Lab#7: Security with iptables

As it was explained in class, TCP/IP protocol stack is part of most current operating systems such as GNU/Linux, OS X or Windows. It was not always so, as such was not part of MS-DOS or CP/M.

Since the code of the operating system handles requests for communication applications is quite natural to raise the possibility of being able to adjust its behaviour to specify which policies are most appropriate at the discretion of the administrator. Different operating systems include different ways of specifying these preferences. In the case of GNU/Linux iptables is the tool that allows the administrator to view and modify policies for network traffic management in a computer system.

Iptables uses the concept of separate rule tables for different packet processing functionality.

Non default tables are specified by a command-line option. Three tables are available:

- `filter`

  —The `filter` table is the default table. It contains the actual firewall filtering rules.

  The built-in chains include these:

  - `INPUT`
  - `OUTPUT`
  - `FORWARD`
- `nat`

  —The `nat` table contains the rules for Source and Destination Address and Port Translation. These rules are functionally distinct from the firewall filter rules. The built-in chains include these:

  - `PREROUTING` (DNAT/REDIRECT)

- `OUTPUT` (DNAT/REDIRECT)
- `POSTROUTING` (SNAT/MASQUERADE)

- `mangle`

  —The `mangle` table contains rules for setting specialized packet-routing flags. These flags are then inspected later by rules in the `filter` table. The built-in chains include these:

  - `PREROUTING` (routed packets)
  - `INPUT` (packets arriving at the firewall but after the `PREROUTING` chain)
  - `FORWARD` (changes packets being routed through the firewall)
  - `POSTROUTING` (changes packets just before they leave the firewall, after the `OUTPUT` chain)
  - `OUTPUT` (locally generated packets)

In this lab we will see some of the basic uses of this powerful tool. A detailed knowledge of iptables requires much longer than the practice time.

Essentially this tool allows you to modify the behaviour of the operating system for various types of traffic. Since communications management is at the core of the operating system, only administrator can change the connectivity of the system, making it completely invisible to other machines on the network or on the contrary he is want. For this reason and because we do not have administrator access on the computers in the lab we use the sudo command before iptables.

The first operation is to verify the status of our computer. To do this type `sudo iptables -L` and see the result.

The result should be like this:

```
$ sudo iptables -L

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

This situation shows that for each of the three chains (INPUT, FORWARD and OUTPUT) policy is to accept any type of traffic (policy ACCEPT) no matter their origin or destination (source, destination) or the protocol used. This configuration, which could be called completely open, no restrictions on traffic input (INPUT) or output (OUTPUT) nor on the possible traffic "pass" if our computer is acting as a router (FORWARD). The latter option is not enabled on laboratory equipment, but all have two network connections and could eventually work as routers.

Exercise 1

1. start changing the configuration to verify its effects. To do this we use the -P option (policy) by typing the following command:
   `# sudo iptables -P OUTPUT DROP`
2. then execute the command `ping localhost` and see what happens.

This command modifies the default policy for all outgoing traffic, including one that does not leave the system (localhost) so that the system has become a kind of black hole in the network that no packet can leave. A similar effect (although not as dramatic) may occur if you disconnect the network cable (in this case the above ping continue operating correctly).

Exercise 2

1. This first experience did not seem too useful because it produces a total disconnect, we will return the computer to its original state. Type now: `sudo iptables -P OUTPUT ACCEPT`
2. Check that new network services become operational. Run the command `ping localhost` and see what happens now.

In the current situation, which is the same as at the beginning, all traffic can flow without restriction. Sure you know the term "firewall" in a context of computer networking. These devices (traditionally hardware) that allows you to filter certain types of traffic in order to protect some computers and networks of external threats.

Many companies put a firewall between their LAN and Internet connection. This device includes some rules that filter out certain packages to improve the security of the internal network.

This practice can cause your computer to reject, on a selective basis, certain types of traffic. To do this we need a little more fine rules that used in the exercise 1.

Default policy

It has been mentioned previously that using `iptables -P ...` you can change the policy of accepting (ACCEPT) or discarded (DROP) certain packages. But this generic behaviour affects all kinds of traffic. It can be refined by adding new rules that modify the default behaviour. For example: you can choose to accept all traffic initially, but after adding a new rule that prevents certain types of traffic, such as an FTP connection.

Instead of dropping a packet with DROP it is possible to make a REJECT, that sends an ICMP datagram "port unreachable" (this is the default action). Employing REJECT instead DROP prevents access to ports in a more polite way, but also allows an attacker to check which ports are open in our system.

With iptables, the administrator defines a default policy for incoming or outgoing traffic and then an additional set of rules that enable or block specific network traffic. But in this process it is essential to define the most convenient default policy.
If we want a system as restricted as possible, then it should discard any traffic except that which is expressly authorized. In this case we cannot start as in exercise 1, preventing any outbound traffic.

However, it is also possible that what we want is just to prevent certain types of traffic that is "annoying" without affecting other services. For example, maybe we want to avoid that users print to some printer from a specific computer remotely. In this case, we should define as default policy to accept all kinds of traffic, and after that, we will introduce a rule that specifically blocks the traffic that is directed to the network printer.

Throughout this lab, we will use a default policy based on accepting all traffic that is not explicitly rejected by other rules.

List of Rules

As iptables commands are running, we are adding or removing rules associated with each of the input or output flows. We can add a rule at the end of the chain with the parameter **`-A` (Append)**, at the beginning with the parameter **`-I` (Insert)** and delete it with the parameter **`-D` (Delete)**. When we add a rule we have to specify to which chain (INPUT, OUTPUT, or FORWARD) it should be applied. Also, **we can delete all the**

rules of a chain with the parameter –F (flush).
It is important to understand that the rules are added to be processed sequentially until a rule matches with the properties of the analysed packet. Once it happens, the following rules of the chain are not processed, and the matching rule will be applied to the packet. However, if none of the rules match, the default policy will be applied. This means that to undo the effect of a rule, the solution is to eliminate the rule, instead of trying to add another to contradict the first (i.e. it is not worth adding an ACCEPT after a DROP because the second one does not override the first).

Exercise 3

1. Let's block local traffic (interface `lo`) typing the following command: `sudo iptables -A INPUT -i lo -j DROP`
2. Next check if you have the desired effect by typing `ping localhost`. Do you get an answer? Try `ping www.upv.es` does it work?
3. To restore the local traffic, simply delete the above rule, type: `sudo iptables -D INPUT -i lo -j DROP` (remember that -D parameter means delete the rule while the -A means add)

In this exercise we use the `-i` parameter to specify a network device (you can run `ifconfig` to see the list of network devices and their current configuration) and `-j` specifies what to do with traffic that matches this rule.

The device "lo" is not really a network card but the representation of internal communications through the loopback address 127.0.0.1

In the example we have determined that all traffic that is received (INPUT) by "lo" device must be discarded (DROP).

Some rules will require not only the device but the port numbers or addresses involved to be really useful.

Exercise 4

1. Check if you can access via SSH to zoltar.redes.upv.es and establish a connection using the username and password provided by your teacher.
2. Open another terminal on your computer, and then type: `sudo iptables -A INPUT -p tcp --sport 22 -j DROP`
3. Now return to the window with your SSH connection to zoltar and type 123456. What happens and why?
4. Back to the second terminal, now type `sudo iptables -D INPUT -p tcp --sport 22 -j DROP` What now? Still works with your ssh session zoltar?

In this example we have created a rule that does not specify the network device but protocol (`-p tcp`) and the source port of the segments (`-- sport 22`). In an SSH connection packets coming from zoltar has as source port 22, which is the port where SSH service is offered.

Similarly it is possible to apply the same strategy to block access to any other service. However, the rules can be applied to either outgoing and incoming traffic, or both. In the previous exercise incoming SSH traffic (INPUT) was blocked. If you've taken more than a minute between step 2 and step 4 of the exercise the SSH connection may have been interrupted. In that case, try it again faster (which allows you to keep the connection and to get a different result).

You can also create rules that respond to the packet addresses.

Exercise 5

1. Open a browser and load the page www.upv.es
2. Now in a terminal window type `sudo iptables -A OUTPUT -p tcp -d www.upv.es --dport 80 -j DROP`
3. Try to refresh the page in the browser. What happens?
4. Try to visit another site, such as `www.ua.es` , does it work?
5. Type `sudo iptables-L` to display the current rule list.

As you can see, in this exercise we have created a rule that drops the outgoing TCP traffic destined to the www.upv.es server and port 80 (HTTP). This rule does not affect similar traffic sent to any other server. This rule only prevents access to the main Web server of the UPV. You can create a set of similar rules in order to prevent access to a list of different web servers. Remember that unless you delete this rule, its effect will last throughout the lab session.

It can be also convenient the use of negation operator `!` to modify the meaning of parameters as: protocol (`-p`), origin (`-s`) or destination (`-d`), destination port (`--dport`) or origin port (`--sport`). For example, `sudo iptables –A OUTPUT ! –d localhost` creates a rule that will be applied to any outgoing packet does not addressed to the localhost.

Exercise 6

1. Create a rule that accepts the outgoing TCP traffic destined to the www.upv.es server and port 80 (HTTP) and drops other outgoing traffic destined to port 80. When you specify a port you should specify the protocol with the parameter `–p` . In the case of HTTP `–p tcp`.
2. Open the browser and try to load www.upv.es and www.redes.upv.es pages, what happen? You should not be able to access the second server.
3. Add another rule to accept only the outgoing traffic of ssh protocol destined to zoltar.
4. Open a terminal window, and try to access ssh server in zoltar, does it work? Can you access ssh server in rdcXX.redes.upv.es?

Event Log

The functionality of iptables rules to specify not only to discard packets (as seen in several examples). With a default policy of discarding packets (DROP) the rules should be to accept certain types of traffic. But besides these actions can produce the rules for registration must be recorded in the system event log (in the / var / log / messages on the computers in the lab).

To create rules that generate an entry in the log when the rule matches a packet, use the option `-j LOG`. These rules do not determine if the packet is accepted or rejected, because the appropriate action will be applied as if this record rule does not exist.

The interest to record certain events associated with network traffic depends on the situations under consideration. They can provide information to the administrator of multitude of data that may be registered in other places or not. For example, if you want to know how many people connect each day to our FTP server is very possible that the server program has a detailed log file containing this information, but if it is a very basic server could not generate any kind of information registration. Let us assume that we are in this second case and we are asked to determine how many clients connect to the server every day.

First we need to determine what condition we consider as valid to establish that it has reached a new customer. The simplest (although not necessarily more accurate) is that each new connection to port 21 on our server is a new customer.

Exercise 7

1. Create a rule that records: `sudo iptables -A INPUT -p tcp --dport 21 –j LOG`
2. Now access the FTP server locally. Type ftp localhost or direct the browser to the url ftp://localhost , What happens?
3. You may already have noticed that there is not an FTP server on your computer. It does not matter. Then type `dmesg` and analyze the last line of the listing. What do you see? Do you know what that means?

With the rule of exercise 7.1 every packet that arrives at your computer and goes to the FTP server (even if you do not have FTP server) is the recorded. However there is a serious problem that is not apparent now, because you do not have local FTP server. The problem is that this rule is complied with registration for each segment received by each FTP connection. This means that a client can generate thousands of entries in a single connection. Fill out a log file with many significant data bit is a very bad idea.

We only need to register once to each client. One way to do this is to consider only the segment of the beginning of the connection, as we know, the SYN flag is active.

Exercise 8

1. Let's delete the previous rule: `sudo iptables -D INPUT -p tcp –dport 21 -j LOG`
2. Next let's create a rule that actually records the connection requests to port 21. To do so type: `sudo iptables -A INPUT -p tcp --dport 21 --tcp-flags ALL SYN -j LOG`
3. And now try to connect to FTP server (that we do not have). With the command `dmesg` we will see a similar output to the previous exercise.

It may seem that we have repeated the same as the exercise 7 but it is not. Now the rule pays attention to the record field of "flags" of the TCP header, we analyse all the bits of the header (which is why the value ALL) and recorded all segments having received the SYN bit on.

So we can see that this idea really works we connect to a server if you have an FTP server, as is the computer zoltar.redes.upv.es and ask to `record` each connection we make to it in the event log.

Exercise 9

1. Let's add a new rule to log connections to the FTP server zoltar.redes.upv.es, type: `sudo iptables -A OUTPUT -p tcp --dport 21 --tcp-flags ALL SYN -j LOG` (notice that we have changed to OUTPUT but we continue analysing the `destination` port). By not specifying the destination server this rule shall register all connections made to any FTP server from this computer.
2. Now from a terminal window (or from the browser) access the server via FTP zoltar.redes.upv.es with `anonymous` user.
3. Check using `dmesg` that access has been recorded as a single line in the log.
4. Check using `sudo iptables -L -v` the list of rules that are active at this time in your system. Note that the pkts column tells you how many times each rule has been met. You can also see the number of packets and bytes received (INPUT) and sent out (output) and, if appropriated, forwarded (FORWARD).

But the functionality of iptables goes on. You can also change any of the

information contained in the packets or their headers.

Changing directions and/or destination ports

Although for reasons of space we avoid giving details of all options of iptables, but at least we want to illustrate with some examples of features. In the following exercise we will create a rule for all affected TCP segments to change their address.

Exercise 10

1. Open a browser window and visit www.redes.upv.es
2. Type the following rule: `sudo iptables -t nat -A OUTPUT -p tcp -d 158.42.0.0/16 --dport 80 -j DNAT --to 158.42.4.23`
3. Reload the website www.redes.upv.es. What happens?
4. Attempt to access the web www.disca.upv.es
5. Delete the rule added in step 2.

If we go through the rule in the exercise 10 rule, we see that the rule is neither `-j LOG` and DROP as in previous cases but DNAT. This rule allows rewriting the addresses and/or destination ports of a segment. In this case, all TCP segments to web servers on campus (subnet 158.42.0.0/16) is redirected to the web server at the address 158.42.4.23 . Also notice that the rule specifies a new table (`-t nat`), while so far the table used was the default table (`-t filter`).

Changes in other fields

For different purposes it is possible to modify the values of other fields in our traffic. One of the candidates is the type of service field (TOS) of the IP header. Customized for a particular application may adjust the value of the field to the needs of it. Even though its value is not respected beyond our system, it may be interesting for different traffic flows can be simultaneously treated adequately in our choice.

For the last example we will choose something simpler: the field of time to live (TTL) of the IP header.

It is known that the value used by our computer is in the default operating system settings, and although you can modify most managers have no reason to do so. However, not all operating systems use the same value, and if we may see computers with different initial values for TTL (we use the ping is to be determined).

Although it may be an urban legend, says that some ISPs may consider whether a particular client is currently one or more traffic with TTL values to assume that the customer may have several computers (with different operating systems) to share the network connection . Currently in Spain the majority of ISPs just do not seem to not have anything against it but also give the NAT router so that customers can easily share your connection among multiple computers.

Anyway, suppose you want to modify the value of the TTL field for a certain type of traffic (not for all our packages). You can create a rule with iptables to make that shift selective.

Exercise 11

1. How do you know the value of the TTL with which you send your traffic?
2. Type the following rule: `sudo iptables -t mangle -A POSTROUTING -j TTL --ttl-set 5`
3. Check if you can access the websites: www.upv.es, www.etsid.upv.es , and www.uv.es

With this value can be only four jumps before the datagram is discarded. This greatly limits the number of networks that can be visited. A value of 1 would prevent a single pass through the router and only allow direct communication with other computers on the same subnet.