Designing an Academic Firewall: Policy, Practice, and Experience With SURF

Michael B. Greenwald Sandeep K. Singhal Jonathan R. Stone David R. Cheriton

{*michaelg,singhal,jonathan,cheriton*}*@CS.Stanford.EDU*
Department of Computer Science
Stanford University
Stanford, CA 94305-9040

Abstract

Corporate network firewalls are well-understood and are becoming commonplace. These firewalls establish a security perimeter that aims to block (or heavily restrict) both incoming and outgoing network communication. We argue that these firewalls are neither effective nor appropriate for academic or corporate research environments needing to maintain information security while still supporting the free exchange of ideas.

In this paper, we present the Stanford University Research Firewall (SURF), a network firewall design that is suitable for a research environment. While still protecting information and computing resources behind the firewall, this firewall is less restrictive of outward information flow than the traditional model; can be easily deployed; and can give internal users the illusion of unrestricted e-mail, anonymous FTP, and WWW connectivity to the greater Internet. Our experience demonstrates that an adequate firewall for a research environment can be constructed for minimal cost using off-the-shelf software and hardware components.

1 Introduction

Growth of the Internet has increased awareness of the need for security to prevent destruction of data by an intruder, maintain the privacy of local information, and prevent unauthorized use of computing resources [15]. Indeed, analysts estimate that 50% of large corporations have seen a computer break-in over the past year [27]. Corporations are concerned with preventing unauthorized leakage of corporate intellectual property.

To achieve these goals, most corporate environments have deployed *firewalls* to block (or heavily restrict) access to internal data and computing resources from *untrusted* hosts and limit access to untrusted hosts from inside. A typical corporate firewall is a strong security perimeter around the employees who collaborate within the corporation, as shown in Figure 1a. The network security perimeter surrounds the corporate network and occasionally includes machines at employee homes. Everything else is excluded. The security perimeter carefully controls the transfer of information and, in many instances, forbids all outward information flow. Although many corporations desire more open access to the Internet, they limit Internet access and the sacrifice such services as personal World-Wide-Web pages to achieve corporate security. Even recent firewall designs which attempt to relax some of these limitations [14] only support specific interactions between sites located within a single authentication domain or private network.

Academic institutions also face concerns about the security of computing resources and information. Academic research groups often need to maintain the privacy of research grant proposals, patent applications, ideas for future research, or results of research in progress. Administrative organizations need to prevent leakage of student grades, personal contact information, and faculty and staff personnel records. Moreover, the cost of security compromises is high. A research group could lose its competitive edge, and administrative organizations could face legal proceedings for unauthorized information release. Furthermore, academic institutions are visible targets for hackers and intruders. Indeed, a large percentage of “crackers” are physically located within academic environments, and they are highly motivated to access and modify grades and other information. Network break-ins—and subsequent time lost recovering from break-ins and deletion of data—have become a fact of day-to-day life at educational institutions [5, 7]

In a corporate environment, the natural place to draw a security perimeter is around the corporation itself. However, in an academic environment, as depicted in Figure 1b, it is nearly impossible to draw a perimeter surrounding all of the people with whom we need to interact closely—and only those people. If the firewall is too big, it includes untrusted people, as shown by the dashed box. For instance, a corporate firewall erected around the entire University would contain many of the untrusted students and malicious hackers that the firewall should keep outside the perimeter. Moreover, universities offer almost no physical security. On the other hand, if the firewall is too small, then it will exclude some of the people with whom we must share data, as shown by the dotted box. A corporate firewall erected around a research group would exclude collaborators located in other departments or even at other universities. This tradeoff between safety and collaboration is unacceptable. Consequently, the traditional corporate firewall is ill-suited for academic environments.

While corporations tolerate limited Internet connectivity in the name of security, research organizations simply cannot function under such limitations. First, trusted users need unrestricted and transparent access to Internet resources (including World-Wide-Web, FTP, Gopher, electronic mail, etc.) located outside the firewall. Researchers

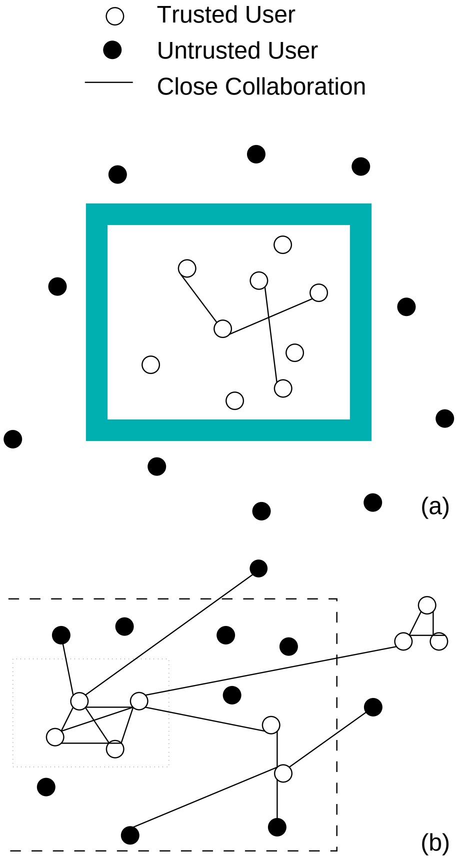


Figure 1: Nature of Trusted Users, Untrusted Users, and Collaboration in (a) Corporate Environments and (b) Academic Environments

rely on fingertip access to on-line library catalogs and bibliographies, preprints of papers, and other network resources supporting collaborative work. Second, trusted users need the unrestricted ability to publish and disseminate information to people outside the firewall via anonymous FTP, World-Wide-Web, etc. This dissemination of research results, papers, etc. is critical to the research community. Third, the firewall must allow access to protected resources from trusted users located outside the firewall. An increasing number of users work at home or while traveling. Research collaborators may also need to enter the firewall from remote hosts. Besides these factors, the usual considerations of cost, ease-of-management in a heterogeneous computing environment, performance, and reliability apply. Budget cuts are precluding academic institutions from allocating money for computer security hardware, software, or personnel; we expect this to be a long-term constraint. The lack of dedicated security staff means, for example, that a firewall might be managed by inexperienced people (i.e. incoming graduate students) with no expertise in custom components.

In this paper, we describe the Stanford University Research Firewall (SURF), a firewall implementation developed by the Distributed Systems Group at Stanford University to achieve the above goals. The next section describes our security policy and its motivation. We then discuss how we implemented the three components of this policy: "request-response" packet filtering for information security, "expendable hosts" for information dissemination, and a "virtual enclave" to support trusted users outside the firewall. We highlight the lessons learned during the firewall development process and then discuss how application protocols can be modified to better support implementation of our security policy. We conclude with an indication of future work.

2 The SURF Security Policy

A firewall implements a particular security policy that trades off the need to support collaborative work by maintaining open Internet connectivity against the need to provide security by restricting this connectivity. The security policy is affected by how much trust is given to internal users and is limited by implementation considerations. After all, there is little point in adopting a security policy that is impractical to implement.

Our firewall policy attempts to balance the collaboration and security concerns better than in traditional corporate firewalls. To achieve this goal, we trust our users to understand the importance of security and not intentionally attempt to bypass the mechanisms in place. This trust is comparable to allowing an individual to participate in a research group, attend meetings, and have a computer account. By making the firewall policy as unobtrusive to users as possible, we also increase this trust by eliminating the temptation to bypass the security mechanisms.

Our security policy can be stated in three simple rules, summarized in Figure 2:

1. All outbound packets are allowed to travel outside, and inbound packets are allowed inside the firewall *only if* they can be determined to be responses to outbound requests.

2. Packets to or from outside-the-firewall "expendable hosts" are unrestricted (aside from normal operating system and application-level access controls) because they are outside the security perimeter.

3. Packets known to be from authenticated hosts or users outside the firewall are allowed inside the firewall.

The rationale for this policy is straightforward. Rule 1 follows from our recognition that open network access is a necessary component of a research environment. The rule relies on the assumption that we trust our users to understand and adhere to the research group's security goals. The *Request-Response* security policy states that an outgoing request implicitly grants permission to admit its response into our secure network. Rule 2 addresses our need to support information dissemination (FTP, WWW, etc.) in a research environment. We simply accept that these expendable hosts may be compromised and choose to automatically recover their state on a regular basis from information kept securely behind the firewall. Compromises to expendable hosts therefore do not affect the security of the private network. Rule 3 grants access to protected resources to users as they work from home or while travelling, as well as to collaborators located outside the research group. We rely

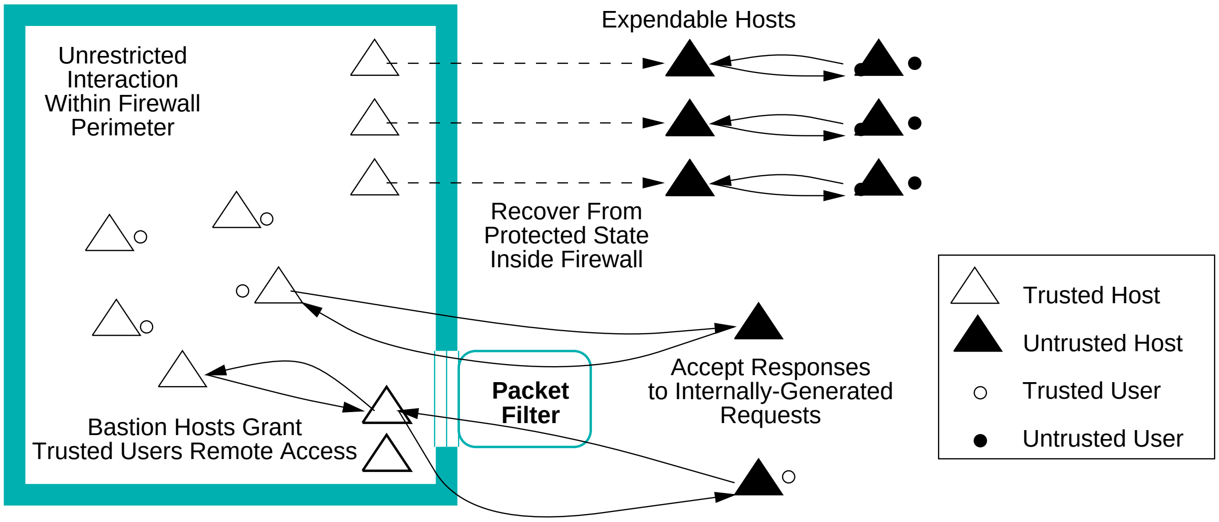


Figure 2: SURF Design with a Request-Response Security Policy, Expendable Hosts, and Bastion Hosts Supporting Remote Access for Trusted Users

on secure IP tunnels and carefully selected authentication mechanisms to implement this *virtual enclave* environment.

This security policy addresses the needs of academic environments—and indeed the needs of many corporate environments. The next three sections describe how we implemented these security rules within our research environment.

3 Request-Response Policy

Our implementation of the request-response policy relies primarily on a packet filter located at the firewall perimeter. Some of the responsibilities for recognizing responses to outstanding requests must be delegated to operating system mechanisms provided by protected hosts themselves. However, when application protocols are ill-suited for our request-response paradigm, we must rely on application-level proxies running on *bastion hosts*.

3.1 Packet Filtering and Host Filtering

The packet filter is responsible for detecting whether an incoming packet is actually a response to a prior outbound request. This task is difficult, however, because packet-filtering hardware is usually stateless; it must decide whether to drop or pass a packet based only on the physical port on which the packet arrives, the port on which it will be forwarded, and data in the packet itself. Packet filters do not, in general, have access to either previous packets or any internal host state. Furthermore, caching information about previous packets in a packet filter is equivalent to writing an application-level proxy and is impractical because it would increase the latency of all incoming packets.

TCP Request-Response: Filtering at Internal Hosts: Packet filtering is best suited to handle TCP-based services that establish explicit connections. In terms of our request-response policy, we treat connection establishment as a "request" and data transmission on an established TCP connection as a "response." Intuitively, the fact that an internal host decides to establish a TCP connection to an external host is, in itself, implicit permission for that external host to send packets back through our firewall on that TCP connection.

In other words, the request-response policy implies that our firewall should allow external hosts to participate in TCP connections established by any host inside our firewall, but external hosts may not initiate such connections themselves. We achieve this by filtering incoming connection requests and relying on our internal hosts to drop extraneous TCP segments.

We filter incoming connection requests using standard SYN filtering. The TCP connection-establishment phase is the three-way handshake: SYN, SYN+ACK, ACK. Our filtering policy for TCP admits all inbound TCP packets, *except* packets attempting to initiate the three-way handshake—that is have SYN set but not ACK.

External hosts can therefore inject arbitrary TCP packets into our secure network except connection-request packets. We rely on the TCP module on all of our hosts to function correctly and discard TCP segments that do not match an established connection. Because an external host cannot initiate a TCP connection, TCP packets constructed by a would-be intruder should always be dropped by the destination host.

Thus, the combination of filtering external SYN packets, admitting all other TCP segments, and a working TCP implementation achieves precisely the desired request-response policy. This implementation does leave us vulnerable to denial-of-service and hijacking attacks, which are discussed in Section 6.2.

Connection-Based UDP Services: A few UDP-based application protocols, such as AFS, use well-known kernel ports at both the client and server hosts. We treat such protocols much like TCP connections by deploying a heuristic filter (based on source and destination addresses and port numbers and packet contents) and trusting internal hosts to further filter inappropriate responses.

NFS-client filtering at hosts: NFS [26] is a request-response protocol: the NFS client sends RPC [25] requests to the NFS server, and the server returns the results. The request-response rule of our security policy implies that NFS clients protected by our firewall should be permitted to NFS-mount filesystems from NFS-servers outside our firewall if group members behind the firewall choose to do so. We have found this to be a useful way to access resources such as campus software archives and Linux slackware distributions. This policy decision is markedly different from traditional network firewall policies.

As with TCP, we support a request-response policy for NFS by using a coarse-grain filter and trusting internal hosts to safely ignore and drop unexpected response packets. Our packet filter allows any packets out, including NFS requests. It also allows *in* any packets with a source port of 2049, with an SunRPC type of RPC_REPLY, and with the SunRPC program number set to RPCPROG_NFS.

The process of performing an NFS mount actually requires an exchange with a "mount daemon" on the remote NFS server to obtain an initial NFS "file handle." The mount daemon is, in turn, located via an RPC request to the remote server's portmapper. To avoid problems with UDP port mapping, we use a modified NFS-mount utility, that performs these RPC transactions over TCP instead of UDP.

Our heuristic test for NFS response packets is currently the biggest hole in our firewall: A would-be intruder could generate UDP packets with all of the attributes of a valid response and send them to an arbitrary machine behind our firewall. We trust that each host's NFS client software correctly drops packets that are not valid responses to an outstanding NFS request and that our heuristic is strong enough to prevent forged responses from affecting other UDP clients. The risks that our policy entails in our environment are further discussed in Section 6.2. Modern NFS clients and servers are configurable to run entirely over TCP, which fits entirely within our TCP request-response implementation. Unfortunately, because our internal workstations run Unix with NFS implemented inside the kernel, using these newer implementations would require kernel modifications, compromising our goal of easy firewall installation.

3.2 Application-Level Proxies for Ill-Behaved Application Protocols

We have found that many application protocols are not immediately amenable to our request-response paradigm. These applications typically fall into two categories: connectionless UDP-based applications and "reverse-channel" TCP-based applications. For a few critical applications, we introduce application-level proxies to create a request-response illusion over an otherwise ill-behaved protocol. The proxies run on a *bastion host*, an internal host configured specifically for the purpose. Unlike many traditional designs, our firewall permits multiple bastion hosts, thereby both eliminating the performance bottlenecks often seen in other firewall designs and allowing each bastion host to be stripped-down to support a limited set of applications. The packet filter accepts incoming packets for these restricted protocols if they are destined for an appropriate bastion host.

Connectionless UDP-Based Applications: Most applications do not use UDP sockets in connected mode, so an application receives all packets addressed to its UDP socket, regardless of the foreign address or port. Because there is no connection setup, a packet filter and endpoint host cannot distinguish between a UDP "response" from an external host and a subsequent "request" from that external host that coincidentally or maliciously uses the same remote and local port numbers as an earlier response. To evaluate whether an inbound packet is indeed a valid UDP "response," the packet filter or host operating system would need to retain some application-specific state. Clearly, our TCP approach of a packet filter and host filtering is impractical for connectionless UDP protocols.

Two connectionless UDP services that we regard as essential are NTP [20] for time synchronization, and DNS [21] for mapping between hostnames and addresses.

NTP servers are configured in a tree; higher-level servers periodically broadcast their current time statistics to lower-level servers, so the lower-level servers can measure and adjust for clock drift. This server-initiated traffic does not match our request-response paradigm.

We allow NTP through our perimeter by designating a set of internal hosts as NTP "bastions." Our packet filters allow traffic from the NTP port of any external host to the NTP port of any NTP bastion. All other internal machines synchronize to the bastion machines. This approach fits naturally within the architecture of NTP, and has not caused any problems to date.

Although DNS fits a request-response paradigm and can be configured to use either TCP or UDP, DNS clients typically use UDP from an arbitrary UDP port. We manage DNS like we manage NTP. We have configured a set of internal "slave" nameservers that forward all requests they receive to a set of external nameservers. Our packet filters admit UDP traffic with source and destination ports set to the DNS reserved port from external hosts to our internal nameservers. This design has worked satisfactorily, and has actually produced an unexpected positive side effect: because the local nameservers maintain a cache, we have seen improved performance during intermittent failures of the campus nameservers.

The approach we have taken for other UDP services is unconventional: we simply require users to invoke these services from our expendable machine outside the firewall. Services like `talk`, `archie`, and `rusers` use UDP and are currently blocked by our packet filters. While far from ideal, maintaining a login session to an expendable host for these services has proved to be a workable interim solution.

Proxies for Reverse-Channel Services: Our request-response implementation also cannot currently handle TCP-based services that require a "back-channel" connection. For example, FTP clients implement requests like `dir` and `get` by creating a local "throwaway" TCP socket, sending a message to the remote server asking that the resulting output be sent to the throwaway port, and then waiting for the remote server to connect to the throwaway port and send the data. Our request-response implementation blocks the server's connection to the throwaway port. The FTP protocol is almost rich enough so that minor changes to the client can eliminate most of FTP's uses of back-channels.[1] Unfortunately, few protocols are as flexible as FTP. For example, the back-channel problem also arises

---

[1] By turning off the PORT command, we can eliminate the back-

with the Berkeley `rsh` command, which establishes back-channels for the remote `stderr`.

The back-channel problem arises in a different form when a user inside the firewall tries to start an X windows [24] client on an external host to appear on the user's local display. To connect to the display, the external client must establish a TCP connection to the X server inside the firewall. Our access policy does not allow this connection, however. We could choose to let the connection through the firewall and rely on the X server to reject unauthorized connections. Unfortunately, authentication systems such as Kerberos [23, 19] are not deployed on all potential external client machines. Without a convenient, unobtrusive, and ubiquitous authentication mechanism, we believe the temptation to disable server authentication is too great. At this point, we have not fully assessed the implications of allowing arbitrary client connections to reach internal X servers.

We currently use off-the-shelf application-level proxies for remote X clients, FTP, and login. This solution is similar to that taken by existing corporate firewalls. For example, the X proxy simply allocates a new logical display on the bastion host corresponding to the display on a trusted, internal host H. If a client wants to open a window on host H, it instead opens the corresponding logical display on the bastion host. The proxy queries the user on host H for permission to open the window. If the user on host H grants permission, then the bastion translates all operations on the logical display to the corresponding window on host H. At one level, the bastion is actually seeking explicit human-level acknowledgement that the incoming client connection was actually requested. This is, therefore, consistent with the request-response policy, though the filtering is not automatic.

Ultimately, we regard the use of application-level proxies as a temporary workaround for ill-designed application protocols. We would prefer that our packet filter could fully implement the request-response policy. As the need for security becomes more prevalent, we hope and expect that many of the connectionless UDP and back-channel TCP protocols will be redesigned to support the automated request-response paradigm.

4 Exposing a Secure Public Image

A carefully controlled "network identity" is a second element of our academic firewall. The public image—the host names, addresses, and services—exposed to the Internet become the first target for intruders. By limiting the number of exposed hosts and strategically setting up those resources, we reduce our exposure to the most common attacks.

Our public image consists only of "expendable" hosts and decoy host names, as shown in Figure 3. An *expendable host* is an outside-the-firewall machine whose data can be easily re-created from information kept securely behind the firewall. A *decoy host name* points either to a non-existent machine or to a machine instrumented to simply log all accesses. We do not publish DNS entries for any of the machines located inside the firewall, except for bastion hosts discussed in Section 3.

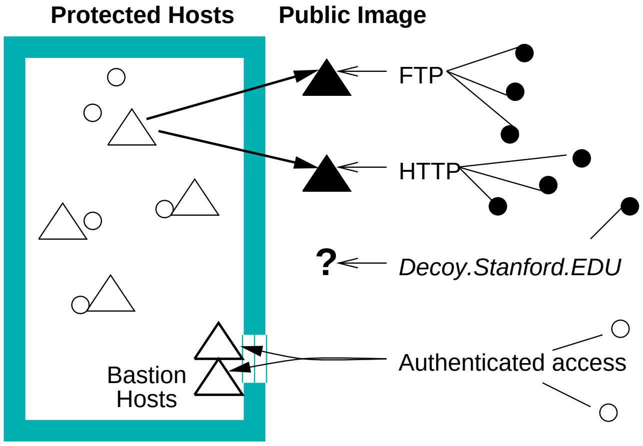


Figure 3: Only Expendable Hosts and Decoys are Exposed to the Internet

Our public image is designed to support untrusted interactions with the wider Internet community. These interactions fall into three categories: information dissemination, insecure software and experimental protocols, and guest users.

4.1 Information Dissemination

All information dissemination services—such as anonymous FTP, HTTP (World-Wide-Web), Gopher, WAIS, and finger—are provided by one or more expendable servers outside the firewall. Providing these services is critical in a research environment, but because they involve interactions with unauthenticated users and hosts, they pose considerable security risks. In contrast to traditional firewall designs that try to use "secure" or chroot'ed proxies to monitor the behavior of these interactions, we simply assume that these protocols are inherently insecure. Consequently, we run standard server implementations on expendable hosts and just assume that those server hosts are regularly compromised by intruders.

This assumption of insecurity in the server hosts has three major implications. First, private data should never be placed on the server machines. By creating a physical separation between publicly accessed data and non-public data, we aim to prevent accidental release and/or corruption of non-public data *when* break-ins occur on the server machines. Second, users should never need to log onto the server machines. By keeping users off the insecure servers, we reduce the chance of accidental information release or corruption. We can provide a separate expendable host for users to run untrusted software or protocols. Third, all of the state on the server hosts must be easily recoverable because we assume it gets corrupted. In fact, each of the expendable hosts should be automatically restored regularly from uncompromised sources.

To implement these policies, all data stored on server machines is simply a shadow of directory hierarchies stored on protected machines inside the firewall. For example, files made available for anonymous FTP are stored in `.../pub/ftp` and files made available for HTTP are stored in `.../pub/www` inside the firewall. These directories are rdist'ed hourly to the server machines. Recovery

---

channel problem for `get` requests, but it remains for `dir`.

from data corruption on a server machine is, therefore, automatic. As a natural extension to this approach, the entire operating system on the expendable hosts could be re-installed nightly from a read-only hard drive.

By making the shadowed directories publicly writable on the protected machines, we allow any trusted user to add, remove, or modify his own public data. We find this capability to be a powerful departure from corporate environments where individual users are not permitted to make information public.[2] The capability means that users do not need to access the expendable host, thereby enhancing their own security. Moreover, the automated shadowing process ensures that public data on the expendable machine is configured correctly (i.e. owned by the appropriate user (ftp, http, etc.), placed in the appropriate directories, and given the appropriate access permissions). In effect, individual users explicitly decide which data is public and which data is private, and the system then assures that public data is disseminated, that private data is kept secure, and that the two types of data are kept separated.

Our approach to information dissemination has three significant advantages for an academic environment. First, it eliminates almost all management overhead. System maintenance is largely automatic. Other than occasionally scanning log files to note any anomalous access patterns, no human intervention is needed. Second, in many ways, our approach is more secure than those used by traditional corporate firewalls. We do not need to rely on a proxy to catch security threats, and we do not make assumptions about the security of the underlying operating system. Third, our use of expendable hosts offers a degree of extensibility that is not provided by traditional firewall implementations. As new Internet services and protocols are defined, we can immediately deploy standard, possibly insecure, implementations on an expendable server. This option is very attractive, particularly when compared to the alternative of obtaining or creating a secure implementation of each new service.

4.2 Insecure Software

We have also explicitly decided that certain types of software are intrinsically insecure, especially when inputs arrive from outside the firewall. For example, in our research environment, users often develop experimental network protocols that they wish to test over the Internet. Expendable machines provide an environment in which we can deploy such experimental protocols. Testing behind the firewall is permitted only if the tester can control all inputs and no packets cross the firewall.

Our treatment of electronic mail delivery represents a broader example of this approach to insecure software. Based on `sendmail`'s long history of security problems [3, 4, 6], we have decided not to trust `sendmail` to deliver electronic mail originating from outside the firewall. The fact that four more security holes [9, 8, 10, 11] were found in `sendmail` after we deployed our firewall reinforces the wisdom of this decision. Consequently, `sendmail` processing and delivery of all externally-originating mail must occur on an expendable mail server outside the firewall. In this way, all external mail is completely processed by the insecure sendmail before ever touching a protected machine.[3] Internally-originating mail, on the other hand, is processed by `sendmail` on a host inside the firewall. This separation of internal and external mail is entirely consistent with our policy of physically separating public and private data.

We accomplish the mail separation by publishing different MX records to protected and unprotected hosts. Protected hosts are only aware of internal mail servers, while external hosts only see the external mail servers. If a research group does not have its own DNS domain, it can implement the mail separation by associating each host with multiple mail servers: the internal mail server is the primary and the expendable mail server is the secondary. The packet filter blocks incoming TCP connections, so all outside mailers fall back to the secondary (external) mail server. Internal mailers, on the other hand, can contact the primary (internal) mail server directly. This alternative design has the drawback that if the primary mail server fails, all internal electronic mail flows to the untrusted external host, where it might be snooped or corrupted. Providing multiple primary mail servers behind the firewall reduces the chance of internal e-mail leakage, but ultimately, preventing electronic mail leakage requires end-to-end encryption.

Normally, this separation of internal and external mail would force users to access two separate spool files, thereby hurting usability and transparency of the firewall. To address this problem, the internal mail server uses NFS to mount the spool directory of the external mail host. A "Magic Carpet" process runs as a `cron` job on the internal firewall at one minute intervals. For each authorized user on the internal system, it checks for mail in the external server's spool directory. E-mail for those users is appended to their internal spool files, and an appropriately formated "Received" line [12] is added to the mail header to reflect the transfer. Biff is notified of each delivered message. To the user, therefore, internal and external mail are indistinguishable, even though they are initially processed by sendmail processes running on different machines. By only transferring mail for internal users, we can support "guest" accounts on the external mail server.

The Magic Carpet program does not provide a general-purpose `rdist` capability. Instead, it is written to operate only with inert data files. For example, we do not preclude the possibility of electronic mail "bombs" that exploit bugs in mail reading software. Such errors manifest themselves in user space, however, so the potential damage is less severe. Moreover, the mail bomb problem is a larger issue that the research community must address, particularly as electronic mail is embedding a growing variety of data types.

The expendable mail server is reconstructed nightly from information stored inside the firewall. Its aliases file is reconstructed behind the firewall by taking the internal

---

[2] Indeed, our expendable firewall machine provides the homepage for the footbag competition. A group alumnus is a footbag competition organizer, but his employer's firewall implementation does not enable him to make documents available to the public. He has a guest account on our expendable host through which he periodically refreshes the footbag web pages.

[3] Currently, all users in the internal mail server's /etc/passwd file must have entries in the expendable server's /etc/passwd file, though the external account shell is typically set to /bin/false. We are currently exploring ways to simplify this aspect of the setup.

server's aliases file and appending information collected from user `.forward` files. This technique provides additional security by ensuring that mail enters the firewall only if it is locally deliverable. It also makes the mail forwarding process transparent to users. We support logging of mail to mailing lists by introducing specially tagged "dummy" user names into the expendable server's aliases file. For example, the Magic Carpet delivers mail for users of the form `LOG-foo` on the expendable server to a logfile `foo` inside the firewall.

Our design for electronic mail processing offers two significant advantages over alternatives such as Post Office Protocol (POP) [22]. First, except for a slight delay in arrival of mail from outside the firewall, local and remote delivery is transparent to users. Moreover, standard mechanisms for mail forwarding, aliases, and logging are naturally extended to the firewall environment. Our polling model also supports standard mail notification mechanisms such as biff. Second, because internal mail never leaves the firewall, our system allows internal mail delivery to continue and remain secure even when outside network connectivity is lost. As soon as the network is restored, delivery of external mail resumes as usual.

4.3 Supporting Guest Users

Many academic environments offer guest accounts to short-term visitors, collaborators, and alumni. Because these users are potentially numerous, are not always accessible, and have no direct interest in maintaining the security of our information, they are typically not trusted. We place their accounts on an expendable machine outside the firewall. These users are warned not to store critical information in these courtesy accounts. This policy does not pose any significant problems for guest users, however, because they typically only require the guest account for accessing their normal machine remotely or for receiving electronic mail.

Expendable machines can also benefit trusted group members in a similar way. An expendable machine with a shadow copy of the internal `/etc/passwd` file can serve as an emergency access point for users who are otherwise unable to access protected resources from outside the firewall (see Section 5). While the expendable machine does not provide users with access to protected resources, it can still provide basic electronic mail, news, FTP, and web access to our users. Furthermore, exporting user passwords onto an expendable machine does not introduce any additional security risks because user passwords are insufficient for gaining access to protected hosts from outside the firewall.

5 Secure Non-Local Access

Traditional corporate firewalls assume that most users are physically co-located inside the corporate firewall. The few users who are travelling on business or working from home may be authorized to gain access through the corporate firewall. Such access typically involves a modem call to a private, corporate point-of-presence, or "smart card" shared-secret authentication, or (in some cases) both. Corporate travellers do desire to access their secure machine from an insecure terminal and network connection, such as from facilities provided at a conference, but such access is typically not permitted.

This assumption of physical co-location or secure network access is not viable in research environments. Most users do considerable work at home or while traveling. Furthermore, mobile hosts, particularly laptops running Linux, are becoming commonplace among researchers. In addition, collaborative research often requires that certain individuals from other organizations be granted access to resources that we do not wish to make available to the general public. In each case, the remote user is potentially approaching the firewall through an insecure network. Within an academic research group, "smart cards" or dedicated dial-in points-of-presence are too costly to acquire and administer, and they do not fully address our need to support access from insecure hosts connected to the Internet.

Moreover, the SURF architecture defines security perimeters at a much finer granularity than is usual: at the level of an academic research group or department rather than a single corporate entity. This finer granularity means that extending a firewall to *permanently* encompass sites outside a single building or a single LAN segment is more important. Together, these multiple sites constitute a *virtual security enclave*, and the enclave must have a firewall at every point of connection to an insecure network.

The remote user, once authenticated to a trusted host, should have access to protected resources as if she were physically located behind the firewall—or as much as practicable. However, the security mechanisms must ensure that granting this remote access does not compromise either the user's own, or anyone else's, data security.

Our firewall provides two methods of non-local access with different levels of functionality. The first access method provides authenticated, secure IP access by means of an IP tunnel. This tunnel allows the remote user to freely send IP packets without restriction through the firewall. The second access method provides authenticated remote login. It is intended for use by users who are travelling or who work off-site but cannot set up a secure IP channel.

5.1 Secure IP tunnelling

To support users who regularly work at home or trusted collaborators at other schools, we need to connect isolated sites and give them unrestricted access to all internal machines. We assume that the campus internetwork, which is presumed to be insecure, will be used as the primary conduit to provide users with access from home machines.

To connect two isolated sites that are protected and mutually trusting, we establish a secure IP tunnel between them. Arbitrary IP packets are then encapsulated and tunnelled between the two sites. We chose to use Ioannidis' swIPe protocol [18] to implement an authenticated, encrypted IP tunnel. The endpoints of the tunnel are known, and the filter ensures that incoming swIPe packets are actually part of an already registered tunnel.

Alternatively, we can support a dedicated connection (either leased line or modem) between two sites protected by their own firewalls. For example, our security enclave includes a subnet, directly connected via leased line, in the home of a faculty member. However, due to cost considerations, secure network connections are not generally feasible.

5.2 Remote Login and Remote Execution

Permanent secure IP tunnels do not meet all of our needs, however. While travelling, users cannot always gain access to a trusted host, and those connections have a short duration. To support incoming access in these cases, we allow remote login channels providing the illusion of a terminal connection to a machine inside the firewall. Support for this service is straightforward: the packet filter allows access to a single bastion machine acting as a gatekeeper. After authenticating themselves to the bastion machine, users are given a proxy that permits a single telnet session to another machine inside the firewall. We do not want shells running on the bastion machine, because we believe that a machine upon which one can execute a shell is more easily compromised.

The need for remote login access is not unique to our firewall. Many mechanisms are in widespread use. We provide two remote login services: S/Key and Kerberos.

S/Key [16] generates a set of one-time passwords enabling the remote user to log onto a bastion from anywhere. The keys are provided by interacting with the S/Key server over a secure channel (local or Kerberos-encrypted telnet). Users must be careful to acquire enough keys before going off-site, because they have no safe way to acquire more keys using *only* S/key access through the firewall.

S/Key provides only limited functionality. It is intended for users who only have access to an off-the-shelf telnet client. Because the conversation is not likely to be encrypted, we do not consider it completely trustworthy. In particular, users of unencrypted S/Key must assume that their passwords are compromised when typed over an insecure network.

Encrypted Kerberos connections [23, 19] are more secure and, therefore, can permit greater functionality. Kerberos is used to authenticate the remote user, and encryption is used to keep the connection secure. The Kerberos password is never typed over the network in the clear; so it is not compromised. Because the connection is encrypted, a Kerberized login may, for example, be used to acquire more S/Key passwords.

We provide a Kerberos remote login server on a bastion machine that only allows users to execute an rlogin command. No shells are permitted on this machine. Kerberos does not eliminate the need for S/Key: Kerberized telnet clients are not available for all of our portable platforms, and even when a Kerberized telnet client is available on a visited machine, we should be wary of trusting it with a secret key.

6 Experiences in Designing and Deploying a Firewall

Our most positive experience has been other people's break-ins. We saw three network break-ins in our building over a three month period. After each break-in, the affected groups have had several days down-time, while system administrators re-install software and data from distribution media and backups. We have not observed any break-ins through our firewall in that period, but the firewall also has a downside. To date, we have spent more time developing our firewall than we would have spent recovering from those break-ins. However, we view firewall development as a time investment that will prove its worth as more break-ins occur around us; moreover, other research groups that choose to follow our firewall architecture need not invest time reinventing this particular wheel.

Our firewall policy has evolved over nearly two years, and we expect this evolution to continue. This evolution confirms an important tenet of firewall design: having established a firewall, one cannot become complacent. Evolution occurs because of three forces. First, new security concerns arise, whether as a result of CERT advisories, recent breakins at other sites, or even penetration of one's own site. Second, application requirements change over time. When a new networked application becomes commonplace, the firewall's packet filters must eventually be extended to support the new protocol. One might also eliminate support for applications that are no longer widely used. Third, user access requirements change. In particular, the need for flexible authentication for incoming access seems to increase over time. The most difficult element of this evolution has been trying to maintain the firewall's simplicity (and hence its maintainability and security) without compromising flexibility. This evolution has led us to the request-response policy that we aim to fully support.

6.1 Off-The-Shelf Software and Hardware

For simplicity and maintainability, we only used off-the-shelf components, even though available components did not always behave optimally. For example, we would prefer that our telnet server and clients encrypt an S/Key session with the next key in sequence, or that our X server use a well-defined static mapping function so that a given client could be told, under program control, which logical display on the bastion host to use. Hopefully, to the extent that such features are generally useful for firewalls, they will be incorporated into these components in the future.

We are using an off-the-shelf switching Ethernet bridge for our packet filtering. As with most Ethernet bridges, it was not designed with firewalls specifically in mind, and we consequently faced several problems. The switch's biggest shortfall occurs during reboots. It begins forwarding packets *before* its filters are enabled. For approximately 30 seconds, the bridge forwards packets with no filtering. Furthermore, carefully chosen packet sequences can crash the bridge, so an intruder could intentionally crash the bridge and wreak havoc during the window of opportunity.

To reduce the likelihood of such a break-in, we have installed a simplified version of our filter in the router connecting our network to the rest of the campus network. Because this machine is not under our administrative control, we choose not to *depend* on the router's filters during the normal course of events. However, it does provide an added level of protection, and in particular, protects against compromising our network while the bridge is rebooting.

Currently, even when the network is idle (fewer than 15 incoming packets per second), our filters block one or two packets per second. These packets are usually *not* breakin attempts but instead are benign packets such as `rwho` packets, unauthorized multicast packets, SNMP, etc.

Our experience with the switch's packet filtering performance reveals that anyone deploying a firewall must measure the actual packet latencies, which may not match the manufacturer's specifications. Our switch re-parses its filtering rules for every incoming packet, and we conjecture that our packet filtering rules are much larger than the manufacturer expected. Without packet filters, the switch takes about 6.4 microseconds to process each packet. According to the manufacturer's specification, our filters should add about 11.8 and 14.1 microseconds, tripling the total processing time. This overhead is still less than the time it takes to transmit a minimum size Ethernet packet, so if dispatching were done in parallel with transmission, the firewall filters should introduce no additional latency. However, actual measurements show that packet filtering increases the round-trip delay per packet by approximately 180 microseconds. In addition, this increased processing time also increases the average packet queue length in the firewall during heavy load. Under load, the packet filter introduces an average additional latency (including additional waiting time on the queue) of 400 microseconds.

The expressive power of the filter rule language also leaves much to be desired—to the point that we developed a richer filter language of our own and an optimizing translator to the switch's native inputs. Although these observations are based upon experiences with a single switch, poor filtering performance is characteristic of most switches on the market. Switch and router vendors rate their products on packets-per-second throughput, ignoring the need for filtering in today's Internet. As switch manufacturers realize the increasing importance of firewalls, we hope and expect that such failings will be eliminated and that necessary features will be added to better support firewall development.

6.2 Vulnerabilities of the SURF Firewall

The SURF firewall design faces three potential sources of vulnerability: the open environment behind the firewall and absence of outgoing packet filtering, the coarseness of the incoming packet filter, and the potential for hijacking otherwise acceptable connections. We feel that each of these risks is either necessary to maintain an acceptable research environment or is already present with traditional firewalls.

Open Research Environment: The first risk arises because we do not restrict traffic between machines behind the firewall and because we do not filter any outbound packets. As a result, an intruder who penetrates the security perimeter has unrestricted access to all internal hosts. Moreover, once an intruder gains access to an internal host, SURF does not prevent any outbound operation, including data transfer. In effect, once it has been breached, the firewall no longer offers any protection. However, any attempt to ease these risks would significantly affect the openness of our research environment and thereby make the firewall unacceptable to our users.

At first, it might appear that having multiple physical enclaves connected by secure IP tunnels (swIPe) is an additional security risk. However, each enclave is surrounded by its own trusted firewall; these firewalls can be identical. Clearly, an attack that breaches one of these firewalls would also have breached a single physical enclave. Furthermore, the swIPe tunnels themselves are secure because we trust the DES encryption that they use. Therefore, our multiple physical enclaves do not introduce any additional risk.

Coarse-Grain Packet Filter: Our second risk comes from the design of our incoming packet filters. We employ a coarse-grain packet filter and rely on internal hosts to perform additional filtering. As a consequence, our packet filter admits packets that may not in fact be responses to outstanding requests. This policy potentially exposes us to a denial-of-service attack flooding our internal network with "nuisance" packets that internal hosts must subsequently process and drop. However, this risk is also not significant. An intruder can flood *any* protected network by discovering a packet which elicits a response from behind the firewall. Obviously, a packet filter cannot protect anything outside the LAN, such as the tail circuit or the filtering hardware itself.

In trusting internal hosts to discard inappropriate packets and unsolicited responses admitted by the filter, we must carefully choose the set of acceptable protocols that the packet filter accepts. We might otherwise accidentally admit Trojan Horse packets *masquerading* as responses to one application but in fact are requests to another; a poorly-written application may fail to implement end-to-end checks to validate that incoming packets are actually responses to outstanding requests. However, these risks are known in advance, and before allowing a particular application protocol through the firewall, the network managers must determine that the benefits of admitting apparent responses of the new protocol outweigh the additional masquerading risk. The vulnerability largely results from limitations in the application protocols themselves (and the semantics of UDP ports), and in the long-term, these protocols should be redesigned, as discussed in Section 7. In our current environment, the risks associated with over-permissive response filtering are worth the benefits of transparent access using off-the-shelf clients and servers.

Connection Attacks: Our final risk arises from potential attacks to authorized connections through the firewall. TCP sessions to external hosts are subject to hijacking, man-in-the-middle, and eavesdropping attacks. In addition, NFS requests to a compromised external fileserver may cause us to read or execute intruder-supplied data. To address these dangers, one must simply be careful in selecting which applications are available to users. For example, one would be foolhardy to permit an application through the firewall if its *response packets* could obtain control of a shell running inside the firewall.

SURF is also vulnerable to this class of attack because it supports authenticated access through potentially insecure hosts and networks outside the firewall. If a research group member is travelling and gains legitimate access to our internal machines from some machine, then that connection is vulnerable to attack at any point outside our network. By inserting himself between the user and the firewall, an intruder can gain full access to our network. These risks will only be eliminated in the future when portable "bastion hosts" (personal devices capable of encrypted telnet) are ubiquitous. However, the ability to log in while traveling is so important that, for the moment, we trust the discretion of our group members. When unable to plug their own personal computers into the network, users are expected to use one-time passwords, carefully choose the machines they use to log in, and be aware of the risks of connection eavesdropping and hijacking.

6.3 User Acceptance

Over the past eighteen months, user response to our firewall has been generally positive. All common outgoing services are implemented transparently, so most users are completely unaware of the firewall until they need to access internal resources from outside the firewall.

We have, unfortunately, discovered that many people in research environments are unaware of network security risks. Even those who are aware of the issues are reluctant to change their behavior, either because they expect to face considerable inconvenience or because they do not feel that the risks are significant ("breakins happen everywhere else but not to me"). As a result, we must educate new group members about the importance of security, remind them to explicitly separate public and private data, and train them on how to access data through the firewall.

Most user complaints have centered around the support for incoming access. Users complain about being required to carry around a list of one-time passwords for S/Key access. FTP access to secure machines from outside the firewall is also a problem. Our filters disallow FTP connections from outside the firewall, so users must log onto a secure machine and then initiate the FTP from inside the firewall. Finally, our filters block `finger` requests originating outside the firewall. We found that Internet users rely on `finger` to provide telephone and address information. To address these concerns, we plan to support `finger` on one of the expendable hosts.

Although we have amassed a considerable body of experience dealing with *one* firewall, we do not yet have experience with an environment where *all* groups are behind their own academic firewall. File transfer between two protected hosts would then presumably require staging on some public machine; for truly sensitive data it would also require encryption and, consequently, more user training.

Finally, the level of interest in firewalls has been noticeably lacking in academic circles, as evidenced by the dearth of forums for exchanging firewall design ideas, issues, and experiences. We see this as unfortunate because many academic institutions are completely unprotected from network attacks. As a result of the lack of interest and experience in this area, those who wish to design an academic firewall have been largely on their own. As the Internet grows and security becomes more of a concern, we expect interest in academic firewalls to grow. A greater body of experience in this area will considerably ease firewall deployment.

7 Protocol Design Issues

Given current application protocol design, we do not believe that it is possible to implement a pure request-response firewall policy. For example, we cannot deploy request-response filters for FTP, rsh, and other protocols that require back-channel connections. UDP-based services, in general, are also not filterable: packet filters do not maintain state information about previous packets, and without that state, the filter cannot ascertain whether a given UDP packet is a response to an outstanding request.

We address these protocol limitations in our currently-deployed firewall by either dropping packets and causing application-level failure or by providing application-level gateways on bastion hosts. We believe that in the medium-term, protocols will have to change to address problems like spoofing and connection hijacking, as well as accommodate IPv6. We therefore examine what additional protocol changes would allow a realistic packet filter to implement a pure request-response policy.

7.1 TCP Back-Channels

A SYN filtering strategy works extremely well for TCP. The only change we advocate to protocol designers and implementors is that TCP-based protocols should *never* require applications to open a back-channel connection. The problem here is obvious: a request-response filter allows a SYN packet to go out through the firewall but drops the SYN packet by which the external host requests a back-channel. Although a TCP connection originating from outside the firewall might be an application-level "response" to an FTP or rsh request originating within the firewall, a stateless packet filter has no way of knowing this.

Protocols like FTP and rsh use back-channels to create an additional connection over which an application can send a data stream that is, at the application level, asynchronous or conceptually separate from the "main" application connection. FTP uses a back-channel to separate command and data streams; rsh does it to separate UNIX `stdout` and `stderr`. Where a service requires additional TCP connections to implement the desired application semantics, we see no reason why those additional connections must be opened by the server. The same effect can be obtained by having the server bind a local socket, find the local port number, send that port number in an application-level message to the client, and start a passive TCP open. The client then initiates an active open to the specified port on the server which is waiting for the client machine to connect.[4] The result is functionally identical to having the server establish a channel to the client but without requiring a SYN packet from the server through the firewall.

However, replacing TCP back-channels with active connections is not an adequate solution for X, where the roles of client and server are reversed. There is no channel between the X server and any external process over which external clients can bootstrap connections. Addressing this version of the back-channel problem without using an application-level gateway would therefore require far-reaching changes to the X window protocol and is an area for future research.

7.2 Connectionless Protocols

Implementing a request-response filtering policy for UDP-based protocols is far more difficult than for TCP. Because there is no explicit connection, we cannot simply filter connection requests: each UDP (and ICMP) packet must be examined individually. Although a packet filter should only admit packets containing responses to earlier requests, individual packets generally do not carry enough information for a filter to correctly determine whether a packet is a request or a response. Moreover, the large number of UDP-based application protocols makes such protocol-specific packet processing unmanageable.

In addition, any filtering scheme that tries using well-known UDP port numbers to determine the protocol type of a packet is, at best, a heuristic. For example, consider a packet from source address A and UDP port 161 (SNMP) to destination machine B and UDP port 53 (DNS). This packet could be a DNS request from A (not running an SNMP server) with a DNS client coincidentally running on port 161. The same packet could equally well be an SNMP response from A, to a client coincidentally running at port 53 on B, which is not running a DNS server. This

---

[4] The TCP specification [17] (page 45) requires that the server be able to bind the client host as the foreign address on the local socket before initiating the passive TCP open. However, the BSD socket API does not provide this functionality, so the server must explicitly verify that the accepted connection actually originated from the expected client host.

exact example is unlikely between two machines running Unix because of the BSD reserved port semantics; however, it could easily arise when one of the endpoints has been compromised, is running a UDP implementation that has no notion of reserved ports, or is running an unfriendly application (e.g. one that calls `bindresvport()`). A filter rule therefore cannot use either the source or destination port numbers to determine the protocol type of a given packet, which it needs in order to parse the packet and classify it as a request or a response.

We conclude, therefore, that UDP port numbers are unreliable filtering criteria and UDP application protocols are too varied for monitoring in a filter. To support a reliable request-response filter, we must therefore either:

- Add a request-response bit to the UDP header (or equivalently to a well-known location in *all* UDP packets) that applications set and receive with end-to-end significance.
- Partition the UDP port space between clients and servers.

With the first alternative, the packet filter does not need to examine the source or destination ports. It simply drops all inbound packets which have the request-response bit set to "request." The request-response bit must be available to the application so that servers can detect requests masquerading as response packets.

With the second alternative, the filter uses the disjoint client and server port spaces as an implicit request-response bit by verifying that the packet is destined for a UDP port in the client range. This approach assumes that *all* UDP implementations honor the same port space partitioning. Partitioning the UDP port space between clients and servers (e.g. above and below port 32678) can be implemented straightforwardly in Unix kernels. Servers can always explicitly bind to a port in the server range. Clients must allow the kernel to assign a local port, which would fall in the client port space (32768 or above). By using a fixed cutpoint between the client and server portions, the filter rules can use a simple comparison operator which can remain unchanged even as new UDP application protocols are deployed. The concept of "privileged" UDP ports would be abolished. The 4.4-BSD implementation attempts to partition its UDP port space: otherwise unspecified local ports are bound in the range 1025–5000; however, this range is not honored on other systems and overlaps with many IANA-assigned registered ports.

Even with a firewall that discards all inbound UDP packets marked as requests, external hosts can still send anything they want to a client port. Ultimately, only applications themselves have sufficient knowledge and state to perform exact request-response filtering. For example, only the client can detect and discard unsolicited responses from external hosts, as only the client knows exactly which requests are still outstanding. UDP applications receiving a request packet on a client port must treat the packet as an *invalid* protocol operation.

Of course, the final approach to re-designing UDP protocols is to discard UDP in favor of an explicit transaction-oriented protocol. Either VMTP [2] or transactional TCP [1] would allow exact request-response filtering. VMTP has an explicit request-response bit in the header, while with transactional TCP, we can again apply TCP SYN filtering. However, to the best of our knowledge, neither protocol has ever seen production use, and transactional TCP is also unable to address the second use of UDP—namely IP multicast traffic.

7.3 Multicast traffic

We currently see no way to apply the request-response filtering policy for IP multicast applications. In most multicast applications, it is not even meaningful to attempt to classify packets as either requests or responses. Applications use multicast for a wide range of purposes, such as resource location, one-to-many request-response, information dissemination, and peer-to-peer conferencing. Different applications can use the same multicast address simultaneously without restriction. Even within a single application, a single multicast address can be used simultaneously for different purposes.

Consider Van Jacobson's **wb** [13] application as an example. This distributed whiteboard is a peer-to-peer application in which each participant subscribes to, and transmits on, a single multicast address. Though the protocol treats all hosts symmetrically, we can classify individual packets as unsolicited information dissemination (for advertising local additions to the shared whiteboard), one-to-many requests (for requesting the retransmission of a previous dissemination), or one-to-many responses (in which some host near the requesting host re-transmits the requested data). A packet filter cannot discriminate between these different uses of the multicast address without having considerable application-specific knowledge.

Furthermore, filtering any of these three classes of packets would significantly disrupt a wb session. If the dissemination packets are dropped, then local clients only see the locally-contributed whiteboard data. If the retransmission request packets are dropped and only local hosts possess the requested data, then no outside participant will ever see that data. Even if the data is available elsewhere, the inability of local hosts to provide the data introduces an artificial latency into the recovery process because more distant hosts wait to see if hosts near the requester will answer first. Finally, if retransmission responses are dropped, then local hosts may again never see all available whiteboard data. Packet filtering must therefore be all-or-nothing on a per-application basis, meaning that the firewall would need to be informed about the multicast address used by each active application session.

Filtering on a per-multicast address basis is not secure, however. Because multiple applications may coincidentally use the same multicast address simultaneously, multicast applications themselves currently use heuristics to distinguish their own packets from those of other applications. These heuristics typically search for private application and session signatures in every received packet. Packets that do not have the appropriate signatures are dropped. Any packet admitted by the filter may be delivered to several different applications behind the firewall, and we cannot be sure that each of them will properly discard inappropriate packets. Because the application-level signatures are only a heuristic, we face the same masquerading problem as for unicast; however, the number of potential application-level protocol interactions is considerably higher with multicast than with unicast. Consequently, without any further guarantee about how applications handle foreign packets, we must drop all inbound multicast packets.

IP multicast is relatively new, and its implementation and delivery semantics at endpoint hosts are still in flux. In addition, most existing IP multicast-based applications are still undergoing development. We regard this as a perfect opportunity to develop a new non-UDP protocol above IP multicast that provides reliable end-to-end application and session authentication. Only when such a protocol is available can we trust endpoint applications to reliably discard unexpected packets. Development of such a protocol is an area of on-going research.

8 Conclusion

The SURF design meets the needs of research environments. The firewall has three basic elements:

Request-Response Policy: Incoming packets are dropped unless they can be directly linked to a request originating inside the firewall.

Public Image: Use of expendable hosts and conscious selection *and physical separation* of public data and private data.

Virtual Enclaves: Linking isolated, mutually trusting host groups, each protected by their own security perimeter, using an IP tunnel.

As a consequence, the firewall is largely transparent to trusted users and therefore retains the sense of "openness" critical in a research environment. This transparency and perceived openness actually *increase* security by eliminating the temptation for users to bypass our security mechanisms.

Our implementation experience demonstrates that a research firewall can be constructed with low costs in acquisition and maintenance. Because our implementation required no modifications to any operating system kernel, it can be used to protect a heterogeneous set of machine architectures. Indeed, our research environment includes workstations from at least seven vendors. Furthermore, our use of general-purpose software and hardware components allows individual groups to easily customize the set of exported network services and accepted connections.

In deploying our firewall, we have reduced our research group's outside dependencies. We have functioned virtually unaffected even during failures of the campus nameservers and routers and during occasional broadcast packet "storms" caused by misconfigured hosts on the campus networks. For example, by servicing DNS requests from their caches, our internal nameservers can still function during network outages; internal electronic mail delivery is oblivious to the outside network's existence. We feel that the fault-tolerance granted by this autonomy is truly valuable.

However, our experience has also revealed that many existing application protocols are not designed to operate within a secure network environment. We have outlined how protocols might be modified to better fit within a request-response paradigm and therefore obviate the need for application-level proxies on bastion hosts. Ideally, the request-response policy could be enforced entirely by the packet filter, with bastions only used to implement the virtual enclave. This protocol re-design is an area of on-going research.

We observe that implementation of a security policy shares many of the same issues faced in mobile computing environments. In both cases, one seeks to support the autonomy of several "enclaves" while still supporting communication between those disconnected machines. Moreover, office environments are seeing increased use of wireless LANs, so security policies must adapt to protect such environments. We are exploring how secure IP tunneling might be replaced with the encrypted IP used in mobile environments, and we are also exploring how a wireless computing environment would affect security policy.

Rather than using statically-set filter rules, we are considering a security perimeter in which internal hosts dynamically program the filter to control which packets are admitted. Dynamic filtering would allow implementation of an exact request-response filtering policy. It introduces the cost and complexity of a protocol allowing applications to add and remove filter "rules" (i.e. UDP source/destination address/port four-tuples), and timely removal of stale rules left by applications and hosts that crash. Dynamic filtering obviates the need to modify existing protocols, but it requires substantial changes to all application implementations. We intend to investigate whether this is an effective tradeoff, particularly for connection-oriented networks.

In designing the SURF firewall, we have identified that network security for research institutions is a problem in its own right and that traditional corporate firewalls impose excessive restrictions. Research firewalls represent a difficult three-way tradeoff between perceived security risks, user desires for an open research environment, and implementation difficulty. While corporations also face this tradeoff, security usually overshadows all other concerns. Such choices are simply not as clear-cut within research institutions.

Acknowledgements

We are indebted to numerous individuals who enabled us to implement our firewall using off-the-shelf components: John Ioannidis, Neil Haller, MIT project Athena, and Steve Crocker, Marcus J. Ranum, and others at Trusted Information Systems. Because these individuals made their respective software available for non-commercial use, we were able to focus on the design and policy needs of a research environment. In this spirit, we have made our locally-developed software available for anonymous FTP from `gregorio.Stanford.EDU` in `/pub/firewall/`.

Our firewall development resulted from enormous effort by several other members of our research group, particularly Ken Duda, Hugh Holbrook, and Mark Steiglitz. This paper benefited greatly from feedback of our colleagues—particularly Mary Baker, Stuart Cheshire, and Craig Partridge.

The authors were supported by ARPA under contract DABT63-91-K-0001. Michael Greenwald was supported by a Rockwell Fellowship. Sandeep Singhal was supported by a Fannie and John Hertz Foundation Fellowship.

Appendix A Filter Rules

Figure 4 illustrates the filter rules used to implement our security policy discussed in Section 2. If a filtered protocol is needed for our research or for a particular application, then we either run the process on an expendable machine or establish a proxy on a bastion and change the filters.

References

[1] Braden, Bob. “Extending TCP for Transactions.” Internet RFC 1379, November 1992.

[2] Cheriton, David R. “VMTP: A Transport Protocol for the Next Generation of Communication Systems.” In *Proceedings of SIGCOMM 1986*, Pages 406–415, Stowe, VT, August 1986. ACM SIGCOMM. Published as *Computer Communications Review* 16(3), October 1986.

[3] Computer Emergency Response Team. “Sun Sendmail Vulnerability.” CERT Advisory CA:90:01, January 1990.

[4] Computer Emergency Response Team. “Sendmail Vulnerability.” CERT Advisory CA:93:16, November 1993.

[5] Computer Emergency Response Team. “Ongoing Network Monitoring Attacks.” CERT Advisory CA-94:01, February 1994.

[6] Computer Emergency Response Team. “Sendmail Vulnerabilities.” CERT Advisory CA:94:12, July 1994.

[7] Computer Emergency Response Team. “IP Spoofing Attacks and Hijacked Terminal Connections.” CERT Advisory CA:95:01, January 1995.

[8] Computer Emergency Response Team. “Sendmail v.5 Vulnerability.” CERT Advisory CA:95:08, August 1995.

[9] Computer Emergency Response Team. “Sendmail Vulnerabilities.” CERT Advisory CA:95:05, February 1995.

[10] Computer Emergency Response Team. “Sun Sendmail -oR Vulnerability.” CERT Advisory CA:95:11, September 1995.

[11] Computer Emergency Response Team. “Syslog Vulnerability–A Workaround for Sendmail.” CERT Advisory CA:95:13, October 1995.

[12] Crocker, David H. “Standard For the Format of ARPA Internet Text Messages.” Internet RFC 822, August 1982.

[13] Floyd, Sally, Van Jacobson, Charley Liu, Steven McCanne, and Lixia Zhang. “A Reliable Multicast Framework for Light-Weight Sessions and Application-Level Framing.” In *Proceedings of SIGCOMM 1995*, Pages 342–356, Cambridge, MA, August 1995. ACM SIGCOMM. Published as *Computer Communications Review* 25(4), October 1995.

[14] Ganesan, Ravi. “BAfirewall: A Modern Firewall Design.” In *Proceedings of the Internet Society Symposium on Network and Distributed Systems Security*, Pages 99–108, San Diego, CA, February 1994. Internet Society.

[15] Germain, Ellen. “Guarding Against Internet Intruders.” *Science*, 267:608–610, February 1995.

[16] Haller, Neil M. “The S/Key One-Time Password System.” In *Proceedings of the Internet Society Symposium on Network and Distributed System Security*, Pages 151–157, San Diego, CA, February 1994. Internet Society.

[17] Information Sciences Institute. “Transmission Control Protocol.” Internet RFC 793, September 1981.

[18] Ioannidis, J. and M. Blaze. “The Architecture and Implementation of Network-Layer Security Under Unix.” In *UNIX Security Symposium IV Proceedings*, Pages 29–39, Santa Clara, CA, October 1993. USENIX Association.

[19] Kohl, John and B. Clifford Neuman. “The Kerberos Network Authentication Service (V5).” Internet RFC 1510, September 1993.

[20] Mills, David L. “Network Time Protocol (version 3) Specification, Implementation.” Internet RFC 1304, March 1992.

[21] Mockapetris, Paul V. “Domain Names - Implementation and Specification.” Internet RFC 1035, November 1987.

[22] Myers, John G. and Marshall T. Rose. “Post Office Protocol - Version 3.” Internet RFC 1725, November 1994.

[23] Neuman, B. Clifford and Theodore Ts’o. “Kerberos: An Authentication Service for Computer Networks.” *IEEE Communications*, 32(9):33–38, September 1994.

[24] Scheifler, Robert W. and James Gettys. “The X Window System.” *ACM Transactions on Graphics*, 5(2):79–109, April 1986.

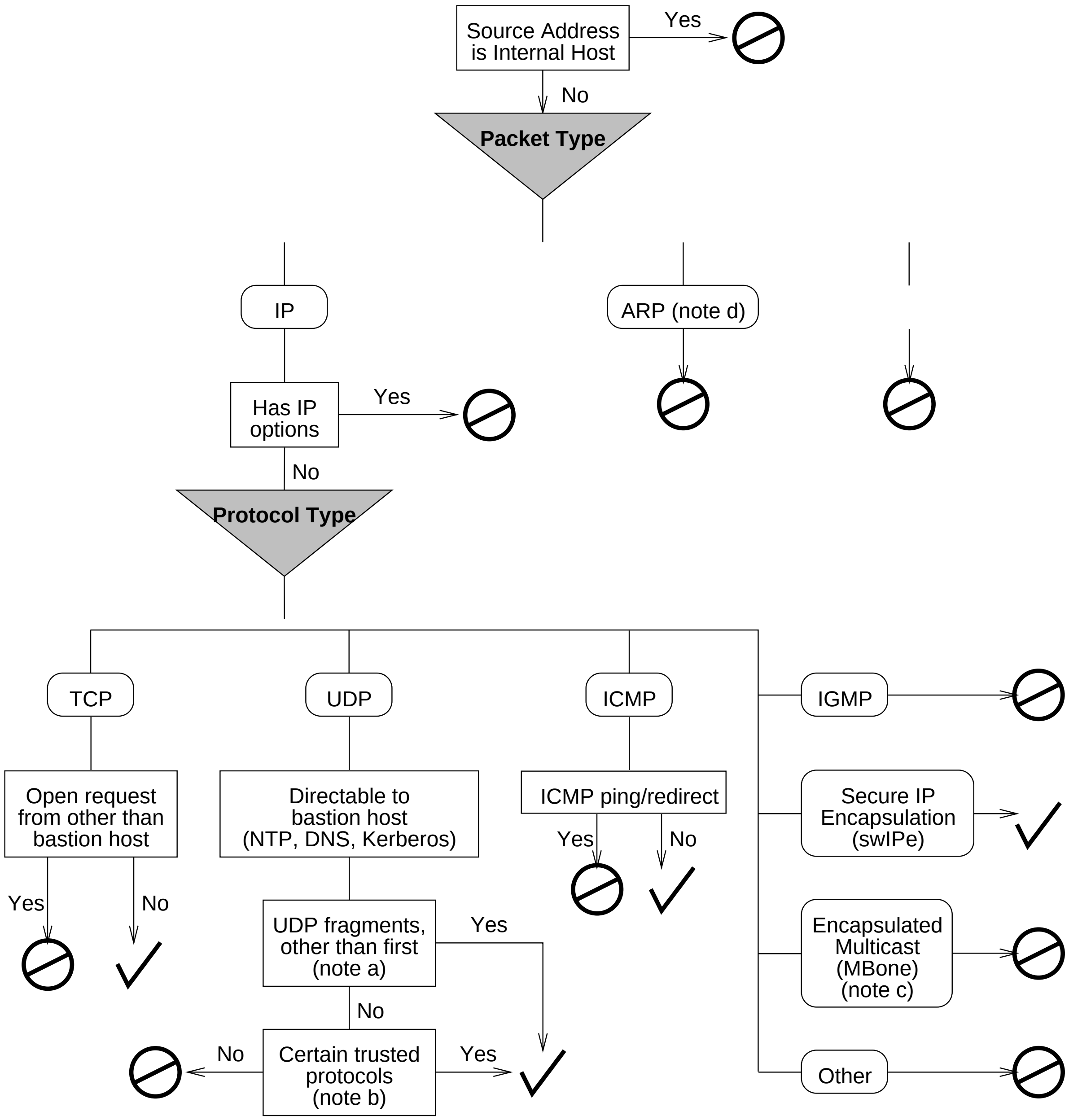
[25] Sun Microsystems, Inc. “RPC: Remote Procedure Call Protocol Specification: Version 2.” Internet RFC 1057, June 1988.

[26] Sun Microsystems, Inc. “NFS: Network File System Specification.” Internet RFC 1094, March 1989.

[27] Violino, Bob. “Break-Ins Are Rife: Survey Suggests an Epidemic of Internet Hacking.” *Information Week*, 268:98, 9 Oct 1995.

All CERT Advisories are published by the CERT Coordination Center, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA. They are available by anonymous FTP from `cert.org`.

All Internet RFCs are published by the USC Information Sciences Institute, Marina Del Rey, CA. They are available by anonymous FTP from `venera.isi.edu`.


Source Address is Internal Host
Yes
No
Packet Type
IP
ARP (note d)
Has IP options
Yes
No
Protocol Type
TCP
UDP
ICMP
IGMP
Open request from other than bastion host
Yes
No
Directable to bastion host (NTP, DNS, Kerberos)
ICMP ping/redirect
Yes
No
Secure IP Encapsulation (swIPe)
UDP fragments, other than first (note a)
Yes
No
Encapsulated Multicast (MBone) (note c)
Certain trusted protocols (note b)
No
Yes
Other