FERMILAB-13-CONF-369-CD

Security in the CernVM File System and the Frontier Distributed Database Caching System

D Dykstra[1] **and J Blomer**[2]
[1] Scientific Computing Division, Fermilab, Batavia, IL 60510, USA
[2] PH-SFT Department, CERN, Geneva, Switzerland

E-mail: dwd@fnal.gov or jblomer@cern.ch

Abstract. Both the CernVM File System (CVMFS) and the Frontier Distributed Database Caching System (Frontier) distribute centrally updated data worldwide for LHC experiments using http proxy caches. Neither system provides privacy or access control on reading the data, but both control access to updates of the data and can guarantee the authenticity and integrity of the data transferred to clients over the internet. CVMFS has since its early days required digital signatures and secure hashes on all distributed data, and recently Frontier has added X.509-based authenticity and integrity checking. In this paper we detail and compare the security models of CVMFS and Frontier.

1. Introduction

The CernVM File System [1] (CVMFS) and the Frontier Distributed Database Caching System [2] (Frontier) are two systems for providing copies of data to hundreds of thousands of jobs on a worldwide computing grid of tens of thousands of computers. CVMFS is designed to distribute software releases, and Frontier is designed to distribute the results of database queries. Both have been deployed by the large LHC collaborations ATLAS and CMS on the Worldwide LHC Computing Grid (WLCG), in addition to other collaborations. Both systems use RESTful http protocols for transport and make use of http proxy caches at each site to enable scaling to such large numbers of computers and long distances. Because the data has to be read in so many locations, and because it contains no information that needs to be kept private, the data is publicly available; there is no access control on reading it. However, it is important that the data is not tampered with during transport. For this reason, both systems provide mechanisms for clients to securely verify the integrity and authenticity of the data, to ensure that the data received matches what was originally sent by the trusted servers. Write access to the original data the servers read is controlled through separate secure mechanisms that are beyond the scope of this paper.

Section 2 contains the details of the CVMFS authenticity and integrity checking mechanism that has been in production for a number of years. Section 3 has the corresponding Frontier mechanism that has recently been implemented. Section 4 compares the CVMFS and Frontier security mechanisms.

2. CVMFS Authenticity and Integrity Checking

Since CVMFS distributes executable software, the need for checking the authenticity and integrity of the transferred data is obvious: if someone inserted a fake server or tampered with the transferred data,

Operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy

they could easily take over enormous computing resources by inserting their own code. Figure 1 shows the components and information flow in CVMFS authenticity and integrity checking.

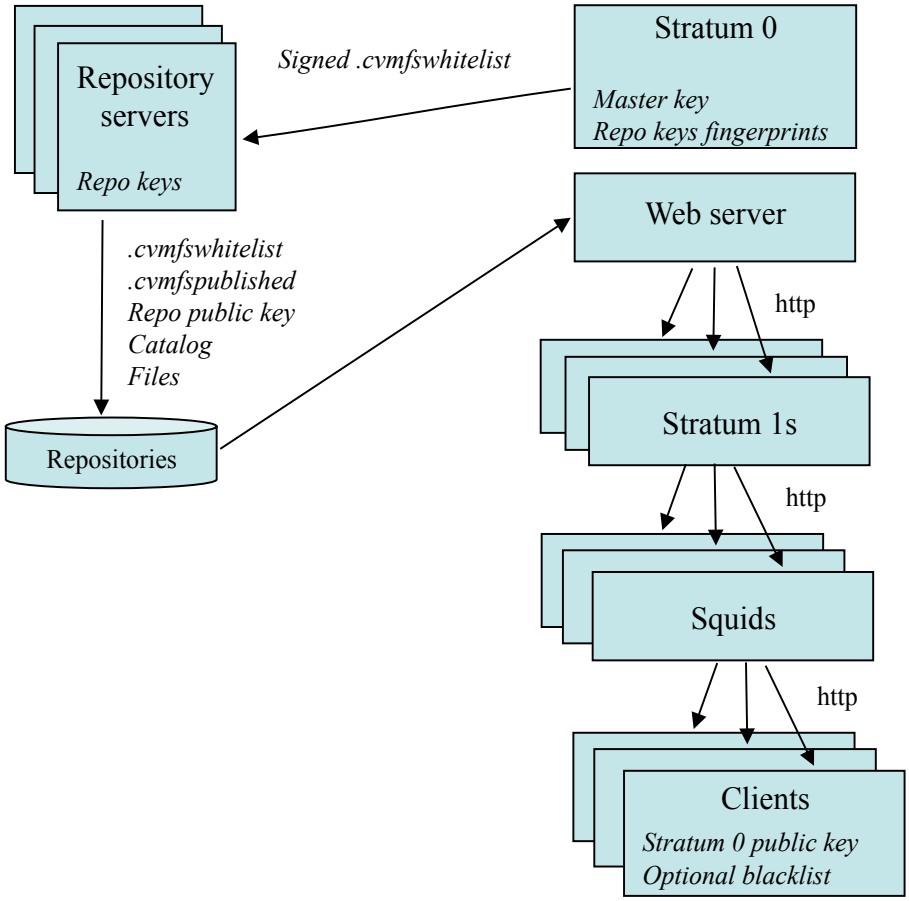

Figure 1: CVMFS authenticity and integrity checking components and flow

This is a summary of how CVMFS authenticity and integrity checking works:

1) There is a public/private cryptographic key pair for each CVMFS "Stratum 0" computer. A Stratum 0 computer corresponds to an internet domain name, for example cern.ch. The public half of the pair is distributed to every CVMFS client computer, usually via signed rpms. The private half, known as the "master key," is used only on the Stratum 0 computer.
2) There is another public/private cryptographic key pair for each CVMFS repository, for example for atlas.cern.ch and cms.cern.ch. Once every 30 days or less, for each of its repositories the Stratum 0 generates and signs using the master key a file called ".cvmfswhitelist" that contains the secure fingerprint(s) of the repository's public key(s) with an expiration date 30 days later, and sends the file to the computer that publishes the repository. The private half of the key stays on the repository server, and the public half is distributed like other files in the repository.
3) For every publish operation, the repository server stores each new file under a name that is the secure hash of the contents of the file, and creates a catalog translating all original file names to the secure names. The catalog then becomes a file distributed like other files in the repository. The publish operation also creates and digitally signs with the repository's private key a file called ".cvmfspublished" which contains the secure hash of the catalog and the secure hash of the repository's public key file. The repository server then makes all the files

available including .cvmfswhitelist on a web server (which is typically on the Stratum 0 computer but doesn't have to be). The files are replicated on Stratum 1 servers and cached on intermediate squids, but that has no impact on the security.

4) CVMFS clients confirm that the signature on .cvmfswhitelist is a valid signature of the Stratum 0's public key which has not expired, that the enclosed fingerprint matches the fingerprint of the repository's public key which it finds by the hash in .cvmfspublished, and that the signature on .cvmfspublished is valid based on that repository's public key. This establishes authenticity, and from then on all accesses are only to files that are named by secure hashes. Every time one of the files is downloaded, including catalogs, CVMFS clients confirm that the secure hash of the contents matches the name, which establishes the file's integrity. The signatures on .cvmfswhitelist and .cvmfspublished are verified at the time the repository is mounted and when they're re-downloaded, which happens when repositories are accessed after an expiration time (typically 60 minutes).

The rest of this section gives more details about the CVMFS authenticity and integrity checking implementation.

2.1. Cryptographic functions

The cryptographic library that CVMFS uses is libcrypto from the OpenSSL project [3]. The Stratum 0 public/private key is a currently a 2048-bit RSA key, but any size RSA key is supported. The secure hash function for files is SHA1.

The digital signatures on .cvmfswhitelist and .cvmfspublished are simply an RSA encryption of the SHA1 hash of the first part of the file (before the line containing "—") using the appropriate private key; the signatures are verified by decrypting the signature with the public key and comparing the result to the hash of the first part of the file.

The repository servers' keys are in X.509 format, for the convenience of including extra information such as the name of the publisher. They also use 2048-bit RSA keys and work with other sizes. Certifying authorities and expiration times are not checked, however. Also, only SHA1 fingerprints are currently supported.

2.2. Stratum 0 keys

Since the Stratum 0 private master key is only used about once a month, to protect against the possibility of it getting stolen it can be kept offline on a smartcard and not kept online anywhere. The cern.ch master key is managed this way.

Clients can be configured with more than one public key for each Stratum 0. This is especially useful for when the master is only kept on a smartcard; if the card malfunctions, there can still be a backup with another master key.

2.3. Details of .cvmfswhitelist

This are the items in .cvmfswhitelist relevant to security:

1) *The expiration timestamp*. Clients make sure this time isn't in the past.
2) *The repository name*. Clients make sure the name matches the expected name.
3) One or more repository key fingerprints. Clients make sure that the key referred to in .cvmfspublished matches one of these fingerprints.

2.4. Details of .cvmfspublished

These are the items in .cvmfspublished relevant to security:

1) *The hash of the root catalog*. Clients use this hash to find the catalog, and then use the catalog to find the files in it and their hashes. They make sure all hashes match file contents. Catalogs may also refer to other, nested catalogs, so the first one is called the root catalog.
2) *The name of the repository*. Clients make sure the name matches the expected name.

3) *The number of seconds to cache .cvmfs* files.* Clients re-load and re-verify signatures on .cvmfswhitelist and .cvmfspublished after this number of seconds have elapsed.
4) *The version number of the repository.* Clients make sure the number does not decrease, to prevent replay attacks where someone tries to insert an old version.
5) *The timestamp of repository publication.* Clients make sure the timestamp does not decrease, again to prevent replay attacks.
6) *The hash of the repository's public key.* Clients use this hash to find the key (actually an X.509 certificate), and make sure its fingerprint is one of the ones validated by .cvmfswhitelist.

2.5. The client blacklist

If a repository signing key is compromised, and the .cvmfswhitelist is updated to remove the fingerprint corresponding to that key, there is still a possibility that for as long as 30 days the key thief could combine the compromise with a replay attack using an old .cvmfswhitelist that still lists the compromised key as valid, and publish data using the stolen key. For that reason, CVMFS clients also honor an optional /etc/cvmfs/blacklist which lists key fingerprints to reject, in the same format as the fingerprints in .cvmfswhitelist.

2.6. X.509 plan

Since the current CVMFS authenticity and integrity checking mechanism is weak on revocation if there is a compromised key, plus since signing the .cvmfswhitelist files every 30 days can be problematic if it is a manual operation, there is a plan to introduce an option of X.509 verification. That would require having Certifying Authority (CA) certificates and Certificate Revocation Lists (CRLs) on each client, which is already common practice on the WLCG. A file similar to .cvmfswhitelist (e.g. .cvmfsXwhitelist) would be added that is signed by a master X.509 certificate's key. Rather than distributing the Stratum 0's public key to clients, clients would then be configured with either the Distinguished Name (DN) of the key or the DNS name of the Stratum 0 for validation of the Stratum 0's certificate. Each repository would also distribute the Stratum 0's certificate like they currently do with the repository's certificate, and reference it from .cvmfsXwhitelist. Full X.509 validation of the repository certificate would also be introduced so they could be revoked if necessary. Instead of listing the fingerprint of the certificate, .cvmfsXwhitelist would list the certificate's DN..

3. Frontier Authenticity and Integrity Checking

Since Frontier only distributes data from a database which is not executable code, it isn't as obvious what an attacker could gain by tampering with the data. This is why up until recently there was no cryptographic authenticity and integrity checking mechanism for Frontier. However, the data is used by complex code that assumes the data is well-formatted, so if someone were to carefully analyze the application there is likely to be a way to insert code by overflowing buffers. Figure 2 shows the components and information flow in the new Frontier authenticity and integrity checking mechanism.

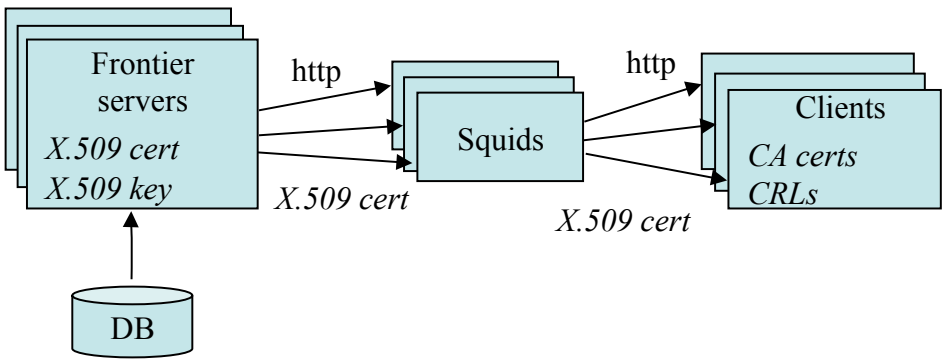

Figure 2: Frontier authenticity and integrity checking components and flow

This is how Frontier authenticity and integrity checking works:

1) When a client with authenticity and integrity checking enabled starts reading from a server, it first requests the X.509 certificate of that server. It then confirms that the certificate is valid according to the CA certificates it has, confirms that the server certificate has not been revoked by a CRL, and performs all other standard OpenSSL X.509 certificate verifications including making sure the certificate has not expired. It also confirms that the server name is listed in the certificate's Common Name or one of the certificate's alternate DNS names. The client then extracts the RSA key from the certificate, which can be of any length supported by OpenSSL.
2) Then for all queries, the Frontier client adds a parameter to each requested URL that asks that the response be signed by the server. The server calculates a SHA-256 hash covering the requested URL and the response data, encrypts the hash using the RSA key from it's X.509 private key, and attaches the result to the end of the response as a signature.
3) The client calculates the SHA-256 hash of the URL it sent and of the data in each response, decodes the signature using the server certificate's RSA key, and confirms that the decoded result matches the hash. Note that this confirms that the response is a valid response for the corresponding URL, but it does not guard against replay attacks. The assumption is that previous valid responses for the same request will do little harm, and there would be little motivation for an attacker to insert an old response.

4. Comparison of the CVMFS & Frontier Security Mechanisms

CVMFS & Frontier use the same caching infrastructure and both verify authenticity and integrity of data with digital signatures. Since CVMFS has the ability to pre-prepare the served data, it needs only two digital signatures for the whole repository: one from the Stratum 0 to authenticate the repository's key, and one on the whole published version of the repository. Everything else is authenticated by catalogs of secure hashes when downloading, and doesn't need to be re-authenticated if a file is accessed more than once from the client cache, so the performance impact is very low.

Frontier, on the other hand, cannot prepare digital signatures ahead of time for the results of every possible query, so it has to dynamically sign the results as they're read. The RSA encryption algorithm used for signatures is a very slow process, so especially for small queries this is a high overhead on the Frontier server, as much as a factor of ten. Fortunately, the high hit rate on caching, especially when expired caches are reused with If-Modified-Since [4], results in only a very small percentage of the total queries getting processed on the Frontier servers, so the existing servers have plenty of capacity. Decrypting the RSA signature on the client side also has some overhead, but it is much less than encrypting and it doesn't significantly impact client performance.

The Frontier mechanism is simpler and easier to explain, but it isn't as efficient as the CVMFS mechanism.

Frontier uses X.509 for authentication, and CVMFS currently uses only lower-level OpenSSL authentication mechanisms, but CVMFS is expecting to add X.509 capability.

5. Conclusions

CVMFS and Frontier both have effective mechanisms for ensuring that data they read over unsecured networks match what is sent by small numbers of shared servers. The CVMFS mechanism is highly efficient, and the Frontier mechanism is adequately efficient.

Acknowledgements

Fermilab is operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy.

References

[1] Blomer J, Buncic P, Charalampidis I, Harutyunyan A, Larsen D, and Meusel R 2012 Status and future perspectives of CernVM-FS *J. Phys.: Conf. Ser.* **396** 052013

[2] Dykstra D 2011 Scaling HEP to Web Size with RESTful Protocols: The Frontier Example *J. Phys.: Conf. Ser.* **331** 042008

[3] Viega J, Messier M, and Chandra P 2002 *Network Security with OpenSSL* (Sebastapol, CA: O'Reilly)

[4] Dykstra D and Lueking L 2010 Greatly improved cache update times for conditions data with Frontier/Squid *J. Phys.: Conf. Ser.* **219** 072034