UNIVERSITEIT VAN AMSTERDAM

MSc. Systems and Networking Engineering

Security assessment on a VXLAN-based network

Author:
Guido Pineda Reyes
guido.pineda@os3.nl

Supervisors:
Maarten Dammers
maarten.dammers@vancis.nl
Maarthen Kastanja
maarthen.kastanja@vancis.nl

March 11, 2014

Abstract

This project focuses on implementing a VXLAN environment, which is a technology used in cloud computing deployments to solve scalabilty problems in large production environments, and later do a security assessment by deploying some of the known VLAN attacks in the VXLAN environment. Since this is a relatively new technology, no data on security tests measurements are available yet. The purpose of this project is to determine how feasible the known VLAN attacks are in a VXLAN environment. The first approach for this project is building the actual VXLAN environment and secondly performing the attacks on it. The research showed that some attacks were feasible in the VXLAN environment.

Contents

List of Figures

List of Tables

1 Introduction

In current cloud infrastructure of service providers, most of the servers are virtual machines (VMs). Sometimes VMs need to be migrated from one environment to another. Currently migration between different environments is done by connecting them on a layer 2 infrastructure with IEEE standard (802.1Q tags or QinQ). However, there are some limitations, in a VLAN environment, migration can only be done using Layer 2. VXLAN (Virtual eXtensible Local Area Network) came as a solution and has been submitted to the IETF for standardization [1]. This protocol can extend logical networks in different Layer 2 domains via a Layer 3 network. The purpose of this project is to deploy a security assessment on a VXLAN environment to see if there are differences between a VLAN and a VXLAN-based network.

Security assessments on a VXLAN environment have not yet been fully deployed, even in the IETF draft where some security considerations are mentioned, indicates that there are still no data on any security tests measurements [2]. For this project, some known VLAN attacks were deployed on a VXLAN environment and verified if these attacks were feasible or not.

The VXLAN environment was configured using a Linux kernel tunnel device [3].

1.1 Research

This project is the first of two projects in the master course "System and Networking Engineering" in the University of Amsterdam, and is result of four weeks of research.

1.1.1 Research question

The main research question of this research project is:

How feasible are the known VLAN attacks in a VXLAN environment?

This question can be split into different subquestions:

- Of the known VLAN attacks, which attacks were successful?
- What is the difference between this attacks in a VLAN and a VXLAN environment?
- Is there any way to mitigate this attacks or how to prevent them?
- How is the process of cloud migration in a VXLAN environment?

1.1.2 Scope

There are a few topics concerning security in the VXLAN technology, but the aim of this project is limited by deploying the known VLAN attacks in the VXLAN environment and determine how feasible they are.

To build the VXLAN environment three alternatives are tested, but the one used is the VXLAN feature on the Linux kernel.

1.1.3 Related work

Security flaws on a VXLAN-based network have not been fully tested or documented, but some security concerns have been pointed out [4], like session hijacking and other existing network security issues such as ARP spoofing, broadcast storms, and others [5].

2 Virtual eXtensible LAN

2.1 Introduction to VXLAN

In the cloud environment, customer demand has increased in recent years, and because of this, the actual physical infrastructure of data centers is required to support large numbers of virtual machines hosted on physical servers. Each virtual machine has its own MAC address and IP address and hosts are grouped in their according VLAN, but in a large production scenario, one might need to group thousands of hosts in one single VLAN. The limitation of VLAN identifiers is 4096.

Another requirement for a large production scenario, where multiple tenants exist, is to isolate them completely from each others traffic, and this is not optimal to implement over a shared network infrastructure.

Another requirement for virtualized environments is to have a Layer 2 network scale across the entire data center.

To address these requirements pointed out above and other requirements, Virtual eXtensible LAN provides an encapsulation scheme, which uses an overlay network to carry the MAC traffic from the individual Virtual Machines in an encapsulated format over a logical tunnel [6].

2.2 VXLAN Implementation

Network devices process VXLAN traffic transparently, that is, encapsulating traffic and routing it as IP traffic. VXLAN gateways, also called Virtual Tunnel End Points (VTEP), provide the encapsulation/de-encapsulation function. VTEP can be bridges in the hypervisor, VXLAN aware VM applications or VXLAN capable switches hardware.

Each VXLAN network segment is associated with a unique 24-bit VXLAN Network Identifier (VNI). The 24-bit address space allows scaling virtual networks beyond the 4096 available with 802.1Q to 16.7 million possible virtual networks. However, multicast and network hardware limitations reduce the real usable number of virtual networks.

Virtual machines on a VXLAN segment, do not require any special configuration, since all the encapsulation/de-encapsulation process takes part on the VTEP.

2.2.1 VXLAN frame encapsulation and forwarding

These following forwarding rules are applied in the VTEP:

- If two virtual machines try to communicate, and both reside on the same host, traffic is locally switched and no VXLAN encapsulation/de-encapsulation is performed.
- If two virtual machines try to communicate, and they reside on different hosts, all traffic is encapsulated/de-encapsulated with the appropriate VXLAN header in the built in VTEP and forwarded to the destination VTEP based on its local table, also called Forwarding Data Base (FDB), the destination VTEP will de-encapsulate the packet and deliver it to the recipient virtual machine.
- For unknown traffic or broadcast/multicast traffic, the local VTEP encapsulates the packet in a VXLAN header and multicasts the encapsulated frame to the VNI multicast address that is assigned to the VNI. The recipient VTEP sees the packet and processes it as normal unicast traffic.

2.2.2 Encapsulation format

For a packet leaving the VTEP, the encapsulation is as follows, see Figure: 1.

And adding to this headers, there is the original payload and lastly the Frame Checksum Sequence (FCS) for the Outer Ethernet Frame.

VXLAN traffic is encapsulated in a UDP packet, with the following overhead on each packet:

Outer Ethernet header (14) + UDP header (8) + IPv4 header (20) + VXLAN header (8) = 50 bytes

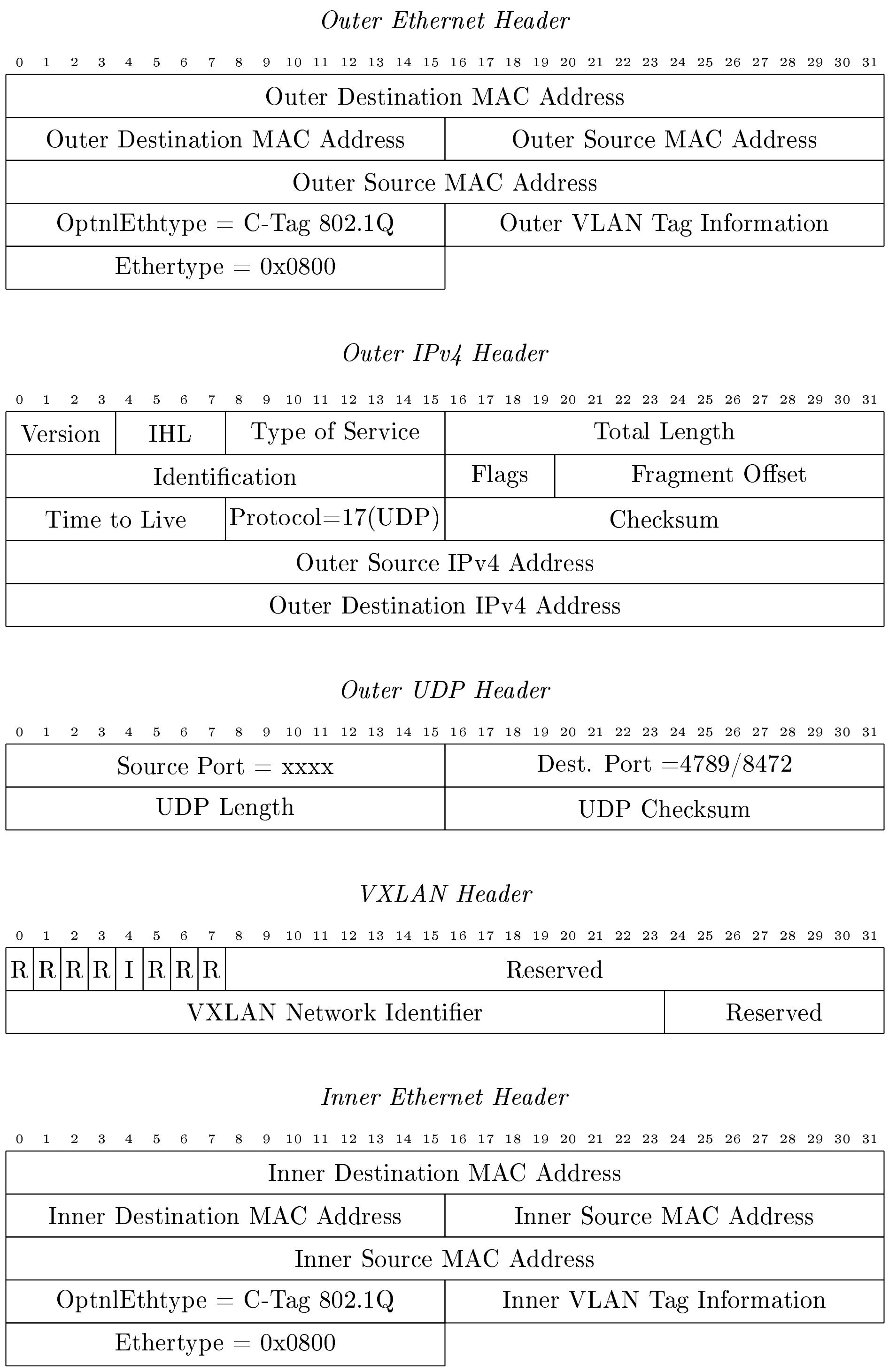


Figure 1: Encapsulation headers for a packet leaving a VTEP

Since all network devices need to handle 50 bytes greater than maximum transmission unit (MTU) size for the expected frame, the MTU must be adjusted for all devices that will carry VXLAN traffic. This includes the core network devices such as switches, routers and the VTEP device [7].

2.3 VM to VM communication

The basic use for VXLAN is to connect two virtual machines and make them look like they are in they are attached in the same layer two domain, even if the underlying network is not the same for each virtual machine, see Figure 2.

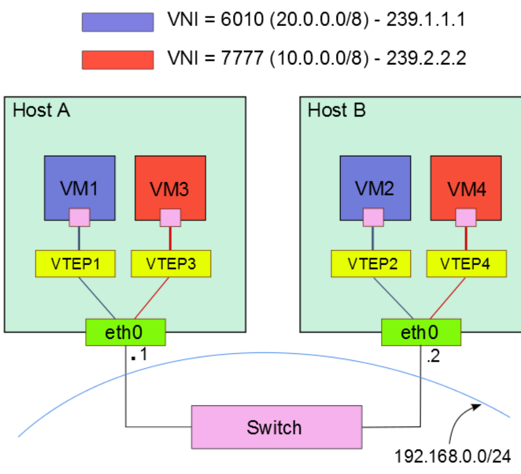


Figure 2: VXLAN simple use case

When two virtual machines try to communicate, they need to know each others MAC address, and the process of communication is as follows:

- VM1 sends an ARP packet requesting the MAC address associated with VM2.
- The ARP request is encapsulated by VTEP1 into a multicast packet to the multicast group associated with VNI.
- All VTEPs in the network see this multicast packet and the association of VTEP1 and VM1 to its VXLAN table (Forwarding Data Base).
- VTEP2 receive the multicast packet and de-encapsulates it, and sends the original ARP packet to the port group associated with the intended VNI .
- When VM2 sees the ARP packet, replies back with its MAC address.
- VTEP2 encapsulates the response in a unicast IP packet and sends it back to VTEP1.
- VTEP1 de-encapsulates the packet and passes it on to VM1

3 VLAN vs. VXLAN

In this chapter, a smal discussion is made about the conceptual differences between VLAN and VXLAN techniques and the hypotheses about the expected results for this research project.

3.1 Overview

VLAN is a technique to create independent logical networks within the same physical network, this technique uses the IEEE 802.1Q protocol to tag packets to identify traffic comming from a particular VLAN. It runs directly over the Ethernet frame with an Ethertype of value 0x8100. It allows to have up to 4096 virtual network identifiers (VNI).

VXLAN provides the same service to End Systems as a VLAN, this technology allows to have up to 16 million logical networks by using a VXLAN header with a virtual network identifier of 24 bits. Layer 2 segments generated by the virtual hosts are encapsulated in VXLAN/UDP/IP headers and are sent as regular IP packets over a regular layer 3 network.

VXLAN as in VLAN, has no control plane, that means that it still uses the same layer 2 mechanism to communicate (dynamic MAC learning). In a VXLAN environment, layer 2 flooding mechanisms, that is broadcast and multicast, use multicasting techniques between the communicating hosts in the overlay network, ideally a dedicated multicast group per VNI, so that only those hosts, that run virtual machines in that particular segment get the multicast. This is achieved by using ARP in a VLAN based network.

In a VLAN environment, communication between two hosts is done by using the same VNI, or by using trunking ports to carry multiple VLAN tags. For a VXLAN based network, all communication between hosts uses IP, therefore there is no requirement to use trunk ports to communicate two hosts. This translates to no more per tenant VLAN trunking on server ports, which is an issue in large scale cloud environments, where there are multiple tenants.

3.2 Security

Concerning to security, all layer 2 VLAN attacks are known to be successful when the proper security guidelines are not followed. In this research project the following attacks are about to be assessed:

- MAC Flooding attacks
- Double Encapsulation attacks
- ARP Attack
- UDP Flood Attack

The MAC Flooding attack in a VLAN environment is an exploit of a limitation of the way the switches and bridges work. There is a limit of addresses that can be learned in the memory of the switch, therefore, an attacker may flood the network with random MAC addresses and make the switch act as a hub, redirecting all traffic to addresses that cannot be learned anymore. The hypothesis for the outcome of this attack in a VXLAN environment is that, since the communication between virtual machines is encapsulated within UDP packets, the switch will not see this traffic and it will be discarded.

The Double Encapsulated attack has the main goal to send traffic to a host that belongs to a different VLAN by adding an additional 802.1Q header to the original frame. In a VXLAN environment, the goal is to add an additional VXLAN header to the original packet, and see how the packet is processed by the core network devices, in this case, the switch that communicates the two hypervisors.

The ARP attack has the goal to fool the switch into forwarding packets to a malicious device that claims to be the legitimate destination. In a VXLAN environment, the attacker may be either on the overlay network, and on the VXLAN segment of the two communicating virtual machines, and for both scenarios the traffic should be intercepted by the attacker. For the scenario where the attacker is on the overlay network, that is the physical network, where the hypervisors are located, all traffic coming from the virtual machines should be intercepted, knowing beforehand that the communication is encapsulated. With the attacker in the logical network, that is the same VXLAN segment where the virtual machines are communicating, the communication should be redirected to the attacker with no problem.

The UDP Flood attack has as main goal to flood the network with UDP packets and then cause a Denial of Service between two hosts that are trying to communicate. It is known that in a VLAN environment, the target host will become unreachable for other hosts in the network. In a VXLAN environment, this attack should behave as it does on a VLAN environment, taking into account that all communication is encapsulated in UDP packets.

In a layered scheme as is present in the OSI model, if one layer is victim of an attack, then all communications may be compromised without other layers being aware of the problem. In a VXLAN-based network, this behaviour should not be different from a VLAN-based environment.

4 Approach

The approach of this project is mainly trying to answer the research questions proposed in the Introduction. In order to do this, the following procedure is going to take place:

- Build the VXLAN prototype
- Deploy the security assessment on the prototype
- Focus on the most successful attacks
- Understand how this attacks work to give a solution on how to mitigate or prevent them.

5 Implementation

5.1 Building the VXLAN prototype

To build the VXLAN prototype, the following design was used, see Figure 2, where Host A and Host B, represent two clusters on different data centers, physically connected and both are in the same network, for the purpose of the prototype, this network is 192.168.0.0/24.

Two Virtual Machines reside on each host, and each Virtual Machine is in a different VXLAN segment (VNI), that is 6010 and 7777. Virtual Machines that are in the same VXLAN segment can communicate with each other.

The VXLAN segment with ID 6010 is associated with the multicast group 239.1.1.1, and every Virtual Machine is in network 20.0.0.0/8.

The VXLAN segment with ID 7777 is associated with the multicast group 239.2.2.2, and every Virtual Machine is in network 10.0.0.0/8.

Available options to deploy a VXLAN environment

There are a few options to configure VXLAN, and for the purpose of the project the following options were tested:

- VMware vSphere products
- VMware vSphere with Cisco Nexus 1000V
- Linux Kernel modification to support VXLAN

To build the VXLAN environment, the Linux option was used, and the configuration is as follows:

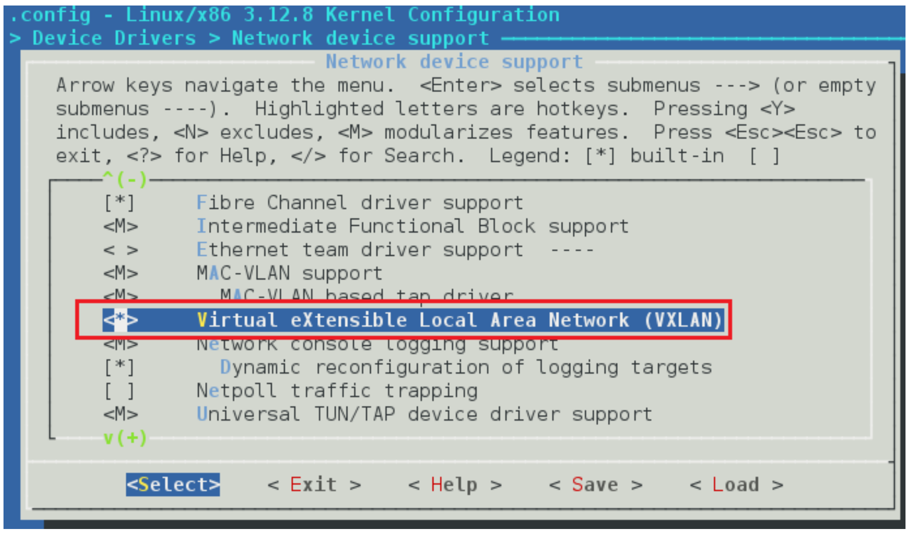
Virtual Tunnel End Point implementation (VTEP)

VXLAN is supported in at least kernel's version 3.7.0, for this purpose, to implement the VTEP, the kernel version that was used is version 3.12.8. The kernel used for this purpose was modified to enable VXLAN features, as well as the latest version of iproute2 (3.12.0) which supports VXLAN commands needs to be installed. The latest version of Debian (Debian 7.3.0) as the Operating System was used.

Compiling the kernel

In order to enable the VXLAN feature, which by default is disabled, the kernel has to be modified and later re-compiled. The following steps need to be followed:

- Before installing a new kernel, the following packages need to be installed: `bzip2, fakeroot, kernel-packages, ncurses-dev`
- Download the latest version of kernel, in this case the latest version at the moment is linux-13.12.8 [8].
- Decompress the file.
  - `tar -xJf linux-3.12.8.tar.xz`
- Under the `/usr/src/` directory, create a soft link, which will contain all the uncompressed files.
  - `ln -s linux-3.12.8 linux`
- A patch to support a TTL more than 1 for multicast packets needs to be applied [9].
  - `pwd:  /src/usr/linux/drivers/net/`
  - `patch -p1 < vxlan-allow-a-user-to-set-TTL-value.patch`, specify the vxlan.c file
- Prepare the compiling process.
  - `make clean & make mrproper`
- Copy the running configuration of the boot file
  - `cp /boot/config-‘uname -r‘ ./.config`
- Enable the VXLAN feature in the configuration command
  - `make menuconfig`, see Figure 3.
  - To enable the VXLAN feature: Device Drivers >Network device support >Virtual eXtensible Local Area Network (VXLAN), mark with <Y>key.
- After every modification has been made, the following command has to be run:
  - `make-kpkg clean`
  - `fakeroot make-kpkg -initrd -append-to-version=-vxlan_customized kernel_image kernel_headers`



```
.config - Linux/x86 3.12.8 Kernel Configuration
> Device Drivers > Network device support ─────────────────────────────
                          Network device support
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
  submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
  includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to
  exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]
  ────^(-)──────────────────────────────────────────────────────────
        [*]     Fibre Channel driver support
        <M>     Intermediate Functional Block support
        < >     Ethernet team driver support  ----
        <M>     MAC-VLAN support
        <M>       MAC-VLAN based tap driver
        <*>     Virtual eXtensible Local Area Network (VXLAN)
        <M>     Network console logging support
        [*]       Dynamic reconfiguration of logging targets
        [ ]     Netpoll traffic trapping
        <M>     Universal TUN/TAP device driver support
  └─────v(+)────────────────────────────────────────────────────────

         <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 3: Enabling VXLAN feature on kernel

Building the kernel

The files created, have to be installed to build the new customized kernel:

1. linux-image-3.12.8-vxlan_3.12.8-vxlan_customized-10.00.Custom_amd64.deb
2. linux-headers-3.12.8-vxlan_3.12.8-vxlan_customized-10.00.Custom_amd64.deb

This following command will install the new modified kernel with all features previously enabled:

- `dpkg -i linux-image-3.12.8-vxlan_3.12.8-vxlan_customized-10.00.Custom_amd64.deb`
- `dpkg -i linux-image-3.12.8-vxlan_3.12.8-vxlan_customized-10.00.Custom_amd64.deb`

Installing iproute2

VXLAN features are available on the new version of iproute, to install iproute, the following packages must be installed first:
`pkg-config, iptables-dev, bison, flex, db4.8-util, libgdbm-dev, libperl-dev, libsasl2-dev`
To install iproute2:

- `apt-get install pkg-config iptables-dev bison flex db4.8-util libgdbm-dev libperl-dev libsasl2-dev libdb*-dev`
- `./configure`
- `make`
- `make install`

Creating VXLAN interface (VTEP)

To create a new VTEP, the following commands have to be run:

For Host A, VNI 6010:

- `ip link add vxlan0 type vxlan id 6010 group 239.1.1.1 ttl 4 dev eth0`
- `ip addr add 20.0.0.1/8 dev vxlan0`

- `ip link set up vxlan0`

For Host A, VNI 7777:

- `ip link add vxlan0 type vxlan id 7777 group 239.2.2.2 ttl 3 dev eth0`
- `ip addr add 10.0.0.1/8 dev vxlan0`
- `ip link set up vxlan1`

For Host B, VNI 6010:

- `ip link add vxlan0 type vxlan id 6010 group 239.1.1.1 ttl 4 dev eth0`
- `ip addr add 20.0.0.2/8 dev vxlan0`
- `ip link set up vxlan0`

For Host B, VNI 7777:

- `ip link add vxlan0 type vxlan id 7777 group 239.2.2.2 ttl 3 dev eth0`
- `ip addr add 10.0.0.2/8 dev vxlan1`
- `ip link set up vxlan1`

Creation and configuration of Virtual Machines

On each host (Host A and Host B), two virtual machines, in total four, were installed using Virtual Box software. The purpose is to verify connection between the Virtual Machines that belong to the same VNI.

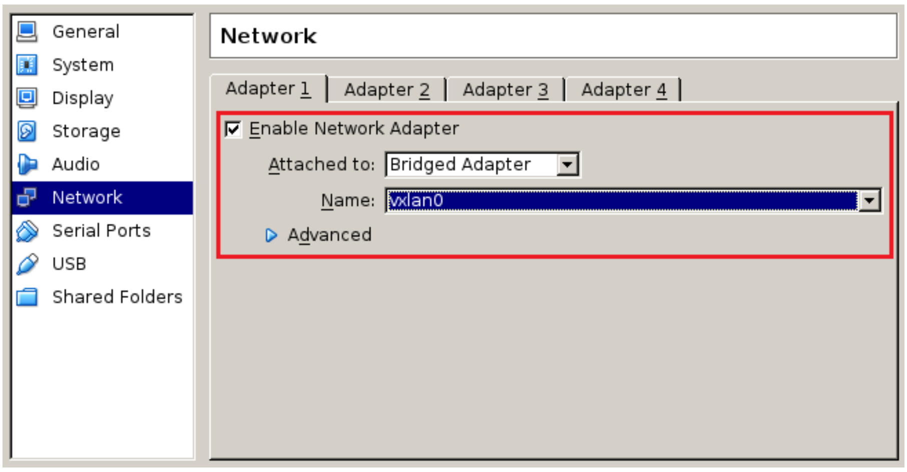
The configuration of the IP addresses for each Virtual Machine is as follows, see Table 1:

Host	Virtual Machine	IP address	Net mask	Gateway	VTEP
Host A	VM1	20.0.0.3	255.0.0.0	20.0.0.1	vxlan0
	VM3	10.0.0.3	255.0.0.0	10.0.0.1	vxlan1
Host B	VM2	20.0.0.4	255.0.0.0	20.0.0.2	vxlan0
	VM4	10.0.0.4	255.0.0.0	10.0.0.2	vxlan1

Table 1: IP Configuration for Virtual Machines

5.1.1 Configuring VM to belong to certain VNI

To configure a VM to belong to a certain VNI, the configuration has to be done on the hypervisor, that is, the Virtual Box software, where the virtual NIC for the virtual machine must be bridged with the VXLAN interfaces previously created (vxlan0, vxlan1), see Figure 4.



☑ Enable Network Adapter

Attached to: Bridged Adapter

Name: vxlan0

Figure 4: Configuration of Virtual Machine

5.1.2 Connectivity tests

The connectivity tests were successful, the communication of Virtual Machines in the same VNI can communicate with each other.

To verify if the environment is working, a sniffer is placed in the overlay network (i.e. 192.168.0.0/24), see Figure 5.

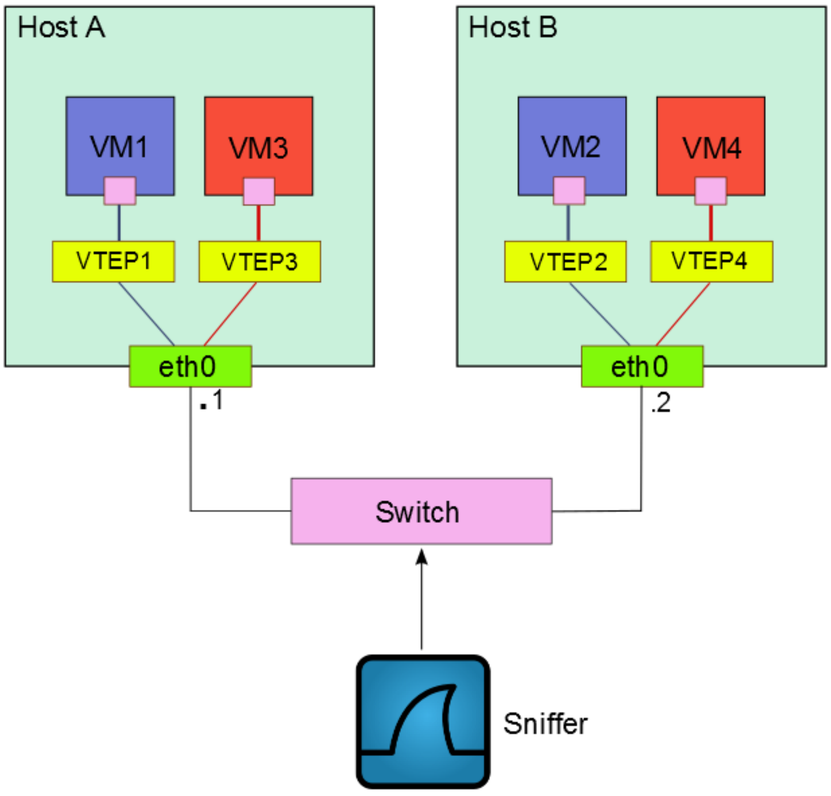
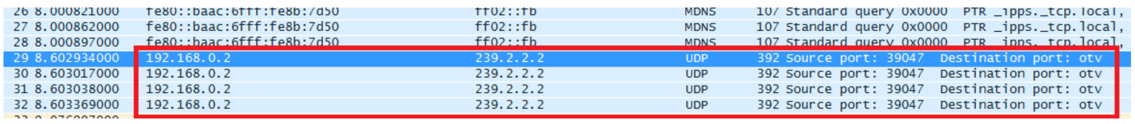


Figure 5: Connectivity test and VXLAN traffic verification

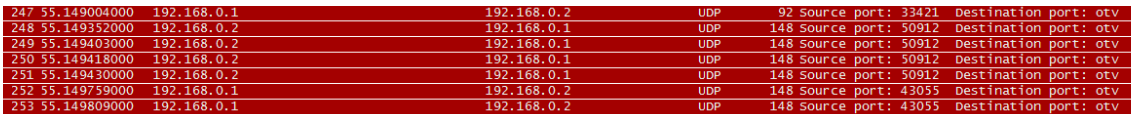
The result of the sniffing process was successful, in Figure 6, which shows the first stage of the VXLAN communication, in which a multicast message is sent to all the VTEPS in the network.



```
26 8.000821000   fe80::baac:6fff:fe8b:7d50   ff02::fb    MDNS   107 Standard query 0x0000  PTR _ipps._tcp.local,
27 8.000862000   fe80::baac:6fff:fe8b:7d50   ff02::fb    MDNS   107 Standard query 0x0000  PTR _ipps._tcp.local,
28 8.000897000   fe80::baac:6fff:fe8b:7d50   ff02::fb    MDNS   107 Standard query 0x0000  PTR _ipps._tcp.local,
29 8.602934000   192.168.0.2                 239.2.2.2   UDP    392 Source port: 39047  Destination port: otv
30 8.603017000   192.168.0.2                 239.2.2.2   UDP    392 Source port: 39047  Destination port: otv
31 8.603038000   192.168.0.2                 239.2.2.2   UDP    392 Source port: 39047  Destination port: otv
32 8.603369000   192.168.0.2                 239.2.2.2   UDP    392 Source port: 39047  Destination port: otv
```

Figure 6: First stage of communication, Multicast Message

And after that, all communication is sent directly using the Outer IP Address, see Figure 7.

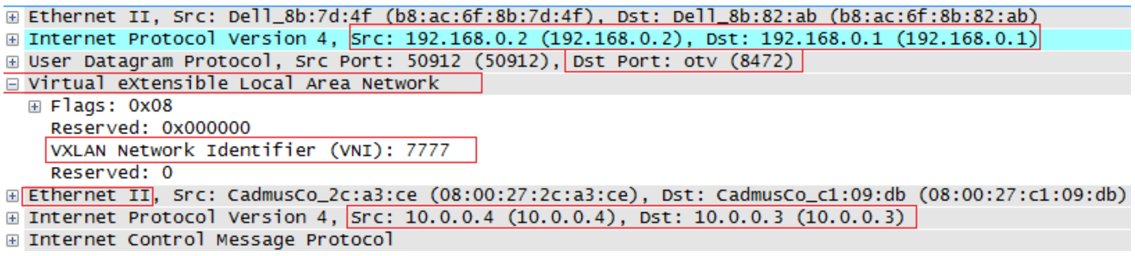


```
247 55.149004000   192.168.0.1   192.168.0.2   UDP    92 Source port: 33421  Destination port: otv
248 55.149352000   192.168.0.2   192.168.0.1   UDP   148 Source port: 50912  Destination port: otv
249 55.149403000   192.168.0.2   192.168.0.1   UDP   148 Source port: 50912  Destination port: otv
250 55.149418000   192.168.0.2   192.168.0.1   UDP   148 Source port: 50912  Destination port: otv
251 55.149430000   192.168.0.2   192.168.0.1   UDP   148 Source port: 50912  Destination port: otv
252 55.149759000   192.168.0.1   192.168.0.2   UDP   148 Source port: 43055  Destination port: otv
253 55.149809000   192.168.0.1   192.168.0.2   UDP   148 Source port: 43055  Destination port: otv
```

Figure 7: Direct communication using Outer IP Address

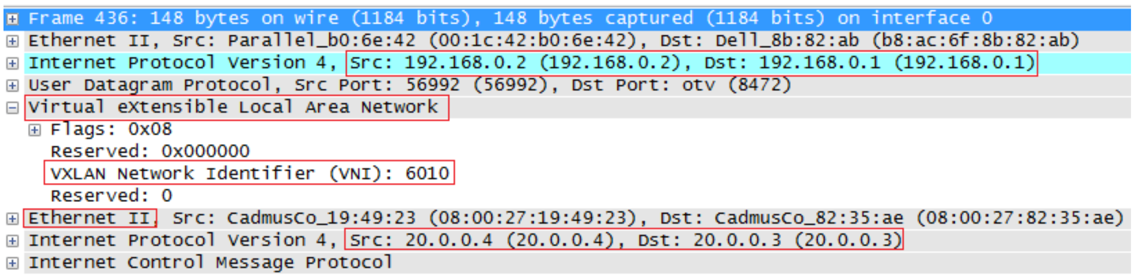
As seen before, in both Figures 6 and 7, the destination port is "otv", the reason is because the Linux implementation of VXLAN uses the same port as the OTV (Overlay Transport Virtualization) protocol [10], which is a proprietary protocol from Cisco.

In order to see the actual VXLAN traffic, the decoding feature in the sniffer needs to be enabled, after that, the VXLAN traffic communication can be verified, being able to see all the headers for a VXLAN packet, see Figures 8 and 9.



```
⊞ Ethernet II, Src: Dell_8b:7d:4f (b8:ac:6f:8b:7d:4f), Dst: Dell_8b:82:ab (b8:ac:6f:8b:82:ab)
⊞ Internet Protocol Version 4, Src: 192.168.0.2 (192.168.0.2), Dst: 192.168.0.1 (192.168.0.1)
⊞ User Datagram Protocol, Src Port: 50912 (50912), Dst Port: otv (8472)
⊟ Virtual eXtensible Local Area Network
  ⊞ Flags: 0x08
    Reserved: 0x000000
    VXLAN Network Identifier (VNI): 7777
    Reserved: 0
⊞ Ethernet II, Src: CadmusCo_2c:a3:ce (08:00:27:2c:a3:ce), Dst: CadmusCo_c1:09:db (08:00:27:c1:09:db)
⊞ Internet Protocol Version 4, Src: 10.0.0.4 (10.0.0.4), Dst: 10.0.0.3 (10.0.0.3)
⊞ Internet Control Message Protocol
```

Figure 8: VXLAN traffic decoded, from VM3 to VM4



```
⊞ Frame 436: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface 0
⊞ Ethernet II, Src: Parallel_b0:6e:42 (00:1c:42:b0:6e:42), Dst: Dell_8b:82:ab (b8:ac:6f:8b:82:ab)
⊞ Internet Protocol Version 4, Src: 192.168.0.2 (192.168.0.2), Dst: 192.168.0.1 (192.168.0.1)
⊞ User Datagram Protocol, Src Port: 56992 (56992), Dst Port: otv (8472)
⊟ Virtual eXtensible Local Area Network
  ⊞ Flags: 0x08
    Reserved: 0x000000
    VXLAN Network Identifier (VNI): 6010
    Reserved: 0
⊞ Ethernet II, Src: CadmusCo_19:49:23 (08:00:27:19:49:23), Dst: CadmusCo_82:35:ae (08:00:27:82:35:ae)
⊞ Internet Protocol Version 4, Src: 20.0.0.4 (20.0.0.4), Dst: 20.0.0.3 (20.0.0.3)
⊞ Internet Control Message Protocol
```

Figure 9: VXLAN traffic decoded, from VM1 to VM2

5.2 Security assessment

The security assessment on this VXLAN environment has the following scope: Deploy the known VLAN attacks, and see how feasible they are.

This project focuses on the following attacks [11]:

- MAC Flooding Attack
- Double-Encapsulated 802.1Q/Nested VLAN Attack
- ARP Attack
- UDP Flood Attack

For some attacks, two scenarios were considered, one with the attacker on the VXLAN segment or in the Tenant Network, and one with the attacker in the Overlay Network or in the Service Provider Network.

5.2.1 MAC Flooding Attack

In this attack, the target is the core switch that connects all hosts in the network. The purpose of this attack is flooding the switch with random MAC Addresses, and thus the switch acts as a hub where all traffic is directed to all hosts in the same VLAN.

Tools

To deploy the attacks, the tool `macof` was used. This tool floods the network with random MAC Address.

Scenario 1: Attacker in the Overlay Network

For this attack, the attacker lays in the overlay network, see Figure 10.

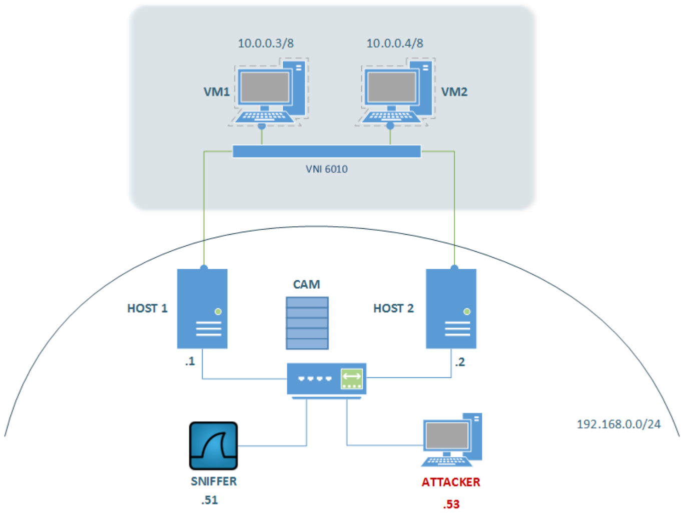


Figure 10: MAC flood attack, attacker in the overlay network

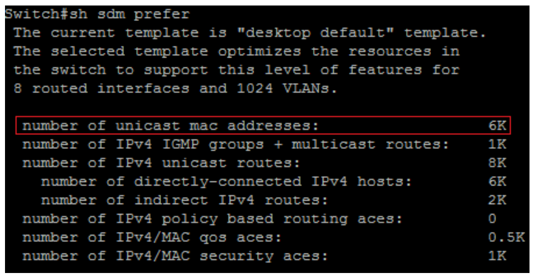
The MAC Address Table capacity of the switch is 6K, see Figure 11a.

Before the attack, the switch's MAC Address table shows a total of 24 MAC Addresses, see Figure 11b.

The attack is run:

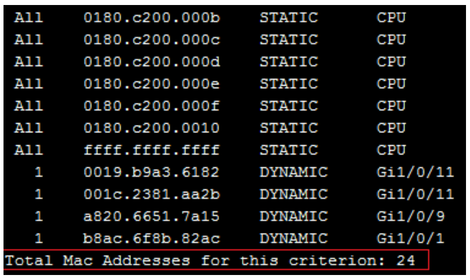
- `macof -i eth0 -n 1000000000`

After the attack, the switch's MAC Address Table shows a total of 6012 MAC Addresses see, Figure 11c.



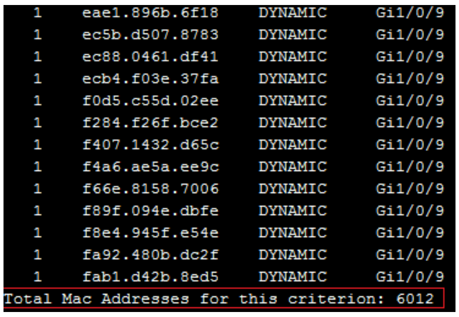
```
Switch#sh sdm prefer
 The current template is "desktop default" template.
 The selected template optimizes the resources in
 the switch to support this level of features for
 8 routed interfaces and 1024 VLANs.

  number of unicast mac addresses:                  6K
  number of IPv4 IGMP groups + multicast routes:    1K
  number of IPv4 unicast routes:                    8K
    number of directly-connected IPv4 hosts:        6K
    number of indirect IPv4 routes:                 2K
  number of IPv4 policy based routing aces:         0
  number of IPv4/MAC qos aces:                      0.5K
  number of IPv4/MAC security aces:                 1K
```

(a) CAM capacity



```
 All    0180.c200.000b    STATIC      CPU
 All    0180.c200.000c    STATIC      CPU
 All    0180.c200.000d    STATIC      CPU
 All    0180.c200.000e    STATIC      CPU
 All    0180.c200.000f    STATIC      CPU
 All    0180.c200.0010    STATIC      CPU
 All    ffff.ffff.ffff    STATIC      CPU
   1    0019.b9a3.6182    DYNAMIC     Gi1/0/11
   1    001c.2381.aa2b    DYNAMIC     Gi1/0/11
   1    a820.6651.7a15    DYNAMIC     Gi1/0/9
   1    b8ac.6f8b.82ac    DYNAMIC     Gi1/0/1
Total Mac Addresses for this criterion: 24
```

(b) Before MAC flood attack



```
   1    eae1.896b.6f18    DYNAMIC     Gi1/0/9
   1    ec5b.d507.8783    DYNAMIC     Gi1/0/9
   1    ec88.0461.df41    DYNAMIC     Gi1/0/9
   1    ecb4.f03e.37fa    DYNAMIC     Gi1/0/9
   1    f0d5.c55d.02ee    DYNAMIC     Gi1/0/9
   1    f284.f26f.bce2    DYNAMIC     Gi1/0/9
   1    f407.1432.d65c    DYNAMIC     Gi1/0/9
   1    f4a6.ae5a.ee9c    DYNAMIC     Gi1/0/9
   1    f66e.8158.7006    DYNAMIC     Gi1/0/9
   1    f89f.094e.dbfe    DYNAMIC     Gi1/0/9
   1    f8e4.945f.e54e    DYNAMIC     Gi1/0/9
   1    fa92.480b.dc2f    DYNAMIC     Gi1/0/9
   1    fab1.d42b.8ed5    DYNAMIC     Gi1/0/9
Total Mac Addresses for this criterion: 6012
```

(c) After MAC flood attack

Figure 11: MAC flood attack, scenario 1

Scenario 2: Attacker in the Tenant Network

For this attack, the attacker is in the Tenant Network, see Figure 12.

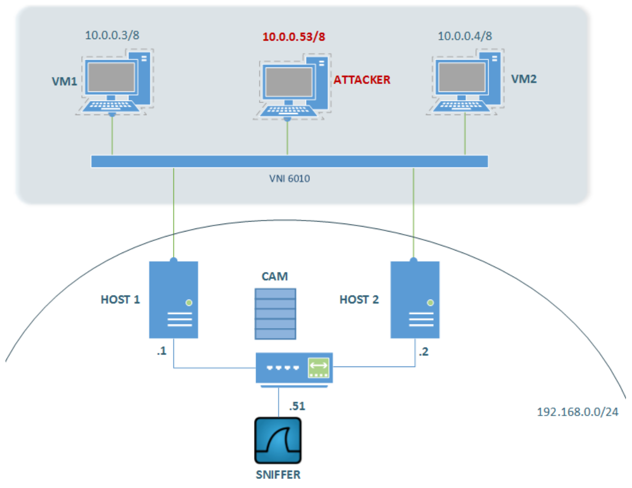
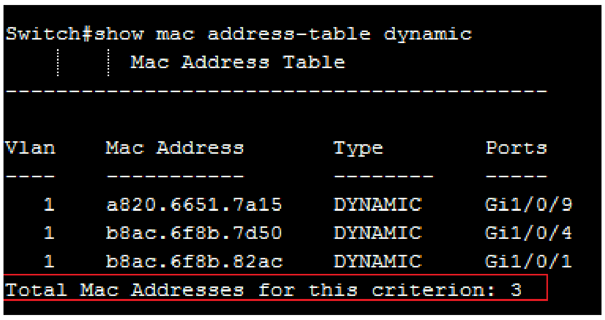


Figure 12: MAC flood attack, attacker in the tenant network

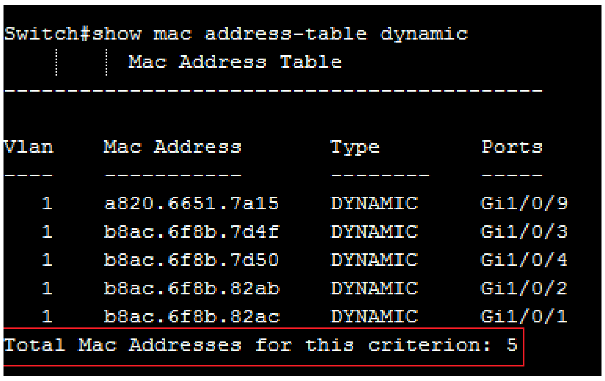
Before the attack, the MAC table shows 3 dynamic MAC Addresses, this MAC Addresses correspond to Host 1 and Host 2, and other hosts connected in the overlay network for testing, see Figure 13a

After the attack, running `macof` on a host on the tenant network, that is the VXLAN segment, the MAC table in the switch shows the same output as it was before the attack with the MAC Addresses of Hosts 1 and 2, see Figure 13b. The attack was not successful because all traffic is encapsulated in UDP packets and is not seen as ARP traffic at the level of the physical network



```
Switch#show mac address-table dynamic
          Mac Address Table
-------------------------------------------

Vlan    Mac Address       Type        Ports
----    -----------       --------    -----
   1    a820.6651.7a15    DYNAMIC     Gi1/0/9
   1    b8ac.6f8b.7d50    DYNAMIC     Gi1/0/4
   1    b8ac.6f8b.82ac    DYNAMIC     Gi1/0/1
Total Mac Addresses for this criterion: 3
```

(a) Before MAC flood attack



```
Switch#show mac address-table dynamic
          Mac Address Table
-------------------------------------------

Vlan    Mac Address       Type        Ports
----    -----------       --------    -----
   1    a820.6651.7a15    DYNAMIC     Gi1/0/9
   1    b8ac.6f8b.7d4f    DYNAMIC     Gi1/0/3
   1    b8ac.6f8b.7d50    DYNAMIC     Gi1/0/4
   1    b8ac.6f8b.82ab    DYNAMIC     Gi1/0/2
   1    b8ac.6f8b.82ac    DYNAMIC     Gi1/0/1
Total Mac Addresses for this criterion: 5
```

(b) After MAC flood attack

Figure 13: MAC flood attack, scenario 2

5.2.2 Double-Encapsulated 802.1Q/Nested VLAN Attack

This attack refers to sending 802.1Q double encapsulated frames. In the case of VXLAN the goal is to test if a VM in a different VXLAN segment is able to communicate with a host on a different VXLAN segment.

The only possible scenario is to have the attacker on a different VXLAN segment than the target, see Figure 12.

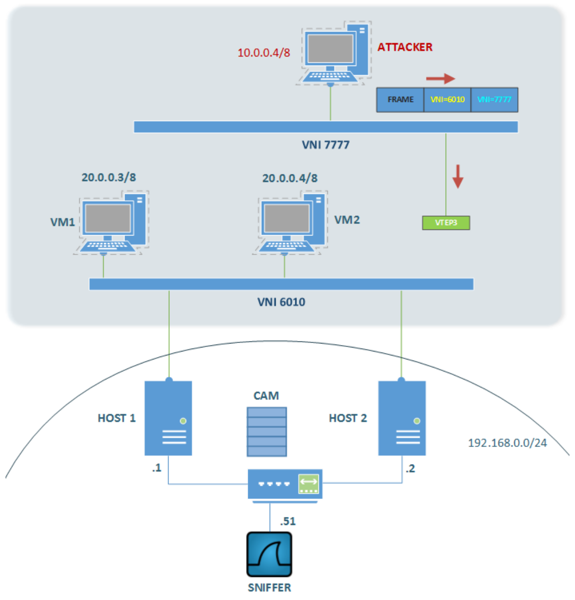


Figure 14: Double Tagging Attack

Tools

The tool used for this attack was `Scapy`. This tools allows to forge packets, and send them to the specified target. To use this tool, the VXLAN packet needs to be created. The following code needs to be inserted, while running `Scapy` in order to define the VXLAN packet [12]:

```
class ThreeBytesField(X3BytesField, ByteField):
    def i2repr(self, pkt, x):
        return ByteField.i2repr(self, pkt, x)

class VXLAN(Packet):
    name = "VXLAN"
    fields_desc = [ FlagsField("flags", 0x08, 8, ['R', 'R', 'R', 'I', 'R', 'R', 'R', 'R']),
                    X3BytesField("reserved1", 0x000000),
                    ThreeBytesField("vni", 0),
                    XByteField("reserved2", 0x00)]

def mysummary(self):
    return self.sprintf("VXLAN (vni=%VXLAN.vni%)")

bind_layers(UDP, VXLAN, dport=8472)
bind_layers(VXLAN, Ether)
```

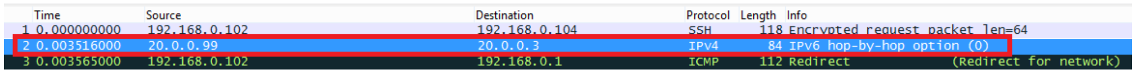
Deploying the attack

For a VXLAN environment, a double VXLAN packet was formed, by adding first a VXLAN header for the attacker, that is VNI 7777 , and then adding a header which belongs to the target, in this case, the target is VM1 and the VNI is 6010.

To run Scapy, the forged packet needs to be created, in order to do this, we need to specify the parameters for all headers:

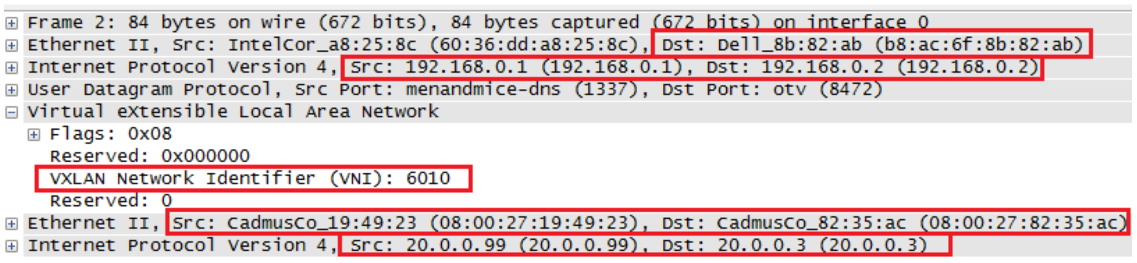
```
OuterMAC = Ether()
OuterMAC.dst = "b8:ac:6f:8b:82:ab"
OuterMAC.src = "b8:ac:6f:8b:7d:4f"

OuterIP = IP()
OuterIP.dst = "192.168.0.2"
OuterIP.src = "192.168.0.1"

InnerMAC = Ether()
InnerMAC.dst = "08:00:27:82:35:ac"
InnerMAC.src = "08:00:27:19:49:23"

InnerIP = IP()
InnerIP.dst = "20.0.0.3"
InnerIP.src = "20.0.0.99"

p = OuterMAC/OuterIP/UDP(sport=1337,dport=8472)/VXLAN(vni=6010)/InnerMAC/InnerIP/ICMP()
```

To verify that `Scapy` is working, a test packet was sent, and successfully capturing a packet from two VMs in the same VXLAN, see Figure 15a and Figure 15b .



```
   Time          Source                                  Destination                  Protocol  Length  Info
 1 0.000000000   192.168.0.102                           192.168.0.104                SSH         118 Encrypted request packet len=64
 2 0.003516000   20.0.0.99                               20.0.0.3                     IPv4         84 IPv6 hop-by-hop option (0)
 3 0.003565000   192.168.0.102                           192.168.0.1                  ICMP        112 Redirect              (Redirect for network)
```

(a) Scapy verification



```
⊞ Frame 2: 84 bytes on wire (672 bits), 84 bytes captured (672 bits) on interface 0
⊞ Ethernet II, Src: IntelCor_a8:25:8c (60:36:dd:a8:25:8c), Dst: Dell_8b:82:ab (b8:ac:6f:8b:82:ab)
⊞ Internet Protocol Version 4, Src: 192.168.0.1 (192.168.0.1), Dst: 192.168.0.2 (192.168.0.2)
⊞ User Datagram Protocol, Src Port: menandmice-dns (1337), Dst Port: otv (8472)
⊟ Virtual eXtensible Local Area Network
  ⊞ Flags: 0x08
    Reserved: 0x000000
    VXLAN Network Identifier (VNI): 6010
    Reserved: 0
⊞ Ethernet II, Src: CadmusCo_19:49:23 (08:00:27:19:49:23), Dst: CadmusCo_82:35:ac (08:00:27:82:35:ac)
⊞ Internet Protocol Version 4, Src: 20.0.0.99 (20.0.0.99), Dst: 20.0.0.3 (20.0.0.3)
```

(b) Scapy verification, VXLAN encapsulation

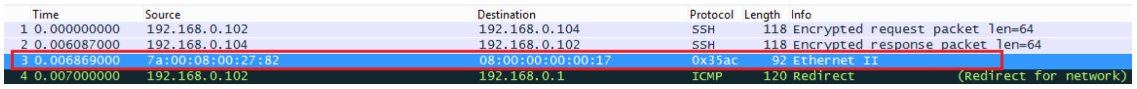
Figure 15: Verification for Scapy, two VMs in the same VXLAN segment

To test the attack, the attacker is on a different VXLAN segment, that is 7777, and a new tag (VNI 6010) is added in the packet.

```
p = OuterMAC/OuterIP/UDP(sport=1337,dport=8472)/VXLAN(vni=7777)/VXLAN(vni=6010)/InnerMAC/InnerIP/ICMP()
```

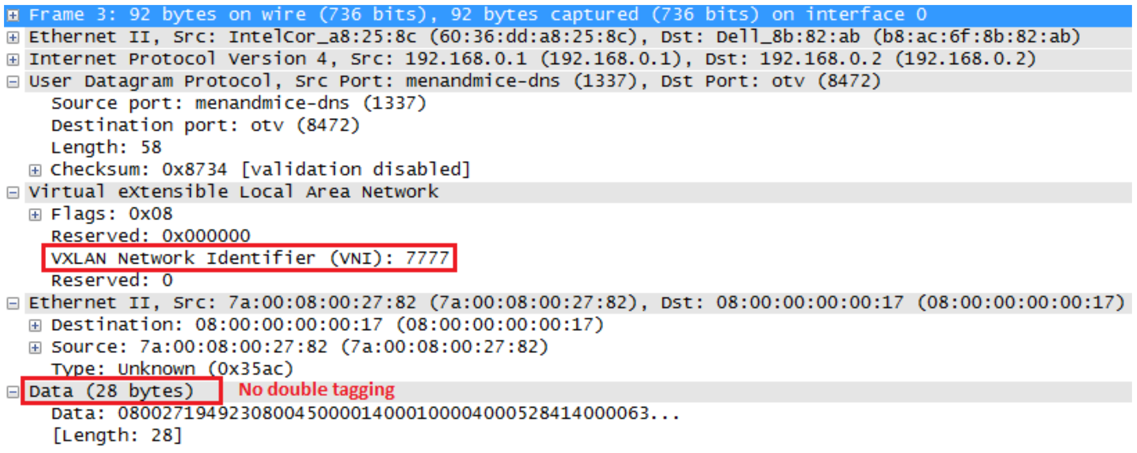
For this attack, a sniffer was placed in the VXLAN segment 6010, to capture an ICMP packet coming from a different VXLAN segment (VNI 7777). Also, a sniffer was placed in the generating packet VXLAN segment, to capture the outgoing traffic.

The test was not successful, and by looking at the packet generated, it is not well formed, and the double tagging can not be verified, see Figure 16.



```
   Time          Source                                  Destination                  Protocol  Length  Info
 1 0.000000000   192.168.0.102                           192.168.0.104                SSH         118 Encrypted request packet len=64
 2 0.006087000   192.168.0.104                           192.168.0.102                SSH         118 Encrypted response packet len=64
 3 0.006869000   7a:00:08:00:27:82                       08:00:00:00:00:17            0x35ac       92 Ethernet II
 4 0.007000000   192.168.0.102                           192.168.0.1                  ICMP        120 Redirect              (Redirect for network)
```

(a) Scapy attack



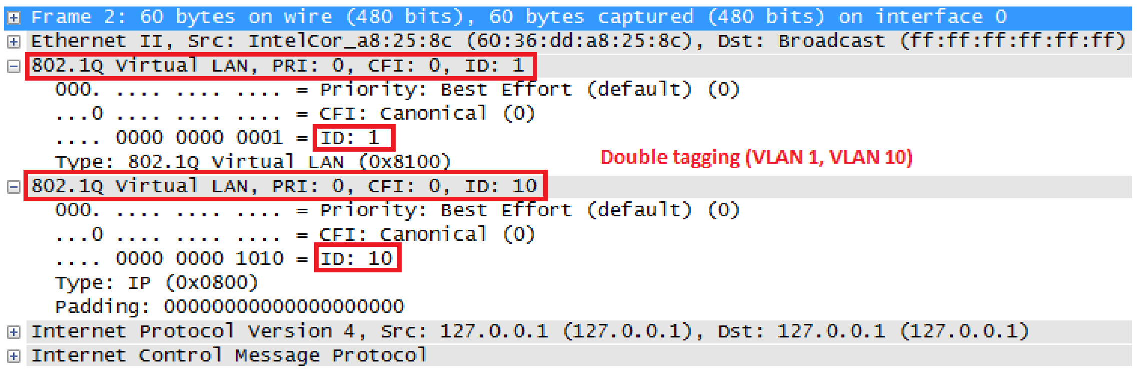
```
⊞ Frame 3: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface 0
⊞ Ethernet II, Src: IntelCor_a8:25:8c (60:36:dd:a8:25:8c), Dst: Dell_8b:82:ab (b8:ac:6f:8b:82:ab)
⊞ Internet Protocol Version 4, Src: 192.168.0.1 (192.168.0.1), Dst: 192.168.0.2 (192.168.0.2)
⊟ User Datagram Protocol, Src Port: menandmice-dns (1337), Dst Port: otv (8472)
    Source port: menandmice-dns (1337)
    Destination port: otv (8472)
    Length: 58
  ⊞ Checksum: 0x8734 [validation disabled]
⊟ Virtual eXtensible Local Area Network
  ⊞ Flags: 0x08
    Reserved: 0x000000
    VXLAN Network Identifier (VNI): 7777
    Reserved: 0
⊟ Ethernet II, Src: 7a:00:08:00:27:82 (7a:00:08:00:27:82), Dst: 08:00:00:00:00:17 (08:00:00:00:00:17)
  ⊞ Destination: 08:00:00:00:00:17 (08:00:00:00:00:17)
  ⊞ Source: 7a:00:08:00:27:82 (7a:00:08:00:27:82)
    Type: Unknown (0x35ac)
⊟ Data (28 bytes)
    Data: 080027194923080045000014000100004000528414000063...
    [Length: 28]
```

(b) Scapy attack, no double tagging

Figure 16: Scappy attack

To compare with a normal VLAN attack, a packet is forged using Scapy, and a double tagging should look like Figure 17.

- sendp(Ether()/Dot1Q(vlan=1)/Dot1Q(vlan=10)/IP()/ICMP())



```
⊞ Frame 2: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
⊞ Ethernet II, Src: IntelCor_a8:25:8c (60:36:dd:a8:25:8c), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
⊟ 802.1Q Virtual LAN, PRI: 0, CFI: 0, ID: 1
    000. .... .... .... = Priority: Best Effort (default) (0)
    ...0 .... .... .... = CFI: Canonical (0)
    .... 0000 0000 0001 = ID: 1
    Type: 802.1Q Virtual LAN (0x8100)
⊟ 802.1Q Virtual LAN, PRI: 0, CFI: 0, ID: 10
    000. .... .... .... = Priority: Best Effort (default) (0)
    ...0 .... .... .... = CFI: Canonical (0)
    .... 0000 0000 1010 = ID: 10
    Type: IP (0x0800)
    Padding: 00000000000000000000
⊞ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
⊞ Internet Control Message Protocol
```

Figure 17: Double Tagging VLAN format

The attack was not successful because the packet can not be processed as a double tagged frame. This attack also shows that this attack is not feasible on a VXLAN environment as it is on a VLAN-based network.

5.2.3 ARP Attack

An ARP attack exploits weaknesses on the ARP implementation, where any device in the network can claim that its MAC address is associated to any IP address. There is no verification mechanism of the correctness of the identities of these devices. The ARP attack fools the switch into forwarding packets to a device in the same VLAN, in this case, the traffic should be forwarded to devices in the same VXLAN segment.

For this attack there are two scenarios, in which the attacker is in the same VXLAN segment, and in the second scenario, the attacker is in the overlay network. The purpose of this attack is to perform a Man in the Middle attack and capture packets that communicate VM1 and VM2.

A sniffer is located in the overlay network, and in the same VXLAN segment as the virtual machines, to capture the network traffic.

5.2.4 Tools

The tool used to perform this attack is `arpspoof`, which is a tool to deploy ARP Spoofing attacks.

Scenario 1: Attacker in the Overlay Network

For this attack, the attacker is in the Overlay Network, where he floods the network with ARP packets, to redirect alll traffic coming from Host 1 to the attacker, see Figure 18.

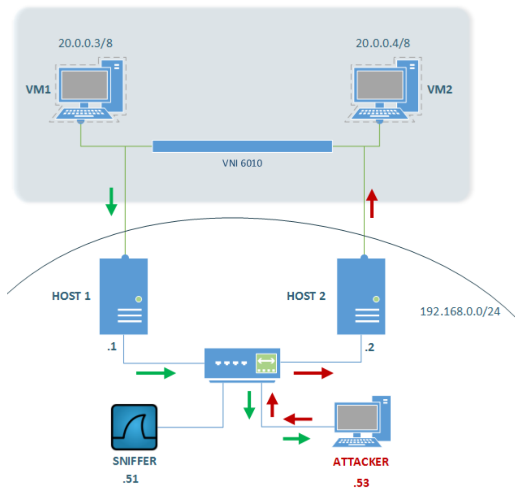


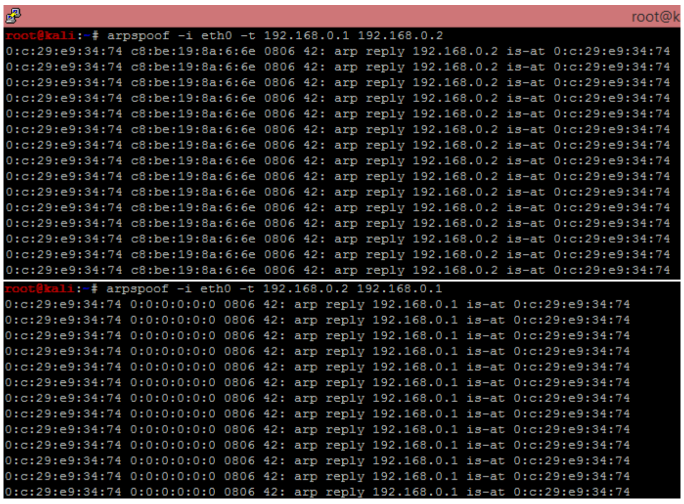
Figure 18: ARP Attack, scenario 1: Attacker on the Overlay Network

Before running the attack, the forwarding feature on the attacker has to be enabled, otherwise, a Denegation of Service (DoS) attack can be deployed, which is another alternative of attack.

- `echo 1 > /proc/sys/net/ipv4/ip_forward`

The goal is to tell Host 1 and Host 2 that their default gateway is the attacker's MAC address. The attack is performed by running the following commands on the attacker, see Figure 19:

- `arpspoof -i eth0 192.168.0.1 192.168.0.2`
- `arpspoof -i eth0 192.168.0.2 192.168.0.1`



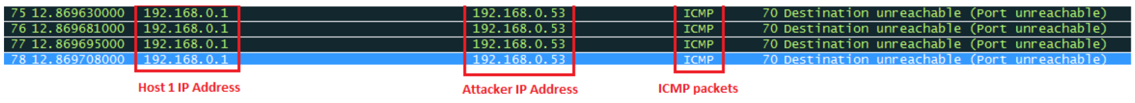
```
root@kali:~# arpspoof -i eth0 -t 192.168.0.1 192.168.0.2
0:c:29:e9:34:74 c8:be:19:8a:6:6e 0806 42: arp reply 192.168.0.2 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 c8:be:19:8a:6:6e 0806 42: arp reply 192.168.0.2 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 c8:be:19:8a:6:6e 0806 42: arp reply 192.168.0.2 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 c8:be:19:8a:6:6e 0806 42: arp reply 192.168.0.2 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 c8:be:19:8a:6:6e 0806 42: arp reply 192.168.0.2 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 c8:be:19:8a:6:6e 0806 42: arp reply 192.168.0.2 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 c8:be:19:8a:6:6e 0806 42: arp reply 192.168.0.2 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 c8:be:19:8a:6:6e 0806 42: arp reply 192.168.0.2 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 c8:be:19:8a:6:6e 0806 42: arp reply 192.168.0.2 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 c8:be:19:8a:6:6e 0806 42: arp reply 192.168.0.2 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 c8:be:19:8a:6:6e 0806 42: arp reply 192.168.0.2 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 c8:be:19:8a:6:6e 0806 42: arp reply 192.168.0.2 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 c8:be:19:8a:6:6e 0806 42: arp reply 192.168.0.2 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 c8:be:19:8a:6:6e 0806 42: arp reply 192.168.0.2 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 c8:be:19:8a:6:6e 0806 42: arp reply 192.168.0.2 is-at 0:c:29:e9:34:74
```

```
root@kali:~# arpspoof -i eth0 -t 192.168.0.2 192.168.0.1
0:c:29:e9:34:74 0:0:0:0:0:0 0806 42: arp reply 192.168.0.1 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 0:0:0:0:0:0 0806 42: arp reply 192.168.0.1 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 0:0:0:0:0:0 0806 42: arp reply 192.168.0.1 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 0:0:0:0:0:0 0806 42: arp reply 192.168.0.1 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 0:0:0:0:0:0 0806 42: arp reply 192.168.0.1 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 0:0:0:0:0:0 0806 42: arp reply 192.168.0.1 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 0:0:0:0:0:0 0806 42: arp reply 192.168.0.1 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 0:0:0:0:0:0 0806 42: arp reply 192.168.0.1 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 0:0:0:0:0:0 0806 42: arp reply 192.168.0.1 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 0:0:0:0:0:0 0806 42: arp reply 192.168.0.1 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 0:0:0:0:0:0 0806 42: arp reply 192.168.0.1 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 0:0:0:0:0:0 0806 42: arp reply 192.168.0.1 is-at 0:c:29:e9:34:74
0:c:29:e9:34:74 0:0:0:0:0:0 0806 42: arp reply 192.168.0.1 is-at 0:c:29:e9:34:74
```

Figure 19: Running `arpspoof`

While running the attack, VM1 is sending echo replies (ICMP packets) to VM2.

After deploying successfully the attack, al traffic is redirected to the attacker, and the ICMP packets that were supposed to be encapsulated were captured by the sniffer, see Figure 20.



```
75 12.869630000  192.168.0.1     192.168.0.53    ICMP    70 Destination unreachable (Port unreachable)
76 12.869681000  192.168.0.1     192.168.0.53    ICMP    70 Destination unreachable (Port unreachable)
77 12.869695000  192.168.0.1     192.168.0.53    ICMP    70 Destination unreachable (Port unreachable)
78 12.869708000  192.168.0.1     192.168.0.53    ICMP    70 Destination unreachable (Port unreachable)
```

Figure 20: ICMP packets captured by sniffer, ARP attack

Scenario 2: Attacker in the Tenant Network

For this scenario, the attacker is in the Tenant Network, specifically in VXLAN 6010, where VM1 and VM2 reside, see Figure 21.

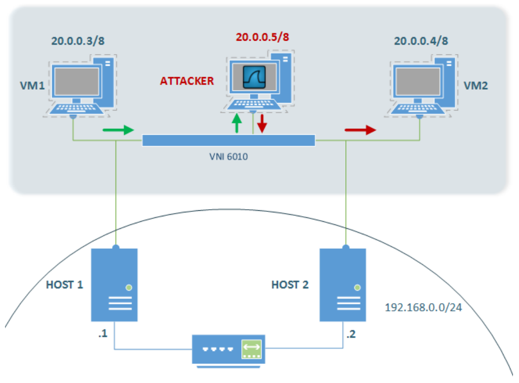
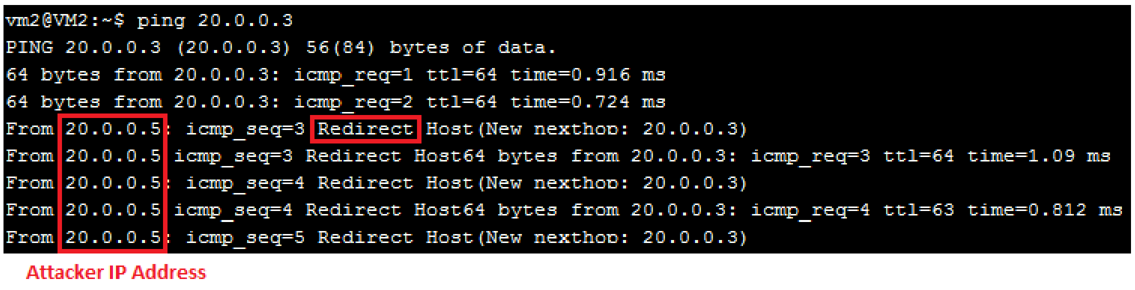


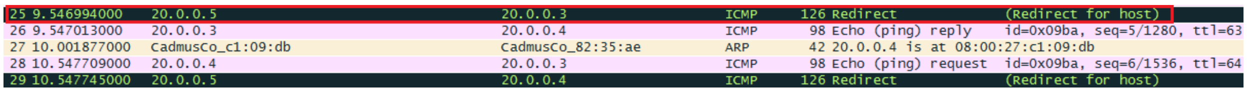
Figure 21: ARP attack, scenario 2: Attacker on Tenant Network

The result of this attack is successful when trying to deploy a Man in the Middle Attack, since all traffic was captured from the source VM1 and redirected to the destination VM2, see Figure 22b. Also, when sending echo requests messages, the "Redirected" message appears, see Figure 22a.



```
vm2@VM2:~$ ping 20.0.0.3
PING 20.0.0.3 (20.0.0.3) 56(84) bytes of data.
64 bytes from 20.0.0.3: icmp_req=1 ttl=64 time=0.916 ms
64 bytes from 20.0.0.3: icmp_req=2 ttl=64 time=0.724 ms
From 20.0.0.5: icmp_seq=3 Redirect Host(New nexthop: 20.0.0.3)
From 20.0.0.5 icmp_seq=3 Redirect Host64 bytes from 20.0.0.3: icmp_req=3 ttl=64 time=1.09 ms
From 20.0.0.5: icmp_seq=4 Redirect Host(New nexthop: 20.0.0.3)
From 20.0.0.5 icmp_seq=4 Redirect Host64 bytes from 20.0.0.3: icmp_req=4 ttl=63 time=0.812 ms
From 20.0.0.5: icmp_seq=5 Redirect Host(New nexthop: 20.0.0.3)
```

(a) Traffic redirected from the attacker



```
25 9.546994000   20.0.0.5             20.0.0.3             ICMP   126 Redirect              (Redirect for host)
26 9.547013000   20.0.0.3             20.0.0.4             ICMP    98 Echo (ping) reply     id=0x09ba, seq=5/1280, ttl=63
27 10.001877000  CadmusCo_c1:09:db    CadmusCo_82:35:ae    ARP     42 20.0.0.4 is at 08:00:27:c1:09:db
28 10.547709000  20.0.0.4             20.0.0.3             ICMP    98 Echo (ping) request   id=0x09ba, seq=6/1536, ttl=64
29 10.547745000  20.0.0.5             20.0.0.4             ICMP   126 Redirect              (Redirect for host)
```

(b) Traffic captured

Figure 22: ARP Attack from Tenant Network

The result shows that it is feasible to launch this attack from within any network, that is from a VXLAN segment, or from the physical network, and all the traffic is redirected to the attacker.

5.2.5 UDP Flood Attack

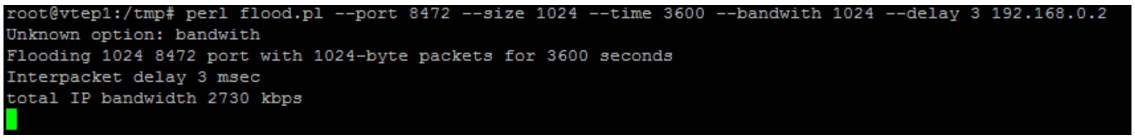
UDP flood attack is a kind of DoS attack using the User Datagram Protocol. The goal of this attack flood the network with UDP packets, to possibly force the victim to be unreachable by other hosts in the network.

Tools

The tool used for this attack is a `Perl` script, [13]. This tool specifies a target, a UDP port to attack and the size of the packets to be sent.

The attacker should be on the Overlay Network, since there is where UDP packets flow. In this case, the victim is Host 2, with IP Address 192.168.0.2/24.
The attack is run by the following command, also, see Figure 23.

- `perl flood.pl -port 8472 -size 1024 -time 3600 -bandwith 1024 -delay 3 192.168.0.2`

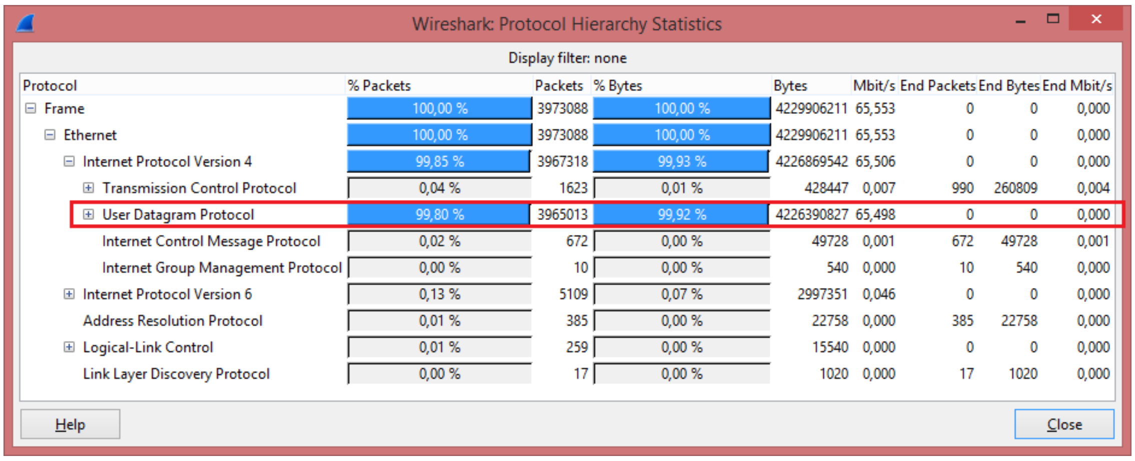


```
root@vtep1:/tmp# perl flood.pl --port 8472 --size 1024 --time 3600 --bandwith 1024 --delay 3 192.168.0.2
Unknown option: bandwith
Flooding 1024 8472 port with 1024-byte packets for 3600 seconds
Interpacket delay 3 msec
total IP bandwidth 2730 kbps
```

Figure 23: UDP flood attack, with perl script

After the attack is performed while VM1 is sending echo requests to VM2, the network is flooded with UDP packets, the attack runs for 3600 seconds, but the virtual machines can communicate normally, which means the attack using this tool is not very effective.

All packets were captured during the attack by a sniffer in the Overlay Network, and shows the amount of UDP traffic captured, which is very high, but the communication in the Tenant Network is still working, see Figure 24.



Display filter: none

Protocol	% Packets	Packets	% Bytes	Bytes	Mbit/s	End Packets	End Bytes	End Mbit/s
⊟ Frame	100,00 %	3973088	100,00 %	4229906211	65,553	0	0	0,000
⊟ Ethernet	100,00 %	3973088	100,00 %	4229906211	65,553	0	0	0,000
⊟ Internet Protocol Version 4	99,85 %	3967318	99,93 %	4226869542	65,506	0	0	0,000
⊞ Transmission Control Protocol	0,04 %	1623	0,01 %	428447	0,007	990	260809	0,004
⊞ User Datagram Protocol	99,80 %	3965013	99,92 %	4226390827	65,498	0	0	0,000
Internet Control Message Protocol	0,02 %	672	0,00 %	49728	0,001	672	49728	0,001
Internet Group Management Protocol	0,00 %	10	0,00 %	540	0,000	10	540	0,000
⊞ Internet Protocol Version 6	0,13 %	5109	0,07 %	2997351	0,046	0	0	0,000
Address Resolution Protocol	0,01 %	385	0,00 %	22758	0,000	385	22758	0,000
⊞ Logical-Link Control	0,01 %	259	0,00 %	15540	0,000	0	0	0,000
Link Layer Discovery Protocol	0,00 %	17	0,00 %	1020	0,000	17	1020	0,000

Figure 24: UDP flood attack, captured traffic

The possible failure of this attack, would be by not running for enough time the script, since this is a proven attack, it should have worked in the only possible scenario where the attacker is on the physical network. The known effect of this attack is that it causes a Denial of Service, bringing down the access to other network resources.

5.2.6 Evaluation of Attacks

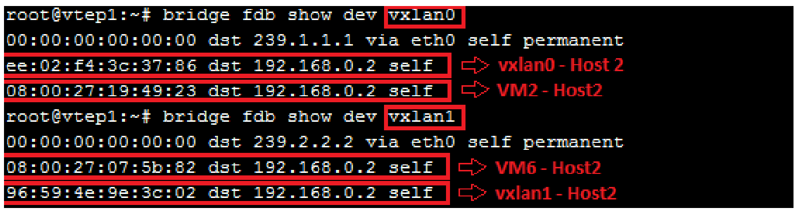
The evaluation of all the attacks performed on the VXLAN environment is summarized in the Table 2:

Attack	**Results: Scenario on**		**Tools**
	Overlay Network	**Tenant Network**	
MAC Flooding Attack	🟩 Successful	🟥 Failed	`macof`
Double-Encapsulated/Nested VLAN Attack	N/A	🟥 Failed	`scapy`
ARP Attack	🟩 Successful	🟩 Successful	`arpspoof`
UDP Flood Attack	🟥 Failed	N/A	`flood.pl`

Table 2: Evaluation of attacks performed

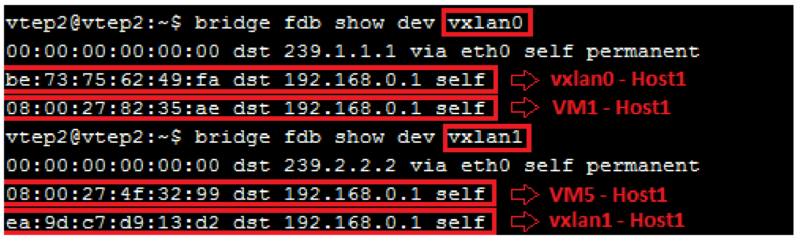
In a VXLAN environment, communication between virtual machines is done by learning and creating a Forwarding Table Entries. The learning process is based on packets received. The VTEP learns based on the inner and outer header of the packets received.

This tables show the routes for outgoing packets. For Host 1, the Forwarding Table shows the MAC address of the device VXLAN interface (VTEP: vxlan0 and vxlan1) on Host 2, and also shows the MAC address of Virtual Machine 1 and 6 (VM1, VM6), which are on Host 2 as well. For Host2, it shows the same information, the VXLAN interface (vxlan0 and vxlan1) on Host1, and the MAC addresses of Virtual Machines 2 and 5 (VM2, VM5), which are on Host 1. See, Figure 25.



```
root@vtep1:~# bridge fdb show dev vxlan0
00:00:00:00:00:00 dst 239.1.1.1 via eth0 self permanent
ee:02:f4:3c:37:86 dst 192.168.0.2 self
08:00:27:19:49:23 dst 192.168.0.2 self
root@vtep1:~# bridge fdb show dev vxlan1
00:00:00:00:00:00 dst 239.2.2.2 via eth0 self permanent
08:00:27:07:5b:82 dst 192.168.0.2 self
96:59:4e:9e:3c:02 dst 192.168.0.2 self
```

(a) FDB on Host 1



```
vtep2@vtep2:~$ bridge fdb show dev vxlan0
00:00:00:00:00:00 dst 239.1.1.1 via eth0 self permanent
be:73:75:62:49:fa dst 192.168.0.1 self
08:00:27:82:35:ae dst 192.168.0.1 self
vtep2@vtep2:~$ bridge fdb show dev vxlan1
00:00:00:00:00:00 dst 239.2.2.2 via eth0 self permanent
08:00:27:4f:32:99 dst 192.168.0.1 self
ea:9d:c7:d9:13:d2 dst 192.168.0.1 self
```

(b) FDB on Host 2

Figure 25: Forwarding Table for Hosts 1 and 2

A possible attack to the Forwarding Table, would be changing it, in order to redirect all traffic to the attacker that should understand VXLAN traffic. This attack seems like the ARP Attack, where the goal of the attacker is to redirect all the traffic to itself. This attack should be referred as a further research and it is out of the scope of this research project.

All this attacks are proven to be successful in a VLAN environment. In a VXLAN environment, this are the results, see Table 3:

5.3 Mitigation and prevention

The mitigation and prevention techniques for known VLAN attacks are based on security best practices on network devices.

Here are the attacks, and how to prevent or mitigate them.

Attack	**VXLAN (Obtained results)**	
	Attacker on physical network	**Attacker on logical network**
MAC Flood Attack	Successful attack. The CAM Table is filled with random MAC Addresses	Failed attack. The CAM Table is not affected by this attack.
Double Encapsulated 802.1Q/Nested VLAN Attack	N/A	Failed attack. The packets seems like is not well formed when adding the additional VXLAN header.
ARP Attack	Successful attack. The attacker is able to eavesdrop communication from the logical network.	Successful attack. The result is similar to the VLAN environment.
UDP Flood Attack	Failed attack. The communication between the virtual machines is still reliable.	N/A

Table 3: Results of attacks

5.3.1 MAC Flooding Attack

The alternatives to prevent and mitigate these attacks are:

- Enable a mechanism that restricts the number of MAC Addresses allowed to connect to a switch port.
- Specify statically the MAC Address of the Host that connects to the specific port.
- By using an Intrusion Detection System, to detect unusual ARP traffic.

5.3.2 Double-Encapsulated/Nested VLAN Attack

This attack was not successful on the VXLAN environment, but in a VLAN environment, the mitigation mechanisms for this attack are:

- Not using the native VLAN in any port by forcing all traffic on trunk to always carry a tag.

5.3.3 ARP Attack

This attack can be prevented by blocking the direct communication between the attacker and the victim. That can be achieved by configuring private VLANs, or private communication at the level of the Service Provider network, between the hosts.

5.3.4 UDP Flood Attack

This attack was not successful on the VXLAN environment. But to mitigate it, an Intrusion Detection System should be able to detect unusual UDP traffic.

6 Migration process

Cloud migration, is the process of moving data, applications or other busines elements from an organization's on site computers to the cloud, or moving them from one environment to another [14].

The process of migration in a VXLAN environment, where usually physical hosts are located in different networks, need the following requirements, to move applications from one host to another, see Figure 26.

- Multicast enabled in physical networking devices to carry multicast traffic using the IP multicast groups, this is to minimize flooding frames to VTEPs that do not need them.
- Multicast routing enabled, to carry multicast traffic across networks.
  - IGMP (Internet Group Management Protocol), used between hosts and routers on a LAN to track the multicast groups of which hosts are members.
  - PIM (Protocol Independent Multicast), used between routers to track multicast packets that are forwarded to each other and to their connected LANs.

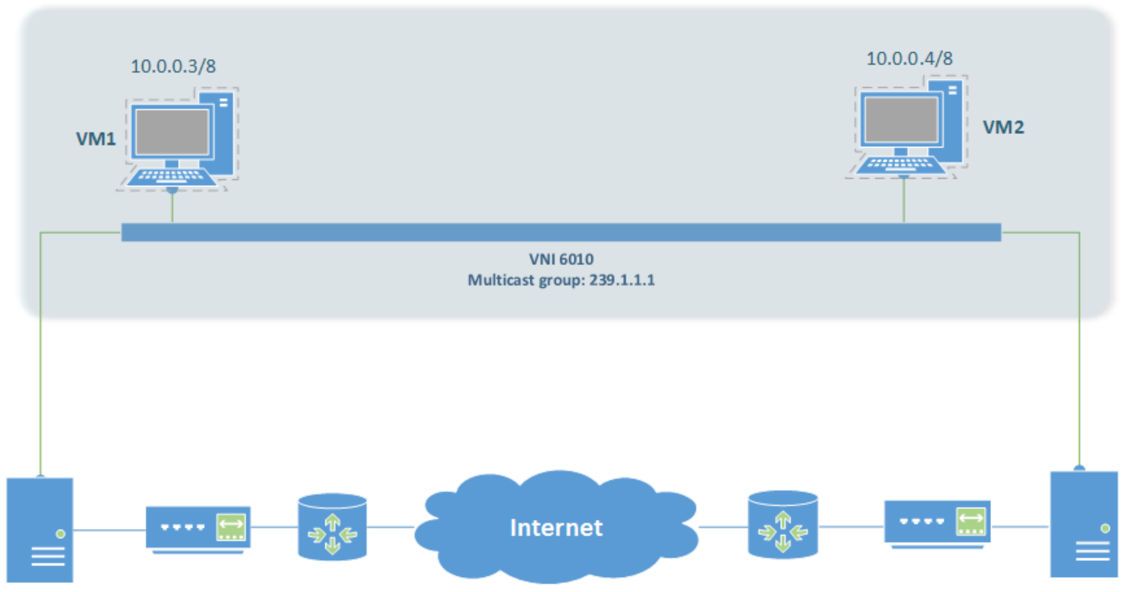


Figure 26: Cloud Migration

There are some reasons to migrate elements in a cloud environment, here are the most important:

- Move virtual machines to balance load of hosts.
- Move virtual machines temporally for maintenance.
- To allow scalability in tenant's applications.

7 Conclusion

In this section, the conclusions for this research project are made.

VXLAN is a new technique that allows to extend logical networks over an overlay network by encapsulating/de-encapsulating traffic from and to virtual machines that reside on a physical host. This technology has some vulnerabilities that are also applicable in a VLAN environment.

The security assessment showed that some known VLAN attacks are feasible in a VXLAN environment, the predicted outcome for this project is as expected. During the development of these research project, some attacks such as Private VLAN and Spanning Tree Protocol attacks, turned out to be irrelevant in a VXLAN environment and thus they were not considered during this evaluation.

The MAC flood attack, showed that an attacker in the overlay network, that is the service provider's or the physical network, can affect the core network devices flooding them with random MAC addresses and making the switch act as a hub redirecting traffic to every host connected to it. On the other hand, if the attacker is in the VXLAN segment, this attack has no effect, since all the packets are encapsulated and have no destination, this also proves that hosts in a VXLAN segment remain isolated from other networks.

The Double-Encapsulation 802.1Q/Nested VLAN attack showed that the attack is not feasible on a VXLAN environment using the tool `Scapy`, the reason is because the forged packets are not processed by the core network devices. By capturing packets with a sniffer, shows that the packet is not well formed. This attack failure shows that VXLAN segments are isolated from each other, and virtual machines that belong to a certain VXLAN segment cannot communicate with virtual machines that belong to a different VXLAN segment unless they are connected through a routing device.

The ARP attack showed that both scenarios, where the attacker is in the overlay network, and where the attacker is in the VXLAN segment, it is feasible to redirect all traffic to the attacker and perform a Man in the Middle Attack or even a DoS attack.

The UDP Flood attack showed that even if the network was flooded with UDP packets, the communication remains stable between virtual hosts. Unlike in a VLAN environment, where it is known that this attack causes a Denial of Service between the communicating hosts, in a VXLAN environment, this attack did not behave as expected in the hypotheses.

The found vulnerabilities may be reduced by following best practices and correctly configure the core network devices, although for this research project, these techniques that are used to reduce or mitigate these attacks were not implemented in a practical way due to limited time. The mitigation and prevention techniques are the same as in the VLAN environment, that is by correctly configuring the network devices and also by implementing mechanisms such as Intrusion Detection Systems to detect malicious and unusual traffic.

A VXLAN technology is used to scale capacity in a service provider data center, where there are thousands of tenants. In such environments it is required to have multicast capabilities enabled, this is to route multicast traffic across networks, where there are multiple VTEPs in place, and to minimize flooding frames to VTEPs that do not need them.

8 Further research

This research project was deployed using some features available in Linux kernel, but many more options are available to deploy a VXLAN environment, and some of the options are:

- VMware vSphere products
- Cisco Nexus 1000V

Due to limited time, and lack of experience with other options to deploy a VXLAN environment, such as VMware and Cisco Nexus 1000v, other types of weakness might be spotted and reviewed.

The way VTEPs communicate is by learning and creating forwarding tables entries, one possible attack would be by trying to modify this table entry and to redirect all traffic to an attacker in the overlay network.

VXLAN makes use of multicasting to carry traffic across networks by using protocols such as IGMP and PIM, another security assessment should be made by trying to exploit these protocols' weaknesses.

Because of time constraints, the prevention and mitigation techniques are just theoretical, for future a research, this techniques could be applied in a practical way in the VXLAN environment.

References

[1] VXLAN: A framework for overlaying virtualized layer 2 networks over layer 3 networks. http://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-07.

[2] Security considerations: VXLAN, a framework for overlaying virtualized layer 2 networks over layer 3 networks. http://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-07#section-7.

[3] Virtual extensible local area networking documentation. https://www.kernel.org/doc/Documentation/networking/vxlan.txt.

[4] Encapsulation risk and VXLAN. http://www.flyingpenguin.com/?p=13702.

[5] What is VXLAN? http://securitytheatre.tumblr.com/post/11682956200/all-about-the-path.

[6] Introduction: VXLAN, a framework for overlaying virtualized layer 2 networks over layer 3 networks. http://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-07#section-1.

[7] Encapsulation: Deploying the VXLAN feature in cisco nexus 1000v series switches. http://www.cisco.com/en/US/prod/collateral/switches/ps9441/ps9902/guide_c07-702975.html.

[8] The linux kernel archives. https://www.kernel.org/.

[9] Patchwork vxlan: allow a user to set TTL value. http://patchwork.ozlabs.org/patch/195622/.

[10] VXLAN: Virtual extensible local area network, source code. https://github.com/torvalds/linux/blob/master/drivers/net/vxlan.c.

[11] What are the possible attacks in a VLAN-based network? http://www.cisco.com/en/US/products/hw/switches/ps708/products_white_paper09186a008013159f.shtml#wp39042.

[12] Layer for vxlan using scapy. http://bb.secdev.org/scapy/pull-request/26/added-a-layer-for-vxlan/diff#chg-scapy/contrib/vxlan.py.

[13] UDP flood in perl. http://wiki.nil.com/UDP_flood_in_Perl.

[14] Definition: Cloud migration. http://searchcloudapplications.techtarget.com/definition/cloud-migration.

[15] Vmware VXLAN deployment guide. http://www.vmware.com/files/pdf/techpaper/VMware-VXLAN-Deployment-Guide.pdf.