Software Update Security Framework: Repository Library Design and Attack Protection

Justin Cappos, University of Washington
Justin Samuel, University of Washington

1. Overview

The repository library generates the metadata that enables software updates to be securely available to clients. The content of this metadata, how it is signed, and who is trusted to sign it are vital to the client library's ability to remain secure in the event of replay attacks and freeze attacks. Additionally, its design promotes high availability through the use of repository mirrors and static content.

This document covers the repository library as well as descriptions of the files that exist on a repository. A discussion of replay and freeze attacks and how clients use the metadata to protect themselves can be found in the client library design document.

1.1. Terminology

In the list below, we explain how we use various terms in this document. For simplicity, we have used the terms "projects" and "project members" and defined them in ways that correspond to a common use case. This choice of terminology does not restrict the ways in which the software update security framework can be used.

Repository. A repository is the collection of all of the metadata files and target files (software update files) which a project makes available to clients.

Mirror. A mirror is a location (a URL) from which a repository's files are available. A single repository has one or more mirrors.

Project. A project is the individual or organization that is using the software update framework to securely update software on client systems. In essence, this is the organization which develops the software.

Project member (developer). A project member is an individual that is affiliated with a project and has some responsibility for making updates available to clients. Developers are project members who push software update releases and are trusted by clients to do so.

Target files. All files that are part of software updates available to clients are called target files. We will not use the term "update files" by itself in this document to avoid confusing updated target files with updated metadata files.

Metadata. Metadata files describe target files which are available for download as well as other metadata files. Everything on the repository that is not a target file is a metadata file. All metadata is signed.

2. Design Goals

The repository library must provide a way for the project members to add target files and signed metadata to the repository. The repository library will make available the files that project members have added to the repository.

All repository content will consist of static files. This allows for flexibility in how repository content is made

available to clients (e.g. choice of web server software) and makes repositories easy to mirror.

In addition to being easy to mirror a complete repository, it is desirable for a mirror to be able to only mirror part of the repository.

The contents of a repository's signed metadata will provide sufficient information to allow clients to protect themselves against replay and freeze attacks as well as to detect stale mirrors.

The process by which developers push updates to the repository must be simple. The repository library will perform sanity checking of developer-provided metadata in order to help project members identify mistakes but this is not required for client security. Clients do not ultimately trust the repository and so will perform all required signature and trust checking on their side.

The repository library and signed metadata will allow clients to safely use mirrors that do not support SSL (TLS). This does not exclude the optional use of SSL when available, but clients will be secure without it.

The repository library incorporates the functionality required to generate and sign all types of metadata. A front-end to the repository library will provide the ability to generate and sign metadata files both interactively and non-interactively. The repository library will also provide the ability to validate the contents of a repository.

The repository library will form the basis of developer push mechanisms which will simplify the process by which developers will submit new target files and signed metadata to the repository.

3. Metadata Files

Each metadata file is signed and includes both the time it was signed and when the file should be considered expired. Clients use this information to prevent replay and freeze attacks.

Even though all signed metadata includes expiration dates, some of these expiration dates may be quite long. However, long expiration dates open up the possibility of freeze attacks.

To prevent an adversary from replaying a signed metadata file whose signature has not yet expired but which has been superseded by a newer file, a frequently signed timestamp file is used. This timestamp file has a short expiration time (for example, a few hours). An automated process periodically signs this timestamp file which contains hashes of the other metadata files.

4. The Repository

A repository is a conceptual source of target files and metadata files provided by a project. Each repository has one or more mirrors which are the actual providers of files to be downloaded. For example, each mirror may specify a different host where files can be downloaded from over HTTP.

The mirrors can be full or partial mirrors as long as the client side of the framework can ultimately obtain all of the files it needs. A mirror is a partial mirror if it only provides some of the repository's files. The repository paths which are available on each mirror are specified in the mirror list.

The keys, metadata, and target files are completely separate between repositories. A multi-repository setup is a multi-root system. When a software update system uses the client library with multiple repositories, the client library does not perform any mixing of the trusted content from each repository. It is up to the software update system using the client library to determine the significance of the same or different target files provided by separate repositories. Multi-root arrangements such as this are useful to certain types of package managers (programs that maintains much

or all of the installed software on a system) but generally are not of interest to application updaters belonging to individual projects.

4.1. Repository Layout

The following are the metadata files and directories that exist on all repositories.

/meta/mirrors.txt

Provides the locations of mirrors and indicates which parts of the repository are available at each mirror.

/meta/timestamp.txt

Indicates hashes and lengths of mirrors.txt and main.txt

This is the only file that needs to be downloaded when clients poll for the existence of updates to any metadata files or target files.

/meta/main.txt

Indicates the hashes and lengths of the latest versions of all available target files. Also includes any additional metadata about target files that the project chooses to include.

/targets/

This directory contains all target files, including any directory structure the project wants to use. The directory structure used here enables cleanly dividing content between mirrors.

The format, types, and sizes of files in this directory are irrelevant to the framework. All target files are opaque as far as the framework is concerned.

5. Threat Model And Analysis

The security of clients when confronted with various adversaries is discussed in the client library design document. Here we will specifically consider the issue of denial-of-service attacks on mirrors. Resilience to key disclosure and server compromises are outside the scope of the current document.

If clients cannot access any mirrors, they will not be able to obtain updates. In situations where an attacker can prevent an individual client from communicating with mirrors, the client library is responsible for ensuring the attack is detected. However, it is the project's responsibility to ensure that their mirrors are online and available. An adversary who can successfully carry out a denial-of-service attack on all of a project's mirrors would be able to prevent clients from receiving critical security updates.

There is nothing the repository library can directly do to prevent denial-of-service attacks on mirrors. However, the repository is designed to enable projects to withstand such attacks to the best of their ability. The two aspects of the design critical to this are the support for multiple mirrors and the usage of only static files on mirrors. By using multiple mirrors, a denial-of-service attack must make inaccessible enough mirrors such that the remaining mirrors cannot service all clients. By using only static files on mirrors, denial-of-service attacks involving multiple

simultaneous client connections are more difficult because mirrors require minimal resources for each client.

Specific methods of protecting against denial-of-service attacks are outside the scope of the software update security framework. Such methods include protecting web servers through the use of special-purpose software and, in the event of more serious denial-of-service attacks, filtering traffic as early as possible through hardware firewalls and coordination with one's bandwidth providers.