Customizing the password popup window

PASSWORD TRICKS

We'll show you some tricks for configuring the root password popup window on Red Hat-based systems. **BY ARMIJN HEMEL**

If you use Red Hat Enterprise Linux or a Red Hat derivative such as Fedora Core or CentOS, you are probably familiar with the popup window that asks you for the root password before you can launch a program that should run as root (Figure 1).

Most programs that use this authentication mechanism are tools that need extra privileges on the system. An example of this kind of tool is a program that requires access to a raw network device or a program that needs write access to directories with configuration information.

What you may not know is that the whole mechanism behind this popup is completely customizable. In this article, I'll show you how this mechanism works, and I'll also show you how you can change the default behavior.

consolehelper and userhelper

The two packages that make this possible are called *usermode* and *usermode-gtk*. The *usermode* package contains two tools that are important: *consolehelper* and *userhelper*. The *usermode-gtk* package contains the familiar graphical popup window and a few other tools.

consolehelper and *userhelper* work together to let a non-root user execute a program that runs as root. Consider, for instance, the network sniffer Ethereal. Ethereal requires root privileges because it needs to set the network device to promiscuous mode.

On a default install, the menu is configured so that if you select Ethereal, the program */usr/bin/ethereal* is launched. This program is just a symbolic link to *consolehelper*:

```
$ ls -l /usr/bin/ethereal
lrwxrwxrwx  1 root root ➥
13 aug 11 20:01 ➥
/usr/bin/ethereal -> ➥
consolehelper
```

When *consolehelper* is executed, it starts *userhelper*. The permissions of *userhelper* are set to SUID root, so it runs as root when it is invoked. If a program is SUID (commonly referred to as "setuid"), when the program is started, it will run with all the privileges of the owner of the program. Since *userhelper* is owned by root and SUID, it will execute with full root privileges. When *consolehelper* starts *userhelper*, it passes the name of the program it needs to run (in our example, Ethereal) to *userhelper*, which then tries to authenticate the user. If authentication succeeds, *userhelper* (running as root) will start the program as root.

The *userhelper* program first checks a configuration file to determine which program really should be run; it then looks at a number of other configuration options. *userhelper* then invokes the Pluggable Authentication Modules system (PAM) to see if the user is allowed to use the service. If the service is allowed, the real program is launched. If the user has failed to authenticate correctly, an error message is displayed (Figure 2).

Figure 1: Red Hat users are familiar with the popup that asks for the root password.

You'll find configuration files for *userhelper* in */etc/security/console.apps/*. The configuration for Ethereal looks like:

```
USER=root
PROGRAM=/usr/sbin/ethereal
SESSION=true
FALLBACK=true
```

In this example, the program will ask for the password of user *root*. If the authen-

tication via PAM is successful, the program located in */usr/sbin/ethereal* will be executed. The variable *SESSION* indicates whether or not to use session management in PAM.

For graphical applications, such as Ethereal, session management is needed. The *FALLBACK* variable is used to determine if the program should still be run even if authentication fails. The program will, of course, be run without all privileges and not everything will work. In the case of Ethereal, it can still read network dumps, but it will not be able to make any new ones.

Another part of the configuration is done via PAM. These settings, which can be found in */etc/pam.d/*, follow the PAM standards.

This approach has one weakness: if the directory that contains the "real" program (in this case */usr/sbin*) is earlier in the *$PATH* than the directory where *consolehelper* is located (by default, */usr/bin*), *consolehelper* will not be invoked first and the program will either not run, or it will not run with the right permissions. You should keep this in mind if you decide to rearrange the order in which directories in the *$PATH* are searched.

Simple Adaptations

I personally have always felt it was a bit dangerous when credentials are cached. You have probably seen the little keyring icon in one of your menu bars after giving the root password through the usermode mechanism (see Figure 3). If this keyring is visible, you can execute programs that would otherwise require you to give the root password via usermode.

Many programs have this rule in their PAM configuration file by default:

```
session  optional ➥
pam_timestamp.so
```

This setting means that PAM will keep a timestamp. When a program has this rule in its PAM configuration file, the program will first check if there has already been a valid, cached, authentication:

```
auth sufficient ➥
pam_timestamp.so
```

Commenting out the rules mentioned above will prevent the user's credentials from being cached after a successful authentication and will prevent authorization using cached credentials.

Figure 3: If the system has credentials for the user, the keyring will be visible in the panel.

Advanced Usage

Say you want to lock down which users can execute certain programs. There are several ways you can achieve this. The first way is through the configuration of *consolehelper* itself; the other, and more powerful way, is through PAM.

If you are configuring through *consolehelper*, you can prevent users from having to know the root password by having them authenticate using their own passwords. Just change the line in the *userhelper* configuration in */etc/security/console.apps/* from authenticating as root to the following:

```
USER=<user>
```

If you do this, you will be asked for your own password. If you enter your password successfully, the program will start. Keep in mind that the fact that you use your own password doesn't change anything with regard to the program's privileges. The program that is started is still run as root, so be sure you use this feature carefully.

If you set *USER* to the special variable *<none>*, no access will be allowed:

```
USER=<none>
```

If you try to launch Ethereal now, you will get an error and the program will not start (Figure 4). You will probably wonder why this feature is useful. One of the other configuration parameters is *UGROUPS*. This parameter can be assigned a comma-separated list of groups. Members of these groups will be authenticated as if their names were given in the configuration file (*USER = <user>*). That is, they will be authenticated with their own passwords. If the configuration has *USER = root* set, all other users will have to give the root password. If you use both *USER = <none>* and *UGROUPS* in the configuration, you can give members of certain groups the ability to run programs without them having to know the root password. This way you can create sudo-like behavior.

Figure 2: The user fails to authenticate correctly.

Customization using PAM

PAM offers a great variety of modules, including options for authenticating against a database or hardware token, and even tools for voice recognition. In this article I will only give a few small ready-to-work examples of how you can play with PAM inside *consolehelper*.

On certain machines on my network, I only want certain users to be able to sniff network traffic. For this purpose, PAM has a module called *pam_localuser*. It normally checks in */etc/passwd* if a certain account is listed, but it can also be configured to use a separate file. I keep the users I want to give permission in a file called */etc/localusers*, which I make only readable and writable for root. In this file, I only have two users. The file is in the same format as */etc/passwd*:

```
root:x:0:0::
armijn:x:500:500::
```

In the PAM configuration, I add the following rule:

```
account required ➥
pam_localuser.so ➥
file=/etc/localusers
```

Now when Ethereal is started via usermode, it also checks whether or not the user is in */etc/localusers*. Before this can work cleanly, one change is needed: the user that *consolehelper* authenticates should be changed from root to the actual user. If not, the system will check if root is in */etc/localusers* instead of checking for the real user! The configuration for Ethereal should be changed to:

```
USER=<user>
PROGRAM=/usr/sbin/ethereal
```

```
SESSION=true
FALLBACK=true
```

If I remove the user *armijn* from this file, I can't launch Ethereal with full privileges anymore, except as root. With the localuser PAM module, you can have separate files with permissions for each program.

Figure 4: The user doesn't have enough permissions to launch the program.

Other uses for this particular module are the power-off, halt, and reboot commands. On Red Hat Linux (and derivates) these tools are also controlled by usermode. By adding a file where you list authorized users, you can control which users have which rights – for example, you can control which users have the right to power off the machine. These three commands also use another interesting PAM module titled *pam_console*, which checks if the user is currently logged in at the local console.

Two of the rules in the PAM configuration for these commands are:

```
auth sufficient ➥
pam_rootok.so
auth required ➥
pam_console.so
```

These rules mean that only root and the user who is logged in at the console can power off, halt, or reboot. Users who are just logged in remotely, except root, are not allowed to turn off or reboot the machine. By removing the second rule, the user currently logged in at the console can't perform these actions anymore, except if that user happens to be root. Of course, you can still power off the system using GDM.

Wise words

The techniques described in this article are useful in some environments, but in other cases, they may be totally unnecessary. Before you start rebuilding your whole system to use *usermode*, you should keep in mind that the tools that make use of *usermode* all have one characteristic in common: they need to access files that normally cannot be accessed or modified by mere mortal users.

Often there is no reason at all to change the default security system. Before you decide to deploy *usermode* to restrict access to programs, you should check to see if this option is really worth the effort, and make sure you aren't just needlessly complicating your security configuration.

More information

If you're looking for more information, you could start with the manpages for *consolehelper* and *userhelper*. The *userhelper* manpage is especially useful. There are a lot of resources for help with PAM. Start with the Linux-PAM webpage, which you will find at *http://www.kernel.org/pub/linux/libs/pam/*. Another good resource on PAM is *Essential System Administration, 3rd edition*, by AEleen Frisch, which is published by O'Reilly. ■