Password expiry

Reviewing password attributes

Skill Level: Introductory

David Tansley
System Administrator
Ace Europe

31 Aug 2010

Without a doubt, password policy is an audit requirement. If audited, as a system administrator you will need to prove your policy with ad-hoc reports and listings.

Introduction

Password policy is the mechanism of ensuring and enforcing that an account is protected by a password authentication process. System administrators, who apply the rules of password policy, should ensure that passwords used are not easily determined by outsiders. The password policy should be in place on all active user accounts.

One part of a password policy is to interrogate password expiry and restriction for review. It enables the system administrator to see if there are any inconsistencies in their own password policy for users and thus can be amended. Generally, these are looked at in an ad-hoc way but is best done through reporting on all users. Typically, this would involve considerations such as when a password was last changed, when it is due for expiry, and any password flags set for each user. Password policy, in general, is an audit reporting requirement.

This article is not a guide on AIX hardening or how to implement a security policy, but rather attributes relating to password policies that should be considered.

Password policy overview

A password policy is put in place to ensure the systems that users access are secure. In essence, this means the password is not an easily guessed one and should be changed frequently. Certain rules will be enforced upon the user when they try to create their password; these rules should be global to all users. Cases of exception are application owners. Let us look at a basic password policy regarding the makeup of a password. The user account attributes, such as system defaults or user specific, can be found in the file: `/etc/security/user`.

I would suggest a sound password policy should have at least the following:

The user gets three login attempts at entering the correct password, if this value is reached, the account is locked.

```
loginretries 3
```

How long (in weeks) before a user can change their password since the last password change?

```
minage 1
```

How long in weeks before the user is forced to change their password?

```
maxage  5
```

How many days (and each day before expiration) before the password is due to expire should the user be informed?

```
pwdwarntime 5
```

How many weeks after `maxage` has passed can the password be changed by the user? In this case, no weeks (denoted by -1) means the user must change their password when `maxage` is reached.

```
maxexpired -1
```

The minimum alpha characters to be used in a password.

```
minalpha 1
```

The minimum non-alpha characters to be used in a password.

```
minother 1
```

The minimum number of characters in a password.

```
minlen  8
```

The minimum number of characters that were present in the previous password that cannot be used in the new password.

```
mindiff 2
```

The number of characters that cannot be repeated in the new password.

```
maxrepeats 2
```

The number of weeks that a previous password can be reused.

```
histexpire 26
```

Application owner accounts should be the only exception to the password policy rule that is being used. Notably, the password could be set to never expire, and the amount of password retries could be increased before the account is locked. The reason for this is simple: If an application or interface gets locked in production, then that affects your business. These types of account attributes should be managed manually. Typically attributes to change could be:

```
maxage=0
loginretries=10
```

A schedule change should then be put in place to change the password when the applications have closed.

Implementing a password policy can be time consuming when presenting the information in a clearer format for reporting to security managers, especially when dealing with a large enterprise network. The user_defaults script contained in Listing 1 below presents this information in a more readable format. First, it extracts the system defaults, then lists each account contained in `/etc/security/user` that has attributes which overrides the system defaults. If no system default has a value, then the attribute is flagged with messages '**No Defaults Set'. In this format, it is much easier to see what system defaults one has implemented. The script could be run on each remote host, and the system defaults can be compared to see if there are discrepancies between hosts. Looking at the users overrides of system defaults, one can then determine if the attributes for the users set are valid according to the implemented security policy.

Listing 1. user_defaults

```
#!/bin/sh
```

```
# user_defaults

get_defaults()
{
# get listing of defaults entry in /etc/security/user and present

grep -p -w "default:"
 /etc/security/user| grep -v "*"|grep -v "default:"|grep -v "\"" >/tmp/defaults.tmp
# the sed '/ //g , is a <CTRL-V> then hit the tab key
sed '/^$/d;s/ //g' /tmp/defaults.tmp >/tmp/hold.tmp && \
mv /tmp/hold.tmp /tmp/defaults.tmp

IFS="="
echo "Default Attributes of /etc/security/user"
echo "Key Value
=========================="
cat /tmp/dt | while read key value
 do
 if [ "$value" = "" ] || [ "$value" = " " ]
 then
 value=" **No Default Set"
 fi
printf "%-15s %-15s\n" "$key" "$value"
done
}

get_users()
# get list of attributes that override the defaults and present
{
echo "\nUser Defined Attributes
----------------------------"
> users.tmp
list=""
# the sed '/ /g , is a <CTRL-V> then hit the tab key
list=$(grep ":" /etc/security/user | grep -v default| grep -v "*" |sed 's/\:/ /g')
for users in $list
 do
 echo "[$users]"
 grep -p -w "$users:"
 /etc/security/user|grep -v "*"|sed '/^$/d;s/ / /g'|sed '1d'>users.tmp
 cat users.tmp | while read line
 do

 key=$( echo $line | awk -F= '{print $1}')
 value=$(echo $line |awk -F= '{print $2}')

 printf "%-15s %-15s\n" "$key" "$value"
 done
echo "-----------------------"
done
}

savedIFS="$IFS"
get_defaults
IFS="$savedIFS"
get_users
```

Typical output from the user_defaults script could be the following:

Listing 2. Typical output from user_defaults script

```
Default Attributes of /etc/security/user
Key              Value
==========================
admin            false
```

```
login            true
su               true
daemon           true
rlogin           true
sugroups         NONE
admgroups        **No Default Set
ttys             ALL
auth1            SYSTEM
auth2            NONE
tpath            nosak
umask            022
expires          0
logintimes       **No Default Set
pwdwarntime      0
account_locked   false
loginretries     3
histexpire       26
histsize         15
minage           1
maxage           5
maxexpired       -1
minalpha         1
minother         1
minlen           0
mindiff          0
maxrepeats       8
dictionlist      **No Default Set
pwdchecks        **No Default Set

User Defined Attributes
------------------------------
[root]
admin            true
SYSTEM           "compat"
registry         files
loginretries     0
account_locked   false
sugroups         admin,sysadmin
-----------------------
[daemon]
admin            true
expires          0101000070
-----------------------
[ukflag]
admin            false
sugroups         !fire,!cloud,earth
-----------------------
[testme]
admin            false
-----------------------
[john]
admin            false
-----------------------
[peter]
admin            false
-----------------------
[jane]
admin            false
-----------------------
[plutt]
admin            false
maxage           1
-----------------------
[spoll]
admin            false
maxage           4
-----------------------
```

Those pwdadm flags

The pwdadm flags allow the system administrator to control and view the password characteristics of a user. Though you can do other tasks with this command, this is its primary use. The basic format is:

```
pwdadm -f <FLAGS> <user>
```

Where the flags are:

- NOCHECK: The user need not follow the password policy in setting their password. This means no restraint of password length or repeating characters.
- ADMIN: States that only the user root can change the password information.
- ADMCHG: When set, the user will be prompted to change their password when next logging in. This overrides any maxage setting.

```
pwdadm -c <user>
```

The above example will clear all pwdadm flags previously set.

```
pwdadm -q <user>
```

The above example queries the pwdadm flags and reports back any flags set and the last time the password was changed (if the password has been set).

Examples of using pwdadm are now presented. First, query pwdadm to see what flags are set. In the this example user alpha is queried:

```
# pwdadm -q alpha
alpha:
        lastupdate = 1267374936
```

From the above output, you can tell that user alpha does have a password set by the lastupdate entry. If lastupdate is not present, then a password has never been set.

Now let's set the ADMCHK flag for that user:

```
```

```
# pwdadm -f ADMCHG alpha
```

Check again it has been set by querying `pwdadm`:

```
# pwdadm -q alpha
alpha:
        lastupdate = 1267374936
        flags = ADMCHG
```

Now let's clear all the flags:

```
# pwdadm -c alpha
```

Check it has been cleared:

```
# pwdadm -q alpha
alpha:
        lastupdate = 1267374936
```

Changing user attributes

To make a global change for the password or login attributes, edit the file `/etc/security/user`

In the defaults stanza section, change the default value of the attribute you wish to implement globally, as summarized in the previous section. To make individual user account changes use the `chuser` command. For example to change the `maxage` of user charlie to three weeks, you could use:

```
# chuser maxage=3 charlie
# lsuser -a maxage charlie
charlie maxage=3
```

To list all the attributes of a user, it may be easier for to use:

```
# lsuser -f charlie
charlie:
        id=211
        pgrp=staff
        groups=staff
        home=/home/charlie
        shell=/usr/bin/ksh
        gecos=charlie.suppt
        login=true
        …..
```

When does a password expire

When a user logs in, and their password is about to expire, they are presented with a warning. However, unless the user is observant, they will generally ignore this warning and wait until the actual password expires before changing it. If this is the case, you can be sure that some users will lock up their account trying their old password beyond what the `loginretries` allows. This means a call to the support desk to unlock the account, before they can change their password. Being proactive, you can determine when the password is due to expire and send a warning email to the user. They are more likely to pay attention to an email than a message on a screen once logged in. For local authentication, the attributes are present using the `pwdadm` and `lsuser` command to determine the next password change or the password expiry of that user.

As demonstrated earlier in this article, the expiry of a password is governed by the maxage attribute. For example:

`maxage=0` means never to expire

`maxage=2` means will expire in two weeks.

AIX (UNIX® and Linux®) stores the time in the epoch format in seconds, so first you must determine how many seconds in a week, as this is how `maxage` measures the time between password expiry, that is in week numbers.

There are 86400 seconds in a day, so multiplying that by seven comes in at 604800. So there are 604800 seconds in a week.

The next command you need to look at is the `pwdadm`, which in turn queries the file `/etc/security/passwd`. This file holds the values in seconds when a user last changed their password. Interrogating the file or using the `pwdadm` command will return the same result. For this demonstration, let us query the user spoll:

```
# grep -p "spoll:" /etc/security/passwd
spoll:
        password = EvqNjMMwJzXnc
        lastupdate = 1274003127
        flags =       ADMCHG

# pwdadm -q spoll
spoll:
        lastupdate = 1274003127
        flags = ADMCHG
```

You can see the `lastupdate` value in seconds from the above output. In other words, the last time the password was changed:

```
1274003127
```

Next, using the `lsuser` or interrogating the file with `/etc/security/user`, you can determine the number of weeks before the user spoll password will expire:

```
# grep -p "spoll:" /etc/security/user
spoll:
        admin = false
        maxage = 4

# lsuser -a maxage spoll
spoll maxage=4
```

You can see from the above output that the number of weeks before password expiry is 4.

The next task is then to multiply the number of seconds in a week by the number of weeks before the user spoll password is due to expire. In this case, it is 4:

```
604800 * 4
# expr 604800 \* 4
2419200
```

Next, you need to add the `maxage` value in seconds (604800 * 4) to the last time the password was changed:

```
2419200 +   1274003127
# expr 2419200 + 1274003127
1276422327
```

You can now convert that number of seconds from UNIX epoch into a more meaningful current time presentation. You can use different tools, but for this demonstration you'll use gawk with the `strftime` function:

```
# gawk 'BEGIN {print strftime("%c",'1276422327')}'
Sun Jun 13 10:45:27 BST 2010
```

The above calculation gives the time of the next password expiry.

So, you now know that user spoll's password was last changed on (from the `pwdadm` command):

```
# gawk 'BEGIN {print strftime("%c",'1274003127')}'
Sun May 16 10:45:27 BST 2010
```

And that it will expire on:

```
Sun Jun 13 10:45:27 BST 2010
```

Now you have the building blocks to determine when a password expires. It is a straight forward process to loop through all users and produce a report. Producing a report, especially when being audited, saves you time from the consuming task of extracting individual information. Once the report is produced you can print it or email it to the security administrators for review. I state this from experience, an auditor likes nothing better than a report detailing all the attributes you are required to produce, rather than ad-hoc screen shots. The report also serves as a reminder to system administrators to make sure their password policy is implemented as a standard across all servers. Viewing the report, you can spot any inconsistencies on the password policy between different users.

Ideally, a report should cover at least the following password attributes for each user:

- maxage
- pwdadm flags set
- last password change date
- next password change date

Listing 4 contains a script that will generate such a report on login password attributes. When this script is executed, a report is generated on the user's password attributes. The output taken from my system is contained in Listing 3. Looking more closely at Listing 3, it contains 5 columns:

- User: The actual user
- Change weeks: Weeks before the next password change (maxage value)
- Last change password: Date of last password change
- Flags: Any pwdadm flags set
- Next change password: Date of the next due password change

Listing 3. next_pwch command

```
# next_pwch1
user  change           last change        flags             next change
       weeks           password                             password
root     0  Sun Feb 21 09:44:59 GMT 2010
daemon   0  password never set
charlie  0  Sun Feb 28 16:35:36 GMT 2010
```

```
kilo     5  Mon May 17 09:38:57 BST 2010 NOCHECK   Mon Jun 21 09:38:57 BST 2010
xray     0  Sun Feb 28 16:42:20 GMT 2010
jane     0  password never set
plutt    1  Sat Apr 24 20:21:17 BST 2010           Sat May  1 20:21:17 BST 2010
spoll    4  Sun May 16 10:45:27 BST 2010           Sun Jun 13 10:45:27 BST 2010
foxtrot  5  Sat Feb 20 19:25:48 GMT 2010 ADMCHG    Sat Mar 27 19:25:48 GMT 2010
```

From the users on my system as contained in Listing 3. I can determine from the report the following:

- User daemon and jane have never had their initial password set.
- User root, charlie and xray do not have an entry for next password change; this is due to the `maxage=0` on the accounts (password never to expire).
- User kilo will not be forced to adhere to password rules, as denoted by the pwdadm NOCHECK flags.
- The `pwdadm` ADMCHG flags have been set on user foxtrot's account. This means he will be forced to change his password upon next login.

The script could easily be amended to include other user attributes like `rlogin, login, su` values.

Listing 4. next_pwch

```
!/bin/sh
# next_pwch
# display date of next passwd change

# maxage value is in number of weeks
# secs in a day is:86400 ..so
secs_in_week=604800

log=/home/dxtans/next_pwch.log
>$log
myhost=`hostname`
mydate=`date +"%d-%m-%y"`
 echo " Date: $mydate" >>$log

 echo "Local Password Expiry $myhost">>$log
list=$(lsuser -a registry ALL|grep -w files| awk '{print $1}')

echo "user  change           last change            flags         next change"
echo "       weeks           password                             password"

for user in $list
do
wks_before_ch=$(lsuser -a maxage $user | awk -F '=' '{print $2}')


 if [ "$wks_before_ch" = "" ]
  then
   # krb5 / ldap /limbo"
   expire="??"
  else
   expire=$wks_before_ch
```

```
 fi

last_ch_pw=$(pwdadm -q $user | grep last | awk '{print $3}')
# echo "last pw change : $last_ch_pw"
  if [ "$last_ch_pw" = "" ]
   then
    passset="password never set"
else
last_ch_pw_conv=$(gawk 'BEGIN {print strftime("%c",'$last_ch_pw')}')
 last_pw_ch=$last_ch_pw_conv
 passset=$last_pw_ch
total_secs=$(expr $secs_in_week \* $wks_before_ch)
#echo "total secs: $total_secs"

# weeks + last pw change
date_to_ch=`expr $total_secs + $last_ch_pw`

pw_flags=$(pwdadm -q $user | grep flags | awk '{print $3}')
 pw_flags=$pw_flags

# now convert to normal
next_pw_ch=$(gawk 'BEGIN {print strftime("%c",'$date_to_ch')}')
fi

#echo "..$user..$wks_before_ch..$passset"
  if [ "$wks_before_ch" = "0" ]
  then
  next_pw_ch=""
 else
  next_pw_ch=$next_pw_ch
fi

if [ "$passset" = "password never set" ]
 then
  #echo "..$user.."
    next_pw_ch=""
 fi
printf "%-8s %-2s %-28s %-10s%-28s\n"
 "$user" "$expire" "$passset" "$pw_flags" "$next_pw_ch"

done
```

Conclusion

Producing reports on security attributes of users is one method of determining and identifying if there are any inconsistencies in your password or security policy. When dealing with many remote hosts that are under your control, the scripts presented in this article are best ran under SSH from a deployment server to populate and then generate the reports to be sent to the system administrators.

Resources

Resources

Learn

- For additional information on the pwdadm command and /etc/security/user file.

Discuss

- Follow developerWorks on Twitter.
- Get involved in the My developerWorks community.
- Participate in the AIX and UNIX® forums:
  - AIX Forum
  - AIX Forum for developers
  - Cluster Systems Management
  - IBM Support Assistant Forum
  - Performance Tools Forum
  - Virtualization Forum
  - More AIX and UNIX Forums

About the author

David Tansley

David Tansley is a freelance writer. He has 15 years of experience as a UNIX administrator, using AIX the last eight years. He enjoys playing badminton, then relaxing watching Formula 1, but nothing beats riding and touring on his GSA motorbike with his wife.