# Authentication Server Examples

David King

Information Technology Center

## 1. Workstation examples

### 1.1. Example of Workstation Login

Here is a fragment from my test program, which shows how code within the workstation can log in to an Authentication Server.

```
#include <sys/time.h>
#include </usr/local/rpc/include/rpc.h>
#include <authuser.h>
/*
 * Global variables.
 */
char my_user[8];        /* The user's username */
int my_job;             /* The job number assigned by login */
char my_key[8];         /* The session encryption key assigned by login */

struct RPC_ReqBuffer *reqbuff;  /* The RPC request buffer */
struct RPC_RespBuffer *respbuff;/* The RPC response buffer */
                                   by RPC_MakeRPC */
struct timeval tout = {60, 0};  /* RPC timeout of one minute */

/*
 * Initialize to use RPC.
 */
do_initialize ()
{
    if (RPC_ClientInit (2, RPC_VERSION) != RPC_SUCCESS)
        abort ();
    reqbuff = (struct RPC_ReqBuffer *) malloc (1000);
    reqbuff->ActualRequest.Header.ProtoVersion = 0;
    reqbuff->ActualRequest.Header.Subsys = 2;
    reqbuff->ActualRequest.Header.NoReturnValue = 0;
}
```

```
/*
 * Log the user in.
 */

do_login ()
{
    int authcid;        /* The RPC connection id, to be filled in */
    char user_pwd[16];  /* The user's password */
    char pwd_key[8];    /* The password turned into an encryption key */
    int self_id[10];    /* An RPC_BoundedBS containing the user's self-
                           identity.  An RPC_BoundedBS must be on an
                           integer boundary. */

    /*
     * Initialize the user's username and password, which are
     * constants for this example.
     */
    bzero (my_user, 8);
    strcpy (my_user, "djk");
    bzero (user_pwd, 16);
    strcpy (user_pwd, "secret");
    /*
     * Turn the user's password into an encryption key according to
     * the crunching algorithm.
     */
    crunch_pwd_DES (user_pwd, pwd_key);
    /*
     * Now we must generate what RPC calls my 'self-identity' string.
     * For the login case, this is simply the user's eight-character
     * username.
     */
    {
        register RPC_BoundedBS *self_id_ptr;
        self_id_ptr = (RPC_BoundedBS *) self_id;
        self_id_ptr->MaxSeqLen = 8;
        self_id_ptr->SeqLen = 8;
        bcopy (my_user, self_id_ptr->SeqBody, 8);
    }
    /*
     * Now we will bind to the server, requesting an authenticated and
     * encrypted connection.  We will provide our self-identification
     * and the encryption key for this connection.  For this example,
     * the cluster machine name is the constant 'admin'.
     */
    if (RPC_Bind (RPC_SECURE, "auth", "admin", 0, &authcid,
                        (RPC_BoundedBS *) self_id,
                        (RPC_EncryptionKey *) pwd_key) != RPC_SUCCESS)
        abort ();
    /*
     * We have a connection, now we must make a Connect request over it.
```

```
        * This will do the actual login, and will return to us our job
        * number and session encryption key.
        */
       {
           /*
            * Build the message in the request buffer.
            */
           register struct ms_connect *msg_ptr;
           msg_ptr = (struct ms_connect *) reqbuff->ActualRequest.Body;
           reqbuff->ActualRequest.Header.Opcode = AAS_CONNECT;
           reqbuff->ActualRequest.Header.BodyLength = sizeof *msg_ptr;
           bcopy ("r602", msg_ptr->account, 4);
           bcopy (user_pwd, msg_ptr->password, 16);
       }
       /*
        * Make the request.
        */
       if (RPC_MakeRPC (authcid, reqbuff, &respbuff, &tout) != RPC_SUCCESS)
           abort ();
       if (respbuff->ActualResponse.Header.ReturnCode != AAR_SUCCESS)
           abort ();
       /*
        * Get the information out of the reply message.
        */
       {
           register struct ms_connect_r *rep_msg_ptr;
           rep_msg_ptr = (struct ms_connect_r *) respbuff->ActualResponse.Body;
           my_job = ntohl (rep_msg_ptr->job);
           bcopy (rep_msg_ptr->key, my_key, 8);
       }
       /*
        * Done with the connection.
        */
       RPC_Unbind (authcid);
   }
```

## 1.2. Example of Workstation Logout

Here is the logout routine from my test program.

```
do_logout ()
{
    int authcid;        /* The RPC connection id */
    int self_id[10];    /* An RPC_BoundedBS for self-identity */

    /*
     * Generate the self-identification string, for the case where
```

```
         * the user is already logged in.  In this case, the self-identity
         * contains the username and the job number.
         */
        {
            register RPC_BoundedBS *self_id_ptr;
            register struct ms_service_si *self_id_blk;

            self_id_ptr = (RPC_BoundedBS *) self_id;
            self_id_ptr->MaxSeqLen = sizeof *self_id_blk;
            self_id_ptr->SeqLen = sizeof *self_id_blk;
            self_id_blk = (struct ms_service_si *) self_id_ptr->SeqBody;
            bcopy (my_user, self_id_blk->username, 8);
            self_id_blk->job - htonl (my_job);
        }
        /*
         * Bind to the server, requesting a secure connection but this time
         * the encryption key is the previously assigned session key, and the
         * self-identity string is of the type containing the job number.
         */
        if (RPC_Bind (RPC_SECURE, "auth", "admin", 0, &authcid,
                            (RPC_BoundedBS *) self_id,
                            (RPC_EncryptionKey *) my_key) != RPC_SUCCESS)
            abort ();
        /*
         * Build the Disconnect request in the request buffer.
         */
        reqbuff->ActualRequest.Header.BodyLength = 0;
        reqbuff->ActualRequest.Header.Opcode = AAS_DISCONNECT;
        /*
         * Make the request.
         */
        if (RPC_MakeRPC (authcid, reqbuff, &respbuff, &tout) != RPC_SUCCESS)
            abort ();
        if (respbuff->ActualResponse.Header.ReturnCode != AAR_SUCCESS)
            abort ();
        /*
         * It has been done, close the connection.
         */
        RPC_Unbind (authcid);
    }
```

## 1.3. Example of Workstation Password Change

Here is the password-change routine from my test program.

```
do_pwdchg ()
{
    int authcid;          /* The RPC connection id */
    int self_id[10];      /* An RPC_BoundedBS for self-identity */
    char old_pwd[16];     /* The old password */
    char new_pwd[16];     /* The new password */

    /*
     * Make the old and new passwords, from constants for the example.
     */
    bzero (old_pwd, 16);
    strcpy (old_pwd, "secret");
    bzero (new_pwd, 16);
    strcpy (new_pwd, "moresecret");
    /*
     * Generate the self-identification string, for the case where
     * the user is already logged in.  In this case, the self-identity
     * contains the username and the job number.
     */
    {
        register RPC_BoundedBS *self_id_ptr;
        register struct ms_service_si *self_id_blk;

        self_id_ptr = (RPC_BoundedBS *) self_id;
        self_id_ptr->MaxSeqLen = sizeof *self_id_blk;
        self_id_ptr->SeqLen = sizeof *self_id_blk;
        self_id_blk = (struct ms_service_si *) self_id_ptr->SeqBody;
        bcopy (my_user, self_id_blk->username, 8);
        self_id_blk->job - htonl (my_job);
    }
    /*
     * Bind to the server, requesting a secure connection but this time
     * the encryption key is the previously assigned session key, and the
     * self-identity string is of the type containing the job number.
     */
    if (RPC_Bind (RPC_SECURE, "auth", "admin", 0, &authcid,
                        (RPC_BoundedBS *) self_id,
                        (RPC_EncryptionKey *) my_key) != RPC_SUCCESS)
        abort ();
    /*
     * Build the Change Authorization request in the request buffer.
     */
    {
        register struct ms_chgauth *msg_ptr;

        msg_ptr = (struct ms_chgauth *) reqbuff->ActualRequest.Body;
        reqbuff->ActualRequest.Header.BodyLength = sizeof *msg_ptr;
        reqbuff->ActualRequest.Header.Opcode = AAS_DISCONNEC;
        bcopy (my_user, msg_ptr->username, 8);
        bcopy (old_pwd, msg_ptr->oldpwd, 16);
```

```
        msg_ptr->item = htonl (AAO_CHGAPWD);
        bcopy (new_pwd, msg_ptr->newpwd, 16);
    }
    /*
     * Make the request.
     */
    if (RPC_MakeRPC (authcid, reqbuff, &respbuff, &tout) != RPC_SUCCESS)
        abort ();
    if (respbuff->ActualResponse.Header.ReturnCode != AAR_SUCCESS)
        abort ();
    /*
     * It has been done, close the connection.
     */
    RPC_Unbind (authcid);
}
```

# 2. Cluster Server Examples

## 2.1. Initial Connection Example

When a user workstation makes initial connection to a server, it will request either a completely secure connection or an authenticated connection. In either case, the workstation will provide a self-identification string to RPC, and an encryption key. The self-identification string should include, at the least, the user's username and Authentication job number, and the encryption key should be the session key assigned at login.

The server's connection accepting routine should contain an RPC_Accept coded something like this:

```
do_accept ()
{
    int whoami;          /* Filled in with parent/child indicator */
    int bulkproto;       /* Filled in with bulk transfer protocol id */
    int howsecure;       /* Filled in with the user's security level */
    RPC_BoundedBS *clientident; /* Filled in with pointer to self-ident */
    int do_getkeys ();  /* The routine to get the encryption key */

    /*
     * Wait for a connection and accept it.  This will, "behind the scenes",
     * call do_getkeys to figure out the encryption key the user presumably
     * is using.
     */
    RPC_Accept (&whoami, &bulkproto, do_getkeys, &howsecure, &clientident);
    /*
     * The connection has been made, the encryption key set up, and the
     * handshake messages have been exchanged.  We now are certain the
     * the user is who he says he is.
     */
}
```

The "do_getkeys" routine must be provided to the RPC call, so that the RPC code can find out what a given user's encryption key is supposed to be. In the environment we have, where the file server (for instance) is being connected to, the file server must get that information from the Authentication Server. Here is a Get Key routine which may do the job.

```
/*
 * Get encryption key for RPC authentication handshake.  RPC_Accept calls
 * here when a connection arrives and the client requested a secure
 * connection.  The self-identification string, which the client must
 * provide must be (in this example) 12 bytes long, containing the user's
 * 8-byte username and his job number.  We will here ask the Authentication
 * Server to tell us the session key it assigned to him, so we can return
 * the key to RPC.
 */
```

```
struct self_id_blk {
    char user[8];
    int job;
};

int do_getkeys (client_ident, ident_key, session_key)
                                /* Returns 0 if successful, 1 if not */
RPC_BoundedBS *client_ident;    /* The user's self-identification */
RPC_EncryptionKey *ident_key;   /* The encryption key for this authentication
                                   handshake */
RPC_EncryptionKey *session_key; /* The encryption key for data transmission
                                   following the handshake */
{
    char username[8];           /* The user's username */
    int jobno;                  /* The user's job number */
    int authcid;                /* RPC connection id to Authentication
                                   Server */
    /*
     * Reject if the length is incorrect.
     */
    if (client_ident->SeqLen != 12)
        return (1);
    /*
     * Get the data out of the string.
     */
    {
        register struct self_id_blk *self_id_ptr;
        self_id_ptr = (struct self_id_blk *) client_ident->SeqBody;
        bcopy (self_id_ptr->user, username, 8);
        jobno = ntohl (self_id_ptr->job);
    }
    /*
     * Make a connection to the Authentication Server.
     * For this example, we assume that we are running on host "admin".
     */
    if (RPC_Bind (RPC_OPENKIMONO, "auth", "admin", 0, &authcid, 0,
                        0) != RPC_SUCCESS)
        abort ();
    /*
     * Build the Get User request.
     */
    {
        register struct ms_getuser *msg_ptr;
        msg_ptr = (struct ms_getuser *) reqbuff->ActualRequest.Body;
        reqbuff->ActualRequest.Header.BodyLength = sizeof *msg_ptr;
        reqbuff->ActualRequest.Header.Opcode = AAS_GETUSER;
        bcopy (username, msg_ptr->username, 8);
        msg_ptr->job = jobno;
    }
    /*
```

```
     * Make the Get User request.
     */
    if (RPC_MakeRPC (authcid, reqbuff, &respbuff, &tout) != RPC_SUCCESS)
        abort ();
    /*
     * Done with the RPC connection.
     */
    RPC_Unbind (authcid);
    /*
     * Return the data to RPC if the user was found.
     */
    if (respbuff->ActualResponse.Header.Return == AAR_SUCCESS) {
        register struct ms_getuser_r *rep_msg_ptr;
        rep_msg_ptr = (struct ms_getuser_r *) respbuff->ActualResponse.Body;
        bcopy (rep_msg_ptr->key, ident_key, 8);
        bcopy (rep_msg_ptr->key, session_key, 8);
        return (0);
    }
    else
        return (1);
}
```