# Computer Break-ins: A Case Study*

*Leendert van Doorn*

Vrije Universiteit
Amsterdam, The Netherlands

*ABSTRACT*

Computer break-ins are getting more common every day. Log files and even program binaries are changed, making it very hard for the system administrators to assess the damage and track down the intruders. This paper describes the *modus operandi* of hackers based on multiple hacking attempts that occurred during this year at some department computers. Special attention is paid to the methods they use to break into computer systems and what they do once they are in.

## 1. Introduction

Computer break-ins are much more common now days than they were, say, ten years ago. This is mainly caused by the technological change that took place during this last decade: Computers are interconnected using networks [7] and most systems even rely on this. For example, a network operating system like SunOS†, which is based on the client/server abstraction [1] (where client and servers reside on different machines) relies heavily on its interconnection. Unfortunately, network operating system designers have the tendency to model the network as an extension of the internal data bus. They neglect the fact that a computer network can very easily be eavesdropped -a simple program suffices- while a data bus requires considerably more effort. Eavesdropping is only a tip of the iceberg when it comes to masquerading.

Modern *hackers*‡ are very aware of this mis-modeling and are using it for their benefit. A decade ago, when computers where still unconnected, the only way to get into it unauthorized was either by guessing a name/password pair or by obtaining it from a sloppy user. Now days, a hacker can write a simple program that masquerades as a Network File System (NFS) client and, bypassing all the normal access control mechanisms, access user files directly. NFS is not alone in being vulnerable to these kind of attacks, almost all of the network services are.

This paper surveys some of the techniques used by hackers to break into computer systems. It analyses various kinds of attacks, including the traditional password guessing attack, and the network service attacks as the one in the example above. The information presented in this paper was gathered from traces of computer break-ins that occurred at various Vrije Universiteit department computer centers.

The organizational structure of the Vrije Universiteit is such that each department has its own autonomous computer center, providing the specific services for that field of science. All the centers are connected to a campus wide network which in turn is connected to the Internet. At the time these break-ins took place, there was no umbrella organization that took care of security; each department had to deal with this on its own. But, since most centers are understaffed, security was not one of their main concerns. ‘‘A

*This is a very preliminary version of a more extensive paper describing our experiences with hackers and computer break-ins.

†SunOS is a trademark of Sun Microsystems Inc.

‡There has always been some controversy over the term *hackers,* versus *crackers.* However, *crackers* call themselves *hackers,* and, in a much less positive sense, we do too.

computer is something you take from the box, plug in, and use; possibly browsing through the manual when something goes wrong.'' Unfortunately, and unknown to the administrators, the default setup is often a bit more *user friendly* than one would generally like. It is exactly this atmosphere that made it possible for the hackers to break in.

An often heard tale is that computer security at companies is much tighter than computer security at universities. In principle, this is true due to the nature of the organizations, but from the traces we made, it appears that many companies are vulnerable to exactly the same kind of attacks as universities. The traces showed that research and development machines of several large companies were broken into by the hackers. Fortunately for these companies the hackers were only interested in penetrating the machine rather than stealing trade secrets.

The remainder of this paper is organized as follows: Section 2 describes a profile of the intruder(s): who they are, and what kind of background they have. The bulk of this paper appears in Section 3, their *modus operandi*, in which their attacks, cover-ups, and their behavior on the system are described. Section 4 describes a little bit about detection and what kind of things you should look for, and Section 5 gives a non-exhaustive list of defenses for the attacks described in this paper. Finally, Section 6 contains some reflections.

## 2. Intruder profile

An often neglected subject in computer security literature is a description of the *average* hacker. Probably because no one really knows who they are, and only their traces are known. However, the hackers of the cases on which this paper is based are known. All of them were male, and computer science students doing their master's. They all had access to the Internet, and were reasonable well acquainted with UNIX.† All of the hackers, except one, had the level of an ordinary UNIX programmers with a little bit more understanding of the network software. Only one hacker seemed to have considerable more knowledge, he was able to exploit the more esoteric operating system bugs (the illustrious div/mul instructions on the sparc). Although it is questionable whether he discovered this himself.

† UNIX is a Registered Trademark of AT&T Bell Laboratories.

One can only guess about the hackers motives, why do they break in and what do they gain by it? From the traces we made it became clear that hackers we observed were only interested in collecting as many user/password entries as possible. Once they broke into a computer they used it as a spring board to get into other computers. They were not interested in trade secrets, even though some broke into large software companies and were clearly in a position to get access to them. Sometimes they used the computers to crack password files which were obtained from other systems.

## 3. Modus operandi

Usually a hacker follows a more or less standard pattern to break into computer systems. First he tries to determine a set of potential machines, then tries to get in, and once he has succeeded, he tries to consolidate his position. Finding potential machines is often the easiest part of the process. Either they are machines mentioned in *.rhosts* and *.netrc* files found on systems that are already broken into, or a list is obtained using the domain name system.

The domain name system (DNS) is a hierarchical naming service that maps machine names to their machine Internet addresses, and provides additional host information. A popular DNS feature among hackers is the so called ``zone transfer'' request. When a DNS server receives such a request it will return all the information about the specified domain that DNS knows about. This includes machine names, Internet addresses, host information like the type of the machine, and often the owner and room in which the machine is located.

Using this information a hacker can construct a very specific list of machines that he thinks have the potential to be broken into. For example, from the zone list information he could select all the UNIX machines running BSD network software and attack only those.

## 3.1. Attacking a system

Once a hacker has determined a set of potential machines, he sees himself faced with the problem of getting in. Most multi-user systems provide some kind of authentication mechanism that requires a user to identify himself before he can make use of it. Under UNIX the authentication consists of the traditional user name and password pair. User names are usually publicly known; passwords are supposed to be secret. Even when the user names are not publicly known, they can be easily obtained using the various information services (i.e. *finger*, and *ruser*).

It is interesting to note that the most obvious kind of attack, i.e. trying different user/password pairs at the login prompt, seldom leads to a successful break-in. There are too many combinations, and the login program is usually slow, and disconnects the line after three unsuccessful attempts. However, from the sporadic messages on our consoles we see that attempts to log into the system as *net* or *system* do occur.

For a more successful break-in attempt the hackers turned to the network services that are provided by most systems. Interesting services are: NIS, RLOGIN/RSH, NFS, and FTP. Each of these services is described in turn, giving a general idea of the security problems as we observed them from the traces. It should be noted that the security problems described below are already public knowledge and fixes for them do exist (see § 5).

### Network information service

The network information service (NIS), also known by its former name *yellow pages*, is essentially a simple distributed database service with multiple clients and replicated servers (for availability and load sharing). The service is used to store various site wide databases, like the password file, group file, and many others. Since many of these files contain sensitive information (as opposed to secret information), access is limited to those clients that know the NIS domain name. This scheme allows only machines that know the NIS domain name to retrieve database entries.

Unfortunately the NIS domain name is often the same as the DNS domain name for a site (the result of the default installation procedure), or some simple derivation from it. One of the programs that was used by the hackers was a simple NIS client that tried various permutations of the DNS domain name. This program was primarily used to get a copy of the password file, and once they got that, a password crack program (Alec Muffett's crack) was used to break the entries in the file. This is a nice example of where the technological advances weakened the assumptions underlying the UNIX security mechanism. In line with the UNIX philosophy, passwords are stored in an encrypted form in a public readable file. This file was readable since the designers did not think that a brute force attack was feasible [5]. However studies [4] showed that given an English, French, and Dutch dictionary, a list of male and female names, and a lot of clever permutations of them, 25% of the passwords can be broken. This result is consistent with our traces: whenever a hacker got hold of a password file he was always able to break at least some entries, hereby getting access to the system.

To solve this problem, the concept of shadow passwords was introduced. A shadow password system consists of two databases: the actual password file which is only available for privileged users, and a shadow password file without the encrypted passwords which is available for everyone. This mechanism makes it harder for a hacker to retrieve the encrypted passwords, but does not eliminate the problem. A shadow password file can still be read by anyone as long as they make the request from a *privileged* port. On a UNIX system only a privileged user can make such connections, but on a non UNIX system this restriction does not exist. For example, any PC with a TCP/IP package can make network connections from a privileged port. And hereby retrieve the shadow password file.

The traces showed that at least one hacker was capable of entering about 250 sites using the method described above. This was mostly done by hand (sometimes with the help of a shell script), and thus very laborious. But since we are in field of automating laborious tasks, just imagine the impact of a program that would automatically traverse the DNS directory graph and would retrieve the password file from every machine it came across.

## Remote login/shell service

The remote login service provides a remote terminal service, i.e. users can login over the network as if they were attached to a direct terminal. Ordinarily users only have to provide a password to login using this service, since the user's name is given to the remote server as part of the initial handshake. To save the user from having to type his password every time he logs in from one system to another, the login service provides an automatic authentication facility that gives immediate access to *trusted users*, or users who are logging in from *trusted hosts*.

A related service is the remote shell service which provides a remote command execution facility. Anyone coming from a trusted host or is a trusted user can execute remote commands. This service uses the same authentication mechanism as the login service; for this reason the two services are discussed here as one.

A trusted host is one whose name appears in a special database *(/etc/hosts.equiv)*. This database contains the names of all the hosts for which the users do not need to supply a password when they want to login (using the remote login service) or execute a command (using the remote execution service), provided that the user name exists on the local machine. An important feature of this database is that it can contain wildcards. For example a ‘‘+’’ (plus) means than any host is allowed access. It is especially this wildcard feature that is responsible for a lot of security problems. Most computer systems that are shipped today (most notably Sun systems) distribute their system with a single plus in the trusted host database, meaning that any host is trusted. It is surprising to see that many system administrators aren't aware of this feature and do not disable it.

From our traces we observed that most of the attacked systems still had a plus in their trusted hosts database. The only problem which the hacker had to solve was to find a user name that existed on his and the remote system. But given the generous variety of standard UNIX login names this wasn't a difficult exercise.

Apart for the system-wide database, each user can have his own private database, named *.rhosts*, which is located in his home directory. A user's .rhost file can also contain a list of *trusted users* on a per machine basis. For example, the following .rhosts file allows user Joe at flytour.cs.vu.nl or splash.cs.vu.nl to log into the system without the need of a password.

```
flytour.cs.vu.nl joe
splash.cs.vu.nl joe
splash.cs.vu.nl marcy
```

Also user Marcy at the machine splash.cs.vu.nl can login without typing her password. The wildcards in the trusted host database can also be applied to the host names and trusted user names in the .rhost file. For example, a .rhosts file containing the line ‘‘+ +’’ allows any user from any system to login without providing a password.

When used wisely these trust mechanisms are very valuable. Users don't have to type their password every time they log into a trusted machine and remote commands can be executed without logging in first. Unfortunately, letting the user decide who to trust and who not to trust is in general a very bad idea. Suppose that the owner of the example .rhosts file trusts Joe and Marcy, and in turn Marcy trusts Karen, and Karen trusts Mark. Using this trust relation Mark can become Karen, and then Marcy and finally login using the account of the owner of the .rhost file. When Mark's account is broken into, then Karen's and Marcy's accounts are also broken.

From our traces it appeared that only a few hackers actually tried to follow this trust relation chain (others probably thought it was too much work). Most of them misused this trust system by planting a .rhosts that allowed full access from anywhere into a home directory, using NFS, or FTP.

## Network file system

Sun's network file system (NFS) [6] is a system that allows system administrators to *mount* file systems that reside on remote computers in exactly the same manner as they would mount a local disk. NFS uses the client server abstraction, i.e. some machines function as servers (those which have disks with file systems) and others function as clients (diskless clients, but possibly also the servers themselves). The benefits of NFS are widely advocated: files don't have to be replicated over all the machines which saves disk space. users are not tied to one particular machine since their directory can be accessed from any machine, and maintenance becomes easier since all files are centralized.

NFS is probably the most important and vulnerable network service in the system, because it provides full access to files and directories. It is therefore unfortunate that the current NFS implementation has more than its fair share of security problems. The primary problem is that NFS's access control mechanisms are very hard to maintain, and are hardly adequate (except for sites which have installed the latest NFS patches). Another problem is NFS's lack of user authentication, even when using the so-called *secure* NFS implementation.

To implement the access control mechanism, the NFS service is broken up into two separate servers: the mount daemon†, and the NFS daemon. Access control was deemed inherently more operating system dependent and therefore was provided by a separate server, the mount daemon. Its primary function is to handle mount requests and determine whether a client is allowed to access the file system. When a client is allowed access, the mount daemon returns a *file handle* for the root directory of the file system. With this file handle the client can make a request to the NFS daemon. Every file or directory has its own unique file handle, and given the file handle for a directory, the client can obtain new handles for files and sub-directories from the NFS daemon. This daemon performs the normal file system operations (e.g. read, write) on files which are represented by their file handle. In a sense, a file handle resembles a UNIX file system *i*-node.

The problem with this type of access control is that once the hacker gets hold of a handle, or is able to make one up, he has full access to the file system. And this access cannot be revoked. Getting hold of a file handle is not that hard. Often the export lists (which file systems may be exported to which system) are badly maintained and allow any access. From our traces it became clear that bad export lists were one of most successful ways for a hacker to get into a new system. The strategy he used was the following: he would mount the /usr file system, put a .rhosts file in the /usr/bin directory giving him access, and then login as the user that owns all program binaries, named bin, using the remote login service. Since /usr/bin (actually /bin but this is a symbolic link to /usr/bin) is the home directory for this user, and contained a valid .rhosts file the remote login would succeed. Being bin on a machine gave him write access to all system binaries and important data files and it was only a matter of minutes before he got the permissions of the privileged user.

The hacker could write the .rhost file in the /usr/bin directory because he was able to specify the user and the group ids of the file. This works for all user and group ids, except that of the privileged user, but even this is a configurable parameter. This is a very important flaw in the NFS system. Every user can write his own NFS client (no special privileges are required), specify any identity and read or write files. An NFS client that provides this basic functionality can easily be written in about 300 lines of C code.

Secure NFS tries to remedy this flaw but succeeds only partially. Secure NFS's basic problem is that the underlying crypto-system is broken, and programs exists that in one minute generate all the secrets involved. However, from our traces there was no evidence that the hackers used such programs, probably because it takes considerable cryptographic skill to write them. Another interesting flaw is the problem with file handles. They can be constructed without the help of the mount daemon. With this the client can go directly to the NFS daemon, and bypass the access control mechanisms which are enforced by the mount daemon. Fortunately, this bug has recently been fixed (see § 5).

### File transfer protocol service

The file transfer protocol (FTP) service allows clients to copy files from one machine to another. It resembles NFS in a way, but is intended for long haul networks. Normally the client has to provide a user name and password to get access to the remote system, although a public account like *anonymous* or *ftp* is often available. Internet archive sites make use of this feature to allow any Internet user access to

†Daemon is a UNIX term for a process which has no terminal attached to it. We use the term loosely as a synonym for server process.

their archives without the need to create individual entries for them.

The current FTP implementation has been notorious for its security problems. The FTP service used to be a very simple one, only providing login, store, and retrieval functionality. But over the years many features were added making it a complex and difficult to understand program. One of its problems was that it could be tricked into giving the hacker the permissions of, for example the user Joe, while the hacker actually logged in using a public account. Even though these particular bugs have all been fixed, the FTP service stays suspect and should be watched very carefully.

From our traces it became clear that it is not the FTP program bugs which are exploited by the hackers, but rather the negligence of the administrators. One of the first attacks a hacker would undertake was to FTP to a system using the public access account, and determine whether the directory he got in was writable. If so, he would put a .rhosts file into it (which contained his name and his current machine). Since the directory is often the home directory of the user ftp (or ftpd), a simple remote login sufficed to get into the system. It was surprising to see that systems that provided public FTP access had a writable ftp root directory, and thus were liable for these trivial attacks.

Another security risk lies with the client program of the FTP service. This program saves the user from typing his user name and password every time he connects to a remote machine. It does this by looking up the machine name in a file called .netrc in the user's home directory and use the login name and password stated in that file instead. However, this password is stored in plain-text, and the hacker only has to be able to read the file to get yet another entry. Storing passwords in plain-text is always a bad idea!

## 3.2. Covering up tracks

One of the first things a hacker will when he breaks into a system is to cover up his tracks. This is usually done by modifying or even removing system log files. Another trick that is commonly used is to bypass the logging facilities of the remote login and remote shell service is to use the remote execution service (REXEC). This service allows a client to execute commands given a valid user name and password. In this respect it is very much the same as the remote shell service, but without the automatic authentication, and more important, without making log entries†. It is exactly this last feature that makes the service so popular among hackers. A common method to get into a system unnoticed is to use the remote execution server to make a copy of the log files, and then use the remote login service to login. Then he would try to become root and overwrite the log files with the saved copies (which did not contain any evidence of his login). Becoming root never seemed to be the real problem.

## 3.3. Hackers behavior

Once a hacker successfully breaks into a computer system and has removed the evidence of his presence, he goes about consolidating his position. This can be as simple as placing a .rhosts file in the home directory of a cracked account or as devious as replacing system binaries by patched versions. For this last action he needs to have root privileges (sometimes the owner of the binaries suffices). Besides consolidation, hackers are also very interested in mailboxes and user's .netrc and .rhosts files.

From our traces we've seen hackers place .rhosts files that trust any user from any host in home directories of unused accounts and even in the root directory. The later allowed any user from any host to login as root. Much more devious actions were those where the hacker replaced existing system binaries by augmented ones. For example, a hacker replaced the su and newgrp commands by special versions that gave him a privileged shell whenever he supplied a special password. Other favorite targets were programs that asked for a passwords (e.g. the FTP and TELNET clients, but also their server programs). The augmented programs continued to work, but logged every password that were given to them into a special file which was, of course, only known to the hacker.

†On some systems the log server (syslog) can be set up in such a way that it logs successful REXEC calls. None of the hackers seemed to be aware of this.

```
#!/bin/sh
LOGFILE=logfile
while true; do
    case ‘date | cut -d" " -f5 | cut -d: -f1‘ in
    18|19|20|21|22|23|00|01|02|03|04|05|06|07)
        (echo "======= "; date) >> $LOGFILE
        (echo "who"; who) >> $LOGFILE
        (echo "ps axl"; ps axl) >> $LOGFILE
        (echo "netstat -n"; netstat -n) >> $LOGFILE
        sleep 600
        ;;
    *)
        sleep 3600
        ;;
    esac
done
```

**Figure 1.** Example shell script used for spotting a hacker.

A more clever attack was carried out by a hacker who modified the tcpdump program (see § 4) and saved every network packet containing a name and password destined for the FTP server on the same network. This way he didn't have to install a modified version of the FTP server program, in fact he didn't even have to be on the machine where the server ran as long as he was on the same network segment.

Once they became the privileged user they were also very interested in user's mail boxes, .rhosts, and .netrc files. Mail boxes were frequently scanned for words like *hacker*, *security*, and *password*, and read subsequently. System administrators were identified either from the mail system's alias list or special group identities. Their directories were carefully scanned for all kinds of interesting things, including bug reports and fixes, lists of newly created user accounts, and so on. We also observed that hackers killed or reniced programs because they used too much CPU time on *their* system.

## 4. Detection

Detecting a break-in is often difficult, especially when hackers try to cover up their tracks. In fact, all the hacking attempts on which this paper is based were detected by mere accident, either because an administrator noticed something ‘‘strange’’ (e.g. a gap in a log file, or an absent user starting to use his account), or were warned by other administrators (e.g. who had found foreign password files).

Even though detection depends a lot on just luck, a number of general points can be made. You should always examine your log files regularly, especially the files generated by the system log service and the wtmp file. Watch for connections made from unusual hosts or at unusual times (Wietze Venema's TCP wrapper package [8] is very useful in this respect). Also accounts that have been unused for a while and start being used again are suspect. A common rule is that hackers usually go about their business between the hours of 18.00 and 8.00, and on Saturdays, Sundays, and national holidays.

An easy way to check whether a hacker is working during these hours is run a simple shell script every 10 minutes that logs all the processes and network connections to a file. An example of such a shell script is given in figure 1. In fact, it is a reminiscent of the shell script we initially used to find out what the hackers were doing (this was after they were detected, and before the trace program was installed). This script logs a number of things: a list of current users (although hackers usually remove themselves from that list), a list a all current process, and a list of all current network connections.

Of course, the use of such a shell script is rather limited. It doesn't take a smart hacker very long to find out that he is being watched, since the commands in the script show up in the process list. A for the hacker unobservable tracing method is to use a program that snoops the network and prints the contents of the network packets. Of course, this program should run on a stand-alone computer that is connected to the network, but in such a way that the hacker doesn't have access to it. Examples of such programs are etherfind and tcpdump. Especially tcpdump can quite easily be modified to display the contents of every network packet.

## 5. Defenses

Defenses against computer break-ins is the subject of many books that have been written (the most useful one for UNIX system administrators is no doubt [3]). Without repeating all that is written in those books, some general remarks, with respect to the problems described in this paper, can be made:

- When using NIS, be sure to install the latest patches (see below).
- Remove any wildcards from the trusted hosts database (/etc/hosts.equiv).
- Use .rhosts and .netrc file wisely, perhaps disallow foreign hosts. Do not use wildcards, and do not store plain-text passwords in .netrc files.
- When using NFS, be sure to install the latest patches (see below). You should also be sure to specify to which hosts (or groups of hosts) you export your file systems. When you export the user file system (/usr) consider exporting it *read-only*. Consider disallowing setuid and root access for any NFS file system (except the roots of diskless workstations, of course).
- Also consider turning off the ''-n'' option of the mount daemon (see /etc/rc.local). The mount daemon manual says that your system is slightly less secure, this is **wrong**. Your system is wide open.
- When you offer FTP access be sure that the FTP spool directory is read-only.
- Subscribe to the CERT advisory mailing list (send mail to cert-advisory-request@cert.sei.cmu.edu to join), and apply these fixes regularly.

The latest NIS, NFS and many more security software fixes for SUN computers can be found in ftp.uu.net:/systems/sun/sun-fixes, or from be obtained your own vendor.

A very useful tool to detect security weaknesses (possibly caused by hackers) is the freely available program called COPS [2]. A copy of COPS can be found in cert.sei.cmu.edu:/pub/tools/cops.

## 6. Reflections

All the break-ins described in this paper could have been prevented when the administrators of the attacked systems were more aware of computer security. However, this would have meant that all of them should carefully study the UNIX security system, install all the programs and frequent updates as soon as they appear, browse through all the log files frequently, and more important keep up with all the news and rumors that appear on the many security news groups. This is a full man's job. I therefore believe that you can not blame the system administrators, they simply didn't have the extra time for all this given their already hectic job.

On the other hand, if they would have known the information contained in this paper the risks of being attacked and the resulting damage would have been much less. This is exactly the reason why I wrote this paper, to give an survey of the way hackers work and what a system administrator can do to prevent attacks. But by no means, I have the illusion that a system will be impossible to penetrate when all the problems described in this paper are properly addressed.

It is also my belief that security is not so much a technological problem as well as a software engineering one. For most of the technological problems adequate solutions do exist, it is only matter of vendor interest whether they apply them in their products. Although vendors are becoming more aware of security, it is still something you add-on as an afterthought. This is wrong. Security should be kept into account during the whole phase of product design. This will also make security much more user friendly. There is absolutely no reason why security is as user hostile as it is today. This is also the main cause of the administrative problems. A lot of switches and buttons must be pushed in order to get security working, and often something is missed. If security would be the default and nicely integrated into the system none of these problems would occur. Getting a remote password file using the NIS service would be impossible, so would mounting a remote file system, and a remote login as user bin would never succeed since the trusted hosts database would not contain a wildcard as an ‘‘acceptable’’ default.

## 7. Acknowledgments

I would like to thank Mark Wood for reading the manuscript and especially the system administrators of the attacked systems for their dedication, since they were the ones who did the actual tracing, examined the log files, wrote the shell scripts, and took subsequent actions. My input was merely to give advice on how to set up a trace, use my trace programs, and interpret the results.

## References

1. A. D. Birrell and B. J. Nelson, Implementing remote procedure calls, *Trans. Computer Systems 2*, 1 (Feb. 1984), 39-59.
2. D. Farmer and E. Spafford, The COPS Security Checker System, *USENIX Conference Proceedings*, Anaheim, CA, 1990.
3. S. Garfinkel and G. Spafford, *Practical Unix security*, O’Reilly & Associates, Inc., 1991.
4. D. V. Klein, Foiling the Cracker: A Survey of, and Improvements to, Password Security, *EKUUG Summer 90*, London, July 1990, 147-154.
5. R. H. Morris and K. Thompson, Password Security: A Case Study, *Comm. of the ACM 22*, 11 (November 1979), 594-597.
6. H. Stern, *Managing NFS and NIS*, O’Reilly & Associates, Inc., 1991.
7. A. S. Tanenbaum, *Computer networks*, Prentice Hall, Englewood Cliffs, NJ, second edition, 1988.
8. W. Venema, *TCP WRAPPER, een tool voor het bewaken van netwerkactiviteit*, Technische Universiteit Eindhoven.