

# MAC Address Table Overflow

Chatziefrimidis, Lefteris  
echatzie@inf.uth.gr

Evangelou, Sotiris  
sevagelou@inf.uth.gr

LastName2, Spiros  
first2.last2@xxxxx.com

December 25, 2019

## 1 Introduction - Description of the attack

MAC address flooding attack is very common security attack. MAC address table in the switch has the MAC addresses available on a given physical port of a switch and the associated VLAN parameters for each.

This attacks are sometimes called MAC address table overflow attacks. To understand the mechanism of a MAC address table overflow attack we must recall how does a switch work in the first place.

### 1.1 Switch before attack

When switch receives a frame, it looks in the MAC address table (sometimes called CAM table) for the destination MAC address. When frames arrive on switch ports, the source MAC addresses are learned from Layer 2 packet header and recorded in the MAC address table. If the switch has already learned the MAC address of the computer connected to his particular port then an entry exists for the MAC address. In this case the switch forwards the frame to the MAC address port designated in the MAC address table. If the MAC address does not exist, the switch acts like a hub and forwards the frame out every other port on the switch while learning the MAC for next time.

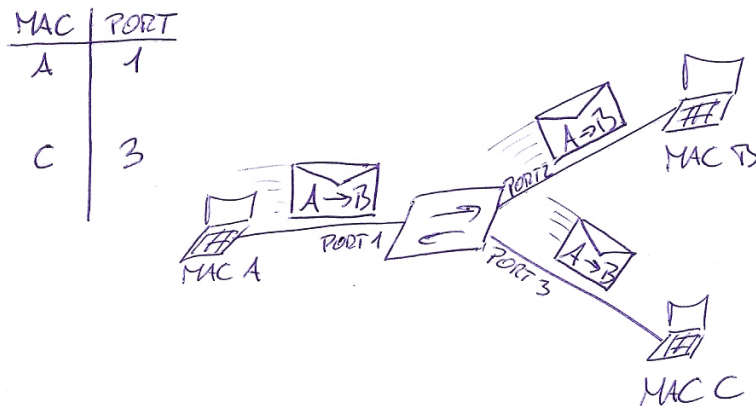


Figure 1: Switch acts as hub with empty mac address table

Computer A sends traffic to computer B. The switch receives the frames and looks up the destination MAC address in its MAC address table. If the switch does not have the destination MAC in the MAC address table, the switch then copies the frame and sends it out every switch port like a broadcast. This means that not only PC B receives the frame, PC C also receives the frame from host A to host B, but because the destination MAC address of that frame is host B, host C drops that frame.

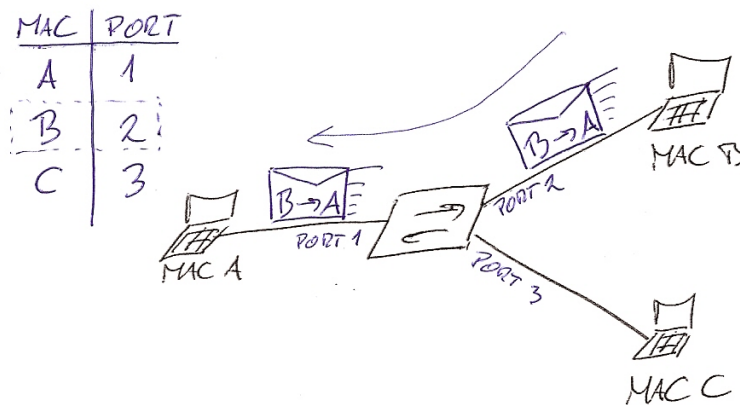


Figure 2: Switch learns mac address from source MAC address in the layer 2 headers from frames – switch is populating his mac table

## 1.2 Normal switch function

PC B receives the frame and sends a reply to PC A. The switch then learns that the MAC address for PC B is located on port 2 and writes that information into the MAC address table. From now on any frame sent by host A (or any other host) to host B is forwarded to port 2 of the switch and not broadcast out every port. The switch is working like it should. This is the main goal of switch functionality, to have separate collision domain for each port on the switch.

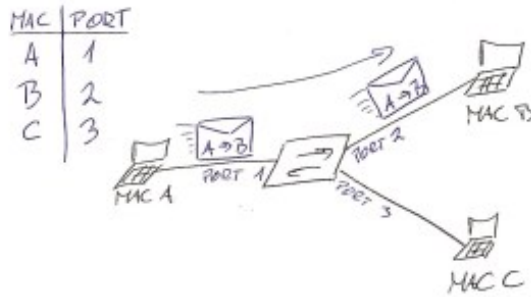


Figure 3: When the switch learns about all MAC addresses on his different ports switch acts like switch – mac address table complete

## 1.3 Attack

But this is where the attacker is coming into play. The key to understanding how MAC address table overflow attacks work is to know that MAC address tables are limited in size. MAC flooding makes use of this limitation to send to the switch a whole bunch of fake source MAC addresses until the switch MAC address table is fully loaded and can not save any more MAC address – Port mapping entries. The switch then enters into a fail-open mode that means that it starts acting as a hub. In this situation switch will broadcast all received packets to all the machines on the network. As a result, the attacker (in our case “PC C”) can see all the frames sent from a victim host to another host without a MAC address table entry.

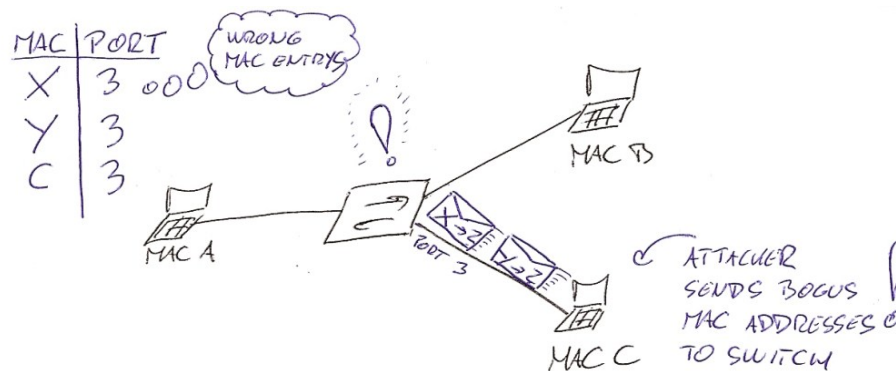


Figure 4: Switch Mac flooding attack will populate the entire mac address table with bogus mac addresses

In this case, an attacker will use legitimate tools for malicious actions. The figure shows how an attacker can use the normal operating characteristics of the switch to stop the switch from operating.

Let's get into more detail about filling up the MAC address table. To do this attacker uses network attack tools for MAC. The network intruder uses the attack tool to flood the switch with a large number of invalid source MAC addresses until the MAC address table fills up. When the MAC address table is full, the switch floods all ports with incoming traffic because it cannot find the port number for a particular MAC address in the MAC address table. The switch, in essence, acts like a hub. In this lab we will reproduce the above example in order to understand how this attack works.

## 2 Environment Setup

Firstly, we open up a terminal and type the below command:

### 2.1 System update

```
$ sudo apt update && sudo apt upgrade
```

After that our system is already updated and set in order to download the packages that we need for this lab. We will use a GitHub repository to retrieve the source code that we will use. So we clone the repository locally.

### 2.2 Install Python and Clone the repo

```
$ sudo apt install git && git clone https://github.com/echatzief/MAC_Address_Overflow
$ sudo apt install python
$ sudo apt install python-pip
```

## 2.3 Install prerequisites

```
$ cd MAC_Address_Overflow/  
$ sudo apt-get update  
$ sudo apt-get install -y git vim-nox python-setuptools python-all-dev flex bison traceron  
$ pip install impacket
```

## 2.4 Install Mininet

```
$ cd mininet  
$ ./util/install.sh -fnv  
$ sudo apt-get install mininet  
$ sudo apt-get install xterm
```

## 2.5 Install ltprotocol

```
$ cd .. && cd ltprotocol/  
$ pip install setuptools  
$ sudo python setup.py install
```

## 2.6 Link POX into the Directory

```
$ cd .. && cd lab  
$ rm pox && ln -s ../pox/
```

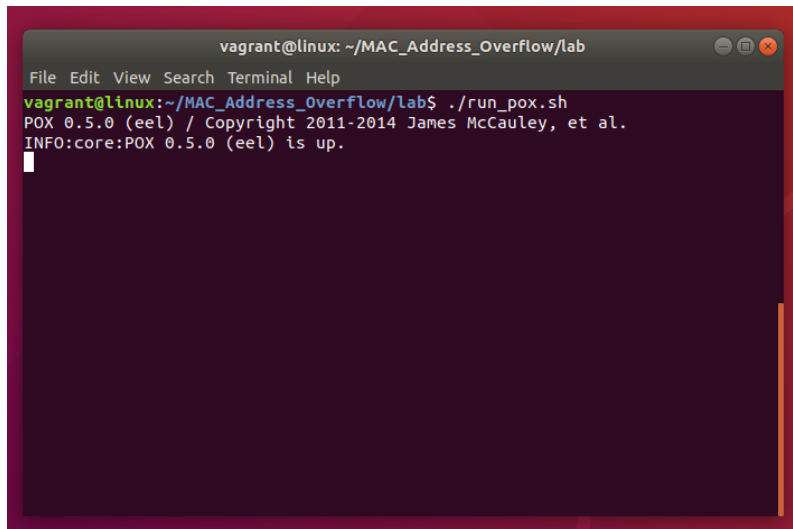
## 2.7 Configure the Environment

```
$ bash ./config.sh
```

# 3 Attack Demonstration

Firstly we launch a terminal in order to start the POX network controller, which will emulate the behavior of a L2 learning switch.

```
$ ./run_pox.sh
```

A terminal window titled 'vagrant@linux: ~/MAC\_Address\_Overflow/lab' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the command 'vagrant@linux:~/MAC\_Address\_Overflow/lab\$ ./run\_pox.sh' and its output: 'POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.' and 'INFO:core:POX 0.5.0 (eel) is up.' followed by a cursor.

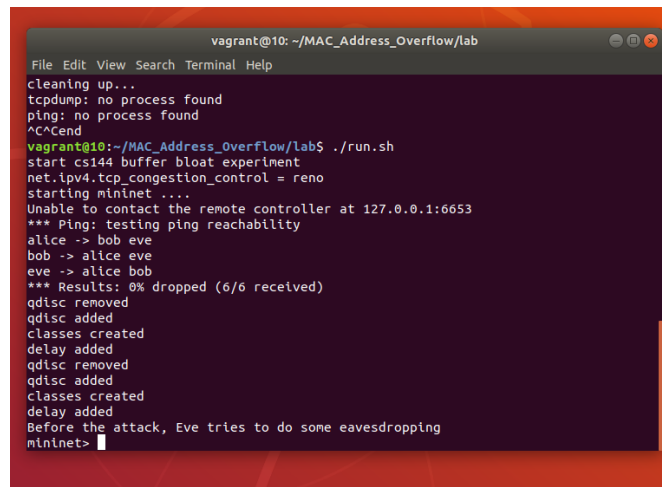
```
vagrant@linux: ~/MAC_Address_Overflow/lab
File Edit View Search Terminal Help
vagrant@linux:~/MAC_Address_Overflow/lab$ ./run_pox.sh
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.5.0 (eel) is up.
█
```

Figure 5: Start the pox

In another terminal (if you are using a remote machine, make sure the X-forwarding is enabled.):

```
$ ./run.sh
```

This will start the Mininet network emulator and there will be terminals pops up for each of the nodes in the network. Close the terminals for switches and controllers, but keep the terminals for Alice, Bob and Eve.

A terminal window titled 'vagrant@10: ~/MAC\_Address\_Overflow/lab' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the command 'vagrant@10:~/MAC\_Address\_Overflow/lab\$ ./run.sh' and its output, which includes cleanup messages, network setup for CS144, mininet startup, ping tests, and configuration of queue disciplines and classes for the network.

```
vagrant@10: ~/MAC_Address_Overflow/lab
File Edit View Search Terminal Help
cleaning up...
tcpdump: no process found
ping: no process found
^^^end
vagrant@10:~/MAC_Address_Overflow/lab$ ./run.sh
start cs144 buffer bloat experiment
net.ipv4.tcp_congestion_control = reno
starting mininet ....
Unable to contact the remote controller at 127.0.0.1:6653
*** Ping: testing ping reachability
alice -> bob eve
bob -> alice eve
eve -> alice bob
*** Results: 0% dropped (6/6 received)
qdisc removed
qdisc added
classes created
delay added
qdisc removed
qdisc added
classes created
delay added
Before the attack, Eve tries to do some eavesdropping
mininet> █
```

Figure 6: Start the run script

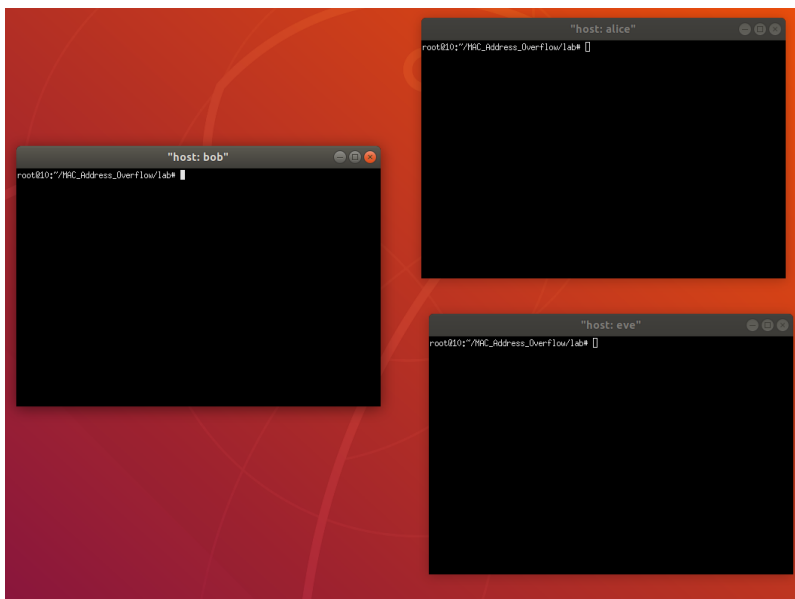


Figure 7: The terminals that spawn

### 3.1 In Alice's Terminal

Alice will now create some traffic by pinging Bob:

```
$ ping 10.0.0.2
```

You should be able to see some output like the following:

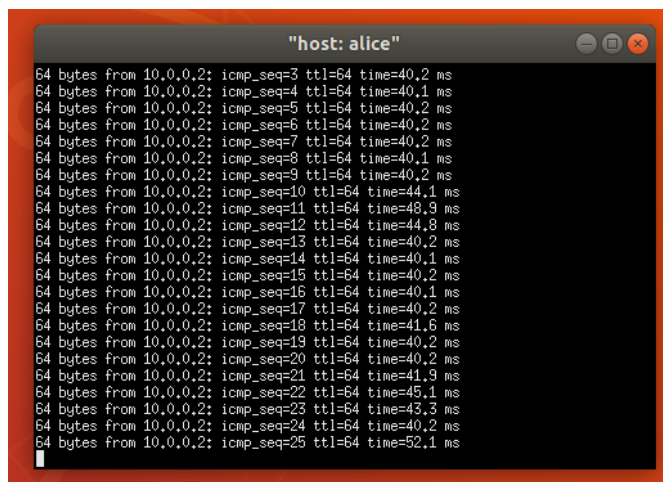
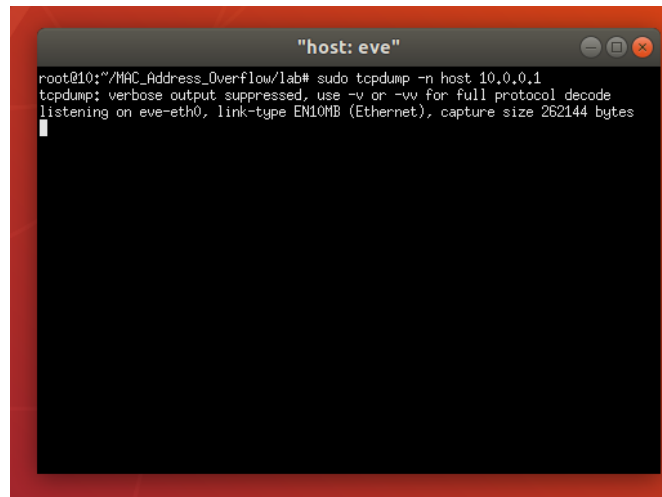


Figure 8: Alice pings Bob

### 3.2 In Eve's Terminal

We will now run tcpdump to eavesdrop the traffic between Alice (10.0.0.1) and Bob (10.0.0.2):

```
$ sudo tcpdump -n host 10.0.0.1
```



Since the switch between Alice/Bob/Eve already learned about the address of Alice and Bob, it will not broadcast the packet and therefore Eve will not be able to see the packets between Alice and Bob.

Now, we will let Eve generate some ethernet packets with randomly generated source MAC address to overflow switches' MAC address table. To do so, let's create another terminal in screen for Eve by (Ctrl-a + c), then run the following command in the new screen:

```
$ python attack.py
```



```
"Node: eve"
send packet # 8196 with src_mac: 0a:b0:e2:4c:32:64
send packet # 8197 with src_mac: 04:41:eb:3d:0b:7e
send packet # 8198 with src_mac: a9:d6:ab:10:57:de
send packet # 8199 with src_mac: 4b:70:67:ac:f1:13
send packet # 8200 with src_mac: 8c:99:d0:79:77:1e
send packet # 8201 with src_mac: 28:13:9a:5f:ca:64
send packet # 8202 with src_mac: 34:de:4a:56:fe:38
send packet # 8203 with src_mac: 7c:d1:6a:56:d1:db
send packet # 8204 with src_mac: 6e:8d:61:29:cc:e9
send packet # 8205 with src_mac: 95:da:3d:bc:b0:27
send packet # 8206 with src_mac: 1a:21:d5:3d:c4:f1
send packet # 8207 with src_mac: 1a:da:b3:d6:26:6b
send packet # 8208 with src_mac: 31:d0:a3:0e:e3:f1
send packet # 8209 with src_mac: 1c:c9:b3:ef:28:e0
send packet # 8210 with src_mac: 8f:08:23:b7:0b:70
send packet # 8211 with src_mac: 18:05:70:5a:e7:82
send packet # 8212 with src_mac: ad:12:67:2f:1d:5e
send packet # 8213 with src_mac: 5d:2f:56:18:22:e6
send packet # 8214 with src_mac: 48:f2:53:1c:31:09
send packet # 8215 with src_mac: 25:b7:49:ce:64:45
send packet # 8216 with src_mac: 34:4f:d2:ab:e2:83
send packet # 8217 with src_mac: d5:43:49:8a:6b:30
send packet # 8218 with src_mac: 50:15:33:c7:a6:12
```

You should be able to see Eve starts sending a lot of packets into the network. Back to Eve's first terminal (switch back by "ctrl+a 0). After the attack traffic overflowed switches' address table, switches will start to broadcast Alice and Bob's traffic and they should start showing up in Eve's tcpdump trace:

```
"host: eve"
h 64
09:59:04.786619 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 9565, seq 727, len
gth 64
09:59:04.811200 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 9565, seq 727, lengt
h 64
09:59:05.785505 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 9565, seq 728, len
gth 64
09:59:05.809448 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 9565, seq 728, lengt
h 64
09:59:06.008711 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
09:59:06.029152 ARP, Reply 10.0.0.1 is-at a6:24:00:31:26:c4, length 28
09:59:06.791069 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 9565, seq 729, len
gth 64
09:59:06.826936 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 9565, seq 729, lengt
h 64
09:59:07.790758 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 9565, seq 730, len
gth 64
09:59:07.818253 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 9565, seq 730, lengt
h 64
09:59:08.789955 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 9565, seq 731, len
gth 64
09:59:08.812722 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 9565, seq 731, lengt
h 64

```