Scripting `iptables`

Andreas Rottmann

E-mail: a.rottmann@gmx.at

Abstract

This article describes a dsolution to problems inherent to traditional ***iptables*** *shell scripts. It describes a framework that transforms S-Expressions as found in Lisp-like languages into* ***iptables*** *firewall rules and allows for flexible maintainance of these rules.*

Keywords

Linux, firewall, packet filter, netfilter, iptables, Scheme, rule generator

1 Introduction

The "traditional" way to set up a Linux `netfilter` based packetfilter is writing a shell script that issues the `iptables` invocations necessary to set up the desired rules. A firewall setup that uses the shell script approach can be found in [Rottmann 2003]. This approach, besides being simple and straightforward, also has some drawbacks:

- The shell scripts are often redundant.
- There is no easy way to remove part of the rules currently in action, e.g. in response to an interface going down.

While the redundancy issue has caused a lot of `iptables` script generators to be created, none of them seem to allow partial dynamic reconfiguration. Such a facility is needed is when interfaces go up and down during operation, since most likely part of the rules refer to the interface or its IP address – those rules become obsolete and should be removed. The system presented in this paper, inspired by FIAIF [1], an `iptables` configurator script written in `bash`, allows the configuration of "zones" and dynamic removal of a zone and all rules associated with it.

[1] `http://www.fiaif.net`

2 Expressing packet filter rules in Scheme

The presented system is implemented in Scheme. The language was chosen since S-expressions provide a convinient syntax to express the packet filter rules. For example, the following rule will only allow TCP connections to the SSH, SMTP and HTTP ports, as well as UDP DNS queries, which come in on the `eth0` interface:

```
(accept :in "eth0" (tcp :dport (ssh smtp http)))
(accept :in "eth0" (udp :dport dns))
```

The first item in the list denotes the target of the rule when it is matched; `accept` means the packet is accepted. After the target there may be an optional list of keyword arguments (`:in "eth0"`). Any sublists are "extension lists", which start with a symbol denoting the extension (`tcp` and `udp` in the above example), followed by keyword arguments.

Rules, like in `netfilter`, are organized in chains. Additionally, "zones" are provided. A zone [2] defines a network to which the firewall is connected, and enables the administrator to setup different security policies for each zone. A zone can be "brought up" with a specific IP address and network interface. The rules in a zone, which are divided into input, output and forward sections, implicitly reference the interface the zone was brought up with and may reference the address of the zone.

The following example configuration file defines a few chains and a zone `internet`.

```
;; -*- Mode: scheme; fill-column: 60; -*-

;; log-drop: Simply log a package, the drop it
(chain 'log-drop
       '(((log level: debug prefix: "IPT packet died: ")
          (limit rate: "3/minute" burst: 3))
         (drop)))

;; wintraffic-drop: Drop all Windows noise
(chain 'wintraffic-drop
       (let ((winports '(137 138 139 445)))
         `((log-drop (tcp dport: ,winports))
           (log-drop (udp dport: ,winports))
           (log-drop (tcp sport: ,winports))
           (log-drop (udp sport: ,winports)))))

;; allowed: For TCP connections to allowed ports. Allow all
```

[2] The "zone" concept is taken from FAIF

```
;; connection requests and packets belonging to
;; already-established connection, drop everything else.
(chain 'allowed
       '((accept (tcp syn: set))
         (accept (tcp) (state: established related))
         (log-drop (tcp))))

;; The internet zone. We accept connections on the ports
;; used by eMule/xMule ;-) and the HTTP port, as well as
;; packets from already established connections to our
;; IP. We get rid of all Windows traffic that would leave
;; the box.
(zone 'internet
      '((input
         (allowed dst: zone-address (tcp dport: (http 4662)))
         (allowed dst: zone-address (udp dport: 4672))
         (accept dst: zone-address
                 (state: established related)))
        (output
         (wintraffic-drop)
         (accept))
        ))
```

3 The `fidfw` command-line tool

With a suitable configuration file, further operation is simple. Zones may be brought up and down with the obvious commands:

```
fidfw up internet ppp0 80.109.53.157
fidfw down internet
```

`fidfw` will issue the commands needed to bring the zone up and down, including creating the chains referenced from the zone.

References

[Rottmann 2003] Firewalls – An Introduction to Concepts and Implementation using Linux `netfilter`;
`http://yi.org/rotty/fw-intro`