JULY 1991

NSL
Technical Note TN-2

Using *screend* to Implement IP/TCP Security Policies

Jeffrey Mogul

digital **Network Systems Laboratory** 250 University Avenue Palo Alto, California 94301 USA

Digital E-net:	`DECWRL::NSL-TECHREPORTS`
Internet:	`NSL-Techreports@pa.dec.com`
UUCP:	`decwrl!nsl-techreports`

Using *screend* to Implement IP/TCP Security Policies

Jeffrey Mogul

Digital Equipment Corporation
Western Research Laboratory

July, 1991

Abstract

True network security requires that all hosts attached to the network are themselves made secure, but network administrators often find it helpful to block certain kinds of packets at the routers. The ***screend*** program was designed to provide this function for routers based on the ULTRIX™ operating system. Although ***screend*** is a simple program, creating a ***screend*** configuration that does not compromise security can be a complex problem, requiring deep understanding of how IP/TCP networks are used and abused. This technical note provides some guidance.

Table of Contents

List of Figures

1. Introduction

Internetworking has greatly improved communication between administratively distinct organizations, linking businesses, schools, and government agencies to their common benefit. Unfortunately, internetworks that connect multiple organizations create potential security problems that cannot be solved by the mechanisms used within organizations, such as restricting physical access. In particular, interconnection at the datagram level is an ‘‘all or none’’ mechanism, allowing outsiders access to all the hosts and applications of an organization on the internetwork. The magnitude of this threat has been underscored by several incidents affecting large internetworked communities [19, 22, 24, 25].

To completely protect against penetration, every host within an organization must be made secure, no small feat when it involves tens of thousands of poorly-managed workstations and PCs. One alternative, perhaps less secure but certainly more feasible, is to block certain kinds of packets at the routers that connect between organizations. Most commercial routers now support this function, variously called ‘‘packet filtering’’ or ‘‘gateway screening,’’[1] with varying degrees of flexibility.

Although there are many excellent reasons why one might choose not to use a general-purpose system as a router, in some cases that may be the most appropriate choice: budgets might not allow the purchase of a special-purpose router, or available routers may not provide the required functionality. Often, the performance demanded of routers interconnecting organizations is lower than that required of routers used to connect LANs within an organization; this makes it quite feasible to use a general-purpose system as an interorganizational router.

Routers based on the ULTRIX™ operating system (version 4.2 or later) support the use of the *screend* program. *Screend* does gateway screening for the IP/TCP protocol family, and supports a wide variety of security policies through the use of a flexible configuration mechanism.

Although *screend* is a simple program, creating a *screend* configuration that does not compromise security can be a complex problem, requiring deep understanding of how IP/TCP networks are used and abused. This technical note is meant to assist a network administrator in using *screend* to implement network security policies.

I will assume that the reader is familiar with

- the basic concepts of the IP, TCP, UDP, and ICMP protocols
- the ULTRIX operating system
- the structure of his or her own network environment

and is generally aware of the kinds of threats that the network might be exposed to.

[1]The term ‘‘packet filtering,’’ which is perhaps more natural, had already been used for several years to name an unrelated mechanism for implementing protocol software [14]. To avoid confusion, I will use the word ‘‘screening’’ in this document.

1.1. Better approaches to security

The philosophy behind *screend* is not to provide full security. Rather, it is meant to allow systems that provide their own security to be reachable from the outside world, while banning connections to internal systems that are not known to be secure. In an ideal world, every internal system would be secure, and *screend* would have no purpose.

It is generally accepted that security in distributed systems is best accomplished through the use of encryption technology. Encryption may be employed not only to ensure the privacy of data, but also to ensure the integrity of the systems involved through the use of authentication mechanisms. One such mechanism is the Kerberos authentication system [23]; Kerberos has its drawbacks, but if it or a similar system were ubiquitous, there would not be nearly so much need for *screend*.

Since encryption-based security is not widely available, I'll assume that the reader is not able to employ it, and must depend on the more primitive methods described in this paper.

1.2. Other reading

To find out more about how *screend* works, its performance characteristics, and its application in other areas, the reader should obtain a copy of the original paper on *screend* [11], or the WRL Research Report[2] that reprints the paper [12].

An excellent introduction to the concepts of the IP/TCP protocol family may be found in the books by Douglas Comer [1, 2].

Internet protocols are defined by documents called ''RFCs'' (literally, RFC stands for ''Request For Comments,'' but many RFCs serve as standards documents). The most important RFCs include:

RFC791
: Internet Protocol (IP)

RFC792
: Internet Control Message Protocol (ICMP)

RFC768
: User Datagram Protocol (UDP)

RFC793
: Transmission Control Protocol (TCP)

Additionally, one should obtain an up-to-date version of the ''Assigned numbers'' document, which contains a variety of information but specifically indicates the assignments for TCP and UDP port numbers. At this writing, the most recent version was RFC1060.

[2]For information on obtaining WRL Research Reports, send electronic mail to `WRL-Techreports@wrl.dec.com`, with the word ''`help`'' in the Subject line.

RFCs are available from the Network Information Center (NIC) at SRI International.

> Paper copies of all RFCs are available from the NIC, either individually or on a subscription basis (for more information contact NIC@NIC.DDN.MIL). Online copies are available via FTP or Kermit from NIC.DDN.MIL as RFC:RFC####.TXT or RFC:RFC####.PS (#### is the RFC number without leading zeroes).
>
> Additionally, RFCs may be requested through electronic mail from the automated NIC mail server by sending a message to SERVICE@NIC.DDN.MIL with a subject line of ‘‘RFC ####’’ for text versions or a subject line of ‘‘RFC ####.PS’’ for PostScript versions. To obtain the RFC index, the subject line of your message should read ‘‘RFC index’’

If you cannot send electronic mail to the NIC, they can be reached at:

Network Information Center
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

(800) 235-3155

2. IP/TCP Security Model

The choice of which packets a router should be configured to block or accept is based not only on the policies one is trying to enforce (which will be covered in sections 3 and 4) but also on a model of how IP/TCP networks are used. This section covers the concepts that are important in understanding how to use *screend* to enforce policies; without knowing what can and cannot be enforced, one cannot devise a meaningful security policy.

Even if you think you know all about this, I encourage you to read section 2.7, or you may find yourself vulnerable to forgeries.

2.1. IP addressing

Security policies must be expressed in terms of the active entities involved. Ideally, one would like to know which people (or groups of people, or sometimes autonomous systems acting on behalf of people) are involved in a communication, since ultimate responsibility does rest with people. Unfortunately, absent the use of cryptographic authentication and sealing, IP packets are not tagged with the names of people.

Instead, we must rely on the addressing information present in the packet header when making security decisions. IP packets can carry several levels of addressing information: host addresses in the IP headers, and port numbers in the TCP or UDP headers. While IP routers are not supposed to look at higher-layer headers (such as TCP or UDP headers), *screend* obtains most of its flexibility and precision by such peeking.

IP host addresses originally formed a two-level hierarchy, being divided into ‘‘network number’’ and ‘‘host number.’’ This became unwieldy as the Internet grew, and an intermediate level of hierarchy was introduced [13]. Addresses are now divided into ‘‘network number,’’ ‘‘subnet number,’’ and ‘‘host number.’’

There are several different IP address ‘‘classes,’’ each with a different size of the network number field. The class of any given address, and thus the size of its network number field, can be determined by examination. The choice of how to allocate address bits between subnet number and host number is left to local administration, however, and so additional information is needed to parse the address. This is done using a 32-bit value known as an ‘‘address mask’’ (sometimes ‘‘subnet mask’’). The address mask indicates which bits of the IP address are used to encode the host number, and which bits are used to encode together the network number and subnet number.

In theory, one need only know the address masks for the networks that one is directly connected to. A user of *screend*, however, might want to distinguish between subnets of a distant network; thus, *screend* allows you to tell it the address mask for any given IP network. It is up to you to get the value right.

Because of the relatively small IP address space, some organizations have started to use variable-width subnet fields; that is, on a single IP network the allocation of address bits between subnet number and host number may be different on different subnets. The current version of *screend* does not support this concept; this is only a problem if you want to be able to distinguish between different subnets of such a network. In our experience, there are few useful policies that depend on distinguishing between subnets (but see the example in section 6.2).

In summary: every IP packet carries an IP source address and an IP destination address. *Screend* can implement policies based on individual IP host addresses, on IP network numbers, or on IP net+subnet identifiers.

One relatively new class of IP address is used for multicasting (transmission of a packet to a designated group of recipients) [4]. *Screend* does not currently support the use of multicast addresses.

2.2. TCP and UDP port numbers

Every IP packet carries a Protocol number, which indicates how the next layer of header is to be interpreted. The two most commonly-used protocols in this layer are TCP and UDP[3]. These are often referred to as ‘‘transport protocols,’’ because they are used for transport of data between processes in the end hosts.

Because TCP and UDP are process-to-process protocols, they add another level to the addressing hierarchy; each TCP or UDP header includes source and destination ‘‘port’’ number fields. A port is not exactly a process identifier; rather, it may identify a particular kind of service or a party participating in a given connection.

Ports may be broken down into several classes:

- **Well-known** ports are those assigned to particular services, such as Telnet, File Transfer (FTP), or Mail (SMTP). The well-known port assignments are an Internet standard and are listed in the Assigned Numbers RFC. Ports numbered 0 through 255 are ‘‘well-known.’’

[3]Not every IP packet is a TCP packet!

- **Reserved** or **Unix** ports are those below 1024. These are ports that the 4.2BSD Unix system (and its successors, including ULTRIX, SunOs, OSF/1, AIX, etc.) reserve for use only by privileged processes. This might seem like a good thing for security, but it actually provides at best a weak level of protection. Some reserved ports, like well-known ports, have pre-assigned meanings, but they aren't officially assigned by the NIC.
- Other ports, in the range 1024 through 65535, are occasionally defined by convention for certain specific uses (see the Assigned Numbers RFC) but normally may be assigned to arbitrary processes.

2.3. TCP connections

TCP connections are uniquely identified by a quadruple:

Source IP Address, Source Port, Destination IP Address, Destination Port

This means that several processes on machine host may have separate connections to a single well-known port on a different host. The concept is subtle and understanding it might help you avoid some confusion. However, *screend* does not distinguish among TCP connections.

2.4. ICMP error messages

The Internet Control Message Protocol (ICMP) is used for communicating certain kinds of control information between hosts, or from a router to an end host. From the security point of view, ICMP messages fall into two categories: ‘‘information’’ messages and others. ‘‘Information’’ messages are those that we consider basically harmless; these include Echo, Timestamp, and Address Mask requests and replies. Other ICMP messages, such as Destination Unreachable or Redirect messages, could be used to cause problems for a host that receives them (especially if the messages are forgeries).

Screend allows you to control ICMP access either on a per-message-type basis, or by the Information/other distinction.

2.5. IP header options

The basic IP header contains fairly minimal information, but it may be extended through the use of IP ‘‘Header Options’’. Some of the available options (specifically, the Source Route options) could be used to bypass the checks that *screend* performs on the basic header. Since header options are virtually unused in current practice, *screend* does not currently allow *any* packets with header options to be forwarded.

2.6. Fragmentation

IP datagrams may contain up to 64K bytes of data. Virtually all networking technologies require packets to be smaller than this (for example, the Ethernet limits packets to about 1500 bytes), and in an internetwork it is likely that different subnetworks will have different ‘‘Maximum Transmission Units’’ (MTUs). This means that when a router forwards an IP packet, it may find that the packet is too big to transmit on the outgoing link.

In an attempt to make the MTU issue transparent to applications, IP systems can ‘‘fragment’’ packets that are too large to send in one piece. Each fragment has a copy of the original IP header (more or less), but only the first fragment of a TCP or UDP packets contains the higher-level protocol header. Thus, *screend* cannot look at subsequent fragments in isolation to extract their TCP or UDP ports.

To get around this problem, *screend* keeps track of a number of recently-received ‘‘first fragments,’’ which do contain the TCP and UDP headers. When a non-first fragment arrives, *screend* checks to see if it has already seen the corresponding first fragment; if so, it forwards or rejects the new fragment according to the information in the first fragment.

This works only when the fragments of a packet arrive in order at the system where *screend* is running; this is likely to happen, given current Internet practices, but it is not guaranteed. Also, if the volume of fragmented packets is too high, *screend* may lose track of them. Finally, if packets can follow two or more paths that do not converge before (or at) a single *screend*-equipped router, the fragment-matching algorithm will not work. (That means that in principle you cannot put two *screend*-equipped routers in parallel, unless you don’t need to support fragmented datagrams. However, since most existing routers do not support split-path routing, in practice this has not been a problem.)

Fortunately, people have realized that fragmentation is usually not a good thing (it can cause other problems), and so most TCP implementations attempt to avoid fragmentation. NFS, on the other hand, uses fragmentation as a matter of course; if you want to run NFS over a *screend*-equipped router (probably a bad idea anyway) you might want to adjust the NFS mount options to reduce the read and write buffer sizes.

2.7. Reliability of information in packet headers

Screend makes all its decisions based on the contents of packet headers. Strictly speaking, this is completely unsound, because any field in a packet header might be a forgery.

In practice, the situation is not so bleak, since it is pointless to ‘‘forge’’ certain of the fields in the packet. For example, the destination IP address field is reliable, because once the packet arrives at *screend* there is no question about where the packet will end up. Similarly, the IP protocol type (e.g., TCP or UDP) and the destination port numbers are reliable.

On the other hand, any information about the source of the packet is suspect. One should not base security upon policies that depend upon the validity of source addresses (or ports), unless you can be sure that the source addresses cannot have been forged.

For example, one should not believe source addresses in packets that come from outside your organization. On the other hand, if you trust the computers inside your organization, it might be safe to trust source addresses for those hosts . . . but only in cases where the destination address is outside your organization. To be on the safe side, we recommend that you not normally trust source addresses; you should assume that some hosts inside your organization might have been compromised.

3. An Integrated Security Policy

It makes no sense to think about a network security policy in isolation; you have to understand how the individual systems in your organization are used, what their vulnerabilities are, what the likely threats are, who is trustworthy and who isn't. And, since security inevitably makes some benign activities less convenient (or impossible), one must also consider the tradeoffs between making an organization secure and allowing it to get its job done.

In this section, I will describe the general approach to security taken at the Digital Equipment Corporation Western Research Laboratory. This is not the only reasonable policy, but we have been relatively successful at balancing our security needs with our needs for communication. Discussion of specific IP/TCP issues will be deferred until section 4.

3.1. Basic principles

Our basic principle is that there shall be no direct communication (especially TCP connections, but also other using kinds of datagrams) between hosts within our organization and hosts outside our organization. There are exceptions to this policy, of course: certain of our hosts are ‘‘trusted’’ and these serve as relays for various functions. There are a few minor additional exceptions, described later.

Another principle is ‘‘separation of function.’’ Because we cannot be sure that any single function is actually secure, we create firewalls between functions so that if one is compromised, the rest are still intact. This means, for example, that we have no ‘‘user accounts’’ on mail relay machines, and the machine used as a *screend*-router has no other functions.

We are careful to log as much information as possible; this makes it much easier to detect intrusion attempts, and to (if necessary) understand how they succeed.

One useful principle in designing any kind of security policy is that the default should be to deny access; that is, everything which is not explicitly legal is illegal. While this may not be an appropriate model for structuring human societies, it is the safest way to structure a computer security system.

3.2. Organizational trust model

An important part of creating a security policy is to decide whom you trust. Each organization will have to make its own decision, based on the nature of its business, legal considerations, and its culture. Excessive paranoia causes resentment and inspires people to find ways around the security model; excessive trust leads to breakins.

One also has to distinguish between trusting certain people, and trusting the computers that they use. Most people are trustworthy, and most people are incompetent at securing their own personal systems . . . so most computers are not trustworthy. That is, if an intruder can log in to a computer, that computer could become the agent of an attack without the knowledge of its owner.

Our own model is that we trust no computers outside our organization. Inside our organization, we assume that people are trustworthy but that their computers might have been compromised. This is why we do not allow direct TCP connections across the boundary.

A small set of computers within the organization are deemed trustworthy. These computers must be actively and competently managed according to a set of security principles designed to limit the possibility that they will be compromised. For more information on how keep an individual host secure, you should turn to the published literature pertaining to the operating system you run. For example, Unix™ security has been covered fairly well [3, 5, 6, 15]. On the other hand, it might be impossible to make an MS-DOS system secure enough, and so you should not consider such a system ‘‘trustworthy.’’

It is important to understand that securing your connection to the Internet does not magically protect your organization from all threats. For example, an employee of your organization might mail a magnetic tape to an outsider, or an intruder might break into your building at night and steal a disk drive. The point of enhanced network security is not to stop all security problems; it is to make sure that someone outside your network administration takes the blame for them.

3.3. Threats

A security policy must be based on an estimate of the kinds of threats one is expecting. More precisely, since one can never anticipate all potential threats, the policy should foreclose large classes of threats, rather than being aimed at stopping only the ones you already know about.

We are most concerned about intruders breaking in to our systems via the network. Since we may fail at that, we also want to prevent them from stealing large quantities of information via the network. (It would be impossible to prevent an intruder from stealing small secrets, since they could simply be printed on the intruder’s terminal.)

We would also like to prevent intruders from disrupting our internal communications. This means, for example, that we do not want to allow someone to insert incorrect routing information into our routing tables.

Other kinds of denial-of-service attacks are based on overloading our internal network (and thus preventing legitimate uses). *Screend* by itself cannot rate-limit a packet flow, but by banning direct access to our network and by forcing all incoming traffic to flow through a relatively low-bandwidth pipe, *screend* allows us to localize the potential for mischief.

Because a security policy is inevitably a compromise between security and convenience, it would be a mistake to try to guard against every conceivable threat. Some threats might not be serious enough to worry about; for example, a path for stealing files that does not provide much bandwidth. Others may be so inconvenient that an intruder would not bother to use them. Finally, one must remember that in most organizations, employees expect their communications will remain private, and it would not be ethical (or legal) to monitor the contents of, say, electronic mail messages without some probable cause for suspicion of a particular user.

3.4. Structural model

Our screening router divides the Internet, from our point of view, into two pieces: ‘‘inside’’ and ‘‘outside.’’ In general, outside hosts are assumed under the control of skillful and malicious people. Inside hosts are generally assumed to be under the control of incompetent people, who must be protected against the barbarians outside.

In addition to these two large classes of hosts, there is a small set of trusted hosts. Trusted hosts are those that belong to our organization, and are carefully managed and monitored by skilled and cautious administrators. Some of these trusted hosts are outside the screening router (although still physically under our control); others may be inside. Between the trusted-but-outside hosts and the actual Internet, there is at least one more router, so that the physical network to which these hosts are connected to is also entirely under our physical control.

The trusted-but-outside machines serve as our relay machines. Since nothing stands between these machines and the outside world, they are the weak link in our security scheme. But, since they would have to be accessible from outside even if they were placed inside, they would be no less vulnerable on the inside. We believe that it is safer to place them on the outside, so that we can use *screend* to control how much access they have to internal hosts. Also, if one of them were to be compromised, placing them outside means that the intruder would not have direct access to one of our internal networks (where it might be possible to ‘‘snoop’’ for passwords and the like).

It should be clear that to protect an organization using this model, there must be no ‘‘sneak paths’’ around the *screend* router. Often, in a large organization, people are tempted to create their own paths to the outside world, either through ignorance of the security model or through frustration with the service they are getting. For example, someone with a home computer might set up a SLIP or PPP link to their office system; this would be fine by itself, but if they also have a SLIP link to, say, a neighboring University, their home computer would be a potential sneak path. This problem must be solved through administrative mechanisms, although careful monitoring of routing tables can help identify sneak paths.

Large organizations might require multiple connections to the Internet, for reasons of performance or reliability. *Screend* should be used at all such points, with coordinated configuration files, so that no inconsistencies arise.

3.5. Meeting organizational requirements for communication

The easiest way to secure a network against external threats is to disconnect it from the Internet, but that would be pointless: we have internetworks so that we can communicate. What functions are necessary to support our organization?

3.5.1. Electronic Mail and USENET

Electronic mail (carried by the SMTP protocol [16]) is the most important use of the Internet. Many organizations get by with electronic mail access and nothing else (if only because they cannot afford a direct connection).

Electronic bulletin-board systems are quite similar to electronic mail. In the Internet, the dominant bulletin-board system is called ‘‘USENET’’ or ‘‘netnews’’ or ‘‘news’’; articles are exchanged over IP/TCP networks using the NNTP protocol [7]. USENET follows a ‘‘flooding model’’: every participating server host ends up with the full set of articles. This is unlike certain other systems, where an article is stored by only one server, and the client must communicate over the network to that server.

Since both electronic mail messages and USENET articles can be relayed through a trusted machine, we do not need to let every host inside our organization exchange mail or articles with every host outside the organization. So, we can support mail and USENET without violating our policy of ‘‘no direct connections except through trusted relays.’’

This policy is particularly relevant because a bug in the original implementation of the 4.2BSD *sendmail* daemon creates a serious security hole (and was exploited by the ‘‘Morris Worm’’ [19, 22].) Unless you are sure that none of the *sendmail* daemons within your organization have this bug, it is best not to let anyone test this premise. Using a mail-relay host means that you need only worry about the security of its *sendmail* daemon.

3.5.2. Telnet and FTP

IP supports a number of other services that people have come to depend upon. Within an organization, these are certainly necessary; between organizations, we have found that they are not. Or, rather, our organization has decided that we would rather not risk the security hazards.

The most prominent of these services are Telnet [18], FTP [17], and NFS [20, 26] Telnet provides direct interactive access to a remote system, and we believe that it is too dangerous to allow outsiders such access to our systems. We use a relay machine for Telnet access, which means that only people authorized to use the relay machine can make telnet connections. We monitor the relay machine closely, to ensure that it is not compromised. We believe that this is much safer than trying to ensure the security of the thousands of machines inside our organization, although it does cause some administrative hassles for those people who want accounts on the relay machine.

Our approach to FTP is similar; we only allow FTP access via the relay machine, for specifically authorized users. The relay machine does not run an FTP server (except for read-only ‘‘anonymous’’ access to a restricted-root file system, containing public information). This means that a user must first log in to the relay machine, and then invoke the FTP client twice, in order to transfer files from outside to inside (or vice versa). This is cumbersome, but it guarantees that if the relay machine is secure, no unauthorized FTP can take place.

Many members of our organization, outside the set of authorized relay users, would like to have access to the many ‘‘anonymous FTP’’ sites throughout the Internet. We don’t dare to provide them direct FTP access to these sites, and we don’t want to give everyone an account on the relay machine, but fortunately there are a number of ‘‘FTP by email’’ services running on the Internet. To use this service, one sends a mail message asking it to retrieve a specific file from a specific anonymous FTP site, and the server returns the file via electronic mail.

Because FTP-by-email cannot be used to steal files from within our organization, it does not reduce our security.

We also run an anonymous FTP site for use from hosts both inside and outside our organization. Just as every such service should be, ours is set up so that the files are read-only, and a user cannot see any files (such as `/etc/passwd`) outside the publicly readable file system. We do not let even authorized users of the FTP relay host add files to this archive, in case someone (accidentally or on purpose) tries to export a confidential file through this service. Instead, each exported file must be approved by the administration of our anonymous FTP service.

3.5.3. NFS

We do not allow any direct file access protocols to run across our screening router. Current protocols, such as NFS, are simply too insecure to be trusted in an inter-organizational environment.

There is one exception to this rule: we allow certain inside-hosts to be NFS clients of the anonymous FTP server, which exports its public files as a read-only file system. We believe this is safe because it does not expose any secret files, does not allow any files to be modified, and does not depend on validating IP addresses that are not under our control.

3.5.4. R-commands

Although not described by any IP standards, the so-called ‘‘r-commands’’ introduced by 4.2BSD have become quite popular in certain circles. These commands, including *rlogin*, *rsh*, *rcp*, and *rdist* allow remote access between systems without any passwords being required. This makes them quite convenient, and completely insecure. We do not allow these services to cross our screening router, and we do not run the corresponding servers on our trusted systems.

3.5.5. Name service

Name service (translation between host names and addresses) in the Internet is provided by the Domain Name System (DNS) [9, 10]. DNS is a distributed, hierarchically-organized system in which each organization runs a set of DNS servers for its subtree of the name space.

Because it is useful for insiders to be able to translate names for outside hosts, and vice versa, we allow DNS lookup packets to transit the screening router (UDP packets only; DNS allows lookups to be done via TCP connections, but this is not a necessary service and we do not like to allow direct TCP connections).

We do not allow so-called ‘‘zone transfers’’ to hosts outside of our organization. These would allow someone to obtain a list of all the hosts in our organization, which might be useful information to someone trying to find a vulnerable host (or to someone trying to discover the organizational structure of our company, which is considered proprietary information). This prohibition cannot be entirely enfored by *screend*; if you choose to run a DNS server that is reachable from the Internet, that DNS server software must also reject such transfers.

3.5.6. Time service

We are increasingly dependent on the accuracy of the clocks in our computers, not just because our applications care what time it is, but because many aspects of security depend on the reliability of clocks. For example, the Kerberos authentication system requires that clocks be kept reasonably synchronous, and many similar systems depend upon clock monotonicity (i.e., that the clocks never run backward).

It is now common practice to synchronize clocks using a network-based protocol. The standard protocol in the Internet is the Network Time Protocol (NTP) [8], although several other time protocols are in use or have been proposed.

NTP includes several mechanisms to avoid being misled by malicious (or simply malfunctioning) clocks. The most robust of these is the use of encryption to provide authentication, but for various reasons many NTP servers do not take advantage of this feature. NTP is also careful to combine data from several sources in such a way as to ignore servers that are significantly out of synch. We believe that by taking care in selecting the servers at the top of our NTP distribution tree, we can avoid being plagued by the insertion of incorrect time information into our network. Thus, although we allow free flow of NTP packets across our screening router, we configure our NTP systems to trust only a small set of hosts.

3.5.7. X service

The X Window System [21] uses TCP to allow client programs on one system to use an X server (display) on another. We often get requests to allow X connections across the screening router, since these would allow demonstrations and powerful forms of collaboration. We do not allow X connections, however, because of the lack of any useful security mechanism in X.

For example, if we were to allow X clients (applications) from outside hosts to create windows on X servers (displays) on inside hosts, a miscreant might set up a display that looks like a standard login window, and thereby obtain passwords from unwitting users. If we allowed X clients on inside hosts access to X servers on outside hosts, an intruder could obtain interactive access to one of our hosts through the use, for example, of a Trojan horse program that simply starts an *xterm* process. In any event, to allow X connections in either direction would mean violating our rule against allowing direct TCP connections.

3.5.8. ICMP

Certain ICMP messages (such as Destination Unreachable, Redirect, and Source Quench) are considered necessary to the proper operation of a connection. Since we do not allow direct connections across our screening router, however, we need not allow ICMP messages. In fact, since some ICMP messages can actually be dangerous (bogus Redirects, for example, could create problems) we find it safest to prohibit all of them.

3.5.9. Finger

‘‘Finger’’ is a protocol used for finding out information about remote users (such as their phone number, address, and perhaps a short ‘‘plan’’ file giving additional information) [27]. The main reason to disallow the use of Finger is that a bug in the original implementation of the 4.2BSD Finger daemon creates a serious security hole (and was exploited by the ‘‘Morris Worm’’ [22, 19].) Unless you are sure that none of the Finger daemons within your organization have this bug, it is best not to let anyone test this premise.

Another reason to avoid direct Finger connections is that the protocol provides an easy way to steal large files. An intruder who has penetrated one of your machines could put a file to be stolen in place of the finger ‘‘plan’’ file of a legitimate user, and then finger that user from outside; the finger daemon would blindly transfer the entire file. (It would be possible to modify your finger daemon to allow only short transfers, but you probably would not be able to update all the vulnerable hosts.)

We use a ‘‘finger relay’’ mechanism that allows our users to finger people in other organizations. It also allows us to control, from a single point, incoming finger transactions; we can limit the set of fingerable users, and we can limit how much information can flow.

3.5.10. Other protocols

Many other protocols are in use in the Internet. We prohibit them all.

3.6. Exceptions

Because security and convenience always conflict, there are times when exceptions to these rules are necessary. (A security system that is too inflexible inspires people to work around it, and so it becomes ineffectual.) Exceptions, when made, should be as narrow as possible; *screend* supports this.

For example, one might decide that a particular external host, even one belonging to another organization, is trustworthy enough to allow connections to its FTP server. Or, for the purposes of a particular demonstration, one might temporarily allow X clients on a specific internal host access to the X server on certain external hosts. One should then make sure that the internal host in question is carefully managed and monitored.

4. IP/TCP security rules

In this section, I will give a set of rules that describe how packets are allowed to flow across the screening gateway, given the security policy described in section 3. To make the rules concrete, I will use as an example the *Yoyodyne, Inc.* organization shown in figure 1.

In figure 1, *screend* is running on *Router-B*. Both *Subnet-A* and *Subnet-B* are subnets of *Yoyodyne-net*, as are the ‘‘other internal networks.’’ The Internet is full of malicious characters (as the frowny-face suggests). *Subnet-A* is ‘‘outside’’ the screening router, but is still under the control of Yoyodyne.

We will allow hosts on *Subnet-B* to mount a read-only NFS file system from *Telnet-FTP-Relay*, but we will not allow other hosts in Yoyodyne to do so. We also do not allow *Host-D* to mount this NFS file system (this is a contrived example, whose purpose will become clear later on).

Yoyodyne has decided to allow its employees to finger people at other organizations, but not to allow anyone to finger its own users except for those who have accounts on *Telnet-FTP-Relay*.

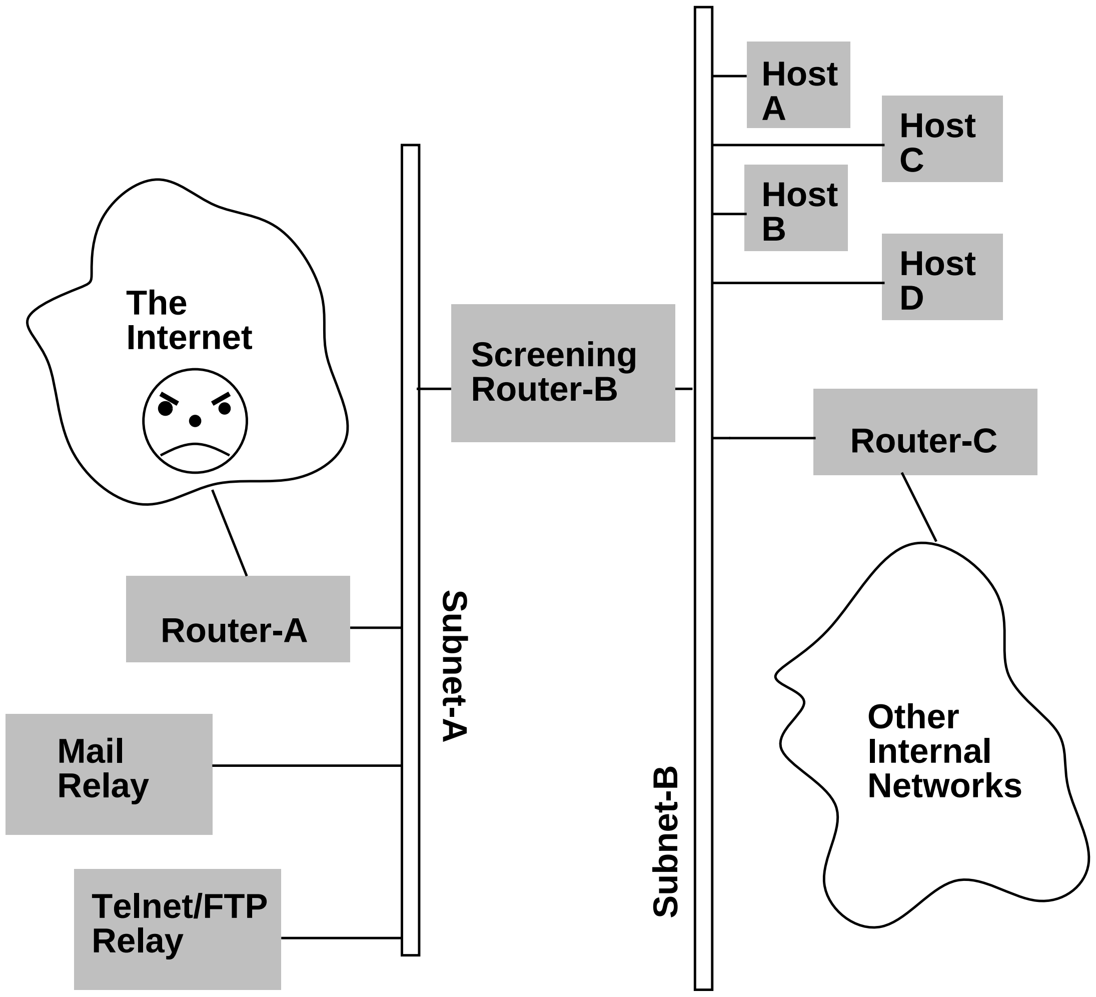

Figure 1: The *Yoyodyne* organization

Since one of our principles is that the default is to deny access, and we allow relatively few kinds of access across the screening router, we need only a few rules to describe our screening policy.

1. The host *Mail-Relay* is allowed to be client or server for SMTP and NNTP connections with any host.
2. The host *Telnet-FTP-Relay* is allowed to be client or server for Telnet and FTP connections with any host.
3. The host *Telnet-FTP-Relay* is allowed to be server, but not client, for Finger connections with any host.
4. The host *Telnet-FTP-Relay* is allowed to exchange UDP packets to or from its NFS server port with hosts on *Subnet-B*, except for *Host-D*.
5. Any UDP packets to or from the DNS (name server) port are allowed between any hosts.

6. Any UDP packets to or from the NTP (time service) port are allowed between any hosts.
7. All other packets are prohibited.

These few rules are sufficient to formalize the screening-router's responsibilities in enforcing the security policies described in section 3. They are not, of course, sufficient to enforce the responsibilities of the hosts *Mail-Relay* and *Telnet-FTP-Relay*. It is our hope that, if these responsibilities are enforced, no amount of incompetence in the administration of hosts on *Subnet-B* or the other internal networks of Yoyodyne, Inc., will expose the company to attack.

5. Implementing a policy using *screend*

In this section, I will show how to use *screend* to implement the rules listed in section 4.

5.1. *Screend* theory of operation

The mechanisms used in the *screend* system are documented in great detail elsewhere [11, 12]. The underlying idea, however, is quite simple.

Normally (i.e., without *screend*), forwarding of IP packets is done entirely in the ULTRIX kernel. When a packet is received, and it is not destined for one of the local host addresses, the kernel attempts to forward it. Forwarding means finding a route for the packet to follow, and transmitting the packet.

When *screend* is in use, the forwarding process starts with one new step. The kernel passes the packet header to the *screend* process, which examines the header and decides if the packet should be forwarded (‘‘accepted’’) or not (‘‘rejected’’). *Screend* returns its decision to the kernel, which then either continues with the normal forwarding process or discards the packet, according to *screend*'s instruction. Optionally, *screend* may instruct the kernel to send an ICMP Destination Unreachable message to the source of a rejected packet; this is done to try to suppress additional attempts to send forbidden packets.

5.2. The *screend* configuration file

The *screend* program is instructed to enforce a particular policy by the rules it finds in its configuration file, `/etc/screend.conf`. To change the policy, you simply edit this file, kill the `/usr/etc/screend` process currently running, and restart `/usr/etc/screend`.

The configuration file is a text file; this makes it easy to edit, but after editing the file you should make sure that *screend* parses it without complaint, since it is possible that you might have introduced syntax errors.

5.3. Grammar for the screend configuration file

This is a guide to the grammar of the ***screend*** configuration file. Syntax rules are expressed in BNF notation (sometimes called Backus-Naur Form). If you don't know what this is, you should turn to a textbook on programming languages or compilers.

If you are confused by this notation, try turning to section 5.4, which shows examples of specific rules.

5.3.1. Lexical structure

- **Comments** can either be ''C-style'' comments, delimited by ''`/*`'' and ''`*/`'', or ''csh-style'' comments begun with ''#'' and terminated by the end of a line. Comments do not nest.
- **Case** is significant in reserved words (all are lower-case). This is actually a benefit, because if a host name happens to conflict with a reserved word, you can use the host name in upper-case.
- **Host names** begin with alphabetics but may contain digits, '-', '.', and '_'. The same is true of network, subnet, and netmask names. All can also be entered in dotted quad notation (for example, ''10.1.0.11'').
- **Numbers** may be in decimal or in hex (0x0 notation). Octal notation is not allowed because nobody uses it in this context. (Actually, hex is almost as useless).
- **Protocol names** and **port names** (for TCP or UDP) are as in `/etc/protocols` and `/etc/services`, respectively. These can also be given as numbers (host byte order).
- **ICMP type codes** must be chosen from this list, or given as numbers:

```
echoreply                      timestamp
unreachable                    timestampreply
sourcequench                   informationrequest
redirect                       informationrreply
echo                           addressmaskrequest
timeexceeded                   addressmaskreply
parameterproblem
```

- **All white space** is the same (including newlines).

5.3.2. Syntax

General syntax rules:

1. The configuration file consists of ''specifications'' terminated by semicolons.
2. There are three kinds of specifications:
   a. **default action specification**: There should only be one of these (the last one is the one that counts); it specifies what action to take if no action specification matches a packet.
   b. **subnet mask specifications**: specifies the subnet mask used for a given network.

c. **action specifications**: specifies a class of packets and the action to take when such a packet is received.

3. Specifications can appear in any order, but the evaluation order of action specifications is the order in which they appear in the file.

In BNF, this is:

<*configuration-file*> ::= { <*specification*> | <*configuration-file*> <*specification*> }
<*specification*> ::= { <*default-action*> | <*subnet-spec*> | <*action-spec*> }

The syntax for a default action specification is:

<*default-action*> ::= **default** {**accept** | **reject**} [**notify**] [**log**] **;**

Note that ‘‘default accept notify;’’ is legal but the ‘‘notify’’ in this case is a no-op. If not specified, the default action is ‘‘reject’’.

The syntax for subnet mask specifications is:

<*subnet-spec*> ::= **for** <*network*> **netmask is** <*maskval*> **;**

The <*network*> is either a network name or a dotted-quad address, such as ‘‘36.0.0.0’’. ‘‘36’’ is *not* a reasonable value. <*Maskval*> is either a name (treated as a hostname) or a dotted-quad address, such as ‘‘255.255.255.0’’ (bits are *on* for the network and subnet fields.)

The syntax for action specifications is:

<*action-spec*> ::= **from** <*object*> **to** <*object*> {**accept** | **reject**} [**notify**] [**log**] **;**

Such a specification says that packets flowing this way between this pair of ‘‘objects’’ (defined below) should either be accepted or rejected. If ‘‘notify’’ is specified, when a packet is rejected an ICMP error message is returned to the source. If ‘‘log’’ is specified, this packet and its disposition are logged.

Conceptually, for each packet the action specifications are searched in the order they appear in the configuration file, until one matches. The specified action is then performed. If no specification matches, the default action is performed.

To simplify the configuration file, the syntax

<*action-spec*> ::= **between** <*object*> **and** <*object*> {**accept** | **reject**} [**notify**] [**log**] **;**

may be used to indicate that the same action should be performed on packets flowing in either direction between the specified pair of ‘‘objects.’’ Note that this is simply syntactic sugar; it has the same effect as specifying the two unidirectional rules, with the ‘‘forward’’ direction listed first.

An ‘‘object’’ is a specification of the source or destination of a packet. The syntax for object specifications is somewhat complex, since certain fields are optional:

<*object*> ::= { <*address-spec*> | <*port-spec*> | <*address-spec*> <*port-spec*> }

If the <*address-spec*> is not given, ‘‘any host’’ is assumed. If the <*port-spec*> is not given, ‘‘any protocol and port’’ is assumed.

<*address-spec*> ::= { <*net-spec*> | <*subnet-spec*> | <*host-spec*> | **any** }

<net-spec> ::= { **net** *<name-or-addr>* | **net-not** *<name-or-addr>* }
<subnet-spec> ::= { **subnet** *<name-or-addr>* | **subnet-not** *<name-or-addr>* }
<host-spec> ::= { **host** *<name-or-addr>* | **host-not** *<name-or-addr>* }

The ‘‘-not’’ convention means that the object specification matches if the specified field does *not* have the specified value. For example, ‘‘`from host-not sri-nic.arpa to host any reject`’’ means that packets *not* from `sri-nic.arpa` are dropped. The ‘‘subnet’’ and ‘‘subnet-not’’ forms match against the entire address under the subnet mask (for example, if the netmask for net 36 is 255.255.0.0, then ‘‘`subnet 36.8.0.0`’’ matches a packet address of 36.8.0.1).

<name-or-addr> ::= { *<name>* | *<dotted-quad>* | **any** }

<port-spec> ::= { **proto** *<proto-name-or-number>*
| **icmp type** *<type-name-or-number>* | **icmp type-not** *<type-name-or-number>*
| **tcp port** *<port-name-or-number>* | **tcp port-not** *<port-name-or-number>*
| **udp port** *<port-name-or-number>* | **udp port-not** *<port-name-or-number>* }

<proto-name-or-number> ::= { *<name>* | *<number>* }
<type-name-or-number> ::= { *<name>* | *<number>* | **any** | **infotype** }
<port-name-or-number> ::= { *<name>* | *<number>* | **any** | **reserved** }

‘‘Reserved’’ ports are those reserved by 4.2BSD Unix for privileged processes. ‘‘Infotype’’ ICMP packets are those that are purely ‘‘informational’’: echo, timestamp, information, and addressmask requests, and the corresponding replies.

5.4. Implementing the example of section 4

The first step in creating a `screend.conf` file is to tell *screend* the subnet structure of Yoyodyne’s network. I will assume that *yoyodyne-net* is a Class-B network with an 8-bit wide subnet field

```
for yoyodyne-net netmask is 255.255.255.0;
```

Now I will create *screend* rules corresponding to the informal rules in section 4. Note that *screend* interprets the rules in the order that they appear in the configuration file, so if two rules conflict I will put the more specific rule first.

1. The host *Mail-Relay* is allowed to be client or server for SMTP and NNTP connections with any host.

```
between host mail-relay tcp port smtp
        and any accept;
between host mail-relay
        and any tcp port smtp accept;
between host mail-relay tcp port nntp
        and any accept;
between host mail-relay
        and any tcp port nntp accept;
```

2. The host *Telnet-FTP-Relay* is allowed to be client or server for Telnet and FTP connections with any host.

```
between host telnet-ftp-relay tcp port telnet
        and any accept;
between host telnet-ftp-relay
        and any tcp port telnet accept;

between host telnet-ftp-relay tcp port ftp
        and any accept;
between host telnet-ftp-relay
        and any tcp port ftp accept;
between host telnet-ftp-relay tcp port 20
        and any accept;
between host telnet-ftp-relay
        and any tcp port 20 accept;
```

Note that if one wants to allow FTP data transfers, one must allow access to the FTP-Data port, because of the way that the FTP protocol works. Since the FTP-Data port is not normally listed in `/etc/services`, we must give its numeric value. (See section 5.5 for more about using numbers instead of symbolic names.)

3. The host *Telnet-FTP-Relay* is allowed to be server, but not client, for Finger connections with any host.

```
between host telnet-ftp-relay tcp port finger
        and any accept;
```

4. The host *Telnet-FTP-Relay* is allowed to exchange UDP packets to or from its NFS server port with hosts on *Subnet-B*, except for *Host-D*.

   Since the rule about *Host-D* is an exception to the rule about *Subnet-B*, we must put it first.

```
between host telnet-ftp-relay udp port nfs
        and host host-d reject;
between host telnet-ftp-relay udp port nfs
        and subnet subnet-b accept;
```

A NFS client must also be able to contact the *mountd* process on the server host. Normally, the *mountd* process does not used a fixed port number (it registers itself with the *portmap* process, which does have a fixed port number.) In order to have a specific port number to put into the *screend* configuration file, we need a special version of the *mountd* program that binds to specified port number. We have arbitrarily chosen port number 3210 for this purpose.

```
between host telnet-ftp-relay udp port 3210
        and host host-d reject;
between host telnet-ftp-relay udp port 3210
        and subnet subnet-b accept;
```

5. Any UDP packets to or from the DNS (name server) port are allowed between any hosts.

```
between any udp port domain
        and any accept;
```

6. Any UDP packets to or from the NTP (time service) port are allowed between any hosts.

```
between any udp port ntp
        and any accept;
```

7. All other packets are prohibited.

```
default reject;
```

Screend rules let one specify more than simply whether to accept or reject the packet. In most cases, if we reject a packet we want to have *screend* notify the sender (via an ICMP message). We probably want to log most, but not all, of the rejected attempts, and we might want to log some of the accepted packets (see section 5.7).

The following example shows the same rules as above, collected together into a single commented `screend.conf` file, with `notify` and `log` sprinkled throughout. I have added a few extra rules to suppress logging of rejections of uninteresting, but common, broadcast packets. Note: some earlier versions of the *screend* software only accepted the `default` rule if it comes as the first rule in the file; that is why it appears first in this example.

```
# All packets not covered by an explicit rule
# are prohibited, and logged.
default reject notify log;

# Specify any necessary network masks
for yoyodyne-net netmask is 255.255.255.0;

# The host Mail-Relay is allowed to be client or server
# for SMTP and NNTP connections with any host.
between host mail-relay tcp port smtp
        and any accept;
between host mail-relay
        and any tcp port smtp accept;
between host mail-relay tcp port nntp
        and any accept;
between host mail-relay
        and any tcp port nntp accept;

# The host Telnet-FTP-Relay is allowed to be client or
# server for Telnet and FTP connections with any host.
between host telnet-ftp-relay tcp port telnet
        and any accept;
between host telnet-ftp-relay
        and any tcp port telnet accept;

between host telnet-ftp-relay tcp port ftp
        and any accept;
between host telnet-ftp-relay
        and any tcp port ftp accept;
# Note that if one wants to allow FTP data transfers,
# one must allow access to the FTP-Data port, because
# of the way that the FTP protocol works.
between host telnet-ftp-relay tcp port 20
        and any accept;
between host telnet-ftp-relay
        and any tcp port 20 accept;

# The host Telnet-FTP-Relay is allowed to be server, but
# not client, for Finger connections with any host.
between host telnet-ftp-relay tcp port finger
        and any accept;
```

```
# The host Telnet-FTP-Relay is allowed to exchange UDP
# packets to or from its NFS server port with hosts on
# Subnet-B, except for Host-D.
between host telnet-ftp-relay udp port nfs
        and host host-d reject notify log;
between host telnet-ftp-relay udp port nfs
        and subnet subnet-b accept;
# A special version of mountd, running on port 3210, is
# used to allow us to list a specific port number here.
between host telnet-ftp-relay udp port 3210
        and host host-d reject;
between host telnet-ftp-relay udp port 3210
        and subnet subnet-b accept;

# Any UDP packets to or from the DNS (name server) port
# are allowed between any hosts.
between any udp port domain
        and any accept;

# Any UDP packets to or from the NTP (time service) port
# are allowed between any hosts.
between any udp port ntp
        and any accept;

# Reject high-volume broadcast junk without wasting
# log space
from any to any udp port whod reject notify;
from any to any udp port timed reject notify;
```

5.5. Use of symbolic names in configurations

In the Yoyodyne example, I used symbolic names for hosts, networks, subnets, and ports; although *screend* allows either symbolic names or numbers for all these objects, it is much easier to understand the example if names are used.

Using names has its pitfalls, however. The most important is that it depends upon the security of your name service. If someone has compromised your name server (or a non-local name server, if you are brave enough to include the name of a non-local host in one of your rules) then they may be able to fool your *screend*. For this reason, we translate all names into numbers by hand, and put only numbers into the `screend.conf` file. (This does run the risk of human error, of course, and it can lead to problems when a name-to-address binding is changed.)

Another problem is that some host names are bound to more than one address. If you specify the rules using names, rather than numbers, *screend* does not know what you mean. It will pick the primary host address, and issue a warning message. If you use host numbers, no ambiguity is possible.

5.6. Performance considerations

In order to provide consistent and understandable behavior when `screend.conf` contains conflicting rules, for each packet *screend* searches the rule database in the order that the rules appear. This means that you should put the rules most likely to match an incoming packet at the front of the configuration file. It also means that you should not put more rules than necessary into the file, since that will lengthen the search process.

Screend avoids some of the searching by keeping a cache of a small number of recent decisions. You cannot do much to affect the performance of this cache, except for keeping your rules as simple as possible.

Since the *screend* process is invoked for every packet received for forwarding, it is not a good idea to run other active jobs on the system. You should, of course, continue to run the normal bookkeeping jobs (such as *syslog*, *cron*, *update*, etc.) but you should refrain from running nameservers or user applications. It is probably more important to run *screend* alone on a dedicated system than it is to run *screend* on a fast CPU, because switching between processes is one of the least efficient functions of modern computers.

5.7. Logging of actions

We log all failed access attempts; this allows us to detect intrusions, or problems arising from incorrect configuration (such as incorrect use of MX records). *Screend* can be set up to log via the *syslog* service, or directly to a file. The latter mechanism is much more efficient, and produces slightly less log output; the former includes timestamps on every log record. In most cases, the timestamps can be inferred from the hourly summary records written to the log.

If *screend* were to log a record for every packet, it would run like molasses and your disks would fill up. One can limit the log volume by not logging accepted packets (since those *should* be far more common than rejected packets), and by not logging certain common kinds of rejected packets (see the example in section 5.4).

Screend also helps out by suppressing duplicate log records. Since repeated attempts to communicate over a given path would give rise to identical log entries (except for the timestamp), *screend* immediately issues the first of a series of identical records, and then simply counts the number of repetitions. Every 10 minutes or so, *screend* logs a summary entry showing how many superfluous entries have been suppressed. The program has limited buffer space to keep suppressed log entries, so if this space overflows then additional summary entries will be generated as necessary, in least-recently-used order. The net effect is that you can tell exactly how many packets of each sort would have been logged, although not until a few minutes after the fact and not in precise time-order.

Our *screend* router is on a fairly busy path, and we find that it generates on the order of 360000 log entries (20 megabytes) every day. We use a script similar to `/usr/adm/syslog` to compress and rename the log file once a night, keeping the last 7 days of logs. (Compression can reduce the size of the old logs by a factor of 10 or more.) *Screend*, unlike *sendmail*, does not need to be sent a signal to cause it to use a new log file; it simply reopens its log file (in append-mode) once an hour. The script we use is:

```
#! /bin/sh
cd /usr/spool/log
/bin/rm screend.7.Z
/bin/mv screend.6.Z  screend.7.Z
/bin/mv screend.5.Z  screend.6.Z
/bin/mv screend.4.Z  screend.5.Z
/bin/mv screend.3.Z  screend.4.Z
/bin/mv screend.2.Z  screend.3.Z
/bin/mv screend.1.Z  screend.2.Z
/bin/mv screend.0    screend.1
/bin/mv screend      screend.0
/bin/cp /dev/null screend
/bin/chmod 644 screend
/etc/chown root screend
/bin/chgrp system screend
# We cannot yet compress screend.0 because it is still
# open for writing by the screend process
/usr/ucb/compress screend.1
```

This script is invoked once a day from `/etc/crontab`.

Log entries come in several forms:

- `REOPEN: Fri Apr 19 05:23:18 1991`

  Indicates that the program has reopened the log file

- `REJECTN: TCP [8.0.0.1]->[16.99.1.2](1044->25)`

  A TCP packet from 8.0.0.1, port 1044, to 16.99.1.2, port 25, was rejected.

- `SUMMARY: 4 of TCP [8.0.0.1]->[16.99.1.2](1044->25)`

  4 TCP packets from 8.0.0.1, port 1044, to 16.99.1.2, port 25, were rejected. (The `SUMMARY` entry does not say what action was taken, but that can be inferred from a previous log entry for the same path.

- `REJECTN: ICMP Quench [192.5.25.7]->[16.99.1.2]`

  An ICMP Source Quench message from 192.5.25.7 to 16.99.1.2 was rejected.

-
  ```
  STATS: 14492592 pkts (51% hits), 10555 drops:
              Fri Apr 19 14:23:18 1991
  ```

  `STATS` entries are issued once an hour. Since the program was started, 14492592 packets were processed. 51% of the decisions were made based on cached data, rather than a complete search through the rule database. 10555 packets were dropped (rejected) by the kernel because the *screend* program was too slow to keep up.

5.8. Debugging a *screend* configuration file

It is not always clear why *screend* is preventing communication that you thought it should allow (or allowing communication that you thought it should prevent). Often, running `tail -f` on the log file, while you try to send the problematic packets, will show what is going on. (Remember that the log-compression mechanism might delay some entries.)

If you are confused about which rule in the configuration file is responsible for rejecting a packet, you can ask *screend* to log the ‘‘rule number’’ of the rule that was used to decide the disposition of each packet, using the `-r` option when you start the program. Rules are numbered starting at zero; the default action does not have a rule number, but is distinguished in the log entries.

Because lines using the `between` syntax actually generate two rules, it can be confusing to figure out the rule numbers simply by reading the configuration file. One easy way to generate a listing of the rules with their rule numbers is to give the command

```
    /usr/etc/screend -c -d
```

The `-c` option says ‘‘check the configuration file but do not start the *screend* daemon’’; the *-d* option says to dump debugging information. In this case, the debugging information consists of the list of rules, with their numbers.

If you turn on logging of rule numbers, you will also get information about cache hits. Log entries with `+` before the rule number represent cache hits; those with `-` represent cache misses.

5.9. Starting *screend* at boot time

By default, when an ULTRIX system boots it does not pass packets through *screend*. To be secure, your system should require *screend* approval as soon as possible after booting. You should edit your `/etc/rc.local` file to contain commands similar to these at the very beginning:

```
    [ -f /usr/etc/screend -a -f /usr/etc/screenmode -a -f /etc/screend.conf ] && {
        /etc/screenmode on && {
            echo 'screenmode on - no forwarding until screend starts'>/dev/console
        } || {
            echo 'cannot set screenmode on, reboot failed' >/dev/console
                exit 1
        }
    } || {
            echo "cannot start screend, reboot failed" >/dev/console
    }
```

Later on in the file, after `/etc/syslog` has been started, you should put the commands:

```
    [ -f /usr/etc/screend -a -f /usr/etc/screenmode -a -f /etc/screend.conf ] && {
      /usr/etc/screend -L /usr/spool/log/screend & echo -n ' screend'>/dev/console
    }
```

You may, of course, choose a different location for the log file.

5.10. Other helpful hints

We try to make sure that the configuration file contains a comment showing who last edited it and when; this helps us when several people are authorized to make changes to the file and someone gets confused.

Comments in the configuration file do not affect the performance of *screend*; you should take advantage of this and use them to make it clear what your rules are there for.

We also use RCS (Revision Control System) to track changes to our configuration file. This provides information about who made changes in the past, and why. It also lets us recover older (working) versions of the file, if necessary.

We also keep the contents of the configuration file as confidential as possible. Nobody without a need to know has access to the file. ‘‘Security through obscurity’’ should not be the primary protection for your systems, but it is a useful backstop to other methods. For example, if you accidentally create a configuration file with a security hole in it, do you want to make it easy for an intruder to exploit the hole?

6. Further examples

In this section, I give some additional examples in which *screend* may be employed to enhance network security.

6.1. Limiting access to licensed software

The Biology department at Podunk University has licensed some expensive source code from a software company. Before the software vendor will deliver the source code, the department must demonstrate to the vendor’s satisfaction that the security precautions are adequate. They wish to be able to read mail on the protected network, and to make remote terminal connections in or out. They do not want to allow file transfer between the department and the rest of the campus. Because this is the Biology department and not the Computer Science department, they do not expect their users to be competent at securing their individual hosts.

The structure chosen to provide security is shown in figure 2. The Biology department has one subnet, connected to the rest of the campus (and the rest of the world) via a screening router. One machine in the department (App-server) has been designated the Application Server, which is then managed as carefully as possible. The screening router is configured to allow only that machine to communicate with the rest of the campus.

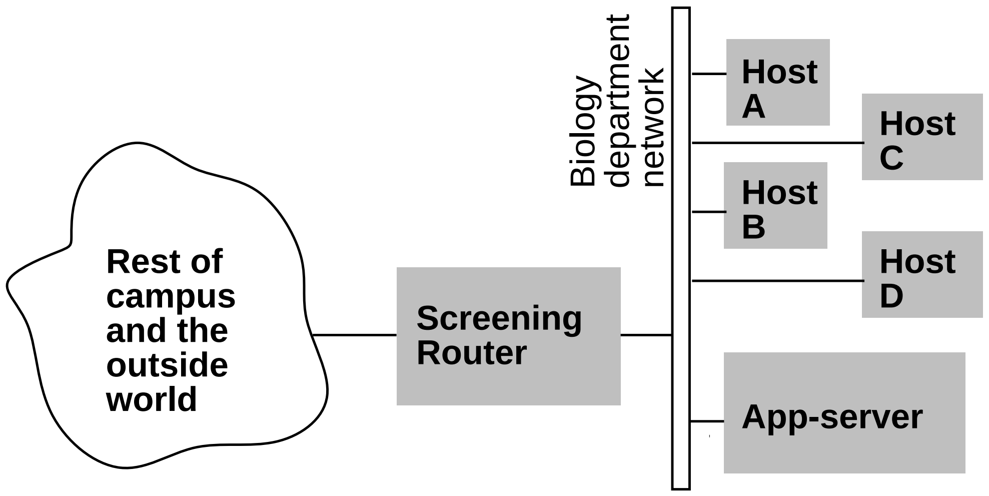
Rest of campus and the outside world
Screening Router
Biology department network
Host A
Host B
Host C
Host D
App-server

Figure 2: Podunk University Biology department

Every member of the department has an account on App-server. All mailboxes are on this server; all incoming and outgoing mail flows via this server. Telnet connections are allowed between the server and the rest of the world. This allows department members to access external machines (or access their own machine from outside) but forces them to log in to App-server along the way. *rlogin* connections are allowed from App-server to the outside world, but not in the opposite direction. Name service (but not ‘‘zone transfer’’) and time service packets are allowed ubiquitously.

Since the original purpose was to protect the licensed software, this scheme by itself is insufficient. It allows a user of a Biology department host to send anything out via mail. *Screend* cannot distinguish between ‘‘good’’ mail messages and ‘‘bad’’ ones, so by itself it cannot solve this problem. The restriction that mail must flow via the Application Server, however, means that a variety of methods might be used on that machine. For example, the mail system keeps logs of all message traffic; these logs could be checked to discover any unusually large messages. Of course, a software thief need not use the network to transfer stolen software, so ultimately the problem depends on careful administration of the machine where the software is stored.

The following *screend* configuration file is used in this situation:

```
# All packets not covered by an explicit rule
# are prohibited, and logged.
default reject notify log;

# Specify any necessary network masks
for podunk-net netmask is 255.255.255.0;

# The host App-server is allowed to be client or server
# for SMTP and Telnet
between host app-server tcp port smtp
        and any accept;
between any tcp port smtp
        and host app-server accept;
between host app-server tcp port telnet
        and any accept;
between any tcp port telnet
        and host app-server accept;

# The host App-server is allowed to be client, but
# not server, for rlogin connections with any host.
between host telnet-ftp-relay
        and any tcp port rlogin accept;

# Any UDP packets to or from the DNS (name server) port
# are allowed between any hosts.
between any udp port domain
        and any accept;

# Any UDP packets to or from the NTP (time service) port
# are allowed between any hosts.
between any udp port ntp
        and any accept;
```

6.2. Keeping junk packets off a slow link

This example shows how *screend* might be used to improve performance, instead of security. Figure 3 depicts the Lollipop General company, a large organizational internetwork connected to the Internet via a low-speed link (the company isn't selling as many lollipops as it used to, and cannot afford a T1 line). Suppose that the slow link is a subnet of Lollipop General's network.

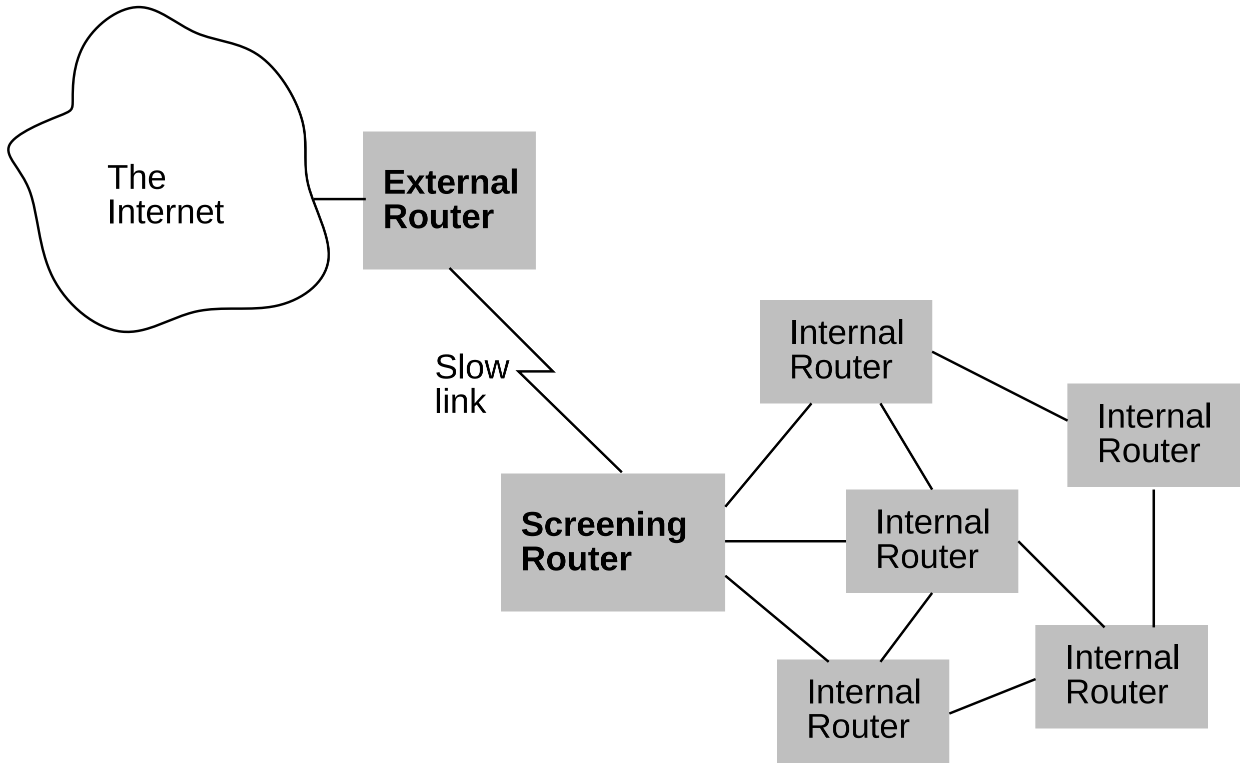

Figure 3: Organization connected to Internet via slow link

The various routers belonging to Lollipop General exchange routing information about the structure of the internal network, but they do not know the structure of the Internet. Instead, they have a default route to the External Router, which maintains a complete routing table for the Internet. The External router does not have a complete routing table for the internal network, but knows that the router at the other end of the slow link (marked ‘‘Screening Router’’ in figure 3) has the internal routing table.

The problem with this configuration is what happens when some host on the internal network attempts to send a packet to a non-existent subnet of Lollipop General's network (lollipop-net). The internal routers, not knowing a route to this subnet, will forward the packet to the External Router. The External Router, knowing that that the destination is a subnet of lollipop-net, forwards the packet to the Screening Router. These two routers then bounce the packet back and forth until its Time-To-Live field reaches 0, at which point it is dropped. By then, of course, the original host has probably sent another packet to the same destination, and the process repeats. The slow link is thus consumed by these useless packets, and valid external connections start to suffer.

The proper solution to this problem is either to provide the External Router with a complete routing table for lollipop-net, or to modify the lollipop-net internal routers to reject packets to non-existent subnets, or both. But let us suppose that, for technical and administrative reasons, neither approach is possible.

It turns out that *screend*, running at the Screening Router, can be configured to block the useless packets from ever crossing the slow link. This is done by banning packets between hosts on lollipop-net from being forwarded by the Screening Router, unless they start or end on the slow link itself. A packet to a non-existent subnet of lollipop-net will not fit this exception (unless its original source is the External Router; we will assume that this router does not make this mistake).

A *screend* configuration to keep junk packets off of the slow link might look like:

```
# All packets not covered by an explicit rule
# are allowed
default accept;

# Specify any necessary network masks
for lollipop-net netmask is 255.255.255.0;

# Packets between lollipop-net hosts and hosts
# on the slow-link subnet are allowed
between net lollipop-net and subnet slow-link accept;

# Other packets between lollipop-net hosts are not
# allowed to cross this router
between net lollipop-net
        and net lollipop-net reject notify log;
```

In this example, the order in which the rules appear is quite important. If the rule allowing packets to and from hosts on the slow-link subnet appeared after the rule banning packets between hosts on the entire lollipop-net, it would be impossible for an internal host to communicate directly with the External Router.

Note that by sending notification (i.e., ICMP Destination Unreachable messages) to the sources of bogus packets, we will probably prevent them from sending more than one at a time. The log of bad packets is useful in determining which hosts are improperly configured.

6.3. A simpler mail gateway configuration

In section 4, I explained a configuration for Yoyodyne, Inc., that included a mail gateway along with a lot of other gateway functions. If the only external access one wants to provide is electronic mail, then a simpler and somewhat more secure system can be configured, using two screening routers.

Figure 4 shows the network organization for the *SmallTime* company. The company's *Mail-Relay* is attached to *Subnet-X*, and is isolated from the Internet by *Screening-Router-A* and from the rest of SmallTime's network by *Screening-Router-B*. The *Mail-Relay* host needs SMTP, ICMP, and DNS access to both the Internet and SmallTime's network. One probably wants telnet access to the *Mail-Relay* from the internal network (for management purposes), but no other access needs to be allowed.

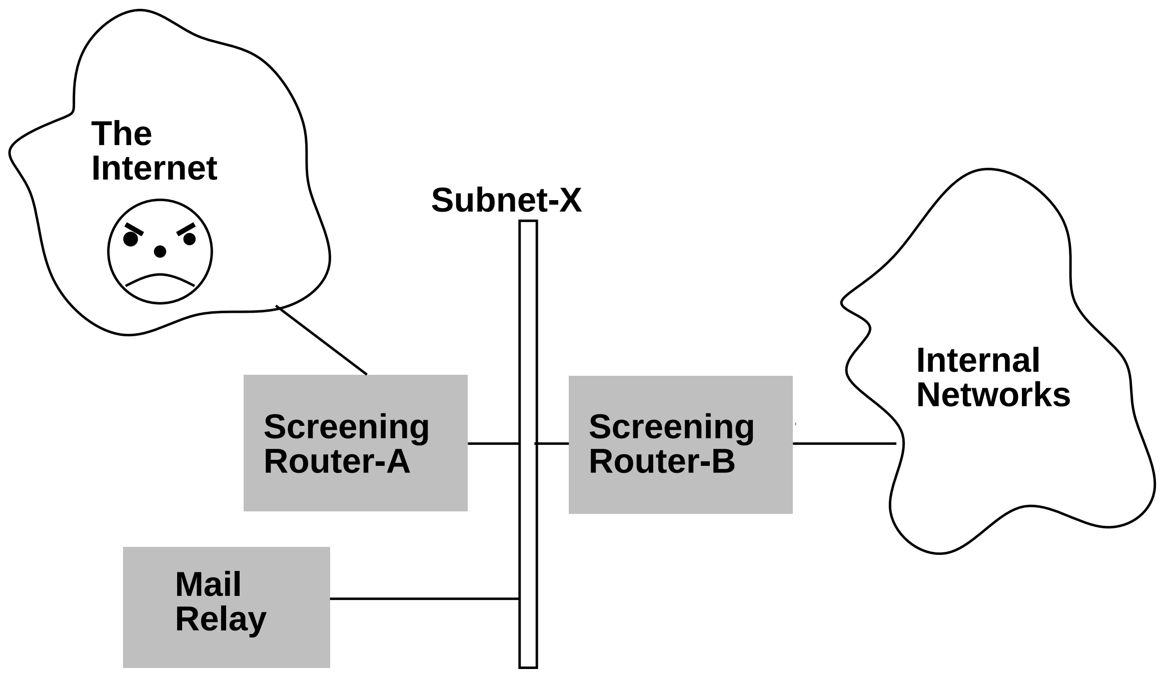

Figure 4: The *SmallTime* organization

To implement this policy, two *screend* configuration files are used. On *Screening-Router-A*, one would use:

```
# All packets not covered by an explicit rule
# are prohibited, and logged.
default reject notify log;

# The host Mail-Relay is allowed to be client or server
# for SMTP
between host mail-relay tcp port smtp
        and any accept;
between any tcp port smtp
        and host mail-relay accept;

# The host Mail-Relay is allowed to send and receive ICMP
# messages.
between host mail-relay
        and any icmp type any accept;

# The host Mail-Relay is allowed to send and receive DNS
# UDP messages.
between host mail-relay udp port domain
        and any accept;
between any udp port domain
        and host mail-relay accept;
```

On *Screening-Router-B,* one would use:

```
# All packets not covered by an explicit rule
# are prohibited, and logged.
default reject notify log;
```

```
# The host Mail-Relay is allowed to be client or server
# for SMTP
between host mail-relay tcp port smtp
        and any accept;
between any tcp port smtp
        and host mail-relay accept;

# The host Mail-Relay is allowed to send and receive ICMP
# messages.
between host mail-relay
        and any icmp type any accept;

# The host Mail-Relay is allowed to send and receive DNS
# UDP messages.
between host mail-relay udp port domain
        and any accept;
between any udp port domain
        and host mail-relay accept;
```

Note that the combination of these two configuration files prohibit any direct exchanges of packets between Internet and the internal network, since all the rules require at least one of the endpoints to be the *Mail-Relay* host. This should make it impossible for an outsider to break in to an internal host (provided that the screening routers are secure).

In practice, one might want to elaborate slightly on this configuration, for example to allow DNS zone transfers and FTP access to the *Mail-Relay* from the internal network, and perhaps to provide NTP access.

7. Acknowledgements

I would like to thank Paul Vixie and Brian Reid, both for reading early drafts of this document and also for their work on the DECWRL gateway complex. The expertise they have developed over the years in creating a security policy for the DECWRL gateway, in evolving a *screend* configuration to implement that policy, and in devising ways of dealing with *screend,* forms the basis for this document.

I thank Brad Chen and Win Treese for their help in proofreading the final draft.

8. References

Note: Many of the protocols described in ‘‘RFC’’ documents are evolving. The RFCs cited are the most up-to-date versions available at the time of writing, but you should check to make sure that they have not been updated.

[1] Douglas Comer. *Internetworking with TCP/IP: Volume I, Principles, Protocols and Architectures.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1990.

[2] Douglas Comer and David L. Stevens. *Internetworking with TCP/IP: Volume II, Design, Implementation, and Internals.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1990.

[3] David A. Curry. *Improving The Security Of Your Unix System*. Technical Report ITSTD-721-FR-90-21, SRI International, April, 1990.

[4] Steve Deering. *Host Extensions for IP Multicasting*. RFC 1112, Network Information Center, SRI International, August, 1989.

[5] Simson Garfinkel and Gene Spafford. *Practical UNIX Security*. O'Reilly & Associates, Inc., Newton, MA, 1991.

[6] F. T. Grammp and R. H. Morris. UNIX Operating System Security. *AT&T Bell Laboratories Technical Journal* 63(8):1649-1672, October, 1984.

[7] B. Kantor and P. Lapsley. *Network News Transfer Protocol*. RFC 977, Network Information Center, SRI International, February, 1986.

[8] David L. Mills. *Internet time synchronization: The Network Time Protocol*. RFC 1129, Network Information Center, SRI International, October, 1989.

[9] Paul V. Mockapetris. *Domain names - concepts and facilities*. RFC 1034, Network Information Center, SRI International, November, 1987.

[10] Paul V. Mockapetris. *Domain names - implementation and specification*. RFC 1035, Network Information Center, SRI International, November, 1987.

[11] Jeffrey C. Mogul. Simple and Flexible Datagram Access Controls for Unix-based Gateways. In *Proc. Summer 1989 USENIX Conference*, pages 203-221. Baltimore, MD, June, 1989.

[12] Jeffrey C. Mogul. *Simple and Flexible Datagram Access Controls for Unix-based Gateways*. Research Report 89/4, Digital Equipment Corporation Western Research Laboratory, March, 1989.

[13] Jeffrey Mogul and Jon Postel. *Internet Standard Subnetting Procedure*. RFC 950, Network Information Center, SRI International, August, 1985.

[14] Jeffrey C. Mogul, Richard F. Rashid, Michael J. Accetta. The Packet Filter: An Efficient Mechanism for User-Level Network Code. In *Proc. 11th Symposium on Operating Systems Principles*, pages 39-51. Austin, Texas, November, 1987.

[15] Robert Morris and Ken Thompson. Password Security: A Case History. *Communications of the ACM* 22(11):594-597, November, 1979.

[16] Jonathan B. Postel. *Simple Mail Transfer Protocol*. RFC 821, Network Information Center, SRI International, August, 1982.

[17] Jon Postel. *File Transfer Protocol*. RFC 959, Network Information Center, SRI International, October, 1985.

[18] Jonathan B. Postel and Joyce K. Reynolds. *Telnet Protocol specification*. RFC 854, Network Information Center, SRI International, May, 1983.

[19] Jon A. Rochlis and Mark W. Eichin. With Microscope and Tweezers: The Internet Worm from MIT's Perspective. *Communications of the ACM* 32(6):689-698, June, 1989.

[20] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and Implementation of the Sun Network Filesystem. In *Proc. Summer 1985 USENIX Conference*, pages 119-130. Portland, OR, June, 1985.

[21] Robert W. Scheifler, James Gettys, and Ron Newman. *X Window System: C Library and Protocol Reference*. Digital Press, Bedford, MA, 1988.

[22] Eugene H. Spafford. The Internet Worm: Crisis and Aftermath. *Communications of the ACM* 32(6):678-687, June, 1989.

[23] J. Steiner, C. Neuman, and J. I. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *Proc. Winter 1988 USENIX Conference*, pages ?-?. Dallas, ?, 1988.

[24] Clifford Stoll. Stalking The Wily Hacker. *Communications of the ACM* 31(5):484-497, May, 1988.

[25] Clifford Stoll. *The Cuckoo's Egg*. Doubleday, New York, 1989.

[26] Sun Microsystems, Inc. *NFS: Network File System Protocol specification*. RFC 1094, Network Information Center, SRI International, March, 1989.

[27] D. Zimmerman. *The Finger User Information Protocol*. RFC 1196, Network Information Center, SRI International, December, 1990.