Network (In)Security Through IP Packet Filtering

D. Brent Chapman

Great Circle Associates

Brent@GreatCircle.COM
+1 415 962 0841

1057 West Dana Street
Mountain View, CA 94041

ABSTRACT

Ever-increasing numbers of IP router products are offering packet filtering as a tool for improving network security. Used properly, packet filtering is a useful tool for the security-conscious network administrator, but its effective use requires a thorough understanding of its capabilities and weaknesses, and of the quirks of the particular protocols that filters are being applied to. This paper examines the utility of IP packet filtering as a network security measure, briefly contrasts IP packet filtering to alternative network security approaches such as application-level gateways, describes what packet filters might examine in each packet, and describes the characteristics of common application protocols as they relate to packet filtering. The paper then identifies and examines problems common to many current packet filtering implementations, shows how these problems can easily undermine the network administrator's intents and lead to a false sense of security, and proposes solutions to these problems. Finally, the paper concludes that packet filtering is currently a viable network security mechanism, but that its utility could be greatly improved with the extensions proposed in the paper.

1. Introduction

This paper considers packet filtering as a mechanism for implementing network security policies. The consideration is from the point of view of a site or network administrator (who is interested in providing the best possible service to their users while maintaining adequate security of their site or network, and who often has an "us versus them" attitude with regard to external organizations), which is not necessarily the same point of view that a service provider or router vendor (who is interested in providing network services or products to customers) might have. An assumption made throughout is that a site administrator is generally more interested in keeping outsiders out than in trying to police insiders, and that the goal is to keep outsiders from breaking in and insiders from accidentally exposing valuable data or services, not to prevent insiders from intentionally and maliciously subverting security measures. This paper does not consider military-grade "secure IP" implementations (those that implement the "IP security options" that may be specified in IP packet headers) and related issues; it is limited to what is commonly available for sale to the general public.

Packet filtering may be used as a mechanism to implement a wide variety of network security policies. The primary goal of these policies is generally to prevent unauthorized

Published in *Proceedings of the Third USENIX UNIX Security Symposium*; Baltimore, MD; September, 1992.

network access without hindering authorized network access; the definitions of "unauthorized access" and "authorized access" vary widely from one organization to another. A secondary goal is often that the mechanisms be transparent in terms of performance, user awareness, and application awareness of the security measures. Another secondary goal is often that the mechanisms used be simple to configure and maintain, thus increasing the likelihood that the policy will be correctly and completely implemented; in the words of Bill Cheswick of AT&T Bell Laboratories, "Complex security isn't". Packet filtering is a mechanism which can, to a greater or lesser extent, fulfill all these goals, but only through thorough understanding of its strengths and weaknesses and careful application of its capabilities.

Several factors complicate implementation of these policies using packet filtering, including asymmetric access requirements, differing requirements for various internal and external groups of machines, and the varying characteristics of the particular protocols, services, and implementations of these protocols and services that the filters are to be applied to. Asymmetric access requirements usually arise when an organization desires that its internal systems have more access to external systems than vice versa. Differing requirements arise when an organization desires that some groups of machines have different network access privileges than other groups of machines (for instance, the organization may feel that a particular subnet is more secure than standard, and thus can safely take advantage of expanded network access, or they may feel that a particular subnet is especially valuable, and thus its exposure to the external network should be as limited as possible). Alternatively, an organization may desire to allow more or less network access to some specific group of external machines than to the rest of the external world (for instance, a company might want to extend greater network access than usual to a key client with whom they are collaborating, and less network access than usual to a local university which is known to be the source of repeated cracker attacks). The characteristics of particular protocols, services, and implementations also greatly affect how effective filtering can be; this particular issue is discussed in detail below, in Section 3 and Appendix A.

Common alternatives to packet filtering for network security include securing each machine with network access and using application gateways. Allowing network access on an all-or-nothing basis (a very coarse form of packet filtering) then attempting to secure each machine that has network access is generally impractical; few sites have the resources to secure and then monitor every machine that needs even occasional network access. Application gateways, such as those used by AT&T [Ches90], Digital Equipment Corporation [Ranum92], and several other organizations, are also often impractical because they require internal hosts to run modified (and often custom-written or otherwise not commonly available) versions of applications (such as "ftp" and "telnet") in order to reach external hosts. If a suitably modified version of an application is not available for a given internal host (a modified TELNET client for a personal computer, for instance), that internal host's users are simply out of luck and are unable to reach the past the application gateway.

2. How Packet Filtering Works

2.1. What packet filters base their decisions on

Current IP packet filtering implementations all operate in the same basic fashion; they parse the headers of a packet and then apply rules from a simple rule base to determine whether to route or drop† the packet. Generally, the header fields that are available to the filter

† "Permit" and "deny" are used synonymously with "route" and "drop" throughout this paper. If a router decides to "permit" or "route" a packet, it is passed through to its destination as if filtering never occurred. If a router decides to "deny" or "drop" a packet, the packet is simply discarded, as if it never existed; depending on the filter-

are packet type (TCP, UDP, etc.), source IP address, destination IP address, and destination TCP/UDP port. For some reason, the source TCP/UDP port is often *not* one of the available fields; this is a significant deficiency discussed in detail in Section 4.2.

In addition to the information contained in the headers, many filtering implementations also allow the administrator to specify rules based on which router interface the packet is destined to go out on, and some allow rules based on which interface the packet came in on. Being able to specify filters on both inbound and outbound† interfaces allows you significant control over where the router appears in the filtering scheme (whether it is "inside" or "outside" your packet filtering "fence"), and is very convenient (if not essential) for useful filtering on routers with more than two interfaces. If certain packets can be dropped using inbound filters on a given interface, those packets don't have to be mentioned in the outbound filters on all the other interfaces; this simplifies the filtering specifications. Further, some filters that an administrator would like to be able to implement require knowledge of which interface a packet came in on; for instance, the administrator may wish to drop all packets coming inbound from the external interface that claim to be from an internal host, in order to guard against attacks from the outside world that use faked internal source addresses.

Some routers with very rudimentary packet filtering capabilities don't parse the headers, but instead require the administrator to specify byte ranges within the header to examine, and the patterns to look for in those ranges. This is almost useless, because it requires the administrator to have a very detailed understanding of the structure of an IP packet. It is totally unworkable for packets using IP option fields within the IP header, which cause the location of the beginning of the higher-level TCP or UDP headers to vary; this variation makes it very difficult for the administrator to find and examine the TCP or UDP port information.

2.2. How packet filtering rules are specified

Generally, the filtering rules are expressed as a table of conditions and actions that are applied in a certain order until a decision to route or drop the packet is reached. When a particular packet meets all the conditions specified in a given row of the table, the action specified in that row (whether to route or drop the packet) is carried out; in some filtering implementations [Mogul89], the action can also indicate whether or not to notify the sender that the packet has been dropped (through an ICMP message), and whether or not to log the packet and the action taken on it. Some systems apply the rules in the sequence specified by the administrator until they find a rule that applies [Mogul89][Cisco90], which determines whether to drop or route the packet. Others enforce a particular order of rule application based on the criteria in the rules, such as source and destination address, regardless of the order in which the rules were specified by the administrator. Some, for instance, apply filtering rules in the same order as

ing implementation (and sometimes on the filtering specification), the router might send an ICMP message (usually "host unreachable") back to the source of a packet that is dropped, or it might simply pretend it never received the packet.

† Throughout this paper, the terms "inbound" and "outbound" are usually used to refer to connections or packets from the point of view of the protected network as a whole, and sometimes used to refer to packets from the point of view of the filtering router (which is at the edge of the internal network, between the internal network and the external world), or to the router interfaces those packets will pass through. A packet might appear to be "inbound" to the filtering router on its way to the external world, but that packet is "outbound" from the internal network as a whole. An "outbound connection" is a connection initiated from a client on an internal machine to a server on an external machine; note that while the connection as a whole is outbound, it includes both outbound packets (those from the internal client to the external server) and inbound packets (those from the external server back to the internal client). Similarly, an "inbound connection" is a connection initiated from a client on an external machine to a server on an internal machine. The "inbound interface" for a packet is the interface on the filtering router that the packet appeared on, while the "outbound interface" is the interface the packet will go out on if it isn't denied by the application of the filtering rules.

routing table entries; that is, they apply rules referring to more specific addresses (such as rules pertaining to specific hosts) before rules with less specific addresses (such as rules pertaining to whole subnets and networks) [CHS91][Telebit92a]. The more complex the way in which the router reorders rules, the more difficult it is for the administrator to understand the rules and their application; routers which apply rules in the order specified by the administrator, without reordering the rules, are easier for an administrator to understand and configure, and therefore more likely to yield correct and complete filter sets.

2.3. A packet filtering example

For example, consider this scenario. The network administrator of a company with Class B network 123.45 wishes to disallow access from the Internet to his network in general (123.45.0.0/16)†. The administrator has a special subnet in his network (123.45.6.0/24) that is used in a collaborative project with a local university which has class B network 135.79; he wishes to permit access to the special subnet (123.45.6.0/24) from all subnets of the university (135.79.0.0/16). Finally, he wishes to deny access (except to the subnet that is open to the whole university) from a specific subnet (135.79.99.0/24) at the university, because the subnet is known to be insecure and a haven for crackers. For simplicity, we will consider only packets flowing from the university to the corporation; symmetric rules (reversing the SrcAddr and DstAddr in each of the rules below) would need to be added to deal with packets from the corporation to the university. Rule C is the "default" rule, which specifies what happens if none of the other rules apply.

Rule	**SrcAddr**	**DstAddr**	**Action**
A	135.79.0.0/16	123.45.6.0/24	permit
B	135.79.99.0/24	123.45.0.0/16	deny
C	0.0.0.0/0	0.0.0.0/0	deny

Consider these "sample" packets, their desired treatment under the policy outlined above, and their treatment depending on whether the rules above are applied in order "ABC" or "BAC".

Packet	**SrcAddr**	**DstAddr**	**Desired Action**	**ABC action**	**BAC action**
1	135.79.99.1	123.45.1.1	deny	deny (B)	deny (B)
2	135.79.99.1	123.45.6.1	permit	permit (A)	*deny (B)*
3	135.79.1.1	123.45.6.1	permit	permit (A)	permit (A)
4	135.79.1.1	123.45.1.1	deny	deny (C)	deny (C)

A router that applies the rules in the order ABC will achieve the desired results: packets from the "hacker haven" subnet at the university to the company network in general (such as packet 1 above) will be denied (by rule B), packets from the university "hacker haven" subnet at the university to the company's collaboration subnet (such as packet 2 above) will be

† Throughout this paper, the syntax "*a.b.c.d*/*y*" denotes "the address *a.b.c.d*, with the top *y* bits significant for comparison". In other words, 123.45.0.0/16 means that the top 16 bits (123.45) are significant for comparisons to other addresses. The address 123.45.6.7 thus matches 123.0.0.0/8, 123.45.0.0/16, and 123.45.6/24, but not 123.45.99.0/24. A pattern with 0 significant bits (such as 0.0.0.0/0) matches any address, while a pattern with 32 significant bits (such as 123.45.6.7/32) matches only that particular address (123.45.6.7). This syntax is a simpler form of expressing an address pattern than the traditional "address, wildcard mask" tuple, particularly when the boundary between the wildcarded and non-wildcarded bits doesn't fall on an 8-bit boundary (for instance, on a Cisco router, the pattern 123.0.0.0/8 would be represented as "123.0.0.0 0.255.255.255", 123.45.6.0/24 would be represented as "123.45.6.0 0.0.0.255", and 123.45.6.240/28 would be represented as "123.45.6.240 0.0.0.15"). This syntax was originated in the *KA9Q* networking package for PCs, and is used in the Telebit *NetBlazer* and other products.

permitted (by rule A), packets from the university's general network to the company's "open" subnet (such as packet 3 above) will be permitted (by rule A), and packets from the university's general network to the company's general network (such as packet 4 above) will be denied (by rule C).

If, however, the router reorders the rules by sorting them into order by number of significant bits in the source address then number of significant bits in the destination address, the same set of rules will be applied in the order BAC. If the rules are applied in the order BAC, packet 2 will be denied, when we want it to be permitted.

2.4. Packet filtering caveats

2.4.1. Complexity of packet filtering specifications

In fact, there's a subtle error in this example that illustrates how difficult it is to correctly set up filters using such low-level specifications. Rule B above, which appears to restrict access from the "hacker haven" net, is actually superfluous and unnecessary, and is the cause of the incorrect denial of packet 2 if the rules are applied in the order BAC. If you remove rule B, both types of routers (those that apply rules in the order specified, and those that reorder rules by number of significant bits in source or destination addresses) will process the rules in the order AC. When processed in that order, the result table becomes:

Packet	SrcAddr	DstAddr	Desired Action	AC action
1	135.79.99.1	123.45.1.1	deny	deny (C)
2	135.79.99.1	123.45.6.1	permit	permit (A)
3	135.79.1.1	123.45.6.1	permit	permit (A)
4	135.79.1.1	123.45.1.1	deny	deny (C)

There are two points here. First, correctly specifying filters is difficult. Second, reordering filtering rules makes correctly specifying filters even more difficult, by turning a filter set that works (even if it's in fact overspecified) if evaluated in the order given into a filter set that doesn't work.

Even though the example presented above is a relatively simple application of packet filtering, most administrators will probably read through it several times before they feel they understand what is going on. Consider that the more difficult the rules are to comprehend, the less likely the rules will be correct and complete. The way in which filtering rules must be specified and the order in which they are applied are key determinants of how useful and powerful a given router's filtering capabilities are. Most implementations require the administrator to specify filters in ways which make the filters easy for the router to parse and apply, but make them very difficult for the administrator to comprehend and consider.

2.4.2. Reliance on accurate IP source addresses

Most filtering implementations, of necessity, rely on the accuracy of IP source addresses to make filtering decisions. IP source addresses can easily be faked, however, as discussed in [Bellovin89], [Kent89], [Bellovin92a], and [Bellovin92b]. This is a particular case where being able to filter inbound packets is useful. If a packet that appears to be from one internal machine to another internal machine comes in over the link from the outside world, you should be mighty suspicious. If your router can be told to drop such packets using inbound filters on the external interface, your filtering specifications for internal interfaces can be made both much simpler and more secure.

2.4.3. Dangers of IP source routing

Another IP feature ripe for potential abuse is IP source routing. Essentially, an IP packet with source routing information included tells routers how to route the packet, rather than letting the routers decide for themselves. An attacker could use this to their advantage [Bellovin89]. Unless you have a specific need to allow packets with IP source routes between your internal network and the outside world, it's probably a good idea for your router to ignore IP source route instructions; whether source routing can be disabled, whether it is enabled or disabled by default, and how to disable it vary from vendor to vendor.

2.4.4. Complications due to IP fragmentation

Yet another complication to packet filtering is IP packet fragmentation. IP supports the notion that any router along a packet's path may "fragment" that packet into several smaller packets, to accommodate the limitations of underlying media, to be reassembled into the original IP packet at the destination. For instance, an FDDI frame is much larger than an Ethernet frame; a router between an FDDI ring and an Ethernet may need to split an IP packet that fit in a single FDDI frame into multiple fragments that fit into the smaller Ethernet frames. The problem with this, from a packet filtering point of view, is that only the first of the IP fragments has the higher-level protocol (TCP or UDP) headers from the original packet, which may be necessary to make a filtering decision concerning the fragment. Different filtering implementations take a variety of responses to this situation. Some apply filters only to the first fragment (which contains the necessary higher-level protocol headers), and simply route the rest, on the assumption that if the first fragment is dropped by the filters, the rest of the fragments can't be reassembled into a full packet, and will cause no harm [CHS91]. Others keep a cache of recently-seen first fragments and the filtering decision that was reached, and look up non-first fragments in this cache in order to apply the same decision [Mogul89]. In particular, it is dangerous to suppress only the first fragment of outbound packets; you might be leaking valuable data in the non-first fragments that are routed on out.

3. Filtering-Related Characteristics of Application Protocols

Each application protocol has its own particular characteristics that relate to IP packet filtering, that may or may not differ from other protocols. Particular implementations of a given protocol also have their own characteristics that are not a result of the protocol per se, but a result of design decisions made by the implementors. Since these implementation characteristics are not covered in the specification of the protocol (though they aren't counter to the specification), they are likely to vary between different implementations of the same protocol, and might change even within a given implementation as that implementation evolves. These characteristics include what port a server uses, what port a client uses, whether the service is typically offered over UDP or TCP or both, and so forth. An understanding of these characteristics is essential for setting up effective filters to allow, disallow, or limit the use of these protocols. Appendix A discusses in detail the filtering-related characteristics of several common protocols.

3.1. "Random" ports aren't really random

Although implementations of various protocols might appear to use a "random" ports for the client end and a well-known port for the server end, the ports chosen for the client end used are usually not totally random. While not explicitly supported by the RFCs, systems based on BSD UNIX usually reserve ports below 1024 for use by "privileged" processes, and allow only processes running as root to bind to those ports; conversely, non-privileged processes must use ports at or above 1024. Further, if a program doesn't request a particular port, it is often simply assigned the port after the last one assigned; if the last port assigned was 5150, the next one assigned will probably be 5151.

3.2. Privileged versus non-privileged ports

The distinction between "privileged" and "non-privileged" ports (those below 1024 and at or above 1024, respectively) is found throughout BSD-based systems (and other systems that draw from a BSD background; keep in mind that almost all UNIX IP networking, including SysV IP networking, draws heavily from the original BSD network implementation). This distinction is not codified in the RFCs, and is therefore best regarded as a widely used convention, but not as a standard. Nonetheless, if you're protecting UNIX systems, the convention can be a useful one. You can, for instance, generally forbid all inbound connections to ports below 1024, and then open up specific exceptions for specific services that you wish to enable the outside world to use, such as SMTP, TELNET, or FTP; to allow the "return" packets for connections to such services, you allow all packets to external destination ports at or above 1024.

While it would simplify filtering if all services were offered on ports below 1024 and all clients used ports at or above 1024, many vulnerable services (such as X, OpenWindows, and a number of database servers) use server ports at or above 1024, and several vulnerable clients (such as the Berkeley r* programs) use client ports below 1024. These should be carefully excepted from the "allow all packets to destination ports at or above 1024" type of rules that allow return packets for outbound services.

4. Problems With Current Packet Filtering Implementations

IP packet filtering, while a useful network security tool, is not a panacea, particularly in the form in which it is currently implemented by many vendors. Problems with many current implementations include complexity of configuration and administration, omission of the source UDP/TCP port from the fields that filtering can be based on, unexpected interactions between "unrelated" parts of the filter rule set, cumbersome filter specifications forced by simple specification mechanisms, a lack of testing and debugging tools, and an inability to deal effectively with RPC-based protocols such as YP/NIS and NFS.

4.1. Filters are difficult to configure

The first problem with many current IP packet filtering implementations as network security mechanisms is that the filtering is usually very difficult to configure, modify, maintain, and test, leaving the administrator with little confidence that the filters are correctly and completely specified. The simple syntax used in many filtering implementations makes life easy for the router (it's easy for the router to parse the filter specifications, and fast for the router to apply them), but difficult for the administrator (it's like programming in assembly language). Instead of being able to use high-level language abstractions ("if this and that and not something-else then permit else deny"), the administrator is forced to produce a tabular representation of rules; the desired behavior may or may not map well on to such a representation.

Administrators often consider networking activity in terms of "connections", while packet filtering, by definition, is concerned with the packets making up a connection. An administrator might think in terms of "an inbound SMTP connection", but this must be translated into at least two filtering rules (one for the inbound packets from the client to the server, and one for the outbound packets from the server back to the client) in a table-driven filtering implementation. The concept of a connection is applied even when considering a connectionless protocol such as UDP or ICMP; for instance, administrators speak of "NFS connections" and "DNS connections". This mismatch between the abstractions used by many administrators and the mechanisms provided by many filtering implementations contributes to the difficulty of correctly and completely specifying packet filters.

4.2. TCP and UDP source port are often omitted from filtering criteria

Another problem is that current filtering implementations often omit the source UDP/TCP port from consideration in filtering rules, leading to common cases where it is impossible to allow both inbound and outbound traffic to a service without opening up gaping holes to other services. For instance, without being able to consider both the source and destination port numbers of a given packet, you can't allow inbound SMTP connections to internal machines (for inbound email) and outbound SMTP connections to all external machines (so that you can send outbound mail) without ending up allowing all connections between internal and external machines where both ends of the connection are on ports at or above port 1024. To see this, imagine your router's rule table has 6 variables for rules on a given interface: direction (whether the packet is inbound to or outbound from internal network), packet type (UDP or TCP), source address, destination address, destination port, and action (whether to drop or route the packet). You would need 5 rules in such a table to allow both inbound SMTP (where an external host connects to an internal host to send email) and outbound SMTP (where an internal host connects to any external host to send mail). The rules would look something like this:

Rule	**Direction**	**Type**	**SrcAddr**	**DstAddr**	**DstPort**	**Action**
A	in	TCP	external	internal	25	permit
B	out	TCP	internal	external	>=1024	permit
C	out	TCP	internal	external	25	permit
D	in	TCP	external	internal	>=1024	permit
E	either	any	any	any	any	deny

The default action (rule E), if none of the preceding rules apply, is to drop the packet.

Rules A and B, together, allow the "inbound" SMTP connections; for inbound packets, the source address is an "external" address, the destination address is "internal", and the destination port is 25, while for outbound packets, the source address is "internal", the destination address is "external", and the destination port is at or above 1024. Rules C and D, together, similarly allow the "outgoing" SMTP connections. Consider, however, a TCP connection between an internal host and an external host where both ports used in the connection are above 1023. Incoming packets for such a connection will be passed by rule D. Outgoing packets for such a connection will be passed by rule B. The problem is that, while rules A and B together do what you want and rules C and D together do what you want, rules B and D together allow all connections between internal and external hosts where both ends of the connection are on a port number above 1024. Current filter specification syntaxes are ripe with opportunities for such unexpected and undesired interactions.

If source port could be examined in making the routing decisions, the rule table above would become:

Rule	**Direction**	**Type**	**SrcAddr**	**DstAddr**	**SrcPort**	**DstPort**	**Action**
A	in	TCP	*external*	*internal*	>=1024	25	permit
B	out	TCP	*internal*	*external*	25	>=1024	permit
C	out	TCP	*internal*	*external*	>=1024	25	permit
D	in	TCP	*external*	*internal*	25	>=1024	permit
E	either	any	*any*	*any*	any	any	deny

In this case, all the rules are firmly anchored to port 25 (the well-known port number for SMTP) at one end or the other, and you don't have the problem of inadvertently allowing all connections where both ports are at or above 1024. Consider again the example given above, a TCP connection between an internal and an external host where both ends of the connection were at or above 1024; such a connection doesn't qualify with any of the above filtering rules, since in all of the above rules, one end of the connection has to be at port 25.

4.3. Special handling of start-of-connection packets is impossible

Note that the even the above filters with source port still don't protect your servers living at or above port 1024 from an attack launched from port 25 on an external machine (which is certainly possible if the person making the attack controls the machine the attack is coming from); rules C and D will allow this. One way to defeat this type of attack is to suppress TCP start-of-connection packets (packets with the TCP "SYN" flag set) in rule C; at least one filter implementation provides a mechanism for stating that rules apply *only* to packets in "established" connections (those packets without the SYN bit set) [Cisco90].

Unfortunately, UDP sessions are "connectionless", so there is never a "start-of-connection" packet that can be suppressed in a UDP session. A solution for UDP is often to disallow UDP entirely except for a specific exception for DNS. This exception for DNS can generally be made safely even with a filtering implementation that ignores source port, because of a quirk in the most common DNS implementation. The quirk causes DNS server-to-server queries made over UDP to always use port 53 at both ends of the connection, rather than a random port at one end. Disallowing UDP except for DNS also allows you to avoid most of the problems with filtering RPC-based services (since most RPC services are UDP based) that are discussed in Section 4.6.

4.4. Tabular filtering rule structures are too cumbersome

While tabular rule structures such as those shown above are relatively easy and thus efficient for the router to parse and apply, they rapidly become too cumbersome for the administrator to use to specify complex independent filtering requirements. Even simple applications of these cumbersome syntaxes are difficult, and often have unintended and undesired side effects, as demonstrated in Section 4.2.

4.5. Testing and monitoring filters is difficult

With many router products, the beleagured administrator's life is further complicated by a lack of built-in mechanisms to test the filter set or to monitor its performance in action. This makes it very difficult to debug and validate filtering rule sets, or to modify existing rule sets; the administrator always has to wonder if the filtering rules are really accomplishing what was intended, or if the rule set has some inadvertent hole in it that the administrator has somehow overlooked.

4.6. RPC is very difficult to filter effectively

Finally, RPC-based protocols offer a special challenge, since they don't reliably appear on a given UDP or TCP port number. The only RPC-related service that is guaranteed to be at a certain port is the "portmapper" service. Portmapper maps an RPC service number (which is a 32-bit number assigned by Sun Microsystems to each individual RPC service, including services created by users and other vendors) to the particular TCP or UDP port number (which are much smaller 16-bit numbers) that the service is currently using on the particular machine being queried. When an RPC-based service starts up, it registers with the portmapper to announce what port it is living at; the portmapper then passes this info along to anyone who requests it.

The portmapper isn't required in order to establish an RPC connection, except to determine exactly which port to establish the connection to; if you know (or can guess) which port to establish the connection to, you can bypass the portmapper altogether. What port a given RPC protocol (such as YP/NIS, NFS, or any of a number of others) ends up using is random enough that the administrator can't effectively specify filters for it (at least, not without risking the inadvertent filtering of something else that happened to end up on the same port the administrator thought an RPC-based service *might* end up at), but not so random that an attacker can't easily "guess" where a given protocol lives. Even if they can't or don't guess, a systematic search of the entire port number space for the RPC service they're interested in attacking is not that difficult. Since RPC-based services might be on any port, the filtering implementation has no sure way of recognizing what is and what isn't RPC; as far as the router is concerned, it's all just UDP or TCP traffic.

Two fortuitous characteristics of most RPC-based services can be used to save us from this morass, however. First, most RPC-based services are offered as only on UDP ports; we can simply drop UDP packets altogether except for DNS, as described above. Second, almost all of those that are offered on TCP ports use ports below 1024, which can be protected by an "deny all ports below 1024 except specific services like SMTP" type of filter, such as shown in the example in Section 4.2.

5. Possible Solutions for Current Packet Filtering Problems

5.1. Improve filter specification syntax

The major improvement that could be made to many vendor packet filtering implementations would be to provide better filter specification mechanisms. The administrator should be able to specify rules in a form that makes sense to the administrator (such as a propositional logic syntax), not necessarily a form that is efficient for the router to process; the router can then convert the rules from the high-level form to a form amenable to efficient processing. One possibility might be the creation of a "filter compiler" that accepts filters in a high-level syntax that was convenient for the administrator, and emits a "compiled" filter list that is acceptable to the router.

Addressing the conceptual mismatch between administrators, who think in terms of connections, and routers, which operate in terms of the packets making up those connections, as discussed in Section 4.1, might also prove valuable.

5.2. Make all relevant header fields available as filtering criteria

The administrator should be able to specify all relevant header fields, particularly including TCP/UDP source port (which is currently often omitted from many filtering implementations), as filter criteria. Until this key feature is provided, it will be difficult or impossible to effectively use filtering in certain common situations, as demonstrated in the example in Section 4.2. The administrator should also be able to specify whether a filter rule should apply only to established TCP connections.

5.3. Allow inbound filters as well as outbound filters

The administrator should be able to specify both inbound and outbound filters on each interface, rather than only outbound filters. This would allow the administrator to position the router either "inside" or "outside" the filtering "fence", as appropriate. It would also allow simpler specification of filters on routers with more than two interfaces by allowing some cases (such as a packet appearing from the outside world that purports to be both to and from internal hosts) to be handled by the inbound set of filters on the external interface, rather than having to duplicate these special cases into the outbound filter set on each internal interface. The desired functionality may not even be possible with only outbound filters; the case of a fake internal-to-internal packet showing up on the external interface, as discussed in Section 2.4.2, can't be detected in an outbound filter set.

5.4. Provide tools for developing, testing, and monitoring filters

Better tools for developing, testing and validating rule sets, perhaps including test suites and automatic test probe generators, would make a big difference in the usability of packet filtering mechanisms. Such an automated test system might well be a part of the "filter compiler" described in Section 5.1.

5.5. Simplify specification of common filters

It would be useful if administrators could specify common filtering cases (for instance, "allow inbound SMTP to this single host") simply, without having to understand the details of the protocols or filtering mechanisms involved.

6. Conclusions

Packet filtering is currently a viable and valuable network security tool, but some simple vendor improvements could have a big impact. There are several critical deficiencies that seem to be common to many vendors, such as the inability to consider source TCP/UDP port in filters, that need to be addressed. Other improvements to filter specification mechanisms could greatly simplify the lives of network administrators trying to use packet filtering capabilities, and increase their confidence that their filters are doing what they think they are.

7. Acknowledgements

Thanks to Steve Bellovin and Bill Cheswick of AT&T Bell Laboratories for several lively and fruitful discussions of packet filtering as a network security tool; in particular, I'd like to thank Steve for providing me with prepublication copies of two of his IP security-related papers and of his 1989 article on TCP/IP security problems. Thanks to Ed DeHart of the Computer Emergency Response Team for strongly and repeatedly encouraging me to write this paper after listening to me moan about the issues discussed herein. Thanks to Elizabeth Zwicky of SRI International, Brian Lloyd of Lloyd & Associates, and Steve Bellovin of AT&T Bell Laboratories for reviewing drafts of this paper and providing valuable feedback and suggestions.

8. References

[Bellovin89]
S. M. Bellovin, "Security Problems in the TCP/IP Protocol Suite"; *Computer Communications Review*, Volume 9, Number 2; April 1989; pp. 32-48.

[Bellovin92a]
Steven M. Bellovin, "Packets Found on an Internet"; in preparation; 1992.

[Bellovin92b]
Steven M. Bellovin, "There Be Dragons"; *Proceedings of the Third USENIX UNIX Security Symposium*; Baltimore, MD; September, 1992.

[Ches90]
Bill Cheswick, "The Design of a Secure Internet Gateway"; *Proceedings of the USENIX Summer 1990 Conference*; Anaheim, CA; June 11-15, 1990; pp. 233-237.

[CHS91]
Bruce Corbridge, Robert Henig, Charles Slater, "Packet Filtering in an IP Router"; *Proceedings of the Fifth USENIX Large Installation and System Administration Conference (LISA V)*; San Diego, CA; October, 1992; pp. 227-232.

[Cisco90]
Cisco Systems (Menlo Park, CA); "Gateway System Manual; Software Release 8.2"; 1990.

[CMQ92]
Smoot Carl-Mitchell and John S. Quarterman, "Building Internet Firewalls"; *UnixWorld*; February, 1992; pp 93-102.

[Comer91]
Douglas E. Comer, *Internetworking with TCP/IP, Volume I*; Second Edition, 1991; Prentice-Hall, Inc.

[Kent89]
Stephen Kent, "Comments on 'Security Problems in the TCP/IP Protocol Suite'"; *Computer Communications Review*; July 1989.

[Mogul89]
Jeffrey C. Mogul, "Simple and Flexible Datagram Access Controls for UNIX-based Gateways"; *Proceedings of the USENIX Summer 1989 Conference*; pp. 203-221.

[Ranum92]
Marcus J. Ranum, "A Network Firewall"; *Proceedings of the World Conference on System Administration and Security*; July 1992; Washington, D.C.; pp. 153-163.

[RFC1058]
C. Hedrick, "Routing Information Protocol", Request For Comments 1058; available from the DDN Network Information Center (NIC.DDN.MIL).

[RFC1340]
J. Reynolds and J. Postel, "Assigned Numbers", Request For Comments 1340; available from the DDN Network Information Center (NIC.DDN.MIL).

[Telebit92a]
Telebit Corporation (Sunnyvale, CA), "NetBlazer Command Reference"; 1992.

[Telebit92b]
Telebit Corporation (Sunnyvale, CA), "NetBlazer Version 1.4 Release Notes"; 1992.

Appendix A — Filtering Characteristics of Common IP Protocols

A.1. SMTP

SMTP is provided as a TCP service with the server end of the connection at port 25 and the client end at a random port.

A.2. TELNET

TELNET is provided as a TCP service with the server end of the connection at port 23, and the client end at a random port.

A.3. FTP

FTP is slightly tricky, in that an FTP conversation actually involves two TCP connections in typical UNIX implementations: one for connection for commands, and one for data. The command connection is at port 21 on the server, and the data connection is at port 20 on the server; both connections use random ports on the client side.

A.4. NNTP

NNTP is provided as a TCP service with the server end at port 119, and the client end at a random port.

A.5. DNS

DNS is provided as both a TCP and UDP service at port 53. The UDP service is usually used for client-to-server queries (the client end will be at a random port) and server-to-server proxy queries (where a server queries another server on behalf of a client), while the TCP service is usually used for server-to-server bulk data transfers (typically zone transfers from primary to secondary DNS servers for a given zone).

One implementation characteristic of the most common DNS server implementation (the "BIND", or "Berkeley Internet Name Daemon," implementation) is that server-to-server proxy queries are made via UDP with both ends of the connection using port 53. Packet filtering specifications can take good advantage of this characteristic, since DNS is often the only UDP-based protocol that sites want to allow bidirectionally (i.e., allow both inbound and outbound) between their internal machines and the outside world. The fact that DNS uses port 53 for both ends of such a connection, rather than port 53 for answering server end and a random port for the requesting server end, allows DNS to be bidirectionally enabled in filtering implementations that examine only destination ports (not source ports) *without* running afoul of the "allowing any connection where both ends are above 1023" problem with allowing bidirectional services in such routers (see Section 4.2 for a detailed discussion of this problem).

A.6. BSD r* services (rlogin, rsh, rcp, and rexec)

The BSD r* services (rlogin, rsh, rcp, and rexec) are another tricky case because they use privileged ports (ports below 1024; see below for a discussion of "privileged" and "non-privileged" ports) for both the server (port 512 for rexec, 513 for rlogin, and 514 for rsh and rcp) and client (a random privileged port). A typical filtering set that allows outbound services by allowing outbound packets to specific privileged ports and inbound packets to non-privileged ports won't allow any of these services, since their inbound packets will be coming to random privileged ports. If you then allow inbound packets to random privileged ports, you've just opened up all your own services on privileged ports to attacks from the outside world. One possible solution is to this quandry is to allow only packets from "established" connections inbound, if your filtering implementation has that capability (see Section 4.3).

A.7. RIP

RIP broadcasts between routers uses UDP port 520 as for both source and destination. A RIP query may use some other UDP port as their source port with 520 as the destination port; replies to the query will use 520 as the source port and the query's source port as the reply's destination port [RFC1058].

A.8. RPC and RPC-based services (YP/NIS and NFS)

RPC (Sun's Remote Procedure Call mechanism, which is at the heart of a number of other protocols, notably YP/NIS and NFS) is a real can of worms when it comes to packet filtering. The only ports a machine running RPC is certain to be using are UDP and TCP ports 111, for the "portmapper" process which maps requests for specific RPC services to the particular ports (somewhat randomly determined) that they are running on at the moment on that particular machine. See the complete discussion of the problems with filtering RPC and RPC-based services in Section 4.6.

A.9. Window systems

Various window systems vary in what ports they use. X11, for instance, typically uses TCP port 6000 for the first display on a given machine, port 6001 for the second display (if the machine has a second display), and so forth; to protect machines running X11 servers, you must filter ports 6000 through 6000+*n*, where *n* is the maximum number of X11 servers running on any single machine behind your filtering screen.

OpenWindows uses port 2000.

A.10. ICMP

ICMP is a protocol parallel to TCP and UDP, layered on top of IP, that is used to transmit control, information, and error messages between the IP software on different machines. Rather than having source or destination ports, ICMP packets simply have a "type" code that indicates the nature of the ICMP packets. Most packet filtering implementations can filter ICMP packets by type in the same way as they can filter TCP or UDP by port. Some of these ICMP packet types are informational in nature (such as messages that a packet failed to reach its destination because the destination is unreachable or because the packet traveled through too many routers enroute and timed out), and should almost certainly be permitted through filters. Other ICMP packet types are useful for network management and debugging (such as "echo request" and "echo reply" messages), and should probably be permitted through filters. Still other ICMP packet types are instructions (such as "redirect") that probably should *not* be permitted through filters.†

Common network management tools such as "ping" and "traceroute" depend on being able to send and receive ICMP messages. Ping works by sending ICMP echo request messages, and listening for ICMP echo response messages. Traceroute works by generating UDP probe packets that are destined to a random UDP port, then listening for ICMP destination unreachable messages sent in response to the probe packet.

A.11. Other services

Other network services, such as databases, license servers, print servers, "rlogin" and "rsh" servers, and so forth, all use TCP or UDP ports. In general, if these servers are intended and required to run as "root", they use BSD privileged ports (ports below 1024), and if not, they use BSD unprivileged ports (ports at or above 1024), though this is not always true. If there's a particular service that's not discussed here that you're interested in special-casing, you can often figure out what ports it uses by examining the RFCs describing the service, the source code implementing the service, or (as a last resort) the output of "netstat -a" while the service is in use.

† ICMP redirect messages should never *need* to pass through a filtering router, anyway, since they are only supposed to be generated by the first router a packet reached after leaving its originating host; that router should be able send any necessary ICMP redirect back directly to the originating host, without having to send it through any other routers. An attempt to route an ICMP redirect message is a sign of either network misconfiguration, routing software bugs, or malicious activity by someone probing for weaknesses.