SSH Users beware: The hazards of X11 forwarding

By Brian Hatch.

Summary: *Logging into another machine can compromise your desktop...*

The last two articles have discussed the security model of X11, the guts behind Linux window managers and all things graphical.[1] Essentially, if you can contact the X11 server process, you can do anything you want to it, such as sniffing all keystrokes, dumping or manipulating windows, etc.

In order to access the server, you must have two things:

1. The MIT Magic Cookie that the server requires, if any. (Most distros set up X11 to require these, which is good.)
2. Access to the X11 server's socket, be it a network TCP socket or a unix domain socket.

In my previous examples, I showed you how you can satisfy these requirements by being root on the machine on which the X11 server is running. I got lots of hate mail because of it, with arguments like the following:

> "But if they already have root, the game is lost!" "I don't let anyone on my machine, so it's a moot point!" "I don't have sshd running, so how could they get in anyway?"

These are all valid (and anticipated) statements. Here's where I get to say *"Trust me, I was getting somewhere important..."*

Enter SSH, a wonderful encrypted remote login/file transfer/port forwarding/you name it protocol. You probably use it when you log into to other Linux machines, such as your shell server, email account, etc.[2]

SSH has the ability to tunnel X11 connections through it - this feature is called X11 Forwarding. In brief, if you are on your desktop attached to an X11 display (you can run `xclock for example) then when`

```
you SSH to a different machine, it can tunnel X11
over the connection. You can run graphical X11
applications on the remote machine, but they display
back on your desktop.
```

Here's the nitty gritty: upon logging into the remote system, the ssh server process binds a TCP port (let's say 6010), creates you an MIT Magic Cookie on the server by running `xauth`, and then sets the $DISPLAY environment variable to point to it's port (for example `$DISPLAY=localhost:10.0` [3]) When you run an X11 application, it reads the `$DISPLAY` variable, connects to the X11 server (in this case the `sshd` process on the remote system) and provides the magic cookie (by reading `~/.Xauthority`). `sshd` verifies the cookie, and passes he data back to the `ssh` process on your desktop over the encrypted link. `ssh` on your desktop then forwards the data to the actual X11 server on your desktop, using the desktop's cookie.

Now all of this happens behind the scenes -- all you notice is that you log into the remote machine, and when you run an X11 application, the window appears on your desktop. This is cool, this is great, this is secure - encrypted from end to end.[4]

But even though the X11 application is secure, you've opened up a new vulnerability. If someone on the server can read your `~/.Xauthority` file (hopefully only root, but if you have bad file permissions you're in trouble), and can connect to the port that `sshd` has bound (which anyone can) then they can access your desktop's X11 server, even if they're not anywhere near you!

Let's reiterate: if you log in via SSH to a remote server with X11 forwarding, root on that server can access your desktop, sniff your keystrokes, abuse your windows, you name it. If you have bad permissions on your `~/.Xauthority` file, then **anyone** on that server can control your desktop.

OpenSSH used to have X11 forwarding enabled by default, but luckily newer versions have luckily changed this. Unfortunately, some Linux distributions still enable it by default in the global `/etc/ssh/ssh_config` file.[5] This means that any time you SSH to another machine, that machine's administrators could attack you. Not good, definitely not good.

Now is this something that occurs in the real world? Heck yes -- I've seen more than one free shell account provider with unethical administrators who used this feature to snoop passwords and other information addresses. Again, you may point out that they can already gather any of this data sent to the machine you've logged into. But the fact they can access keystrokes that are never going to their server at all is a very different and worrisome situation.

So, when should you enable X11 forwarding? Only when you really

really need to, and only to machines which you trust. In addition, if you must perform actions outside the X11 application (for example opening up a different terminal and logging in somewhere) you can enable the 'secure keyboard' feature of some programs (for example hitting 'ctrl right-button' in `xterm` and selecting the first option) to keep your keystrokes from being available to anything but that one window. But a malicious user could still perform all of the other tricks discussed last time, such as getting screenshots of those secure windows.

It's best to enable X11 forwarding manually, ssh to the other system, run your X11 application, and log out as soon as possible.

To turn off X11 forwarding by default, add the following to the bottom of your `~/.ssh/config` file, or the global `/etc/ssh/ssh_config` file:

```
Host *
ForwardX11 no
ForwardAgent no
```

(Note: the last line also disables the SSH Agent forwarding - you can probably guess why that's a bad idea at this point.)

If you need to have X11 forwarding for a connection, run `ssh` with the `-X` flag, for example:

```
$ ssh -X server /usr/bin/display filename.jpg
```

Following this method, you'll never accidentally log in with SSH X11 Forwarding enabled.

SSH X11 Forwarding is a wonderful thing when you need it - it's much better than sending your connections back to your desktop in the clear - but you need to understand that you open your entire environment up to any attacker on the server. Use it wisely and sparingly.

NOTES:

[1] Ok, tis true, there are some things that let you have graphics even in plain text TTYs, such as `w3m`, the greatest text based web browser in the world. If you never go into X11, you can stop reading this article now.

[2] From my desktop alone, I have 45 outbound SSH connections at the time I write this. Probably half of those are to bounce through firewalls and are running multiple SSH sessions via screen. Thank goodness for SSH -- I don't know how all those point-and-click users administer their machines.

[3] Why is it `localhost:10.0` instead of `localhost:6010`? Normally, the first X11 display is on port 6000, the next on port 6001,

which get abbreviated as `:0`, `:1` and so on. SSH binds higher than the number of actual physical displays that are expected (very few desktops run more than one X11 display, much less nine of them) which is why it starts at 6010 and works it's way up.

[4] Those who try this over anything but a LAN connection will also note that this is **slow**... X11 can use a lot of bandwidth.

[5] Just to be more confusing, some disable X11 forwarding on the server by default, which means the user has no ability to use it even if they want to, even though this could only be used to attack the user, not the server. Very weird -- I don't grasp the logic here.

Brian Hatch is Chief Hacker at Onsight, Inc and author of ***Hacking Linux Exposed*** and Building Linux VPNs. He looks back on his college days of playing xtank at 3am and wonders "Did anyone steal my passwords when we all ran 'xhost +' " ? Brian can be reached at brian@hackinglinuxexposed.com.
