

# Enhancing XFree86 Security

Matthieu Herrb

July 2003



# Introduction

# What is XFree86 ?

---

- a Free implementation of the X Window System, based on X.Org's sample implementation (X11R6.6).
- History: X386 (*Three Eight Six*) → XFree86 (*Free Eight Six*)
- Runs on: Unix<sup>TM</sup>, SVR4, \*BSD, Linux, Windows (Cygwin), Mac OS X, etc.
- Foundation for most modern GUIs and Desktops on Linux/Unix-like systems.

# Security concerns - summary

---

Client side:

- Setuid clients
- Scripting toolkits

Networking:

- connection setup / connection sniffing

Server side:

- Server is setuid root

Various issues: xdm, xfs, proxies

# Chapter 1

# Client side security

# Which problems ?

---

Traditional application level security: handling (and limiting) privileges

Some remaining issues:

- xterm (on systems where pty allocation needs root)
- xlock like applications
- ICCM and other scriptable X Toolkits and applications → out of the scope of this talk.

# Chapter 2

## Networking issues

# Security at the transport level

---

Access control done mainly at the transport level:

if a client is able to get a connection to the X server, it can take full control of it:

- display windows
- get events
- send synthetic events to other windows
- change X resource database
- access server-side data (atoms, pixmaps, etc)
- tweak video modes



# X11 Transport Security models

---

Transport types:

- TCP
- local (Unix domain sockets + SVR4 local) connections
- SYSV SHM

Authentication types:

- Host based - `xhost`
- Magic cookies - `xauth`
- XDM
- Kerberos, SUN-DES-1 (not widely used AFAIK)

Too many people still use `xhost` + . . .

# Host-based access control

---

The server has a list of authorized hosts (really network addresses).

When a client tries to connect, its address is compared to the list.

If found → access granted.

## Problems:

- any user on authorized hosts can connect
- some difficulties with multi-homed hosts

# Magic cookie protocol

---

## MIT-MAGIC-COOKIE-1

Cookie: a 128 bits random number

- generated at startup time
- stored in a file designated by the `XAUTHORITY` environment variable
- manipulated (exported/imported) by the `xauth` command.

The client sends the magic cookie to the X server in the initial connection request.

The server checks if the cookies matches. If yes, connection accepted

Access control often equivalent to Unix file system access rights to `.Xauthority`.

The cookie is sent in clear text over the network.

# The XDM-AUTHORIZATION-1 protocol

---

Based on DES encryption - not exportable outside the US.

- create a 192 bit value with network address, port number, current time
- crypt it using a DES key
- send it to the server
- the server decrypts the packet, verify that the data is valid to allow access

The DES key is the shared secret.

Somewhat better than magic cookies, but not widely used.

# Other access control protocols

---

## **SUN-DES-1**

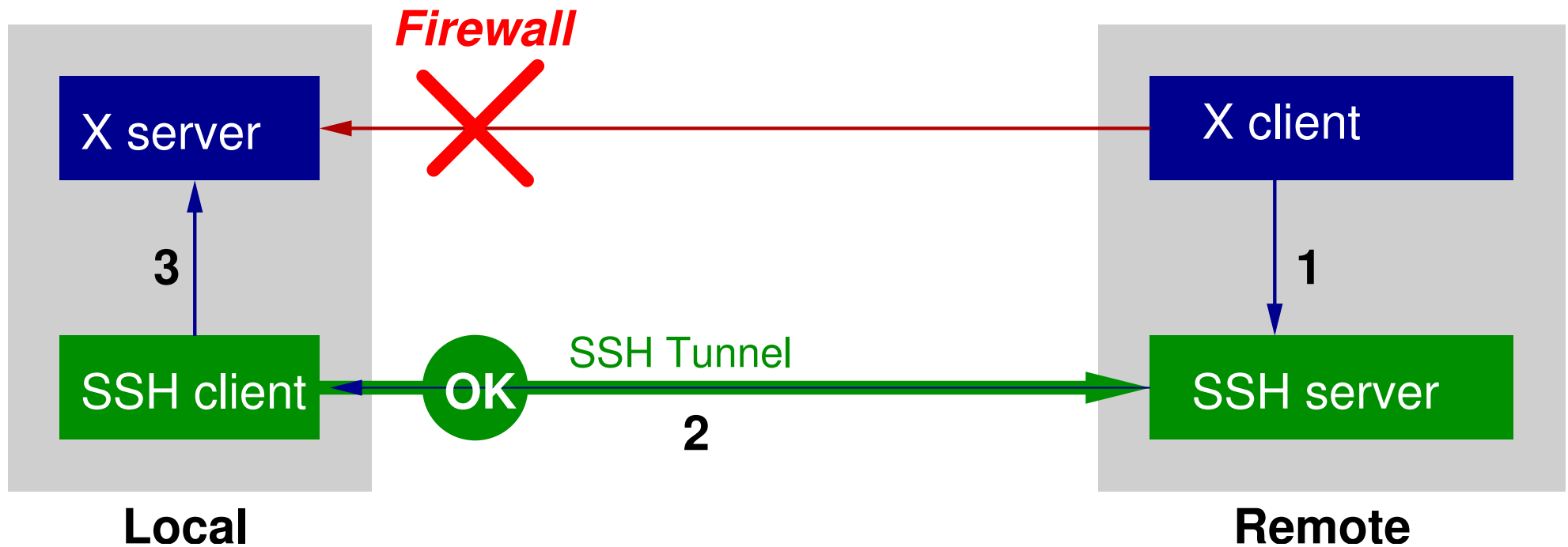
Based on secure-RPC credentials. A valid secure-RPC credential is needed to connect to the X server.

xhost is used to add authorized principals  
(example: `xhost +nis:unix.3715@xfree86.org`)

## **MIT-KERBEROS-1**

Based on kerberos tickets.

# Secure Shell tunnels



**Note:** the cookie is stored on the remote system  
→ one needs to trust the remote admin  
(he can gain access to the local X server).

# The X Security extension

---

Aimed to control access from clients to X server resources.

Policy based (`/etc/X11/xserver/SecurityPolicy`).

Almost ignored in practice.

# Chapter 3

## Server side security



# Threats

---

The X server is traditionally run as root.

Lots of non-trusted data are handled by the X server:

- X requests
- Fonts
- Driver modules (but only root can install them)
- Configuration file (but only root can set the pathname)

In addition to spawning a root shell one could potentially:

- decrement system security level
- load a user-land driver for a device that's not configured
- Brutal DoS: toast the machine (stop fans, re-flash BIOS, etc...)

# Unix domain sockets

---

Created in `/tmp/.X11-unix/`

Several processes owned by different users need to be able to create sockets here.

Difficult to handle socket creation correctly

**Solution:** (ab)use of server privileges to create the sockets.

System	X server implementation	privilege used
Solaris	Xsun	setgid root
Linux, *BSD, etc.	XFree86	setuid root

Causes problems to 3rd party servers (ICE, xfs, etc. ) that need to be installed with the same privileges.

One solution: only use TCP connections (example: SSH X11 forwarding).

## MIT-SHM extension - CAN-2002-0164

---

An extension designed to share pixmaps between a client and the X server for faster image display.

**Flaw:** The X server needs to be able to read the shared memory segment (and sometimes to write it). The client needs to create it mode 644 (or even 777).

→ the data is exposed to the world.

**Solution:** this extension needs to be redesigned from scratch.

# XFree86 internals

---

XFree86 needs direct access to the graphics card hardware (DMA, registers, etc.) for:

- legacy ISA (+ VLB) device probes
- hardware accelerated 2D rendering

On most systems this requires one or several of these:

- **root privileges**
- low kernel security level
- a special `ioperm` syscall

# Kernel security levels

---

An \*BSD feature.

if `securelevel = 1` some operations are prohibited:

- lowering `securelevel`
- accessing physical memory through `/dev/mem`
- other file system operations, out of scope here

XFree86 needs `/dev/mem` access on i386 and alpha → cannot run with `securelevel=1`

→ The “aperture” driver is a special `/dev/mem` driver that restricts accesses to the memory above the physical memory, and to the BIOS and interrupt vectors.

PCI only architectures can provide a better access control through a `vga_pci` driver.

# Lowering the privileges

---

Most of the operations that need root privileges are done during startup.

- move all privileged operation as early as possible in the startup sequence
- switch to a non privileged id once done

The XFree86 version shipped with OpenBSD 3.2 was using this technique.

- Problems:
- reopening the mouse device after a VT switch
  - correct synchronization for startx

# Privilege separation

---

Technique developed by Niels Provos for the SSH server.

Can be used in the X server for privileged operations that need to be done any time (VT switches)

While the main process is running privileged, fork a child that will keep the privileges.

Then change uid in the main process (lower privileges).

Use a pipe to send requests / get data back from the privileged child.

The child makes strict controls on what it accepts to do.

The child's code is kept small and simple - easy auditing

# Privilege separation for XFree86

---

As shown above, most privileged operations can be done early

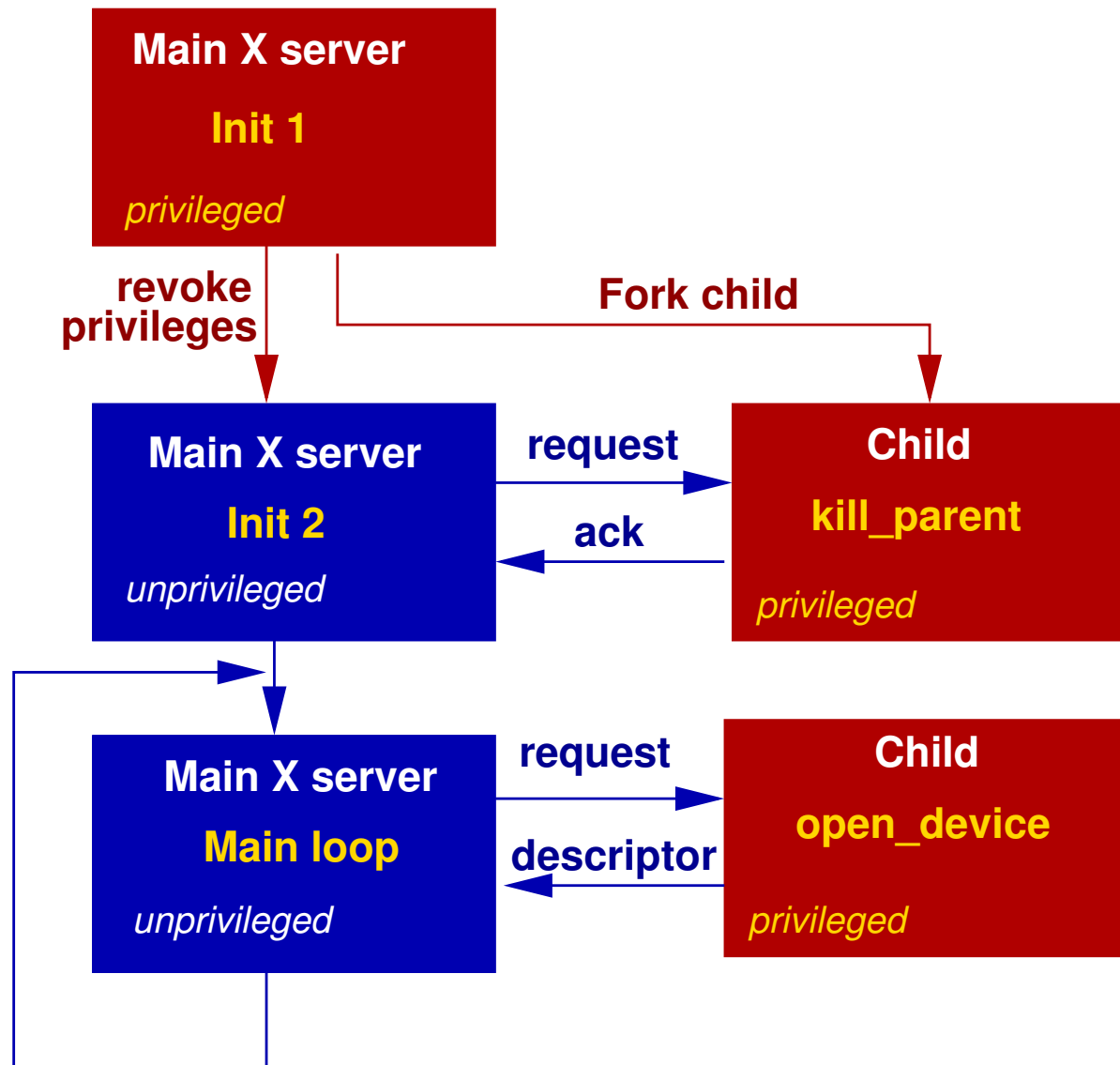
Two remaining operations:

- (re)opening input device after a VT switch
- sending a signal to the process that started the X server to notify that it's ready to accept connections.

Shipped in OpenBSD 3.3, being ported to XFree86-current on systems that provide file-descriptor passing.



# Privilege separation for XFree86



# Using systrace to control XFree86 privileges

---

systrace is another tool by Niels Provos.

Allows to define a policy that allows/denies system calls

Makes it possible to do controlled privilege elevation.

Available on Linux, NetBSD, OpenBSD, Mac OS X.

Used with a standard XFree86 server

Much finer grain.

Still some issues.

# An experimental systrace policy for the X server

---

```
root# systrace -a -c 35:35 XFree86
```

(Only privilege elevation lines are shown)

```
Policy: /usr/X11R6/bin/XFree86, Emulation: native
  native-rename: filename eq "/var/log/XFree86.0.log" and filename[1] \
    eq "/var/log/XFree86.0.log.old" then permit as root
  native-fswrite: filename eq "/var/log/XFree86.0.log" then permit as root
  native-fswrite: filename eq "/tmp/.tX0-lock" then permit as root
  native-fsread: filename eq "/tmp/.X0-lock" then permit as root
  native-fswrite: filename eq "/tmp/.X0-lock" then permit as root
  native-open: filename match "/dev/tty*" then permit as root
  native-fswrite: filename eq "/dev/mem" then permit as root
  native-fswrite: filename eq "/dev/xf86" then permit as root
  native-sysarch: true then permit as root
  native-kill: true then permit as root
  native-open: filename match "/dev/*mouse*" then permit as root
```

# XFree86 policy - other rules

---

```
native-fsread: filename eq "/var/run/ld.so.hints" then permit
native-fsread: filename match "/usr/lib" then permit
native-fsread: filename match "/usr/lib/lib*.so.*" then permit
native-fsread: filename eq "/usr/X11R6/lib" then permit
native-fsread: filename match "/usr/X11R6/lib/lib*.so.*" then permit
native-fsread: filename eq "/etc/X11/XF86Config" then permit
native-fsread: filename eq "/etc/X11/XF86Config-4" then permit
native-fsread: filename eq "/etc/malloc.conf" then permit
native-link: filename eq "/tmp/.tX0-lock" and filename[1] eq "/tmp/.X0-lock" then permit
native-fsread: filename eq "/var/log/XFree86.0.log" then permit
native-fswrite: filename eq "/tmp/.X11-unix" then permit
native-fsread: filename eq "/tmp/.X11-unix" then permit
native-fswrite: filename eq "/tmp/.X11-unix/X0" then permit
native-bind: sockaddr eq "/tmp/.X11-unix/X0" then permit
native-fsread: filename eq "/etc/X0.hosts" then permit
native-fsread: filename match "/usr/share/zoneinfo/*" then permit
native-fsread: filename match "/usr/X11R6/lib/X11/fonts/*" then permit
native-fsread: filename match "/usr/local/lib/X11/fonts/*" then permit
native-fsread: filename match "/usr/X11R6/lib/X11/locale/*" then permit
native-fswrite: filename eq "/dev/tty" then permit
native-fswrite: filename match "/dev/ttyC*" then permit
native-fsread: filename eq "/dev/apm" then permit
native-fsread: filename eq "/usr/X11R6/lib/X11/rgb.txt" then permit
native-fsread: filename eq "/usr/X11R6/lib/X11/XKeysymDB" then permit
native-fsread: filename match "/usr/X11R6/lib/modules" then permit
native-fsread: filename match "/usr/X11R6/lib/modules/*" then permit
native-fsread: filename eq "/etc/X11/xserver/SecurityPolicy" then permit
```

```
native-fsread: filename eq "/etc/X11/xkb/X0-config.keyboard" then permit
native-fsread: filename match "/etc/X11/xkb/*" then permit
native-fsread: filename eq "/var/tmp/server-0.xkm" then permit
native-fswrite: filename eq "/var/tmp/server-0.xkm" then permit
native-fsread: filename match "<non-existent filename>: *" then permit
native-fsread: filename eq "/usr/share/nls/C/libc.cat" then
native-chdir: filename eq "/etc/X11/xkb" then permit permit
```

```
native-issetugid: permit
native-mprotect: permit
native-mmap: permit
native-__sysctl: permit
native-fstat: permit
native-close: permit
native-read: permit
native-mquery: permit
native-sigprocmask: permit
native-break: permit
native-geteuid: permit
native-getuid: permit
native-getrlimit: permit
native-write: permit
native-getpgrp: permit
native-getpid: permit
native-sigaction: permit
native-setitimer: permit
native-umask: permit
native-getppid: permit
native-gettimeofday: permit
native-setpgid: permit
native-kqueue: permit
native-kevent: permit
```

```
native-munmap: permit
native-nanosleep: permit
native-shmget: permit
native-shmctl: permit
native-pipe: permit
native-fork: permit
native-wait4: permit
native-getgid: permit
native-setgid: gid eq "35" then permit
native-setuid: uid eq "35" and uname eq "_x11" then permit
native-dup2: permit
native-execve: permit
native-select: permit
native-sigreturn: permit
native-exit: permit
native-ioctl: permit
native-fcntl: permit
native-fstatfs: permit
native-getdirentries: permit
native-lseek: permit
native-sigsuspend: permit
native-getrusage: permit
native-fchmod: fd eq "0" and mode eq "444" then permit
native-socket: sockdom eq "AF_INET" and socktype eq "SOCK_STREAM" then permit
native-socket: sockdom eq "AF_UNIX" and socktype eq "SOCK_STREAM" then permit
native-setsockopt: permit
native-bind: sockaddr eq "inet-[0.0.0.0]:6000" then permit
native-listen: permit
native-getsockname: permit
```

# Chapter 4

## Miscellaneous issues

# The font server

---

Used to be run as root.

Need to trust font data.

TrueType font renderers have a byte-code interpreter. Is it safe?



# XDMCP

---

The protocol used to manage X terminals

Issues too

# Write xor eXecute

---

Technique developed in OpenBSD and other systems:  
one page is either writeable or executable but not both at the same time.

Makes exploit almost impossible to write

The XFree86 module loader is not compatible with W^X on i386. :-)

# Thanks

---

**Theo de Raadt** has suggested some of the idea that were implemented to lower the X privileges.

**Niels Provos and Marc Espie** have provided some of the code that is used in the X privilege separation code.

**Todd Fries** and many OpenBSD developers for testing and finding bugs in the code.

# Bibliography

---

1. Unix manual page, section 7. *Xsecurity - X display access control*.
2. L. Mui and E. Pearce. *X Window System Administrator's Guide*, volume 8 of *The Definitive Guides to the X Window System*. O'Reilly, October 1992.
3. D. P. Wiggins. Security extension specification. Technical Report Version 7.1, X Consortium, Inc., November 1996.
4. D. J. Barret and R. Silverman. *SSH, The Secure Shell: The Definitive Guide*. O'Reilly, January 2001.
5. Niels Provos. Preventing privilege escalation. In *12th USENIX Security Symposium*, Washington, DC, August 2003. <http://www.citi.umich.edu/techreports/reports/citi-tr-02-2.pdf>.
6. Niels Provos. Improving host security with system call policies. In *12th USENIX Security Symposium*, Washington, DC, August 2003. <http://www.citi.umich.edu/techreports/reports/citi-tr-02-3.pdf>.

## Conclusion

---

- Some issues with XFree86 security have been shown
- Solutions exist
- More work is needed
- Don't be afraid to run X on your machine

# Questions ?