On Stateful Firewalls

or Bob’s story

Lorenzo Angeli, Liviu Bogdan, Bertalan Borsos

Recap - Firewalls

- Firewalls are Layer-3 entities
- They are used to filter traffic going through networks
- Filters can be set up for multiple factors
  - IPs
  - Ports
  - Flags
  - Packet content
  - Amount of traffic
  - ...

Recap - Stateless vs Stateful

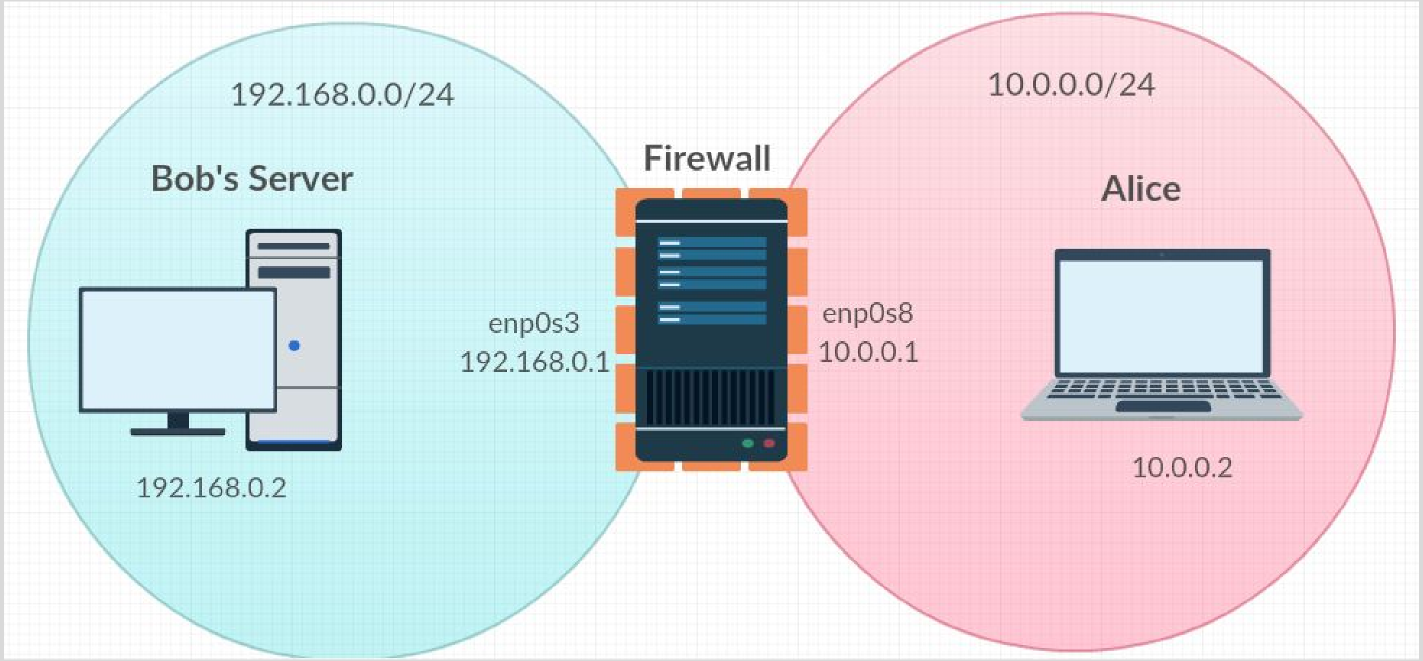
Stateless firewalls…

- Are simpler, easier to configure
- Look at packets *one at a time*, independent of context
- Only look at headers
- Generally perform faster

Stateful firewalls…

- Are less immediate to configure
- Examine packets *keeping track of connection history*
- Can also check *content*

Lab Architecture



Usernames and passwords are always `netsec`

Outline

- First, we’ll spin a masterful tale to give you the setting
- Then, we’ll give you some scenarios that you can solve with a *stateless* configuration
- Hopefully, these will show the limitations of that model
- Which will lead us to *stateful* extensions

Context and tools

- In this lab, we'll only provide guidance
- We'll leave you time to actually think about the problems on your own

You have these tools:

- `wireshark` - Use this to monitor network traffic
- Web browser - To check availability of the website
- `iptables` - Your friendly neighborhood firewall

The Scenario

- Bob works at a particle accelerator
- Bob has a web server on his workstation (not good)
- Bob’s favourite thing in the universe is his cat
- He decided to set up a website for his cat
- …but he also got a virus
- …and now he’s on vacation

The Scenario - part 2

- You are the the sysadmin
- You do not have access to Bob’s personal computer
- But you still have to stop any information leakage

The current situation

- You realized with WireShark that Bob’s computer is infected by `mal.py`
- You want to stop `mal.py` from leaking Bob’s secret data
- But you don’t want to call Bob, so you can only operate on the firewall
- Exception: For convenience, you’ll have to launch `mal.py`

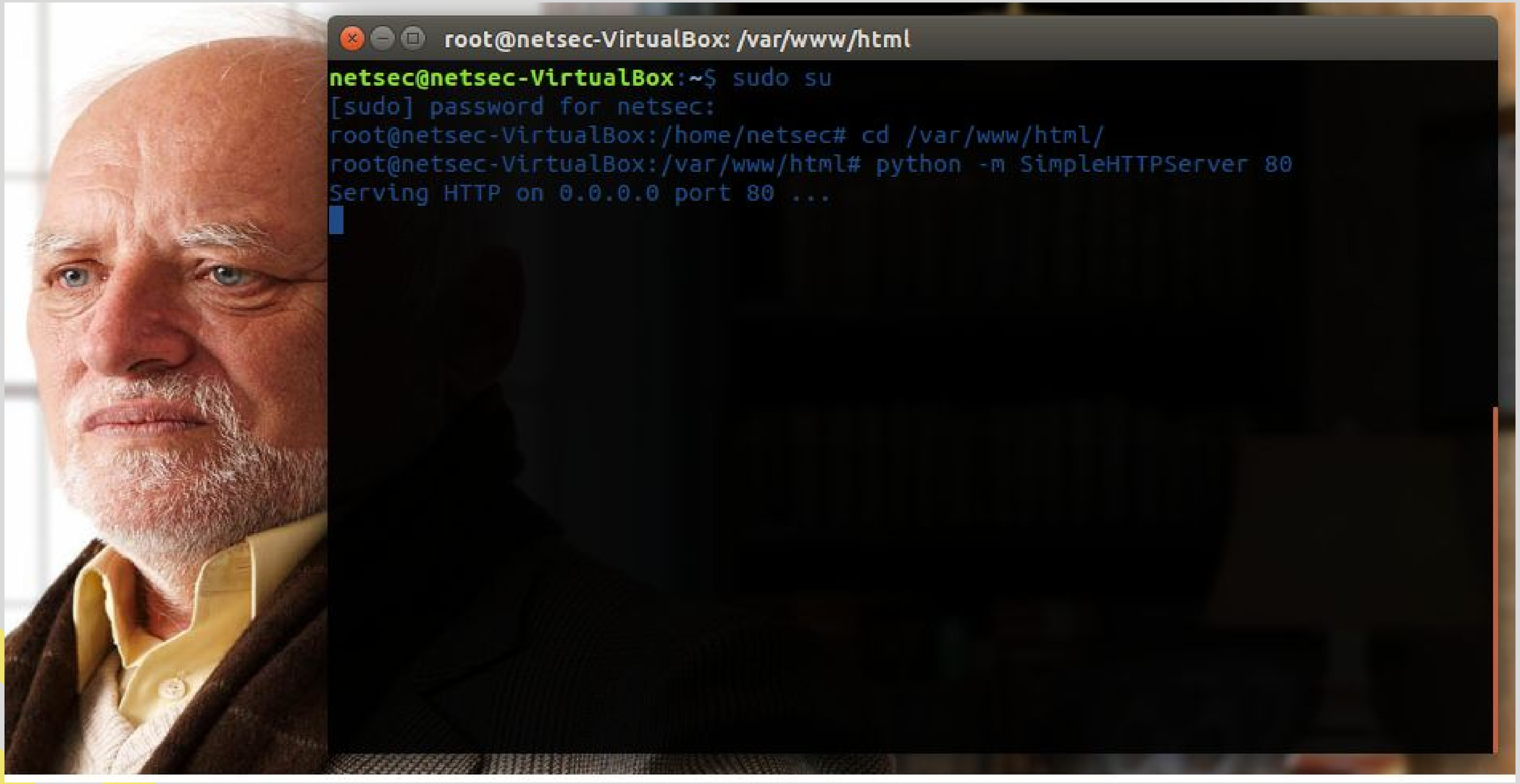
`mal.py`

- Behaves like a simplified Trojan Horse
- “Leaks” a TCP packet with information
- It will “evolve” under certain circumstances
  - To simulate this, you’ll launch different versions of `mal.py`
- Your task is, of course, to stop the data leakage

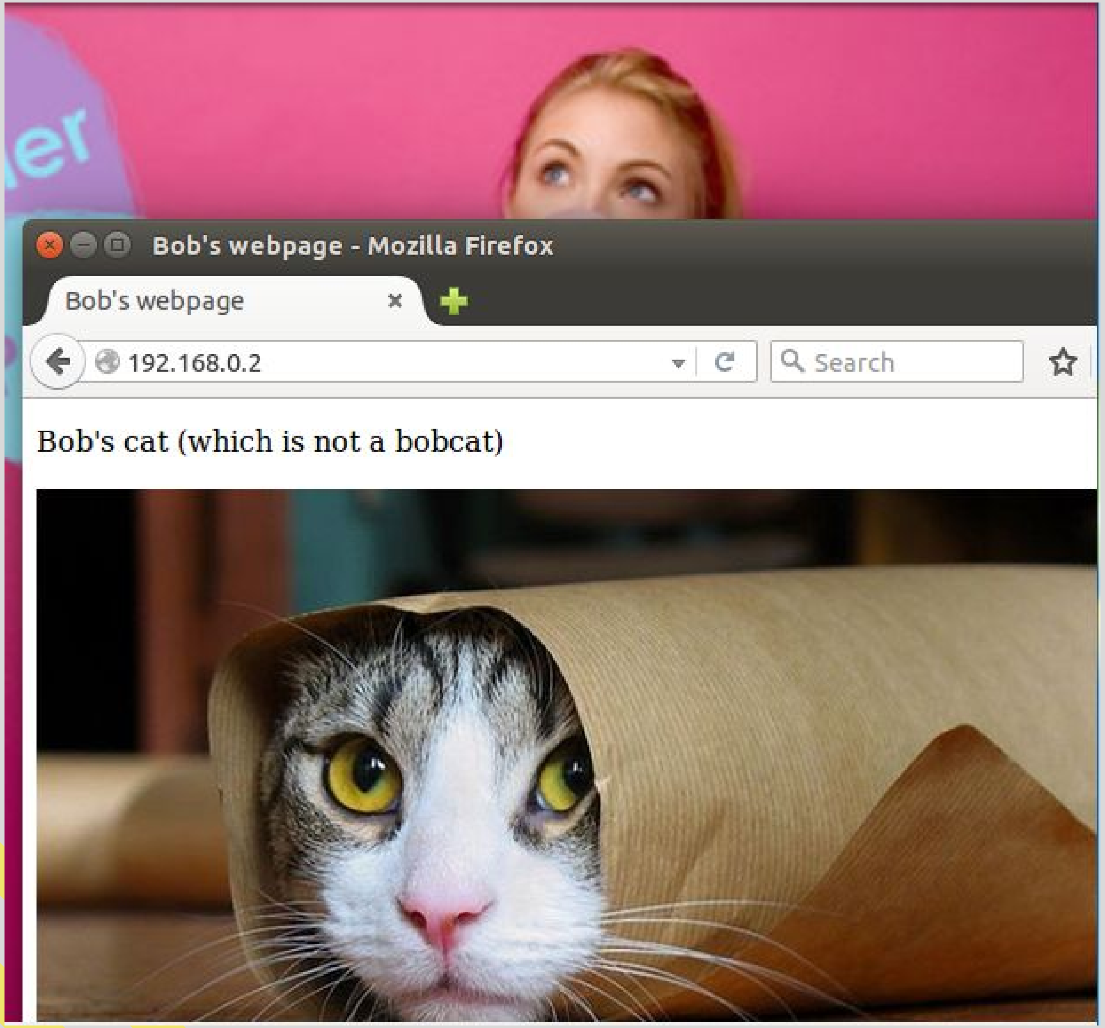
One more setup step...

- To launch the python SimpleHTTPServer...
  - Open a terminal
  - `sudo su -`
    - (password is “netsec”)
  - `cd /var/www/html`
  - `python -m SimpleHTTPServer 80`
  - Keep that terminal open!

HTTP server is now running on port 80


root@netsec-VirtualBox: /var/www/html

```
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 ...
```

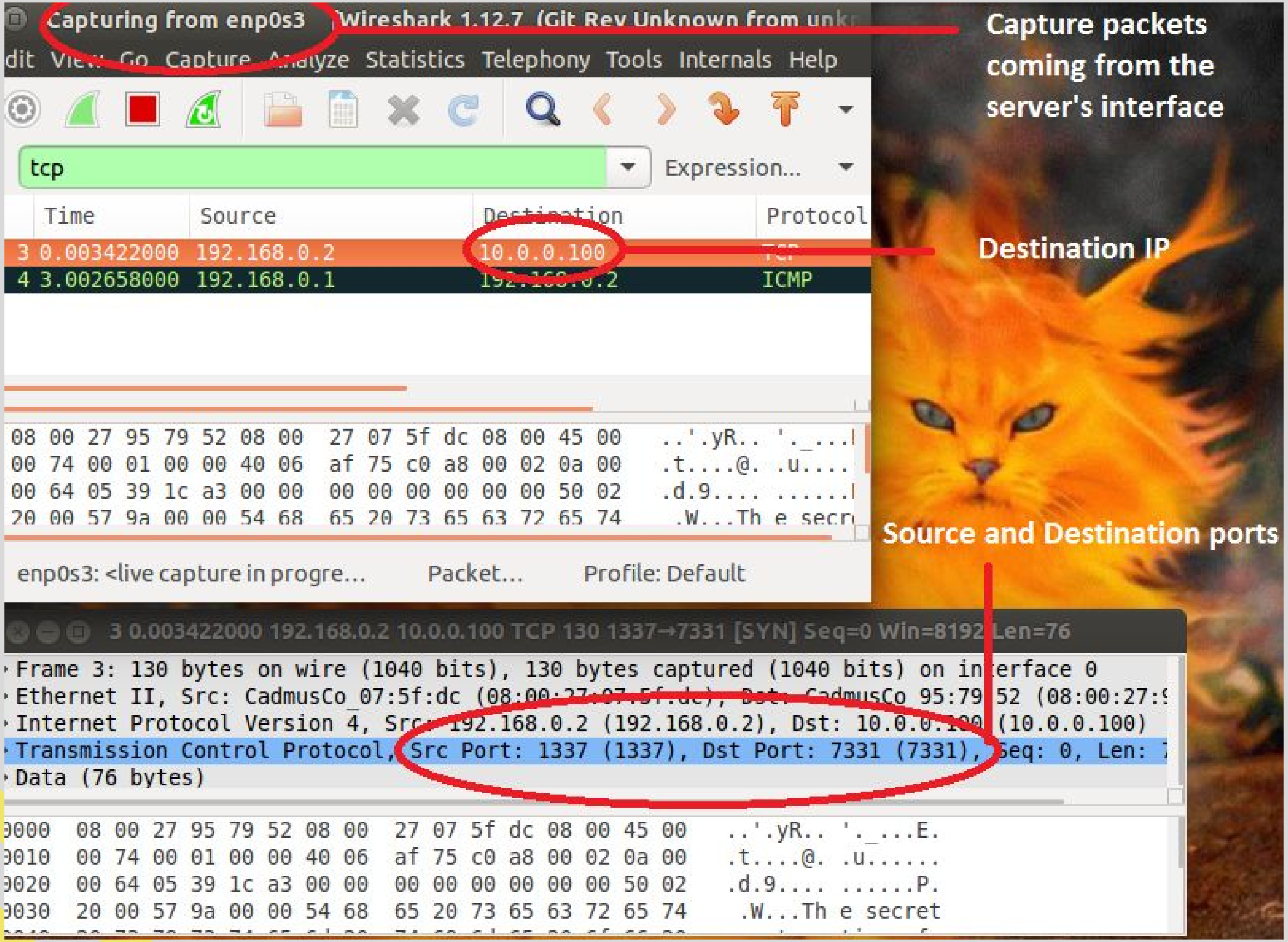


Bob's cat (which is not a bobcat)

We try to test the server

Now, get a clearer picture

- Launch `mal1.py` with Wireshark open
- ...ideas on how to block it?



Capture packets coming from the server's interface

Destination IP

Source and Destination ports

```
Capturing from enp0s3   Wireshark 1.12.7 (Git Rev Unknown from unk
dit View Go Capture Analyze Statistics Telephony Tools Internals Help
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
```

1 - Filter by IP

- As a first solution, you may think of blocking the *target* IP address
- `iptables` in *stateless* mode can do this
- …check your cheat sheet if you need help with the syntax

First task

- Use `iptables` to filter *outbound* packets that go to the attacker
- Verify that the firewall is actually blocking the traffic
- Check that Bob’s website is still available

...okay, that’s a first step

```
iptables -A FORWARD -d 10.x.y.z -j DROP
```

```
“Drop all packets that the firewall would forward to destination 10.x.y.z”
```

- You blocked the IP - good
- ...but try running it again (`mal1.py`)

Notice

- …it’s randomizing IPs!
- Is it sensible to block all IPs?

The answer

No.

Okay, next!

Moving on...

- For the sake of convenience, flush `iptables`
  - Check your friendly cheat sheet if you forgot how to do that
- What else can we do?

2 - Filter by port

- You think a bit more...
- This time, you may want to filter traffic by port
- ...but remember **not** to block port 80
- Again, `iptables` in stateless mode can do it

Second task

- Run `mal1.py`
- Use `iptables` to block the port `mal.py` is using to leak info
- Again, check that you blocked the trojan
- Again, check for website availability

Operation successful!

```
iptables -A FORWARD -p tcp --sport # --dport # -j DROP
```

“Drop all packets that the firewall would forward from a specific port to a specific port”

- The website is accessible!
- The malware is blocked!

The first “evolution”

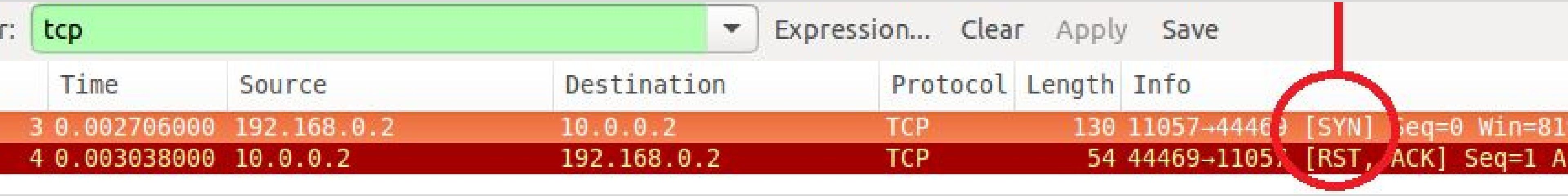
- Run `mal2.py`
- Check Wireshark - something changed

Next..?

- Again, remember to flush `iptables`

3 - Filter by flags

- Notice that `mal.py` is always sending **SYN**s
- So you might want to block *flags* instead
- `iptables` can still do it statelessly



```
   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=81
 4 0.003038000 10.0.0.2      192.168.0.2   TCP          54 44469→11057 [RST, ACK] Seq=1 A
```

Third task

- Use `iptables` to filter outgoing **SYN**s
- For the record: can you do this safely?
- Verify you blocked the trojan
- Check Website availability

Aha!

```
iptables -A FORWARD -i enp0s3 -p tcp --tcp-flags ALL SYN -
j DROP
```

“Of TCP packets forwarded by the firewall through interface enp0s3, inspect them all and drop those that are flagged with exactly SYN”

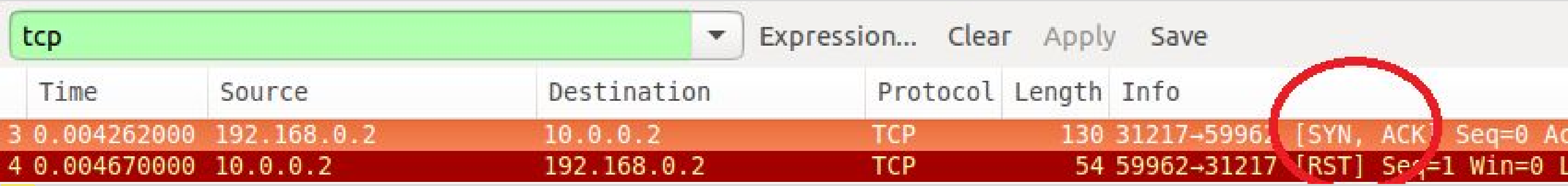
- Okay, that’s all good
- …but you know what’s about to happen, right?

Another “evolution”

- Now, launch `mal3.py`
- With more careful WireShark inspection you should notice something, though…

Another “evolution”

- Now, launch `mal3.py`
- With more careful WireShark inspection you should notice something, though…
- Now the Trojan sends **SYN_ACK**s!



tcp | Expression... Clear Apply Save

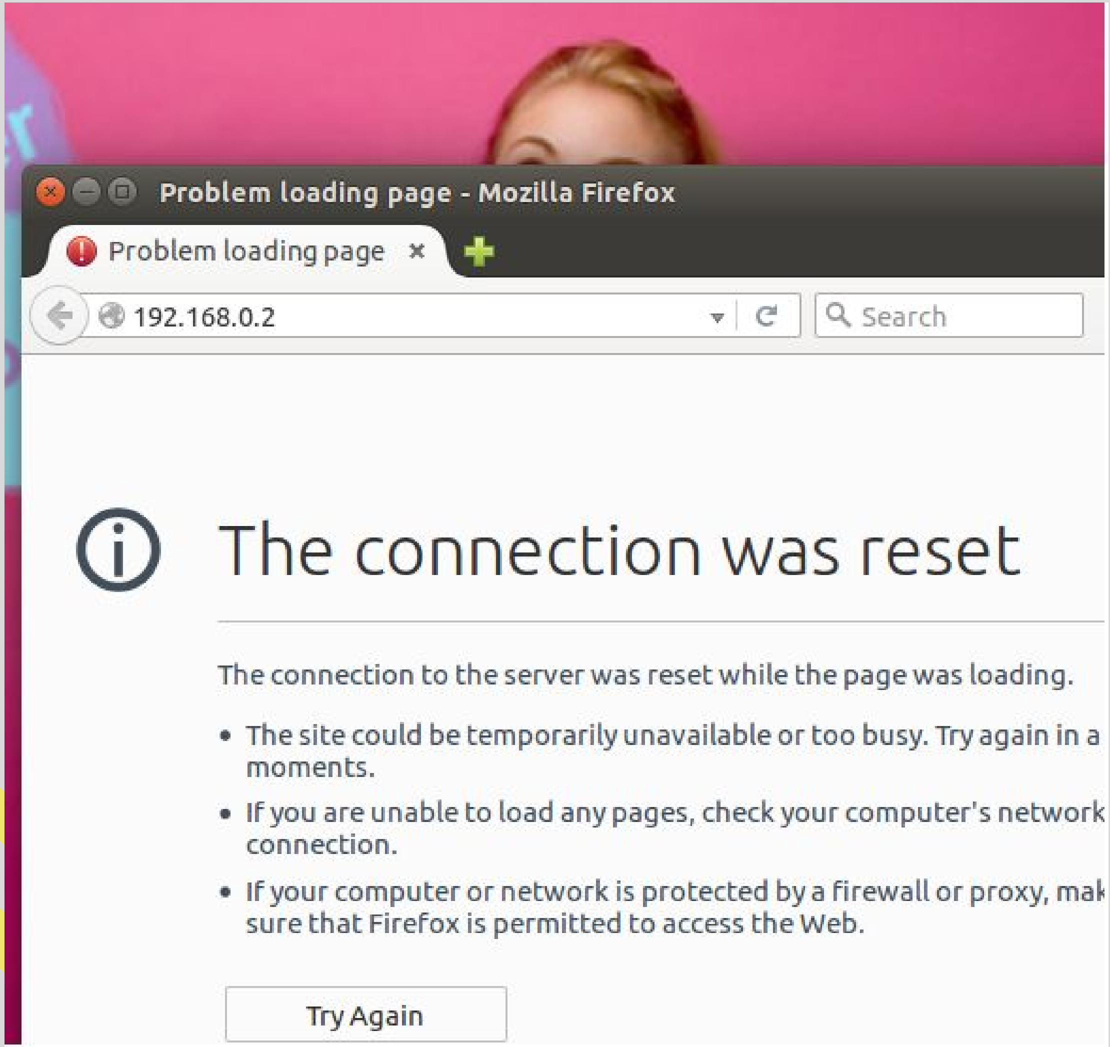
```
  Time         Source       Destination   Protocol Length Info
3 0.004262000  192.168.0.2  10.0.0.2      TCP         130 31217→59962 [SYN, ACK] Seq=0 Ac
4 0.004670000  10.0.0.2     192.168.0.2   TCP          54 59962→31217 [RST] Seq=1 Win=0 L
```

When the going gets tough...

- ...can you block those?
- As usual, flush `iptables`
  - You know how to do this by now, right?
- What could we do?

Okay, good job

- You probably guessed that we could do stateful filtering…
- But you don’t want to eat your dessert before you’re finished with the rest, right?



Problem loading page - Mozilla Firefox

Problem loading page

192.168.0.2

The connection was reset

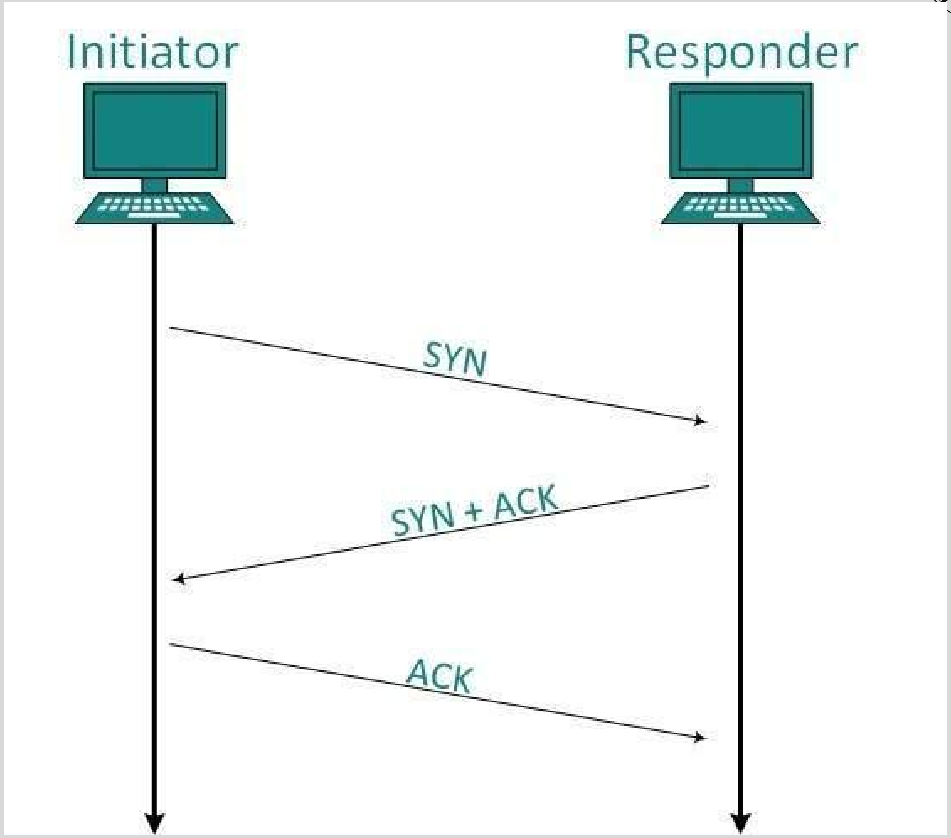
The connection to the server was reset while the page was loading.

- 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.

Server is no longer working

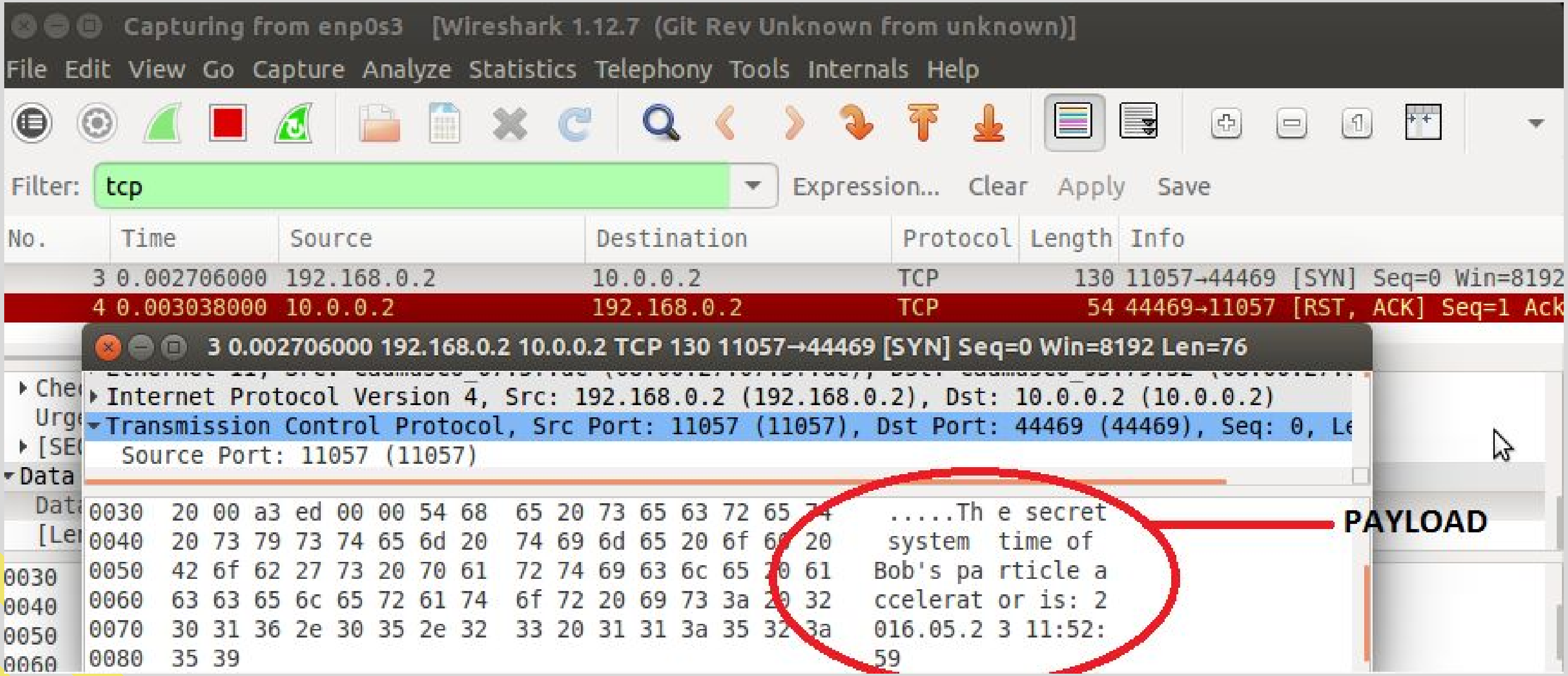
Why?

3-Way Handshake



So… Any more ideas?

– Hint: `mal.py` is sending out *secret* messages…



```
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
```

4 - Deep inspection

- This time, we inspect the packet, not just the header
- `iptables` has a module that can match packet content!

Fourth task

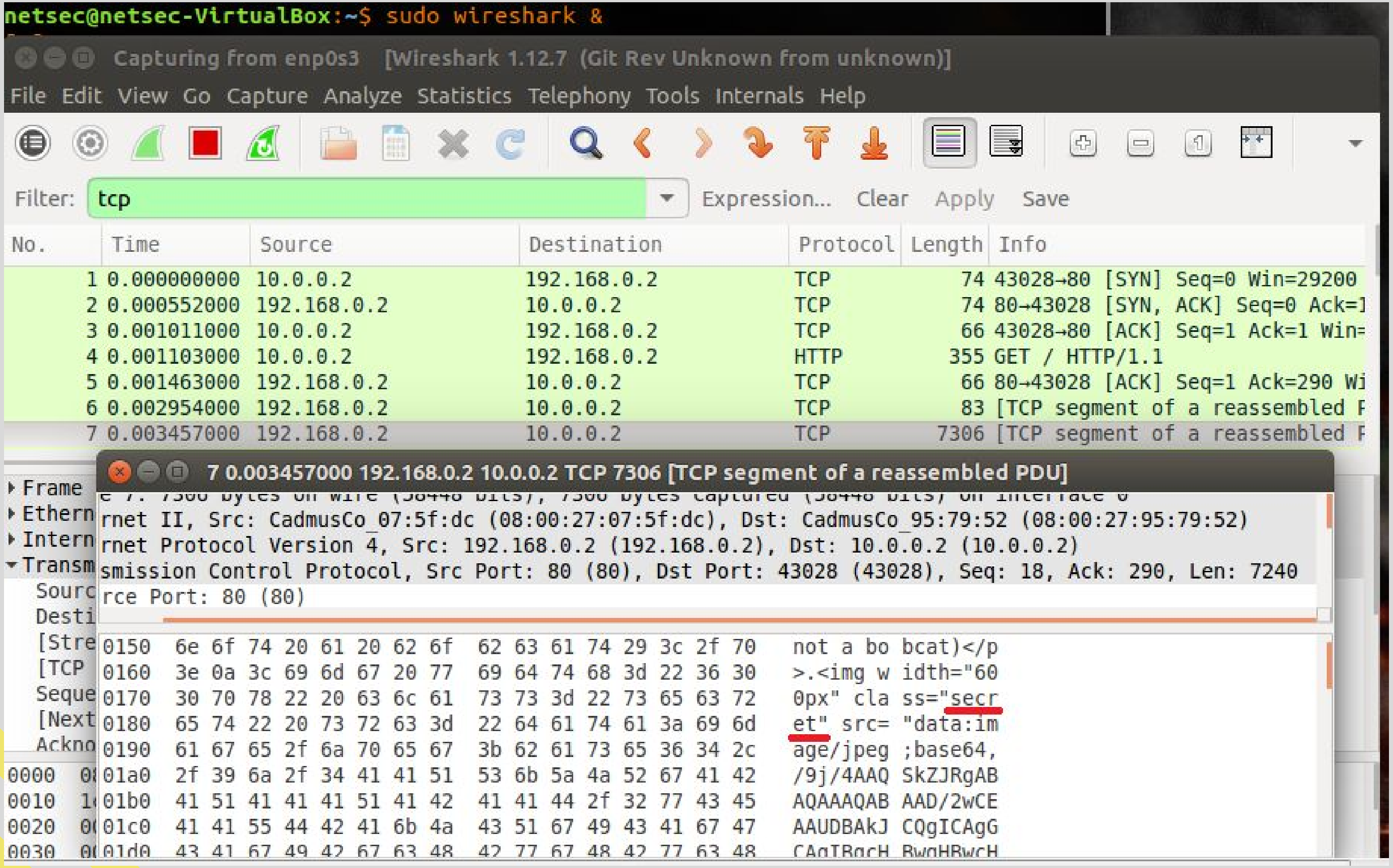
- Block packets that contain the word *secret* in them
- As usual, refer to the syntax cheat sheet
- Check that the attack is blocked
- And that the website is working

What just happened here?

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

 “Take forwarded packet, and deeply inspect the text content. If it contains the substring *secret* (matched with Boyer-Moore), drop it”

- The filter is correct
- …but something was in the HTML code


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

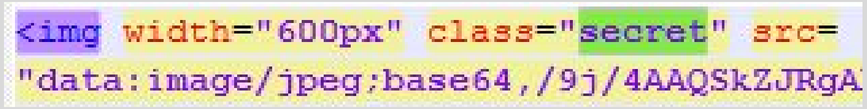
```
netsec@netsec-VirtualBox:~$ sudo wireshark &
```

```
No.    Time         Source        Destination   Protocol Length Info
     1 0.000000000 10.0.0.2      192.168.0.2   TCP         74 43028→80 [SYN] Seq=0 Win=29200
     2 0.000552000 192.168.0.2   10.0.0.2      TCP         74 80→43028 [SYN, ACK] Seq=0 Ack=1
     3 0.001011000 10.0.0.2      192.168.0.2   TCP         66 43028→80 [ACK] Seq=1 Ack=1 Win=
     4 0.001103000 10.0.0.2      192.168.0.2   HTTP       355 GET / HTTP/1.1
     5 0.001463000 192.168.0.2   10.0.0.2      TCP         66 80→43028 [ACK] Seq=1 Ack=290 Wi
     6 0.002954000 192.168.0.2   10.0.0.2      TCP         83 [TCP segment of a reassembled P
     7 0.003457000 192.168.0.2   10.0.0.2      TCP       7306 [TCP segment of a reassembled P
```

```
7 0.003457000 192.168.0.2 10.0.0.2 TCP 7306 [TCP segment of a reassembled PDU]

▸ Frame    e 7: 7306 bytes on wire (58448 bits), 7306 bytes captured (58448 bits) on interface 0
▸ Ethern   rnet II, Src: CadmusCo_07:5f:dc (08:00:27:07:5f:dc), Dst: CadmusCo_95:79:52 (08:00:27:95:79:52)
▸ Intern   rnet Protocol Version 4, Src: 192.168.0.2 (192.168.0.2), Dst: 10.0.0.2 (10.0.0.2)
▾ Transm   smission Control Protocol, Src Port: 80 (80), Dst Port: 43028 (43028), Seq: 18, Ack: 290, Len: 7240
    Sourc  rce Port: 80 (80)
    Desti
    [Stre
    [TCP
    Seque
    [Next
    Ackno

0000  08 0150  6e 6f 74 20 61 20 62 6f  62 63 61 74 29 3c 2f 70   not a bo bcat)</p
0010  1( 0160  3e 0a 3c 69 6d 67 20 77  69 64 74 68 3d 22 36 30   >.<img w idth="60
0020  00 0170  30 70 78 22 20 63 6c 61  73 73 3d 22 73 65 63 72   0px" cla ss="secr
0030  00 0180  65 74 22 20 73 72 63 3d  22 64 61 74 61 3a 69 6d   et" src= "data:im
         0190  61 67 65 2f 6a 70 65 67  3b 62 61 73 65 36 34 2c   age/jpeg ;base64,
         01a0  2f 39 6a 2f 34 41 41 51  53 6b 5a 4a 52 67 41 42   /9j/4AAQ SkZJRgAB
         01b0  41 51 41 41 41 51 41 42  41 41 44 2f 32 77 43 45   AQAAAQAB AAD/2wCE
         01c0  41 41 55 44 42 41 6b 4a  43 51 67 49 43 41 67 47   AAUDBAkJ CQgICAgG
         01d0  43 41 67 49 42 67 63 48  42 77 67 48 42 77 63 48   CAgIBgcH BwgHBwcH
```

Here’s the culprit

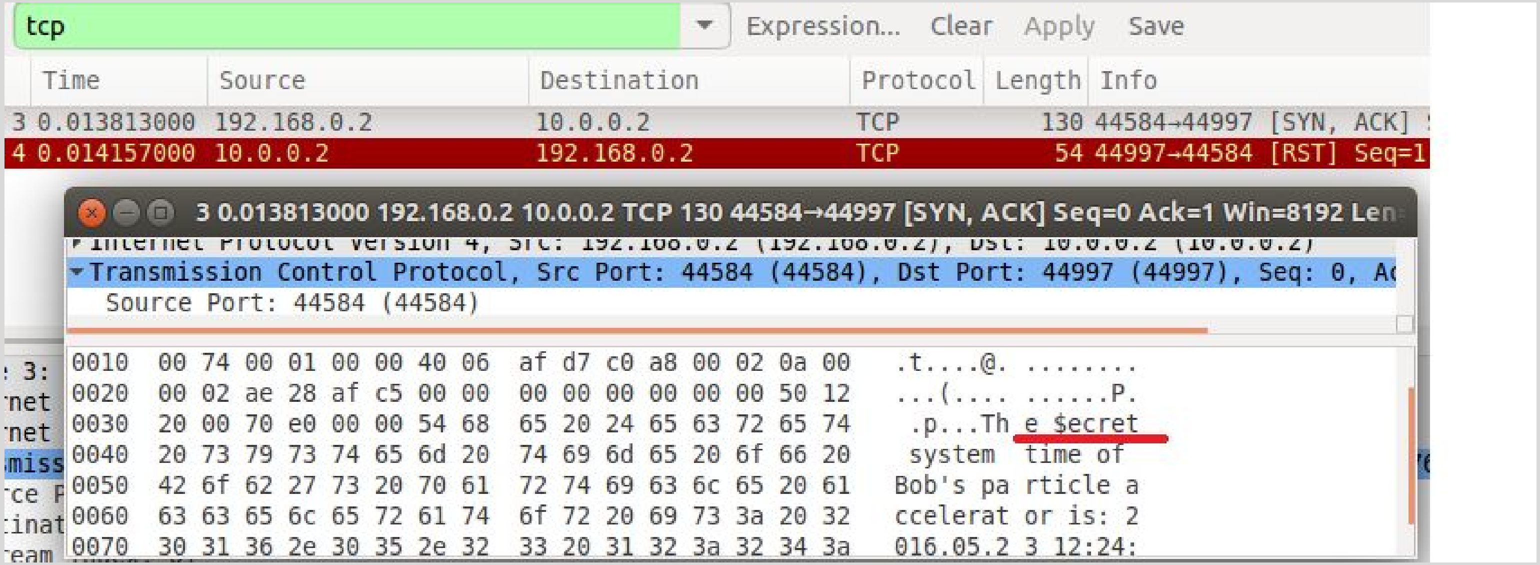
- A “secret” class in the HTML code means that packet will be blocked
- Which, in turns, breaks the page
- How?



```
<img width="600px" class="secret" src=
"data:image/jpeg;base64,/9j/4AAQSkZJRgA
```

Time to evolve again!

- Okay, time to launch `mal4.py`
- It’s not sending *secret* anymore
- So, what are we doing?



tcp   Expression... Clear Apply Save

```
  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

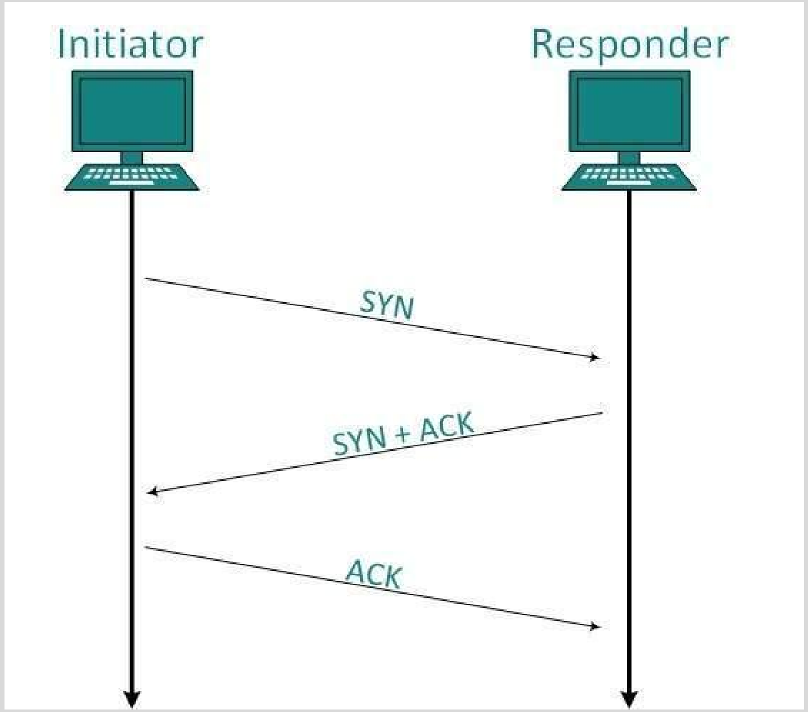
```
▸ Internet Protocol Version 4, Src: 192.168.0.2 (192.168.0.2), Dst: 10.0.0.2 (10.0.0.2)
▾ Transmission Control Protocol, Src Port: 44584 (44584), Dst Port: 44997 (44997), Seq: 0, Ac
    Source Port: 44584 (44584)

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:
```

At long last

- We can track the connection *statefully*
- This way, we can filter out *unsolicited* **SYN_ACK**s

5 - Stateful Filtering

- Well, we can now say that the server will allow only *established* connections



Fifth (and final?) task

- Add a stateful rule to `iptables`: only allow *established* connections

We did it?

```
iptables -A FORWARD -m state --state INVALID -j DROP
```

“Inspect forwarded packets by state. Keep only those that are valid connections”

- Website is accessible
- The Trojan isn’t leaking any more information
- Moving on to some recap...

Reflection and recap

What happened today…

1. Stateless filtering by IP
   - You can’t really block possibly legitimate IPs
2. Stateless filtering by port only
   - Malicious traffic could piggyback on legitimate traffic ports
3. Stateless filtering by flags
   - Can still be worked around
4. Deep packet inspection
   - There might be too many sub-cases, might filter legit content
5. Stateful filtering
   - Stops packets with “illogical” flags

Thanks for joining!