CernVM-FS from a Security Perspective

Jakob Blomer

January 27, 2012

Contents

1 Introduction

The CernVM File System is a caching HTTP file system developed to deliver binary software directory trees. CernVM-FS is implemented as a FUSE module. It makes a specially prepared directory tree (*repository*) stored on a web server (*repository server*) look like a local read-only file system on the client machine. The delivered binaries on the repository server reflect usually the result of a `make install`. Figure 2 shows general idea of distributing software with CernVM-FS. Figure 3 shows how CernVM-FS interlocks with Fuse and a web server in order to deliver files.

A CernVM-FS repository is defined by its *file catalog*. The file catalog is an SQLite database having a single table that lists files and directories together with its meta data. A file catalog is identified by the SHA-1 key of the content of the SQLite database file. In addition to the POSIX meta data, the file catalog contains SHA-1 keys of the content of all regular files. These SHA-1 keys are used to access files in the *data store* (cf. Figure 1). In this respect, data organization in CernVM-FS is comparable to the Git version control system.

On the client side CernVM-FS requires only outgoing HTTP connectivity to a web server and/or an HTTP proxy server. Provided that the files are read only, CernVM-FS

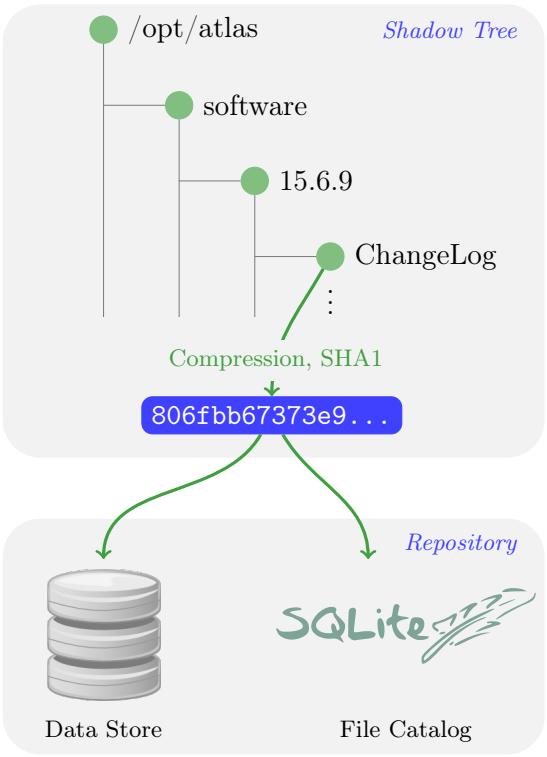

Figure 1: Converting a file system tree into a repository. The file catalog contains the directory structure as well as file metadata, symbolic links, and secure hash keys of regular files. Regular files are compressed and renamed to their secure hash key before copied into the data store.

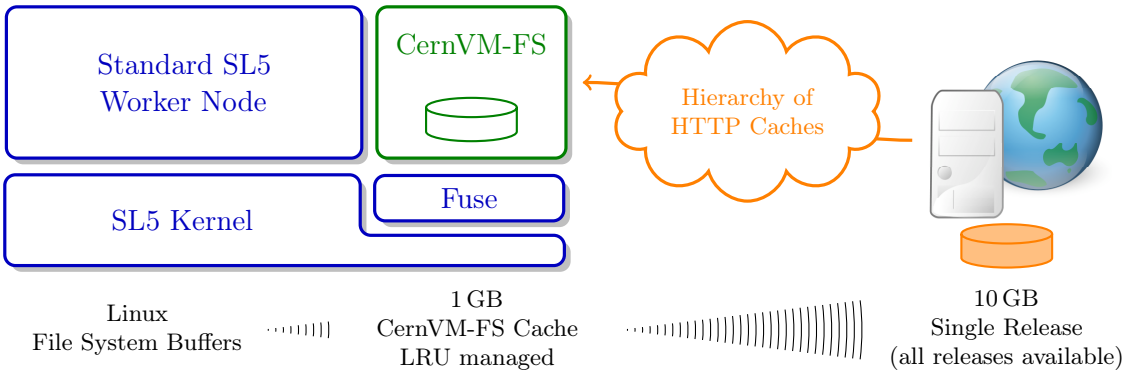

Figure 2: Architecture of CernVM-FS. Experiment software is loaded file by file on demand and is locally cached.

does local file data and file catalog caching. In case files are cached, they are served from the local disk cache. Otherwise they are loaded on first access file by file via the HTTP connection. The file data cache is LRU managed.

2 Security Objectives

The following lists assumptions and assertions of CernVM-FS:

1. CernVM-FS trusts the local cache directory. Although we provide a utility to check local cache integrity (`cvmfs_fsck`), during runtime CernVM-FS assumes that it has full and exclusive control over the cache directory and that everything in there is valid. The content of the cache directory is owned by user `cvmfs` and group `fuse`. It is not accessible by anybody else (except root).

2. Because once data is in the local cache it is considered to be valid, we need CernVM-FS to check incoming data before committing it to the cache. CernVM-FS relies on libcurl to handle HTTP downloads. Downloaded files and file catalogs are stored in a temporary directory and verified against their SHA-1 hash before they are committed to the cache. For signed file catalogs, the signature and the signing certificate are verified. For cryptographic routines, CernVM-FS relies on the libcrypto from OpenSSL. See Section 3 for details.

3. In case of unexpected failures, the impact of the failing CernVM-FS instance shall be as little as possible to the rest of the system. Therefore, CernVM-FS drops root privileges on start-up. Furthermore, it forks a watchdog process that creates a stack trace on failure allowing for post mortem analysis. Such failures as well as decisions about acceptance or rejection of signed file catalogs are logged to the syslog facility. See Section 4 for details.

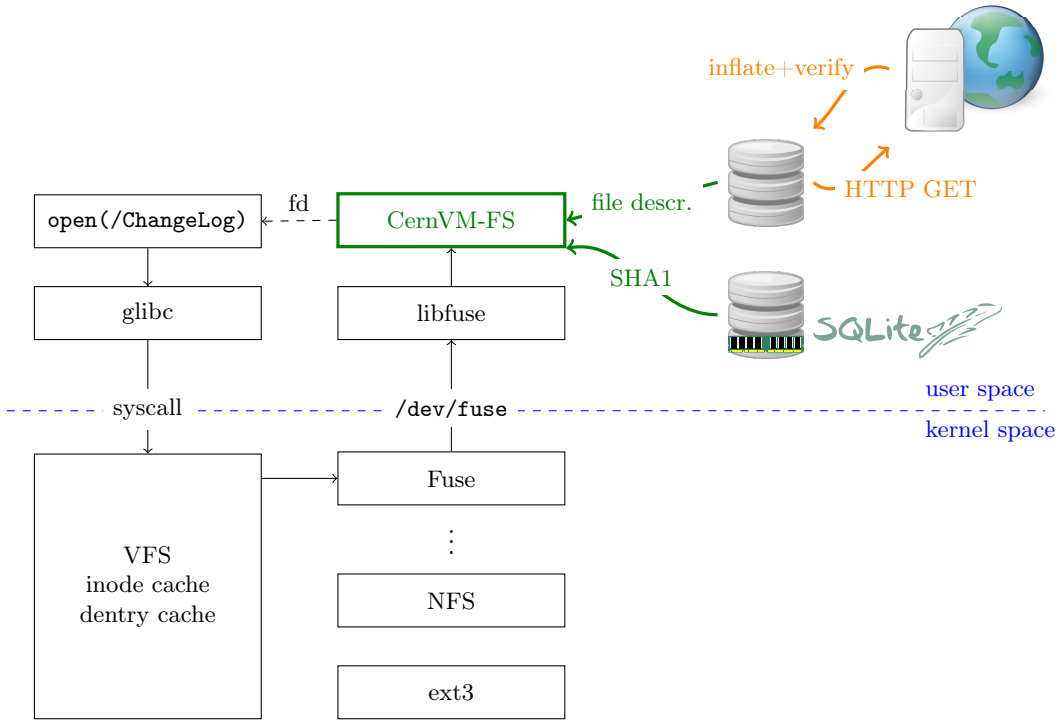

Figure 3: Process of opening a file. CernVM-FS resolves the name by means of an SQLite catalog, which is prepended by a direct-mapped memory cache. Downloaded files are verified against the secure hash of the corresponding catalog entry. The `read()` and the `stat()` system call can be entirely served from the in-kernel file system buffers.

CernVM-FS does *not* include any measure to prevent it from targeted attempts to stop it from fetching new files from the web server. Stopping CernVM-FS from fetching new files can be achieved by either tampering with the data packets or (easier) by blocking network access. However, CernVM-FS does include measures to prevent it from accidental network failures: it can take multiple web server host names as well as multiple HTTP proxy servers. CernVM-FS automatically tries another host or proxy server in case it encounters an error. In any case, as long as current data are available in the cache, CernVM-FS does not produce any network traffic.

CernVM-FS sends a custom HTTP header with all requests. That allows web servers and proxies to separate and manage CernVM-FS traffic, provided that no other network entities fake a CernVM-FS header.

3 Data Integrity and Authenticity

In order to provide authoritative information about a repository publisher, file catalogs may be signed by an X.509 certificate. It is important to note that it is sufficient to sign just the file catalog. Since every file is listed with an SHA-1 checksum inside the catalog, we gain a secure chain and may speak of a "signed repository". Figure 4 shows the trust chain with a signed repository.

The signature is created using the `cvmfs_sign` utility. The `cvmfs_sign` utility takes a X.509 certificate together with its private key in order to sign a catalog and its nested catalogs. On the client side, CernVM-FS supports a *secure mode* in which only validly signed catalogs are mounted.

In order to validate file catalog signatures, CernVM-FS uses a *white-list* procedure. The white-list contains the SHA-1 fingerprints of known publisher certificates and a timestamp. A white-list is valid for 30 days. It is signed by a private RSA key, which we refer to as CernVM *master key*. The master key is kept offline on a secure smart card at cern. So the security of the master key is ensured by 2-factor authentication (possession of the smart card and knowledge of the PIN). The smart card is used with a smart card reader with pin pad. Neither the key nor the pin are ever exposed outside the scope of those secure devices. The public RSA key that corresponds to the master key is distributed with the CernVM-FS sources and the CernVM-FS YUM/RPM package as well as with every instance of CernVM.

4 Implementation Notes

CernVM-FS is written in C and C++ with auxiliary scripts in bash and Perl. The CernVM-FS C/C++ code base is checked by PH-SFT's Coverity static code analyzer. CernVM-FS relies on a couple of standard libraries, all of them part of SL5 (cf. Figure 5). With CernVM-FS making heavy usage of SQLite, libcurl, and libfuse, several bugs were discovered in the versions shipped with SL5. So, in the RPM/YUM distribution these packages are statically linked newer versions.

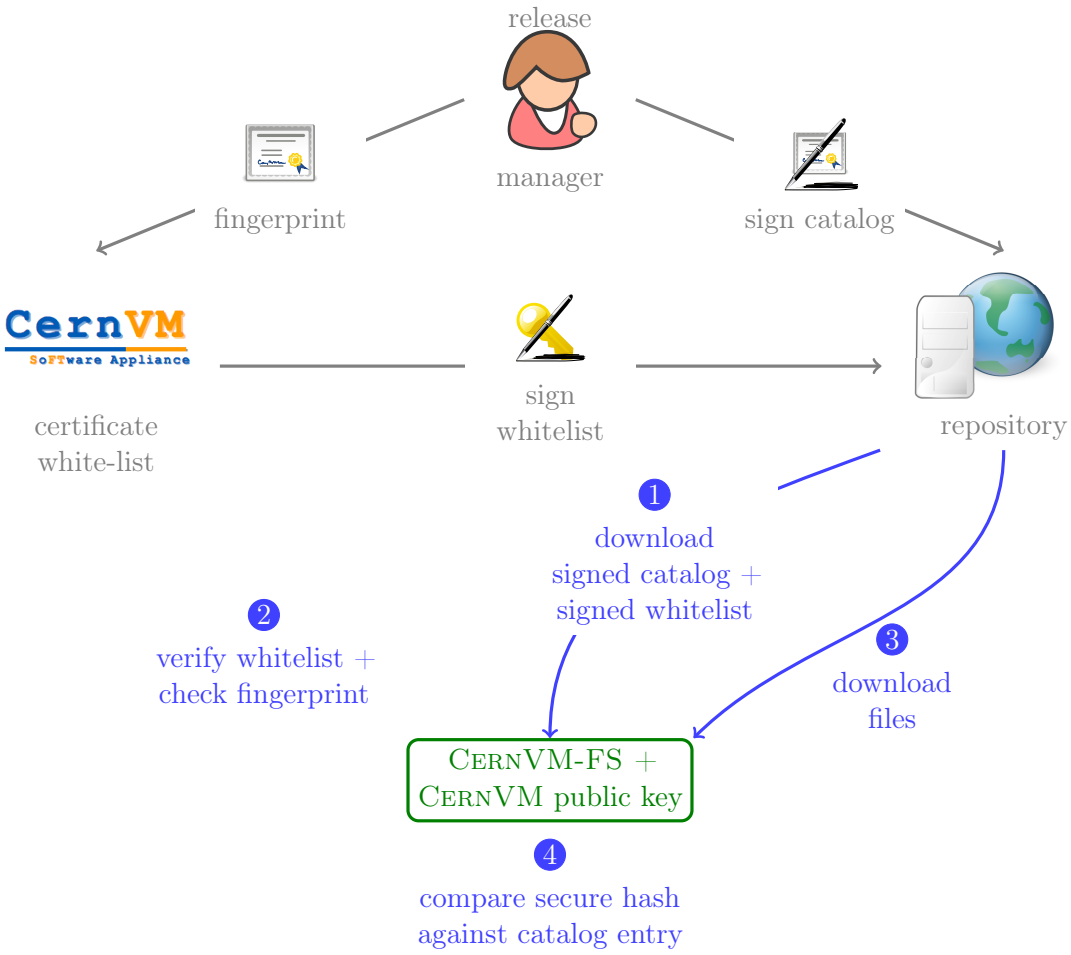

Figure 4: Trust chain with a signed repository.

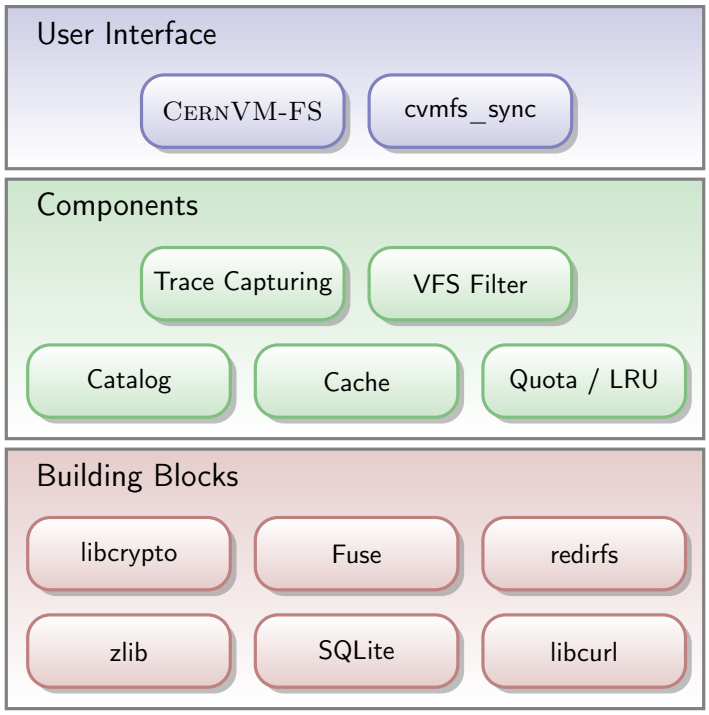

Figure 5: CernVM-FS block diagram.

The mount procedure is handled by AUTOMOUNT and results in invoking the `cvmfs2` process as follows:

1. A system call on a subdirectory of `/mnt/cvmfs` is forwarded to AUTOMOUNT
2. AUTOMOUNT looks for an entry for `/mnt/cvmfs` in its master map `/etc/auto.master`, which is set to `/etc/auto.cvmfs`
3. The CernVM-FS auto map reads the configuration from `/etc/cvmfs` and returns the mount option `-fstype cvmfs`
4. AUTOMOUNT calls `mount` which in turn calls the mount helper `/sbin/cvmfs.mount`
5. The mount helper reads the configuration from `/etc/cvmfs` and assembles the mount options string for the `cvmfs2` process. It starts the `cvmfs2` process and sets the file system type to `cvmfs`
6. CernVM-FS sets its umask to 007
7. CernVM-FS uses Fuse routines to parse the option string
8. CernVM-FS increases the maximum number of open files for the `cvmfs2` process (default: 32768). This is necessary because all `open()` requests by other processes on the mounted file system are channeled through the `cvmfs2` process.
9. CernVM-FS changes ownership of the mount point to user `cvmfs`. This is necessary to successfully mount as non-root.
10. CernVM-FS drops its privileges to user `cvmfs` and group `fuse`.
11. CernVM-FS continues initialization with dropped privileges. This includes spawning the watchdog process and the actual mount done by LIBFUSE.

Being a Fuse module, CernVM-FS implements a couple of Fuse call-back functions:

`stat` Requests for file attributes are entirely served from the cached file catalog. This callback also takes care of the catalog TTL. If the TTL is expired, the CernVM-FS checks for a newer catalog version, which is loaded on the fly if necessary (*re-mount*). Note that a re-mount might possibly break running programs. We rely on careful repository publishers that produce more or less immutable directory trees, i. e. new repository versions just add files.

`readlink` A symlink is served from the file catalog. As a special extension, CernVM-FS detects environment variables in symlink strings written as `$(VARIABLE)`. Those variables are expanded by CernVM-FS dynamically on access. This way, a single symlink can point to different locations depending on the environment. This is helpful, for instance, to dynamically select software package versions residing in different directories. The evaluation of the environment variable is done in the context of the `cvmfs2` process, i. e. it is determined as of `cvmfs2` process creation.

`readdir` A directory listing is served by an SQL query on the file catalog.

`open / read` The `open()` call has to provide Fuse with a file handle for a given path name. In CernVM-FS file requests are always served from the disk cache. If the file is not in the disk cache, it is downloaded from the repository first into a temporary path. CernVM-FS uses `mkstemp()` for temporary files. Once verified, the downloaded file is atomically committed to the cache by a `rename()` call. If the verification fails, the temporary file is unlinked and and I/O error is returned.

The Fuse file handle is a read-only file descriptor valid in the context of the CernVM-FS process. It points into the disk cache directory. Read requests are translated into the `pread()` system call.

CernVM-FS is capable of communicating to the user at runtime. Thereby certain parameters can be checked and configured while the file system is mounted. To do so, CernVM-FS sets up a temporary socket in the cache directory, `cvmfs_io`. Access is controlled by the access rights to this file, which is owned by user `cvmfs` and group `fuse` and has mode 660. The communication is based on a simple command-response scheme. The `cvmfs-talk` Perl script takes care of writing to and reading from the socket.

5 Customization

CernVM-FS comes with a default configuration installed in `/etc/cvmfs`. This configuration consists of parameters and mount/unmount helper functions. The local customization of CernVM-FS is controlled by "plug-in" files in `/etc/cvmfs`. For the repositories at CernVM we provide default customizations by the separate RPM/YUM package `cvmfs-init-scripts`.

6 Distribution

CernVM-FS is distributed as YUM package via our YUM repository at http://cvmrepo.web.cern.ch/cvmrepo/yum. The integrity of the YUM repository is verified by a GPG key (http://cvmrepo.web.cern.ch/cvmrepo/yum/RPM-GPG-KEY-CernVM). There are as well corresponding RPMs and source tarballs of CernVM-FS available at the HTTPS location https://cernvm.cern.ch/project/trac/cernvm/downloads.