CIM Server Security
-
Local CIM Clients Authentication

Author:
Bapu Patil (bapu_patil@hp.com)
Hewlett-Packard Company

CIM Server Security: Local CIM Clients Authentication

Purpose

Purpose of this document is to provide an overall overview of how authentication is handled by CIM Server to authenticate local CIM Client applications. This is to ensure the user of the process has permissions to use the local CIM Clients. This document however will not include how the authentication is handled for remote CIM Clients.

Authentication

Authentication is a process of verifying the identity of a user, or other entity in a computer system, as a prerequisite to allowing access to the resources in a system.

Requirements

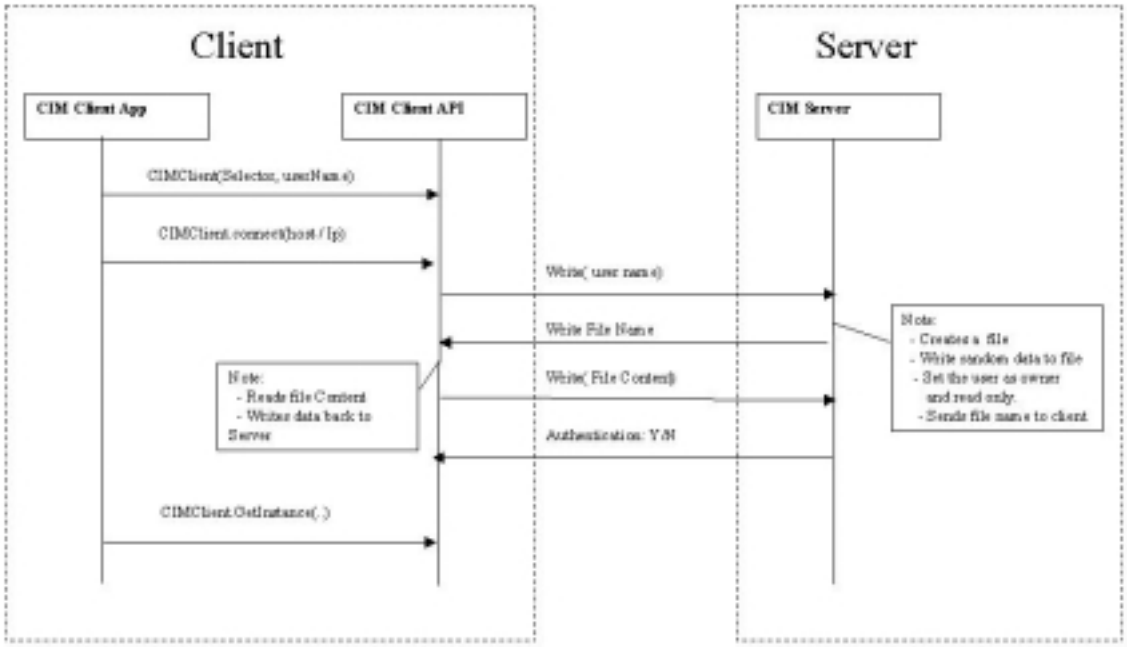
- Support authentication over local communication channel.
- Ensure CIM requests are authenticated for the logged in user without the client providing a password.

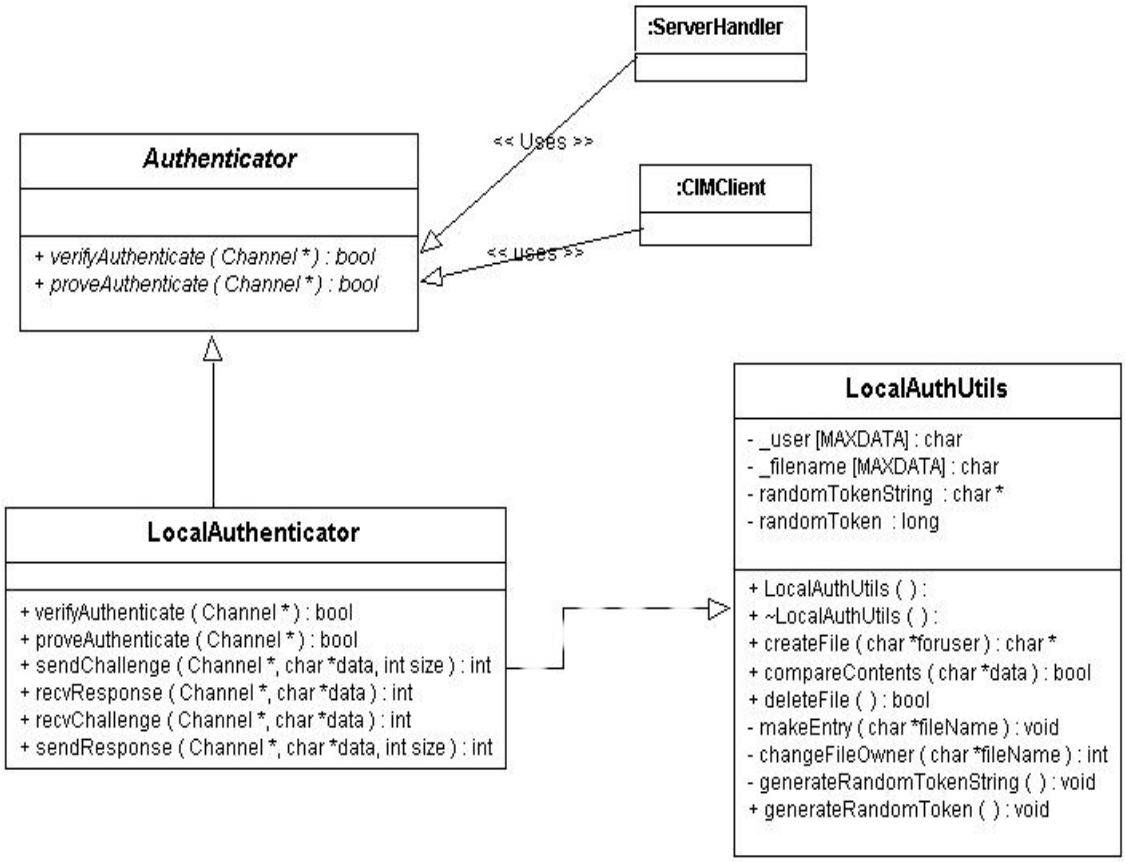
Authentication over Local Channel connection from the Clients

To ensure security over Client Local Channel connection, there will be a local authentication security mechanism designed where the CIM clients would require opening a session before sending any CIM requests to the CIM Server. The session opening will include local clients passing user name to CIM Server for authentication purposes. CIM Server will challenge the local CIM Client to make sure that the client user has permission to use the application. These local clients will be authenticated using the HP-UX file system security mechanism.

Local CIM Clients wish to connect to CIM Server must pass user name to CIM Server. CIM Server will use specified user and will create a file on local system (file name will include a uniquely generated identification), generates a random data and writes to this file. It then sets the client user as the owner of the file with read-only permission for only this user. CIM Server will then send this file name to CIM Client. CIM Client should read the file and send the content over to CIM Server. CIM Server will authorize the client and open the Channel to communicate.

Design Overview




:ServerHandler
Authenticator
+ verifyAuthenticate (Channel *) : bool
+ proveAuthenticate (Channel *) : bool
<< Uses >>
:CIMClient
<< uses >>
LocalAuthUtils
- _user [MAXDATA] : char
- _filename [MAXDATA] : char
- randomTokenString : char *
- randomToken : long
+ LocalAuthUtils ( ) :
+ ~LocalAuthUtils ( ) :
+ createFile (char *foruser) : char *
+ compareContents (char *data) : bool
+ deleteFile ( ) : bool
- makeEntry (char *fileName) : void
- changeFileOwner (char *fileName) : int
- generateRandomTokenString ( ) : void
+ generateRandomToken ( ) : void
LocalAuthenticator
+ verifyAuthenticate (Channel *) : bool
+ proveAuthenticate (Channel *) : bool
+ sendChallenge (Channel *, char *data, int size) : int
+ recvResponse (Channel *, char *data) : int
+ recvChallenge (Channel *, char *data) : int
+ sendResponse (Channel *, char *data, int size) : int

Calling Local Authentication on CIM Server Side

//

// Here is a sample code that will call Local authentication on Server side. This code will go in

// Pegasus/Server/ClientConnection.cpp

//

```
Boolean ClientConnection::handleOpen(Channel* channel)
{
   _channel = channel;
   DDD(cout << "Handle Open" << endl;)

   // Handles File system based local authentication
   Authenticator* authHandler  =
               (Authenticator *)new LocalAuthenticator();

    // Block for authentication
    _channel->enableBlocking();
    Boolean authorized = authHandler->verifyAuthenticate(_channel);
    _channel->disableBlocking();

    if ( authorized != true )
    {
         // we just send false
         Logger::put(Logger::TRACE_LOG,
                    "ServerHandler",Logger::INFORMATION,
                     "Authentication : Not a Valid Client ");
         return false;
    }
    else
    {
          Logger::put(Logger::TRACE_LOG,
                     "ServerHandler",Logger::INFORMATION,
                      "Authentication : A valid client ");

          return Handler::handleOpen(channel);
    }
}
```

Calling Local Authentication on CIM Client API Side

```
//
// Here is a sample code that will call Local authentication on client side. This code will go in
//  Pegasus/Client/CIMClient.cpp
//
/**
 *
 * CIMClient - used by local clients applications that wish to connect.
 * The client may pass the process owner name or this will use
 * the  current process owner id.
 *
 */
CIMClient::CIMClient(char* userName,
                     Selector* selector,
                     Uint32 timeOutMilliseconds)
                     : _channel(0),
_timeOutMilliseconds(timeOutMilliseconds)
{
   _selector = selector;

    if (userName != NULL || strlen(userName) > 0)
    {
      _user = userName;
    }
    else
    {
       // If the user name is null we will get current
       // process owner
       // Should write getCurrentLoginName() in Systems.
       //
       _user = System::getCurrentLoginName();

       // we just error out - Null user
       if ( _user == NULL )
           throw AuthenticationFailed(" Unknown User ");
    }
}

//
// Now respond to server challenge after connect
//
//
void CIMClient::connect(const char* address)
{
    if (_channel)
        throw AlreadyConnected();

    ChannelHandlerFactory* factory =
              new ClientHandlerFactory(_selector);

    TCPChannelConnector* connector
        = new TCPChannelConnector(factory, _selector);
```

```
    // ATTN-A: need connection timeout here:
    _channel = connector->connect(address);

    if (!_channel)
        throw FailedToConnect();

    //
    // handle client side authentiocation
    // Added the supoort for client side local authentication module
    //
    // The local those wish to connect through channel must be
    // authenticated by CIM Server.

    Authenticator* authHandler  =
                  (Authenticator *)new LocalAuthenticator();

   // Block for authentication
   _channel->enableBlocking();
    Boolean authorized =
                authHandler->proveAuthenticate(_channel, _user);
   _channel->disableBlocking();

    if ( authorized != true )
    {
         // we just close the channel and error out
         // Not a Valid Client

         _getHandler()->handleClose(_channel);
         throw AuthenticationFailed((char *)" Unknown user");
    }
    else
    {
         // LOG this
         //a Valid Client
    }
}
```

SystemUnix.cpp : One way to get process User Id if the Client has not specified, Used in Client API.

```
//
// Here is code to get client process owner ID.
// This code will go in   Pegasus/Common/SystemUnix.cpp
//

/**
 *  TODO - Is this HP-UX Specific ?
 *  gets the user name of the process
 */
char*  System::getCurrentLoginName()
{

   static const size_t PWENT_BUFFER_LEN  = 1024;
   char *currentUser;
   char buffer[PWENT_BUFFER_LEN];
   struct passwd pwd;

   //
   //  get the real user's UID.
   //
   uid_t uid = getuid();


   //
   //  lookup that UID in the password list.
   //
   int pwRetVal = System::getPwd(uid, pwd, buffer, PWENT_BUFFER_LEN);
   if( pwRetVal != 0 )
   {
     // LOG  TODO
     //cout << "user may have been removed just after user logged in" << endl;
      return((char *)NULL);
   }

   // extract user name
   currentUser = new char[strlen(pwd.pw_name)];
   strncpy(currentUser, pwd.pw_name, strlen(pwd.pw_name));
   currentUser[strlen(pwd.pw_name)]=0;

   return((char *)currentUser);
}

/**
 *  TODO - Is this HP-UX Specific ?
```

```
 *    Returns passwd struct for a given UID.
 *      If an error occurs a NULL is returned and errno is recorded.
 *
 *      Parameters:
 *         uid       - A UNIX UID
 *         pwdStruct - A pwd struct.
 *         buffer    - A char buffer used to store data for pwd struct.
 *         buflen    - A the length of the char buffer (Recommend 1024).
 *      Returns:
 *         Zero if lookup succeeds or 1 on error.
 */
Uint32  System::getPwd(
                  uid_t          uid,
                  struct passwd & pwdStruct,
                  char         * buffer,
                  size_t         buflen)
{
   struct passwd *   pwd    = &pwdStruct;
   struct passwd **  result = &pwd;
   int pwRetVal = 0;


#ifdef  _REENTRANT
   //
   // We want to use _r REENTRANT functions for thread-safe
   // This requires -D_REENTRANT flag to makefile
   //

   // SYNTAX:
//  int  getpwnam_r(char *name, struct passwd *pwd, char *buffer,
   //                size_t buflen, struct passwd **result);

   pwRetVal = getpwuid_r( uid, pwd, buffer, buflen, result);
   if ( pwd == NULL )
   {
     // LOG perror("getpwnam failed:");
      pwRetVal = 1;
   }

#else

   //SYNTAX:
   //  struct passwd *getpwuid(uid_t uid);
   //
   //
```

```
    // Log the user  TODO
    // << "getpwnam user is :"<< _user << "strlen:" << strlen(_user) << endl;
    pwd = getpwuid(uid);
    if ( pwd == NULL)
    {
      // LOG perror("getpwuid failed:");
        pwRetVal = 1;
    }
#endif

    return pwRetVal;

}
```