Design Document
For
Shared Key Authentication and Encryption
in Lustre 2.x

Scope Statement

Revision History

Date	**Revision**	**Author**
2012-07-22	Created	ajk

Table of Contents

Introduction

The Lustre filesystem currently supports Kerberos authentication to protect file data and metadata from network eavesdropping and tampering. While Kerberos is an excellent authentication protocol, it requires some infrastructure at the client end that is not trivial to deploy nor always permissible by site policy.

A coexisting shared-key, host-based authentication system would preserve the authenticity, integrity, and confidentiality of file data and metadata but without requiring any additional infrastructure on the client side. In such a system, a single key would be generated for each client host, and that key would be installed on both the client and the server.

Lustre’s Kerberos support is provided through the Generic Security Services Application Program Interface (GSSAPI), a modular interface for providing authentication and encryption mechanisms. By implementing a shared-key system as a GSSAPI mechanism, Lustre’s Kerberos functionality can be left untouched, and no new authentication and encryption code needs to be added.

Definitions

authenticity

the assurance, when communicating with another party, that that party is who it claims to be (not an impostor)

confidentiality

the assurance that data will not be disclosed between sender and receiver (*e.g.,* protection against eavesdropping)

GSSAPI

Generic Security Services Application Program Interface, a modular interface for providing authentication and encryption mechanisms

integrity

the assurance that data will not be modified between sender and receiver

Kerberos

a network authentication protocol

key management

a system that allows creation of authentication/encryption secrets (keys) and making them available to a service such as Lustre

security flavor

Lustre construct that allows a user to choose the level of security desired (*e.g.,* authentication alone, authentication plus encryption)

shared-key authentication

an authentication method using a single secret password or key stored on both the client and the server

Changes from Solution Architecture

None.

Functional specification

Shared Key Null Security Flavor

Implement a null mechanism security flavor (*sknull*) within Lustre 2.X to provide a method for testing. The *sknull* flavor will use the null GSSAPI mechanism, also included in this design.

Null GSSAPI Mechanism

Implement a null mechanism for GSSAPI. A null mechanism not only provides easy testing but also illustrates the critical path through the GSSAPI model and provides a baseline for performance testing of other security mechanisms.
The null mechanism will use no cryptography for authentication or encryption. It is intended for testing only.

Shared Key Security Flavors for Data Integrity and Privacy

Implement shared-key integrity (*ski*) and privacy (*skpi*) security flavors for Lustre 2.X. Keys will be pulled from a secure module such as the Linux kernel keyring. Keys are per-cluster; a given key may only be used on the cluster with the clusterid for which the key was issued. This restriction will be implemented by using the hash of the clusterid as part of the authentication process; hence, if a key is moved to a different cluster with a different clusterid, authentication will fail. These flavors will use the shared key GSSAPI mechanism, also included in this design.

Shared Key GSSAPI Mechanism

Implement a shared-key GSSAPI mechanism for preserving data integrity and privacy. This phase should use the existing kernel crypto modules to allow for a shared key, known to both the Lustre 2.X servers and the Lustre 2.X clients, to generate a hash-based message authentication code (HMAC), the equivalent of a digital signature, for each message passed between the

server and the client. The mechanism will also optionally encrypt these messages using the shared key and a symmetric cipher.
Generating an HMAC requires a cryptographic hash function. The hash algorithm will be specifiable at run time to the extent permitted by the Linux crypto API, defaulting to SHA-256.
When symmetric encryption is used (*skpi*), the default cipher will be the integer counter mode (CTR) of the Advanced Encryption Standard (AES). CTR was chosen because it doesn't require padding. The Linux crypto API allows other AES modes and other ciphers, but since the initial release of this GSSAPI mechanism may not support padding, some of those modes and ciphers may not be usable.

Userspace Key Management Tools

Implement userspace tools to create shared keys, and methods to make the keys available to Lustre without exposing them. The utilities to create the keys should be implemented in userspace. The tools to push the keys into a secure memory module for accessibility to the Lustre modules should be incorporated into *lctl*.

Logic specification

To be done.

Configurable parameters

The following aspects of functionality will be configurable:

- shared key security flavor: none (*sknull*, for testing only), data integrity only (*ski*), data integrity and privacy (*skpi*)
- HMAC hash function (defaults to SHA-256)
- symmetric cipher (defaults to AES-CTR)

API and Protocol Changes

None known.

Open issues

None known.

Risks and Unknowns

Historically, the GSSAPI has served mainly as a way to provide Kerberos support without locking it in as the only possibility for authentication and encryption. For that reason, not many non-Kerberos mechanisms exist. But because the shared key approach is simpler than Kerberos, we don't anticipate encountering any gaps in GSSAPI.