

# **Anti-DNS Spoofer to Detect, Prevent and Recover**

## **A PROJECT REPORT**

*Submitted by*

21MEI10024	RAGHUNANDAN TOMAR
21MEI10036	SARTHAK YADAV
21MEI10004	ARYAN KHATRI
21MEI10040	VAKUL RAINA

*in partial fulfillment for the award of the degree  
of*

**INTEGRATED MASTER OF TECHNOLOGY**

*in*

**COMPUTER SCIENCE AND ENGINEERING**



**VIT<sup>®</sup>**  
**BHOPAL**  
[www.vitbhopal.ac.in](http://www.vitbhopal.ac.in)

**SCHOOL OF COMPUTING SCIENCE AND ENGINEERING**

**VIT BHOPAL UNIVERSITY**

**KOTRIKALAN, SEHORE  
MADHYA PRADESH - 466114**

SEP 2022

**VIT BHOPAL UNIVERSITY, KOTRIKALAN,  
SEHORE  
MADHYA PRADESH – 466114**

**BONAFIDE CERTIFICATE**

Certified that this project report titled “**Anti-DNS Spoofer to Detect, Prevent and Recover**” is the bonafide work of **Raghunandan Tomar** (21MEI10024), **Vakul Raina**(21MEI10040), **Aryan Khatri** (21MEI10004), **Sarthak Yadav** (21MEI10036)” who carried out the project work under my supervision.

Certified further that to the best of my knowledge the work reported at this time does not form part of any other project/research work based on which a degree or award was conferred on an earlier occasion on this or any other candidate.

**PROGRAMME CHAIR**

Dr. H. Azath

School of Computing Science & Engineering  
VIT BHOPAL UNIVERSITY

**PROJECT GUIDE**

Dr. Hariharasitaraman.S

School of Computing Science & Engineering  
VIT BHOPAL UNIVERSITY

The Project Exhibition I Examination is held on 30 Sept 2022.

## ACKNOWLEDGEMENT

First and foremost, I would like to thank the Lord Almighty for His presence and immense blessings throughout the project work.

I wish to express my heartfelt gratitude to **Dr. S. Poonkuntran.** Head of the Department, School of Computer Science and Engineering for much of his valuable support encouragement in carrying out this work.

I would like to thank my internal guide **Dr. Hariharasitaraman.S,** for continually guiding and actively participating in my project, giving valuable suggestions to complete the project work.

I would like to thank all the technical and teaching staff of the School of Computer Science and Engineering, who extended directly or indirectly all support.

Last, but not least, I am deeply indebted to my parents who have been the greatest support while I worked day and night for the project to make it a success.

## LIST OF ABBREVIATIONS

Serial no	Abbreviation	Definition
1	DNS	Domain Name System
2	DoS	Denial of Service
3	ARP	Address Resolution Protocol

## LIST OF FIGURES AND GRAPHS

Serial No	Title	Page No.
1	System Architecture	13
2	Code Output	14
3	ARP Cache Poisoned	14
4	Man in the Middle Attack Block Diagram	15
5	ARP Poisoning Block Diagram	16
6	Man in the Middle Attack Visualization	16
7	ARP Poisoning	17
8	DoS Client Side Attack Block Diagram	18
9	DoS Attack Block Diagram	18
10	DNS Spoofing Block Diagram	19

## **Abstract**

In this project, the method to perform various man-in-the-middle attacks is shown in a controlled environment. These attacks are performed using various virtual machines on a single system. The attacks shown are ARP poisoning, DOS and DNS spoofing. A single point of failure of all the three attacks are identified and a rudimentary python script to detect and prevent these attacks is also created and tested under different conditions and the results are noted. This script mainly revolves around the fact that if a single MAC address corresponds to two different IP entries in the ARP cache, then it can be said that the cache is poisoned. For preventing undesired changes in the ARP cache, all the entries of the cache are changed into static entries, as a result these cannot be automatically changed even if anyone sends unsolicited ARP replies.

## TABLE OF CONTENTS

CHAPTE R NO.	TITLE	PAGE NO.
	<b>List of Abbreviations</b> <b>List of figures and graphs</b> <b>Abstract</b>	
1	<b>CHAPTER-1:</b> <b>PROJECT DESCRIPTION AND OUTLINE</b>  1.1 Introduction 1.2 Motivation for the work 1.3 Problem Statement 1.4 Literature Review	8
2	<b>CHAPTER-2:</b> <b>INTRODUCTION TO ANTI-DNS SPOOFER</b>  2.1 Introduction 2.2 Hardware and Software requirements 2.3 Module description 2.4 System Architecture	10
3	<b>CHAPTER-3</b> <b>Program Output and Availability</b>  3.1 Code output 3.2 Project outcome 3.3 Methodology	14
4	<b>CHAPTER-4:</b> <b>TECHNICAL IMPLEMENTATION &amp; ANALYSIS</b>  4.1 ARP Spoof, and DNS Spoofing 4.2 Detection, Recovery, and Prevention	21

5	<b>CHAPTER-5:</b> <b>CONCLUSIONS AND RECOMMENDATION</b> 5.1 Limitations and Solution 5.2 Final Summary	23
6	<b>Reference</b>  Appendix A	25

# **CHAPTER-1:**

## **PROJECT DESCRIPTION AND OUTLINE**

### **1.1 Introduction:**

In this project, detailed instruction to perform various man-in-the middle attacks are discussed. The attacks are simulated in a controlled environment using virtual machines hosted on a single system. The attacks discussed are ARP cache poisoning to get username and password of users, DOS (Denial of Service) and DNS(Domain Name System) cache poisoning. All these attacks require both the victim and the attacker to be in the same Local Area Network. This is because in a LAN, the systems communicate with each other using MAC addresses instead of IP addresses. To make this communication faster, every system in that LAN stores a cache of various IP addresses in the network along with its corresponding MAC address. This cache of IP and MAC is called an ARP cache. Thus the Address Resolution Protocol (ARP) is used to provide mapping between the Network Layer and the Data Link Layer. By successfully performing an ARP cache poisoning attack, the attacker will be placed between the victim and the gateway, as a result all the victim's traffic will pass through the attacker. The attacker can now stop, modify or allow the packets to pass. If the packets are not encrypted, then the attacker can view all the information in the packets including confidential information like username and password. If the attacker blocks all the victim's packets, then the victim will be unable to access the internet which is a denial-of-service (DOS) attack. The attacker can also choose to send fake DNS replies to the victim which makes the victim redirected to any website as wished by the attacker. There are a lot of dangerous man-in-the-middle attacks which require ARP poisoning as the first step of the attack. So we can say that ARP poisoning is the single point of failure in all these attacks, and theoretically, by solving the problem of ARP poisoning, we can solve all the other man-in-the-middle attacks. For detecting and preventing ARP poisoning attacks on the victim's system (not the gateway), we have created an application using python called anti-arp. This is capable of detecting and preventing potential ARP attacks just by analysing the ARP cache of the system. The fact that static ARP entries cannot be changed automatically is used to prevent ARP poisoning by attackers.



### **1.2 Motivation for work:**

DNS Spoofing exploits the victim's computer which results in several frauds being committed. The motivation for this is that through Anti-DNS Spoofing, the attack on the Linux system will be detected, prevented and recovered.

### **1.3 Problem Statement:**

Even though these attacks have a limitation of only working on LAN, these are very common and easily performed and virtually undetectable by a novice. As a result, this is a threat that has to be addressed. So, we have come up with a python program which is capable of detecting, preventing and recovering from an ARP poisoning attack on the victim's linux system by which man in the middle attack can be stopped and dns won't be spoofed.

### **1.4 Literature Review**

- [1] Y. Kim, S. Ahn, N. C. Thang, D. Choi and M. Park, "ARP Poisoning Attack Detection Based on ARP Update State in Software-Defined Networks," 2019 International Conference on Information Networking (ICOIN), Kuala Lumpur, Malaysia, 2019, pp. 366-371, doi: 10.1109/ICOIN.2019.8718158.
- [2] B. Zdrnja, "Malicious JavaScript Insertion through ARP Poisoning Attacks," in IEEE Security & Privacy, vol. 7, no. 3, pp. 72-74, May-June 2009, doi: 10.1109/MSP.2009.72.

## **CHAPTER-2:**

### **RELATED WORK INVESTIGATION**

#### **2.1 Introduction:**

With the current rise of Software-Defined Networking (SDN) and Service Function Chaining (SFC), these networking technologies are opening new doors for vulnerabilities in the field of cyber security, which can be exploited by using trivial methods and past approaches. Hence, detection of such exploitations is necessary considering the use of these technologies in the emerging software industry and their applications. The paper [1] talks about how detection of such exploitations can be done. The paper specifically focuses on ARP Poisoning. The paper tries to exploit the repetitive ARP reply sent by the attacker to detect the respective attack. The paper provides an accurate and subtle solution when compared to other solutions.

This paper [2] gives an in depth view of the ARP poisoning methods and lists down different techniques that can be used to achieve the ARP poisoning for a victim system. A detailed description for implementing the attack has been provided in JavaScript that can be applied to most of the websites. As Man-in-the-middle is the most sorted method to execute an ARP cache poisoning, this paper [3] has presented a method to not only detect the attack but also prevent it. Their method proposes to create an ARP table that contains the replies received from the devices on the network, and to check if all the systems have the same table to verify the authenticity of the data collected. Any unusual behavior in the table would mean that an attack had been implemented, thus detecting the attack. After detection, these replies can be discarded, thus preventing the attack. On the other side, this paper [4] provides a solution that enables authorization for ARP, which was a vulnerability, and added features of detection and preventions of ARP cache poisoning of dynamic IP configuration. To check the pair entry of IP-MAC, a secondary cache is used to implement the internet control management protocol (ICMP). DNS spoofing is one of the many prominent attacks used exploiting the system of a target machine. This paper [5] provides a new approach to DNS Spoofing by creating a fake DNS server and redirecting the requests sent to the Destination IP through a fake DNS server, thus enabling to intercept the messages.

Due to the dependency of DNS on UDP, it becomes an open battleground for Distributed Denial of Service (DDoS) attack. This paper [6] proposes a method to detect and mitigate the DNS spoofing attack for SDNs, by using technologies like CAAuth and OpenFlow protocol to manage the packets and the data transferred between the networks. This allows proper surveillance of the data exchanged between the network participants. By the year 2016, where SSLStrip was a tool that showed the vulnerability of HTTPS websites to the world, adding DNS spoofing to the mixture made it deadly for target machines. Hence, this paper [7] suggests a technique that allows the victim to detect the stripping of HTTPS website or a more technical term would be downgrading HTTPS website to HTTP. Thus, allowing the victim to avoid such websites before providing any personal credentials to the attacker who is imposing as the website owner.

Most of the DoS detection methods generally go for making a ping to the website or server, and check if the response has unexpected latency or no response from the server. These signal them as that a DoS attack has been initiated. However, this paper [8] uses an aegis algorithm to find if a specific server or a website has undergone a DoS attack or not. Not only does it detect the attack but it also helps moderate the attack. As DoS attacks have many flavors, one of the types are DoS atomic attacks. Though these types of attacks are classified, the paper [9] somehow tries to present two examples of the atomic DoS attack to give a deeper knowledge of the topic. The paper [10] talks about an optimal method of detecting these DoS attacks with higher precision and notifies the owner, before a considerable damage is made to the server.

[11] Provides a method to secure systems against MITM.

## **2.2 Hardware and Software Requirements:**

Hardware:

- Intel i5-9300H processor
- 4 GB RAM

Software:

- VMware workstation 15
- ISO file of kali, Ubuntu and windows 7
- Ettercap, arpspoof, setoolkit and browser (included with kali)
- Python 3.7
- Any text editor

## **2.3 MODULE DESCRIPTION**

The detection mechanism is called anti-ARP, a simple python program that browses different IP addresses in the ARP cache and checks if any two IP addresses have the same MAC addresses or not. If they have the same MAC address, which means the ARP cache is poisoned. Once the poisoning is confirmed, the user is prompted with a message saying that the ARP cache has been poisoned. Along with this message, the attacker's MAC address is also displayed.

For prevention, we designed an algorithm and coded it in python. This algorithm does the task of making all the entries in the ARP cache as static because of the fact that they cannot be altered by the attacker once made static. As a result, no matter how many fake ARP replies are sent to the victim, its ARP cache won't be changed.

Once we have detected that ARP cache is poisoned, we can execute the recovery algorithm which is also coded in python. The main idea of the recovery process is copying a clean set of entries into the cache or altering a specific entry in the cache. The clean ARP cache entries are stored when the application is started, provided that the cache is cleaned while starting it. If the user knows the MAC address of the gateway, or any other node in the network, they can enter the ARP entry using one of the features of the recovery system.

## 2.4 System Architecture

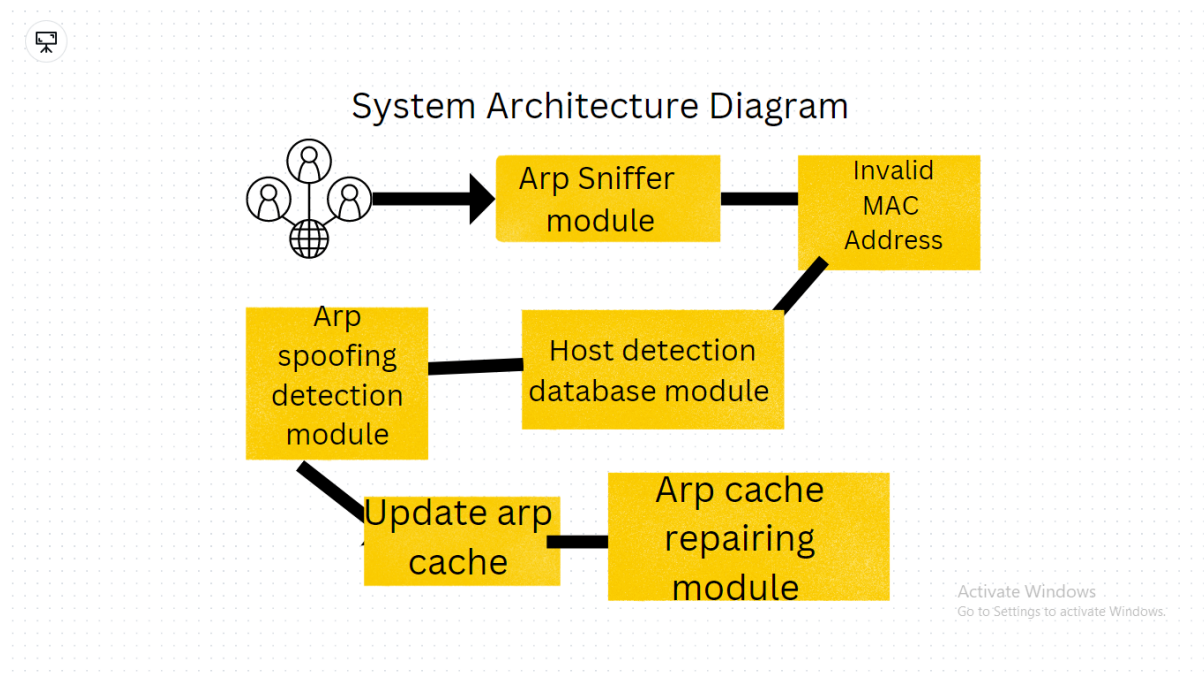
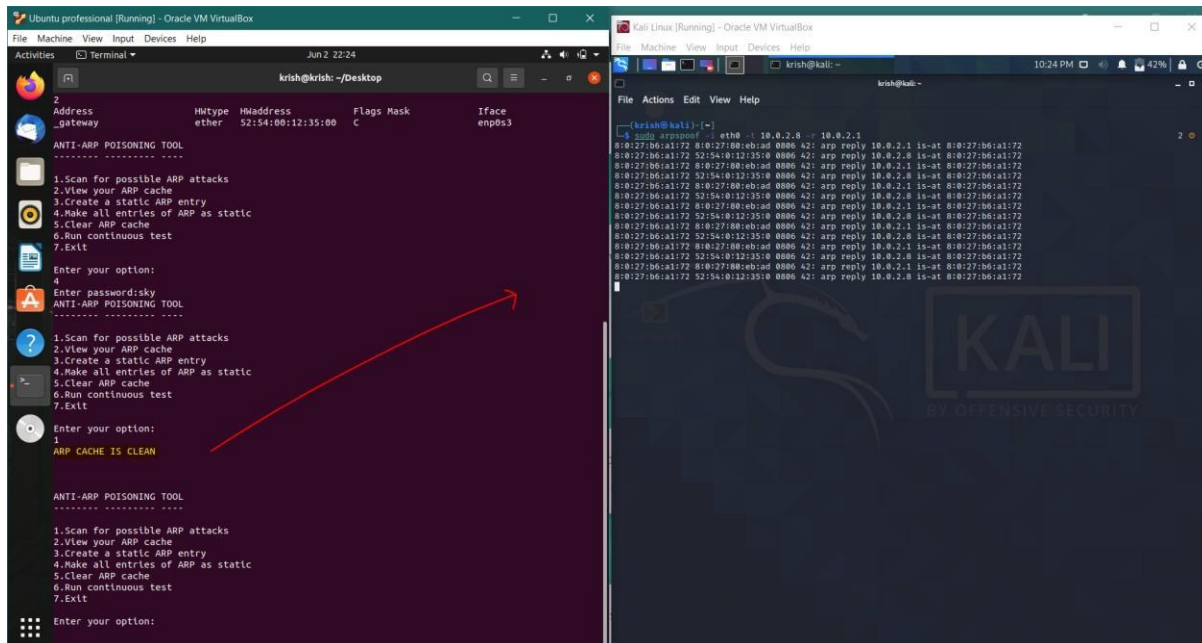


Figure 1: System Architecture

## CHAPTER-3:

### Program Output and Availability

#### 3.1 Code Output:



```
krish@krish: ~/Desktop
ANTI-ARP POISONING TOOL
-----
1.Scan for possible ARP attacks
2.View your ARP cache
3.Create a static ARP entry
4.Make all entries of ARP as static
5.Clear ARP cache
6.Run continuous test
7.Exit

Enter your option:
4
Enter password:sky
ANTI-ARP POISONING TOOL
-----
1.Scan for possible ARP attacks
2.View your ARP cache
3.Create a static ARP entry
4.Make all entries of ARP as static
5.Clear ARP cache
6.Run continuous test
7.Exit

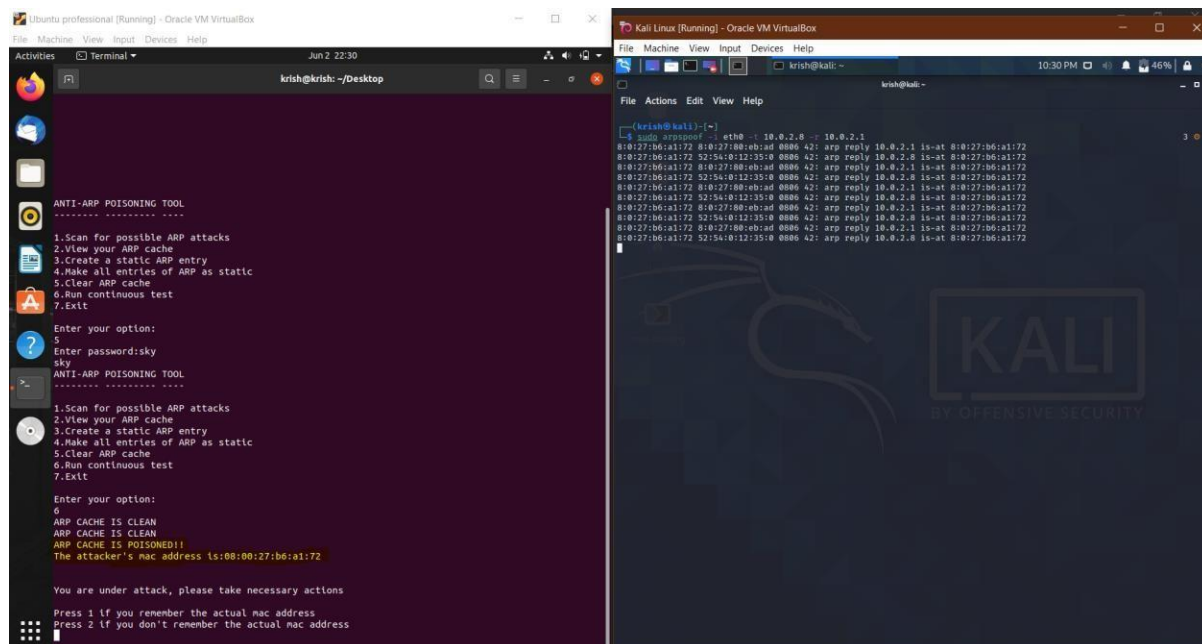
Enter your option:
1
ARP CACHE IS CLEAN

ANTI-ARP POISONING TOOL
-----
1.Scan for possible ARP attacks
2.View your ARP cache
3.Create a static ARP entry
4.Make all entries of ARP as static
5.Clear ARP cache
6.Run continuous test
7.Exit

Enter your option:
```

```
krish@kali: ~
$ sudo arpspoof -i eth0 -r 10.0.2.8 -r 10.0.2.1
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.1 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.1 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.8 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.8 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.1 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.1 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.8 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.8 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.1 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.1 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.8 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.8 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.1 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.1 is-at 8:01:27:b6:a1:72
```

Figure 2: Code Output



```
krish@krish: ~/Desktop
ANTI-ARP POISONING TOOL
-----
1.Scan for possible ARP attacks
2.View your ARP cache
3.Create a static ARP entry
4.Make all entries of ARP as static
5.Clear ARP cache
6.Run continuous test
7.Exit

Enter your option:
5
Enter password:sky
ANTI-ARP POISONING TOOL
-----
1.Scan for possible ARP attacks
2.View your ARP cache
3.Create a static ARP entry
4.Make all entries of ARP as static
5.Clear ARP cache
6.Run continuous test
7.Exit

Enter your option:
6
ARP CACHE IS CLEAN
ARP CACHE IS CLEAN
ARP CACHE IS POISONED!!
The attacker's mac address is:08:00:27:b6:a1:72

You are under attack, please take necessary actions
Press 1 if you remember the actual mac address
Press 2 if you don't remember the actual mac address
```

```
krish@kali: ~
$ sudo arpspoof -i eth0 -r 10.0.2.8 -r 10.0.2.1
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.1 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.8 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.1 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.8 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.1 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.1 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.8 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.8 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.1 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.1 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.8 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.8 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.1 is-at 8:01:27:b6:a1:72
8:01:27:b6:a1:72 8:01:27:b6:a1:72 0000 42: arp reply 10.0.2.1 is-at 8:01:27:b6:a1:72
```

Figure 3: ARP Cache Poisoned

#### 3.2 Project Outcome:

A Python program that browses different IP addresses in the ARP cache and checks if any two IP addresses have the same MAC addresses or not. If they have the same MAC address, which means the ARP cache is poisoned. Once the poisoning is confirmed, the user is prompted with a message saying that the ARP cache has been poisoned. Along with this message, the attacker's MAC address is also displayed.

### 3.3 Methodology:

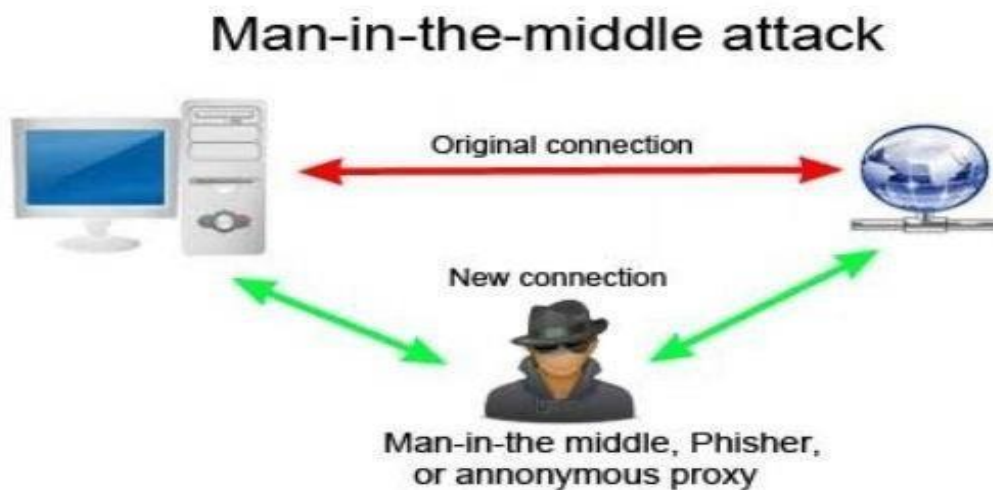


Figure 4: Man-in-the-middle Attack Block Diagram

The aim of this project is to explain what is Man In the Middle attack and demonstrate the various types of MITM attacks namely:-

**A RP Poisoning/Spoofing**:- ARP stands for Address Resolution Protocol is a networking protocol that enables communications in a particular network by linking the network and data link layer. So, ARP spoofing is a type of attack wherein the attacker sends forged APR replies to the victim and as a result, the victim updates its ARP cache with the fake value sent by the attacker, hence the

attacker places itself into the communication between the server and the client without any notice of either if the endpoint devices. Hence to demonstrate this ARP attack, we performed an attack to fetch the Username and password which the victim enters on a particular website. The tool used to perform this is ettercap. Ettercap is an open source network security tool for man-in-the-middle attack on LAN.

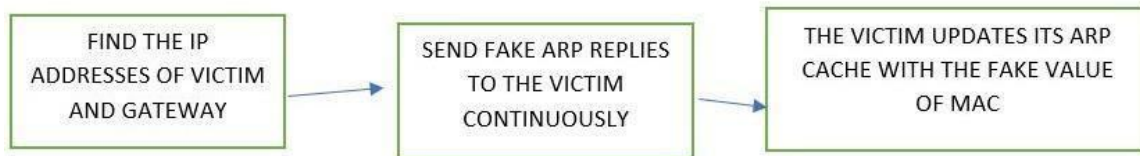


Figure 5: ARP poisoning Block Diagram

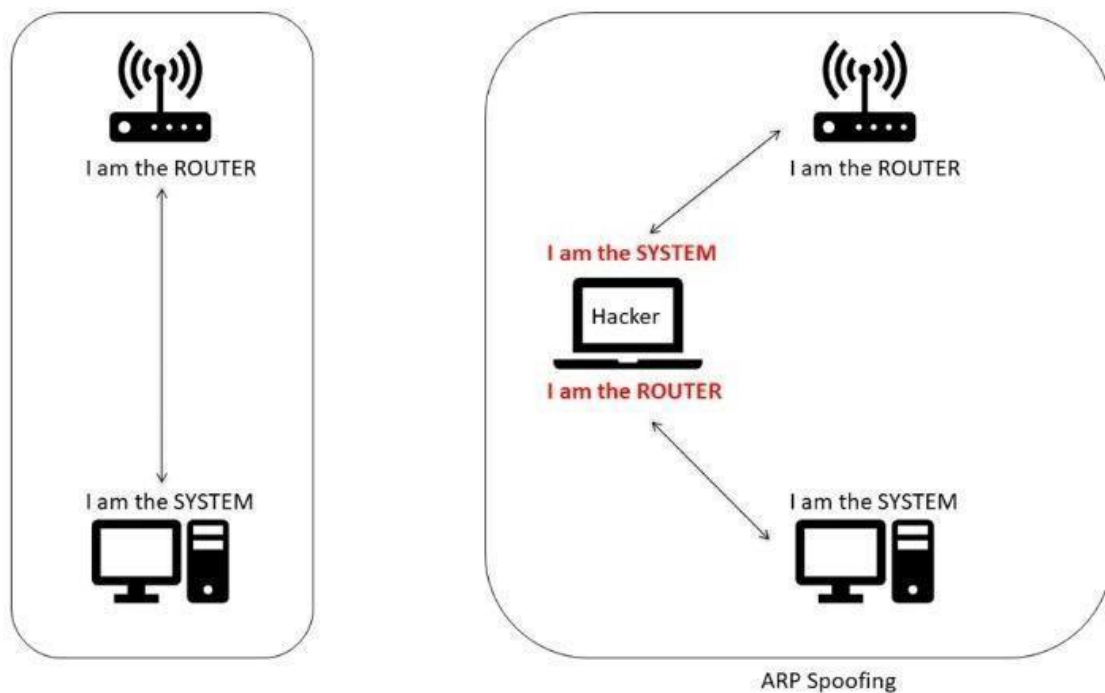


Figure 6: Man-in-the-middle attack visualization



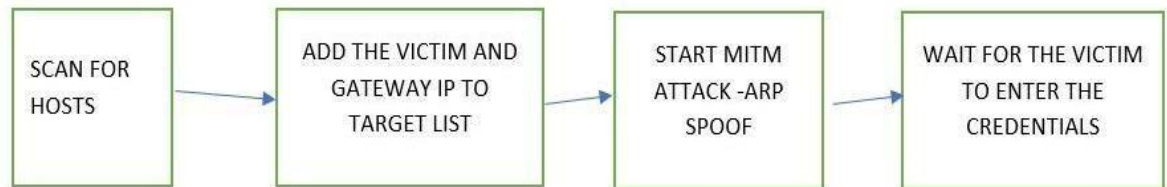


Figure 7: ARP poisoning

**DoS Attack:-** DoS stands for Denial of Service. A DoS attack is an attack that is meant to shut down a particular machine or network hence making it inaccessible to others on the network. This attack is accomplished by flooding the target with traffic or sending some sort of information that triggers a crash. We have tried a non-conventional DoS attack by attacking the user's system instead of a server. By depriving the victim to send packets, in essence we would have performed a DoS attack on the server side. As a result of this attack the whole of the internet and intranet would not be accessible by the victim. For demonstrating this we performed a simple attack by poisoning the arp cache of the victim with the attacker's MAC address. Hence we can alter the packet forwarding for the victim to 0, it denies the access of the whole of internet to the victim as all the packets sent by the victim is

being blocked by the attacker. If we set the packet forwarding to 1, and then the victim can access the internet as usual.

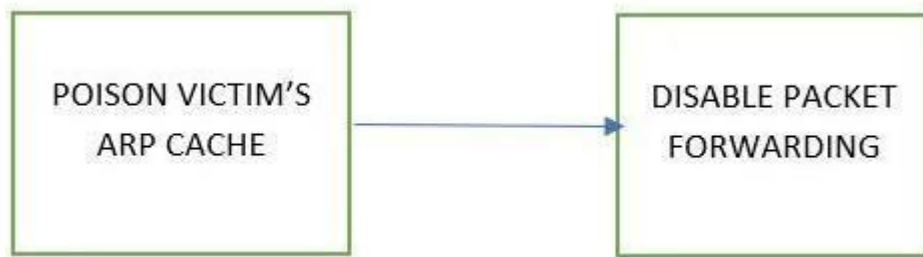


Figure 8: DoS Client side Attack Block diagram

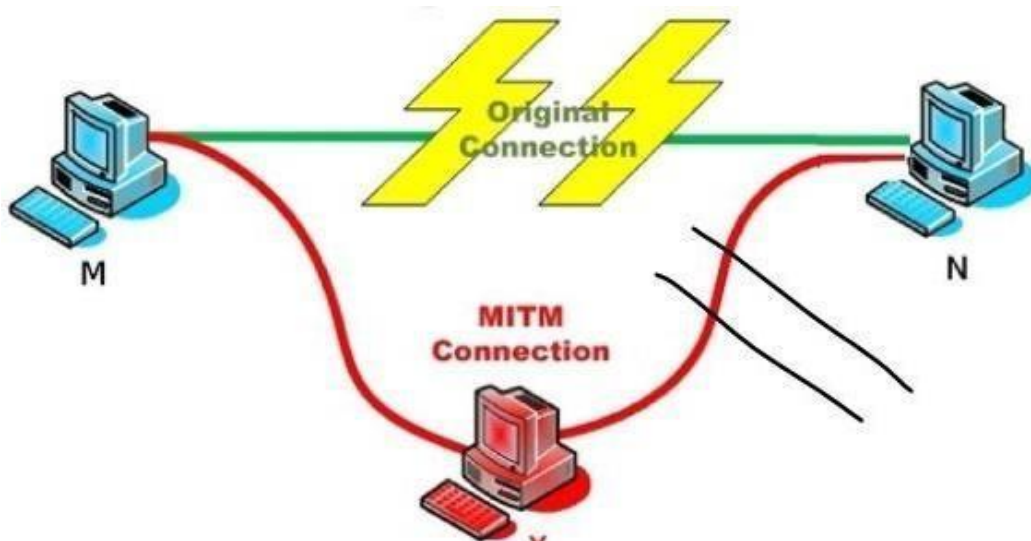


Figure 9: DoS Attack Block Diagram

**DNS Spoof:-** DNS stands for Domain Name Server. So, DNS spoofing is an attack wherein the attacker can redirect the victim to another website which is not intended to be visited by the victim by injecting a fake DNS record in the DNS cache. As a result of these fake DNS entries, the victim

can be redirected to any IP of the attacker's choosing. This attack can be easily performed in the Kali Linux environment using various inbuilt tools. One such tool that we used is Ettercap. Using ettercap, we first poison the victim's ARP cache with our MAC address. Then we search for available hosts in the network and add the gateway as TARGET2 and the victim as TARGET1 and start unified sniffing. Finally we add the dns\_spoof plugin that is available in ettercap and wait for some time for it to start. Once we get the message that the target website's DNS has been spoofed, we can check the victim machine and try opening the website in it. We observe that the victim is redirected to some other website even though the address bar says that it is the original website.

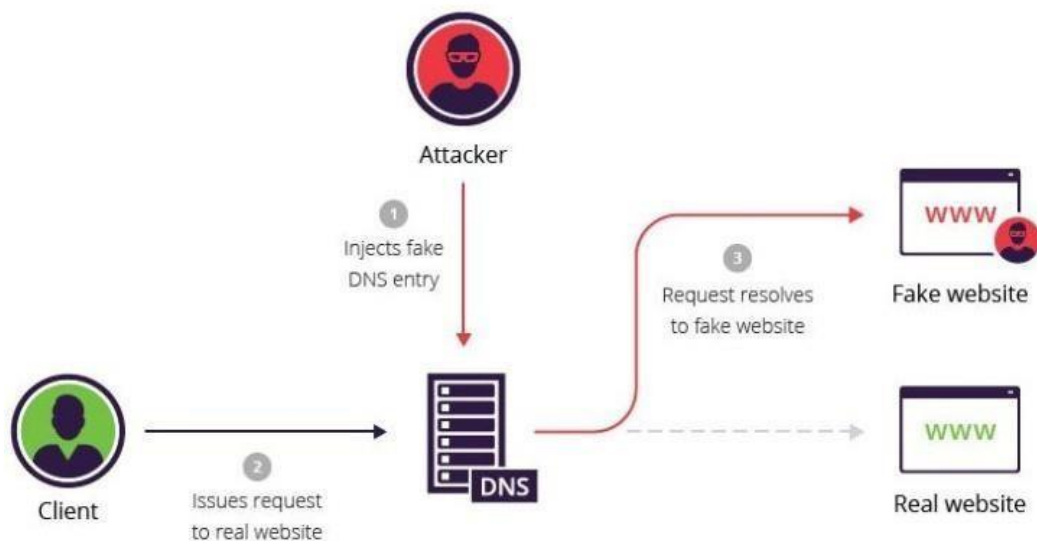


Figure 10: DNS Spoofing Block Diagram

We have also devised an algorithm for preventing, detecting and recovering the ARP poisoning on a particular machine.

1. **Detection**:- The detection mechanism is called anti-ARP, a simple python program that browses different IP addresses in the ARP cache and checks if any two IP addresses have the same MAC addresses or not. If they have the same MAC address, which means the ARP cache is poisoned. Once the poisoning is confirmed, the user is prompted with a message saying that the ARP cache has been poisoned. Along with this message, the attacker's MAC address is also displayed.
  
2. **Prevention** - For prevention, we designed an algorithm and coded it in python. This algorithm does the task of making all the entries in the ARP cache as static because of the fact that they cannot be altered by the attacker once made static. As a result, no matter how many fake ARP replies are sent to the victim, its ARP cache won't be changed.
  
3. **Recovery**:- Once we have detected that ARP cache is poisoned, we can execute the recovery algorithm which is also coded in python. The main idea of the recovery process is copying a clean set of entries into the cache or altering a specific entry in the cache. The clean ARP cache entries are stored when the application is started, provided that the cache is cleaned while starting it. If the user knows the MAC address of the gateway, or any other node in the network, they can enter the ARP entry using one of the features of the recovery system.

## CHAPTER-4:

### TECHNICAL IMPLEMENTATION & ANALYSIS

#### 4.1 ARP Spoof and DNS Spoofing:

We have performed all the attacks and carried out the prevention, detection and recovery mechanisms successfully. We can draw the following inference from each attack:-

1. **ARP spoof (User Id and Password attack):-** In this attack, we scanned for hosts on the local network and then added the victim as one of the targets and the gateway as the other target on ettercap and started ARP poisoning attack. We observe that the arp cache of the victim is poisoned with the attacker's MAC address and hence, the attacker acts as the man in the middle who can see and control the packets transmitted in the network to and from the victim. Hence, whenever the victim enters the credentials or any other information, we get all of the information on ettercap. This was tested on demo.testfire.net and the victim's credentials were successfully captured.
2. **DNS Spoofing:-** In this attack, Ettercap plays a very important role in enabling the ARP spoofing and DNS spoofing. We first scan the network for hosts and subsequently add the victim and the gateway as the targets. In the ettercap config, we enable the commands for Linux as we are working in a Linux based environment. We also used the in-built plugin available in ettercap for DNS spoofing. After waiting for some time, we can see that the victim is redirected to some other website rather than on the original one. When the victim enters their credentials in the fake website, we are able to obtain this information.

We did a bit of research and brainstorming and devised algorithms for detection, prevention and recovery from an ARP spoof attack and coded them in python.

#### 4.2 Detection, Prevention and Recovery:

1. **Detection:-** For detection, we implemented a simple searching algorithm

to search the ARP cache of the victim for 2 IP addresses with the same MAC address. If any such entry is found, that means the ARP cache is poisoned. This was pretty straightforward and hence we got the desired outcome.

2. **Prevention**:- After researching about the various methods of prevention, we came to a conclusion that any ARP poisoning attack can be prevented just by making all entries in the cache as static. By doing so, we make sure that the attacker cannot further make any changes to the ARP cache by sending fake ARP replies.
3. **Recovery**:- The main idea of the recovery process is copying a clean set of entries into the cache or altering a specific entry in the cache. The clean ARP cache entries are stored when the application is started, provided that the cache is clean while starting it. If the user knows the MAC address of the gateway, or any other node in the network, they can modify the ARP entry.

Hence the experiments were successful and we were able to demonstrate the different types of ARP poisoning attacks using inbuilt tools in Kali Linux and programming skills. We were also able to show the prevention, detection and recovery procedures which are essential from an Information Security point of view.

## CHAPTER 5

### CONCLUSIONS AND RECOMMENDATION

#### 5.1 Limitations and Solution:

Common risks of DNS poisoning and spoofing:

- Data Theft
- Malware Infection
- Halted Security Updates
- Censorship

DNS Spoofing poses several risks, each putting your devices and personal data in harm's way.

Data Theft can be particularly lucrative for DNS spoof attackers. Banking websites and popular online retailers are easily spoofed, meaning any password, credit card or personal information may be compromised. The redirects would be phishing websites designed to collect your info. Anti DNS protects its user from all these risks.

#### 5.2 Final Summary:

In this project, we have successfully implemented ARP poisoning, DOS, and DNS poisoning attacks. We have also recorded the procedure and explained it step-by-step in great detail. Even though these attacks have a limitation of only working on LAN, these are very common and easily performed and virtually undetectable by a novice. As a result, this is a threat that has to be addressed. So, we have come up with an app “anti-arp” which is capable of detecting, preventing and recovering from an ARP poisoning attack on the victim's Linux system. This is written in python and mainly focuses on analyzing the ARP cache of the system to detect attacks. Also there are functions which have the capability of performing continuous scans after a certain regular interval of time.

In the future, this application can be made more robust and less resource intensive at the same time. Anti-arp is coded to run in linux systems only. Support for running it on windows and mac os can also be done. Necessary permissions can be

given which enable anti-arp to start executing when the system boots up. The code can also be improved by adding various Object Oriented Programming features.



## Reference

- [3] Y. Kim, S. Ahn, N. C. Thang, D. Choi and M. Park, "ARP Poisoning Attack Detection Based on ARP Update State in Software-Defined Networks," 2019 International Conference on Information Networking (ICOIN), Kuala Lumpur, Malaysia, 2019, pp. 366-371, doi: 10.1109/ICOIN.2019.8718158.
- [4] B. Zdrnja, "Malicious JavaScript Insertion through ARP Poisoning Attacks," in IEEE Security & Privacy, vol. 7, no. 3, pp. 72-74, May-June 2009, doi: 10.1109/MSP.2009.72.
- [5] S. Kumar and S. Tapaswi, "A centralized detection and prevention technique against ARP poisoning," Proceedings Title: 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), Kuala Lumpur, 2012, pp. 259-264, doi: 10.1109/CyberSec.2012.6246087.
- [6] D. R. Rupal, D. Satasiya, H. Kumar and A. Agrawal, "Detection and prevention of ARP poisoning in dynamic IP configuration," 2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, 2016, pp. 1240-1244, doi: 10.1109/RTEICT.2016.7808030.
- [7] M. Janbeglou, M. Zamani and S. Ibrahim, "Redirecting outgoing DNS requests toward a fake DNS server in a LAN," 2010 IEEE International Conference on Software Engineering and Service Sciences, Beijing, 2010, pp. 29-32, doi: 10.1109/ICSESS.2010.5552339.
- [8] N. Sahri and K. Okamura, "Collaborative Spoofing Detection and Mitigation -- SDN Based Looping Authentication for DNS Services," 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), Atlanta, GA, 2016, pp. 565-570, doi: 10.1109/COMPSAC.2016.6.
- [9] A. A. Maksutov, I. A. Cherepanov and M. S. Alekseev, "Detection and prevention of DNS spoofing attacks," 2017 Siberian Symposium on Data Science and Engineering (SSDSE), Novosibirsk, 2017, pp. 84-87, doi: 10.1109/SSDSE.2017.8071970.
- [10] D. S. N. Mary and A. T. Begum, "An Algorithm for Moderating DoS

Attack in Web Based Application," 2017 International Conference on Technical Advancements in Computers and Communications (ICTACC), Melmaurvathur, 2017, pp. 26-31, doi: 10.1109/ICTACC.2017.17.

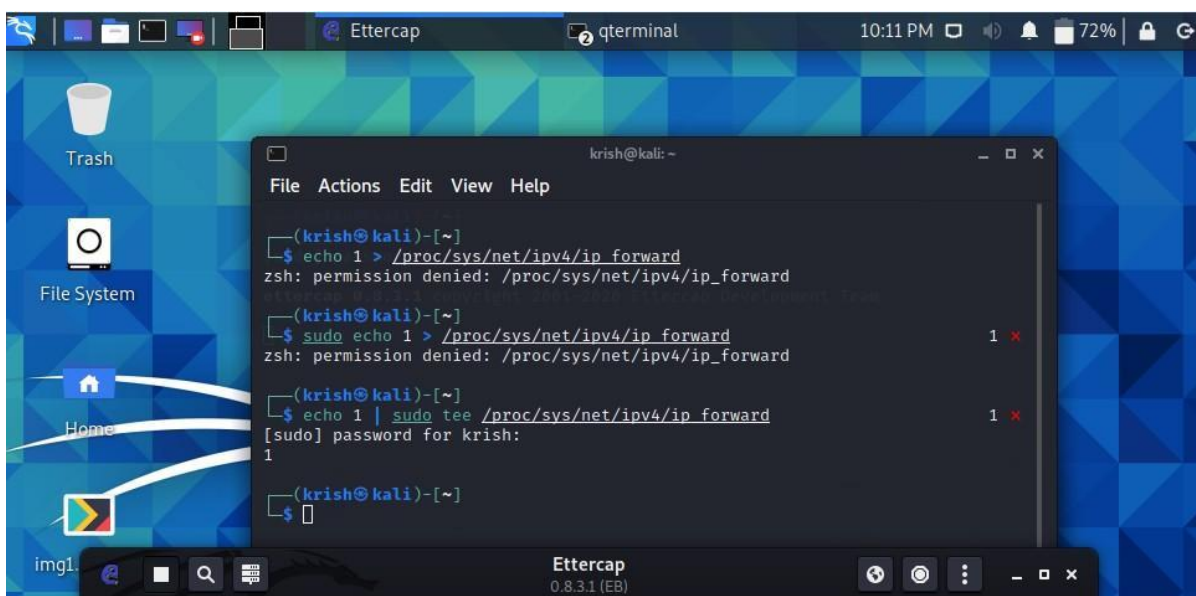
[11] S. Xin, X. Chen, H. Tang and N. Zhu, "Research on DoS Atomic Attack Oriented to Attack Resistance Test," 2008 IEEE International Conference on Networking, Sensing and Control, Sanya, 2008, pp. 1747-1752, doi: 10.1109/ICNSC.2008.4525506.

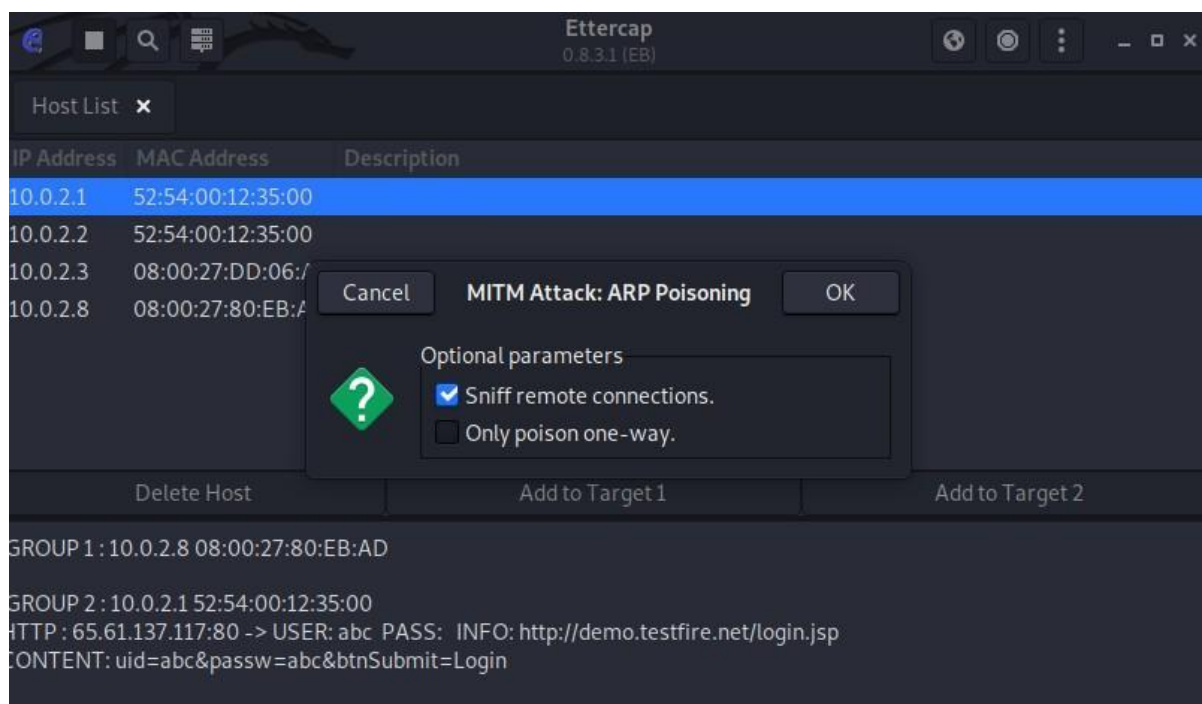
[12] M. A. Saleh and A. Abdul Manaf, "Optimal specifications for a protective framework against HTTP-based DoS and DDoS attacks," 2014 International Symposium on Biometrics and Security Technologies (ISBAST), Kuala Lumpur, 2014, pp. 263-267, doi: 10.1109/ISBAST.2014.7013132.

[13] M. Ordean and M. Giurgiu, "Towards securing client-server connections against man-in-the-middle attacks," 2012 10th International Symposium on Electronics and Telecommunications, Timisoara, 2012, pp. 127-130, doi: 10.1109/ISETC.2012.6408076.

## Appendix A

### Password Attack

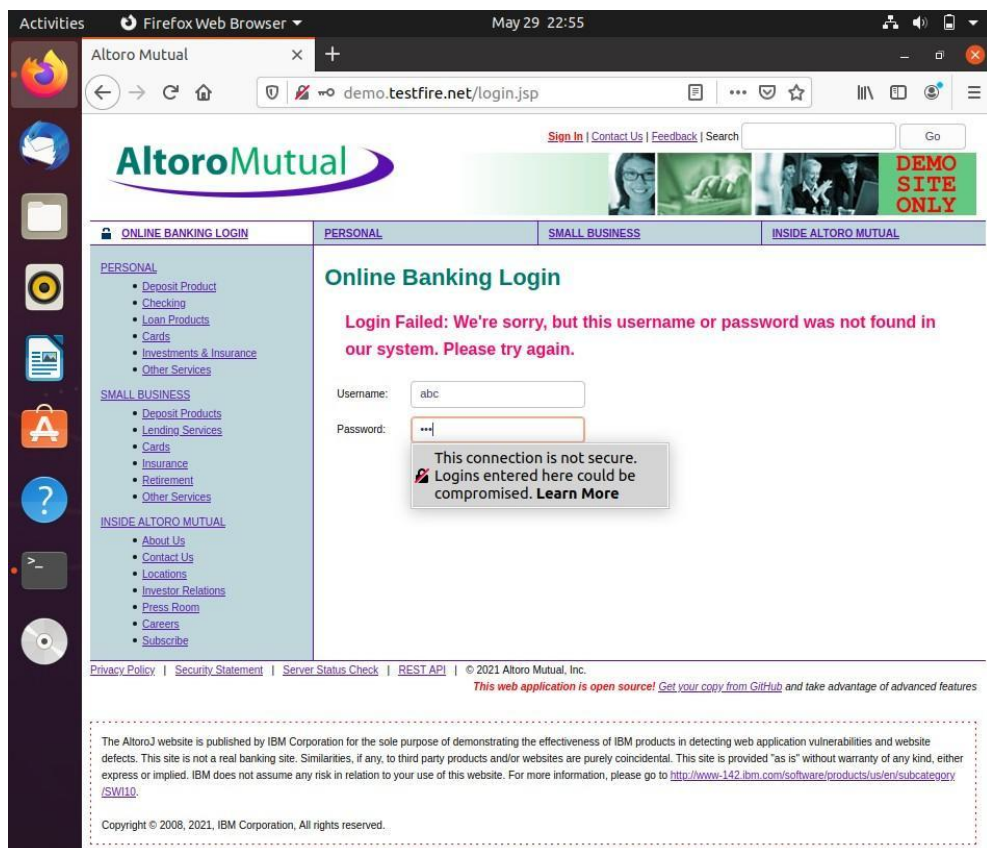
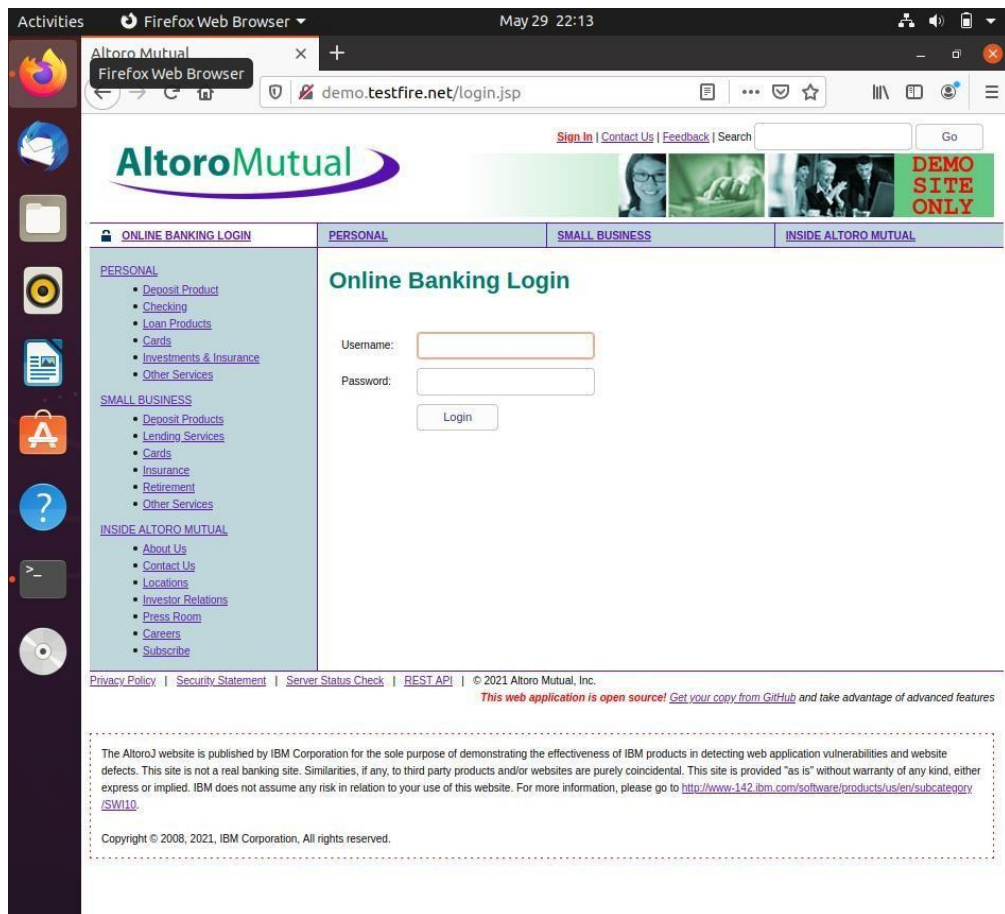


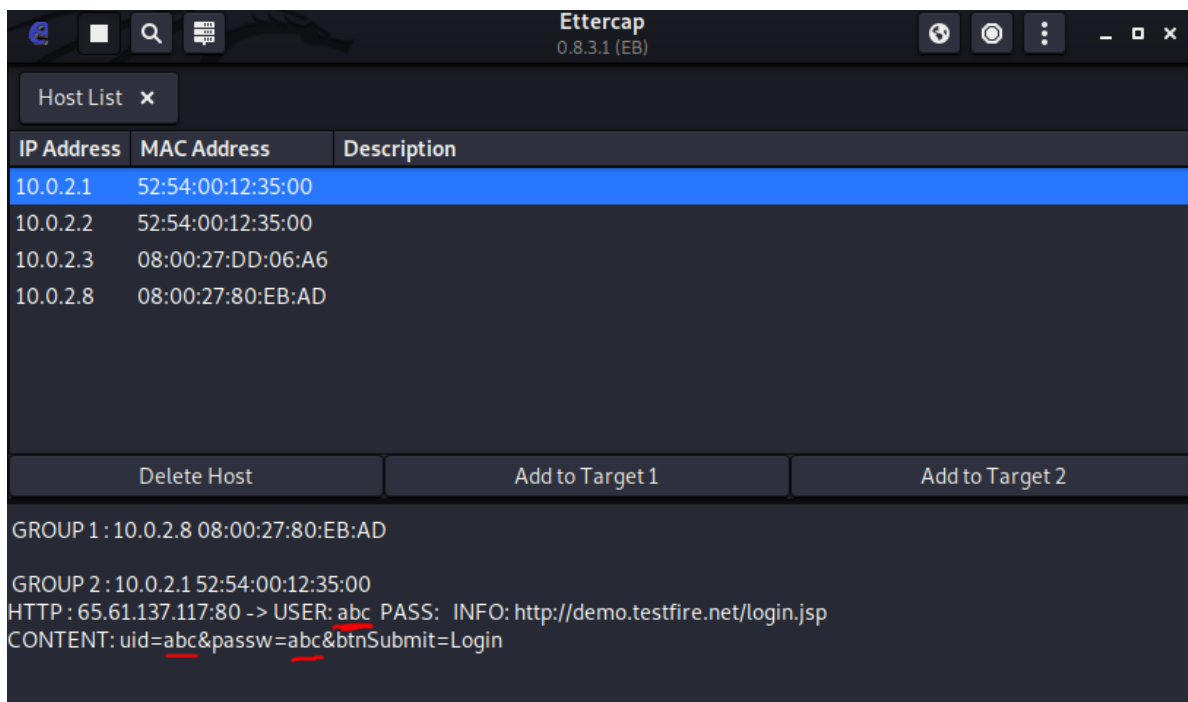


```
krish@krish:~$ arp -a
? (10.0.2.3) at 08:00:27:dd:06:a6 [ether] on enp0s3
_gateway (10.0.2.1) at 52:54:00:12:35:00 [ether] on enp0s3
krish@krish:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.8 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fe80::52cc:214d:538f:d67b prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:80:eb:ad txqueuelen 1000 (Ethernet)
    RX packets 3740 bytes 3900462 (3.9 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 2230 bytes 230302 (230.3 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 494 bytes 50793 (50.7 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 494 bytes 50793 (50.7 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

krish@krish:~$
```





## DNS Spoofing

```
(krish@kali)-[~]
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fe80::a00:27ff:feb6:a172 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:b6:a1:72 txqueuelen 1000 (Ethernet)
    RX packets 3 bytes 710 (710.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 11 bytes 1142 (1.1 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 8 bytes 400 (400.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 8 bytes 400 (400.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

```
(krish@kali)-[~]
$ ipcalc 10.0.2.15
Address: 10.0.2.15 00001010.00000000.00000010. 00001111
Netmask: 255.255.255.0 = 24 11111111.11111111.11111111. 00000000
Wildcard: 0.0.0.255 00000000.00000000.00000000. 11111111
=>
Network: 10.0.2.0/24 00001010.00000000.00000010. 00000000
HostMin: 10.0.2.1 00001010.00000000.00000010. 00000001
HostMax: 10.0.2.254 00001010.00000000.00000010. 11111110
Broadcast: 10.0.2.255 00001010.00000000.00000010. 11111111
Hosts/Net: 254 Class A, Private Internet

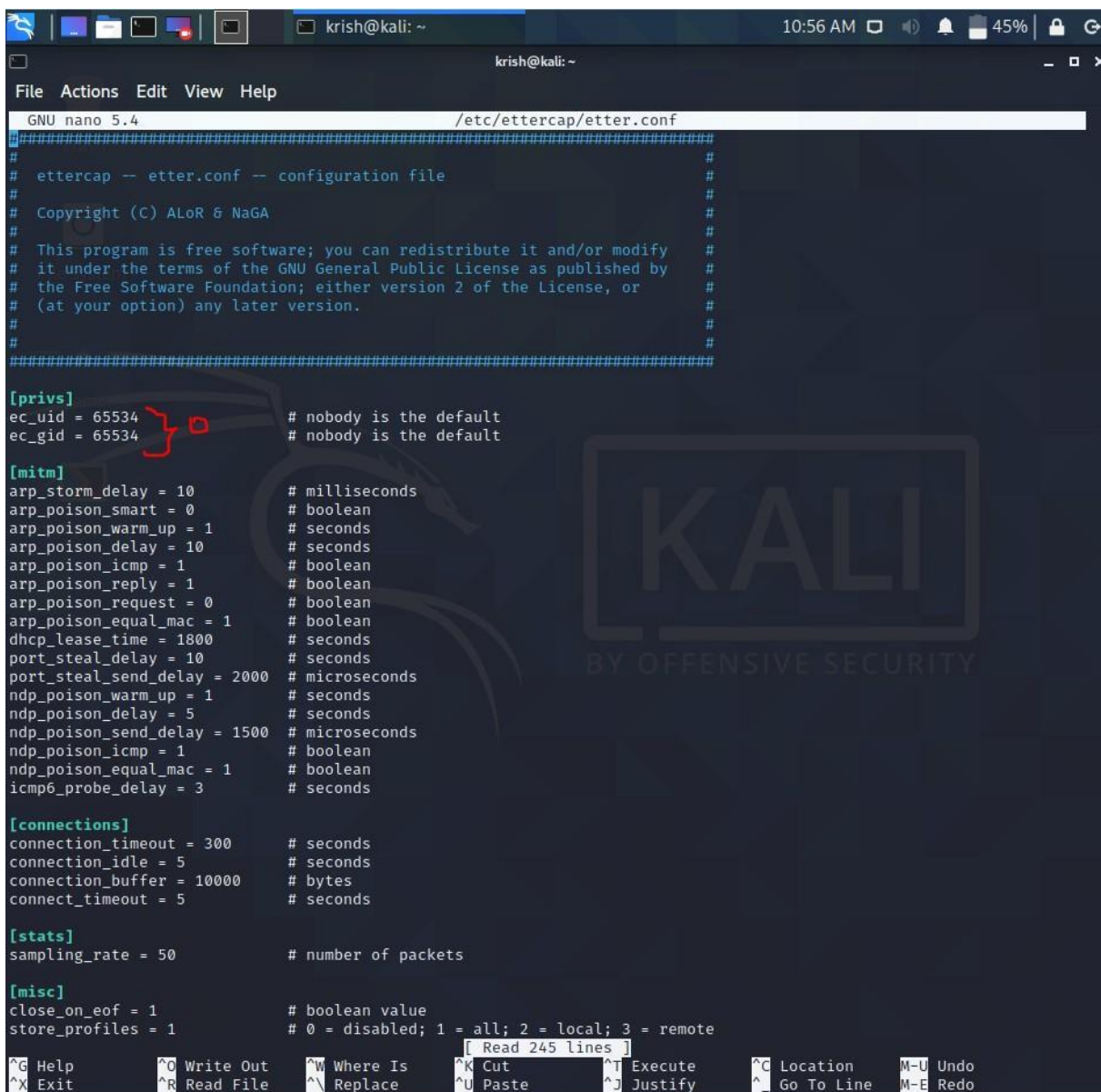
(krish@kali)-[~]
$
```



```
(krish@kali)~$ nmap -sP 10.0.2.0/24
Starting Nmap 7.91 ( https://nmap.org ) at 2021-05-30 10:54 IST
Nmap scan report for 10.0.2.1
Host is up (0.0025s latency).
Nmap scan report for 10.0.2.15
Host is up (0.00041s latency).
Nmap done: 256 IP addresses (2 hosts up) scanned in 3.06 seconds

(krish@kali)~$ route -n
Kernel IP routing table
Destination      Gateway          Genmask         Flags Metric Ref    Use Iface
0.0.0.0          10.0.2.1        0.0.0.0         UG    100    0      0 eth0
10.0.2.0         0.0.0.0         255.255.255.0   U     100    0      0 eth0

(krish@kali)~$
```



```
krish@kali: ~
10:56 AM 45%
File Actions Edit View Help
GNU nano 5.4 /etc/ettercap/etter.conf
#####
#
# ettercap -- etter.conf -- configuration file
#
# Copyright (C) ALOR & NaGA
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
#####

[privs]
ec_uid = 65534 # nobody is the default
ec_gid = 65534 # nobody is the default

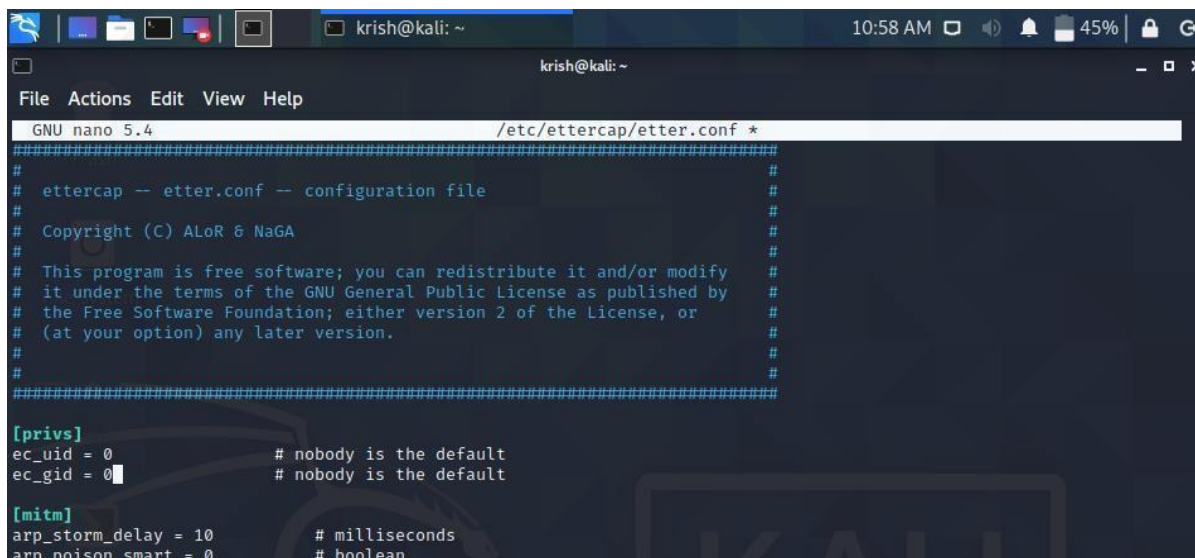
[mitm]
arp_storm_delay = 10 # milliseconds
arp_poison_smart = 0 # boolean
arp_poison_warm_up = 1 # seconds
arp_poison_delay = 10 # seconds
arp_poison_icmp = 1 # boolean
arp_poison_reply = 1 # boolean
arp_poison_request = 0 # boolean
arp_poison_equal_mac = 1 # boolean
dhcp_lease_time = 1800 # seconds
port_steal_delay = 10 # seconds
port_steal_send_delay = 2000 # microseconds
ndp_poison_warm_up = 1 # seconds
ndp_poison_delay = 5 # seconds
ndp_poison_send_delay = 1500 # microseconds
ndp_poison_icmp = 1 # boolean
ndp_poison_equal_mac = 1 # boolean
icmp6_probe_delay = 3 # seconds

[connections]
connection_timeout = 300 # seconds
connection_idle = 5 # seconds
connection_buffer = 10000 # bytes
connect_timeout = 5 # seconds

[stats]
sampling_rate = 50 # number of packets

[misc]
close_on_eof = 1 # boolean value
store_profiles = 1 # 0 = disabled; 1 = all; 2 = local; 3 = remote

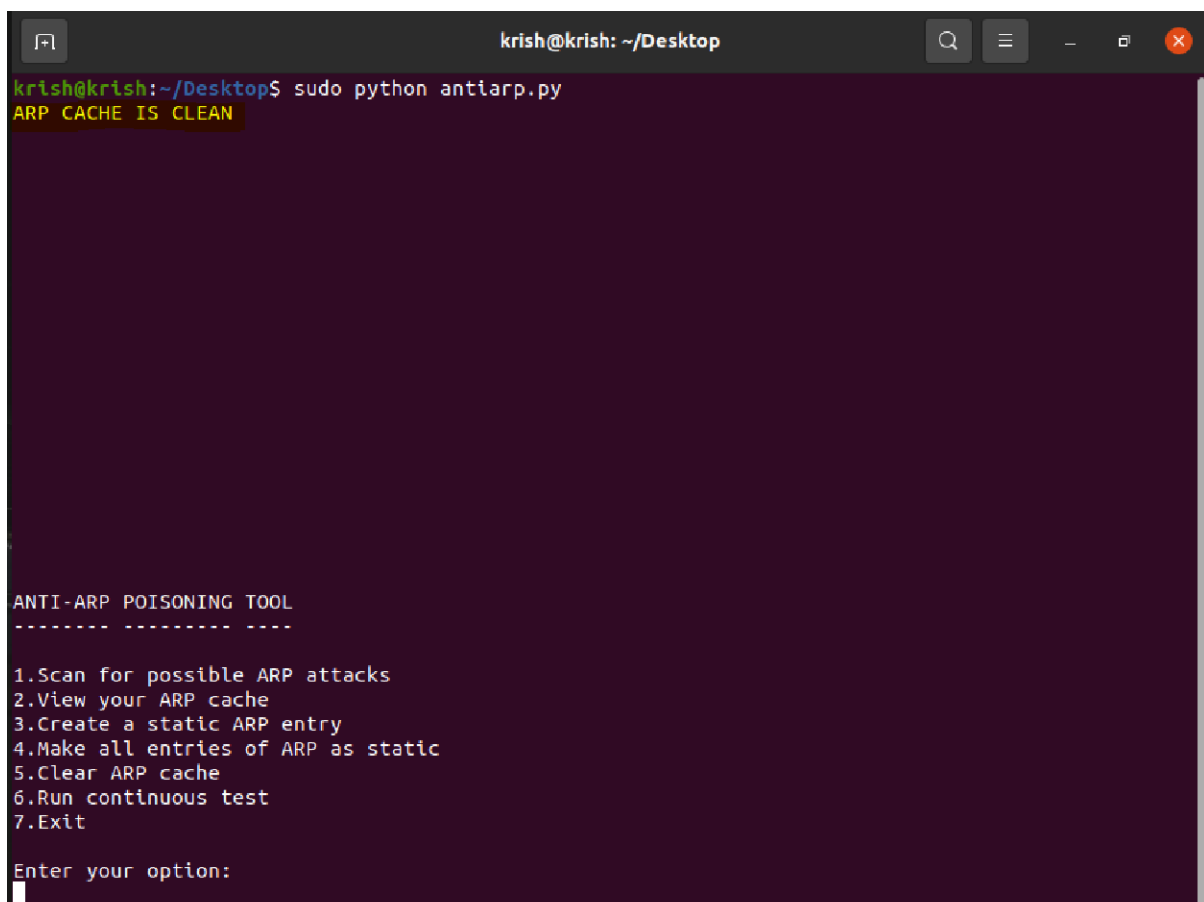
[ Read 245 lines ]
^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo
^X Exit ^R Read File ^_ Replace ^U Paste ^J Justify ^_ Go To Line M-E Redo
```



A screenshot of a terminal window on a Kali Linux system. The window title is 'krish@kali: ~'. The terminal shows the nano text editor editing the file '/etc/ettercap/etter.conf'. The editor's status bar at the top indicates 'GNU nano 5.4' and the file path. The configuration file content is as follows:

```
#####  
#  
# ettercap -- etter.conf -- configuration file  
#  
# Copyright (C) ALOR & NaGA  
#  
# This program is free software; you can redistribute it and/or modify  
# it under the terms of the GNU General Public License as published by  
# the Free Software Foundation; either version 2 of the License, or  
# (at your option) any later version.  
#  
#####  
  
[privs]  
ec_uid = 0          # nobody is the default  
ec_gid = 0          # nobody is the default  
  
[mitm]  
arp_storm_delay = 10      # milliseconds  
arp_poison_smart = 0      # boolean
```

## Detection



A screenshot of a terminal window on a Kali Linux system. The window title is 'krish@krish: ~/Desktop'. The terminal shows the command 'sudo python antiarp.py' being executed, which results in the output 'ARP CACHE IS CLEAN'. Below this, the script displays a menu titled 'ANTI-ARP POISONING TOOL' with a list of options:

```
krish@krish:~/Desktop$ sudo python antiarp.py  
ARP CACHE IS CLEAN  
  
ANTI-ARP POISONING TOOL  
-----  
  
1.Scan for possible ARP attacks  
2.View your ARP cache  
3.Create a static ARP entry  
4.Make all entries of ARP as static  
5.Clear ARP cache  
6.Run continuous test  
7.Exit  
  
Enter your option:  
█
```

# Prevention

```
ANTI-ARP POISONING TOOL
-----

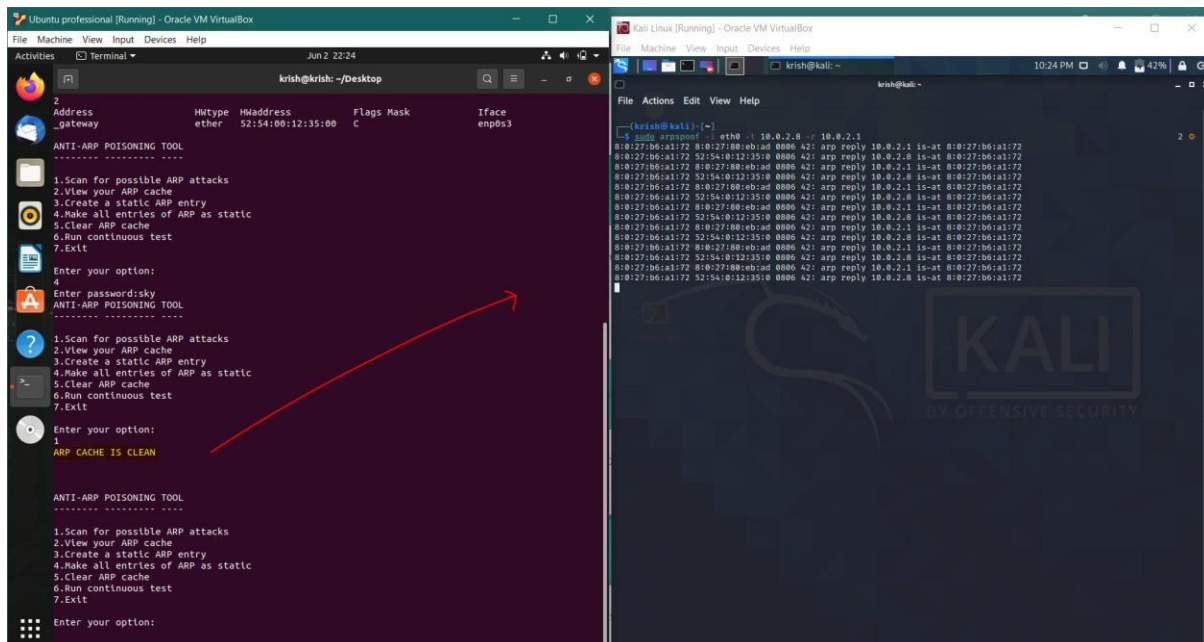
1.Scan for possible ARP attacks
2.View your ARP cache
3.Create a static ARP entry
4.Make all entries of ARP as static
5.Clear ARP cache
6.Run continuous test
7.Exit

Enter your option:
2
Address          HWtype  HWaddress      Flags Mask    Iface
_gateway         ether    52:54:00:12:35:00    C             enp0s3

ANTI-ARP POISONING TOOL
-----

1.Scan for possible ARP attacks
2.View your ARP cache
3.Create a static ARP entry
4.Make all entries of ARP as static
5.Clear ARP cache
6.Run continuous test
7.Exit

Enter your option:
4
Enter password:sky
```



# Recovery





