# **Experiment No. 12**

**Aim:** Detect ARP spoofing using open source tool ARPWATCH.

**Software Required: ARPWATCH** Tool

## Theory:

**Arpwatch Commands and Usage**

To watch a specific interface, type the following command with ‗-i‗ and device name.



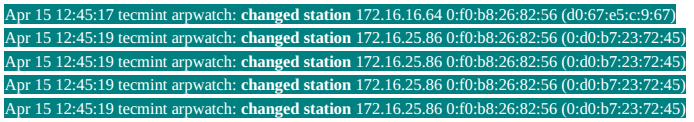
So, whenever a new MAC is plugged or a particular IP is changing his MAC address on the network, you will notice syslog entries at ‗/var/log/syslog‗ or ‗/var/log/message‗ file.



## Sample Output



The above output displays new workstation. If any changes are made, you will get following output.



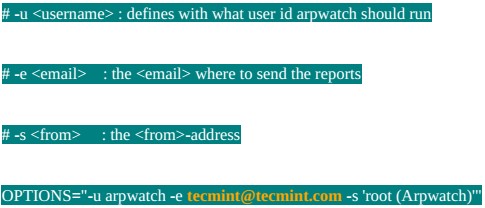
You can also check current ARP table, by using following command.



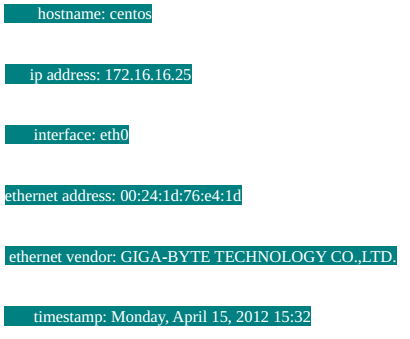
## Sample Output:



If you want to send alerts to your custom email id, then open the main configuration file‗/etc/sysconfig/arpwatch‗ and add the email as shown below.



Here is an example of an email report, when a new MAC is connected on.



# **Experiment No. 14**

**Aim:** Implement a code to simulate buffer overflow attack.

**Hardware / Software Required :** Stack Guard compiler.

## Theory:

A buffer overflow, or buffer overrun, is an anomaly where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory locations. This is a special case of the violation of memory safety.

A buffer overflow occurs when a program or process tries to store more data in a buffer (temporary data storage area) than it was intended to hold.

Buffer overflow: Code:

#include <stdio.h> #include <string.h>

int main(void)

{

char buff[15]; int pass = 0;

printf("\n Enter the password : \n"); gets(buff);

if(strcmp(buff, "thecorrectpaswd"))

{

printf ("\n Wrong Password \n");

}

else

{

printf ("\n Correct Password \n"); pass = 1;

}

if(pass)

{

/\* Now Give root or admin rights to user\*/ printf ("\n Root privileges given to the user \n");

}

return 0;

}

Output :

>>administrator@PIIT-05:~/Desktop/me CS pracs$ gcc -Wall -fno-stack-protector bufferoverflow.c -o

>>bufferoverflow

The above command deactivates the default GC Compiler's flag which detects Stack Smashing

>>administrator@PIIT-05:~/Desktop/me CS pracs$ ./bufferoverflow

>>Enter the password :

thewrong

Wrong Password

>>administrator@PIIT-05:~/Desktop/me CS pracs$ ./bufferoverflow

>>Enter the password :

thecorrectpaswd Correct Password

Root privileges given to the user

administrator@PIIT-05:~/Desktop/me CS pracs$ ./bufferoverflow Enter the password :

thewrongpasswordentered Wrong Password

Root privileges given to the user

Here, the entered password length is above the permissible length with wrong contents still the user is given the ROOT PRIVILEDGES. This demonstrates the Buffer Overflow.

**Experiment No. 15**

**Aim:** Set up IPSEC under LINUX.

**Hardware / Software Required :** L2TP/IPsec VPN client setup

## Theory:

**Internet Protocol Security (IPsec)** is a protocol suite for securing Internet Protocol (IP) communications by authenticating and encrypting each IP packet of a communication session. IPsec includes protocols for establishing mutual authentication between agents at the beginning of the session and negotiation of cryptographic keys to be used during the session. IPsec can be used in protecting data flows between a pair of hosts (host-to-host), between a pair of security gateways (network-to-network), or between a security gateway and a host (network-to-host).

Internet Protocol security (IPsec) uses cryptographic security services to protect communications over Internet Protocol (IP) networks. IPsec supports network-level peer authentication, data origin authentication, data integrity, data confidentiality (encryption), and replay protection.

IPsec is an end-to-end security scheme operating in the Internet Layer of the Internet Protocol Suite, while some other Internet security systems in widespread use, such as Transport Layer Security (TLS) and Secure Shell (SSH), operate in the upper layers at Application layer. Hence, only IPsec protects any application traffic over an IP network. Applications can be automatically secured by IPsec at the IP layer.

The following commands will add the werner-jaeger PPA into your repo's, and then install the 'l2tp-ipsec-vpn' package:

## >>sudo apt-add-repository ppa:werner-jaeger/ppa-werner-vpn

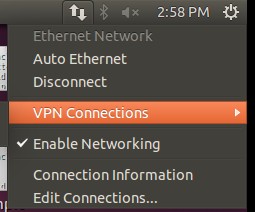
**>>sudo apt-get update**

## >>sudo apt-get install l2tp-ipsec-vpn

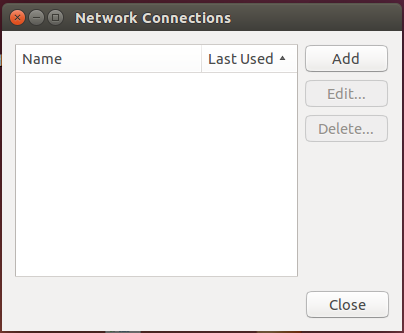
* Now, we will whitelist our system tray which will allow our newly installed package to show up on our system tray:

## >>gsettings set com.canonical.Unity.Panel systray-whitelist "['all']"

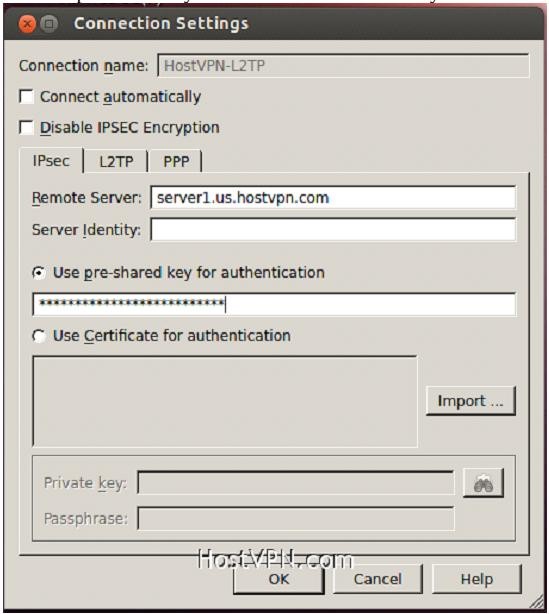
* + After whitelisting our system tray, it's imperative that you reboot/restart your machine.
  + Once your machine has rebooted, click on the new icon, and click 'Edit Connections ...' from the menu.



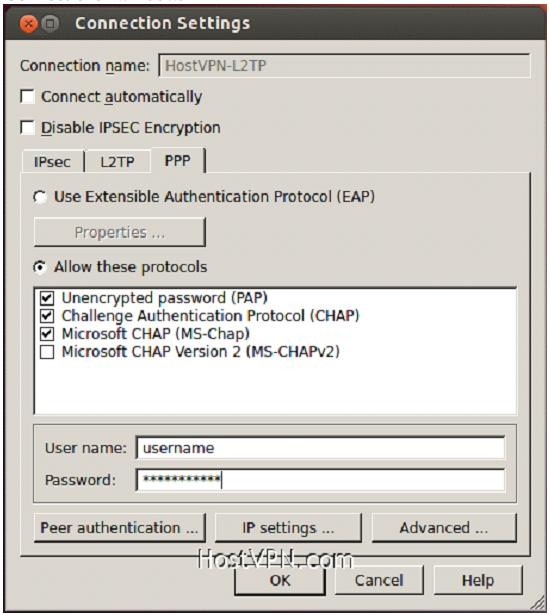
* + This will show the "VPN Connections" window. Click the "Add ..." button and set the connection name to anything you'd like, e.g. "HostVPN-L2TP", and click "OK".



* + Now select your newly added connection, and click "Edit ...".
  + On the IPSec tab, set the remote server to the server name from your HostVPN e-mail. Select the "Use pre-shared key for authentication" and enter your PSK from the HostVPN e-mail.



* + On the PPP tab, select "Allow these protocols", and ensure all are selected except "Microsoft CHAP Version 2 (MS-CHAPv2)". Fill in the "User name:" and "Password:" fields with your HostVPN username and password, and then click "OK". Now click "Close" on the "VPN Connections" window.



* + Click on the L2TP/IPSec VPN icon in the systray again and click on the connection name that we just created.



**Aim:** Install IDS (e.g. SNORT) and study the logs.

**Hardware / Software Required:** Snort tool.

## Theory:

Snort is an intrusion detection system written by Martin Roesch. Snort is a lightweight network intrusion detection system, capable of performing real-time traffic analysis and packet logging on IP networks. It can perform protocol analysis, content searching/matching and can be used to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, and much more. Snort uses a flexible rules language to describe traffic that it should collect or pass, as well as a detection engine that utilizes a modular plugin architecture.

Snort can be configured to run in three modes:

* ***Sniffer mode****,* which simply reads the packets off of the network and displays them for you in a continuous stream on the console (screen).
* ***Packet Logger mode****,* which logs the packets to disk.
* ***Network Intrusion Detection System (NIDS) mode****,* the most complex and configurable configuration, which allows Snort to analyze network traffic for matches against a user-defined rule set and performs several actions based upon what it sees.
* ***Inline mode****,* which obtains packets from iptables instead of from libpcap and then causes iptables to drop or pass packets based on Snort rules that use inline-specific rule types

### Sniffer Mode:

If you just want to print out the TCP/IP packet headers to the screen (i.e. sniffer mode), try following, this command will run Snort and just show the IP and TCP/UDP/ICMP headers, nothing else.

## ./snort –v

If you want to see the application data in transit, try the following, this instructs Snort to display the packet data as well as the headers.

## ./snort –vd

If you want an even more descriptive display, showing the data link layer headers, do this:

## ./snort -vde

### Packet Logger Mode:

If you want to record the packets to the disk, you need to specify a logging directory and Snort will automatically know to go into packet logger mode:

## ./snort -dev -l ./log

Of course, this assumes you have a directory named log in the current directory. If you don't, Snort will exit with an error message. When Snort runs in this mode, it collects every packet it sees and places it in a directory hierarchy based upon the IP address of one of the hosts in the datagram.

If you just specify a plain -l switch, you may notice that Snort sometimes uses the address of the remote computer as the directory in which it places packets and sometimes it uses the local host address. In order to log relative to the home network, you need to tell Snort which network is the home network:

## ./snort -dev -l ./log -h 192.168.1.0/24

This rule tells Snort that you want to print out the data link and TCP/IP headers as well as application data into the directory ./log, and you want to log the packets relative to the 192.168.1.0 class C network. All incoming packets will be recorded into subdirectories of the log directory, with the directory names being based on the address of the remote (non- 192.168.1) host.

**Note:** Note that if both the source and destination hosts are on the home network, they are logged to a directory with a name based on the higher of the two port numbers or, in the case of a tie, the source address.

Once the packets have been logged to the binary file, you can read the packets back out of the file with any sniffer that supports the tcpdump binary format (such as tcpdump or Ethereal). Snort can also read the packets back by using the -r switch, which puts it into playback mode. Packets from any tcpdump formatted file can be processed through Snort in any of its run modes. For example, if you wanted to run a binary log file through Snort in sniffer mode to dump the packets to the screen, you can try something like this:

## ./snort -dv -r packet.log

You can manipulate the data in the file in a number of ways through Snort's packet logging and intrusion detection modes, as well as with the BPF interface that's available from the command line. For example, if you only wanted to see the ICMP packets from the log file, simply specify a BPF filter at the command line and Snort will only see the ICMP packets in the file:

## ./snort -dvr packet.log icmp

### Network Intrusion Detection System (NIDS) mode:

To enable Network Intrusion Detection System (NIDS) mode so that you don't record every single packet sent down the wire, try this:

## ./snort -dev -l ./log -h 192.168.1.0/24 -c snort.conf

where snort.conf is the name of your rules file. This will apply the rules configured in the snort.conf file to each packet to decide if an action based upon the rule type in the file should be taken. If you don't specify an output directory for the program, it will default to

/var/log/snort.

One thing to note about the last command line is that if Snort is going to be used in a long term way as an IDS, the -v switch should be left off the command line for the sake of speed. The screen is a slow place to write data to, and packets can be dropped while writing to the display.

It's also not necessary to record the data link headers for most applications, so you can usually omit the -e switch, too.

## ./snort -d -h 192.168.1.0/24 -l ./log -c snort.conf

This will configure Snort to run in its most basic NIDS form, logging packets that trigger rules specified in the snort.conf in plain ASCII to disk using a hierarchical directory structure (just like packet logger mode).

### Inline Mode:

Snort 2.3.0 RC1 integrated the intrusion prevention system (IPS) capability of Snort Inline into the official Snort project. Snort Inline obtains packets from iptables instead of libpcap and then uses new rule types to help iptables pass or drop packets based on Snort rules.

There are three rule types you can use when running Snort with Snort Inline:

* **drop** - The drop rule type will tell iptables to drop the packet and log it via usual Snort means.
* **reject** - The reject rule type will tell iptables to drop the packet, log it via usual Snort means, and send a TCP reset if the protocol is TCP or an icmp port unreachable if the protocol is UDP.
* **sdrop** - The sdrop rule type will tell iptables to drop the packet. Nothing is logged.

When using a reject rule, there are two options you can use to send TCP resets:

* You can use a RAW socket (the default behavior for Snort Inline), in which case you must have an interface that has an IP address assigned to it. If there is not an interface with an IP address assigned with access to the source of the packet, the packet will be logged and the reset packet will never make it onto the network.
* You can also now perform resets via a physical device when using iptables. We take the indev name from ip\_queue and use this as the interface on which to send resets. We no longer need an IP loaded on the bridge, and can remain pretty stealthy as the config layer2\_resets in snort\_inline.conf takes a source MAC address which we substitue for the MAC of the bridge.

For example:

## config layer2resets

tells Snort Inline to use layer2 resets and uses the MAC address of the bridge as the source MAC in the packet, and:

## config layer2resets: 00:06:76:DD:5F:E3

will tell Snort Inline to use layer2 resets and uses the source MAC of 00:06:76:DD:5F:E3 in the reset packet.

* The command-line option **-disable-inline-initialization** can be used to not initialize IPTables when in inline mode. To be used with command-line option -T to test for a valid configuration without requiring opening inline devices and adversely affecting traffic flow.

# **Experiment No. 16**

**Aim:** Use of iptables in linux to create firewalls

**Theory:**

IPtables are the tables provided by the Linux kernel firewall (implemented as different Netfilter modules) and the chains and rules it stores. Different kernel modules and programs are currently used for different protocols; iptables applies to IPv4, ip6tables to IPv6, arptables to ARP, and ebtables to Ethernet frames.

iptables requires elevated privileges to operate and must be executed by user root, otherwise it fails to function. On most Linux systems, iptables is installed as /usr/sbin/iptables and documented in its man pages which can be opened using man iptables when installed. It may also be found in /sbin/iptables, but since iptables is more like a service rather than an "essential binary", the preferred location remains /usr/sbin.

1. To drop all traffic:

# sudo iptables -P INPUT DROP

# sudo iptables -P OUTPUT DROP

# sudo iptables -P FORWARD DROP # sudo iptables -L -v –n

1. Only Block Incoming Traffic

To drop all incoming / forwarded packets, but allow outgoing traffic, # sudo iptables -P INPUT DROP

# sudo iptables -P FORWARD DROP # sudo iptables -P OUTPUT ACCEPT

# sudo iptables -A INPUT -m state --state NEW,ESTABLISHED -j ACCEPT # sudo iptables -L -v –n

1. Block Outgoing IPaddress host -t a hostname

sudo iptables -A OUTPUT -d outgoing ipaddress -j DROP

1. Block or Allow ICMP ping request

sudo iptables -A INPUT -p icmp --icmp-type echo-request -j DROP/ACCEPT