SESSION AUTHENTICATION FOR THE CEPHX PROTOCOL

Peter Reiher 7/30/12

The original Cephx protocol authenticated the client to the authenticator and set up a session key used to authenticate the client to the server he needs to talk to. It did not, however, authenticate the ongoing messages between the client and server. Based on the fact that they share a secret key, these ongoing session messages can be easily authenticated by using the key to sign the messages.

This document describes changes to the code that allow such ongoing session authentication. The changes allow for future changes that permit other authentication protocols (and the existing null NONE and UNKNOWN protocols) to handle signatures, but the only protocol that actually does signatures, at the time of the writing, is the Cephx protocol.

INTRODUCTION

This code comes into play after the Cephx protocol has completed. At this point, the client and server share a secret key. This key will be used for authentication. For other protocols, there may or may not be such a key in place, and perhaps the actual procedures used to perform signing will be different, so the code is written to be general.

The "session" here is represented by an established pipe. For such pipes, there should be a `session\_security` structure attached to the pipe. Whenever a message is to be sent on the pipe, code that handles the signature for this kind of session security will be called. On the other end of the pipe, code that checks this kind of session security's message signatures will be called. Messages that fail the signature check will not be processed further. That implies that the sender had better be in agreement with the receiver on the session security being used, since otherwise messages will be uniformly dropped between them.

The code is also prepared to handle encryption and decryption of session messages, which would add secrecy to the ingrity provided by the signatures. No protocol currently implemented encrypts the ongoing session messages, though.

For this functionality to work, several steps are required. First, the sender and receiver must have a succesful run of the cephx protocol to establish a shared key. They must store that key somewhere that the pipe can get at later, to permit messages to be signed with it. Sent messages must be signed, and received messages must have their signatures checked.

The signature could be computed in a variety of ways, but currently its size is limited to 64 bits. A message's signature is placed in its footer, in a field called `sig`.

The signature code in Cephx can be turned on and off at runtime, using a Ceph boolean option called `cephx\_sign\_messages`. It is currently set to false, by default, so no messages will be signed. It must be changed to true to cause signatures to be calculated and checked.

STORING THE KEY

The key is needed to create signatures on the sending end and check signatures on the receiving end. In the future, if asymmetric crypto is an option, it's possible that two keys (a private one for this end of the pipe and a public one for the other end) would need to be stored. At this time, messages going in both directions will be signed with the same key, so only that key needs to be saved.

The key is saved when the pipe is established. On the client side, this happens in `connect()`, which is located in `msg/Pipe.cc`. The key is obtained from a run of the Cephx protocol, which results in a successfully checked authorizer structure. If there is such an authorizer available, the code calls `get\_auth\_session\_handler()` to create a new authentication session handler and stores it in the pipe data structure. On the server side, a similar thing is done in `accept()` after the authorizer provided by the client has been verified.

Once these things are done on either end of the connection, session authentication can start.

These routines (`connect()` and `accept()`) are also used to handle situations where a new session is being set up. At this stage, no authorizer has been created yet, so there's no key. Special cases in the code that calls the signature code skip these calls when the `CEPH\_AUTH\_UNKNOWN` protocol is in use. This protocol label is on the pre-authorizer messages in a session, indicating that negotiation on an authentication protocol is ongoing and thus signature is not possible. There will be a reliable authentication operation later in this session before anything sensitive should be passed, so this is not a security problem.

SIGNING MESSAGES

Messages are signed in the write_message call located in msg/Pipe.cc. The actual signature process is to encrypt the CRCs for the message using the shared key. Thus, we must defer signing until all CRCs have been computed. The header CRC is computed last, so we call sign_message() as soon as we've calculated that CRC.

sign_message() is a virtual function defined in auth/AuthSessionHandler.h. Thus, a specific version of it must be written for each authentication protocol supported. Currently, only UNKNOWN, NONE and CEPHX are supported. So there is a separate version of sign_message() in auth/unknown/AuthUnknownSessionHandler.h, auth/none/AuthNoneSessionHandler.h and auth/cephx/CephxSessionHandler.cc. The UNKNOWN and NONE versions simply return 0, indicating success.

The CEPHX version is more extensive. It is found in auth/cephx/CephxSessionHandler.cc. The first thing done is to determine if the run time option to handle signatures (see above) is on. If not, the Cephx version of sign_message() simply returns success without actually calculating a signature or inserting it into the message.

If the run time option is enabled, sign_message() copies all of the message's CRCs (one from the header and three from the footer) into a buffer. It calls encode_encrypt() on the buffer, using the key obtained from the pipe's session_security structure. 64 bits of the encrypted result are put into the message footer's signature field and a footer flag is set to indicate that the message was signed. (This flag is a sanity check. It is not regarded as definitive evidence that the message was signed. The presence of a session_security structure at the receiving end requires a signature regardless of the value of this flag.) If this all goes well, sign_message() returns 0. If there is a problem anywhere along the line and no signature was computed, it returns SESSION_SIGNATURE_FAILURE.

CHECKING SIGNATURES

The signature is checked by a routine called check_message_signature(). This is also a virtual function, defined in auth/AuthSessionHandler.h. So again there are specific versions for supported authentication protocols, such as UNKNOWN, NONE and CEPHX. Again, the UNKNOWN and NONE versions are stored in auth/unknown/AuthUnknownSessionHandler.h and auth/none/AuthNoneSessionHandler.h, respectively, and again they simply return 0, indicating success.

The CEPHX version of check_message_signature() performs a real signature check. This routine (stored in auth/cephx/CephxSessionHandler.cc) exits with success if the run time option has disabled signatures. Otherwise, it takes the CRCs from the header and footer, encrypts the result, and compares it to the signature stored in the footer. Since an earlier routine has checked that the CRCs actually match the contents of the message, it is unnecessary to recompute the CRCs on the raw data in the message. The encryption is performed with the same encode_encrypt() routine used on the sending end, using the key stored in the local session_security data structure.

If everything checks out, the CEPHX routine returns 0, indicating succcess. If there is a problem, the routine returns SESSION_SIGNATURE_FAILURE.

ADDING NEW SESSION AUTHENTICATION METHODS

For the purpose of session authentication only (not the basic authentication of client and server currently performed by the Cephx protocol), in addition to adding a new protocol, that protocol must have a sign_message() routine and a check_message_signature routine. These routines will take a message pointer as a parameter and return 0 on success. The procedure used to sign and check will be specific to the new method, but probably there will be a session_security structure attached to the pipe that contains a cryptographic key. This structure will be either an AuthSessionHandler (found in auth/AuthSessionHandler.h) or a structure derived from that type.

ADDING ENCRYPTION TO SESSIONS

The existing code is partially, but not fully, set up to allow sessions to have their packets encrypted. Part of adding encryption would be similar to adding a new authentication method. But one would also need to add calls to the encryption and decryption routines in write_message() and read_message(). These calls would probably go near where the current calls for authentication are made. You should consider whether you want to replace the existing calls with something more general that does whatever the chosen form of session security requires, rather than explicitly saying sign or encrypt.

SESSION SECURITY STATISTICS

The existing Cephx authentication code keeps statistics on how many messages were signed, how many message signature were checked, and how many checks succeeded and failed. It is prepared to keep similar statistics on encryption and decryption. These statistics can be accessed through the call printAuthSessionHandlerStats in auth/AuthSessionHandler.cc.

If new authentication or encryption methods are added, they should include code that keeps these statistics.