Remote X Apps mini–HOWTO

Table of Contents

Remote X Apps mini−HOWTO....1
Vincent Zweije, zweije@xs4all.nl....1
1.Introduction....1
2.Related Reading....1
3.The Scene....1
4.A Little Theory....1
5.Telling the Client....1
6.Telling the Server....1
7.X Applications from Another User−id....2
8.Running a Remote Window Manager....2
9.Troubleshooting....2
1.Introduction....2
2.Related Reading....2
3.The Scene....3
4.A Little Theory....3
5.Telling the Client....4
6.Telling the Server....5
6.1 Xhost....5
6.2 Xauth....6
Making the Cookie....7
Transporting the Cookie....8
Using the Cookie....9
6.3 Ssh....9
7.X Applications from Another User−id....9
7.1 Different Users on the Same Host....10
7.2 Client User Is Root....11
8.Running a Remote Window Manager....12
9.Troubleshooting....12

Remote X Apps mini−HOWTO

Vincent Zweije, zweije@xs4all.nl

v, 19 November 1999

This mini−HOWTO describes how to run remote X applications. That is, how to have an X program display on a different computer than the one it's running on. Or conversely: how to make an X program run on a different computer than the one you're sitting at. The focus of this mini−HOWTO is on security. This mini−HOWTO also contains information on running X applications locally, but with a different user−id.

1.Introduction

2.Related Reading

3.The Scene

4.A Little Theory

5.Telling the Client

6.Telling the Server

- 6.1 Xhost
- 6.2 Xauth
- 6.3 Ssh

7.X Applications from Another User−id

- 7.1 Different Users on the Same Host
- 7.2 Client User Is Root

8.Running a Remote Window Manager

9.Troubleshooting

1.Introduction

This mini−HOWTO is a guide how to do remote X applications. It was written for several reasons.

1. Many questions have appeared on usenet on how to run a remote X application.
2. I see many, many hints of ``use `xhost +hostname`'' or even ```xhost +`'' to allow X connections. **This is ridiculously insecure**, and there are better methods.
3. I do not know of a simple document that describes the options you *do* have. Please inform me zweije@xs4all.nl if you know more.

This document has been written with unix−like systems in mind. If either your local or remote operating system are of another flavour, you may find here how things work. However, you will have to translate examples yourself to apply to your own system(s).

The most recent version of this document is always available on WWW at http://www.xs4all.nl/~zweije/xauth.html. It is also available as the Linux Remote X Apps mini−HOWTO at http://sunsite.unc.edu/LDP/HOWTO/mini/Remote−X−Apps. Linux (mini−)HOWTOs are available by http or ftp from sunsite.unc.edu.

This is version 0.6.1. No guarantees, only good intentions. I'm open to suggestions, ideas, additions, useful pointers, (typo) corrections, etc... I want this to remain a simple readable document, though, in the best−meant HOWTO style. Flames to `/dev/null`.

Contents last updated on 19 November 1999 by Vincent Zweije

2.Related Reading

A related document on WWW is ``What to do when Tk says that your display is insecure'', http://ce−toolkit.crd.ge.com/tkxauth/. It was written by Kevin Kenny. It suggests a similar solution to X authentication to that in this document (xauth). However, Kevin aims more at using xdm to steer xauth for you.

The X System Window System Vol. 8 X ``Window System Administrator's Guide'' from O'Reilly and Associates has also been brought to my attention as a good source of information. Unfortunately, I've not been able to check it out.

Yet another document much like the one you're reading now, titled ``Securing X Windows'', is available at http://ciac.llnl.gov/ciac/documents/ciac2316.html.

Also check out usenet newsgroups, such as `comp.windows.x`, `comp.os.linux.x`, and `comp.os.linux.networking`.

3.The Scene

You're using two computers. You're using the X window system of the first to type to and look at. You're using the second to do some important graphical work. You want the second to show its output on the display of the first. The X window system makes this possible.

Of course, you need a network connection for this. Preferably a fast one; the X protocol is a network hog. But with a little patience and suitable protocol compression, you can even run applications over a modem. For X protocol compression, you might want to check out dxpc http://ccwf.cc.utexas.edu/~zvonler/dxpc/ or LBX http://www.ultranet.com/~pauld/faqs/LBX−HOWTO.html (also known as the LBX mini−HOWTO).

You must do two things to achieve all this:

1. Tell the local display (the server) to accept connections from the remote computer.
2. Tell the remote application (the client) to direct its output to your local display.

4.A Little Theory

The magic word is `DISPLAY`. In the X window system, a display consists (simplified) of a keyboard, a mouse and a screen. A display is managed by a server program, known as an X server. The server serves displaying capabilities to other programs that connect to it.

A display is indicated with a name, for instance:

- `DISPLAY=light.uni.verse:0`
- `DISPLAY=localhost:4`
- `DISPLAY=:0`

The display consists of a hostname (such as `light.uni.verse` and `localhost`), a colon (`:`), and a sequence number (such as `0` and `4`). The hostname of the display is the name of the computer where the X server runs. An omitted hostname means the local host. The sequence number is usually 0 −− it can be varied if there are multiple displays connected to one computer.

If you ever come across a display indication with an extra `.n` attached to it, that's the screen number. A display can actually have multiple screens. Usually there's only one screen though, with number `n=0`, so that's the default.

Other forms of `DISPLAY` exist, but the above will do for our purposes.

For the technically curious:

- `hostname:D.S` means screen `S` on display `D` of host `hostname`; the X server for this display is listening at TCP port `6000+D`.
- `host/unix:D.S` means screen `S` on display `D` of host `host`; the X server for this display is listening at UNIX domain socket `/tmp/.X11−unix/XD` (so it's only reachable from `host`).
- `:D.S` is equivalent to `host/unix:D.S`, where `host` is the local hostname.

5.Telling the Client

The client program (for instance, your graphics application) knows which display to connect to by inspecting the `DISPLAY` environment variable. This setting can be overridden, though, by giving the client the command line argument `−display hostname:0` when it's started. Some examples may clarify things.

Our computer is known to the outside as light, and we're in domain uni.verse. If we're running a normal X server, the display is known as `light.uni.verse:0`. We want to run the drawing program xfig on a remote computer, called `dark.matt.er`, and display its output here on light.

Suppose you have already telnetted into the remote computer, `dark.matt.er`.

If you have csh running on the remote computer:

```
dark% setenv DISPLAY light.uni.verse:0
dark% xfig &
```

or alternatively:

```
dark% xfig −display light.uni.verse:0 &
```

If you have sh running on the remote computer:

```
dark$ DISPLAY=light.uni.verse:0
dark$ export DISPLAY
dark$ xfig &
```

or, alternatively:

```
dark$ DISPLAY=light.uni.verse:0 xfig &
```

or, of course, also:

```
dark$ xfig −display light.uni.verse:0 &
```

It seems that some versions of telnet automatically transport the `DISPLAY` variable to the remote host. If you have one of those, you're lucky, and you don't have to set it by hand. If not, most versions of telnet do transport the `TERM` environment variable; with some judicious hacking it is possible to piggyback the `DISPLAY` variable on to the `TERM` variable.

The idea with piggybacking is that you do some scripting to achieve the following: before telnetting, attach the value of `DISPLAY` to `TERM`. Then telnet out. At the remote end, in the applicable `.*shrc` file, read the value of `DISPLAY` from `TERM`.

6.Telling the Server

The server will not accept connections from just anywhere. You don't want everyone to be able to display windows on your screen. Or read what you type −− remember that your keyboard is part of your display!

Too few people seem to realise that allowing access to your display poses a security risk. Someone with access to your display can read and write your screens, read your keystrokes, and read your mouse actions.

Most servers know two ways of authenticating connections to it: the host list mechanism (xhost) and the magic cookie mechanism (xauth). Then there is ssh, the secure shell, that can forward X connections.

6.1 Xhost

Xhost allows access based on hostnames. The server maintains a list of hosts which are allowed to connect to it. It can also disable host checking entirely. Beware: this means no checks are done, so *every* host may connect!

You can control the server's host list with the xhost program. To use this mechanism in the previous example, do:

```
light$ xhost +dark.matt.er
```

This allows all connections from host `dark.matt.er`. As soon as your X client has made its connection and displays a window, for safety, revoke permissions for more connections with:

```
light$ xhost −dark.matt.er
```

You can disable host checking with:

```
light$ xhost +
```

This disables host access checking and thus allows *everyone* to connect. You should *never* do this on a network on which you don't trust *all* users (such as Internet). You can re−enable host checking with:

```
light$ xhost −
```

xhost − by itself does *not* remove all hosts from the access list (that would be quite useless − you wouldn't be able to connect from anywhere, not even your local host).

Xhost is a very insecure mechanism. It does not distinguish between different users on the remote host. Also, hostnames (addresses actually) can be spoofed. This is bad if you're on an untrusted network (for instance already with dialup PPP access to Internet).

6.2 Xauth

Xauth allows access to anyone who knows the right secret. Such a secret is called an authorization record, or a magic cookie. This authorization scheme is formally called MIT−MAGIC−COOKIE−1.

The cookies for different displays are stored together in `~/.Xauthority`. Your `~/.Xauthority` must be inaccessible for group/other users. The xauth program manages these cookies, hence the nickname xauth for the scheme.

On starting a session, the server reads a cookie from the file that is indicated by the `−auth` argument. After that, the server only allows connections from clients that know the same cookie. When the cookie in `~/.Xauthority` changes, *the server will not pick up the change*.

Newer servers can generate cookies on the fly for clients that ask for it. Cookies are still kept inside the server though; they don't end up in `~/.Xauthority` unless a client puts them there. According to David Wiggins:

> A further wrinkle was added in X11R6.3 that you may be interested in. Via the new SECURITY extension, the X server itself can generate and return new cookies on the fly. Furthermore, the cookies can be designated ``untrusted'' so that applications making

connections with such cookies will be restricted in their operation. For example, they won't be able to steal keyboard/mouse input, or window contents, from other trusted clients. There is a new ``generate" subcommand to xauth to make this facility at least possible to use, if not easy.

Xauth has a clear security advantage over xhost. You can limit access to specific users on specific computers. It does not suffer from spoofed addresses as xhost does. And if you want to, you can still use xhost next to it to allow connections.

Making the Cookie

If you want to use xauth, you must start the X server with the `−auth authfile` argument. If you use the startx script, that's the right place to do it. Create the authorization record as below in your startx script.

Excerpt from `/usr/X11R6/bin/startx`:

```
mcookie|sed −e 's/^/add :0 . /'|xauth −q
xinit −− −auth "$HOME/.Xauthority"
```

Mcookie is a tiny program in the util−linux package, primary site ftp://ftp.math.uio.no/pub/linux/. Alternatively, you can use md5sum to massage some random data (from, for instance, `/dev/urandom` or `ps −axl`) into cookie format:

```
dd if=/dev/urandom count=1|md5sum|sed −e 's/^/add :0 . /'|xauth −q
xinit −− −auth "$HOME/.Xauthority"
```

If you can't edit the startx script (because you aren't root), get your system administrator to set up startx properly, or let him set up xdm instead. If he can't or won't, you can make a `~/.xserverrc` script. If you have this script, it is run by xinit instead of the real X server. Then you can start the real X server from this script with the proper arguments. To do so, have your `~/.xserverrc` use the magic cookie line above to create a cookie and then exec the real X server:

```
#!/bin/sh
mcookie|sed −e 's/^/add :0 . /'|xauth −q
exec /usr/X11R6/bin/X "$@" −auth "$HOME/.Xauthority"
```

If you use xdm to manage your X sessions, you can use xauth easily. Define the DisplayManager.authDir resource in `/etc/X11/xdm/xdm−config`. Xdm will pass the `−auth` argument to the X server when it starts. When you then log in under xdm, xdm puts the cookie in your `~/.Xauthority` for you. See xdm(1) for more information. For instance, my `/etc/X11/xdm/xdm−config` has the following line in it:

```
DisplayManager.authDir: /var/lib/xdm
```

Transporting the Cookie

Now that you have started your X session on the server host `light.uni.verse` and have your cookie in `~/.Xauthority`, you will have to transfer the cookie to the client host, `dark.matt.er`.

The easiest is when your home directories on light and dark are shared. The `~/.Xauthority` files are the same, so the cookie is transported instantaneously. However, there's a catch: when you put a cookie for `:0` in `~/.Xauthority`, dark will think it's a cookie for itself instead of for light. You must use an explicit host name when you create the cookie; you can't leave it out. You can install the same cookie for both `:0` and `light:0` with:

```
#!/bin/sh
cookie=`mcookie`
xauth add :0 . $cookie
xauth add "$HOST:0" . $cookie
exec /usr/X11R6/bin/X "$@" −auth "$HOME/.Xauthority"
```

If the home directories aren't shared, you can transport the cookie by means of rsh, the remote shell:

```
light$ xauth nlist "${HOST}:0" | rsh dark.matt.er xauth nmerge −
```

1. Extract the cookie from your local `~/.Xauthority` (`xauth nlist :0`).
2. Transfer it to dark.matt.er (`| rsh dark.matt.er`).
3. Put it in the `~/.Xauthority` there (`xauth nmerge −`).

Notice the use of `${HOST}`. You need to transport the cookie that is explicitly associated with the local host. A remote X application would interpret a display value of `:0` as referring to the remote machine, which is not what you want!

It's possible that rsh doesn't work for you. Besides that, rsh also has a security drawback (spoofed host names again, if I remember correctly). If you can't or don't want to use rsh, you can also transfer the cookie manually, like:

```
light$ echo $DISPLAY
:0
light$ xauth list $DISPLAY
light/unix:0 MIT−MAGIC−COOKIE−1 076aaecfd370fd2af6bb9f5550b26926
light$ rlogin dark.matt.er
Password:
```

```
dark% setenv DISPLAY light.uni.verse:0
dark% xauth add $DISPLAY . 076aaecfd370fd2af6bb9f5550b26926
dark% xfig &
[15332]
dark% logout
light$
```

See also rsh(1) and xauth(1x) for more information.

It may be possible to piggyback the cookie on the `TERM` or `DISPLAY` variable when you do a telnet to the remote host. This would go the same way as piggybacking the `DISPLAY` variable on the `TERM` variable. See section 5: Telling the Client. You're on own here from my point of view, but I'm interested if anyone can confirm or deny this.

Using the Cookie

An X application on dark.matt.er, such as xfig above, will automatically look in `~/.Xauthority` there for the cookie to authenticate itself with.

There's a little wrinkle when using `localhost:D`. X client applications may translate `localhost:D` into `host/unix:D` for the purpose of cookie retrieval. Effectively, this means that a cookie for `localhost:D` in your `~/.Xauthority` has *no* effect.

6.3 Ssh

Authority records are transmitted with no encryption. If you're even worried someone might snoop on your connections, use ssh, the secure shell. It will do X forwarding over encrypted connections. And besides, it's great in other ways too. It's a good structural improvement to your system. Just visit http://www.cs.hut.fi/ssh/, the ssh home page.

Who knows anything else on authentication schemes or encrypting X connections? Maybe kerberos?

7.X Applications from Another User−id

Suppose you want to run a graphical configuration tool that requires root privileges. However, your X session is running under your usual account. It may seem strange at first, but the X server will *not* allow the tool to access your display. How is this possible when root can normally do anything? And how do you work around this problem?

Let's generalise to the situation where you want to an X appliation under a user−id `clientuser`, but the X session was started by `serveruser`. If you have read the section on cookies, it is clear why `clientuser` cannot access your display: `~clientuser/.Xauthority` does not contain the right

magic cookie for accessing the display. The right cookie is found in `~serveruser/.Xauthority`.

7.1 Different Users on the Same Host

Of course, anything that works for remote X also works for X from a different user−id as well (particularly `slogin localhost −l clientuser`). It's just that the client host and the server host happen to be the same. However, when both hosts are the same, there are some shortcuts for transferring the magic cookie.

We'll assume that you use `su` to switch user−ids. Basically, what you have to do is write a script that will call `su`, but wraps the command that `su` executes with some code that does the necessary things for remote X. These necessary things are setting the `DISPLAY` variable and transferring the magic cookie.

Setting `DISPLAY` is relatively easy; it just means defining `DISPLAY="$DISPLAY"` before running the su command argument. So you could just do:

```
su − clientuser −c "env DISPLAY=$DISPLAY clientprogram &"
```

This doesn't work yet, because we still have to transfer the cookie. We can retrieve the cookie using `xauth list "$DISPLAY"`. This command happens to list the cookie in a format that's suitable for feeding back to `xauth`; just what we need! So we feed the obtained cookie back to `xauth` in the `su` command, set `DISPLAY there`, and run the command we want.

```
su − clientuser −c "xauth add `xauth list $DISPLAY`; \
                    exec env DISPLAY=$DISPLAY clientprogram"
```

You can write a script around this, parameterizing by `clientuser` and `clientprogram`. Let's improve the script a little while we're at it, making it less readable but more robust. It looks like this:

```
#!/bin/sh
if [ $# −lt 2 ]
then echo "usage: `basename $0` clientuser command" >&2
     exit 2
fi
CLIENTUSER="$1"; shift
exec su − "$CLIENTUSER" −c "xauth add `xauth list \"$DISPLAY\"`; \
                            exec env DISPLAY='$DISPLAY' "'"$SHELL"'" −c '$*'"
```

I think this is portable and works well enough in most circumstances. The only shortcoming I can think of right now is that, due to using `'$*'`, single quotes in `command` will mess up quoting in the `su` command argument (`'$*'`). If there's anything seriously wrong with it, please drop me an email.

Call the script `/usr/local/bin/xsu`, and you can do:

```
xsu clientuser 'command &'
```

Easy, no?

7.2 Client User Is Root

Obviously, anything that works for non−root client users is going to work for root as well. However, with root you can make it even easier, because root can read anyone's `~/.Xauthority` file. There's no need to transfer the cookie. All you have to do is set `DISPLAY`, and point `XAUTHORITY` to `~serveruser/.Xauthority`. So you can do:

```
su - -c "exec env DISPLAY='$DISPLAY' \
                  XAUTHORITY='${XAUTHORITY-$HOME/.Xauthority}' \
                  command"
```

Putting it into a script would give something like:

```
#!/bin/sh
if [ $# -lt 1 ]
then echo "usage: `basename $0` command" >&2
     exit 2
fi
su - -c "exec env DISPLAY='$DISPLAY' \
                  XAUTHORITY='${XAUTHORITY-$HOME/.Xauthority}' \
                  "'"$SHELL"'" -c '$*'"
```

Call the script `/usr/local/bin/xroot`, and you can do:

```
xroot 'control-panel &'
```

Even easier, no?

8.Running a Remote Window Manager

A window manager (like `twm`, `wmaker`, or `fvwm95`) is an application like any other. The normal procedure should work.

Well, almost. At most one window manager can be running on a display at any time. If you are already running a local window manager, you cannot start the remote one (it will complain and exit). You have to kill (or simply quit) the local one first.

Unfortunately, many X session scripts end with an

```
exec window-manager-of-choice
```

and this means that when the (local) window manager exits, your session exits, and the X system (xdm or xinit) considers your session over and effectively logs you out.

You have to jump through a few extra hoops, but it can be done and it's not too difficult. Just play with your session script (normally `~/.xsession` or `~/.xinitrc`) to get it as you want it.

Beware that a window manager often provides ways to run new programs, and that these will run on the local machine. That is, local to where the window manager runs. If you run a remote window manager, it will spawn remote applications, and this may not be what you want. Of course, they still display on the display that is local to you.

9.Troubleshooting

The first time you try to run a remote X application, it usually does not work. Here are a few common error messages, their probable causes, and solutions to help you on your way.

```
xterm Xt error: Can't open display:
```

There is no `DISPLAY` variable in the environment, and you didn't tell the application with the `-display` flag either. The application assumes the empty string, but that is syntactically invalid. To solve

this, be sure that you set the `DISPLAY` variable correctly in the environment (with `setenv` or `export` depending on your shell).

```
_X11TransSocketINETConnect: Can't connect: errno = 101
xterm Xt error: Can't open display: love.dial.xs4all.nl:0
```

Errno 101 is ``Network is unreachable''. The application could not make a network connection to the server. Check that you have the correct `DISPLAY` set, and that the server machine is reachable from your client (it should be, after all you're probably logged in to the server and telnetting to the client).

```
_X11TransSocketINETConnect: Can't connect: errno = 111
xterm Xt error: Can't open display: love.dial.xs4all.nl:0
```

Errno 111 is ``Connection refused''. The server machine you're trying to connect to is reachable, but the indicated server does not exist there. Check that you are using the right host name and the right display number.

```
Xlib: connection to ":0.0" refused by server
Xlib: Client is not authorized to connect to Server
xterm Xt error: Can't open display: love.dial.xs4all.nl:0.0
```

The client could make a connection to the server, but the server does not allow the client to use it (not authorized). Make sure that you have transported the correct magic cookie to the client, and that it has not expired (the server uses a new cookie when a new session starts).