The PFILTER Firewall Compiler for Clusters

Neil Gorsuch[a]

[a]*NCSA, University of Illinois, Urbana, Illinois*

Linux clusters range from tiny to huge, but all sizes can benefit from effective firewalls. Stateful packet filtering firewalls can provide excellent security from network attacks, but are difficult at best to set up and maintain. When packet filtering is combined with packet forwarding, NATTING, and pseudo interfaces, a single machine can provide firewall protection for a private network of machines, while allowing the protected machines to have complete access to the general networks, and can allow the protected machines to be visible at general network addresses while maintaining firewall protection for them. This paper will discuss an open source, easily configurable packet filtering compiler system for clusters that can provide all these benefits.

Les grappes d'ordinateurs de toutes tailles ont avantage à comporter un coupe-feu. Les coupes-feu dynamiques offrent un excellent niveau de sécurité contre les attaques par réseau, mais il n'est pas facile de les configurer et d'en effectuer l'entretient par la suite. Lorsqu'on combine les techniques de filtrage et de réacheminement des paquets, de traduction d'adresses de réseau, ainsi que l'interfaçage réseau virtuel, il est possible d'utiliser un seul ordinateur coupe-feu pour protéger un réseau privé comportant plusieurs ordinateurs, lesquels conservent un accès complet au réseau externe. Cet article présente une application à code source libre facilitant la compilation de règles de filtrage de paquets pour la protection réseau d'une grappe d'ordinateurs.

1 Introduction

Packet filtering is the process of examining each network packet as it comes into or through a device, and either allowing, dropping or rejecting the packet based on various factors such as the source and destination addresses and port numbers, and whether the packet is the start of new connection attempt. Packet filtering can block all incoming network connection attempts except for the ones that are explicitly allowed. This prevents computer configuration mistakes allowing an insecure network service from being accessed by mistake.

Stateful packet filtering keeps track of each network communication sequence of packets, and provides for simpler, easier to setup and maintain, and more secure firewalls.

All cluster machines that can receive packets from outside the cluster should either have their own packet filtering installed, or a common firewall machine should filter all packets going in and out of the cluster

Using packet filtering to set up a firewall on a computer system is difficult at best, being similar to writing programs in assembly language. A method is needed for system administrators to setup and maintain good packet filtering firewalls without having to learn the intricacies of packet filtering commands.

2 Security Layers

To provide effective cluster security, multiple security layers should be provided. Cluster network security layers should ideally consist of: router filtering, network stack protections, IP masquerading and NATTING, packet filtering, disabling unused network services, TCPwrappers, and configuring applications.

2.1 Router Filtering

If possible, routers should be set up to block various types of unwanted packets, including: all source spoofed packets, all packets claiming to be coming from an RFC1918 “private” address that arrive from a public network, all packets coming into the cluster going to most privileged ports except for a few explicitly allowed ports, and all packets destined for various “problem” high ports such as X or NFS. Redirect, echo request, timestamp request, timestamp reply, address mask request, and address mask reply ICMP packets coming into the cluster should be dropped.

Some routers lack these filtering capabilities, while others are difficult to program. Because routers cannot typically track connections, outside attackers can sometimes craft special network packets to bypass router filtering.

2.2 Network Stack Protections

The Linux kernel has a number of attributes that help with network security which can be turned on or off through entries in the /proc directory tree. Some of the more useful features are: source spoofing prevention, disabling of ICMP redirect packets, and syncookie attack protection.

2.3 IP Masquerading and NATTING

If some of a cluster's nodes are only connected to a private cluster network, and it is desired for the nodes to be able to access outside network resources, a machine needs to forward network packets between the private cluster network and other networks. To maintain the privacy of nodes with no direct outside network connection, they should be assigned IP addresses that are not directly accessible outside the cluster. RFC 1918 reserves the IP ranges 10.x.x.x, 172.16-31.x.x, 192.168.x.x that cannot be routed across the Internet. To allow hidden cluster nodes to access outside network resources, the machine doing packet forwarding needs to provide IP masquerading. This alters all packets being forwarded from a hidden cluster node to an address outside the cluster to appear as if they came from the forwarding machine. Packets coming back in as part of a masqueraded connection are translated and forwarded to the appropriate hidden cluster node. NAT (Network Address Translation) is this process of modifying the addresses within the packets as they are being forwarded. Setting up IP masquerading requires special packet filtering rules.

2.4 Packet Filtering

Packet filtering is one of the most effective security layers, because it affects all network communication, intercepting network packets before they are passed up to application programs.

Linux kernels from version 2.4 on can track active network connections packet by packet. This is called stateful connection tracking. In addition to filtering which network connections are allowed to start and proceed, stateful packet filters can block all packets that are not part of a valid ongoing network connection. This prevents attackers from passing specially crafted packets into high network ports.

The main drawback of packet filtering is that it is difficult to configure. BSD derived operating systems use the netfilter command to control packet filtering, while Linux systems use the ipfw, ipfwadmin, ipchains, or iptables commands to control packet filtering. Without special tools, scripts consisting of many commands have to be hand-coded and maintained. It is difficult for non-experts to find adequate information to allow them to produce scripts that provide secure firewalls.

2.5 Disabling Unused Network Services

The most straightforward method of reducing network vulnerabilities is to disable incoming network services. Any unused or unnecessary incoming network services should be disabled. Most of them can be disabled using the linuxconf program. A few of them need to have their entries commented out in the /etc/inetd.conf file. Unfortunately, earlier versions of Linux and some network services in current Linux versions, do not follow this model. They can be inadvertently turned on and it is not always obvious that this has happened.

2.6 TCPwrappers

Newer versions of Linux support TCPwrappers, which adds another layer of application level filtering. This is accomplished through inetd with the files /etc/hosts.deny and /etc/hosts.allow controlling access for each network service. Unfortunately, some network services cannot be filtered this way.

2.7 Application Layer Filtering

Some server applications can be set to block access to certain addresses, or to allow access only to certain addresses. This feature should be used wherever possible. For example, the exporting of NFS shares can be to a limited list of machines and addresses, with or without write access.

3 Packet Re-Writing

Linux and other operating systems can modify network packets as they pass through a system. This is a more sophisticated type of NAT and can yield some very useful results. Since this is controlled through the packet filtering rules, a packet filtering compiler needs to control this.

3.1 Host Aliasing

Consider a firewall machine that has multiple network interfaces and that is acting as an IP masquerading firewall for one of the interfaces. In some cases it is advantageous to be able to access one or more of the machines that are on the private network through a public address. However if the access method desired for more than one of the machines is through the same type of service, a method is needed to have those machines appear as if they were also on the public network, with only outside connections to the desired services being passed on the private address of the machines. This allows one firewall machine to act as a gateway for a number of hidden machines behind it, with all of the firewalling and packet filtering protection only needing to be set up on one machine.

A classic example of this is a development laboratory. There might be a dynamic mix of machines or clusters that are constantly being rebuilt and that don't need the administrative overhead of setting up firewalls on each one every time they are rebuilt. But they all need to be accessible from the public networks through a security communication channel such as SSH. With host aliasing, this needs to only be set up once on the laboratory's firewall machine.

3.2 Packet Redirection or Forwarding

Sometimes packets need to be re-written and sent to a different destination than intended. One example of this would be having all outgoing packets that are destined for web servers being redirected to a proxy web server on a local machine.

4 Packet Filtering Details

Linux kernels use “chains” of packet filtering rules to filter packets. Each filtering rule has a set of matching conditions and an action to take if the packet is matched. Some of the conditions that packets can be matched by are source address, destination address, source port, destination port, TCP flags, type, MAC address, length, throttling limits, specific byte patterns, and the user id of locally generated packets. Some chains are pre-defined and always exist; others can be defined or deleted by the user. Rules can be added to any chain, and any chain can have all it’s rules flushed from it. Packets can traverse more than one chain.

Linux version 2.4 and later kernels support integrated connection tracking and stateful inspection. This adds more ways to match packets, based on whether they are new packets, packets that are part of an existing connection, or packets logically related to an existing connection. In addition greatly increasing security, this also vastly simplifies filtering rulesets.

A script that controls packet filtering is difficult to set up. The cluster administrator would like to specify something like “block every incoming connection attempt except for SSH to machines 1 and 2, and allow all cluster machines to access the outside network. To do this safely requires a script that is dozens of lines long with many commands, and setting up various network stack parameters by writing into the /proc directory tree. Packet filtering scripts are akin to an assembly language. For things to run correctly, “glue” code has to be added.

What system administrators need is a method to “compile” packet filtering firewall scripts that requires no knowledge of packet filtering configuration commands. Given this need, it was decided to implement a packet filtering compiler.

5 Firewall Compiler Requirements

The following requirements were determined for a packet filtering firewall compiler.

Firewall packet filtering rules should be specified in a high level language. The filtering language should be independent of iptables, ipchains, netfilter or whatever other packet filtering commands are actually used to implement the firewall. The compiler shall be designed so that any Unix variant that has commands to configure packet filtering can be supported.

The firewall compiler shall convert a text configuration file consisting of high level filtering language directives into a complete firewall implementation that includes all commands to modify appropriate kernel network stack parameters to provide added security.

A GUI shall be provided that modifies the same firewall text configuration file. An administrator may go back and forth between editing the firewall text configuration file and using the GUI, with changes done by one method being reflected in both.

The compiler defaults should allow no incoming network connection other than those that are specifically enabled.

The compiler defaults should allow all outgoing network connections.

On Linux, the compiler should be installed as a system service so that a firewall can be enabled or disabled with the /sbin/chkconfig command, and started, stopped, and restarted with the /sbin/service command. Every time the compiler is “started” or “restarted”, the firewall configuration file should be re-compiled. When the compiler is “stopped” it should turn off all network packet filtering, letting all packets be accepted. The /sbin/service command shall also support a “status” command, and a “chains” command to list the packet filtering chains currently in effect.

The compiler should be available as binary and source rpms, and as a source tarball that a “make” then “make install” will install correctly.

The compiler should be implemented using an interpretive language. The compiler itself should never need to be compiled and linked as a binary image. The compiler rpm shall be of the “noarch” type, and not tied to any particular hardware platform.

The firewall filtering language should be reasonably free form, and support comment lines, and comments at the end of lines.

The firewall filtering language should allow for sequences of network packet matching, with each successful match allowing the packet to be accepted, dropped, or rejected.

The firewall filtering language should support defined constants whose values cannot be changed once they are set, and symbolic variables that can be redefined and changed. Constant and variable values should consist of strings.

The firewall filtering language should support macros that can be expanded with parameter substitution.

The firewall filtering language should support conditional blocks.

Multiple network interfaces have to be supported per machine.

Each network interface shall be able to be marked separately as being attached to either a trusted network where no filtering is done for connection attempts from that network, or marked as being on an untrusted network where all connection attempts to the host machine shall pass through the filtering rules.

One or more network interfaces can be marked to have packets forwarded by using IP masquerading NAT. All machines attached to interfaces so marked shall have access to public networks through the firewall machine.

One or more machines attached to a network interface that is marked as being firewalled by IP masquerading NAT shall be able to be aliased onto an address on another network. For each aliased machine, a pseudo interface shall be created with it's own separate address. Each pseudo interface shall be able to have it's own filtering rules that are specific it's address, and connection attempts from the outside that are allowed will be forwarded/translated onto the real address of the aliased machine on the private network. Outgoing connections from aliased machines will be translated so that they appear to be originating from the machine's pseudo address.

The compiler shall support connection tracked flexible packet forwarding.

The "glue" code parts of the compiled output should be user configurable through text files.

Network services to be allowed and/or blocked should be able to be defined in user modifiable text files.

Network services should allow for more complex allowing and/or blocking, including embedded shell script fragments.

At present, all of the requirements except the GUI have been met. The GUI is in development.

6 PFILTER – a Firewall Compiler

The firewall compiler that was developed is called PFILTER (Packet FILTER). It was decided to implement PFILTER in the PERL language. There are better interpretive languages available, but none are so widespread as PERL. The main PFILTER program is installed as a PERL executable program at /usr/sbin/pilter. The remainder of the executable program is stored as included PERL files in the /usr/lib/pfilter directory. Included files were chosen instead of the more traditional perl modules to insure that PFILTER functions are placed in a more secure directory, and because the functions are very specific to PFILTER and generally not of use to other programs.

PFILTER is an open source project hosted on SourceForge, see http//sourceforge.net/projects/pfilter/

6.1 Ruleset Files

The PFILTER executable program is designed to do as little as possible. Instead, PFILTER uses built-in files called ruleset files to do most of the work of compiling o**u**tput scripts. The ruleset files are editable text files that can be modified or added to by system administrators. Heavy use is made of conditional text blocks and macros. This design was chosen for the following benefits:

PFILTER determines which packet filtering system is in use such as iptables or ipchains or netfilter. Setting a constant value to indicate which type of system is in use, conditional blocks and macros in the ruleset files are used to generate compiled code for all the different types of systems. This allows for easily adding complete new types of packet filtering systems by simply adding a few macros and some glue code text.

Almost all of the "glue" code that is put in the compiled output is specified in the ruleset files. This allows for a vast amount of flexibility, and support for many types of Unix variants.

New types of network services to be filtered on or off are defined in ruleset files. This allows system administrators to add new types of network services to be supported.

Because network services to be filtered on or off are defined by macros, they can embed shell script fragments in the compiled output code, allowing for filtering services such as NFS which dynamically allocate port numbers.

Because of the built-in constants, variables, conditions, and macros, the main firewall configuration file can be quite sophisticated, and will allow the same configuration file to be used throughout a cluster.

Compiled output scripts shall be optimized, with redundant or impossible combinations of packet filtering sources and destinations not being written to the output scripts.

The compiled output scripts shall include verbose comments explaining what each script line does. The configuration source lines that cause generation of output script lines shall be included as comments immediately above their generated output.

6.2 The PFILTER Firewall Language

The PFILTER firewall configuration is defined in the /etc/pfilter.conf file. Comments start with either the # or % characters and can be complete lines. Comments that start with the % character are not copied to the compiled output script, while comments starting with the # character are copied to the output script when appropriate. All directives and keywords are case-insensitive.

6.2.1 Constants and Variables

Named constants are defined like this:

%constant%   constant_name   strings ...

A constant's value will be set to everything after the name but not including any comments at the end of the line.

Trying to redefine a constant with a new value produces an error. Variables are defined like this:

```
%variable%      variable_name   strings ...
```

Variables can be re-defined any number of times. To substitute a constant or variable value, simply insert it with % characters on each side. For example, these lines:

```
%variable%      var     b       c
%constant%      const   d       e       f
a       %var%   %const%         g
```

will expand into this:

```
a       b       c       d       e       f       g
```

There are a number of constants that are defined by PFILTER during each compilation.

6.2.2 Macros

PFILTER supports macros with named parameters. When a macro is expanded, temporary variables that match the macro's parameter names are created just for that expansion block. Macros are heavily used when generating the compiled firewall output script. As an example, these lines:

```
%macro%     compute-node     management-node
# compute node compute-node
open     ssh      from      %domain%
open     tcp      1024:65536       from      %cluster%
open     tcp      0:1023   from   %management-node%
%endmacro

%compute-node%          node17          mgmt12
```

would expand to this:

```
# compute node node17
open     ssh      from      %domain%
open     tcp      1024:65535       from      %cluster%
open     tcp      0:1023   from      mgmt12
```

Macro definitions and invocations can be nested.

6.2.3 Conditional Blocks

Blocks of text or single lines can be conditionally included in the output based on any of the following types of conditional expressions:

%ifdef	*name*	*is true if constant/variable is defined*
%ifndef	*name*	*is true if constant/variable is undefined*
%if	*string = string*	*is true if strings match*
%if	*string != string*	*is true if strings do not match*
%if	*string*	*is true if the string is non-blank*

Conditionals can either surround a block of lines, or only affect one line. If the conditional expression is at the beginning of a line, the lines following that up until a line that starts with %endif are included in the output if the condition is true. If the end of line is a conditional expression, that line will be included in the output if the condition is true. The conditional expressions and any possible %endif lines are not included in the output.

6.2.4 Protocols, Ports, and Network Services

Some directives include lists of protocols, ports, or network services. These lists can include any of the following, separated by spaces or tab characters: a protocol and then one or more port numbers, port ranges, and network service names, network service name without a proceeding protocol.

Network service names are defined in four ways. First, a matrix of protocols, ports, and matching service names are parsed from the /etc/services system file if it exists. Second, some symbolic names are defined by the iptables and ipchains commands, but it is not advisable to use those, since newer versions of those commands might change or remove symbolic names. Third, network services can be defined in the either the main firewall configuration file or in one of the firewall ruleset files as a simple constant. For example, scattered in the ruleset files are these lines:

```
%define service-x-protocols-ports tcp udp/6000:6063
%define service-ping-protocols-ports icmp/8
%define service-ssh-protocols-ports tcp/22
```

This defines the network service *r* as responding to both TCP and UDP ports ranging from 6000 to 6063, defines the network service *ping* as responding to ICMP packets of type 8, and defines the network service *ssh* to respond to TCP port 22. A service name defined in this way over-rides any definitions in the /etc/services file. In the case of the *SSH* ruleset entry, the /etc/services lines that define the SSH service as being both TCP and UDP ports 22, are over-ridden, so that when the configuration file says:

```
open ssh
```

it will only open TCP port 22 and not open UDP port 22. The last way a network service can be defined is with a pair of PFILTER macro definitions. This allows for opening and closing of network services to involve shell script fragments, or to do anything else that isn't just a list of TCP and/or UDP ports or ICMP types. For example, one of the ruleset files includes these segments:

```
%macro service-multicast-open source destination
# Let all multicast packets through from %sources%.
# The destination is always 224.0.0.0-239.255.255.255.
# This method is used because multicast packets are
# identified by their destination address.
  %open_protocol_port% %source% 224.0.0.0/4 ANY
ANY
%endmacro

%macro service-multicast-close source destination
# Block all multicast packets from %sources%.
# The destination is always 224.0.0.0-239.255.255.255.
# This method is used because multicast packets are
# identified by their destination address.
 %close_protocol_port% %source% 224.0.0.0/4 ANY
ANY
```

%endmacro

This segment defines how to open or close multicast packets going through the firewall. If someone puts this line in their /etc/pfilter.conf configuration file:

open multicast from mydomain.com

then the compiled output script will include directives that will accept packets from mydomain.com going to anywhere in the address range 224.0.0.0/4.

6.2.5 Network Addresses

Network addresses can be specified as simple IP addresses, IP address ranges, DNS host names, or address ranges that include DNS host names.

6.2.6 Directives to Open/Close Access

To allow or block incoming network connections, these directives are used:

OPEN protocols-ports-services [from source(s)] [to destination(s)]
CLOSE protocols-ports-services [from source(s)] [to destination(s)]

If no source addresses are specified, incoming connections to the specified protocols-ports and/or services are allowed (or blocked) from any address. When no destination addresses are supplied, the connections are allowed or blocked going to the firewall machine. Destination addresses can be specified for other machines if the firewall machine is receiving and forwarding packets to the other machines.

6.2.7 Directives for Interface Attributes

The following directives specify attributes of network interfaces. They all take one or more network attribute directives, followed by one or more network interface names as.

FILTERED or UNTRUSTED – all network connection attempts coming from these interfaces are packet filtered to determine if the connections should be allowed.

UNFILTERED or TRUSTED – all network connection attempts coming from these interfaces are allowed, without any packet filtering.

PRIVATE or PROTECTED – interfaces marked this way are presumed to be on a private network that is being protected by the firewall machine. IP masquerading and NAT are applied to all outgoing connections coming from these networks.

PUBLIC or UNPROTECTED – interfaces marked this way are not being protected behind the firewall machine.

6.2.8 Directives to Setup Host Aliasing

To set up host aliasing, where a machine on a protected network is made to be partially accessible on a public network, this type of directive is used:

ALIAS pseudo real [ports-protocols-services] [from source[s]]

For example, if a machine named private1 on a protected network needed to have it's SSH server accessible from machines in mydomain.com, which has a 16-bit mask, appearing to be at public address public1, a line like this could be used:

ALIAS public1 private1 ssh from mydomain.com/16

Ports/protocols and/or services to be passed through do not have to be listed on the line that defined the ALIAS mapping, they can be listed elsewhere. The line above could also have been done like this:

ALIAS public1 private1
OPEN ssh from mydomain.com/16

6.2.9 Directives for Packet Forwarding.

To set up packet forwarding/re-writing a line like this need to be included:

*FORWARD protocols-ports-services [FROM source[s]] *
*[TO destination[s]] ONTO forwarded-desistination-address *
[forwarded-protocols-ports-services]

A list of protocols/ports and/or services, possibly matched by where they are coming from and/or going to, will be translated to instead go to the forwarded-destination-address, possible being translated to a different group of protocols/ports/services.

7 Conclusion

The first method of network security to be installed should be packet filtering. This will block access from outside the cluster to all but needed services while allowing the cluster to access the outside freely. With tools such as pfilter, it is straightforward to generate packet filtering firewalls with ipchains/iptables rule sets.