains an overview of how to set up and run an experimental TUF repository and client.

3 Key Management

Where TUF clients are primarily concerned with appropriately associating roles to public keys, those maintaining TUF repositories are additionally required to keep the relevant signing keys private. Mechanically, this means storing them in a custom AES-encrypted keystore, but other precautions (detailed below) should be observed in order to minimize the risk of a key compromise.

3.1 Key Storage

TUF's keystore takes the form of an AES-encrypted key database with two backing data stores, both Python dictionaries and containing keys (both public and private) and their roles. In addition to the interface provided by its KeyDB parent class (primarily responsible for associating roles and key IDs with keys and delegation information) the keystore also provides three methods:

- *set_password*(*password*), sets the password used to generate the AES key.
- *clear_password*(), clears the aforementioned password.
- and *save*(), encodes, encrypts, and writes the database to disk.

Let's walk through each of these in slightly more detail.

Setting or clearing the password to be used does nothing more than set and clear a password field in the keystore, but note that if a keystore does not have a password field set when *save*() is called, it won't encrypt it- your keys will be on disk in plaintext. Also note that passwords must not be used directly- at the moment TUF clients use RFC 2440 password mangling to derive a key from the original password material, but see the future work section at the end of this document for more information on development of this system.

Saving the keystore is done through TUF's usual method of serialization, which is to say Canonical JSON. Encryption is done using AES-256, after which the result is written to disk to be decrypted at some later date. Due to recent attacks on this key length, clients may also opt to use AES-192.

3.2 Best Practices

While TUF places a great deal of emphasis on the ability to recover from a key compromise and uses strong cryptographic techniques to minimize the attack window, avoiding such scenarios in the first place is always preferable. As a

result, a few best practices are recommended for those responsible for storing keying material:

1. Store the root key offline- while a root key compromise is recoverable, revoking a root key is more troublesome than revoking other keys.
2. No single point of failure should permit access to both the target and release roles' keys.
3. The timestamp role's key can be stored online for automated timestamping.

For some, the above will prove to be difficult to provide inside of an organization. Those users should note that while TUF by no means requries an external PKI to operate, its design permits their use.

3.3 Command Line Interface

To make repository management easier on maintainers, a set of easy-to-use command line tools has been developed that works to simplify most of the tasks above. In particular, the signercli.py script provides an easy way to perform the most common tasks a maintainer will face.

3.3.1 Generating and Listing Keys

Besides generating the initial keystore using the quickstart.py script we saw earlier, the signercli.py script provides an easy way to generate keys using the *genkey* subcommand. Using it is extremely easy:

```
cd demorepo
signercli.py genkey --keystore=../keystore
```

This will give you a good deal of output, ultimately including a line something like the following:

```
[TIME] [tuf] [INFO] Generated new key: KEYID
```

Listing key IDs is similarly easy:

```
cd demorepo
signercli.py listkeys --keystore=../keystore
```

You can also get more information about a particular key using the dumpkey subcommand. To print information about a public key, do the following:

```
cd demorepo
signercli.py dumpkey --keystore=../keystore KEYID
```

If you pass the "–include-secret" option, it will also print the signing key.

3.3.2 Changing Keystore Passwords

To change the keystore password, simply run the signercli.py script with the changepass subcommand, type the current password, and then enter the new password. Here's an example:

```
cd demorepo
signercli.py changepass --keystore=../keystore
```

Note that there is not currently a mechanism for providing these options on the commandline. This is done to prevent the password from being pulled from an ungaurded shell history file.

3.3.3 Delegating Trust

TUF makes delegating trust quite easy, and an example of how to do so is provided in the companion client-side document. Note that the folder a delegated role's ROLE.txt metadata file goes into must exist before running the script.

3.3.4 Revoking Trust

TUF does not currently provide a command line interface for revoking trust, however, doing so is simple using existing tools. To revoke a delegated trust, just delete the accompanying ROLE.txt and create a new update. The revocation procedure for other keys is described in the companion document, section 7.

4 Trust Delegation

The TUF trust delegation model is described in the TUF spec section 4.5 and in the TUF client key management specification section 6.

5 Future Work

Three major areas of future work remain here: first, accounting for improved cryptographic and cryptanalytic results against AES-256 and the RFC2440 key derivation algorithm; second, developing additional tools to provide for automatic revocation of trust for all roles; and third improving the mechanisms for automatic integration of other projects with TUF.