

Time-bound policy rules in netfilter

Fabian Schreiber Benedikt Over Sebastian Kraatz
Center of Informatics
University of Goettingen

April 18, 2006

Abstract

netfilter provides means to intercept and manipulate network packets. This paper describes an approach of extending its functionality to allow stateful packet filtering.

To achieve this goal, an additional module for the netfilter framework is developed. This module allows time-bound policy rules in netfilter by user command line input. A shared library enables interaction between user input from iptables and the kernel module, which in turn alters the netfilter chains. The developed extension works with Linux 2.6 kernel versions.

1 Introduction

1.1 Motivation

The initial goal of the project was the development of a programmable firewall to support stateless and stateful packet filtering. Stateless techniques filter packets according to a specific network protocol or port, while stateful filtering works dynamically after the session was set up.

In the course of the project, it turned out that netfilter already supports stateless packet filtering inside Linux kernels by means of rules that are processed whenever a data packet arrives. What was left to be done was to add stateful packet filtering support to netfilter by using time-bound rules. Adding functionality to netfilter is possible by kernel module extensions.

The addition of time-bound policy rules would allow the filtering of data packets not only by ports or protocols, but also for a specific time period.

1.2 Introducing netfilter and iptables

netfilter is a packet filtering framework inside the 2.4.x and 2.6.x Linux kernel series, containing a set of hooks to intercept and manipulate network packets. The software can be extended by kernel modules. It is licensed under the GNU General Public License (GPL) and was merged into the Linux 2.3 kernel in 2000.

iptables is the userspace command line program used to configure the packet filtering ruleset accessed by netfilter. It is a standard part of all modern Linux distributions. The name iptables is often used to refer to the entire package, including netfilter. Prior to iptables, its predecessors ipchains (Linux 2.2) and ipfwadm (Linux 2.0) were used to create Linux firewalls.

netfilter implements three chains, like shown in figure 1. All incoming packets match the **INPUT** chain, while outgoing packets use the **OUTPUT** chain. In addition, the **FORWARD** chain deals with packets that will be forwarded, not destined for the local system. Depending on the rule, the packet can be accepted (sending it to its specified destination), dropped (the packet is locally deleted) or rejected (the sender will be notified).

As the local administrator of the system, iptables is used to add, remove or alter rules within the chains to allow or deny specific connections.

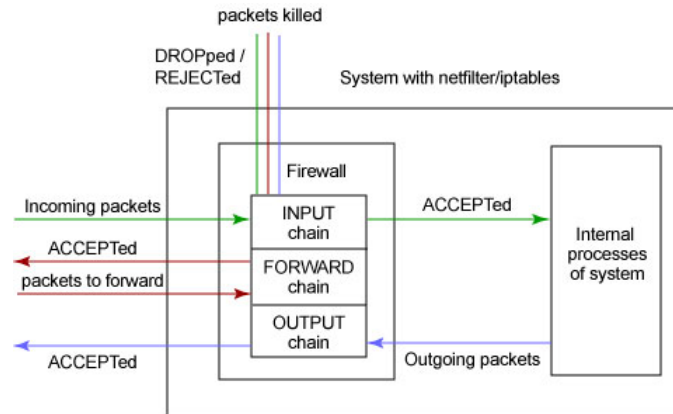


Figure 1: netfilter functionality [5]

1.3 Possible approaches towards a solution

The netfilter framework provides two approaches to extend its functionality. First, *libiptc*, an Application Programming Interface (*API*) can be used to allow kernel-deep communication with netfilter. During research, we came to the conclusion that this approach is not widely accepted, lacking robustness and missing documentation. As a more feasible method, a shared library can be developed that parses user command line inputs through iptables as rulesets. The rules provided by the shared library are stored as a structure which is forwarded to a kernel module that in turn evaluates the applicability of these rules whenever packets traverse the framework. The kernel module registers against the netfilter framework.

Stateful packet filtering could be accomplished by adding a timer value to the rules. This value defines if a rule is valid at a certain time.

2 Development

2.1 The goal

In the end, the user should be able to alter rules using the shared library and kernel module by utilizing the iptables application, invoking it with the additional parameters `-m timer` and `--validtime`. The developed shared library parses the arguments given by the user into an adequate data structure. A pointer to this structure is received by the kernel module which adds, refreshes or deletes the rule depending on the timer value.

2.2 The Shared Library

The shared library is used to interact with the user, handling the arguments the user wants the kernel-part to take into consideration. [2] Some common fields like the library name, iptables version and structure size need to be filled by the creator of the shared library. We used a skeleton for the library, extending it with the required fields.

Common functions of developed shared libraries include the following:

- When the module is loaded by iptables, the `init()` function is called. Within this function, each library must register to iptables by calling `register_match()`.
- `save()` saves the ruleset, dumping it to the kernel module
- `print()` prints out the match, invoked by `iptables -L`
- `help()` prints out available arguments to the command line
- `final_check()` executes a final sanity check after argument parsing is completed, right before exiting
- `parse()` parses and verifies the command options

While most of the functions are self-explanatory, the parsing of rules deserves further examination. It is the most important function, used for verification of the arguments, and setting the information shared with the kernel module. The function is called multiple times, according to the number of arguments passed from the command line. Each argument is therefore parsed according to its position in the user input stream. It is then stored in a linked list structure shared with the kernel module explained in this section.

When parsing the arguments, several checks are made, resulting in error reporting or specific flag settings. First, a parameter problem error will be reported when an argument is used more than once. The current source IP address is set in the shared structure, and the invert flag is set when the appropriate argument was passed.

Additionally to the shared library source, its header file contains definitions required for functionality and the structure shared with the kernel module. This structure keeps the information that is copied to the kernel-part after it is examined by the shared library.

2.3 The Kernel Module

Kernel modules are used by netfilter to access, check, add or delete rules. [4] The module inspects each packet received by the system, deciding whether to work with it or not. Common functions of kernel modules include the following:

- `match` is used to filter connections. If a packet arrives in the system, each table is processed, checking every rule. At some time, the developed module is able to process the packet. To be able to work with the incoming packet information, it is parsed into our own structure. Now, the function checks if the rule is valid, returning an appropriate value: A non-zero value indicates that the packet matches.
- `checkentry` is called when a rule is meant to be added, changed or deleted. Depending on the `validtime` value, the rule is either deleted (when `validtime=0`), or further examination is required: If the rule already exists, it is refreshed with the new time information. Otherwise, the rule is added to the chain. Apart from that, the rule is checked for reasonability.
- `destroy` deletes rules by first parsing the information into our structure, then searching for the passed entry within the linked list. When found, it gets deleted.

2.4 Structure of a Linked List

Our way of storing rules was to implement a linked list. Linked lists are the preferred way of storing data within the netfilter framework, as additional overhead in comparison to a hash table can be neglected. A linked list is a data structure consisting of a sequence of nodes, each containing data fields and references pointing to the next node, like shown in figure 2. [1]

We developed the structure `ipt_timer_info` that has the following fields:

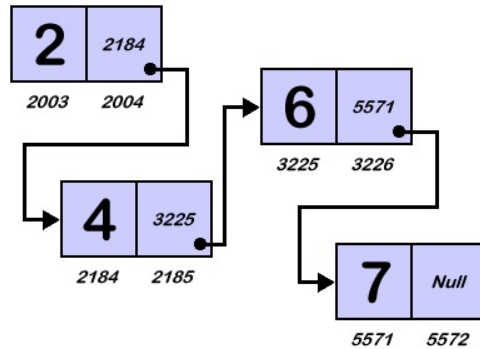


Figure 2: Structure of a linked list [6]

- `saddr` holds the source IP address
- `daddr` holds the destination IP address
- `port` holds the port number
- `protocol` sets the protocol
- `timestamp` sets the validity of the rule

Several functions are available to work with the list items:

- `Conn_create` allocates a new list item
- `Conn_free` frees the list item memory
- `Conn_freeall` frees the memory of all list items
- `Conn_print` outputs a list item
- `Conn_find` returns a specific list item by ID
- `Conn_delete` deletes a specific list item by ID
- `Conn_insert` inserts a list item at the end of the list
- `Conn_Id_find` returns a specific list item ID by source address, destination address and protocol

- `Conn_printall` prints the whole list
- `Conn_refresh` refreshes the `validtime` data of a specific list item
- `Conn_clear` refreshes the whole list, deleting invalid rules

3 Execution

3.1 Building the components

To be able to use the developed kernel module, the Linux kernel has to be rebuilt. Because of specific functions, the Kernel 2.6 series is required for proper operation. As a precondition, `ipt_timer.c` is copied into the `net/ipv4/netfilter` subdirectory of the kernel source directory. Additionally, `ipt_timer.h` needs to reside within the `include/linux/netfilter_ipv4` subdirectory.

Now, the network configuration file `net/ipv4/netfilter/Kconfig` was extended by

```
config IP_NF_MATCH_TIMER
    tristate "timer match support"
    depends on IP_NF_IPTABLES
    help
        To compile it as a module, choose M here.  If unsure, say N.
```

The netfilter makefile `net/ipv4/netfilter/Makefile` needs to know about our timer module:

```
obj-$(CONFIG_IP_NF_MATCH_TIMER) += ipt_timer.o
```

Now, the kernel can be configured the usual way, for example using `make menuconfig`. During configuration, the new kernel module can be selected for module inclusion. When using menu configuration, the appropriate entry can be found in the menu item

- Networking
- Networking options
- Network packet filtering (replaces ipchains)
- IP: Netfilter Configuration

as `timer match support`. netfilter packet filtering has to be selected as built-in. Finally, the new kernel can be built and the modules can be installed:

```
make
make modules_install
```

The created Linux kernel image is now copied into the boot directory, along with the `linux/System.map` file that contains information about the entry points of the functions now compiled into the kernel. [3]

The new kernel can now be started by first changing the boot loader, if applicable.

To make use of the shared library we created, iptables has to be recompiled. The `ipt_timer.ko` file which was created during compilation of the kernel has to be copied to the netfilter modules directory inside the library directory of the system. The shared library `libipt_timer.c` has to be copied into the appropriate build directory `iptables/extensions`.

Finally, the makefile has to be extended with the details of our shared library within the `PF_EXT_SLIB` entry list prior to the compilation of iptables.

3.2 Running the solution

The timer extension can be invoked by using additional parameters when running iptables as follows:

```
iptables -A [chain] -m timer --saddr [ip] --daddr [ip]
--port [port number] --protocol [protocol] --validtime [timestamp]
```

Whenever the parameter `--validtime 0` is set, the corresponding rule (if existant) will be deleted. In any other case, the rule will be refreshed (if it already exists) or added to the linked list.

If a data packet is received, the rules in the linked list are checked. If a rule is not valid anymore, it is deleted. On the other side, if the rule is valid, the data packet will be processed according to the defined rule.

Because of compatibility issues during testing phase of the project, intensive functionality tests are still due.

4 Conclusion

The timer extension developed in this paper provides means to use time-bound policy rules within netfilter.

Extensions to the netfilter framework base on two fundamental principles. First, a shared library running in user space is used to interact with the user, receiving and dealing with rules. These rules are stored in a data structure and passed to a kernel module, interacting with the netfilter chains.

Developing extensions to the netfilter/iptables framework means getting common with Linux kernel programming, the given structures of netfilter chains and the iptables interfaces. Additionally, data structures are developed to store rulesets.

Future development of the timer extension could aim at using a different data structure or extension of its functionality.

References

- [1] Wikipedia - *Linked Lists* - http://en.wikipedia.org/wiki/Linked_list
- [2] Nicolas Bouliane - *Writing your own netfilter match* - <http://www.tldp.org/linuxfocus/English/February2005/article367.shtml>
- [3] Peter Jay Salzman - *The System.map file* - <http://www.dirac.org/linux/system.map>
- [4] *Linux netfilter Hacking HOWTO: Information for Programmers* - <http://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO-4.html>
- [5] Mugdha Vairagade, IBM - *Introduction to netfilter/iptables* - <http://www-128.ibm.com/developerworks/security/library/s-netip>
- [6] Osman Balcı - *Ordered List: The Implementation View* - <http://courses.cs.vt.edu/csonline/DataStructures/Lessons/OrderedListImplementationView>