X Windows Security

Improving X security using **mit-magic-cookie** and **xauth**.

by Ben Gross & Baba Buehler
Beckman Institute System Services
bgross@uiuc.edu, baba@beckman.uiuc.edu

Information culled from *X Window System Administrator's Guide* by Linda Mui and Eric Pearce, from *O'Reilly & Associates*, the *comp.windows.x* faq, and Marc VanHeyningen's X Security notes from Indiana University. *(http://cs.indiana.edu/X/security/intro.html)*

This document is part of the Beckman Institute Systems Services Virtual Library
http://www.beckman.uiuc.edu/groups/biss/VirtualLibrary/

Why magic cookies?

The `xhost` based X authorization that most people use is functional for single-user workstations. For multi-user systems and lab environments, however, `xhost` has some serious holes.

Beyond allowing someone to plaster your X display with pictures of Ren and Stimpy, `xhost` allows anyone who can log into (or break into) your system to run programs on your display. Attacks can include:

1. destroying any (or all) of your windows,
2. opening new windows on *your* screen,
3. viewing the contents of your screen *remotely*,
4. logging keystrokes (*including passwords*) and
5. generating spurious X events remotely.

Most of these can be done with no warning and leaving no trace.

Despite this, `xhost` is still the only X security many people use. The benefits of `xhost` are plain: easily running X programs on remote systems and displaying them locally. Type:

```
% xhost +powdered.toast.com
```

and log into `powdered.toast.com`, then you can run any X program on `powdered.toast.com` and display it locally. This, however, also allows **anyone** else on `powdered.toast.com` to use (or abuse) your local workstation display.

Using `xauth` can prevent this. Running remote X programs is a bit more complex, but it comes much closer to guaranteeing that you are the only one who can run programs on your local X display.

It should be noted that **mit-magic-cookie** only works with X11R4 and X11R5.

How it works

`xauth` works by creating a file called `.Xauthority` in your home directory. This file contains display names followed by a hexadecimal number (the *magic cookie*). The X server reads the magic cookie from this file and then requires any X program you run to provide the same magic cookie.

Your X client programs also read the `.Xauthority` file to obtain the magic cookie. The client sends the magic cookie to the server, then the server verifies that it matches the one in memory. If they match, the client is allowed to open a window on the X server.

If you are running X programs on the same machine your X server is on, then the security is trivial, because both programs have access to the `.Xauthority` file. The `.Xauthority` file permissions are set to `600`, theoretically preventing anyone else on your system from reading your magic cookies.

If you need to run X programs on a remote host, you must first propagate the magic cookie for your local display to the remote host. `xauth` and X provide ways of doing this.

Setting it up

Step 1 - baking magic cookies

The first step is to create an `.Xauthority` file with a magic cookie for your host. This is easiest done in your `.login` or `.kshrc` file. This code can also be included in an `.xserverrc` file.

The `xauth` program creates and manages the `.Xauthority` file. Make sure `/usr/bin/X11` (or wherever your X binaries are) is in your path. It is best to set the X path in your startup files, either `.profile, .kshrc, or .cshrc` since these files are read for `rsh` commands. You'll be using `rsh` to execute X programs on remote hosts. Here is some sample code using `perl` and `sh or ksh` for your Korn Shell or Born Shell `.profile`:

One of the best ways we have found to generate a random non-reproducable number with output in hex is to use common system commands and the MD5 cyptographic sum generator. MD5 can be retrieved from ftp.cert.org.

A `csh` example using system statistics and MD5.

```
set randomkey=`(ps -ael; nfsstat -m; nfsstat -r; netstat -s; netstat -m; date)` |

xauth add `hostname`/unix:0 . $randomkey
xauth add `hostname`:0 . $randomkey
```

A `sh` or `ksh` equivalent would be:

```
randomkey=$( (ps -ael; nfsstat -m; nfsstat -r; netstat -s; netstat -m; date) | md5

xauth add $(hostname)/unix:0 . $randomkey
xauth add $(hostname):0 . $randomkey
```

Note: these command line options should work with BSD and System V varients. A `csh` example using perl.

```
set randomkey=`perl -e 'for (1..4) { \
  srand(time+$$+$seed); \
  printf("%4.5x", ($seed = int(rand(65536)))); } \
print "\n";'`

xauth add `hostname`/unix:0 . $randomkey
xauth add `hostname`:0 . $randomkey
```

An important note: the magic cookie **must** be a string of hexadecimal digits that is an *even* length. MD5 does this automatically.

The benefits here are

- a new magic cookie is generated each time you log in and
- once installed it is virtually transparent

"`xauth add`" stores the magic cookie in your `.Xauthority` file in your home directory, and ensures the permissions are `-rw-------` restricting access to the file.

Step 2 - enabling xauth authentication

Now that you have a `.Xauthority` file with your magic cookie in it you need to tell the X server to use this to authenticate X programs that want to use your display. You can do this by passing the `-auth` argument when starting the X server.

If you are using `xinit` in your `.login` you would change the line to something like:

```
xinit $HOME/.xinitrc -- /usr/bin/X11/X -auth $HOME/.Xauthority
```

It is important to use the "`--`" argumment to `xinit`, this tells `xinit` to pass the following arguments on to the X server. Something like

```
xinit $HOME/.xinitrc -- -auth $HOME/.Xauthority
```

will probably work, while this

```
xinit $HOME/.xinitrc -auth $HOME/.Xauthority
```

will not.

If you use a `.xserverrc` file to start your X server, add the `-auth` option:

```
exec /usr/bin/X11/X -auth $HOME/.Xauthority
```

Once the X server is started with the `-auth` parameter, it will use the magic cookies to verify programs wanting to use your X display. **Unless** there are still hosts authorized with `xhost`.

Step 3 - clear xhost's authorization list

`xhost` supersedes `xauth` so if you still have hosts enabled under `xhost` then the magic cookie scheme will do you no good.

Check your `/etc/X0.hosts` to see which hosts will be automatically authorized, or put a

```
xhost -
```

in your `.login` or equivalent file. This will effectively disable `xhost` authorization, passing all authorization on to `xauth` with magic cookies.

The *X Windows System Administrator's Guide* warns against totally removing or disabling `xhost` because this could be equivalent to having `xhost +`, allowing access to **all hosts**. Using "`xhost -`" leaves `xhost` enabled, but empties the list of authorized hosts so all authorization happens through `xauth` and magic cookies.

At this point, you should be secure to operate X on a single workstation. You should be able to run any clients from your host and they will read your `.Xauthority` file. If your home directory is shared among many hosts (*ex.* through *NFS* mounting), you can run clients from any of those hosts, because they will all have access to the `.Xauthority` file in your shared home directory.

Step 4 - running clients on remote hosts (xauth/rsh)

In order to run clients on remote hosts, you must have a way of getting the magic cookie to those clients so they can authenticate themselves to your local X server. `xauth` provides a way to create a `.Xauthority` file on your remote host so remote clients will be able to read the magic cookie.

You must first make certain that you have your local host in the `.rhosts` or `/etc/hosts.equiv` on the remote machine, otherwise `rsh` will not work. Also make sure the path to `xauth` (usually `/usr/bin/X11`) is set in the `.cshrc` on the remote host.

Now you are ready to propagate the magic cookie to the remote host. For example, if you're on host `ren` and you want to run a client on `stimpy` you would first type:

```
ren% xauth extract - ren:0 | rsh stimpy xauth merge -
```

This pipes the magic cookie to `xauth` on `stimpy` which builds a new `.Xauthority` file with the magic cookie for display `ren:0`. Now you could execute

```
ren% rsh stimpy /usr/bin/X11/xterm -display ren:0
```

and `xterm` on `stimpy` would read the new `.Xauthority` and find a magic cookie allowing it to run on `ren`'s display.

Step 5 - automation with xrsh

The above steps have been combined in a system called `xrsh` which is available as `/contrib/xrsh-5.4.shar` at ftp.x.org. `xrsh` automatically does the `xauth` stuff for you, as well as always using `csh` since `ksh` doesn't read any init files for a `rsh`, leaving you without a path set. `xrsh` provides a `xrlogin` command as well as some other security precautions. We highly recommend installing it.

Using `xrsh`, and assuming the path to the X binaries is set in the `.cshrc` on `stimpy`, the above would become:

```
ren% xrsh -auth xauth -pass ENV stimpy xterm
```

Be sure to include the `-auth xauth` because `xrsh` defaults to use `xhost` authentication. You could alias `xrsh` to include `-auth xauth -pass ENV`:

```
ren% alias xrsh 'xrsh -auth xauth -pass ENV \!*'
```

and then use:

```
ren% xrsh stimpy xterm
```

That should allow you to accomplish anything that was previously done with `xhost`. Furthermore, you can still use `xhost` if you feel the need, although this instantly compromises any `xauth` security you have set up. If you feel the need to use `xhost` we recommend issuing a

```
% xhost -
```

immediately after finishing whatever required `xhost`, or simply logging out and logging back in, which will generate a new magic cookie and restart X with `xauth` authorization reset.

Further X security

For some reason unbeknownst to the authors, many systems come with all window security disabled. The file that controls the owner and group permissions of window related devices is the framebuffer table (`/etc/fbtab`). Read the man page `fbtab(5)` for more information.

From the *comp.windows.x* faq: "...this permits anyone who can log into your workstation to peek at your windowing session by accessing the frame-buffer directly, or, as less of a privacy issue but perhaps more annoying, to [accidentally] start up a second X session on your console display."

Your system formats may vary. We have these lines uncommented in our `/etc/fbtab` on our Suns:

```
/dev/console    0600    /dev/kbd:/dev/mouse
/dev/console    0600    /dev/fb
/dev/console    0600    /dev/cgsix0
/dev/console    0600    /dev/audio
/dev/console    0600    /dev/audioctl
```

The local X server stores files in `/tmp/.X11-unix`. The file X0 is a socket descriptor used by X to connect to the local server. To make sure no other user can delete your X0 file, set the sticky bit on the directory.

```
chmod 1777 /tmp
chmod 1777 /tmp/.X11-unix
```

In case you do not have the keyboard permissions for your `/dev/kbd`, you may want to utilize xterm's Secure Keyboard option. This can be accessed by holding down the control key and pressing the first mouse button (it may vary from system to system). The feature will prevent other users from using the `GrabKeyboard()` protocol request. The screen should reverse colors, if not, check to make sure no one is watching you type. This feature can only be used in one window at a time so it is primarily used for typing sensitive data, such as your password, definitely your root password.

Implementing these measures will go a long way toward ensuring your X workstation is secure from attacks through the X server. The default X security is simply inadequate at preventing attacks and data loss.

---

bzb/bmg (6/13/94)