# IPTABLES Summarizing Assignment

***by Amit Blum***

May 2016

## Table of Contents

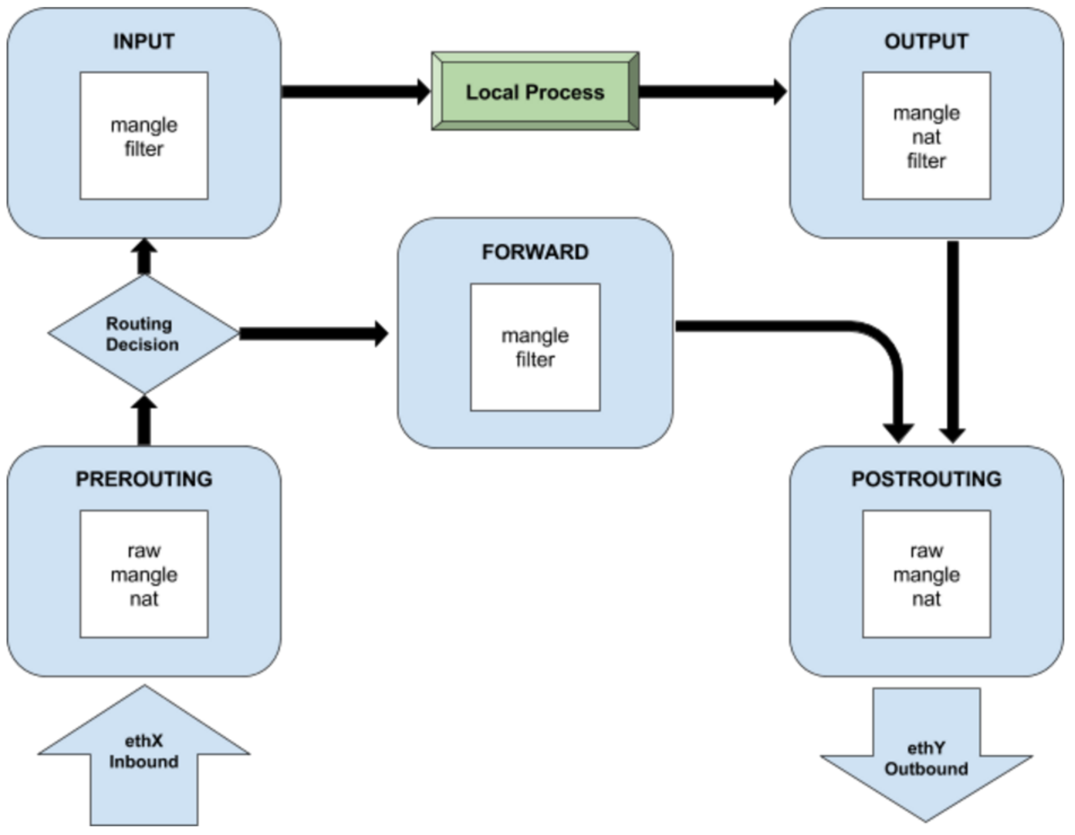
by Amit Blum
May 2016

# 0. Introduction

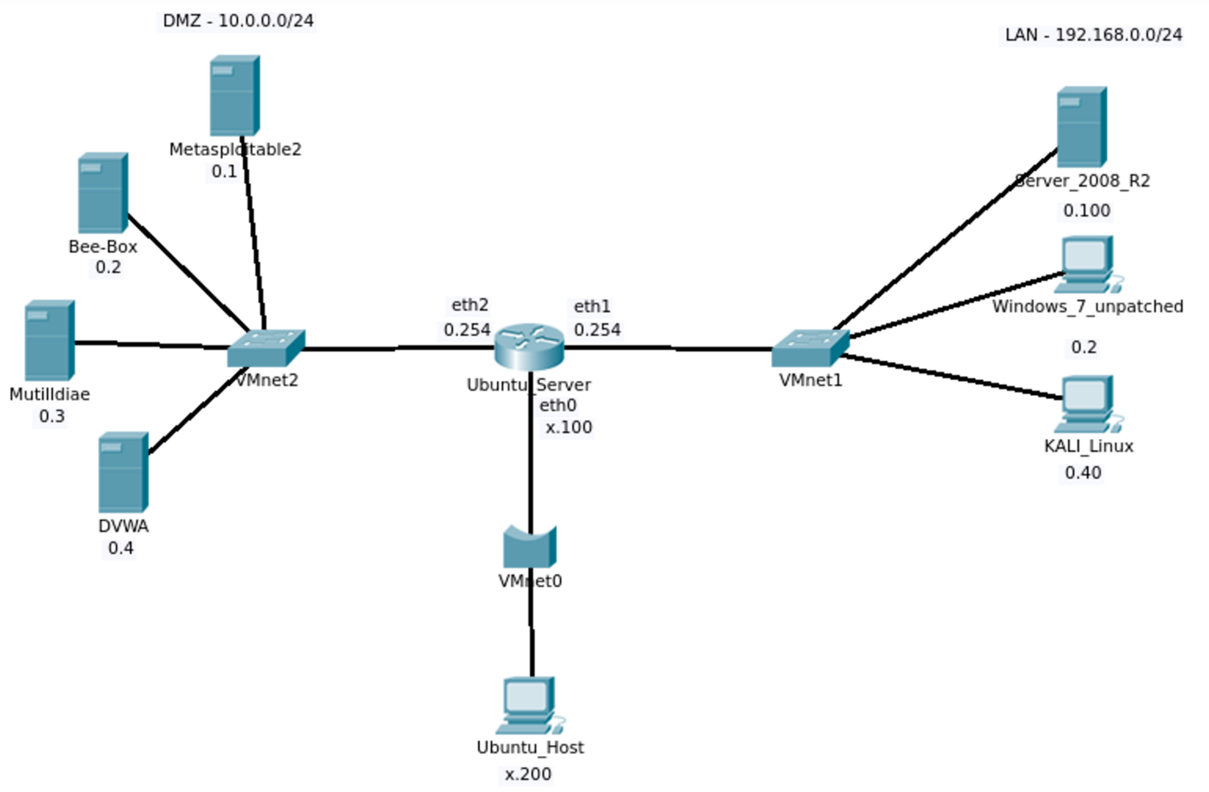
## 0.1 What is iptables?

iptables is a rule-based open-source firewall system that is normally pre-installed on Unix/Linux operating systems. **This makes it one of the most common firewalls in the world** - as it can be found on many Servers, PCs, Laptops, Smartphones (Android), Arduino devices, and many many more. It is also free and robust, and many enterprise firewall products are based on the iptables source code - making the familiarity with it very important for any organization security, management and IT employees.
iptables rules are written within chains - **each chain in a different position in the data flow of packets** to from and through the firewall.
Each chain has several tables with rules. **Each table with its own unique capabilities and uses**. For example - filter rules inspect headers and filters the packets accordingly, while mangle rules can do deep header inspection and even change header values.
Other than the default chains (INPUT, OUTPUT, FORWARD, PREROUTING and POSTROUTING) **custom chains can be made to create a more machine specific data flow logic**.
A custom chain effectively acts as a function/procedure call. If a packet is sent to a custom chain and no rules were matched within the custom chain, the packet will be sent back to the chain that sent it to the custom chain.

## 0.2 iptables (Default) Chains and Tables Flowchart:



## 0.3 Assignment and Topology



Create the lab topology in VMware with the virtual machines and virtual switches depicted in the above scheme.

| | |
|---|---|
| LAN Network: | 192.168.0.0/24 |
| DMZ Network: | 10.0.0.0/24 |
| OUTSIDE: | use any IP range for your convenience |

Main Ubuntu server functions as router, DNS, and DHCP server for all the local networks connected to it.

- **Create iptables rules on the main Ubuntu server that adequately protect it, the DMZ and LAN.**

## 0.4 Methodology:

In solving the assignment I used the rule base that was leaned in class as reference (with a few additions, changes, and twists of my own) - since I found this configuration to be extremely robust and secure.
It is still, however, quite hard to explain the data flow logic in this configuration in meaningful way without breaking it down to the bare details the way I did in the following sections.

I would like to mention however that during my research in the WWW I found a lot of very interesting, very elegant and very robust configurations. **One such configuration is the "Bogus -> Always -> Enemies -> Allow"** (http://www.lammertbies.nl/comm/info/iptables.html) custom chain flow, which blew my mind in how simple, elegant and secure it is. The core of this configuration can and should in my opinion be assimilated into any single-role machine/server with a need for a basic firewall.

## 1. Basic Configuration of iptables:

First we clear the iptables of any previous rules that may exist:

```
iptables -F
```

1.1 **Policy** - change the policy to drop on all chains - effectively "enabling" iptables and letting it perform as "whitelist" like most common modern firewalls:

```
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
```

*** Note that if the configuration is done via SSH then it is important that we change the policy to drop only after the management rules are in place!
Changing the policy to drop without having management rules in place will cause the SSH connection to get dropped and you will no longer have remote access to the server.

1.2 **Stateful** - configure the iptables to follow a session and allow an established/related session to be accepted, on all chains. We do this using the conntrack module:

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

1.3 **Loopback** - loopback is virtual communication port on almost any computer. Sending data streams through it allows them to flow back to the source without processing or modification. This is done primarily for testing the transmission or infrastructure, but many applications also use it for their own purposes. Thus, enabling loopback communication is extremely important on practically any machine:

```
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
```

1.4 **Malicious** - lets create a chain for any "weird" packets that are almost definitely malicious and cause any packets that are matched in it to be dropped:

```
iptables -N MALICIOUS-CHECK
iptables -A INPUT -j MALICIOUS-CHECK
iptables -A OUTPUT -j MALICIOUS-CHECK
iptables -A FORWARD -j MALICIOUS-CHECK
```

```
iptables -N MALICIOUS-DETECTED
iptables -A MALICIOUS-DETECTED -j DROP
```

Conntrack module allows for identifying many such packets with the "invalid" tag. Another example is a new TCP packet with wrong flags lit. Let's send such packets to the MALICIOUS chain (from all chains):

```
iptables -A MALICIOUS-CHECK -m conntrack --ctstate INVALID -j MALICIOUS-DETECTED
iptables -A MALICIOUS-CHECK -m conntrack --ctstate NEW -p tcp ! --syn -j MALICIOUS-DETECTED
```

We also want any NEW malicious packet to not be dropped, but rejected with tcp-reset - so as to better hide the existence of the firewall:

***iptables -I MALICIOUS-DETECTED 1 -m conntrack --ctstate NEW -p tcp ! --syn -j REJECT --reject-with tcp-reset***

In the future, we can add any "malicious" rules to send to the MALICIOUS chain.

*1.5* **Log the malicious traffic** - before dropping any packet in the MALICIOUS chain, we would like to log it:

***iptables -I MALICIOUS-DETECTED 1 -m limit --limit 2/min -j LOG --log-prefix "Malicious Traffic" --log-level 7***

1.6 **Anti-Spoofing** - Let's create a Spoof chain to drop (and also log) any spoofed communication to or through (INPUT and FORWARD chains) our firewall. These rules should change according to the local networks/DMZ the firewall is connected to and its interfaces:

***iptables -N SPOOF-CHECK***
***iptables -A INPUT -j SPOOF-CHECK***
***iptables -A FORWARD -j SPOOF-CHECK***

***iptables -N SPOOF-DETECTED***
***iptables -A SPOOF-DETECTED -m limit --limit 2/min -j LOG --log-prefix "Spoof detected" --log-level 7***
***iptables -A SPOOF-DETECTED -j DROP***

***iptables -A SPOOF-CHECK -i eth1 ! -s 192.168.0.0/24 -j SPOOF-DETECTED***
***iptables -A SPOOF-CHECK -i eth2 ! -s 10.0.0.0/24 -j SPOOF-DETECTED***
***iptables -A SPOOF-CHECK -s 127.0.0.0/8 ! -i lo -j SPOOF-DETECTED***
***iptables -A SPOOF-CHECK -s 169.254.0.0/16 -j SPOOF-DETECTED***

Alternatively (or additionally which is what I did) we can enable "Reverse Path Filtering" which blocks any packet coming from an interface with a source ip outside that interface's network. To do this we need to uncomment rp_filter lines in /etc/sysctl.conf. This is shown in section 5.1.

## 2. Implementation of Management and Stealth Rules:

### 2.1 Allow remote SSH to the firewall:

Security measures to SSH (and other services as well) should always be put in place. This can be done in many different ways. Different security measures can also be taken in tandem. I will elaborate on a few plausible options:

a. If the firewall administrators are behind one or more **static IPs**, then the safest configuration is to ACCEPT all communication in the INPUT chain from those static IPs in port 22.
   **Advantages**: easy to configure. Considered very safe among other options, and other safety measures can easily be configured in tandem with this one.
   **Disadvantages**: overly simple. Susceptible to spoof attacks if security rests solely on it.

b. **Changing the port of the specified service to something non-standard**. This makes scanning the machine and identifying weaknesses harder for any hacker.
   **Advantages**: Easy to configure. Makes the hackers life a bit harder as he is required to sniff and analyze the data inbound on each port to determine what protocol/service is running on it.
   **Disadvantages**: Non-standard ports require micro management of every client who is required to use that service. Can make the service seem like an unmanageable mess to untrained/inexperienced administrators.

c. Allow anyone from anywhere to access SSH to your server while taking under account and **preventing brute-force attacks in a flat-out "no nonsense" approach**. There are many ways to do this, one "easy" way is by using an intrusion prevention app such as "**DenyHosts**" or "**Fail2ban**" (written in python). Such programs monitor log files of different services, detect failed login attempts to those different services (SSH, Apache access log, Asterisk log etc), and ban suspicious IP addresses that seem to be trying to hack in the system.
   **Advantages**: can be configured easily to monitor not just SSH but more complex layer 7 protocols that require authentication such as SIP (VoIP), RADIUS, IMAP, POP, Telnet etc.
   **Disadvantages**: fails to protect against complex distributed brute-force attacks. Lacks IPV6 support. Requires a lot of micro-management of accidently blocked IP addresses. Can block critical administrators out by mistake, or on purpose (admin lockout DOS attack).

d. Allow anyone from anywhere to access SSH to your server, while taking under account and **preventing brute-force attacks in a rate-limiting approach**. This is the approach I picked to protect SSH in my Ubuntu Server.
   While there are existing apps that help do this (like "Stockade" or "IPQ BDB") I chose to do this manually with a custom IP-Tables SSH-MANAGEMENT chain.
   **Advantages**: Removes the micro-management required on accidental lockouts in the previous approach, hackers find it harder to identify the security measure they are facing.
   **Disadvantages**: Still susceptible to administrator lockout DOS, fails to protect against distributed "Zombie" type, slow brute force attacks.

e. Using configurable apps like "**Knockd**" to hide port 22 until a secret "knock" is sent to the server. After the secret knock is detected the IP address which sent the knock is then accepted in the iptables for the specified port 22.
   **Advantages**: Very secure. I'd say even Paranoid. Should only be used on most important servers with the most sensitive data.
   **Disadvantages**: A lot of micro-management required in configuring the server and all the clients that need access to it.

As I previously mentioned, I chose to implement a rate-limiting approach to SSH access security (option d), but with an added twist to help prevent admin lockout DOS attacks.
First, create a SSH-IN chain and send everything that is SSH to it:

***iptables -N SSH-IN***
***iptables -A INPUT -p tcp --dport 22 -j SSH-IN***

Next, create a rule that rejects (not drops, to confuse any hacker attempting brute-force) any 3rd SSH connection attempt within a 30 second timeframe with the "recent" module. Additionally check rttl to make ip spoofing and admin lock DOS attacks that much harder:

***iptables -A SSH-IN -p tcp -m recent --name sshbf --update --rttl --hitcount 3 --seconds 30 -j REJECT --reject-with tcp-reset***

Lastly, we create rules in the SSH-IN to create this timestamp and accept any new requests that are not already set:

***iptables -A SSH-IN -m recent --name sshbf --set -j ACCEPT***

2.2 <u>**SSH Brute-Force Logging**</u>: to log such attempts we will again use the recent module with the --rcheck switch, and same hitcount and seconds parameters. Additionally, since I don't want our log to "explode" when someone tries to brute-force our server, I limit the logs to, let's say, 2 per minute with the "limit" module.
This rule needs to be at the head of the SSH-IN chain, so I will insert it at position 1:

***iptables -I SSH-IN 1 -m recent --name sshbf --rcheck --hitcount 3 --seconds 30 -m limit --limit 2/min -j LOG --log-prefix "SSH-BruteForce" --log-level 7***

2.3 <u>**Allowing ICMP echo-requests**</u>: Here, the administrator must decide on the policy. sending out and replying to ping is a common way to keep track of network connectivity, however enabling it makes the network very vulnerable (and easy) to scan. I have decided to allow all ICMP inbound and outbound to and from all input interfaces, on all chains, since I'm quite positive my configuration is very secure, even if I get scanned. Additionally, just for sanity's sake, I limit the amount of ICMP packets allowed to 1 per second if a burst is detected.

***iptables -A INPUT -p icmp --icmp-type 8 -m limit --limit 1/s --limit-burst 3 -j ACCEPT***
***iptables -A OUTPUT -p icmp --icmp-type 8 -m limit --limit 1/s --limit-burst 3 -j ACCEPT***
***iptables -A FORWARD -p icmp --icmp-type 8 -m limit --limit 1/s --limit-burst 3 -j ACCEPT***

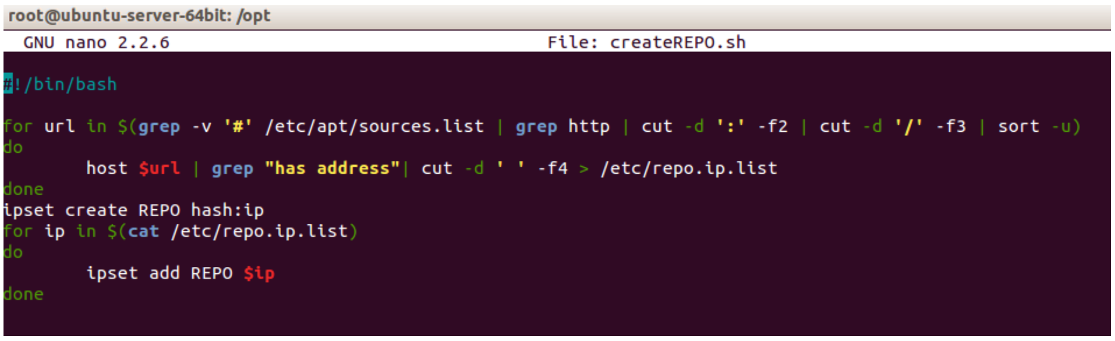
2.4 <u>**Other allowed services and INPUT chain cleanup**</u>: Our server specifically also functions and DNS (bind9) and DHCP server. we need to allow on the input chain packets with DNS and DHCP requests from the LAN and DMZ, as well as allow OUTPUT on DNS for urls that are not cached:

***iptables -A OUTPUT -p udp --dport 53 -j ACCEPT***
***iptables -A INPUT -i eth1 -p udp --dport 53 -j ACCEPT***
***iptables -A INPUT -i eth2 -p udp --dport 53 -j ACCEPT***
***iptables -A INPUT -i eth1 -p udp --dport 67:68 --sport 67:68 -j ACCEPT***
***iptables -A INPUT -i eth2 -p udp --dport 67:68 --sport 67:68 -j ACCEPT***

Since no more services should be allowed on our main server at this point, I will now create an input cleanup chain (CLEANUP-IN) and drop anything coming in to it. In the future, if I want to have temporary rules or to log specific types of packets, I can do so on the cleanup chain as whatever is in it is not part of the core firewall:

***iptables -N CLEANUP-IN***
***iptables -A INPUT -j CLEANUP-IN***

2.5 __**Other allowed services (apt-get update & install) and OUTPUT chain cleanup**__: Our server is not supposed to have any outbound initiated communication at this point except one - allowing "**apt-get update**" and "**apt-get install**" but only from well known (Canonical) repositories.
To do this, first I need to install ipset on the server (temporarily change policy to accept on output chain) "***apt-get install ipset***".
Then, I have built a bash script that finds all the ip addresses of official repositories from the file */etc/apt/sources.list and creates an ipsed REPO holding them.* I put the script in /opt for reference:



```
root@ubuntu-server-64bit: /opt
  GNU nano 2.2.6                              File: createREPO.sh

#!/bin/bash

for url in $(grep -v '#' /etc/apt/sources.list | grep http | cut -d ':' -f2 | cut -d '/' -f3 | sort -u)
do
        host $url | grep "has address"| cut -d ' ' -f4 > /etc/repo.ip.list
done
ipset create REPO hash:ip
for ip in $(cat /etc/repo.ip.list)
do
        ipset add REPO $ip
done
```

Give the script run privileges: ***chmod 775 /opt/createREPO.sh***
Run this script at startup to make sure you have an ipset ready before you load iptables by adding "./opt/craeteREPO.sh" line to /etc/rc.local
Now, accept the repo with the set module to the OUTPUT chain in port 80 to be able to update and install from apt-get:

***iptables -A OUTPUT -m set --match-set REPO dst -p tcp --dport 80 -j ACCEPT***

We can now also add to crontab a scheduled task to update (or even upgrade) regularly but for now I'll leave it to the administrator to update manually when he wishes.
Since no more services should be allowed on the OUTPUT chain, let's do a cleanup:

***iptables -N CLEANUP-OUT***
***iptables -A OUTPUT -j CLEANUP-OUT***

## 3. Implement Internal Rules:

Let's begin by implementing the NAT rules for the LAN and DMZ:

```
iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -o eth0 -j MASQUERADE
iptables -t nat -A POSTROUTING -s 10.0.0.0/29 -o eth0 -j MASQUERADE
iptables -t nat -A PREROUTING -d 172.16.107.101/32 -i eth0 -j DNAT --to-destination 10.0.0.1
iptables -t nat -A PREROUTING -d 172.16.107.102/32 -i eth0 -j DNAT --to-destination 10.0.0.2
iptables -t nat -A PREROUTING -d 172.16.107.103/32 -i eth0 -j DNAT --to-destination 10.0.0.3
iptables -t nat -A PREROUTING -d 172.16.107.104/32 -i eth0 -j DNAT --to-destination 10.0.0.4
```

3.1 **Allow HTTP and HTTPS from LAN** to anywhere:

```
iptables -A FORWARD -s 192.168.0.0/24 -i eth1 -m multiport -p tcp --dport 80,443 -j ACCEPT
```

DNS was already permitted on the INPUT and OUTPUT chains (Section 2.4), since our server performs as DNS.

3.2 **Allow active FTP from LAN to Metasploitable** machine:

```
iptables -A FORWARD -s 192.168.0.0/24 -d 10.0.0.1 -p tcp --dport 21 -j ACCEPT
iptables -A FORWARD -s 10.0.0.1 -d 192.168.0.0/24 -p tcp --sport 20 -j ACCEPT
```

Note: I don't allow anyone from the outside to access FTP service on the DMZ.

3.3 **Prevent SYN attacks** (including HTTP or HTTPS) by throttling them back to 20/min per IP using the hashlimit module:

```
iptables -A FORWARD -o eth2 -p tcp --syn -m hashlimit --hashlimit 20/min --hashlimit-burst 30 --
hashlimit-mode srcip --hashlimit-name synthrottle -j ACCEPT
iptables -A FORWARD -o eth2 -p tcp --syn -j DROP
```

Now that we are safe from syn-attacks (and also non-malicious legitimate surge of requests that may overload our servers) – we now want to allow HTTP and HTTPS from outside to DMZ:

```
iptables -A FORWARD -i eth0 -o eth2 -m multiport -p tcp --dport 80,443 -j ACCEPT
```

3.4 **Extra protection against SynFlood DOS attacks**:

Even though the hashlimit rule is enough to stop any SynFlood attacks from single IP addresses, we want to detect these attacks better (from automated tools, or distributed attacks), we want to stop them earlier in the chain by using the mangle tables instead of the filter, and we also want to log these attempts.
Start by creating a mangle chain called SYNFLOOD and send all new http/https with a too low mss to it:

```
iptables -t mangle -N SYNFLOOD
iptables -t mangle -A PREROUTING -p tcp -m multiport --dport 80,443 -m conntrack --ctstate NEW
-m tcpmss ! --mss 500:65535 -j SYNFLOOD
```

Now, Log 2 attempts per minute, and drop the packets:

```
iptables -t mangle -A SYNFLOOD -m limit --limit 2/min -j LOG --log-prefix “DOS-Attack” --log-level
7
iptables -t mangle -A SYNFLOOD -j DROP
```

We are now safe against any automated tool that does SynFlood attacks that I am aware of. However, just as an extra security measure, **we will enable syncookies on all our Linux servers** – the main server and every DMZ server as well. This is shown in section 5.1.

3.5 **Allow servers from DMZ to update from the official repository** of Ubuntu:
***iptables -A FORWARD -i eth2 -m set --match-set REPO dst -p tcp --dport 80 -j ACCEPT***

## 4. Clean Up Rules

We have already created the CLEANUP-IN and CLEANUP-OUT chains. Let's create CLEANUP-FWD as well:

```
iptables -N CLEANUP-FWD
iptables -A FORWARD -j CLEANUP-FWD
```

Anything coming into these clean-up chains will be dropped/rejected. The purpose of these chains is to log the filtered packets as suspicious for further investigation, and to think of a clever way to handle the filtered packets - i.e. DROP vs REJECT.

**DROP vs REJECT** - contrary to popular belief, DROP does not give better security than REJECT. DROP only inconveniences legitimate users (by making them wait for a time-out) and it's effectively no protection from malicious ones.
Additionally DROP practically screams out "THERE IS A FIREWALL HERE" to any half decent automated scanning tools - allowing a good hacker to quickly assess the machine he's facing, the security measures in place, and to find weak spots / holes from which to try and break in.
REJECT in the other hand, imitates to a degree "normal" OS behavior on truly closed ports. Taking the time to reject correctly all the possible packets (almost impossible) effectively makes the firewall invisible to scanning tools - making a hacker's life a lot harder.

So when do we use DROP? When we don't want to create overhead on the Ubuntu server when it actually is attacked by malicious DDOS or brute force. Such attacks tend to add a lot processing and interface throughput inbound and outbound on the firewall. That is also why these DROP rules (like we did for spoof and synflood) should be as high as possible on the rule tables of the firewall (so that they will be matched as quickly as possible.

I will implement a few such clean-up rules in the CLEANUP-IN and CLEANUP-FWD chains, as CLEANUP-OUT does not require REJECT.

```
iptables -N CLEANUP-REJECT
iptables -A CLEANUP-REJECT -p udp -j REJECT --reject-with icmp-port-unreachable
iptables -A CLEANUP-REJECT -p tcp -j REJECT --reject-with tcp-reset
iptables -A CLEANUP-REJECT -j REJECT --reject-with icmp-proto-unreachable

iptables -A CLEANUP-IN -m limit --limit 2/min -j LOG --log-prefix "Cleanup-INPUT" --log-level 7
iptables -A CLEANUP-IN -j CLEANUP-REJECT

iptables -A CLEANUP-FWD -m limit --limit 2/min -j LOG --log-prefix "Cleanup-FORWARD" --log-level 7
iptables -A CLEANUP-FWD -j CLEANUP-REJECT

iptables -A CLEANUP-OUT -m limit --limit 2/min -j LOG --log-prefix "Cleanup-OUTPUT" --log-level 7
iptables -A CLEANUP-OUT -j DROP
```

## 5. Are we done?! (Conclusion)

iptables configuration-wise yes - but let's do 2 more things.

i. **ipv6 disable:** We are still vulnerable to a very devastating (and easy to reproduce) ipv6 Router Advertisements Flood attack. Since we don't have any devices/servers in our networks that use ipv6 - the easiest way to protect ourselves from this attack (and any other unknown / untested attacks in this new protocol) - we just need to disable it in ***/etc/sysctl.conf***. This is shown in section 5.1.

ii. ***Persistent iptables:*** If we'd like to make our iptables persistent (to reload again the same way on every reboot of the machine) we need to make sure it loads from a file.
***iptables-save > /etc/iptables.rules*** (The output of this file is shown in section 5.2)
add to rc.local file the command ***iptables-restore < /etc/iptables.rules*** after the repo.sh or the iptables load may fail as there is a rule requiring an ipset list named REPO to exist. These are shown in the following sections.

That's it. We need to remember though that iptables is just a firewall that handles connections. If a legitimate connection has been made then the data being sent on layer 7 (application) may still be malicious and this firewall does not deeply inspect the data stream or individual packets.

All our servers are still very vulnerable to specific service attacks such as SQL injection, XSS, file inclusion vulnerabilities, misconfigurations of services, users, and permissions, viruses/worms and other malicious codes. Even layer 7 DOS attacks (like slowlorris) may still pose a serious threat.

We need to asses each machine in our DMZ, configure a specific iptables / firewall for it, and take additional security measures like Anti-Viruses, WAFs, Apache QOS mod, Apache modsecurity, IPS/IDS, encryption, obfuscation, and many many other options. Again, depending on the vulnerabilities we may find on said machines. This, however, is outside the scope of this document.

Hopefully the readers of this documented enjoyed my "two cents", and found some value in some of the configurations I set.

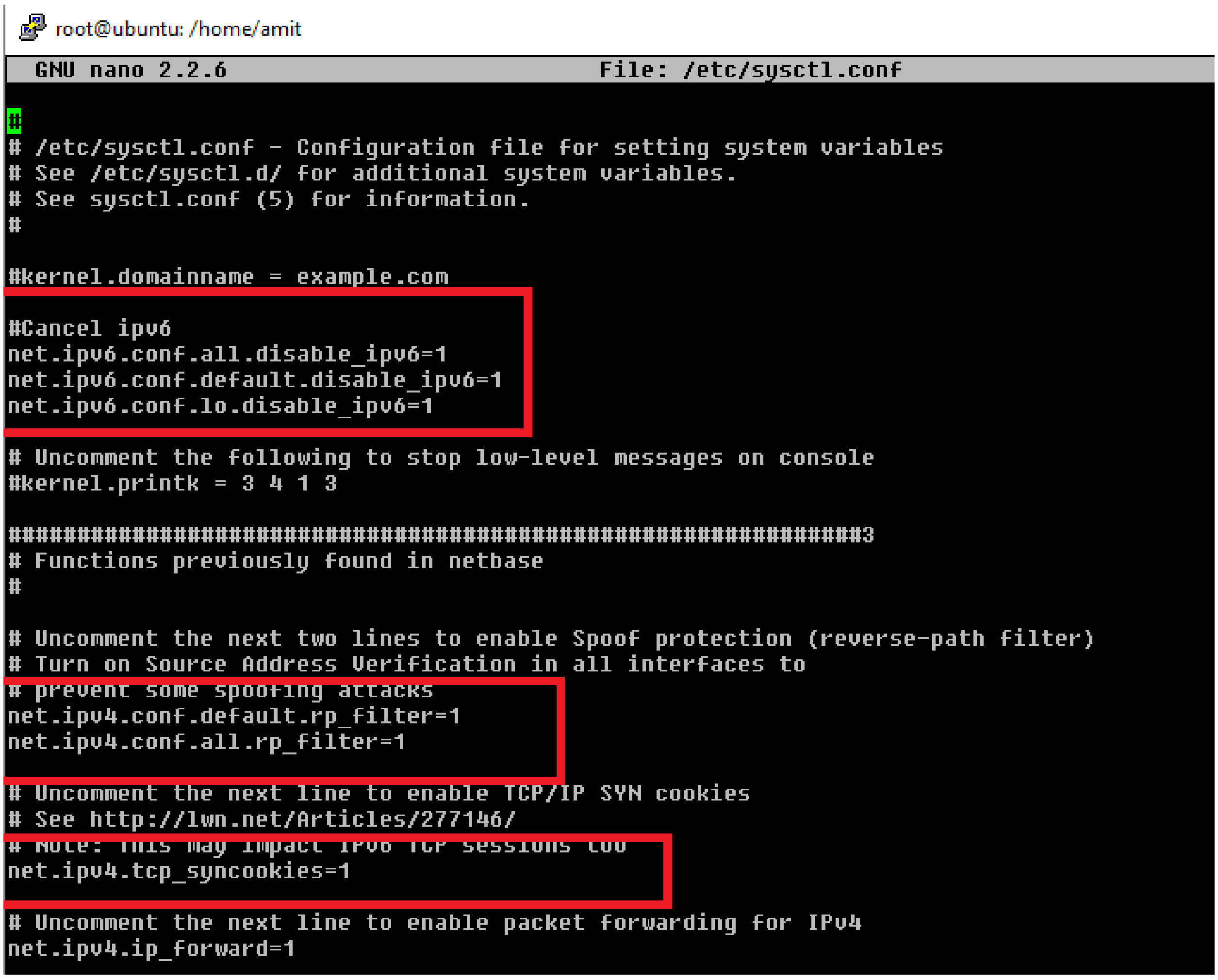
Lior, I hope that the title page's pictures were amusing enough for you ;)

by Amit Blum
May 2016

## 5.1. /etc/sysctl.conf

*disable_ipv6*, *rp_filter*, and *syncookies* should be configured on all our servers (Ubuntu main server and all DMZ Ubuntu machines).

*ip_forward* should only be enabled on our main server which performs as a router between networks.



```
root@ubuntu: /home/amit
  GNU nano 2.2.6                          File: /etc/sysctl.conf

#
# /etc/sysctl.conf - Configuration file for setting system variables
# See /etc/sysctl.d/ for additional system variables.
# See sysctl.conf (5) for information.
#

#kernel.domainname = example.com

#Cancel ipv6
net.ipv6.conf.all.disable_ipv6=1
net.ipv6.conf.default.disable_ipv6=1
net.ipv6.conf.lo.disable_ipv6=1

# Uncomment the following to stop low-level messages on console
#kernel.printk = 3 4 1 3

##############################################################3
# Functions previously found in netbase
#

# Uncomment the next two lines to enable Spoof protection (reverse-path filter)
# Turn on Source Address Verification in all interfaces to
# prevent some spoofing attacks
net.ipv4.conf.default.rp_filter=1
net.ipv4.conf.all.rp_filter=1

# Uncomment the next line to enable TCP/IP SYN cookies
# See http://lwn.net/Articles/277146/
# Note: This may impact IPv6 TCP sessions too
net.ipv4.tcp_syncookies=1

# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1
```

## 5.2. iptables-save output:

```
# Generated by iptables-save v1.4.21 on Wed May  4 10:17:29 2016
*mangle
:PREROUTING ACCEPT [480:32775]
:INPUT ACCEPT [474:32307]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [498:60476]
:POSTROUTING ACCEPT [498:60476]
:SYNFLOOD - [0:0]
-A PREROUTING -p tcp -m multiport --dports 80,443 -m conntrack --ctstate NEW -m tcpmss ! --mss
500:65535 -j SYNFLOOD
-A SYNFLOOD -m limit --limit 2/min -j LOG --log-prefix "“DOS-Attack”" --log-level 7
-A SYNFLOOD -j DROP
COMMIT
# Completed on Wed May  4 10:17:29 2016
# Generated by iptables-save v1.4.21 on Wed May  4 10:17:29 2016
*nat
:PREROUTING ACCEPT [14:1411]
:INPUT ACCEPT [1:78]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
-A PREROUTING -d 172.16.107.101/32 -i eth0 -j DNAT --to-destination 10.0.0.1
-A PREROUTING -d 172.16.107.102/32 -i eth0 -j DNAT --to-destination 10.0.0.2
-A PREROUTING -d 172.16.107.103/32 -i eth0 -j DNAT --to-destination 10.0.0.3
-A PREROUTING -d 172.16.107.104/32 -i eth0 -j DNAT --to-destination 10.0.0.4
-A POSTROUTING -s 192.168.0.0/24 -o eth0 -j MASQUERADE
-A POSTROUTING -s 10.0.0.0/29 -o eth0 -j MASQUERADE
COMMIT
# Completed on Wed May  4 10:17:29 2016
# Generated by iptables-save v1.4.21 on Wed May  4 10:17:29 2016
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:CLEANUP-FWD - [0:0]
:CLEANUP-IN - [0:0]
:CLEANUP-OUT - [0:0]
:CLEANUP-REJECT - [0:0]
:MALICIOUS-CHECK - [0:0]
:MALICIOUS-DETECTED - [0:0]
:SPOOF-CHECK - [0:0]
:SPOOF-DETECTED - [0:0]
:SSH-IN - [0:0]
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -j MALICIOUS-CHECK
-A INPUT -j SPOOF-CHECK
```

```
-A INPUT -p tcp -m tcp --dport 22 -j SSH-IN
-A INPUT -p icmp -m icmp --icmp-type 8 -m limit --limit 1/sec --limit-burst 3 -j ACCEPT
-A INPUT -i eth1 -p udp -m udp --dport 53 -j ACCEPT
-A INPUT -i eth2 -p udp -m udp --dport 53 -j ACCEPT
-A INPUT -i eth1 -p udp -m udp --sport 67:68 --dport 67:68 -j ACCEPT
-A INPUT -i eth2 -p udp -m udp --sport 67:68 --dport 67:68 -j ACCEPT
-A INPUT -j CLEANUP-IN
-A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -j MALICIOUS-CHECK
-A FORWARD -j SPOOF-CHECK
-A FORWARD -p icmp -m icmp --icmp-type 8 -m limit --limit 1/sec --limit-burst 3 -j ACCEPT
-A FORWARD -s 192.168.0.0/24 -i eth1 -p tcp -m multiport --dports 80,443 -j ACCEPT
-A FORWARD -s 192.168.0.0/24 -d 10.0.0.1/32 -p tcp -m tcp --dport 21 -j ACCEPT
-A FORWARD -s 10.0.0.1/32 -d 192.168.0.0/24 -p tcp -m tcp --sport 20 -j ACCEPT
-A FORWARD -o eth2 -p tcp -m tcp --tcp-flags FIN,SYN,RST,ACK SYN -m hashlimit --hashlimit-upto 20/min
--hashlimit-burst 30 --hashlimit-mode srcip --hashlimit-name synthrottle -j ACCEPT
-A FORWARD -o eth2 -p tcp -m tcp --tcp-flags FIN,SYN,RST,ACK SYN -j DROP
-A FORWARD -i eth0 -o eth2 -p tcp -m multiport --dports 80,443 -j ACCEPT
-A FORWARD -i eth2 -p tcp -m set --match-set REPO dst -m tcp --dport 80 -j ACCEPT
-A FORWARD -j CLEANUP-FWD
-A OUTPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A OUTPUT -o lo -j ACCEPT
-A OUTPUT -j MALICIOUS-CHECK
-A OUTPUT -p icmp -m icmp --icmp-type 8 -m limit --limit 1/sec --limit-burst 3 -j ACCEPT
-A OUTPUT -p udp -m udp --dport 53 -j ACCEPT
-A OUTPUT -p tcp -m set --match-set REPO dst -m tcp --dport 80 -j ACCEPT
-A OUTPUT -j CLEANUP-OUT
-A CLEANUP-FWD -m limit --limit 2/min -j LOG --log-prefix Cleanup-FORWARD --log-level 7
-A CLEANUP-FWD -j CLEANUP-REJECT
-A CLEANUP-IN -m limit --limit 2/min -j LOG --log-prefix Cleanup-INPUT --log-level 7
-A CLEANUP-IN -j CLEANUP-REJECT
-A CLEANUP-OUT -m limit --limit 2/min -j LOG --log-prefix Cleanup-OUTPUT --log-level 7
-A CLEANUP-OUT -j DROP
-A CLEANUP-REJECT -p udp -j REJECT --reject-with icmp-port-unreachable
-A CLEANUP-REJECT -p tcp -j REJECT --reject-with tcp-reset
-A CLEANUP-REJECT -j REJECT --reject-with icmp-proto-unreachable
-A MALICIOUS-CHECK -m conntrack --ctstate INVALID -j MALICIOUS-DETECTED
-A MALICIOUS-CHECK -p tcp -m conntrack --ctstate NEW -m tcp ! --tcp-flags FIN,SYN,RST,ACK SYN -j
MALICIOUS-DETECTED
-A MALICIOUS-DETECTED -m limit --limit 2/min -j LOG --log-prefix "Malicious Traffic" --log-level 7
-A MALICIOUS-DETECTED -p tcp -m conntrack --ctstate NEW -m tcp ! --tcp-flags FIN,SYN,RST,ACK SYN -j
REJECT --reject-with tcp-reset
-A MALICIOUS-DETECTED -j DROP
-A SPOOF-CHECK ! -s 192.168.0.0/24 -i eth1 -j SPOOF-DETECTED
-A SPOOF-CHECK ! -s 10.0.0.0/24 -i eth2 -j SPOOF-DETECTED
-A SPOOF-CHECK -s 127.0.0.0/8 ! -i lo -j SPOOF-DETECTED
-A SPOOF-CHECK ! -s 192.168.0.0/24 -i eth1 -j SPOOF-DETECTED
-A SPOOF-DETECTED -m limit --limit 2/min -j LOG --log-prefix "Spoof detected" --log-level 7
```

```
-A SPOOF-DETECTED -j DROP
-A SSH-IN -m recent --rcheck --seconds 30 --hitcount 3 --name sshbf --mask 255.255.255.255 --rsource -
m limit --limit 2/min -j LOG --log-prefix SSH-BruteForce --log-level 7
-A SSH-IN -p tcp -m recent --update --seconds 30 --hitcount 3 --rttl --name sshbf --mask 255.255.255.255
--rsource -j REJECT --reject-with tcp-reset
-A SSH-IN -m recent --set --name sshbf --mask 255.255.255.255 --rsource -j ACCEPT
COMMIT
# Completed on Wed May  4 10:17:29 2016
```

## 5.3. /etc/rc.local

```
root@ubuntu:/home/amit# cat /etc/rc.local
#!/bin/sh -e

# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.

#apt-get update
./opt/repo.sh
/sbin/iptables-restore < /etc/iptables.rules
exit 0
```

## 5.4. repo.sh

```
root@ubuntu:/home/amit# cat /opt/repo.sh
#!/bin/bash
for url in $(grep -v '#' /etc/apt/sources.list | grep http | cut -d ':' -f2 | cut -d '/' -f3 | sort -u)
do
	repoList=(`host $url |grep "has address" | cut -d ' ' -f4`)
done
ipset create REPO hash:ip

for ip in ${repoList[@]}
do
	ipset add REPO $ip
done
```

## 5.5. ipset

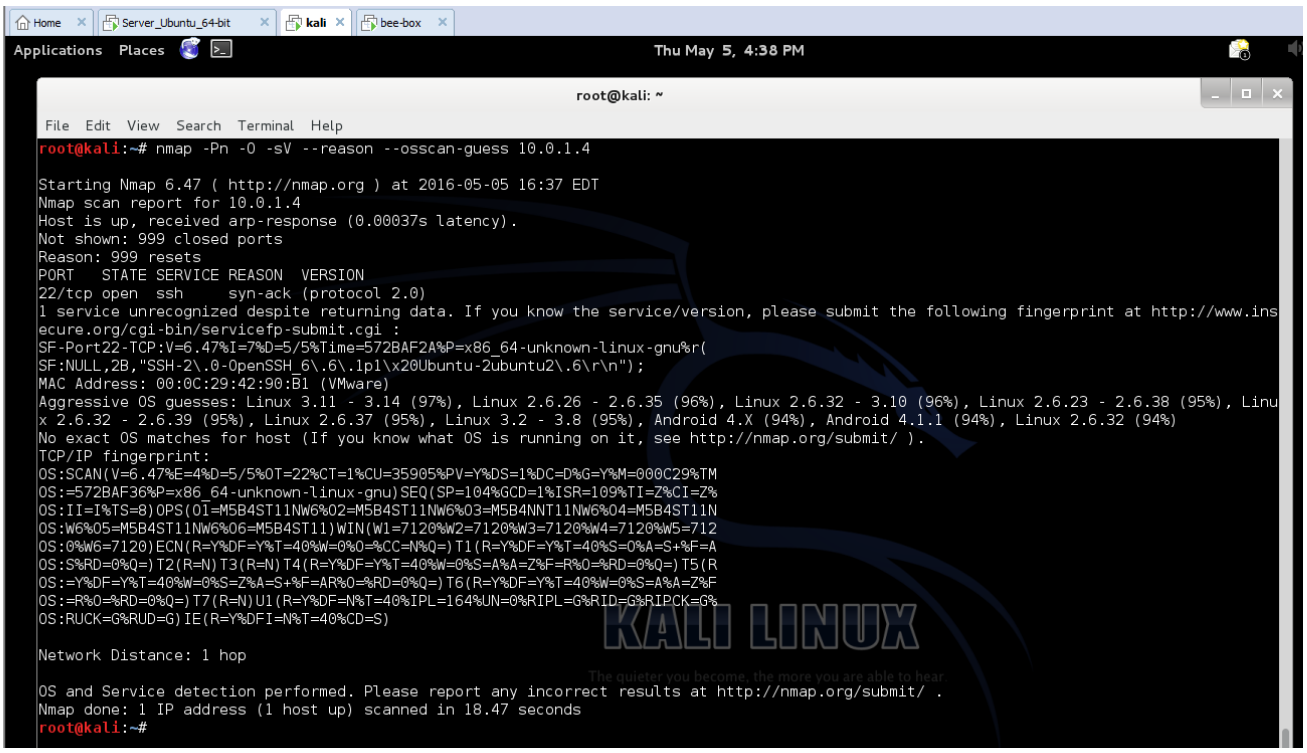
```
root@ubuntu:/opt# ipset list
Name: REPO
Type: hash:ip
Revision: 2
Header: family inet hashsize 1024 maxelem 65536
Size in memory: 16584
References: 0
Members:
91.189.88.161
91.189.88.152
91.189.91.23
91.189.91.26
91.189.88.149
```

## 6. Testing the configuration

Methodology - Kali machine running connected through host bridge interface from outside LAN and DMZ. I will run several scans and attacks and see how the iptables handle them.

### 6.1. Scanning With NMAP

Scan the Ubuntu server against the 1000 most common ports, attempt to guess the OS by the replies, and try to grab banners from any open services:
***nmap -Pn -O -sV --reason --osscan-guess 10.0.1.4***



```
root@kali:~# nmap -Pn -O -sV --reason --osscan-guess 10.0.1.4

Starting Nmap 6.47 ( http://nmap.org ) at 2016-05-05 16:37 EDT
Nmap scan report for 10.0.1.4
Host is up, received arp-response (0.00037s latency).
Not shown: 999 closed ports
Reason: 999 resets
PORT   STATE SERVICE REASON  VERSION
22/tcp open  ssh     syn-ack (protocol 2.0)
1 service unrecognized despite returning data. If you know the service/version, please submit the following fingerprint at http://www.ins
ecure.org/cgi-bin/servicefp-submit.cgi :
SF-Port22-TCP:V=6.47%I=7%D=5/5%Time=572BAF2A%P=x86_64-unknown-linux-gnu%r(
SF:NULL,2B,"SSH-2\.0-OpenSSH_6\.6\.1p1\x20Ubuntu-2ubuntu2\.6\r\n");
MAC Address: 00:0C:29:42:90:B1 (VMware)
Aggressive OS guesses: Linux 3.11 - 3.14 (97%), Linux 2.6.26 - 2.6.35 (96%), Linux 2.6.32 - 3.10 (96%), Linux 2.6.23 - 2.6.38 (95%), Linu
x 2.6.32 - 2.6.39 (95%), Linux 2.6.37 (95%), Linux 3.2 - 3.8 (95%), Android 4.X (94%), Android 4.1.1 (94%), Linux 2.6.32 (94%)
No exact OS matches for host (If you know what OS is running on it, see http://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=6.47%E=4%D=5/5%OT=22%CT=1%CU=35905%PV=Y%DS=1%DC=D%G=Y%M=000C29%TM
OS:=572BAF36%P=x86_64-unknown-linux-gnu)SEQ(SP=104%GCD=1%ISR=109%TI=Z%CI=Z%
OS:II=I%TS=8)OPS(O1=M5B4ST11NW6%O2=M5B4ST11NW6%O3=M5B4NNT11NW6%O4=M5B4ST11N
OS:W6%O5=M5B4ST11NW6%O6=M5B4ST11)WIN(W1=7120%W2=7120%W3=7120%W4=7120%W5=712
OS:0%W6=7120)ECN(R=Y%DF=Y%T=40%W=0%O=%CC=N%Q=)T1(R=Y%DF=Y%T=40%S=O%A=S+%F=A
OS:S%RD=0%Q=)T2(R=N)T3(R=N)T4(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)T5(R
OS:=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F
OS:=R%O=%RD=0%Q=)T7(R=N)U1(R=Y%DF=N%T=40%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%
OS:RUCK=G%RUD=G)IE(R=Y%DFI=N%T=40%CD=S)

Network Distance: 1 hop

OS and Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 18.47 seconds
root@kali:~#
```

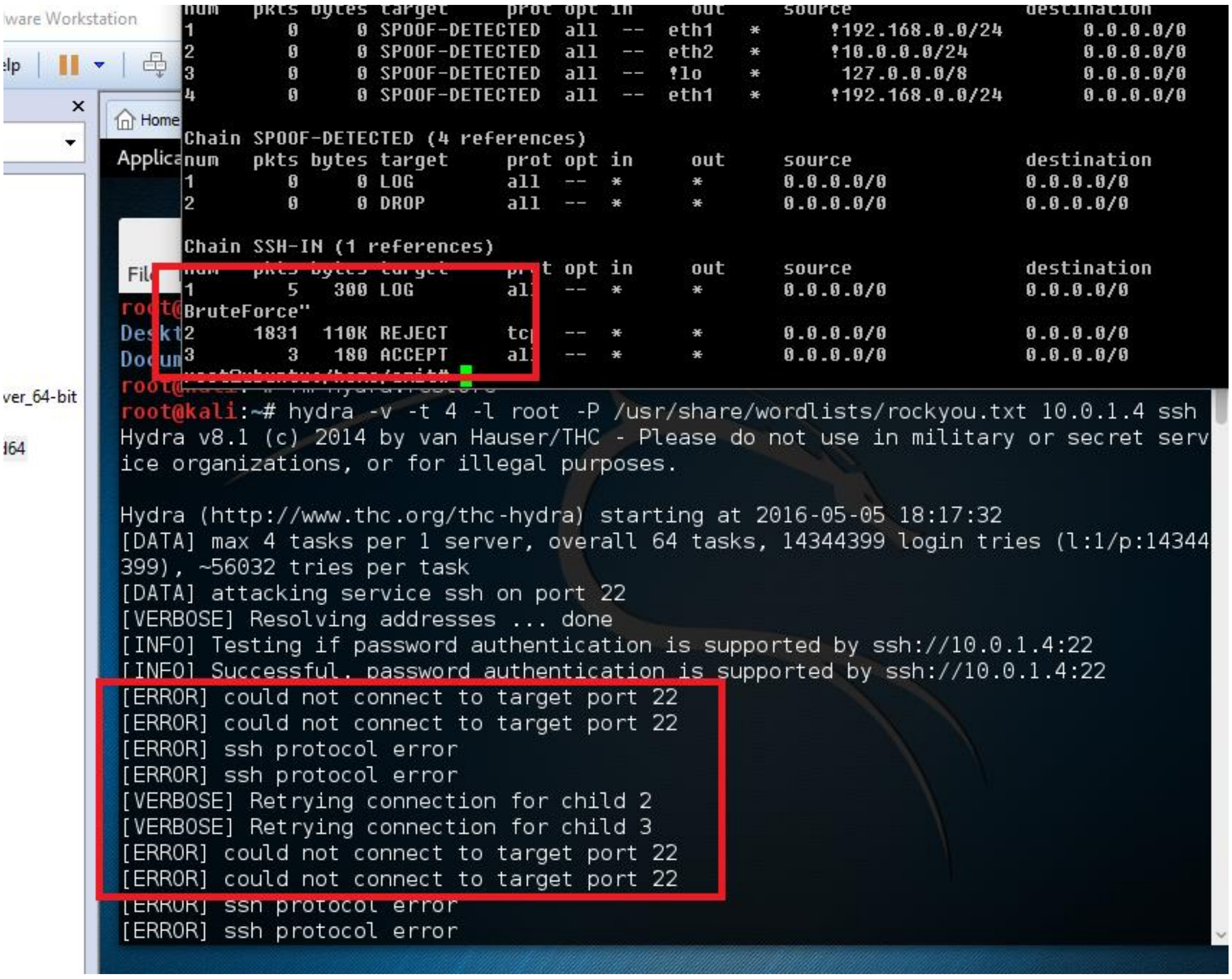
Scan detected port 22 as open (even though there is a very "hardcore" security measure protecting it), however could not identify the OpenSSH banner on that port.
Despite a very aggressive OS scan, nmap could not identify the OS because the iptables changed too many default replies. It is important to note that despite the very aggressive scan - nmap did not notice there is even a firewall in-place and did not identify any ports as filtered, even though effectively they all were filtered.

## 6.2. Attempting SSH Brute-Force With HYDRA

Run hydra verbose, with 4 threads on rockyou password list.

***hydra -v -t 4 -l root -P /usr/share/wordlists/rockyou.txt 10.0.1.4 ssh***

hydra ran for only a few seconds with over a thousand login attempts. Only 3 of these attempts (the first 3) actually passed the firewall - all the rest were filtered.



```
num   pkts bytes target     prot opt in     out     source               destination
1        0     0 SPOOF-DETECTED  all  --  eth1   *      !192.168.0.0/24       0.0.0.0/0
2        0     0 SPOOF-DETECTED  all  --  eth2   *      !10.0.0.0/24          0.0.0.0/0
3        0     0 SPOOF-DETECTED  all  --  !lo    *       127.0.0.0/8          0.0.0.0/0
4        0     0 SPOOF-DETECTED  all  --  eth1   *      !192.168.0.0/24       0.0.0.0/0

Chain SPOOF-DETECTED (4 references)
num   pkts bytes target     prot opt in     out     source               destination
1        0     0 LOG        all  --  *      *       0.0.0.0/0            0.0.0.0/0
2        0     0 DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain SSH-IN (1 references)
num   pkts bytes target     prot opt in     out     source               destination
1        5   300 LOG        all  --  *      *       0.0.0.0/0            0.0.0.0/0
BruteForce"
2     1831  110K REJECT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0
3        3   180 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0
```

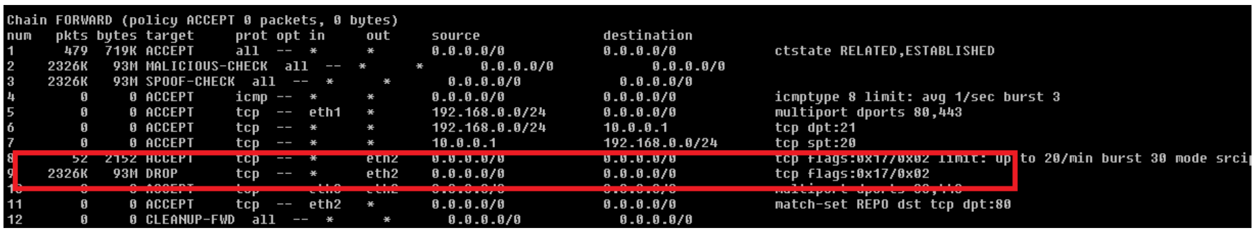
```
root@kali:~# hydra -v -t 4 -l root -P /usr/share/wordlists/rockyou.txt 10.0.1.4 ssh
Hydra v8.1 (c) 2014 by van Hauser/THC - Please do not use in military or secret serv
ice organizations, or for illegal purposes.

Hydra (http://www.thc.org/thc-hydra) starting at 2016-05-05 18:17:32
[DATA] max 4 tasks per 1 server, overall 64 tasks, 14344399 login tries (l:1/p:14344
399), ~56032 tries per task
[DATA] attacking service ssh on port 22
[VERBOSE] Resolving addresses ... done
[INFO] Testing if password authentication is supported by ssh://10.0.1.4:22
[INFO] Successful, password authentication is supported by ssh://10.0.1.4:22
[ERROR] could not connect to target port 22
[ERROR] could not connect to target port 22
[ERROR] ssh protocol error
[ERROR] ssh protocol error
[VERBOSE] Retrying connection for child 2
[VERBOSE] Retrying connection for child 3
[ERROR] could not connect to target port 22
[ERROR] could not connect to target port 22
[ERROR] ssh protocol error
[ERROR] ssh protocol error
```

## 6.3. Attempting Syn-Flood On DMZ (Bee-Box) with HPING3

***hping3 -S --flood -V 10.0.1.12***

Attempted flooding for about a minute. During this time I could still surf Bee-Box web interface. Not even a hiccup. Printscreen of the iptables match counters shows that all syn flood packets were identified and dropped by the iptables:



```
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination
1      479  719K ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0            ctstate RELATED,ESTABLISHED
2    2326K   93M MALICIOUS-CHECK  all  --  *      *       0.0.0.0/0            0.0.0.0/0
3    2326K   93M SPOOF-CHECK  all  --  *      *       0.0.0.0/0            0.0.0.0/0
4        0     0 ACCEPT     icmp --  *      *       0.0.0.0/0            0.0.0.0/0            icmptype 8 limit: avg 1/sec burst 3
5        0     0 ACCEPT     tcp  --  eth1   *       192.168.0.0/24       0.0.0.0/0            multiport dports 80,443
6        0     0 ACCEPT     tcp  --  *      *       192.168.0.0/24       10.0.0.1             tcp dpt:21
7        0     0 ACCEPT     tcp  --  *      *       10.0.0.1             192.168.0.0/24       tcp spt:20
8       52  2152 ACCEPT     tcp  --  *      eth2    0.0.0.0/0            0.0.0.0/0            tcp flags:0x17/0x02 limit: up to 20/min burst 30 mode srci
9    2326K   93M DROP       tcp  --  *      eth2    0.0.0.0/0            0.0.0.0/0            tcp flags:0x17/0x02
10       0     0 ACCEPT     tcp  --  eth0   eth2    0.0.0.0/0            0.0.0.0/0            multiport dports 80,443
11       0     0 ACCEPT     tcp  --  eth2   *       0.0.0.0/0            0.0.0.0/0            match-set REPO dst tcp dpt:80
12       0     0 CLEANUP-FWD  all  --  *      *       0.0.0.0/0            0.0.0.0/0
```

by Amit Blum
May 2016