

X-Window und Systemsicherheit ?

- [Host-Based-Control \(xhost\)](#)
 - [\(User-\)/Permission-based-Control \(MIT-MAGIC-COOKIE-1\)](#)
 - [Generierung eines Schlüssels/Cookies](#)
 - [MAGIC-COOKIE-Schlüssel auf andere Rechner übertragen](#)
 - [XDM-AUTHORIZATION-1](#)
 - [SUN-DES-1](#)
 - [Vorbereitung zur Verwendung von SUN-DES-1](#)
 - [Verwendung von SUN-DES-1 mit xdm](#)
 - [Verwendung von SUN-DES-1 mit xinit](#)
 - [Bewertung von SUN-DES-1 in Bezug auf die Sicherheit](#)
 - [Allgemeine Sicherheitsprobleme bei der Verwendung von X](#)
 - [Die Console und xterm](#)
 - [Aufhängen des Servers \(R3\)](#)
 - [Framebuffer \(Sun Workstations\)](#)
 - [Files in /tmp](#)
 - [Falsch gesetzte Display-Variablen](#)
 - [Das Netzwerk-Design von X11](#)
 - [Literaturverzeichnis](#)
-

X-Window und Systemsicherheit?

Markus Müller

Auf den meisten Workstations hat sich inzwischen das X-Window-System als graphische Benutzeroberfläche etabliert. Dieses System bringt bei der täglichen Arbeit viele Vorteile mit sich, aber - besonders im Hinblick auf die Sicherheit - auch einige wesentliche Nachteile. Um diese Probleme bei X-Window verstehen zu können, soll kurz das grundlegende Prinzip von X-Window erläutert werden.

Das X-Window-System - oder kurz X - stellt eine portable, netzwerkfähige, graphische Benutzeroberfläche dar. X läuft auf den verschiedensten Rechnern unterschiedlicher Hersteller, hat aber stets das gleiche Erscheinungsbild und die gleiche Funktionalität. So zumindest das pure X-Window-System, das nach dem Client-/Server-Konzept arbeitet. Hierbei handelt es sich, im Gegensatz zum klassischen Modell, bei den Clients und Servern nicht um konkrete Maschinen, sondern um Programme. Der Server ist im X-System das Programm, das den Bildschirm und die Tastatur eines Rechners oder X-Terminals verwaltet, auch Display Server genannt. Er sorgt dafür, daß der Bildschirm sowie Tastatur und Maus für beliebige X-Applikationen, den X-Clients, zur Verfügung stehen.

Damit ein Client den Bildschirm und die Tastatur eines bestimmten Rechners benutzen kann, muß er sich beim entsprechenden Display Server anmelden, selbst dann, wenn der Client auf derselben Maschine wie der Display Server läuft. Dazu bedienen sich Client und Server des X-Protokolls. Hat sich ein Client erfolgreich beim Display Server angemeldet, stehen ihm ab sofort der gesamte Bildschirm sowie die Tastatur und Maus zur Verfügung, denn der Display Server informiert jedes Client-Programm über eventuelle Tastatureingaben oder Mausbewegungen, sofern das Client-Programm dem Display Server sein Interesse daran bekundet hat. Ferner kann jedes

Client-Programm den gesamten Bildschirm (das sogenannte Root Window) sowohl beschreiben als auch lesen. Insbesondere trifft dies für Window Manager wie zum Beispiel twm, mwm oder olwm zu, bei denen es sich auch um nichts anderes als X-Clients handelt, die mit dem Display Server kommunizieren. Dadurch ist es möglich, daß jeder Benutzer immer das gleiche Erscheinungsbild seiner gewohnten Arbeitsoberfläche vorfindet, egal ob er gerade an einer Workstation oder an einem X-Terminal sitzt.

Dieses Konzept, daß grundsätzlich jeder X-Client auf jedes X-Display (Bildschirm, Tastatur, Maus) zugreifen kann, ermöglicht außerdem die Darstellung von Graphiken oder Diagrammen einer speziellen (Graphik-) Maschine auf z.B. billigen X-Terminals. Es bietet aber auch mehreren Benutzern die Möglichkeit, gleichzeitig mittels X-Terminals an einem Rechner zu arbeiten, wobei jeder die gleiche Funktionalität erhält, als ob er direkt an der Konsole arbeiten würde. Durch diese Offenheit des X-Window-Systems stehen vorerst allerdings auch möglichen Spionage-Programmen alle Türen offen. Mit dieser Problematik vor Augen, wurden bei X11 vier grundlegende Sicherheitsmechanismen implementiert:

1. Host-based Control (xhost)
2. (User-/) Permission-based Control (MIT-MAGIC-COOKIE-1)
3. (quasi) User-based Control (XDM-AUTHORIZATION-1)
4. (real) User-based Control (SUN-DES-1).

Host-based Control (xhost)

Jeder X-Server verwaltet intern eine Liste (Access Control List) derjenigen Rechner, deren X-Clients-Zugangsberechtigung zum X-Server bekommen. Der Server läßt nur Verbindungen von X-Clients zu, deren Rechner in der Liste stehen, und verweigert allen anderen den Zugriff. Diese Liste besteht beim Start des X-Client-Servers in der Regel nur aus dem eigenen, lokalen Display (:0). Durch den File `/etc/Xn.hosts` kann es aber auch mit weiteren Rechnern vorbelegt werden. Dabei steht `n` für die Nummer des Displays, das der Display Server verwaltet. Da in den meisten Fällen nur ein Bildschirm verfügbar ist, steht hier dann auch meistens eine `0`, sodaß das entsprechende File also `/etc/X0.hosts` heißt. In diesem File stehen einfach in jeder Zeile die möglichst vollständigen Adressen der Rechner (s.u.), die eine Zugangsberechtigung zum X-Server erhalten sollen. Diese Liste kann mit Hilfe des `xhost`-Programms jederzeit erweitert oder gekürzt werden. Voraussetzung dafür ist allerdings, daß man schon eine Verbindung zum entsprechenden X-Server hat, beispielsweise durch ein `xterm`-Client, der auf der Maschine des X-Servers läuft. Gibt man nur `xhost` ein, wird die Access Control List angezeigt:

```
Asn2> xhost
access control enabled (only the following hosts are allowed)
awssn2.rus.uni-stuttgart.de
Asn2>
```

Mit `xhost +hostname` fügt man der Liste weitere Rechner hinzu, während man durch `xhost -hostname` Rechner aus der Liste streicht:

```
Asn2> xhost +awssn1.rus.uni-stuttgart.de
awssn1.rus.uni-stuttgart.de being added to access control list
Asn2> xhost
access control enabled (only the following hosts are allowed)
awssn2.rus.uni-stuttgart.de
awssn1.rus.uni-stuttgart.de
Asn2>
```

Die Host-Namen müssen dabei (wie auch in `/etc/Xn.hosts`) nicht vollständig angegeben werden (es genügt allein der Rechnername ohne die Netzadresse), allerdings können dadurch eventuell

auch Rechner mit demselben Namen, aber mit einer anderen Netzadresse auf den Display Server und somit auf den Bildschirm zugreifen. Daher sollte man möglichst vollständige Namen angeben.

Durch `xhost +` ohne `hostname` schaltet man die Kontrolle aus und erlaubt allen Rechnern im Netz den Zugriff auf den Display Server:

```
Asn2> xhost +
all hosts being allowed (access control disabled)
Asn2> xhost
access control disabled (any host is allowed)
awssn2.rus.uni-stuttgart.de
awssn1.rus.uni-stuttgart.de
Asn2>
```

Daß dies natürlich alle möglichen Sicherheitsmaßnahmen außer Kraft setzt, muß wohl nicht extra erwähnt werden. Deswegen sollte man ein `xhost +` tunlichst vermeiden!

Mit `xhost -` schaltet man entsprechend die Kontrolle wieder ein. Aber Vorsicht! Die Liste wird durch `xhost -` nicht etwa gelöscht! Vielmehr wird nur die Zugangskontrolle wieder eingeschaltet. Alle vorher mit `xhost +hostname` eingetragenen Rechner bleiben weiterhin in der Liste und müssen u. U. einzeln durch `xhost -hostname` gelöscht werden.

```
Asn2> xhost -
all hosts being restricted (access control enabled)
Asn2> xhost
access control enabled (only the following hosts are allowed)
awssn2.rus.uni-stuttgart.de
awssn1.rus.uni-stuttgart.de
Asn2>
```

Desweiteren verhindert ein `xhost -hostname` nur zukünftige Verbindungen, alle schon bestehenden Verbindungen von Clients können dadurch nicht mehr unterbunden werden!

Bewertung des Host-based Access Controls mittels `xhost` unter dem Sicherheitsaspekt

Die Zugriffskontrolle durch `xhost` ist immerhin besser als Nichts, bietet aber keinen wirksamen und praktikablen Schutz, denn der eigentliche Grund, einer Maschine den Zugriff auf das eigene Display zu verweigern besteht darin, daß kein fremder Client etwas auf dem eigenen Bildschirm darstellen kann. Dies bedeutet aber, daß man selbst auch keine Clients auf der Remote-Maschine laufen lassen kann, die ihrerseits ihre Ausgaben auf dem eigenen Bildschirm vornehmen. Man schränkt damit also auch sich selbst ein, was nicht der eigentliche Sinn sein kann. Host-based Access Control kann im Prinzip nur funktionieren, wenn jede Workstation eine Single User-Maschine wäre, denn Host-based Access Control bedeutet, wie der Name auch schon sagt, daß nur einzelnen Rechnern der Zugriff erlaubt wird, und somit auch allen Usern auf solch einem Rechner.

Zu einem richtigen Problem wird dies, wenn Yellow Pages bzw. NIS betrieben wird. Da dabei alle Rechner von Grund auf offen sein müssen, fällt für sie der Schutzmechanismus des Host-based Access Control flach. Das gleiche trifft natürlich erst recht für X-Terminals zu, da sie immer auf einen Rechner angewiesen sind, auf dem nahezu alle Clients laufen (meistens auch der Window-Manager), die ihre Ausgaben dann auf dem Bildschirm des X-Terminals vornehmen.

(User-)/Permission-based Control (MIT-MAGIC-COOKIE-1)

Einen Schritt weiter als `xhost` geht der ab X11 Release 4 implementierte, und damit weit

verbreitete Schutzmechanismus mit dem Namen MIT-MAGIC-COOKIE-1. Dieser Mechanismus stellt eine erste und einfache Form eines User-based Access Controls dar. Der Mechanismus besteht aus einem Schlüssel in Form einer 128 Bit langen Zahl, dem sogenannten Cookie. Dieser Schlüssel wird bei Verwendung von xdm bzw. OpenWindows automatisch bei jedem Einloggen beziehungsweise Starten neu erzeugt und dem Server mit Hilfe des XDMCP (xdm Control Protocol) bekannt gemacht. Startet man X von Hand via `startx / xinit`, muß man sich selbst einen solchen Schlüssel erzeugen (siehe unten). Der erzeugte Schlüssel wird im File `$HOME/.Xauthority` (bzw. durch `$XAUTHORITY` angegeben) in binärer Form abgelegt. Jeder Client, der sich nun zu einem X-Server connecten will, schickt zu Anfang den Schlüssel zum Server. Der wiederum prüft, ob der Schlüssel für das gewünschte Display gültig ist und läßt die Verbindung entweder zu oder weist sie mit der Meldung zurück:

```
Xlib: connection to "awssn2:0.0" refused by server
Xlib: Client is not authorized to connect to server
Error: Can't Open display
```

Wichtig ist, daß nur der Besitzer von `.Xauthority` Lese- / Schreibberechtigung für das File hat (mode `-rw-----`), ansonsten kann sich jeder, vorausgesetzt er darf das Home-Verzeichnis eines anderen Benutzers lesen, den Schlüssel holen und hat dann genauso Zugang zum Display Server. Daher ist dieser Schutz nur so sicher, wie die Zugriffsrechte auf das File `.Xauthority` sicher sind. Die Bezeichnung Permission-based Access Control wäre daher für diesen Schutzmechanismus treffender.

Generierung eines Schlüssels/Cookies

Wird ein Schlüssel nicht automatisch beim Einloggen erzeugt (etwa beim Einloggen via xdm), muß man sich einen eigenen Schlüssel generieren. Dazu sollte man eine Zufallszahl erzeugen, oder wenigstens eine Zahl, die schwer zu erraten ist. Voraussetzung für einen gültigen Cookie-Schlüssel ist, daß er aus einer geraden Anzahl von Hexadezimal-Ziffern ('0'-'9','a'-'f') besteht. Jede Ziffer repräsentiert dabei 4 Bits des Schlüssels. Obwohl ein normaler MAGIC-COOKIE-Schlüssel 128 Bits lang ist, spielt die Länge jedoch keine wesentliche Rolle. Fehlende Stellen werden als 0 angenommen, überzählige zwar abgespeichert, aber ignoriert. Man kann sich solch eine Zahl zum Beispiel mit Hilfe von perl erzeugen lassen:

```
zufallszahl=`perl -e 'srand; printf int(rand(10000000000000000))' `
```

Bei der Korn Shell sowie bei der bash kann man den eingebauten Zufallsgenerator verwenden:

```
zufallszahl=`ksh -c 'echo $(( $RANDOM * $RANDOM * 2 ))' `
```

bzw.

```
zufallszahl=`bash -c 'echo $(( $RANDOM * $RANDOM * 2 ))' `
```

Aber auch mit Standard-UNIX-Tools kann man sich eine Zahl erzeugen lassen, die wenigstens bei jedem Einloggen anders ist. Zum Beispiel mit dem date-Kommando:

```
zufallszahl=`date +"%y%m%d%H%M%S" `
```

Oder aber mit Hilfe der Prozeß-ID und bc:

```
zufallszahl=`echo "{obase=16;$$^3}" | bc`
```

Diese Zahl muß nun in binärer Form zusammen mit dem Display-Namen in die Datei `.Xauthority` abgespeichert werden. Dazu dient das Programm `xauth`. Mit dem `xauth`-Kommando `add` fügt man einen neuen Eintrag hinzu beziehungsweise ändert einen eventuell schon bestehenden. Es müssen nach `add` der Displayname (hier `awssn2:0`), für den der Schlüssel bestimmt ist, gefolgt vom Protokoll, das benutzt werden soll, angegeben werden. Durch

einen Punkt (.) erhält man das Standardprotokoll MIT-MAGIC-COOKIE-1. Zuletzt muß natürlich noch der Schlüssel selbst übergeben werden. Das ganze sieht dann wie folgt aus:

```
xauth add awssn2:0 . $zufallszahl
xauth add awssn2/unix:0 . $zufallszahl
```

Der Eintrag `awssn2/unix:0` ist nötig, damit Clients und Server, die auf derselben Maschine (hier `awssn2`) laufen, miteinander via IPC/Unix Domain Sockets kommunizieren können. Dies sind die Verbindungen, bei denen `$DISPLAY` auf `:0` bzw. `unix:0` gesetzt sind. Daß der Rechnername im `.Xauthority`-File im Falle von lokalen Verbindungen auch angegeben werden muß, und nicht nur `unix:0`, hat seinen Grund darin, daß dadurch mehrere lokale Verbindungen verschiedener Rechner mit jeweils unterschiedlichen Schlüsseln in ein und demselben `.Xauthority`-File abgespeichert werden können.

Mit dem Kommando `xinit -- /usr/bin/X11/X -auth $HOME/.Xauthority` muß der X-Server nun noch gestartet werden. Will man dies nicht jedesmal von Neuem tippen, kann man es auch in `.xserverrc` schreiben. Etwa in der Form:

```
#!/bin/sh
# hostname holen
host=`hostname`
# Neuen Schlüssel (magic cookie) erzeugen
# Hier gewünschte Generierungsmethode eintragen (siehe obige Beispiele)
zufallszahl=`....`
# Neuen Schlüssel zu .Xauthority hinzufügen
xauth add ${host}/unix:0 . $zufallszahl
xauth add ${host}:0 . $zufallszahl
# Server starten
exec /usr/bin/X11/X -auth $HOME/.Xauthority
```

OpenWindows erzeugt mit Hilfe des Programms `$OPENWINHOME/lib/mkcookie` beim Start in ähnlicher Weise einen Schlüssel beim Einloggen.

MAGIC-COOKIE-Schlüssel auf andere Rechner übertragen

Das Programm `xauth`

Wird MIT-MAGIC-COOKIE-1 verwendet und man will einen X-Client auf einer anderen Maschine laufen lassen, muß diesem natürlich der Schlüssel für das entsprechende Display vorher bekannt gemacht werden, da sonst der Client vom Server zurückgewiesen werden würde. Das Problem besteht nun darin, daß in der Datei `.Xauthority` der Schlüssel jedes Displays in binärer Form abgespeichert ist. Um die Datei `.Xauthority` zu bearbeiten, dient das Programm `xauth`, das oben schon Verwendung fand.

Angenommen, Sie sitzen an der Workstation mit Namen `awssn2` und Sie wollen einen X-Client, der auf der CRAY läuft, aufrufen. Dazu muß im Home-Verzeichnis auf der CRAY in der Datei `.Xauthority` der Schlüssel für das Display der Workstation `awssn2` stehen. Dies erreichen Sie, indem Sie zunächst den Schlüssel aus der Datei `.Xauthority` auf der `awssn2` mittels `awssn2>xauth extract keyfile awssn2:0` extrahieren. Der Schlüssel für das Display `awssn2:0` steht nun in der Datei `keyfile` in binärer Form. Soll dieser Schlüssel in nicht binärer Form übertragen werden, etwa mit E-Mail, muß er mit dem `xauth`-Befehl `nextract` extrahiert werden.

Die Datei `keyfile` muß nun mit `ftp` zur CRAY übertragen werden. Dann loggen Sie sich auf der CRAY ein und mischen den Schlüssel mit `cray>xauth merge keyfile` in die Datei `.Xauthority` der CRAY. Nun stehen all Ihren Clients auf der CRAY das Display der `awssn2` zur Verfügung, wenigstens solange, wie kein neuer Schlüssel für den Server, zum Beispiel durch erneutes

Einloggen via `xdm` oder Starten von `OpenWindow`, erzeugt worden ist. Startet man häufig Clients auf fremden Maschinen oder logged man sich oft in unterschiedlichen Maschinen ein, ist die Prozedur mit `xauth` und `ftp` natürlich recht aufwendig und veranlaßt vielleicht viele Benutzer, doch wieder gesamten Rechnern mit `xhost +hostname` den Zugriff, mit allen Nachteilen in Bezug auf die Sicherheit, zu gestatten. Startet man jedoch X mit `startx` oder `xinit` und erzeugt sich nicht jedesmal einen neuen Schlüssel, kann man ein und denselben Schlüssel über mehrere Tage - z.B. eine Woche - verwenden, und muß ihn nur einmal - z.B. am Wochenanfang - auf die anderen Maschinen übertragen. Dadurch wird die Sicherheit allerdings auch herabgesetzt.

Bewertung des MIT-MAGIC-COOKIE-1 Access Control unter dem Sicherheitsaspekt

Die Zugriffskontrolle durch MIT-MAGIC-COOKIE-1 bietet einen recht guten Schutz, der in vielen Fällen ausreichen dürfte. Dennoch gibt es einige Punkte, die man bei der Verwendung von MIT-MAGIC-COOKIE-1 wissen muß:

- **Der Schlüssel geht unverschlüsselt übers Netz.** Auch wenn der Schlüssel in binärer Form per `ftp` übertragen wird, ist er deswegen keineswegs verschlüsselt. Der Schlüssel geht auch jedesmal, wenn ein neuer Client über das Netz gestartet wird, unverschlüsselt über das Netz. Jeder, der das Netz abhören kann, kann sich so auch den Schlüssel mitsamt zugehörigen Display-Namen besorgen.
- **MIT-MAGIC-COOKIE-1 wird durch `xhost + umgangen/ausgeschaltet`.** Damit MIT-MAGIC-COOKIE-1 funktioniert, muß der Host-based Access Control (`xhost`) eingeschaltet sein. Denn steht ein Rechner in der Host Access-Liste eines Servers, sind Verbindungen von diesem Rechner generell erlaubt, auch wenn MIT-MAGIC-COOKIE-1 eingeschaltet ist. Daher sollte man stets auf eine leere Host Access-Liste achten und diese eventuell mit `xhost -` bzw. `xhost -hostname` leeren (s.u. `xhost`). Aus demselben Grund sollte bei Verwendung von MIT-MAGIC-COOKIE-1 auch `/etc/Xn.hosts` leer sein. Sicherheitshalber kann man mit einem `xhost` kontrollieren, ob die Host Access-Liste leer ist.
- **Bei einigen X-Terminals, die MIT-MAGIC-COOKIE-1 unterstützen, läßt sich die Host-based Access Control ausschalten** Dies kann leicht zu der Annahme führen, daß bei Verwendung von MIT-MAGIC-COOKIE-1 die Host-based Access Control ausgeschaltet werden kann / muß, weil so ein besserer Schutzmechanismus zum Einsatz kommt. Irrtum: Auch hier muß - wie oben bereits erwähnt - die Host-based Access Control eingeschaltet bleiben / werden!
- **X-Terminals/PC X-Server, die XDMCP unterstützen.** Auch wenn ein X-Terminal oder ein PC X-Server XDMCP (`xdm` Control Protocol) unterstützt, muß dies nicht zwingend bedeuten, daß sie auch MIT-MAGIC-COOKIE-1 unterstützen. Daher im Zweifelsfall im Handbuch nachschauen oder beim Händler nachfragen.
- **Einige X-Terminals und PC X-Server bieten eigene Schutzmechanismen**, die aber meistens eine Art Host-based Access Control darstellen.

XDM-AUTHORIZATION-1

Ab X11 Release 5 gibt es zwei weitere Schutzmechanismen:

1. XDM-AUTHORIZATION-1 und
2. SUN-DES-1.

Beide Verfahren sind sicherer als MIT-MAGIC-COOKIE-1 oder gar `xhost` und benutzen DES (Data Encryption Standard). Aufgrund von Export-Beschränkungen dürfen DES-Implementierungen jedoch nicht aus den USA exportiert werden (2). Daher kann X11 standardmäßig nicht mit XDM-AUTHORIZATION-1 oder SUN-DES-1 betrieben werden.

XDM-AUTHORIZATION-1 arbeitet prinzipiell wie MIT-MAGIC-COOKIE-1, nur daß der Schlüssel nicht unverschlüsselt über das Netz geht, sondern vorher mittels DES verschlüsselt wird; Net Snooping ist also wirksam vorgebeugt. XDM-AUTHORIZATION-1 kann aber im Gegensatz zu MIT-MAGIC-COOKIE-1 nur bei von xdm kontrollierten Sessions verwendet werden. Will man XDM-AUTHORIZATION-1 wenden, muß man sicherstellen, daß die Authentifizierung von xdm eingeschaltet ist und den richtigen Authentifizierungsmechanismus angeben. Dies erreicht man, indem man folgende Resource-Definitionen im File

/usr/lib/X11/xdm/xdm-config ändert:

```
DisplayManager*authorize:          true
DisplayManager._0.authName:      XDM-AUTHORIZATION-1
```

Die letzte Zeile bestimmt, mit welchem Authentifizierungsmechanismus gearbeitet werden soll (hier für Display :0). Man kann auch mehrere Authentifizierungs-Mechanismen in entsprechender Reihenfolge angeben:

```
DisplayManager*authName: XDM-AUTHORIZATION-1 MIT-MAGIC-COOKIE-1
```

Der Stern besagt, daß bei allen Displays, die von xdm verwaltet werden, XDM-AUTHORIZATION-1 verwendet werden soll, oder aber MIT-MAGIC-COOKIE-1, wenn XDM-AUTHORIZATION-1 nicht zur Verfügung steht. Wie bei MIT-MAGIC-COOKIE-1 wird der Schlüssel in die Datei .xauthority geschrieben. Der Schlüssel besteht nun allerdings aus zwei Teilen, einem 56-bit Encryption Key für DES und 64 bits (Zufalls-)Daten.

Benutzt man XDM-AUTHORIZATION-1, sollte man sich beim ersten Einloggen vergewissern, daß XDM-AUTHORIZATION-1 auch wirklich verwendet wird:

```
Asn2> xauth list
awssn2.rus.uni-stuttgart.de:0 XDM-AUTHORIZATION-1
5a5f231bc1bf1ed9c2d9cfde7dbdbc45
awssn2/unix:0 XDM-AUTHORIZATION-1 5a5f231bc1bf1ed9c2d9cfde7dbdbc45
```

SUN-DES-1

SUN-DES-1 verwendet ebenfalls DES und ist deswegen nicht standardmäßig verfügbar (s.a. XDM-AUTHORIZATION-1). SUN-DES-1 verwendet desweiteren SUN's Secure RPC, um die Authorization-Daten über das Netz zu übertragen. Secure RPC benötigt seinerseits NIS (Network Information Service, früher Yellow Pages). NIS wird von SUN-DES-1 aber auch verwendet, um einen Database Server zur Verfügung zu haben. SUN-DES-1 bietet als einziges Schutzverfahren wirkliche User-based Access Control, da es - anders als bei MIT-MAGIC-COOKIE-1 oder XDM-AUTHORIZATION-1 - völlig unabhängig von der Sicherheit der Datei .xauthority ist. Außerdem muß man mit xhost (3) jedem Benutzer, im Gegensatz zu den bisher beschriebenen Verfahren, explizit Zugriff gewähren. Dies ist jedoch nicht ganz einfach, denn die benötigten Namen sehen etwas kryptisch aus und obendrein muß man auch immer wissen, wer den X-Server gestartet hat. Das Verfahren ist zwar akzeptabler als das mit xauth bei MIT-MAGIC-COOKIE-1 oder XDM-AUTHORIZATION-1, aber auch gerade bei häufigem Einloggen oder oftmaligem Wechseln des Displays nicht sehr benutzerfreundlich.

Eine tiefere Beschreibung des genauen Mechanismus würde hier zu weit führen. Außerdem werden ohnehin viele wegen fehlender DES-Implementierungen SUN-DES-1 nicht verwenden werden (können). Es sei nur soviel gesagt: SUN-DES-1 prüft bei jedem gewünschten Verbindungsaufbau zu einem X-Server nach, ob der entsprechende Benutzer die Zugangserlaubnis vom aktuellen Besitzer des entsprechenden Displays bekommen hat oder nicht. Dies wird - wie bei MIT-MAGIC-COOKIE-1 oder XDM-AUTHORIZATION-1 - mit einem

Schlüssel bewerkstelligt, der aber bei SUN-DES-1 mit dem Public Key-Verfahren verschlüsselt über das Netz geht.

Vorbereitungen zur Verwendung von SUN-DES-1

Um SUN-DES-1 verwenden zu können, müssen vorab einige Dinge installiert sein:

- **Secure RPC muß installiert sein.** Maschinen, die Secure RPC installiert haben, besitzen zum Beispiel das `crypt`-Kommando:

```
Asn2> which crypt
/bin/crypt
```
- **Maschinen, die kein Secure RPC haben,** bringen in etwa folgende Meldung:

```
Asn1> which crypt
no crypt in /home/markus /usr/local /usr/local/bin /usr/openwin/bin /bin
/usr/bin
/etc /usr/etc /usr/ucb /usr/5bin . /usr/local/X11/bin /usr/local/X11/lib
```
- **X muß mit**
`#define HasSecureRPC YES`
installiert worden sein
- **NIS muß installiert sein und laufen.** Maschinen, auf denen NIS läuft, liefern beim `ypwhich`-Kommando ihren Namen:

```
Asn2> ypwhich
awssn2
```
- **Maschinen ohne NIS liefern einen Fehler:**

```
Asn1> ypwhich
ypwhich: awssn1 is not running ypbind
```
- **Der Private Key Server muß laufen.** Er wird normalerweise beim Booten einer Maschine gestartet und kann mit dem `ps`-Kommando kontrolliert werden:

```
Asn2> ps -agx | grep keyserver | grep -v grep
61  ?  IW    0:00  keyserver
```
- **Jeder Benutzer muß einen eigenen Public Key Eintrag haben.** Dies kann entweder jeder Benutzer mit `chkey` selbst bewerkstelligen oder der Systemadministrator generiert für jeden Benutzer und mit `newkey` für root neue Schlüssel
- **Die Public Key-Informationen der NIS Clients** müssen dem NIS Master bekannt gemacht werden, und zwar jedesmal, wenn ein Schlüssel verändert oder neu hinzugefügt wird. Wenn `rpc.yppupdate` läuft kann dies entweder automatisch geschehen oder muß explizit vom Systemadministrator des NIS Masters durchgeführt werden:

```
Asn2> cd /var/yp
Asn2> make
```

Einfacher ist dies natürlich, wenn `rpc.yppupdate` läuft und bei jedem reboot automatisch gestartet wird, etwa in `/etc/rc.local`.

SUN-DES-1 kann nur benutzt werden, wenn man einen Eintrag in der Public Key Map des NIS Masters hat. Dies trifft auch für jeden anderen Benutzer zu, der sich zu einem anderen als seinem X-Server verbinden lassen will.

Verwendung von SUN-DES-1 mit xdm

Soll SUN-DES-1 auf einer Maschine laufen, muß zu seiner Verwendung die Authentifizierung eingeschaltet und `xdm` der Authentifizierungs-Mechanismus bekannt gemacht werden. Dies erfolgt durch Änderungen in der Datei `/usr/lib/X11/xdm/xdm-config` :

```
DisplayManager*authorize:          true
DisplayManager._0.authName:       SUN-DES-1
```

Beim nächsten Einloggen via `xDM` sollte kontrolliert werden, ob SUN-DES-1 auch wirklich verwendet wird.

Verwendung von SUN-DES-1 mit xinit

Genau wie bei MIT-MAGIC-COOKIE-1, muß auch bei Verwendung von SUN-DES-1 mit `xinit` einige Handarbeit erledigt werden, dessen Erläuterungen hier allerdings zu weit führen würden und ohnehin wohl nur die Wenigsten SUN-DES-1 ohne `xDM` verwenden dürften. Nähere Angaben dazu finden sich in [\[1\]](#).

Bewertung von SUN-DES-1 in Bezug auf die Sicherheit

SUN-DES-1 ist, verglichen mit den anderen Sicherheitsmaßnahmen, recht kompliziert, bietet aber als einziges Verfahren wirkliche User-based Access Control, da man jedem Benutzer explizit die Zugangserlaubnis zum X-Server erteilen muß. Außerdem ist SUN-DES-1 unabhängig von der Sicherheit der Datei `.xauthority`. Allerdings treten bei der Verwendung von Set uid Clients (`xterm`,...) gelegentlich Probleme auf, da sie den Principal der Maschine verlangen, die den X-Server gestartet hat, und diesem daher auch die Zugangserlaubnis vom Display-Benutzer gestattet werden muß. Da es für einzelne Benutzer nicht immer offensichtlich ist, welchen Namen sie angeben müssen, um eine Zugangserlaubnis zu erteilen, ist SUN-DES-1 an und für sich nur zu empfehlen, wenn man vorrangig auf einer Maschine und einem Display arbeitet. SUN-DES-1 bietet aber dennoch von allen möglichen Mechanismen den besten Schutz.

Allgemeine Sicherheitsprobleme bei der Verwendung von X

`xterm` und Secure Keyboard

Im Haupt-Menü (4) des `xterm`-Clients befindet sich die Option Secure Keyboard, die ein- oder ausgeschaltet werden kann. Wird diese Option eingeschaltet (erkennbar am Haken), empfängt nur noch dieses `xterm` die Tastatur-Events des Servers, da `xterm` ein Grab Keyboard()-Request durchführt. Dies verhindert, daß ein anderes Programm - eines möglichen Spions - die Tastatureingaben mitempfängt, denn im Normalfall können durchaus mehrere Programme die Tastatureingaben von einem Server empfangen, wie etwa das Programm `xkey`, das die Tastatur eines anderen Rechners abhört und mitprotokolliert. Dies ist sehr hilfreich, wenn man sensible Daten, etwa das Passwort oder geheime Daten, eintippt. Wird diese Option eingeschaltet, geht `xterm` in die Reverse Video-Darstellung über, als Zeichen für eine gesicherte Tastatur werden Vorder- und Hintergrundfarben vertauscht.

Sollte `xterm` nicht in die Reverse Video-Darstellung übergehen, ist Vorsicht geboten. In diesem Fall ist es sehr wahrscheinlich, daß schon ein anderes Programm die Tastatur abhört. Um sicher zu gehen, sollte deswegen schon frühzeitig die Option einschaltet werden, damit noch kontrolliert werden kann, ob eventuelle Ausgaben auch wirklich Reverse dargestellt werden. Wenn nicht, ist die Tastatur nicht gesichert! Eine andere Kontrolle bietet der Haken im Menü. Aber Vorsicht: SecureKeyboard wird automatisch ausgeschaltet, wenn `xterm` zu einem Icon gemacht wird, obwohl nach dem Öffnen des Icons weiterhin die Reverse Video-Darstellung zu sehen ist und auch kein Warnton ertönt! Selbst wenn ein neuer Window Manager gestartet wird, wird Secure Keyboard ausgeschaltet, wobei dann wenigstens die Farbe zurückgesetzt wird. Da aber auch bei eingeschaltetem Secure Keyboard immer noch ein Screen Dump möglich bleibt, ist es nur sinnvoll für Eingaben ohne Echo, wie beispielsweise Passworteingaben. Secure Keyboard ist nur bei `xterm` verfügbar, also nicht etwa bei OpenWindow's `cmdtool` oder `shelltool` oder sonstigen

Eingabeprogrammen (z.B. Editoren). Da außerdem nur dasjenige `xterm` mit eingeschaltetem Secure Keyboard Eingaben empfangen kann, muß jedesmal, wenn außerhalb des `xterms` Eingaben vorgenommen werden sollen, wie zum Beispiel in einem Editor, das Secure Keyboard ausschaltet werden.

Die Console und `xterm`

Bei Angabe der Option `-C` bei `xterm` (oder `cmdtool`) erhält der Benutzer ein Console-Window, auf dem Console-Meldungen des entsprechenden Rechners erscheinen, selbst wenn man über das Netz eingelogged ist. Bei X11R4 (und frühere Versionen) war es möglich, daß mehrere Benutzer ein solches Console-`xterm` aufmachen konnten. Hierbei erhielt immer das jeweils zuletzt geöffnete Fenster die Console-Meldungen ohne daß die anderen, selbst derjenige, der direkt an der Console saß, etwas davon mitbekommen haben, außer, daß keine Meldungen mehr erschienen. Unangenehm macht sich dieses Verhalten bemerkbar, wenn sich der Benutzer an der Console ausloggen wollte, der login-prompt aber nicht mehr erschien, und der Benutzer sich deshalb nicht sicher sein konnte, ob er sich wirklich ausgelogged hat oder nicht. Tatsächlich erschien der login-prompt auf dem zuletzt geöffneten Console-`xterm`. Besonders schwerwiegend ist dieses Phänomen aber, wenn sich jemand unter `root` an der Console eingelogged hat, und ein anderer `xterm -C` aufruft. Dieser Benutzer kann dann eventuell `root`-Berechtigung erhalten. Daher gibt es einige Systeme, die ein `xterm -C` ganz verbieten.

Bei X11R5 sollen diese Probleme dadurch behoben sein, daß nur noch der Besitzer von `/dev/console` ein `xterm -C` aufrufen kann, was einige Versuche allerdings nicht bestätigen konnten. Allenfalls bei X-Sessions, die von `xdm` kontrolliert werden, ist eine Möglichkeit gegeben, die dafür sorgt, daß nur der an der wirklichen Console Sitzende ein Console Window erhält, indem in `/usr/lib/X11/xdm/xdm-config` folgende Zeilen eingetragen werden:

```
DisplayManager._0.startup: /usr/lib/X11/GiveConsole
DisplayManager._0.reset: /usr/lib/X11/TakeConsole
```

Diese Zeilen bewirken, daß der Display Manager für das Display `:0`, also die Console, jedesmal, wenn sich jemand an der Console einlogged, vor dem Start sonstiger Clients das Programm `/usr/lib/X11/GiveConsole` aufruft, und nachdem sich der Benutzer ausgelogged hat, das Programm `/usr/lib/X11/TakeConsole` ausführt. Beide Programme, `GiveConsole` und `TakeConsole`, sind in der X11R5 Distribution vorhanden und werden unter `root`-Berechtigung ausgeführt. `GiveConsole` gibt die Console dem an der Console Arbeitenden und sorgt auch dafür, daß er als Besitzer von `/dev/console` eingetragen wird. `TakeConsole` nimmt sich die Console wieder und gibt `/dev/console` an `root` zurück.

Aufhängen des Servers (R3)

In X11R3 gab es einen Bug im Server, der es ermöglichte, diesen lahmzulegen. Der Server wartete nämlich jedesmal, wenn sich ein client connecten wollte, auf ein kleines Paket am Port 6000, bevor er entschied, ob die Verbindung zulässig ist oder nicht, und stoppte solange alle anderen Aktivitäten. Man kann leicht herausfinden, ob ein Server diesen Bug noch enthält, indem man ein `telnet` auf den Port 6000 macht:

```
% telnet localhost 6000
```

Die Ports `6000+n` sind die Ports der Server für Display `n`. Einige Server haben einen automatischen Timeout eingebaut, der nach 30 Sekunden die Verbindung unterbricht. Aber Vorsicht! Da dieser Befehl eventuell den Server einfriert, ist es besser, ihn nicht von einem Window auf dem lokalen Display auszuprobieren!

Framebuffer (Sun Workstations)

Sun Workstations besitzen einen speziellen Framebuffer, der durch das Device `/dev/fb` angesprochen werden kann. Dieser Framebuffer enthält das Bild der Console. Sun liefert auch gleich die entsprechenden Kommandos `screendump` und `screenload`, mit denen das Auslesen und wieder Darstellen des Framebuffers möglich ist. Dies ist übrigens unabhängig davon möglich, welche X-Sicherheitsmaßnahmen gerade aktiv sind!

Den Bildschirm einer anderen Sun auf der eigenen stellt man folgendermaßen dar:

```
% rsh host screendump | screenload
```

Von einem beliebigen X-Server aus muß man anstatt `screenload` das Kommando `xloadimage` verwenden:

```
% rsh host screendump | xloadimage
```

Um dies zu verhindern, könnte man die Zugriffsrechte von `/dev/fb` ändern (z.B. `chmod 600 /dev/fb`). Dadurch können aber eventuell einige Programme nicht mehr ordnungsgemäß laufen.

Eine andere Möglichkeit besteht darin, `/dev/fb` nur für eine spezielle Gruppe lesbar zu machen (ähnlich `/dev/kmem`) und bei allen Programmen, die auf `/dev/fb` zugreifen müssen, die Permission Flags per `set-gid` auf diese spezielle Gruppe einzurichten. Den besten Schutz bietet jedoch eine Änderung in `/etc/fstab`.

Läßt man in `/etc/fstab` den Kommentar bei der Zeile

```
/dev/console 0600 /dev/fb:/dev/bwone0:/dev/bwtwo0
```

weg, kann nur noch der an der Console Arbeitende den Framebuffer auslesen.

Files in /tmp

In dem Directory `/tmp/.X11-unix` legt jeder X-Server, der auf einer Maschine läuft, einen UNIX Socket Descriptor ab:

```
Asn2> ls -l /tmp/.X11-unix
srwxrwxrwx  1 markus          0 Aug 27 16:06  X0
```

Dieser Socket Descriptor wird vom X-Server bei lokalen Verbindungen (`:0` via IPC) verwendet. Da jeder für das Directory `/tmp/.X11-unix` Schreibberechtigung hat, kann auch jeder Dateien innerhalb dieses Directories löschen. Ein Löschen des Files `/tmp/.X11-unix/X0` führt dazu, daß Clients keine lokalen Verbindungen via IPC (diejenigen mit Displaynamen `:0` oder `unix:0`) zum X-Server mehr vornehmen können. Verbindungen über TCP/IP oder DECnet bleiben weiterhin möglich. Um diese Methode, den Server einer fremden Maschine zu stören, zu unterbinden, kann man sich des sticky-bits bedienen: `Asn2> chmod 1777 /tmp/.X11-unix`

Das sticky-bit verhindert, daß Benutzer die Dateien anderer Benutzer in einem Directory verändern oder löschen können. Das sticky-bit sollte übrigens für `/tmp` selbst auch immer gesetzt sein (erkennbar am `t` in `drwxrwxrwt`). Aber Vorsicht! Das sticky-bit wird nicht an Sub-Directories vererbt. Es genügt nicht, nur das sticky-bit von `/tmp` zu setzen! Ein weiteres Problem ist ein auf manchen Maschinen laufender cron-job, der regelmäßig nicht mehr benötigte Dateien in `/tmp` löscht. Löscht dieser cron-job das File `/tmp/.X11-unix/X0`, ist bei einem erneuten Start eines X-Servers das sticky-bit wieder gelöscht. Daher muß dem cron-job irgendwie mitgeteilt werden, daß er das Directory `/tmp/.X11-unix` mitsamt Inhalt nicht löschen darf, zum Beispiel indem ihm

das Löschen von Sockets verboten wird.

Der NeWS-Server von OpenLook verwendet zusätzlich einen weiteren Socket:

```
Asn2>ls -l /tmp/.NeWS-unix
srwxrwxrwx  1 markus          0 Aug 27  16:06  N0
```

Für diese Datei gilt das gleiche wie das oben für /tmp/.X11-unix/X0 Gesagte.

Falsch gesetzte Display-Variablen

Ein großes Sicherheitsmanko entsteht, wenn man eine falsch gesetzte Display-Variable verwendet. Logged man sich in eine Maschine ein und setzt die Display-Variable auf eine falsche Maschine, so ist es durchaus möglich, daß einen diese Maschine nicht zurückweist, sondern eine Verbindung zuläßt. Startet man dann zum Beispiel eine Shell (`xterm`), und jemand an der falschen Maschine ist schnell genug, kann auch er sich unter Umständen alle Files, auf die man selbst Zugriff hat, lesen oder kopieren. Besonders verherend wird dies, wenn einem dieser Fehler als root passiert, sofern man überhaupt als root unter X11 Arbeiten über das Netzwerk vornehmen sollte. Nicht auszudenken, wenn root etwa einen Texteditor mit `/etc/passwd` startet! Daher ist es sicherlich keine schlechte Angewohnheit, direkt nach Setzen der Display-Variable, zunächst ein ungefährliches Programm im Vordergrund zu starten (z.B. `xclock` oder `xlogo`), und nachdem man sich vergewissert hat, daß es auf dem richtigen Bildschirm erscheint, durch `^C` wieder killen.

Das Netzwerk-Design von X11

In der man-page zu X findet man gleich in den ersten Zeilen folgenden Text:

```
X - a portable, network-transparent window system
```

Dieses netzwerktransparente Design von X ermöglicht zwar, daß Client und Server auf verschiedenen Maschinen unterschiedlicher Hersteller laufen können, macht aber zugleich auch fast alle Sicherheitsanstrengungen zunichte. Denn jeder, der das X-Protocol kennt und weiß, wie man TCP/IP abhören kann, kann auch gleichzeitig verfolgen, was man unter X gerade macht. Das X-Protocol könnte zwar verschlüsselt werden, allerdings nicht, ohne daß man merkliche Geschwindigkeitseinbußen hinnehmen muß. Bei XDM-AUTHORIZATION-1 und SUN-DES-1 geht wenigstens der Schlüssel, der für den Zugang zum X-Server benötigt wird, verschlüsselt über das Netz, die eigentlichen Daten - etwa Textfiles - jedoch nicht! Allerdings ist es sehr schwer und mühsam, das X-Protokoll nach den entsprechenden Daten zu durchsuchen. Außerdem ist diese Art des Spionierens auch bei anderen Netzwerk-Anwendungen (etwa NFS) möglich und vielleicht auch einfacher. Wird jedoch MIT-MAGIC-COOKIE-1 verwendet, geht selbst der Schlüssel unverschlüsselt über das Netz. Jeder, der den Schlüssel einmal erhascht konnte, hat somit auch Zugriff zum X-Server, was ihm wiederum die Arbeit des Spionierens oder gar Sabotierens erleichtert.

Literaturverzeichnis

- [1] Mui, L., Pearce, E.: X Window System Administrator's Guide, Volume 8, O' Reilly & Associates, Inc.
- [2] man pages: X, Xsecurity, Xserver, xhost, xauth, DES, xterm, Xsun, openwin, xnews

Markus Müller, NA-5970

E-Mail: markus.mueller@rus.uni-stuttgart.de

2: Es besteht jedoch die Möglichkeit eine Export-Lizenz zu bekommen. Außerdem dürfte es keine rechtlichen Probleme geben, sich eine eigene Implementierung von DES zu schreiben, da der DES-Algorithmus inzwischen schon veröffentlicht worden ist. `xdm` muß dann allerdings mit der Implementierung von DES neu compiliert werden (`HasXdmAuth` muß auf YES gesetzt sein).

3: `xhost` hat bei Verwendung von SUN-DES-1 eine veränderte Aufgabe. Es dient nicht mehr dazu, einzelnen Rechnern die Zugangserlaubnis zu einem Server zu ermöglichen, sondern gezielt einzelnen Benutzern den Zugang zu gestatten

4: Das Hauptmenu von `xterm` erhält man, indem man innerhalb des Fensters gleichzeitig die Control-Taste und die linke Maustaste drückt