A SIMPLE PRODUCTION PASSWORD SERVER

Greg White, 19-Apr-2010, SLAC
v1.4.2 24-Aug-2010, Updated to reflect that getPwd has been released on production.

This is a proposal for a simple secure password server that may be used within the LCLS control system.

The core method employed is to leverage the fact that we already use RSA/DSA authentication, passwordless login, and encrypted (SSL) socket communication, among our control system application level hosts; and that ssh can execute a remote command passwordlessly using these protocols in combination. The result is a bit like using Apple Keychain, except for an authenticated user, the keychain password need not be given. Authenticated users may simply ask for the password of a given item. Example:

```
$ getPwd MACHINE_MODEL
1passW0rd
```

The demo I include here is ready to go. All we'd have to do to make it "production" is put the getPwd and getConnection scripts in CVS.

If you want to play with it, the are in the directories listed in APPENDIX A.

1. FEATURES

1) Allows any program to securely acquire the password for any oracle account at *runtime* - no local files with passwords lying around
2) Wire and disk resources are all encrypted, no cleartext anywhere
3) Works equally well for clients on the production network, DMZ or AFS
4) Completely integrated with the existing authentication and authorization setup for LCLS controls, using the the "physics", "softegr" etc accounts
5) An interactive user can get a password when they want it
6) Optionally provides connection meta data, as well as password data
7) Not just for Oracle - works just as well without modification for getting login credentials or secure data for any kind of system
8) Works for interactive, scripts (bash, perl etc) or Java
9) So simple and straightforward it's embarrassing.

Treat this as a proposal for a stop-gap mechanism, until the SCS Kerberos system is available.

2. BASIC IDEAS

Passwords are encrypted once and decrypted at runtime using the private DSA key of a single account, to which all potential password users are authenticated (symmetric encryption). We already have such an account - physics@lcls-srv01. We use the DSA private key because it is already well protected, in that only "physics" authorized users can read it. The DSA private key in this way does double duty - being both the DSA authentication key, and the encryption/decryption key for the passwords file. Hence it does both authentication and authorization. Set the file permissions of the password file and the server side executable, to be read-execute by physics authenticated users only.

Passwords are requested and returned by executing a command over ssh, which uses DSA for authentication, and Secure Socket Layer for wire communications. So only authenticated users of physics can execute the password request, and returned data is encrypted over the wire.

Both the server and the client sides are essentially 1 line programs. The clients side is an ssh command "getPwd for <key>" executed in physics@lcls-srv01; and

the server side is "decrypt the password file, then search for <key> and if you find a match, return it."

Since "physics" is a DSA authenticated account with passphraseless authentication, and both private network and AFS users may be DSA authenticated to physics, then any user at SLAC, whether on the private network, DMZ or SLAC Public, can get a password using the same client side interface so long as they're a user of physics. Client code runs without modification anywhere at SLAC.

3. SETUP

This section describes what it takes to create the password server on the server side and client side.

3.1 Create the "Secure Channel"

Create a passwordless login to some account on some machine in the CA network, with similar or DSA authentication properties as the "physics" account. That is, all program running accounts and individual users of the Control System are DSA authenticated to it. This is done by generating an DSA public key and inserting it into the authorized_keys file of the account from which you want access Oracle (or whatever else). Here's a good reference [1] for SLAC.

We could simply use "physics", since the other production accounts such as "softegr", and individual users from lcls-prod02, and authenticated to it. So, this step is probably a no-op for us!

3.2 Create the cleartext password database - just a file

Create a password file in cleartext. I created a 3 password example in a file I called .cdpwds_clear (passwords for the CD department). Everything after the 1st 2 fields of a record are optional, and can be used for metadata. Here I put in the oracle connection URLs for the AIDA schema for SLACPROD and MCCO. See files in physics@lcls-srv01:/home/physics/greg/priv/.

```
bash-2.05$ cat .cdpwds_clear
aidadev sd424se1 jdbc:oracle:thin:@slac-oracle03.slac.stanford.edu:1521:SLACPROD
aidaprod krjr2 jdbc:oracle:thin:@slac-oracle03.slac.stanford.edu:1521:MCCO
machine_model adsfasdfasdf
```

3.3 Encrypt the password database

To avoid cleartext passwords on the disk, we encrypt the password database file (DES3 cypher). The encryption key used is the private DSA key. In this way, all and only accounts authenticated to physics, and that are therefore authorized to the DSA private key file, may retrieve a decrypted password:

```
openssl des3 -salt -in .cdpwds_clear -out .cdpwds_enc -pass file:$HOME/.ssh/id_dsa
```

The line above outputs a file, ".cdpwds_enc" which is the DES3 encrypted version of .cdpwds_clear.

This is wrapped by the script ecrypt_pwds.

```
chmod u+w .cdpwds_enc
./encrypt_pwds
chmod u-w .cdpwds_enc
```

At this point, the cleartext .cdpwds_clear should be deleted. The master should probably be put in a key safe managed by the systems team.

3.4 Server

Still in the remote account, create a tiny script that decrypts the password file, and searches it for the name of a given resource (like an oracle account name) for which it returns the corresponding password. Eg

```
bash-2.05$ cat gimmePwd
#!/bin/sh
name=`(echo $1|tr '[[:upper:]]' '[[:lower:]]')`
pwd=`openssl des3 -d -in ${HOME}/greg/priv/.cdpwds_enc \
   -pass  file:${HOME}/.ssh/id_dsa | awk 'tolower($1)~/^'$name'$/ \
   {print $2}'`

if [ -n "$pwd" ]
then
    echo $pwd
else
    echo $name 'is an invalid key for the password list.' \
        ' Check spelling and case.'
fi
```

So, now, if someone sshs to physics@lcls-srv01 and remotely executes gimmePwd giving an argument, it will return the second field of the record whose first field matches the argument. Ie, it returns the password for a given lookup key.

3.5 Client Side

On the local account (the one which wants to access oracle, such as the account in which a Web server is running, or just an account from which you want to run sqlplus), remotely execute gimmePwd over ssh (for wrapping script, see below). Eg:

```
$ ssh -lphysics lcls-srv01 /home/physics/greg/priv/gimmePwd <resource>
pa33w0rd!
```

where <resource> is something like "machine-model" or "aida". More precisely, this must be executed from an account which is DSA authenticated to physics. These include softegr and iocegr on the CA network production, and simply user accounts on development networks like LCLS-DEV, DMZ and SLAC Public.

3.5.1 One client wrapper for all networks: CA, DMZ, LCLS-DEV, SLAC Public

CA and AFS Clients running on DMZ can execute the same ssh command as above, since they can directly access lcls-srv01. However, AFS Clients on the SLAC public or LCLS DEV networks (eg developers and physicists offices) have to go via DMZ. This is how aidalist running in a matlab session in someone's office would work. A single script can handle all client sources, regardless of network. If you're authenticated to physics, and hence authorized to the DSA private key, you can get a password.

[getPwd on Production is in /usr/local/lcls/tools/script/getPwd]

```
#!/bin/bash -f
if [ -d /usr/local/lcls ]; then
    ssh -lphysics lcls-srv01 /home/physics/.ssh/.private/gimmePwd ${1}
2>/dev/null
else
    ssh lcls-prod02.slac.stanford.edu ssh -lphysics lcls-srv01
'/home/physics/.ssh/.private/gimmePwd' ${1}
fi
```

So production accounts like "softegr","iocman" or any developers account on SLAC Public or LCLS-DEV, can acquire a password identically:

```
$ getPwd aidaprod
krjr2
```

4. CLIENT SIDE USEAGE

This section describes how a user would use the password server from different high level software clients. Use cases of sqlplus, shell scripts, and inside a java program are described.

4.1 Using the password server for interactive SQLPlus etc

This demo shows using getPwd to get the password to log into SQLPlus for the AIDA schema:

```
[softegr@lcls-builder db]$ sqlplus aida/`getPwd aida`
...
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.4.0 - 64bit Production
With the Partitioning and Real Application Testing options

SQL>
```

4.2 Using the password server from inside a script

Here's a csh script that logs into Oracle using SQLPlus (the aidaprod schema) and executes an arbitrary SQLPlus query. The 3rd line gets the password for the "aidaprod" schema and assigns a shell variable to the schemaname/password combination. I use this script in my aida db management scripts, to centralize all database interaction:

```
[softegr@lcls-builder db]$ cat sql_runner
#!/bin/csh
setenv TWO_TASK SLACPROD
set schema = aidaprod
set DB = ${schema}/`getPwd ${schema}`
set cmd = "@$argv:q"
sqlplus -S $DB <<EOF
@$cmd
quit;
EOF
```

4.3 Using the password server from java

Using this system in java, we shall get the password when in the java executable, as opposed to passing as a Java Property. This is so to avoid the password showing up in /proc, and so also accessible to root. Example: DemoPwd.java is a java program that executes the shell script getPwd to securely acquire the password for the item given in its 1st argument.

[Example in /home/softegr/greg/tools/db/pwd]

```
$ cat DemoPwd.java
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;

class DemoPwd
{
    /* Usage: DemoPwd pwditem. Eg: java DemoPwd machine_model */
    public static void main( String[] args )
    {
        String[] cmd = {"getPwd", args[0]};
        try
        {
            Process process = Runtime.getRuntime().exec(cmd);
            InputStream in = process.getInputStream();
            BufferedReader console =
                new BufferedReader(new InputStreamReader(in));
            String pwd = console.readLine();
            System.out.println("The password for "+
                 cmd[1]+" is "+pwd);
```

```
                }
                catch (Exception ex)
                {
                    System.err.println("Error getting password for"+
                        cmd[1]);
                }
            }
        }
```

Demo of usage within Java:

```
$ java DemoPwd machine_model
The password for machine_model is 1passW0rd
```

4.4 Performance

The user time to get a password is about 1/3 - 1/4 of a second on production. So, clients should get the password once, keep it in memory, and reuse it.

APPENDIX A: Demo Directories

Production:
physics@lcls-srv01:/home/physics/greg/priv/ - Server for prod/CA accounts
softegr@lcls-builser:/home/softegr/greg/tools/db/pwd - Client side for prod/CA includes script and java.

EG [softegr@lcls-builder pwd]$ getPwd AIDAPROD
dsafsa

AFS:
/u/cd/greg/lcls/pwds - Client for AFS accounts

REFERENCES

[1] http://www-glast.slac.stanford.edu/software/CodeHowTo/generate_the_ssh.htm