A Report on the IPtables Firewall

Date:
2019-01-29
-
2019-03-25

Anton Wiklund
19880905

Introduction to Computer Security
Kristianstad University

Teacher:
Qinghua Wang

Table of Contents:

1. Introduction.

Iptables is a commandline firewall which provides the user with the ability to configure the Netfilter capabilities(Netfilter* controls access to the network stack on Kernel level), thus creating a firewall. Or in the words of Rash, Michael: ”You can think of Netfilter as providing the framework on which iptables builds firewall functionality.”[p. 30]. The Iptables firewall, found in various UNIX-systems, is a stateful and kernel-level firewall. There does exist graphical interfaces for the Iptables firewall, but these will not be discussed in this report, and the general opinion seems to be, that they are often severly limited in the funcionality which they provide the user with.

In this report, there will be provided a very basic explanation regarding how Iptables can be implemented. The focus of the report will therefore be on providing a general understanding regarding how the Netfilter firewall in theory functions, and how theoretical understanding of it also can be practically implemented, in order to deploy a satisfactory firewall. An additional component of the report will concern Nftables, which is a reworked and updated version of the Netfilter firewall. Nftables seems to be the future standard, when it comes to Linux firewalling. The Nftables has received a separate section in the end of the report, where differences and similiarities between it and Netfilter are discussed.

The general information for the report has been gathered from a large array of internet-sources, as well as from various books(Mainly the Michael Rash one), while also very primitive prior knowledge has been incorporated.

*Since netfilter and iptables are so intertwined, I will in general use Iptables as a synonym for Netfilter, in order to not complicate things. One should however remember the fundamental difference between the two.

1.1 – Tables and Chains.

In order for Iptables to be able to filter network-traffic, packets pass through ’chains’. There are five predefined chains(custom-tailored chains can be created), which map to the netfilter hooks.
# OUT – for outgoing traffic.
# IN – for incoming traffic.
# FORWARD – for traffic which should be forwarded.
# Pre-routing – processes packet when it arrives to the network interface.
# Post-routing – processes packet when it is leaving the network interface.

The chains themselves belong to ’tables’, of which there are four.
# Raw – Allows for handling packets prior to conntracking of them begins – independent processing.
# NAT – Network Adress Translation(packet routing).
# Filter – The default table in which rules are placed. The main filtering table itself.

# Mangle – Allows for alteration of packet headers.

1.2 – Rules(policies).

Each of the chains can be provided with rules(policies), determining how certain kinds of network traffic should be handled. Each chain also has a default policy, even if not one has been explicitly stated by the user. A rules contains a specification for the packets which it should process. The specification is created by adding ’matches’ together. A match is a quality which the packet should adhere to, in order to be processed by the rule(for example having a certain destination port). A rule also has a target or verdict at the end, determining what should be done with the packet. There are four primary targets.
# DROP – (terminating) – Disallow the packet, but don’t send a notification back to the source.
# REJECT – (terminating) – disallow the packet, and send a notification back to the source.
# ACCEPT – (terminating) – let the packet pass.
# LOG – (non-terminating) – log the packet header to syslog, and then allow the packet to continue further down the chain.

1.3 – Stateful tracking.

Since the Iptables firewall is a stateful firewall, it processes packets in accordance to the state in which they arrive, to determine whether they are, for example:
# Part of a new connection.
# Are new but related to a previous connection.
# Part of an already established connection.
# If they’ve been targeted in the ’raw’-table to bypass tracking.

The reason for the stateful tracking, is to enable functionality which the administrator can use in order to pinpoint the connection during the various stages through which it moves. One kind of processing might be desired when the connection is first initiated, through a packet with the ’new’-state. Another kind of processing might be desired when the connection has been established.

2. Fundamental concepts for creating Iptables rules.

When setting up a chain, there are several basic things to keep in mind. There are, beyond the below stated, most likely many more relevant basic concepts to consider.
# One must adhere to the principle of sequential processing: The first rule which the packet fully matches, is the rule which will give the packet a target. Therefore, more specific rules should be placed further up, while more generic rules should be placed further down.
# If the packet reaches the end of the chain, this will result in default policy becoming the target. Default policy can be set.

# Iptables only filter IP-traffic and overlying protocols. Extensions can provide data link processing. Even if Iptables can handle application-layer processes, this is usually more reasonaly left to the application itself. An unwanted user is rather removed from the application, than blocked from contact with the network. Of course: both responses can, and perhaps should, be employed.
# A basic principle is to not respond to any traffic which is not desired. It is, generally, better to just drop the packet silently.
# IPv4 is the default for Iptables. To also enable IPv6 firewalling, the rules must be added to the chains which handle IPv6 traffic. The syntax for rules-creation is similar.

3. How to write, and implement, Iptables rules.

3.1 – Basic syntax.

The Iptables syntax is at first a bit complicated to get into, but after a while it gets a lot easier. It's pretty easy to write basic rules, after one has understood the general principles. Below we will create a rule for outgoing HTTPS traffic to be allowed through.
First one states which chain that the rule should be appended to:

```
$ iptables -A OUTPUT
```

One can also decide to which table's chain, that the rule should be added('filter' is default):

```
$ iptables -t filter -A OUTPUT
```

It also necessary to add the relevant 'matches', determining the protocol and port which the rule will handle.

```
$ iptables -t filter -A OUTPUT -p tcp --dport 443
```

Having done so, it is necessary to set the 'target' for the rule.

```
$ iptables -t filter -A OUTPUT -p tcp --dport 443 -j ACCEPT
```

We now have a rule for allowing all tcp traffic on port 443 to pass the firewall.

However, since Iptables is a stateful firewall, we must make sure that the tcp connection is kept open, and not immediately dropped or rejected. To do so, we have to create a 'conntrack' rule.

```
$ iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
```

This rule states that incoming traffic(INPUT), should be handled by the 'conntrack' module(-m), and if the connection state(--ctstate) is established or connection-related, then it should be accepted to pass the firewall.

Just because a packet is related to a current connection, does however not imply that the packet is desired. We therefore also write a rule for dropping, and logging, invalid packets.

```
$ sudo iptables -A INPUT -m state --state INVALID -j LOG --log-prefix "DROP INVALID " --log-tcp-options --log-ip-options
$ sudo iptables -A INPUT -m state --state INVALID -j DROP
```

In the first rule, the module ’state’ checks if the state of the packet is, for some reason, invalid. If it is, the packet is logged to syslog. The –log-tcp/ip-options state which protocol headers that should be logged to syslog, for analysis.
In the second rule, the packet is dropped. If it is only logged, it will continue being processed.

Having understood some of the syntax, used when writing an Iptables rule, we will now move on the main object of the report. Namely creating a, for the requirements which will be stated in the ’User’-section below, complete and functional Iptables configuration.

4. Writing An Iptables configuration for the average user

4.1 – The user.

The usercase for which an Iptables configuration will be written, is a general homeuser-network. Basic things, such as browsing the web, sending email, listening to music via Spotify, streaming movies via Netflix, should be allowed, while all other functionality, by default, should be blocked. The firewall will also be implemented on a separate system, and not on any single user’s system. The firewall will process all the data which arrives to, and leaves, the network itself.

4.2 – Conceptual outline of the configuration.

The firewall will implement the following functionality:

# Spoofed packets will be dropped – for example hinders TCP-sequence-guessing(Rash, p. 42).
# SSH, FTP, HTTP, HTTPS, DNS will be allowed to open connections out.
# All incoming connections – besides SSH – will be dropped.
# Default policy will be drop, but all packets reaching end of chain will also be logged.
# Limit for number of active connections – ddos protection – will be implemented(Rash, p. 66).
# Invalid packets will be dropped.
# Rules will be saved inbetween reboots.
# Loopback access will be allowed.

4.3 – Creation, and implementation, of the rules.

Below is an outline of the concrete rules which upholds the firewall requirements. I have not used the multiports module, since more specific rules also invites more specific control over the firewall. I add the rules like this, since it makes more sense than adding a screenshot, since then the rules can also be copied to see if they work.

```
# Flush all iptables filter rules first - Flush does not affect default policy
sudo iptables -P INPUT DROP
sudo iptables -P OUTPUT DROP
sudo iptables -P FORWARD DROP
sudo iptables -F INPUT
sudo iptables -F OUTPUT
sudo iptables -F FORWARD

# Allow LoopBack communication
sudo iptables -A INPUT -i lo -j ACCEPT
sudo iptables -A OUTPUT -o lo -j ACCEPT

# Keep established connections open
sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
sudo iptables -A OUTPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

# Drop and log invalid packets
sudo iptables -A INPUT -m conntrack --ctstate INVALID -j DROP
sudo iptables -A OUTPUT -m state --state INVALID -j LOG --log-prefix "DROP INVALID " --log-tcp-options --log-ip-options
sudo iptables -A OUTPUT -m state --state INVALID -j DROP
sudo iptables -A INPUT -m state --state INVALID -j LOG --log-prefix "DROP INVALID " --log-tcp-options --log-ip-options
sudo iptables -A INPUT -m state --state INVALID -j DROP

# Spoofed packets (set desired interface and ip-range)
# sudo iptables -A INPUT -i eth1 -s ! 192.168.10.0/24 -j LOG --log-prefix "SPOOFED PKT "
# sudo iptables -A INPUT -i eth1 -s ! 192.168.10.0/24 -j DROP

# Allow out TCP
sudo iptables -A OUTPUT -p tcp --dport 80 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
sudo iptables -A OUTPUT -p tcp --dport 53 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
sudo iptables -A OUTPUT -p tcp --dport 443 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
sudo iptables -A OUTPUT -p tcp --dport 22 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
sudo iptables -A OUTPUT -p tcp --dport 20 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT

# Allow out UDP (I don’t know why, but the rules doesn’t work without allowing udp.)
sudo iptables -A OUTPUT -p udp --dport 80 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
sudo iptables -A OUTPUT -p udp --dport 53 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
sudo iptables -A OUTPUT -p udp --dport 443 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
sudo iptables -A OUTPUT -p udp --dport 20 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
sudo iptables -A OUTPUT -p udp --dport 22 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT

# Allow in SSH
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT

# Endrule: LOG prior to default DROP
sudo iptables -A INPUT -j LOG --log-tcp-options --log-ip-options
sudo iptables -A FORWARD -j LOG --log-tcp-options --log-ip-options
sudo iptables -A OUTPUT -j LOG --log-tcp-options --log-ip-options
```

4.4 – Verification of the rules.

The rules allows the user to go about everyday business on the internet, while also blocking incoming connections. I verified this by general browsing and using Nmap. Verification of the rules will also be controlled through daily use. I've written the configuration so that it will be viable for my own use, and will therefore use it daily, to see if it functions as is intended. Since I've finished this assignment some time ahead, I will have a lot of time to test the configuration, before providing a final opinion on the correctness of the configuration. This final opinion on the correctness, will be given in the 'conclusion'-section.

4.5 – Managing Iptables rules.

Iptables rules can be managed in several ways. Rules can be added in real-time, they can of course also be deleted. If only appending a rule, it will end up at the bottom of the specified chain, and when deleting a rule, the upmost one will be deleted. But when adding or deleteing rules, it is also possible to determine which line they should be added to, or deleted from. To do so it is first necessary to figure out which line one should specify. In order to get this information, one can call the following command:
”iptables -L –line-numbers”
Afterwards, if one wishes to add a rule at line 2, one simply writes:
”iptables -I OUTPUT 2 -j ACCEPT” (the -A is changed to -I when specifying line).
And if one wishes to delete a rule at line 7:
”iptables -D OUTPUT 7”

One of the most relevant aspects of managing Ipables, is saving the rules which have been implemented. If one does not do so, the rules will have been reset during the next boot. One program which however enables control over rules is 'netfilter-persistent', which provides the user with the ability to: flush; start; stop; reload; restart; save.
To save the rules, one simply writes:
”netfilter-persistent save”.
This will then save the rules, so that they are reloaded during the next reboot.

5. Tools to complement Iptables with.

# Snort – A program which deals in signature-based detection of unwanted network-traffic. The protection is achieved through the pre-processing of network-traffic. Snort can for example assemble TCP-fragments, in order to see what they really contain. When a Snort-rule is broken, a warning is sent to the user, alerting it to undesired or suspicious network-behaviour of a packet. Iptables is a more silent and general method of protection, while Snort goes in-depth. They therefore complement each other in a very good way: ”Linking iptables rules to Snort signatures is the key to getting true intrusion detection capabilities from iptables.”(p. 94). That Snort goes much more in-depth when analysing packets, of course also means that Snort demands a much larger degree of hardware-resources. Writing Snort-rules also seems to be much more complicated.

# Fwlogwatch – A tool which can analyze various kinds of logs. It can for example analyze logs generated by Iptables, and alert the user to details of potential interest.

# Nmap – Nmap is a tool for penetration-testing of, among other things, firewall configurations. For example provding info on which ports that are open/closed/filtered.

# Wireshark – A tool which the user can employ to achieve extensive real-time packet analysis. Has a GUI in which headers, source adresses, ports, protocol-type etc is observable.

# Tripwire – How can one know, 100% for sure, that the system has not been compromised, and that various actions has not been taken, in order to for example hide certain network traffic? To change Iptables rules when so desired? It is impossible. For this reason, Tripwire allows the user to build an encrypted database – in which checksums are stored – covering all the desired files/folders of the system. It is then possible to run checks, to see whether files have been removed or manipulated, when they have been manipulated, in which ways, etc. Also Tripwire could of course be compromised, but it is, at least, an additional layer of defense. Tripwire can also run automated and scheduled checks, and send the results to an external, more locked down, location.

6. Basic overview of Nftables.

Nftables was release in 2014, in order to start replacing Iptables. It is possible to translate Iptables rules into Nftables rules, using iptables-translate. It is therefore, even if the syntax is quite different in Nftables, easy to move over to Nftables.

Nftables provides several new features, among them the ability to construct maps. This allows the user to arrange rulesets in multidimensional trees, which is something that reduces the number of rules which are necessary to pass until a relevant ruled is arrived at.

There are in Ntftable however no predefined chains. This makes it a bit more complicated to set up, but if at the same time provides the user with the incentive to create chains customized and optimized for the intended configuration. In Iptables it is also not possible to delete predefined chains. This leads to unnecessary use of system resources. Something which is avoided in Nftables. Only the desired chains are present.

It is also possible to use various expression-symbols. If the user for example wishes that a rule should handle all ports above 100, then a match is added which states ”> 100”.

7. Conclusion.

The Iptables firewall is a highly configurable firewall. It is however quite complex to use. At least if one compares it with the more standard GUI firewalls available to users. Even so, having taken the time to learn the basic concepts of the Iptables firewall has given me valuable insight into how firewalls actually functions, and being able to write an, even if very simple, configuration, feels like a very good thing.

Prior to the report, I had somewhat of an understanding regarding Iptables, but during the process of writing this report, I have, too a much larger degree, come to understand how rules interact and should be implemented.

During the months in which I have used the configuration, it has functioned ok. It is of course hard to know exactly how well it has functioned, but it has allowed me to surf the web and take care of my daily business without any specific problems occuring in relation to the firewall rules.

8. Bibliography:

1. https://en.wikipedia.org/wiki/Iptables

2. Michael Rash - Linux Firewalls - Attack Detection and Response with iptables

3. https://www.booleanworld.com/depth-guide-iptables-linux-firewall/

4. https://www.frozentux.net/iptables-tutorial/iptables-tutorial.html

5. https://www.digitalocean.com/community/tutorials/a-deep-dive-into-iptables-and-netfilter-architecture

6. https://wiki.nftables.org/wiki-nftables/index.php/Main_Page