UCRL-MA-121788

Department of Energy

CIAC

Computer Incident Advisory Capability

Securing X Windows

CIAC-2316 R.0

John Fisher

August, 1995

Lawrence Livermore National Laboratory

CIAC is the U.S. Department of Energy's Computer Incident Advisory Capability. Established in 1989, shortly after the Internet Worm, CIAC provides various computer security services free of charge to employees and contractors of the DOE, such as:

• Incident Handling consulting
• Computer Security Information
• On-site Workshops
• White-hat Audits

CIAC is located at Lawrence Livermore National Laboratory and is a part of its Computer Security Technology Center. CIAC is also a founding member of FIRST, the Forum of Incident Response and Security Teams, a global organization established to foster cooperation and coordination among computer security teams worldwide.

Work performed under the auspices of the U. S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

Table of Contents

Securing X Windows

Introduction

X Windows enjoys great popularity with users, in a variety of environments. Its client/server model of application management allows for powerful, flexible interaction between users and computers. Unfortunately, this power comes at the cost of security. X Windows, if not managed properly, can create a serious vulnerability. This paper explores many of the security problems and solutions in X Windows.

How X Windows Works

It may seem strange that a graphical user interface can be a potentially serious security vulnerability. So for starters, let’s take a look at how X Windows works, and how it can be a problem.

X Windows is really, at its lowest level, a communication protocol, called sensibly enough, *X Protocol.* This protocol is used within a single computer, or across a network of computers. It is not tied to the operating system and is available on a wide range of platforms. X Windows utilizes a *Client-Server* model of network communication. This model allows a user to run a program in one location, but control it from a different location.

Counter to common client-server convention, the user actually works directly on the *X server*, which offers a screen, a keyboard, and a mouse. It’s referred to as the server because it generates the inputs for and manages the outputs from the clients. The *X clients* are applications, such as *xterm*, *emacs*, or *xclock*. They receive and process inputs and return outputs.

In most cases, the server and the clients are running on the same computer (host). But, X Protocol is flexible and makes many different configurations possible. In fact, an *X terminal* is a screen, keyboard and mouse which has *no* computing capability. The only thing it can do is process X Protocol messages, which come from clients running on other systems. Even if the server is running on a host, it may be desirable for a client to run on a remote host, even if it is located in another building or even another state.

So, what does this have to do with computer security? The clients that are able to run on a server should be carefully controlled. Since multiple clients are running on the same server, careful control of their inter-communication should be observed. If one client is able to send information to another client, or one client is able to capture information meant for another client, the system may be vulnerable.

Unprotected X Windows

Some examples of communication between the X server and an X client include the following:

- X terminal modification - font management, mouse sensitivity, color mapping, keyboard mapping.
- An X Event - keyboard, mouse, etc.
- X data - modification to the X terminal screen, such as writing text, creating a window, or drawing an image.

Any client that can access a server can potentially access and change any X communications that take place on it. This could include the following:

- Modifying session parameters.
- Create/destroy windows - Was that document saved before the window mysteriously disappeared?
- Capture X events - For example, reading keystrokes on an xterm window, which include a login and password.
- Create X events - For example, sending keystroke sequences to an emacs window, or an xterm window, to execute a command.

Clearly, X servers are inherently dangerous. What’s worse, many servers ship with world accessability as the default setting.

Approaches To Security

What is the best way to secure an X server? Two different approaches are available: host authentication and token authentication. Each are discussed below.

Host Authentication

Host authentication is the potential acceptance of a connection based on its origin. Typically, this would be determined by the IP address of the connection's host. Once a user has logged in to an X Server, the server is potentially open to connections from any host. A program called *xhost* is available to control on a host-by-host level which hosts can display clients on the X Server. But, most hosts support multiple users, and it is impossible to specify which users on a particular host have access.

Token Authentication

The second form of authentication is to verify each client based on the token they offer. Using a program called *xauth*, each client is given a "magic cookie," a random value which it must offer to the X Server to be allowed access.

Host Authentication

Certainly the most widely used mechanism for X security is the *xhost* program. While simple to use, *xhost* is rather inflexible.

Using xhost

Using the *xhost* program is straightforward. Each X server maintains a list of hosts which may or may not access it. The *xhost* program is used for modifying that list. The command line syntax is as follows:

· Display a list of hosts allowed to access this X Server:

```
xhost
```

· To add a host, say bar.foo.org, one would type:

```
xhost +bar.foo.org
```

Then, *any* user and program on that machine may communicate with your X server.

· To remove that same host, type:

```
xhost -bar.foo.org
```

· An X server may be opened to the world by disabling access control:

```
xhost +
```

· Access control may be re-enabled (i.e., the current list of hosts is again active) by:

```
xhost -
```

With no parameters, *xhost* returns whether or not access control is currently turned on and which machines are allowed access. This is the *only* way that *xhost* can be run remotely, even if the remote machine is on the access list. When *xhost* is utilized, a user from an unauthorized host attempting to connect will be presented with the following response:

```
Xlib: connection to “display:0.0” refused by server
Xlib: Client is not authorized to connect to Server
```

Note that disabling a host’s access after a connection has been made will have no effect on existing connections. The server must be reset in order to break established connections.

This however, is actually a feature. A smart way to use xhost is to only turn on a host’s access for the period it takes to start a client on that host. Then, access can be disabled. The client will continue to run, but the host’s access will again be disabled.

Benefits:

The *xhost* access control mechanism is certainly easy to use. A single program with a simple syntax is required.

Drawbacks:

The simplicity of *xhost* is both a benefit and a drawback. All connections from a host must be accepted or rejected—not on a user-by-user, program-by-program, or connection-by-connection basis. For many environments, where numerous users are allowed access to a particular host, this is an insufficient solution. And certainly, most computers running X servers have multiple user accounts, and any user that can log in to the computer can access the X server, as the localhost, completely bypassing the *xhost* access control.

Unfortunately, many X servers, such as NCD servers, SGI systems, and Mac X for the Macintosh come with access control disabled by default. For users unfamiliar with the vulnerability of X servers, this can create a real security problem.

Xhost has higher priority than token authentication. Any user can add systems to the *xhost* access list without special privileges or assistance from the system administrator.

Token Authentication

The X server can control a user's access to an X server through the use of a *magic cookie*. This is essentially a machine-readable, randomly generated access code. Each X client must provide the same magic cookie value to the server before being allowed access. This value is stored in the file .Xauthority. It can be either created by the X Display Manager, or by the user, at the beginning of each session.

For the user who is only logged on to one machine, the enhanced security is present but transparent. Each new client executed by that user on that machine will find the magic cookie and start without complaint. But, many users work on multiple machines at once. How would an X client on a remote machine know what the magic cookie is? This is where the *xauth* program comes in.

xauth Program

The xauth program is used for editing and displaying the user's magic cookie authorization information. Once the magic cookie is displayed in a human-readable form, it can be sent to a remote host. On that remote host, *xauth* is used again to merge the magic cookie into the user's .Xauthority file. Assuming a .rhosts file is set up for the user, pushing the authorization information to a remote host (let's say ahost.foo.org) can be done with one command:

```
xauth extract - $DISPLAY | rsh ahost.foo.org xauth
merge -
```

The first command prints the magic cookie for the current host ($DISPLAY) to the standard output (the dash). This information is then piped to the remote shell command, which runs the *xauth* program on the machine ahost.foo.org. The magic cookie is then read from the standard input (again, the dash), and merged into the .Xauthority file. The result is that the user who executed this command can now run X clients on ahost.foo.org, and have them displayed on the X Server. ***It is important to have the permissions set correctly for the .Xauthority file.*** It should be readable/writable by the owner only (that is, set to "-rw-------"). **Furthermore, beware of NFS exporting a home directory, even read-only!** It may be mounted, allowing the .Xauthority file to be read.

Note the key improvement here. ***The user who ran this command is now the only user on ahost.foo.org who can connect an X client to their X server.*** All other users on ahost.foo.org are still blocked out of this X session.

X Display Manager

The X Display Manager, *xdm*, is a client which provides login screens for multiple X Servers. When a user logs in through the X Display Manager, xdm writes a magic cookie to the user's home directory, in the file .Xauthority. X servers are not always stand-alone computers. They can be X terminals as well, whose sole function is to run clients from other systems. These types of machines require a *xdm* to provide the initial login screen. Stand-alone computers may utilize *xdm* as well. In addition to providing a more user friendly login sequence, *xdm* provides support for magic cookie authentication.
This authentication must first be turned on by the following X resource entry in the file /usr/lib/X11/xdm/xdm-config:

```
DisplayManager*authorize: true
```

With this, *xdm* will generate a new magic cookie value each time a user logs in, and store that value in their .Xauthority file.

If *xdm* is not being used, it is still possible to use this type of authentication; this will be explained below.

Generating a Magic Cookie Without Xdm

Xdm will manage your .Xauthority file for you, but if *xdm* is not used, it is still possible to have magic cookie authentication. The only problem is that on many X11 servers, the user needs to generate the magic key value (OpenWindows is one exception—it will generate a magic cookie when started). This can be done in a variety of ways. For example, if Korn shell is being used, it has a built-in random number generator:

```
randomkey=`ksh -c ‘echo $(( $RANDOM * $RANDOM * 2 ))’`
xauth add ${HOST}:0.$randomkey
```

If *ksh* is not being used, the clock may be used to obtain a “random key”:

```
randomkey=`date +”%y%m%d%H%M%S”`
xauth add ${HOST}:0 . $randomkey
```

Xrsh in X11R5

Xrsh is a script provided in X11R5, in the contrib/clients/xrsh directory. For those users who run clients remotely via *rsh*, this can be a handy script. It utilizes xauth to automatically copy the magic cookie code to the remote machine before running the remote client. For example, to run an *xterm* window on the host foo, type:

```
xrsh -auth xauth foo xterm
```

Benefits: Authorization is now done on a user-by-user basis, not a host-by-host basis. In an environment where one host supports a large number of users, this can be very important.

Drawbacks: The xdm and xauth programs are time consuming for both the administrator and the end user to use and maintain. They require a good understanding of the X client-server model on the part of the user.

Note that magic cookie authorization should be used in addition to xhost security. In fact, "xhost-" should be used to disable all host-based access.

Xterm Vulnerabilities

The *xterm* program is used to provide the user with a command line prompt (a shell in Unix). Because a great deal of critical user/computer interaction takes place through a command line prompt, it is important to be able to execute this program safely. The *xterm* program has several security vulnerabilities worth mentioning.

One write-access “feature” provided by *xterm* should NOT be utilized. SendEvents are key and button events that have been generated artificially (i.e., not by a keyboard or a mouse). By default, *xterm* refuses all SendEvent requests from the X server. This can be over-ridden, however, in two ways. The first way is by adding an X resource definition to either the .Xdefaults file or the app-defaults/Xterm file:

```
xterm*allowSendEvents: True
```

The second way of allowing the X Server to send X events is through the *xterm* Main Options menu (accessed by holding down the CTRL key while pressing the left mouse button). NEITHER of these should ever be done, as they open the *xterm* to communication from sources other than the user who initiated it.

Read access is controlled through a different mechanism, however. On the Main Options menu is a “Secure Keyboard” option. When turned on, ALL keyboard events are sent exclusively to the *xterm* window (mouse interaction is not modified). This prevents other clients from capturing critical keyboard events, such as entering a password. Of course, only one X client at a time may have this option turned on. This option is useful for critical data entry, but is really impractical for continuous use because it must be turned off to interact with any other windows.

X Windows Related Security Information

The following CIAC Bulletins have been released, which relate directly to X Windows:

- **E-04:** Versions of X11R5's Xterm that do not have Patch Level 26 contain a security vulnerability in its logging capability. This vulnerability allows an unauthorized user to obtain root access. If the xterm program has the "s" (setuid) permission bit set, and the command "xterm -l" creates a file "XtermLog.axxxx", then the vulnerability may exist.
- **B-11**: A security vulnerability on Sun computers running OpenWindows 2.0 (now a very old version) allows theft of critical files. This explains how to remove this.
- **E-05**, **D-11**: These are announcements for the release of SunOS and Solaris patches. Two of these patches repair security holes in their OpenWindows system. The loadmodule patch is particularly important, as it fixes a widely exploited bug.
- **F-19**: The well known vulnerability probe program known as SATAN looks for X Windows vulnerabilities. This bulletin provides Hewlett-Packard HP-UX related information for protecting the HP-UX X Windows server from SATAN probes.
- **F-21**: This bulletin is similar to F-19, but provides SUN OS specific information.
- **F-23**: This bulletin is similar to F-19, but provides IBM AIX specific information.
- **F-24**: This bulletin is similar to F-19, but provides SGI IRIX specific information.

The following CIAC Notes relate directly to X Windows:

- **Notes 7**: A lengthy article detailing SATAN and its probes covers X-Windows specific information.
- All of the above Bulletins and Notes are available on the CIAC Web server, at http://ciac.llnl.gov.

Appendix A: Contacting CIAC

Contacting CIAC

Phone	(510) 422-8193
Fax	(510) 423-8002
STU-III	(510) 423-2604
Electronic mail	ciac@llnl.gov
Emergency SKYPAGE	800-SKYPAGE pin# 855-0070
Anonymous FTP server	ciac.llnl.gov (IP 128.115.19.53)
BBS	(510) 423-3331 (9600 Baud) (510) 423-4753 (2400 Baud)

Reader Comments

CIAC updates and enhances the documentation it produces. If you find errors in or have suggestions to improve this document, please fill out this form. Mail it to CIAC, Lawrence Livermore National Laboratory, P.O. Box 808, Mail Stop L-303, Livermore, CA, 94551-9900. Thank you.

List errors you find here. Please include page numbers.

List suggestions for improvement here.

Optional:

Name ______________________________ Phone ______________