A Simple Way to Make Passwords More Effective

J. Scott Magruder

The University of Southern Mississippi
601-266-5099/Fax 601-266-4642
James.Magruder@usm.edu

Stanley X. Lewis, Jr.

The University of Southern Mississippi
601-266-6924/Fax 601-266-4642
s.lewis@usm.edu

Eddy J. Burks

Athens State University
205-233-8216/Fax 205-233-8164
burksej@athens.edu

ABSTRACT

A simple change to the login process is shown to make passwords more effective. An illustration is presented where users login through a Web browser, but the logic also applies to any situation where a login/password combination are required. By adding to the existing password security features typically found in a network additional layers of encryption a company's network can be made more secure. For the illustration, the code is written in perl.

THE NORMAL LOGIN PROCESS

The normal login process is illustrated using a Web page login/password requirement. The user points their browser to an HTML page that has a textbox for a username, a textbox for their password and a submit button. Once filled in, the user clicks on the submit button which calls the cgi (Common Gateway Interface) program (written in perl in this example) which checks to see if the username and password are in the password file. If the user's login is correct, then they are allowed access to the Web page, directory, etc. If not, then the user may be allowed to try again. The failed access may be logged.

The perl program reads what the user has typed in the two textboxes. In this case, the input would be of the form

UserName=Name_Typed_In&password=Password_Typed_In

where UserName and password are the names given to the html text boxes. The & separates the two pieces of information. Thus, splitting the line on & gives the two pieces of information in a key=Value structure. Then splitting on the = in both parts gives us the values that the user typed into the textboxes. Next, the password sent via the textbox is encrypted with the system encryption routine. The username (and the associated encrypted password) is found in the password file and the encrypted password from the file is compared with the user's encrypted password. If they are the same, then access is granted. Otherwise, the user will receive a Web page indicating that access is denied. The user may be allowed to try again.
The html code for the login page would look like this:

```
<html>
<body>
<form method="post" action="/cgi-bin1/processPassword.pl">
Username<input type="text" name="UserName"><BR>
Password<input type="password" name="password"><BR><P>
<input type="submit">
</form>
</body>
</html>
```

The user types in their login name (Username) and their password and clicks on Submit. The form statement calls the program (processPassword.pl) to check the Username/password pair.

NORMAL PASSWORD TESTING

In a company's network a password checking routine is implemented as typically as a module in the network operating system. As such the checking routine does not cause a delay in the acceptance or rejection of a person attempting to access the network. The authors have generated a set of three modules (programs) to illustration the typical approach found in a networked environment to provide security over access to the system.

Program 1 - Password Testing

The perl code testing the password would look like this:

```
#!/usr/bin/perl                          # This is where the perl interpreter resides
$in=<STDIN>;                             # Read the information passed from the html page
@split1=split(/&/,$in);                  # Split the information into the two key=value
                                         pieces
@split2=split(/=/,$split1[0]);           #Split the first piece; the username is in $split2[1]
@split3=split(/=/,$split1[1]);           #Split the second piece; the password is in split3[1]
$flag=0;
open(P,"/diskb1/home/demo2/pass");       #Open the password file
print "Content-type: text/html\n\n";      #This is the required first line in the html protocol
                                         when a #page is sent back to the user
print "<html><body>";                    #Start the web page
while ($line=<P>)                        #Read a record from the password file
{
```

```
($name,$encpass)=split(/\|/,$line);          #Split the password record into the username and encrypted password
chomp($encpass);                             #Get rid of the newline (\n) which is the password record delimiter
if ($split2[1] eq $name)                     #split2[1] is the username from the html document; compare it to the #username from the password file [1]
{
$salt = substr($encpass,0,2);                #A match has been found; extract the salt which is the first two #characters of the encrypted password
$checkPass=crypt($split3[1],$salt);          #split3[1] is the password typed in the html document and #passed to this program; Encrypt the password
if ($encpass eq $checkPass)                  #Compare the encrypted password from the html document with the #encrypted password from the password file
{
opendir(DIR,"/diskb1/home/mis408/public_html");
                                             #A match has been found; open the #directory we are protecting and send a web #page back to the user
@dir=readdir(DIR);                           #Read the directory entries into an array
foreach (@dir)                               #Go through each directory entry
{
if (($_ ne ".") && ($_ ne ".."))             #exclude . and ..
{
print "<a href=\"http://131.95.59.37:8080/~mis408/$_\">$_</a>";
                                             #print the directory entry in a #format so the user can click on the #entry and the contents will be #shown to the user
print "<BR>";                                #print a break (html newline) between each entry
}
}
}
Else                                         #A match was not found between the (encrypted) password the user #entered and the encrypted password for that user in the password file
{
print "Login incorrect";                     #print an error message
}
}
}
print "</body></html>";                      #print the ending components of the web page
```

Program 2 - Acceptance & Encryption

Accept the name and password; encrypt the password and compare it with the encrypted password associated with the name in the password file. To create the password file, the following perl program could be used:

```
#!/usr/bin/perl
use Term::ReadKey;                          #We use this perl module so the password will not be echoed to the #screen
print "Enter login name: ";                 #prompt the user
$Name=<>;                                   #Read the user name
chomp($Name);                               #Get rid of the newline attached to what the user types in when they hit #enter
ReadMode 2;                                 #Go into the non-echo mode
print "Enter password: ";                   #prompt the user for the password
$pass=<>;                                   #Read the password
chomp($pass);                               #Get rid of the newline
ReadMode 0;                                 #Go back to regular echo mode
$salt='';                                   #predefine $salt to be empty
$s1=('a'..'z','A'..'Z')[52*rand];           #randomly generate the first salt character
$s2=('a'..'z','A'..'Z')[52*rand];           #randomly generate the second salt character
                                            #Reference (Holzner, 2001)
$salt=$s1.$s2;                              #Combine the two randomly generated characters into one salt
print "\n$salt\n";
$encrypted=crypt($pass,$salt);              #Use the generated salt to encrypt the password
open (P,">>pass");                          #Open the password file (pass) in append mode
print P "$Name|$encrypted\n";               #Print the user name and encrypted password (the salt is the first two #character of the encrypted password) to the password file using |#as the delimiter
```

Program 3 - Encrypted Passwords & Password File Generation

Generate encrypted passwords and put the name and encrypted password in the password file. The perl code is not presented for this program.

HACKING PASSWORDS

Common knowledge of network systems and a public media blitz about the hacking of network systems presents network administrators an increasng challenge to restrict their company's network system (internal network, Internet, intranet and extranet) environment to users authorized by the company.

A hacker would try to break into this system by cracking the passwords in the password file. The hacker would have to have a copy or access to the password file, a dictionary file and a program that would take the list of words in the dictionary file, encrypt them using the same encryption scheme as the system he is trying to break into. If one of the encrypted words is found in the password file, then that password is broken.

PROGRAM 4 – Password Hacked

The following perl program illustrates the process of password cracking:

```
#!/usr/bin/perl
open (PASS,"passwd");                          #open the password file
while ($precord=<PASS>)                        #continue while we have passwords to check
{
($name,$encpass)=split(/\|/,$precord);         #The password record consists of a username and
                                               an #encrypted password separated by the | delimiter
$salt = substr($encpass,0,2);                  #extract the salt --the first two characters of the
                                               encrypted password #from the password file
open (DICT,"dictionary");                      #open the dictionary file
while ($word=<DICT>)                           #read the next word in the dictionary file
{
chomp($word);                                  #get rid of the newline that is attached to the word
$TestPass=crypt($word,$salt);                  #encrypt the word from the dictionary file using the
                                               salt
if ($encpass eq $TestPass)                     #check to see if the encrypted password from the
                                               password file matches the encrypted word from the
                                               dictionary file
{
print "The password is $word for user $name \n";
                                               #the two encrypted words match; thus we
                                               have broken that password
}
else
{                                              # no match found; try another dictionary word until
                                               all of these words have been encrypted and tested
                                               for a match
 }
 close(DICT);
 }
 }                                             #go to the next password, again encrypting all
                                               dictionary words and #comparing the result with the
                                               encrypted password
```

Using the above perl program on the password file, passwd:

John|KGHfKJy1j6AjQ
tom|Q]ieHZZ3qLSWw

lester|_Gb0IxQ.QUSpI

produces the following output:

The password is klop for user lester

A very small dictionary file was used in this demonstration. The dictionary contained the password so that we would be able to demonstrate that a password would be broken.

MAKING THE PASSWORD SCHEME MORE SECURE

Now that the stage has been set, more secure procedures can be presented. Hackers must have access to the password file, a dictionary file, a password cracking program and the password encryption process used by the organization. The more secure password procedures presented here change slightly the organization's password encryption process without letting the hackers know the process has changed.

Encrypt the password more than one time. The first change is to encrypt the password more than once. This requires changes to the code in Program 2 and Program 3:

Program 2 changes:

The line -

```
$checkPass=crypt($split3[1],$salt);        #split3[1] is the password typed in the html
                                           document and passed to this program; Encrypt the
                                           password becomes two lines as shown next.
$checkPass=crypt($split3[1],$salt);        #split3[1] is the password typed in the html
                                           document and  passed to this program; Encrypt the
                                           password
$checkPass=crypt($checkPass,$salt);
```

In this case, the same salt was used when the password was encrypted again.

Program 3 changes:

The line -

```
$encrypted=crypt($pass,$salt);        #Use the generated salt to encrypt the password becomes
                                      two lines as shwon next.
$encrypted=crypt($pass,$salt);        #Use the generated salt to encrypt the password
$encrypted=crypt($encrypted,$salt);
```

Now the same algorithm (encrypting the password twice) is being used by the (encrypted) password generator and the login name/password checker. Thus, the users will be correctly validated and the login process would continue. However, the hacker's cracking algorithm only encrypts one time and thus will not break the password.

Changing the salt

The second salt can also be changed in the above process. The salt could be reversed.

Program 2 code changes:

The line -

```
$checkPass=crypt($split3[1],$salt);          #split3[1] is the password typed in the html document and passed to this program; Encrypt the password is replaced by the next three lines
$checkPass=crypt($split3[1],$salt);          #split3[1] is the password typed in the html document and passed to this program; Encrypt the password by the next lines
$ReverseSalt=reverse($salt);
$checkPass=crypt($checkPass,$ReverseSalt);
```

Program 3 changes:

The line:

```
$encrypted=crypt($pass,$salt);               #Use the generated salt to encrypt the password becomes two lines as shown next
$encrypted=crypt($pass,$salt);               #Use the generated salt to encrypt the password
$ReverseSalt=reverse($salt);
$encrypted=crypt($encrypted,$ReverseSalt);
```

The salt could be the last two characters of the first encrypted result. There are many possible variations on this theme.

Again, this process is not an attempt to make the encryption algorithm more secure. Rather, the overall process is changed so the hacker is wasting their time using the "old process". This makes the overall process more secure to a dictionary and brute force attack .

This process can also be extended to multiple encryptions by putting the code in a loop:

```
$I=5;                                        #This is the number of times to perform the encryption
for ($I=1;$I<=$j;$I++)
{
$encrypted=crypt($pass,$salt);
 }
```

The only problem is "remembering" the number of times to encrypt and allowing the salt to be changed. The first problem could be solved by saving the number of times to encrypt the password in the password file, i.e., a record in the password file would look like:

LoginName|encryptedPassword|Number of times to loop

However, if the hacker does have access to the password file, then the additional field will give them too much information. Rather, the numerical value of the last character of the login name (as an example) can be used to determine the number of loops. For example, the ASCII value of A is 65. We can subtract 64 from that and determine the loop should be done one time. Thus, each login name may have a different number of encryptions associated with the password. This makes the hackers attacks even less profitable.

The Test

The password cracker John-The-Ripper (available on the Internet) was used to test the original change in the login process. Jack was chosen since it is very easy to add words to the word-list. The regular text for a password was added to Jack's list and then encrypted twice and put in the password file. The user was able to login properly, but after several hours of running time, Jack did not crack the password.

CONCLUSION

This paper has demonstrated how making minor changes in the login process may make sites secured by a login process more secure. These minor changes include encrypting the password z multiple times, with the same or changed salt.

REFERENCES

Holzner, Steven. PERL BLACK BOOK, 2nd Edition, Coriolis, 2001, 486-487.