Spring 2015

Information Security Project

Instructor:
Prof Alvaro Cardenas
University of Texas at Dallas
Department of Computer Science
Spring 2015

Team:
Nilesh Gupta – nxg130030
Partha De – pxd141430
Shazia Aftab – sxa141930
Swapnil Hasabe – sdh140430

*Picture Source: http://www.learnonline.edu/

Index

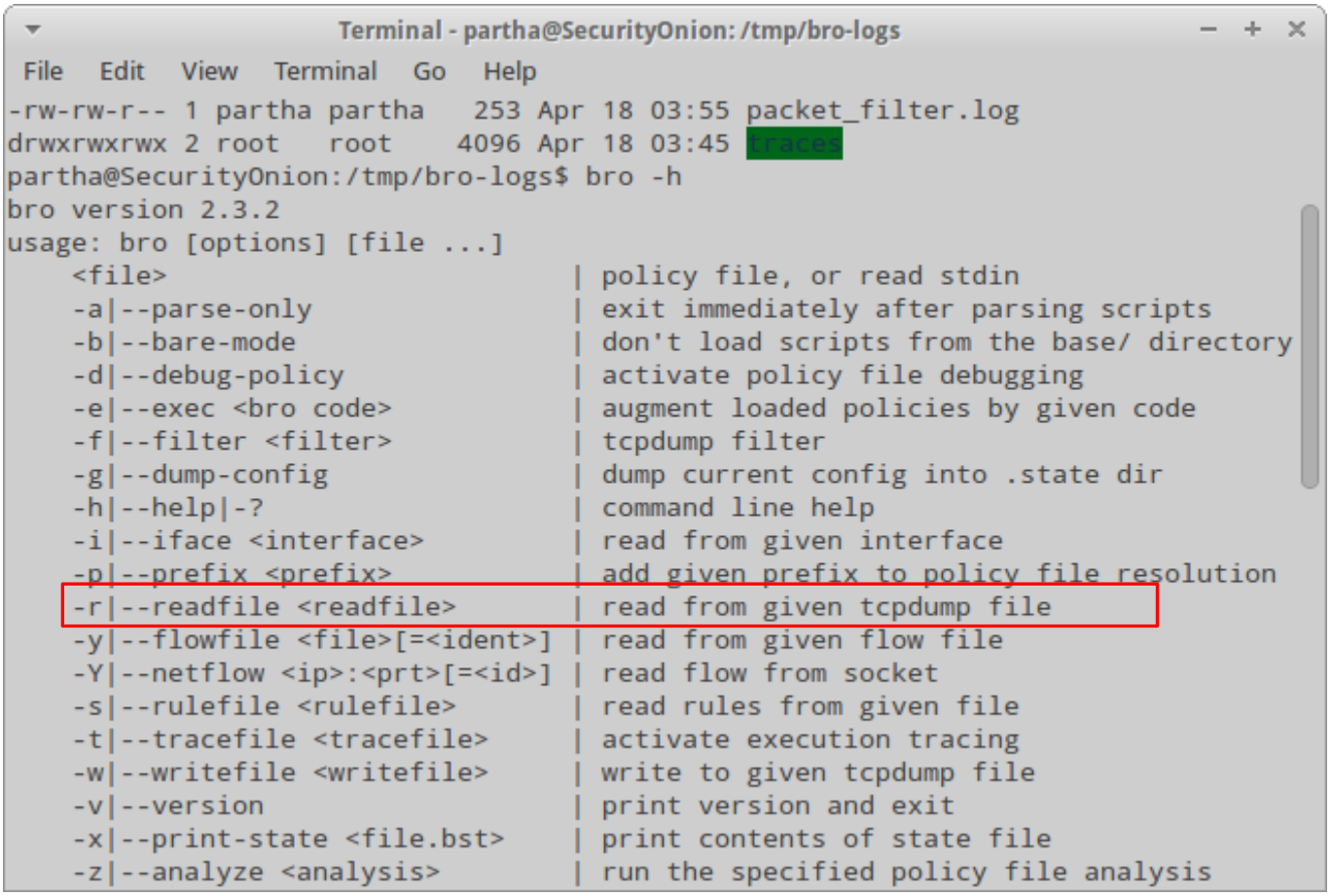
1.Learning how to use Bro

Ans. 1-A: https://www.bro.org/bro-workshop-2011/solutions/logs/index.html

Exercise: Understanding and Examining Bro Logs: While operating on and protecting a system, Bro produces huge amount of log files. The administrators may find it difficult sometimes to gather useful information as the enormous amount of data is written in file system. In this section we have learned about some very useful utilities that can be used to retrieve data as desired.

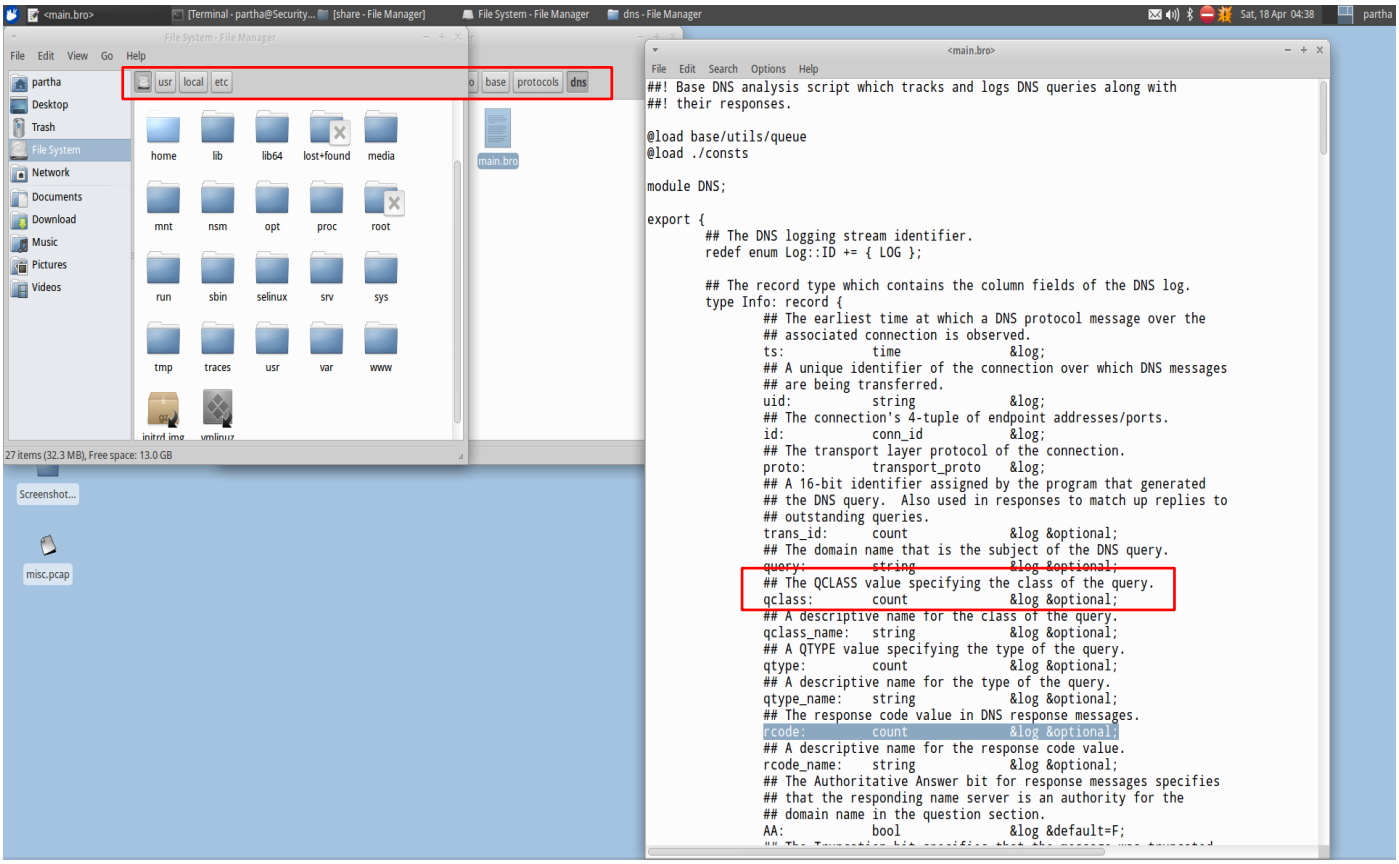
Part-1: Generating Logs: We are provided with an http.pcap file generated by a Bro system. ‘ bro - r ’ option reads from the tcpdump file and generates multiple log files.

Without a meaningful readable log files, users can’t do much in monitoring the system.

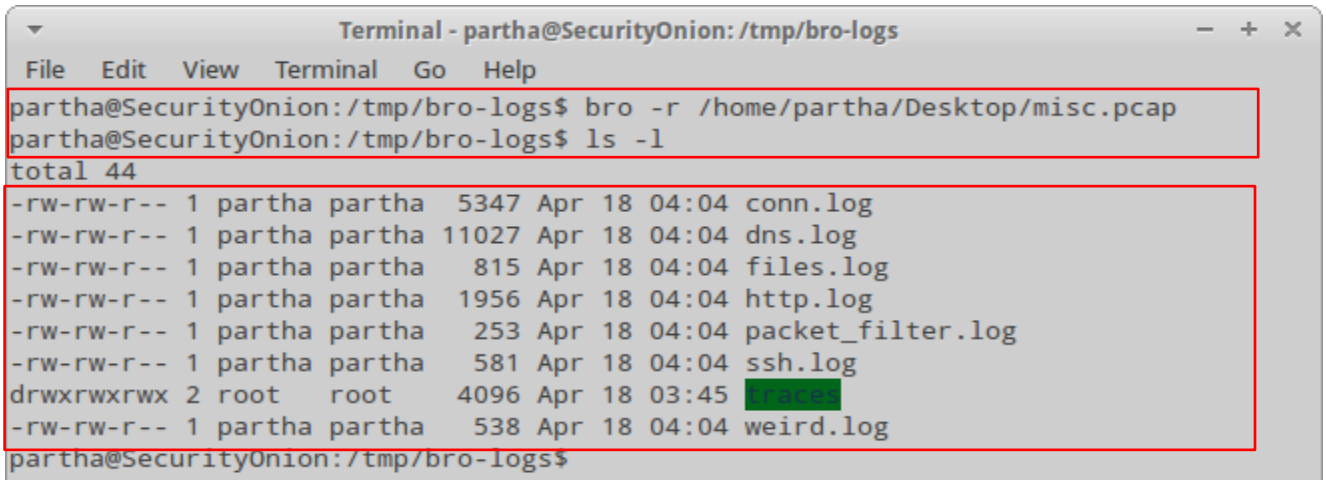


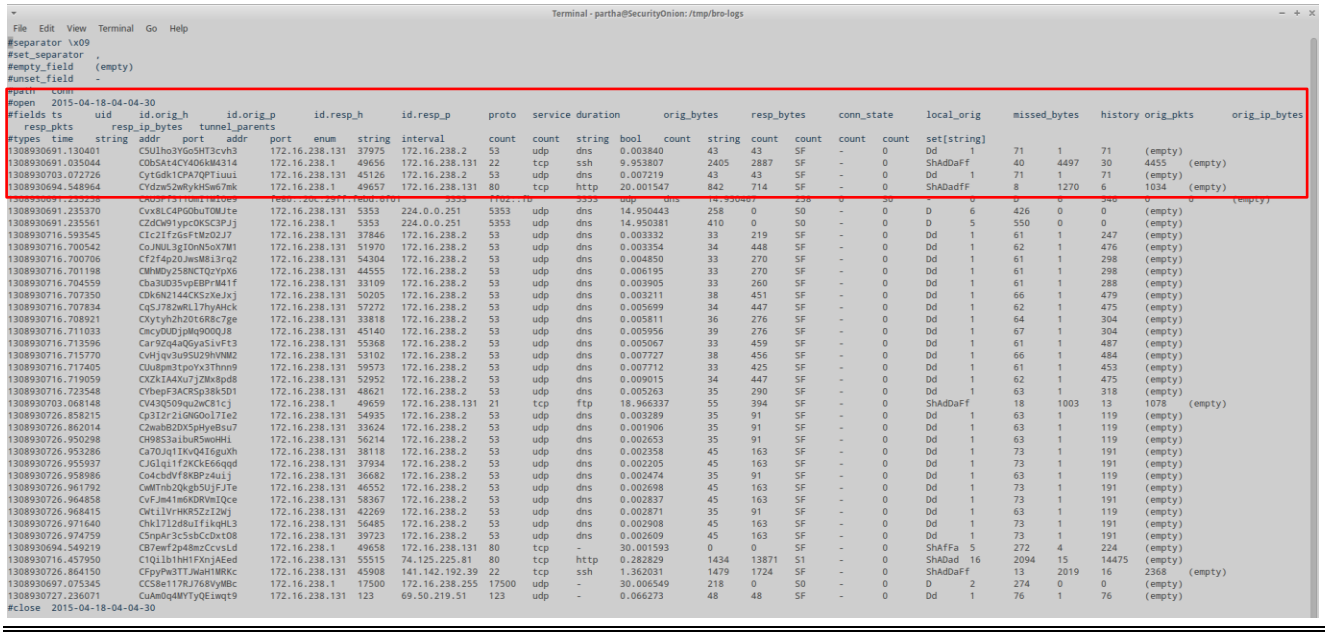
```
Terminal - partha@SecurityOnion: /tmp/bro-logs
File  Edit  View  Terminal  Go  Help
partha@SecurityOnion:~$ cd /tmp/bro-logs/
partha@SecurityOnion:/tmp/bro-logs$ bro -r /home/partha/Desktop/http.pcap
partha@SecurityOnion:/tmp/bro-logs$ ls
conn.log  files.log  http.log  packet_filter.log  traces
partha@SecurityOnion:/tmp/bro-logs$ ls -l
total 124
-rw-rw-r-- 1 partha partha 13201 Apr 18 03:55 conn.log
-rw-rw-r-- 1 partha partha 27235 Apr 18 03:55 files.log
-rw-rw-r-- 1 partha partha 69746 Apr 18 03:55 http.log
-rw-rw-r-- 1 partha partha   253 Apr 18 03:55 packet_filter.log
drwxrwxrwx 2 root   root    4096 Apr 18 03:45 traces
partha@SecurityOnion:/tmp/bro-logs$
```

Part-2: Matching Records to Log Fields: Extract the given file misc.pcap using the command ‘ bro –r ‘. All the fields in the obtained .log files both in the Part-1 and Part-2 are explained in the following directory of our installation. /usr/local/etc/share/bro/base/protocols/<Protocol_Name>/main.bro



It is very important to know the meanings and data types of each field to get some meaningful information from the log file.

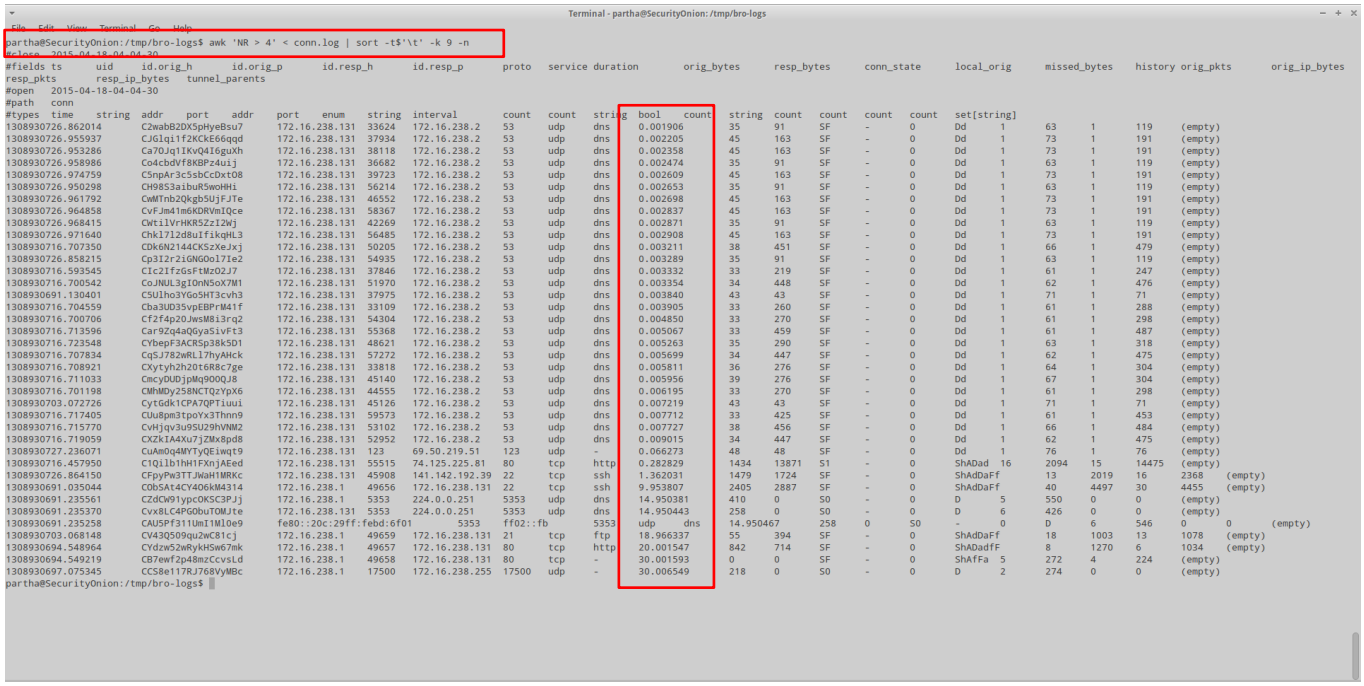




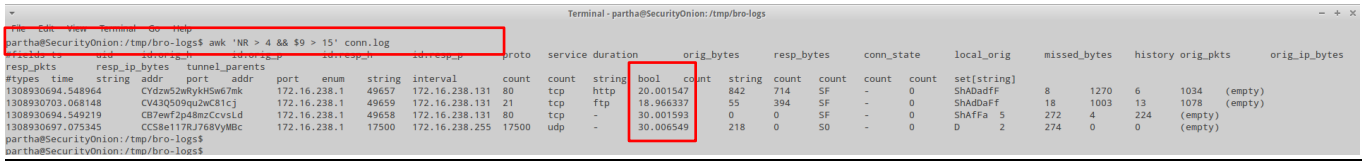
Part-3: Connection Statistics: Bro summarizes each TCP and UDP connection as a single line in the conn.log. We can filter the connections depending on different conditions.

First, these connection summaries are quite detailed, so we can extract plenty useful statistics from it. Moreover, we can concentrate only on those records on which we are interested. The set size will become much smaller compared to the whole dump file.

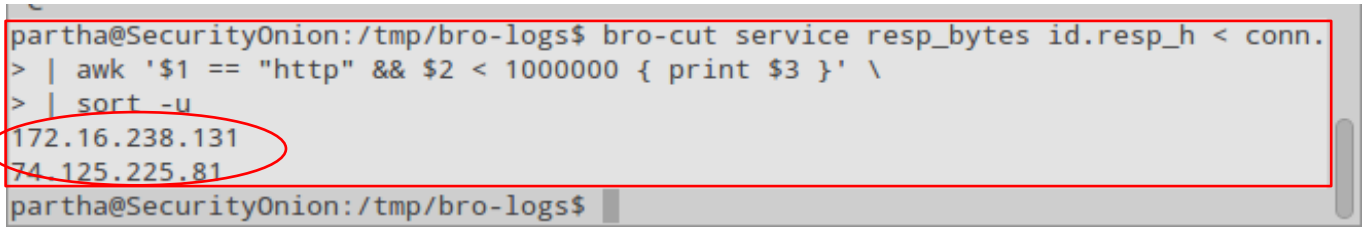
(i) List the connections by in increasing order of duration, i.e., the longest connections at the end. The following command is used to achieve the goal. By 'NR > 4' the first 4 commented lines are ignored and the list is sorted by the 9th column which is by default holds the 'interval'.



(ii) Find all connections that are last longer than 15 seconds. 9th field is checked whether the value is > 15 sec.



(iii) Find all IP addresses of web servers that send more than more than 1 KB back to a client.



(iv) Are there any web servers on non-standard ports (i.e., 80 and 8080)?

```
partha@SecurityOnion:/tmp/bro-logs$ bro-cut service id.resp_p id.resp_h < conn.l
og \
>     | awk '$1 == "http" && ! ($2 == 80 || $2 == 8080) { print $3 }' \
>     | sort -u
74.125.225.81
partha@SecurityOnion:/tmp/bro-logs$
```

Here, by $1, $2, $3 the fields in the conn.log file are indicated.

(v) Show a breakdown of the number of connections by service.

```
partha@SecurityOnion:/tmp/bro-logs$
partha@SecurityOnion:/tmp/bro-logs$ bro-cut service < conn.log | sort | uniq -c
| sort -n
      1 ftp
      2 http
      2 ssh
      3 -
     30 dns
partha@SecurityOnion:/tmp/bro-logs$
```

Here, 'uniq –c' takes the count for each unique type of service and 'sort –n' sorts the list according to the numerical value.

(vi) Show the 5 top destination ports in descending order.

```
partha@SecurityOnion:/tmp/bro-logs$ bro-cut id.resp_p < conn.log | sort | uniq -
c | sort -rn | head -n 5
     27 53
      3 5353
      2 80
      2 22
      1 88
partha@SecurityOnion:/tmp/bro-logs$
```

(vii) What are the top 10 hosts (originators) that send the most traffic?

```
partha@SecurityOnion:/tmp/bro-logs$ bro-cut id.orig_h orig_bytes < conn.log
            | sort                                                  | awk '{ if (hos
t != $1) {                      \
                if (size != 0)                      \
                    print $1, size;                 \
                 host=$1;                           \
                 size=0                             \
              } else                                \
                  size += $2                        \
            }                                       \
            END {                                   \
                if (size != 0)                      \
                    print $1, size                  \
                }'                                      | sort -k 2
                          | head -n 2
172.16.238.131 2623
172.16.238.1 2802
partha@SecurityOnion:/tmp/bro-logs$
```

Part-4: HTTP Statistics: As many of the attacks an suspicious activities are disguised under the HTTP request-reply, some special cares are needed to keep a closer eye on them. So Bro always gather different kinds of information about the hosts, like- Browser, Version, Operating System, etc.

(i) What are the distinct browsers in this trace? What are the distinct MIME types of the downloaded URLS?

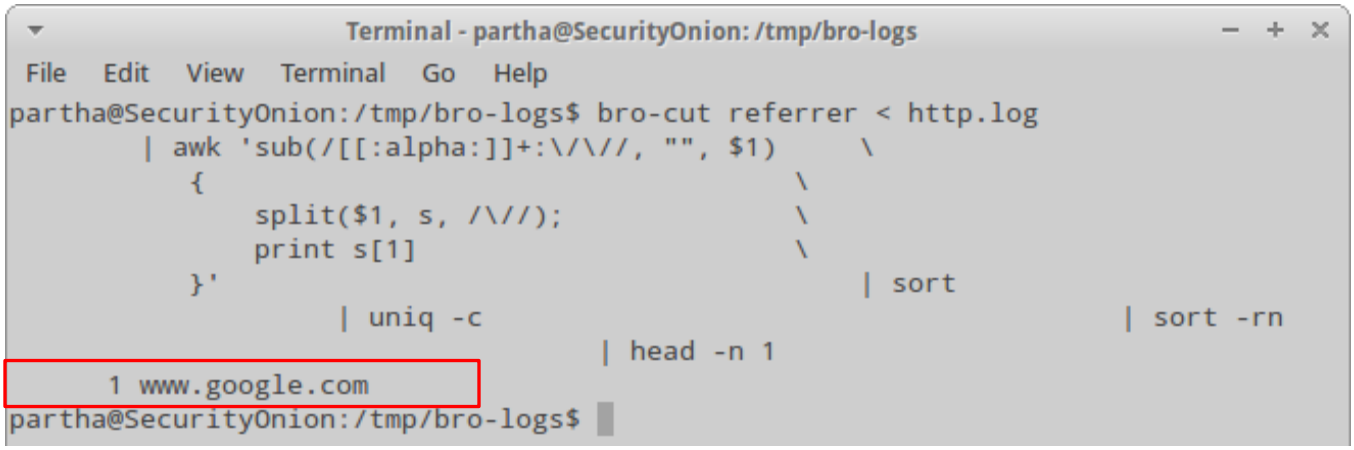
Here we have selected only the desired columns by the command 'bro-cut' and then restrict the output to the distinct values. The "mime_type" field was absent in our file so it returned an "unknown field" message.

```
partha@SecurityOnion:/tmp/bro-logs$ bro-cut user_agent < http.log | sort -u
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8) AppleWebKit/534.30 (KHTML, like G
ecko) Chrome/12.0.742.100 Safari/534.30
Mozilla/5.0 (X11; Linux i686; rv:5.0) Gecko/20100101 Firefox/5.0
partha@SecurityOnion:/tmp/bro-logs$ bro-cut mime_type < http.log | sort -u
bro-cut error: unknown field "mime_type"
partha@SecurityOnion:/tmp/bro-logs$
```

(ii) What are the two most commonly accessed web sites?

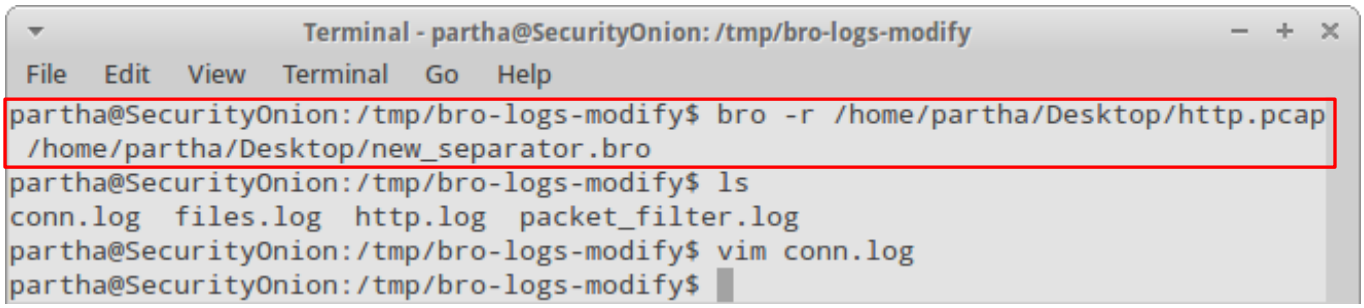
```
partha@SecurityOnion:/tmp/bro-logs$ bro-cut host < http.log | sort | uniq -c | s
ort -n | tail -n 2
      2 172.16.238.131
      2 www.google.com
partha@SecurityOnion:/tmp/bro-logs$
```

(iii) What are the top referred hosts?



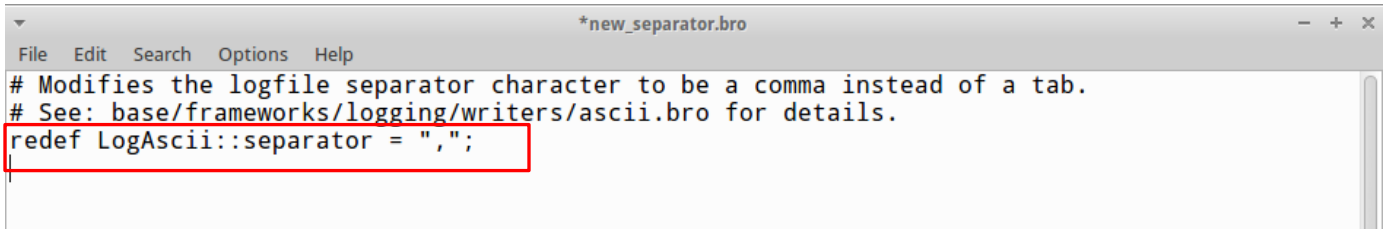
Note: We worked with smaller tcpdump file due to performance reasons and that is why we searched only the top website as we do not have 10 unique site referred.

Part-5: Tweaking Log Output: After writing the script new_separator.bro script which modifies the field separator in the .log files during generation from tcpdump, we reprocessed http.pcap. The new files were generated with comma as the field separator. In the line number 2 in the commented line the change was also reflected.

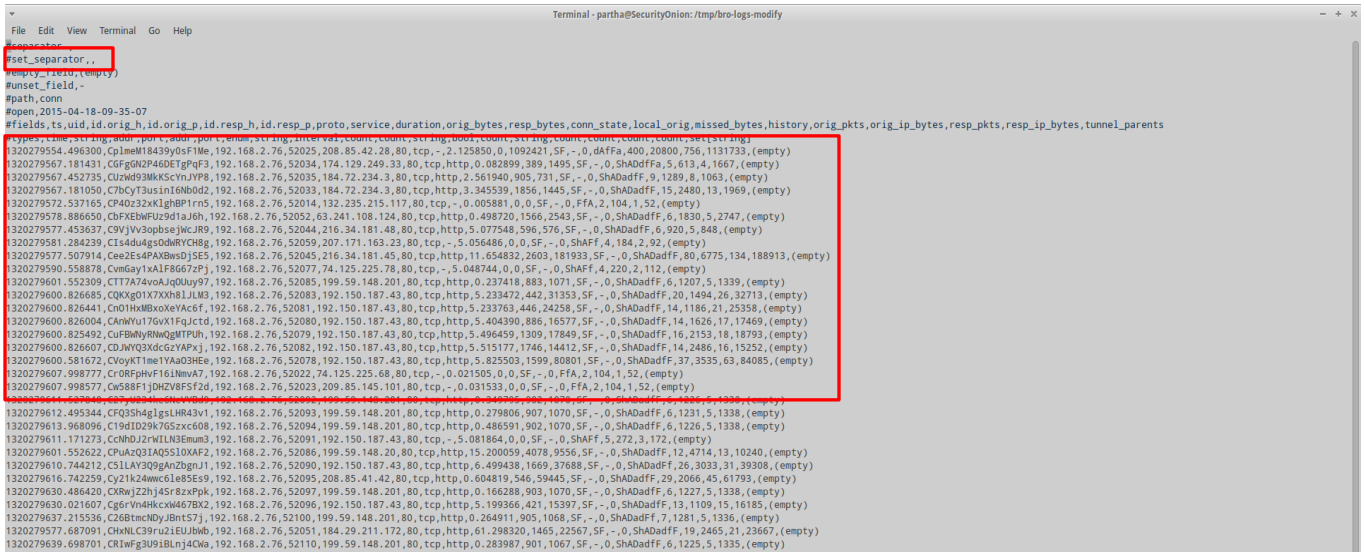


Note: To include the bro script while extraction the location of the bro script should be provided as follows.
partha@SecurityOnion:~$ bro -r /home/partha/Desktop/ssh.pcap local **/home/partha/Desktop/my_basic_policy.bro**

See the body of the bro script.

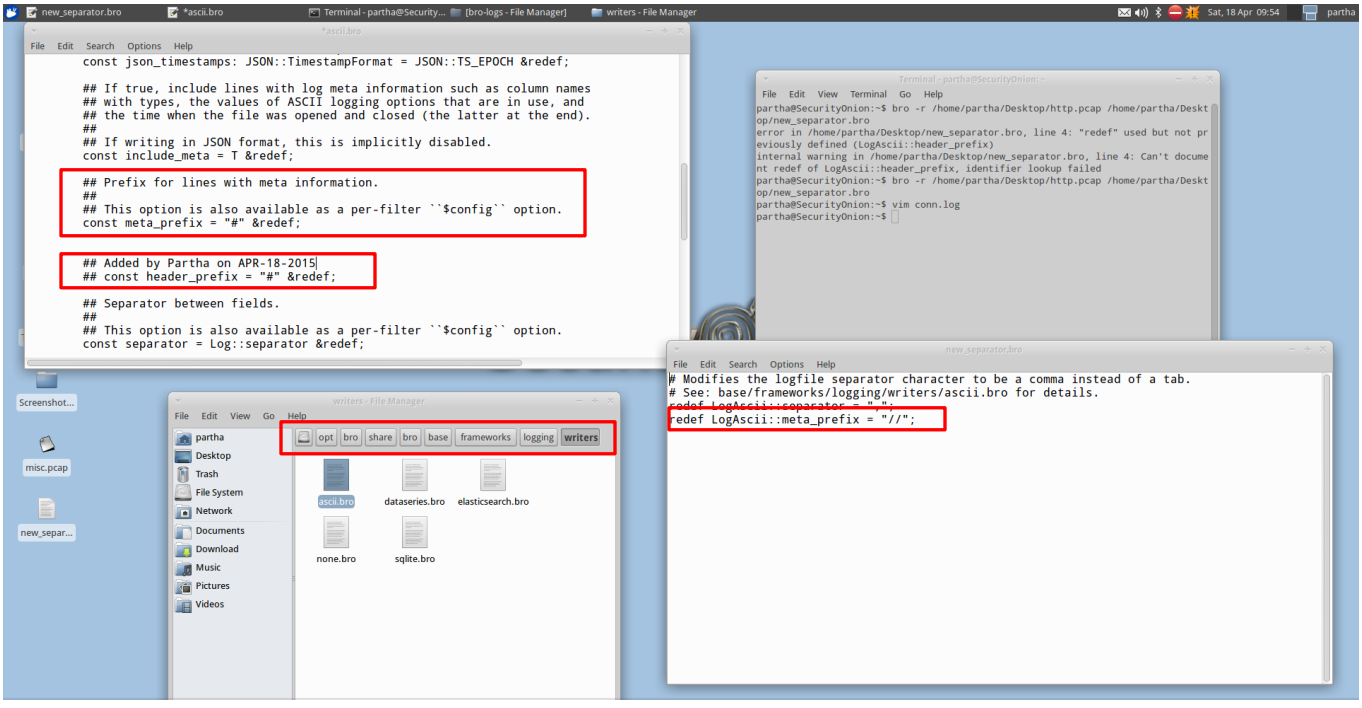


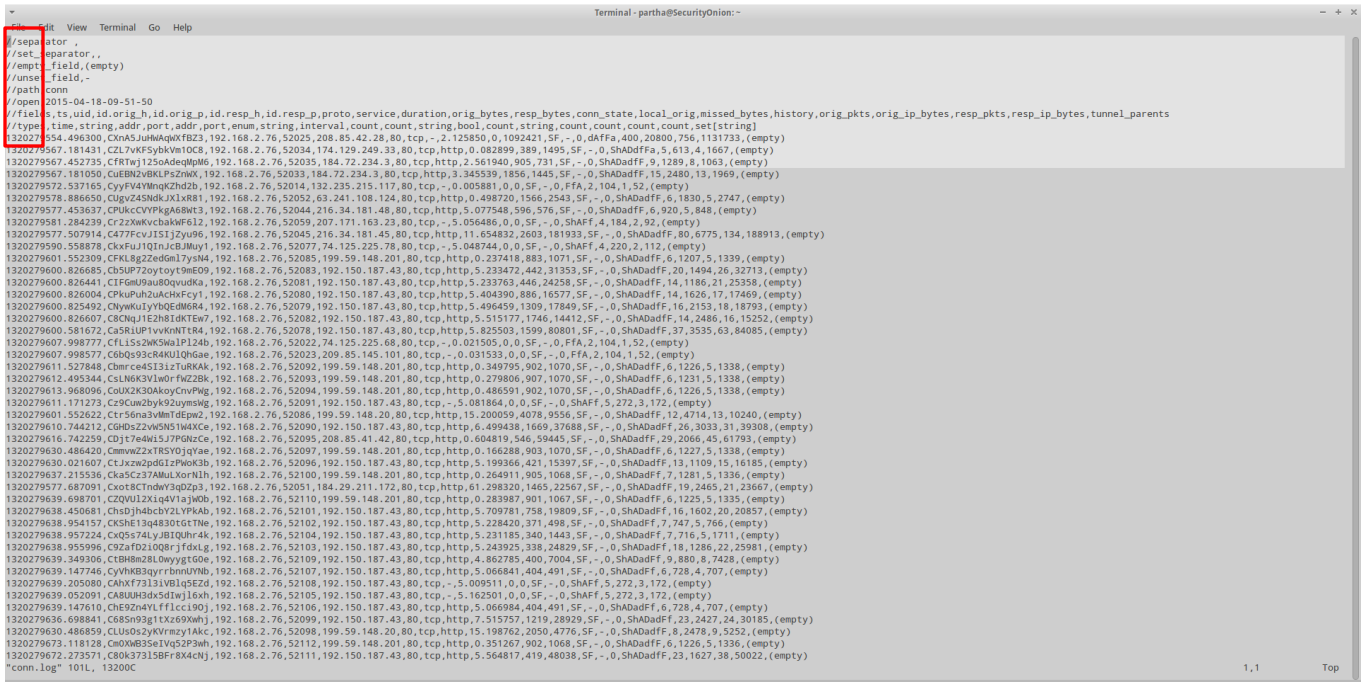
Next one is the log files regenerated with new field separator comma.



Now we wanted to change the comment character to ‘ // ’. For that we consulted the file ascii.bro in the directory /base/frameworks/logging/writers/ and found that character is defined by the key word “**meta_prefix**”. We overrode that value using Bro script.

Note: We also tried to add an entry with the key word “**header_prefix**” (As suggested in the training material). But unfortunately that did not work though we do not have any clue regarding the reason. We tried again by disabling the “**meta_prefix**”, still no luck. We got an error message saying **internal error: internal variable LogAscii::meta_prefix missing.** Newly generated .log file sample screenshot is below.

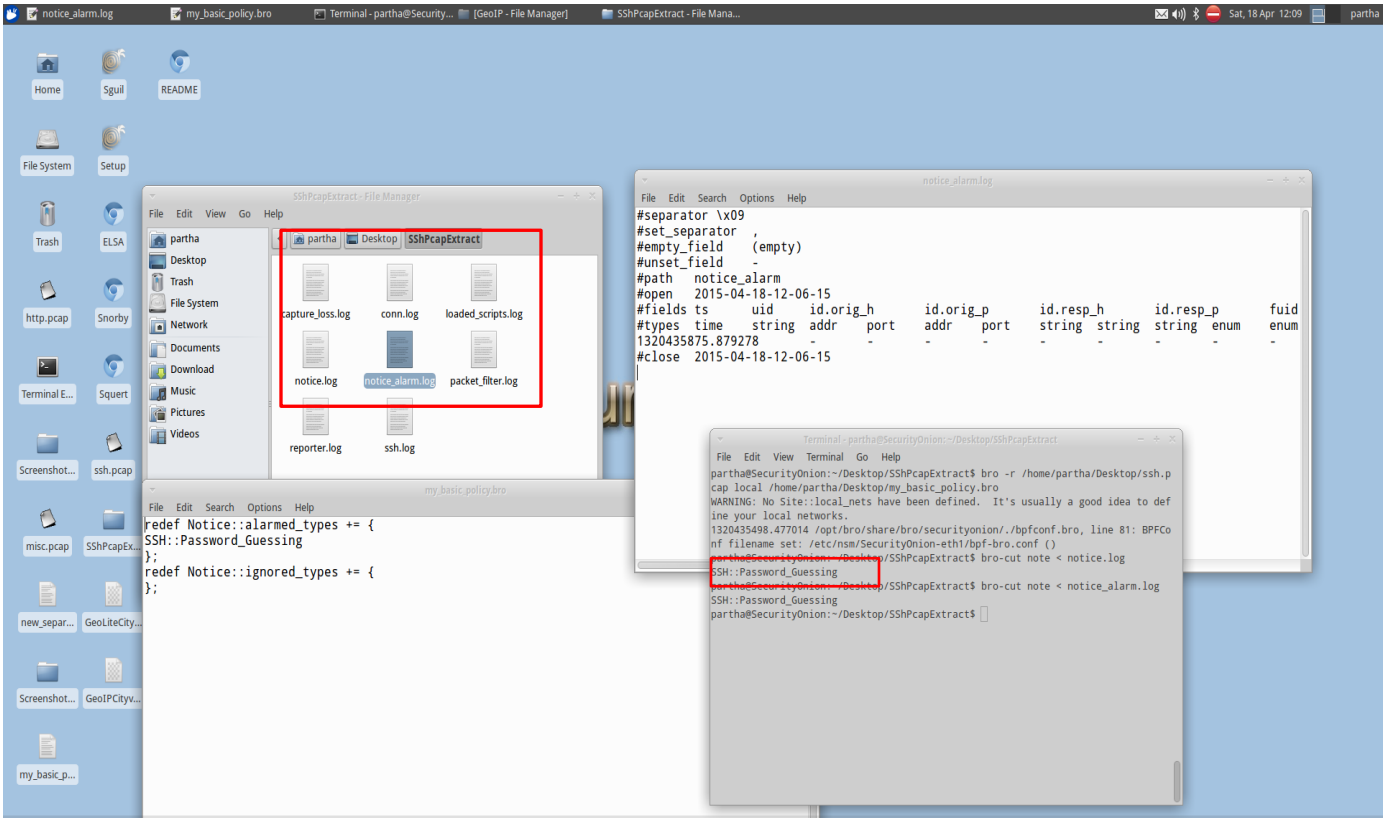




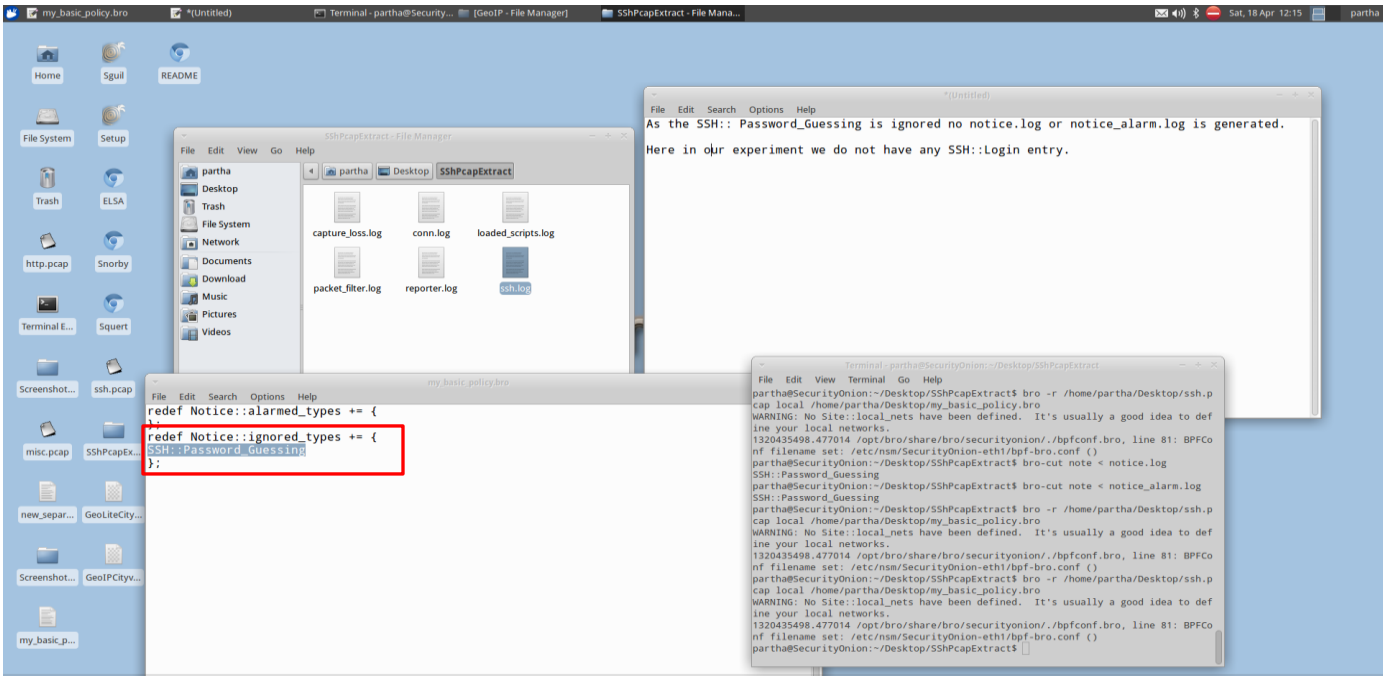
Ans. 1-B: https://www.bro.org/current/solutions/notices/

Exercise: Handling Notices: Bro by default, does some analysis on traffic and can suggest potentially interesting network activity to investigate. These suggestions are known as "notices".

Part 1: Notice Log: Run Bro on a network trace and check if it suggests anything interesting to look at. We have downloaded ssh.pcap for this exercises. We have used the command '*bro –r ssh.pcap local'* that will extract 8 files from the .pcap file. Now we have checked the file 'notice.log' file to see any interesting report and found somebody tried to brute forced to crack password.


notice_alarm.log
my_basic_policy.bro
Terminal - partha@Security...
[GeoIP - File Manager]
SShPcapExtract - File Mana...
Sat, 18 Apr 12:09
partha
Home
Sguil
README
File System
Setup
Trash
ELSA
http.pcap
Snorby
Terminal E...
Squert
Screenshot...
ssh.pcap
misc.pcap
SShPcapEx...
new_separ...
GeoLiteCity...
Screenshot...
GeoIPCityv...
my_basic_p...
SShPcapExtract - File Manager
File Edit View Go Help
partha
Desktop
Trash
File System
Network
Documents
Download
Music
Pictures
Videos
partha
Desktop
SShPcapExtract
capture_loss.log
conn.log
loaded_scripts.log
notice.log
notice_alarm.log
packet_filter.log
reporter.log
ssh.log
my_basic_policy.bro
File Edit Search Options Help
redef Notice::alarmed_types += {
SSH::Password_Guessing
};
redef Notice::ignored_types += {
};
notice_alarm.log
File Edit Search Options Help
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path notice_alarm
#open 2015-04-18-12-06-15
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p fuid
#types time string addr port addr port string string string enum enum
1320435875.879278 - - - - - - - - -
#close 2015-04-18-12-06-15
Terminal - partha@SecurityOnion:~/Desktop/SShPcapExtract
File Edit View Terminal Go Help
partha@SecurityOnion:~/Desktop/SShPcapExtract$ bro -r /home/partha/Desktop/ssh.p
cap local /home/partha/Desktop/my_basic_policy.bro
WARNING: No Site::local_nets have been defined. It's usually a good idea to def
ine your local networks.
1320435498.477014 /opt/bro/share/bro/securityonion/./bpfconf.bro, line 81: BPFCo
nf filename set: /etc/nsm/SecurityOnion-eth1/bpf-bro.conf ()
SSH::Password_Guessing
SSH::Password_Guessing
partha@SecurityOnion:~/Desktop/SShPcapExtract$

Part 2: Basic Filtering on the Notice Log: Now, we have written custom policy using Bro script. We have to pass the custom script while extracting the .pcap file. We have used 'my_basic_policy.bro' as our custom Bro script. In the script we set a policy that if there is any password guessing brute force attack alarm should be raised. We have seen that after the extraction using the Bro script it generated a new file 'notice_alarm.log'. By default Bro checks the policies from the file 'local.bro' in the directory "*<Installation_Dir>/share/bro/site*". Using Bro script we can also send email to the administrators in case of any alarm.



Part 3: More Advanced Notice Policy: More complex policies can be written according to our need. We have used a Bro script 'my_advanced_policy.bro' to implement a policy like when password guessing is done from only some particular IP address happens, then only it will raise alarm. We can add the IP address 172.16.238.129

```
module SSH;

redef enum Notice::Type += {
        Login,
};

const watched_servers: set[addr] = {
        172.16.238.136,
        172.16.238.168,
} &redef;

event SSH::heuristic_successful_login(c: connection)
        {
        NOTICE([$note=Login, $msg="Possible SSH login success", $conn=c]);
        }

hook Notice::policy(n: Notice::Info)
        {
        if ( n$note == SSH::Login && n$id$resp_h in watched_servers )
                add n$actions[Notice::ACTION_ALARM];
        }
```

Part 4: Extra Credit: Till now the custom scripts we have seen are working only on one IP address, in other words, it works with one line in the log files. Now we can write some even more complex policies that catches all the IP addresses that attempted SSH brute force attack for last 24 hours duration and raise alarm if such an IP address does a successful login assuming that the attacker become successful to break in the system.

my_extracredit_policy.bro

File Edit Search Options Help

```
module SSH;

redef enum Notice::Type += {
        Login,
};

global brute_forcers: set[addr] &write_expire=24hrs;

event SSH::heuristic_successful_login(c: connection)
        {
        NOTICE([$note=Login, $msg="Possible SSH login success", $conn=c]);
        }

hook Notice::policy(n: Notice::Info)
        {
        if ( n$note == SSH::Password_Guessing )
                add brute_forcers[n$src];
        else if ( n$note == SSH::Login && n$id$orig_h in brute_forcers )
                {
                add n$actions[Notice::ACTION_ALARM];
                n$sub = "Was previously a password guesser";
                }
        }
```

==========

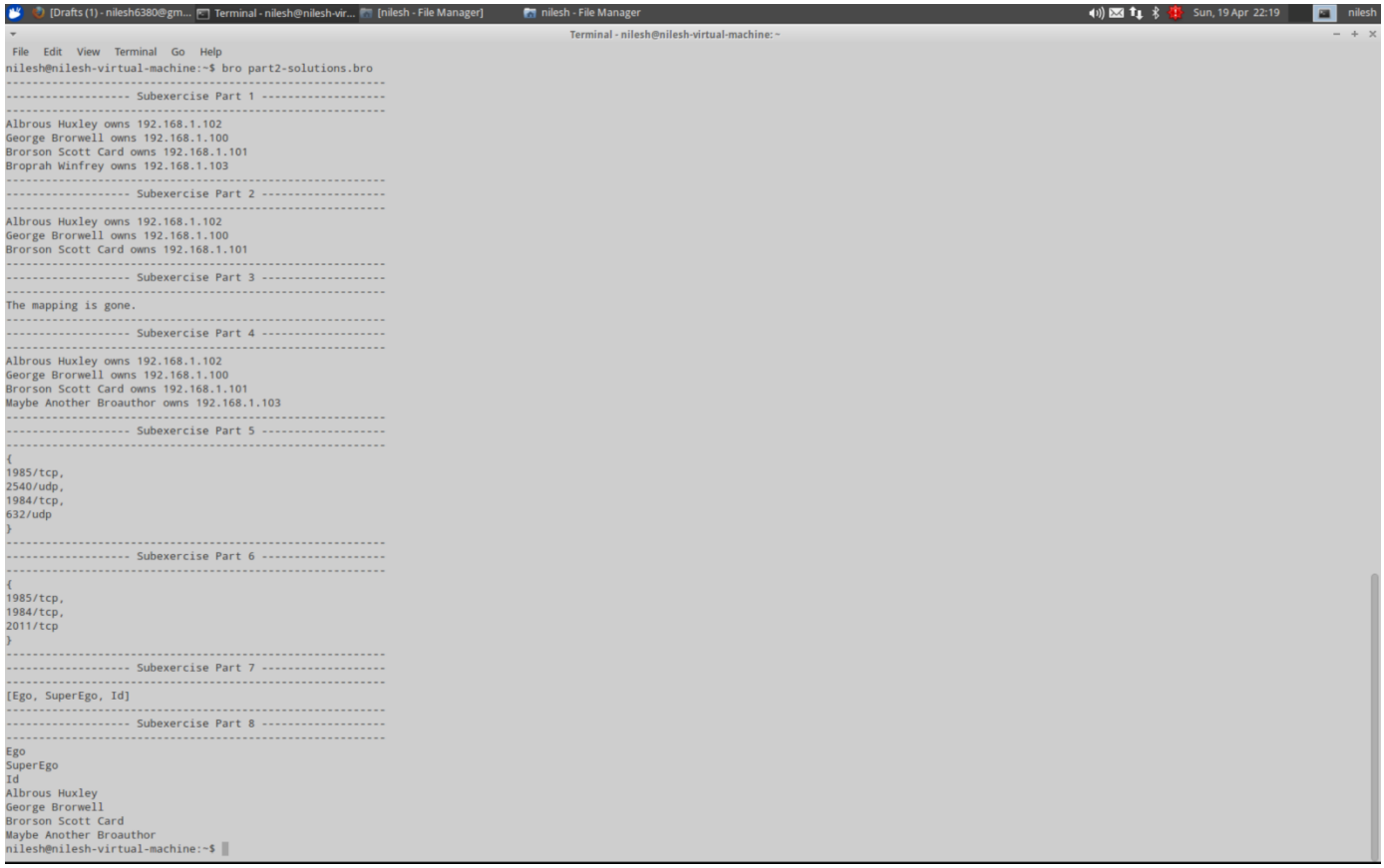
==========

Ans. 1-C: https://www.bro.org/bro-workshop-2011/solutions/prog-primer/index.html

Exercise: Bro Programming Primer: Bro contains a programming language designed specially to be able to represent network-related abstraction (e.g. addresses and ports) and as such offers a great deal of functionality and flexibility in terms of helping you accomplish your network monitoring goals. . In this section we have learned about some very useful utilities that can be used to retrieve data as desired.

Part-1: String manipulations and patterns: We are provided with a part1.bro file with sub exercises inside it. This exercise will help us to learn about String manipulations and patterns.

Part-2: Tables and set and vectors: We are provided with a file part2.bro. There are several containers/collections types that are commonly used in Bro for accumulating/storing state information. Here we are learning how to store data in tables, sets or vectors and manipulate/retrieve from them.



Part-3: Records and functions: We are provided with a file part3.bro with sub exercise in it. A record is a collection of values, with each have a name type pair. There is no restriction on allowed type, which means it can contain other, nested records or even functions as part of record. These exercises will help us learn creating the record, manipulation and deletion on the record and also about function inside it.

```
nilesh@nilesh-virtual-machine:~$ bro part3-solutions.bro
---------- Subexercise Part 1 ----------
192.168.1.111:ssh
192.168.1.222:13131/tcp
192.168.1.111:http
192.168.1.222:ssh
---------- Subexercise Part 2 ----------
192.168.1.111:ssh
192.168.1.222:ssh
192.168.1.111:http
192.168.1.222:13131/tcp
---------- Subexercise Part 3 ----------
*192.168.1.111:ssh
*192.168.1.222:ssh
*192.168.1.111:http
192.168.1.222:13131/tcp
---------- Subexercise Part 4 ----------
*192.168.1.111:ssh
*192.168.1.222:ssh
*192.168.1.111:http
192.168.1.222:13131/tcp
nilesh@nilesh-virtual-machine:~$
```

Part-4: Events: We have provided with a file part4.bro and tcpdump browse.pcap. Bro provides an event engine as the primary facilitator of network analysis. And bro can tap into the engine is at the scripting layer through even handler. In this exercise, we will learn about event and multi handlers.

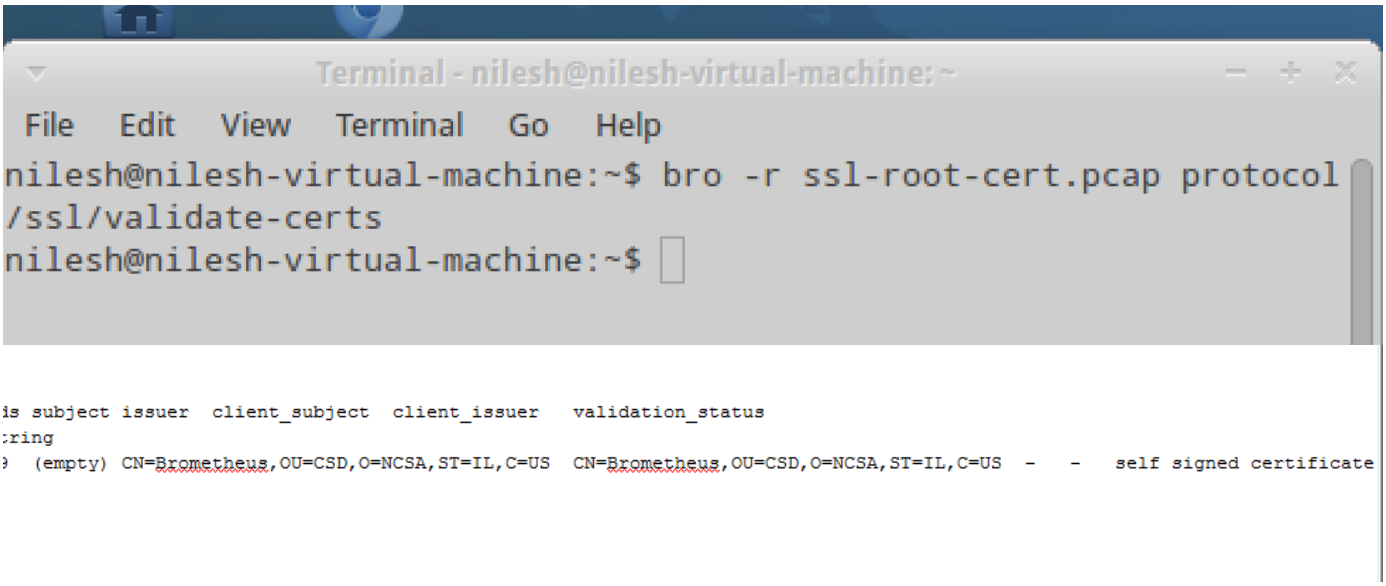
```
nilesh@nilesh-virtual-machine:~$ bro -r browse.pcap part4-solutions.bro
Hello, world!
Hello, Brogrammer!
192.150.187.43 -> /
192.150.187.43 -> /css/pygments.css
192.150.187.43 -> /css/960.css
192.150.187.43 -> /css/bro-ids.css
192.150.187.43 -> /js/jquery.cycle.all.min.js
192.150.187.43 -> /js/jquery.fancybox-1.3.4.pack.js
192.150.187.43 -> /js/jquery.tweet.js
192.150.187.43 -> /js/jquery.zrssfeed.js
192.150.187.43 -> /js/jquery.tableofcontents.js
192.150.187.43 -> /js/superfish.js
192.150.187.43 -> /js/hoverIntent.js
192.150.187.43 -> /js/general.js
192.150.187.43 -> /js/jquery.collapse.js
192.150.187.43 -> /css/print.css
192.150.187.43 -> /documentation/index.html
192.150.187.43 -> /js/breadcrumbs.js
192.150.187.43 -> /documentation/reporting-problems.html
The following users might be experiencing Broblems:
  10.0.2.15
nilesh@nilesh-virtual-machine:~$
nilesh@nilesh-virtual-machine:~$
nilesh@nilesh-virtual-machine:~$
nilesh@nilesh-virtual-machine:~$
nilesh@nilesh-virtual-machine:~$
nilesh@nilesh-virtual-machine:~$
```

==========

Ans. 1-D: http://www.bro.org/bro-workshop-2011/solutions/extending/index.html

Exercise: Extending a Script's Functionality:

Part 1: Customize which certificates to trust: pcap file has been given to us and asks us to perform certificate validation. The pcap file has a self-signed certificate. This is the same file is shown in the log files.
Importance: The admin might have gotten a new certificate and might be interested in adding this new certificate to the trusted list.




```
is subject issuer  client_subject  client_issuer   validation_status
tring
   (empty) CN=Brometheus,OU=CSD,O=NCSA,ST=IL,C=US  CN=Brometheus,OU=CSD,O=NCSA,ST=IL,C=US  -   -   self signed certificate
```

This is ssl.log file

Certificate hex has been given in this exercise and using it, we have to give instructions to Bro to trust the self-signed Certificate.
For this we created a mytrust.bro file as

redef SSL::root_certs += {
["My First Certificate"] = "hex file contents"};
And ran this command

```
nilesh@nilesh-virtual-machine:~$ bro -r ssl-root-cert.pcap protocol
s/ssl/validate-certs mytrust.bro
nilesh@nilesh-virtual-machine:~$
```

then Bro accepted the certificate as we can see the validation status shown below in the log file.

```
n_fuids subject issuer  client_subject  client_issuer   validation_status
ng  string
ttGxj9  (empty) CN=Brometheus,OU=CSD,O=NCSA,ST=IL,C=US  CN=Brometheus,OU=CSD,O=NCSA,ST=IL,C=US  -   -   ok
```

The third exercise asks us to create the certificate file that we used in the last file.
I typed the following command:

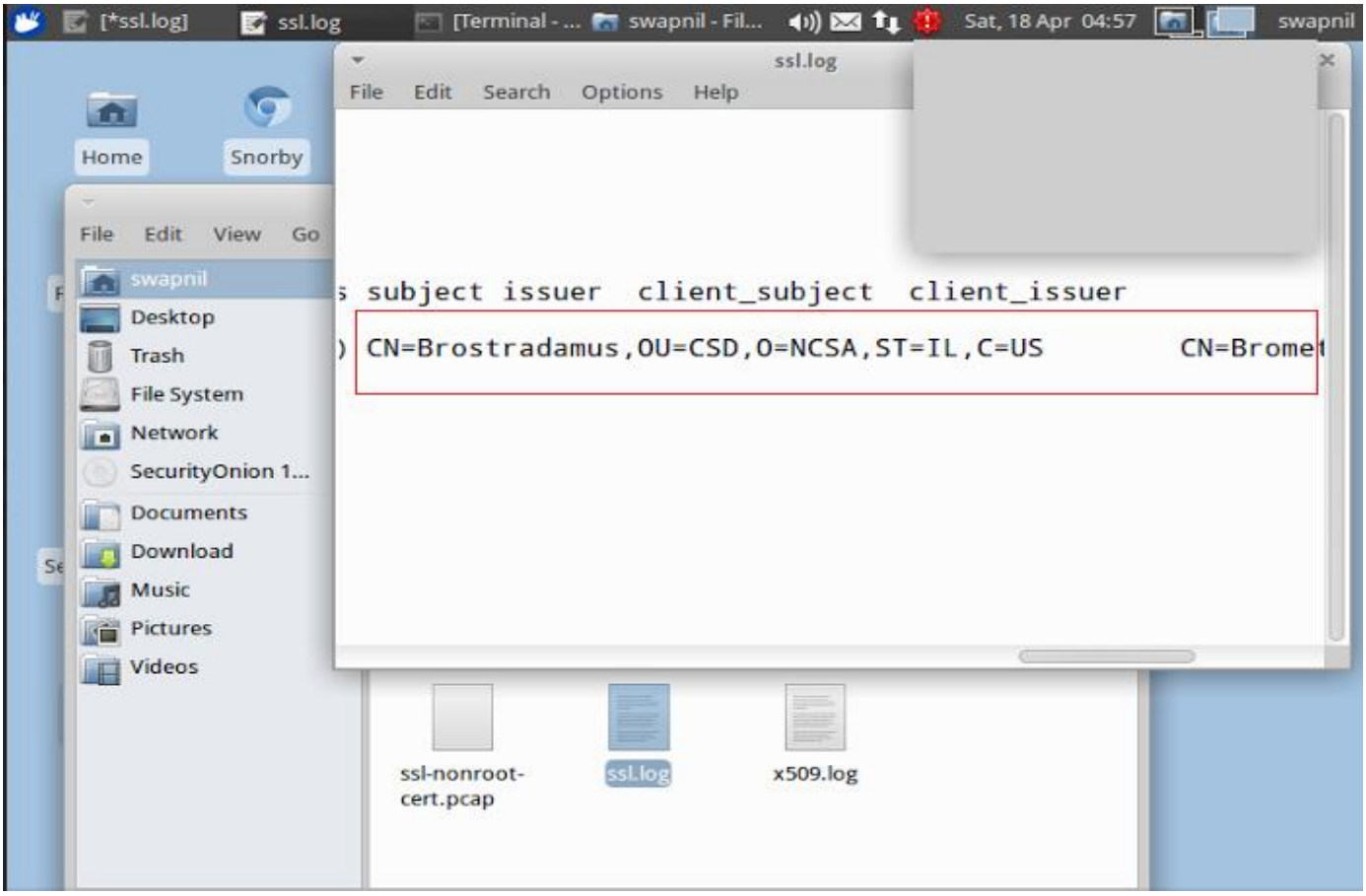
```
nilesh@nilesh-virtual-machine:~$ openssl x509 -in cert.der -inform
DER -outform DER | hexdump -v -e '1/1 "\\\x"' -e '1/1 "%02X"' > my-
cert-hexesc.der
nilesh@nilesh-virtual-machine:~$
```

Part 2: Log More Than Just Server Cert.Subject:

In the first exercise, we include the issuer of the server's certificate information. So, we create a new .bro file called rootissuer.bro. We need to redefine the SSL logging unit (SSL::Info) and handle x509_certificate event.
rootissuer.bro file contains:

```
redef record SSL::Info += {
root_issuer: string &log &optional;
};
event x509_certificate(c: connection, cert: X509, is_server: bool, chain_idx: count, chain_len: count, der_cert:
string)
{
c$ssl$root_issuer = cert$issuer;
}
```

And then run the following command:
bro -r ssl-nonroot-cert.pcap rootissuer.bro

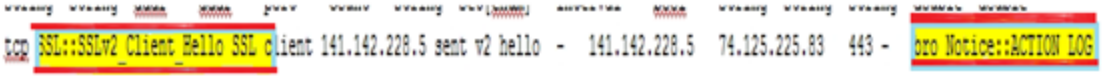


Part 3: Using the Notice Framework:

This exercise asks us to log the event when someone is using sslv2. We use given a pcap file and we just need to run Bro to check if some client is using SSLv2 and if so, create log.

```
module SSL;
redef enum Notice::Type += {
SSLv2_Client_Hello
};
event ssl_client_hello(c: connection, version: count, possible_ts: time, client_random:string, session_id: string,
ciphers: vector of count)

{
if ( version == SSLv2 )
{
local message = fmt("SSL client %s sent v2 hello", c$id$orig_h);
local ident = fmt("%s", c$id$orig_h);
NOTICE ([$note=SSLv2_Client_Hello,
$msg=message,
$conn=c,
$identifier=ident]);
}
}
```

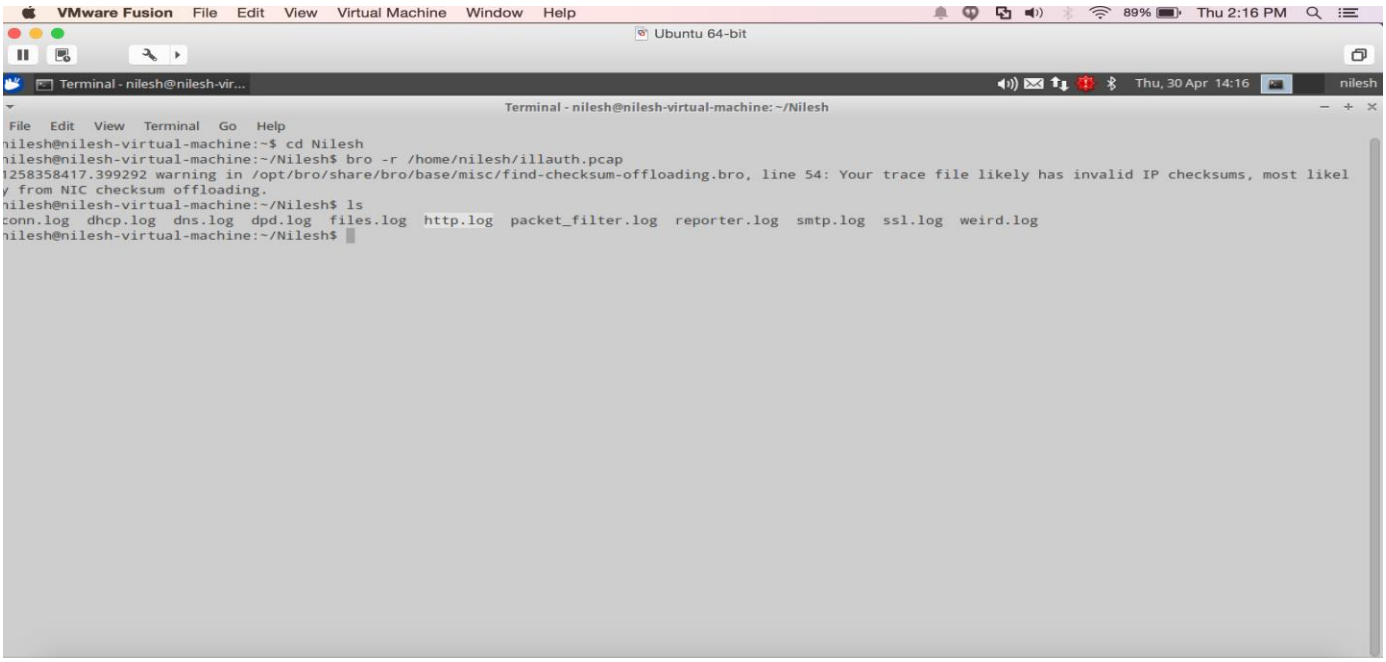


==========

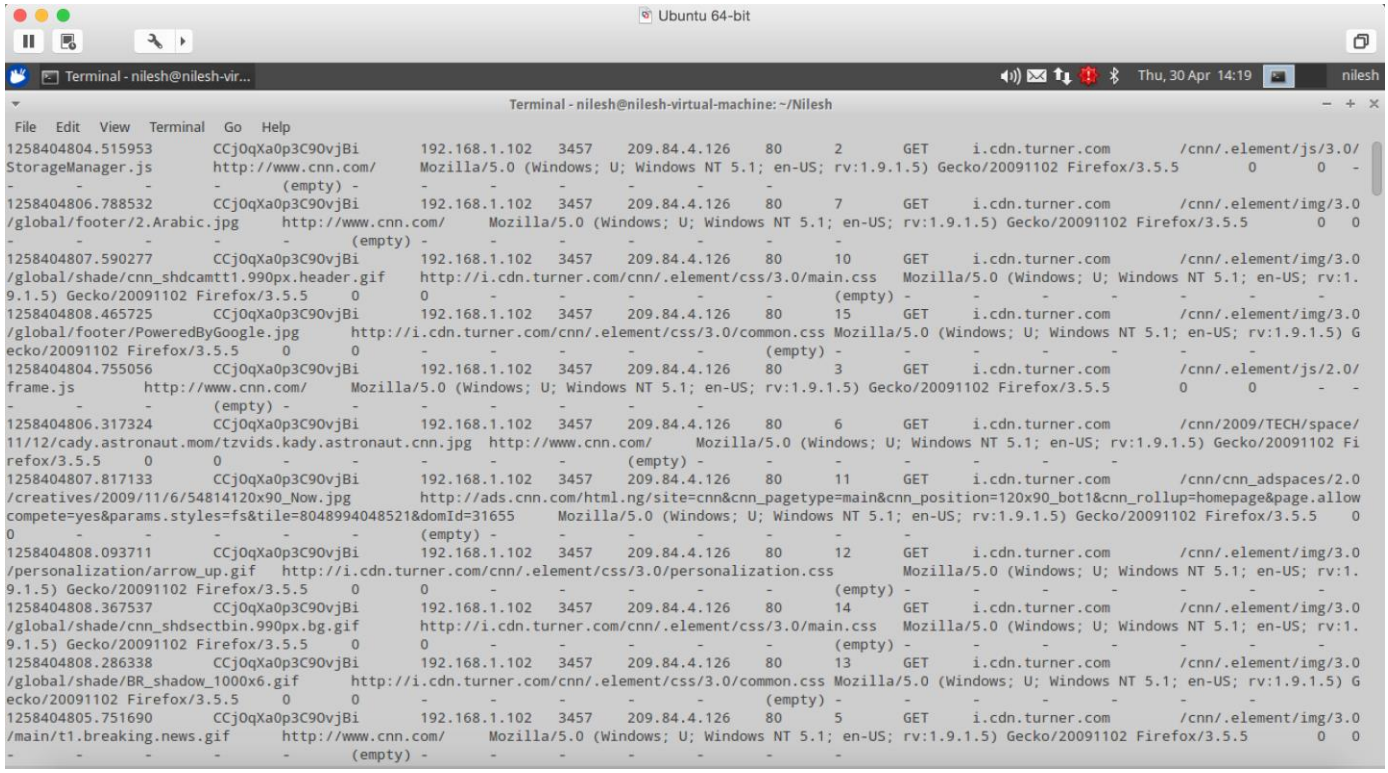
Ans. 1-E:
https://www.bro.org/bro-workshop-2011/solutions/incident-response/index.html

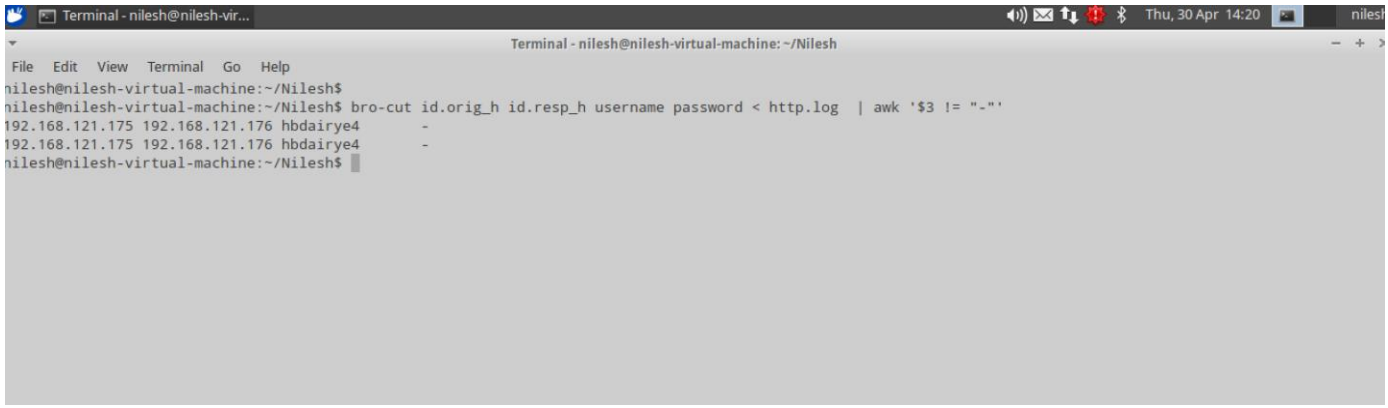
Exercise: Intelligent based incident response: A typical incidents response tasks begin with a piece of obtained intelligence that security analyst uses as starting point in any invesgtigation. The workshop session features several scenario of same kind.

Part-1: Ill consideration authorization: It has been observed that many users will use same password for different website, which may have weak security scheme. From this exercise, we are trying to determine that which webserver will have weak security scheme.
(1). Firstly, we will download the tcpdump named illauth.ncap fill and try to generate log files from that using bro scripts. We are looking for http header which will contain Basic and Digest authentication and for basic authentication, it will contain username and password in the header. The header will contain Basic and string and a encoded string which contain username and password. Bro will fetch all the basic authentication header information in HTTP requests and store it in http.log file. Below are the snapshots.



Below is the snapshot of http.log file, which will have all the information regarding all IPs.





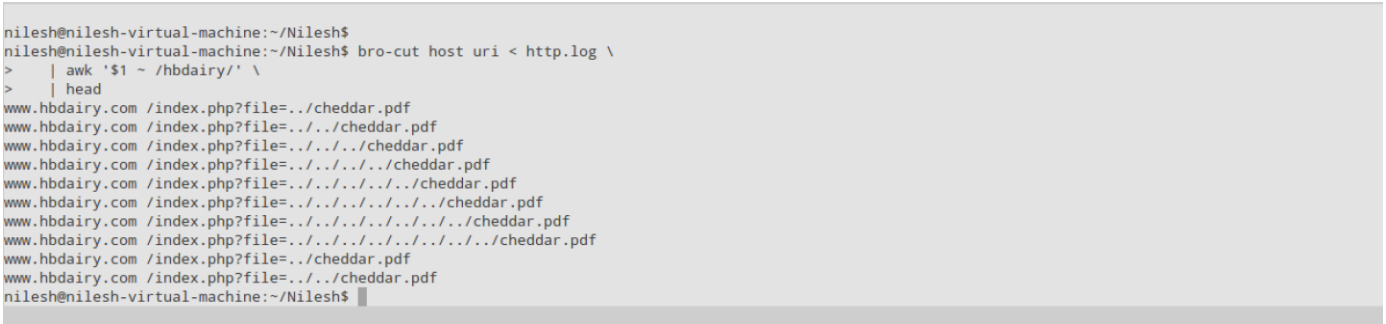
The output of the files shows which IP has weak authentication.

(2). In this part, we are trying to determine the user password for that IP. We will run the script with different parameter as argument so that it will convert blank details to password. Below are the snapshot for the same.

Here we can see that the password for the IP 192.168.121.175 is cheesecake.

Part-2:Information theft from web server: Here, we are determining about theft, which occur in the web server.

(1). In this part we are determining the logs generated by bro script on tcpdump theft.pcap. In this case also, we are checking http.log in which we will determine that what type of attack, an attacker is trying to do.



As we can see from the logs, it is trying to get cheddar.pdf by trying to access all the directories on the server.

(2). In this we are trying to determine whether he has successfully accessed the file or not. For that we are again check the http.log and status of every request and show the data on console to determine.

```
nilesh@nilesh-virtual-machine:~/Nilesh$ bro-cut host status_code < http.log \
>    | awk '$1 ~ /hbdairy/ { print $2 } ' \
>    | sort \
>    | uniq -c
      1 200
     90 404
nilesh@nilesh-virtual-machine:~/Nilesh$
```

```
nilesh@nilesh-virtual-machine:~/Nilesh$ bro-cut host uri status_code < http.log \
>    | awk '$1 ~ /hbdairy/ && $3 != "404"'
www.hbdairy.com /index.php?file=\xc0\xae\xc0\xae\xc0\xaf\xc0\xae\xc0\xae\xc0\xaf\xc0\xae\xc0\xae\xc0\xafcheddar.pdf     200
nilesh@nilesh-virtual-machine:~/Nilesh$
```

From above, we have seen that it has tried 91 times in which 90 times attacker has faced failure and for one time, he gets success.

Part-3:Email Leakage: As part of this exercise, we are trying to determine that someone has shared the username to attacker and now attacker is trying to send unencrypted email.

(1). We are trying to find out the source of the attacker if he sent the unencrypted file or not. Below are the snapshot to find out the one.

```
nilesh@nilesh-virtual-machine:~/Nilesh$
nilesh@nilesh-virtual-machine:~/Nilesh$
nilesh@nilesh-virtual-machine:~/Nilesh$
nilesh@nilesh-virtual-machine:~/Nilesh$ bro -r /home/nilesh/email.pcap
1302419241.847851 warning in /opt/bro/share/bro/base/misc/find-checksum-offloading.bro, line 54: Your trace file likely has invalid TCP checksums, most likel
y from NIC checksum offloading.
nilesh@nilesh-virtual-machine:~/Nilesh$ ls
conn.log  dhcp.log  dns.log  files.log  packet_filter.log  reporter.log  smtp.log  weird.log
nilesh@nilesh-virtual-machine:~/Nilesh$
nilesh@nilesh-virtual-machine:~/Nilesh$
nilesh@nilesh-virtual-machine:~/Nilesh$ grep hbdairy smtp.log
1302419719.507237       CGtIod2gnNrfyoYEh6      192.168.121.179 51158   192.168.121.176 25      1       [192.168.121.179]       <lesharq@dchlaw.com>    <mond
o.cheeze@hbdairy.com>   Sun, 17 Apr 2011 04:44:08 -0400 lesharq <lesharq@dchlaw.com>    mondo.cheeze@hbdairy.com        -       <1303029848.21831.7.camel@see
d-desktop>      -       [Confidential] advice   -       -       -       250 2.0.0 Ok: queued as 1774A73E60      192.168.121.176,192.168.121.179 Evolution 2.2
6.1     F       FMqjvz2vl2w74IDnA6,FAhpVH1rNrOi4Mm9uf
nilesh@nilesh-virtual-machine:~/Nilesh$
nilesh@nilesh-virtual-machine:~/Nilesh$
nilesh@nilesh-virtual-machine:~/Nilesh$ bro-cut uid from to subject < smtp.log | awk '$1 == "WVVfjCeD7"'
```

As part of this file no one is trying to send the unencrypted file to any email address. But he is sending, then we will be able to determine the email of the sender who is sending unencrypted file.

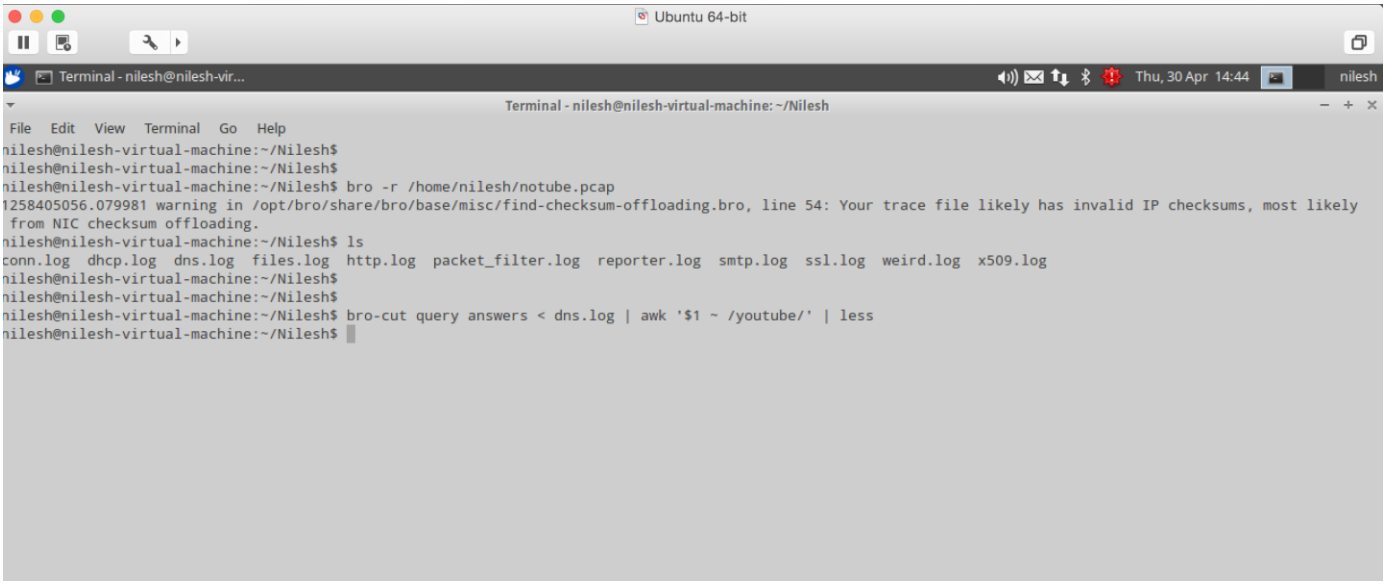
(2). Now, we are trying to determine who authored the document and what are the linked inside the pdf document. This can be achieved by reading the pdf file line by line and store the data into disk. Bro will not read the file on its own but there is an option to do the same in bro. By using **file_type,** we can read the document by below statement.

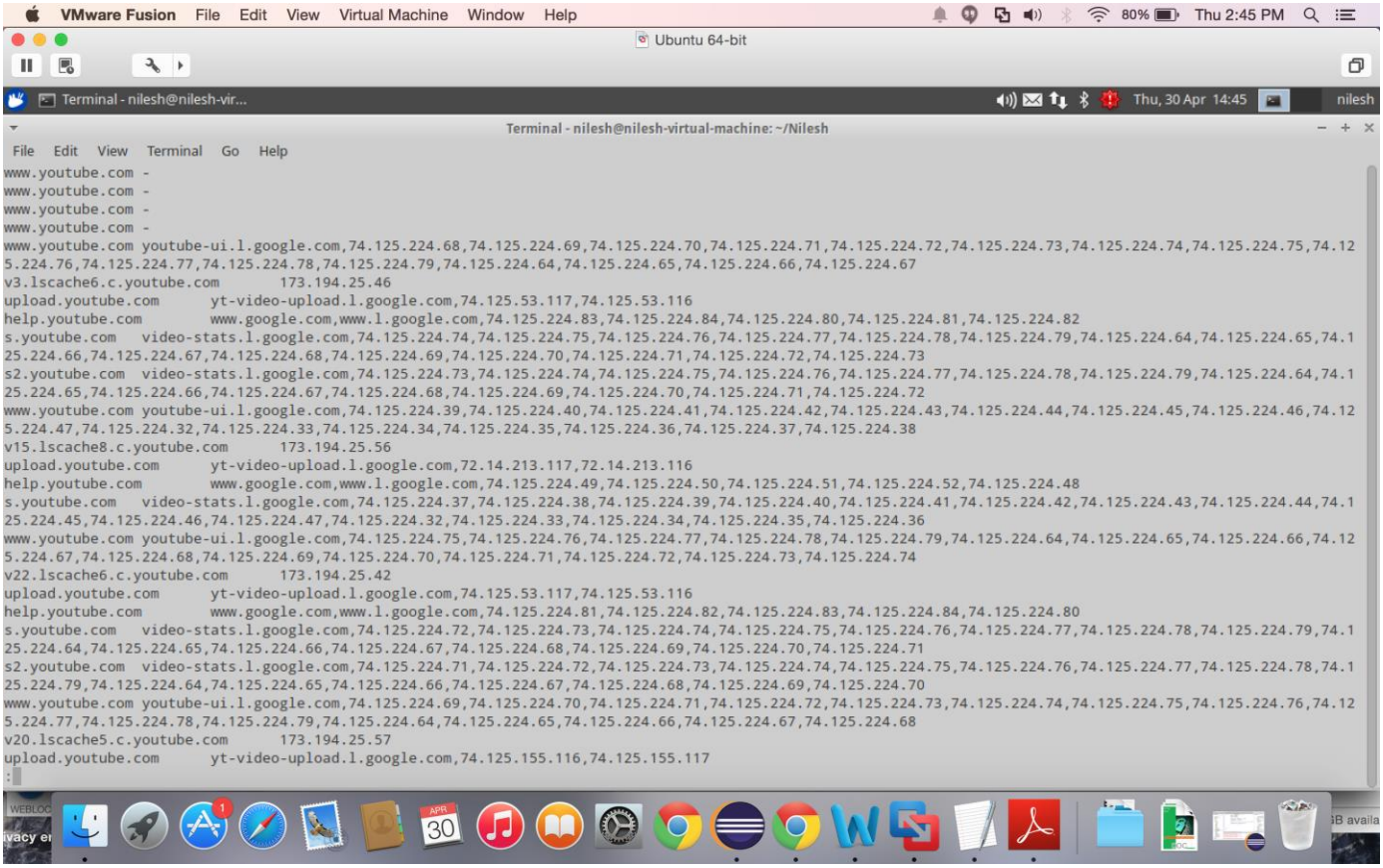
bro -r email.pcap -e 'redef SMTP::extract_file_types = /application\/pdf/;'

This will smtp-entity file, which we can read and check.

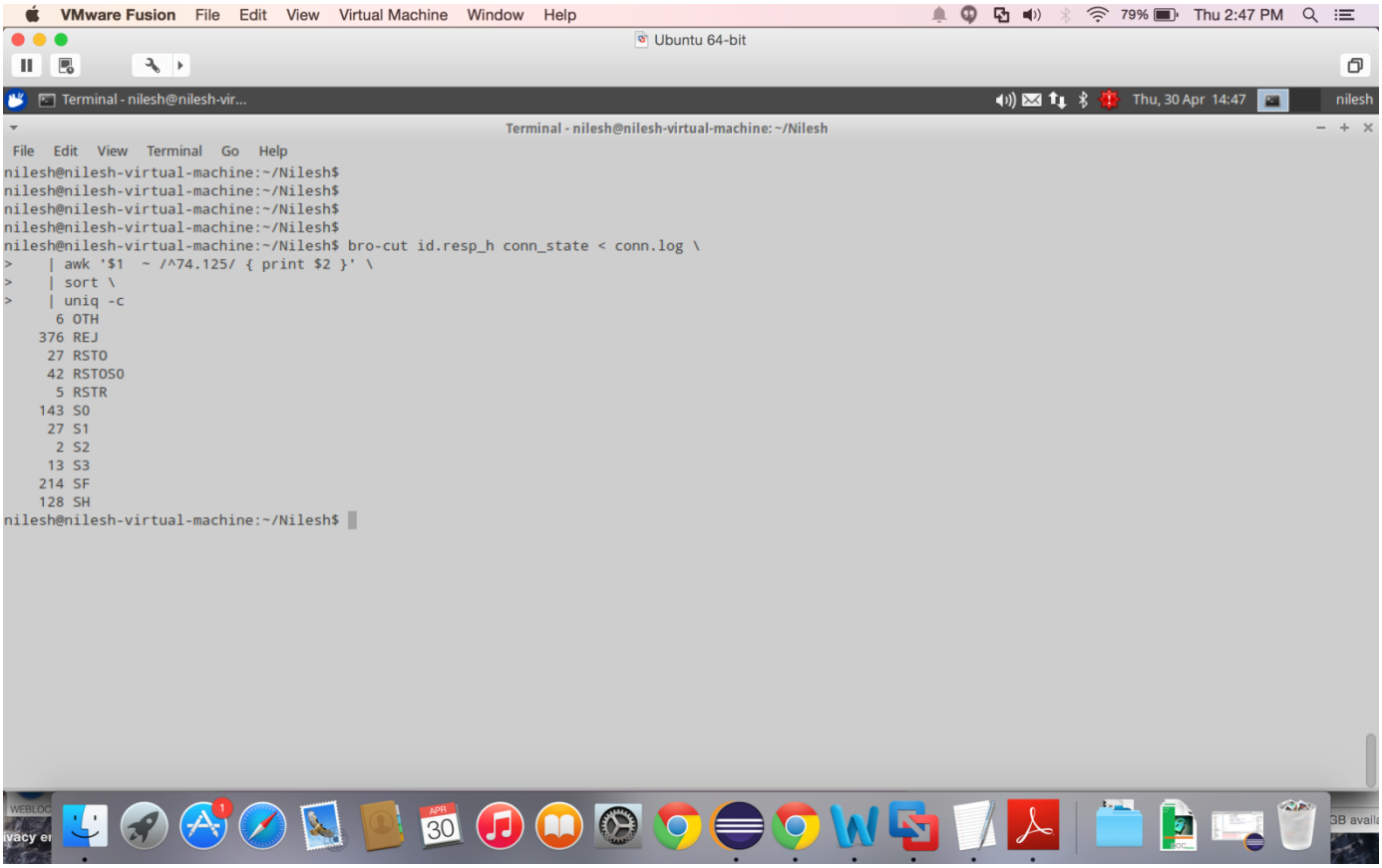
Part-4:YouTube becomes NoTube: When you are giving access to someone to a particular site. And that person will remove/revoke the access of the other users of the same group.

(1). We are trying to find out the how someone has disrupted the Youtube access. For this we can generate the logs and check the dns logs to find out where the key is youtube



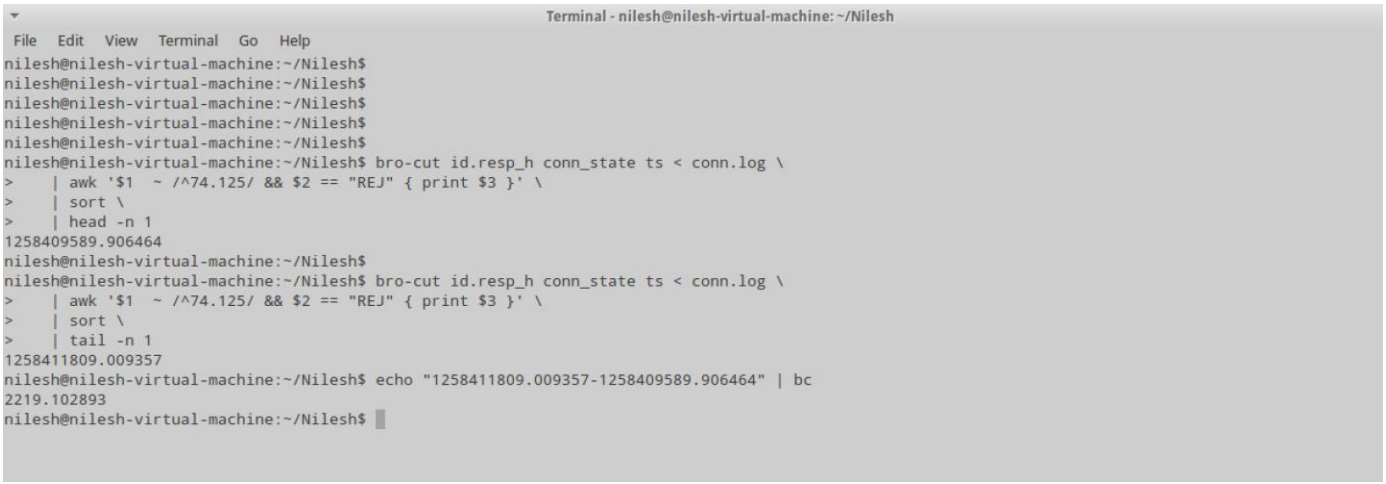


Here we have seen the IP ranges of youtube connections. Now we see a pattern in the IP address from which we are trying to open youtube and fails. Now we can check the status in connection logs where pattern matches with our value. Below are the snapshot for the same.



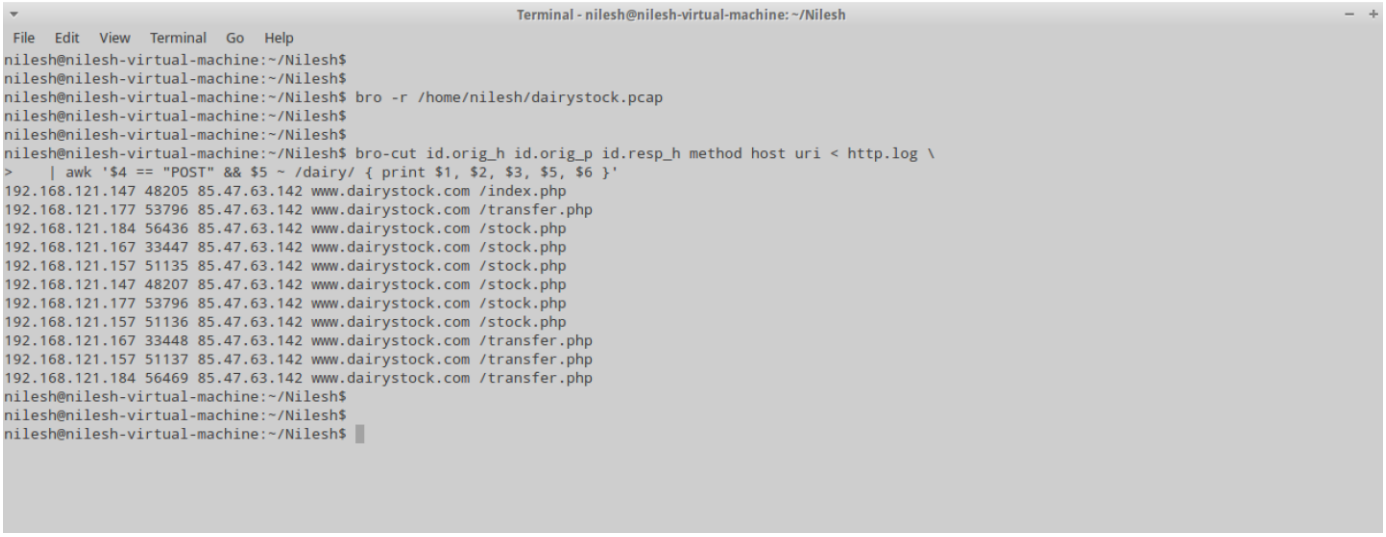
Here we have seen that 376 failed request has been sent to server. This might be because of an attacker sees the pattern of the IP address and block the request from it.

(2). We are determining the downtime of youtube connection. We will search on the basis of connection state which is equal to REJ in our case.



(3). Finding out who all the affected from this outage. For this we will check in the connection log file and check the rejection state of the request, which we have already determined. Now we will fetch the id.orig_h corresponding to the rejection status. Snapshot is below.

Part 5:The Mysterious DairyStock Transaction: In this part, we are trying to check, who the unauthorized/bogus transaction with the Internet security issues. For this, we were given a tcpdump dairystock.pcap. By this part, we are learning how to identify bogus transaction over TCP.

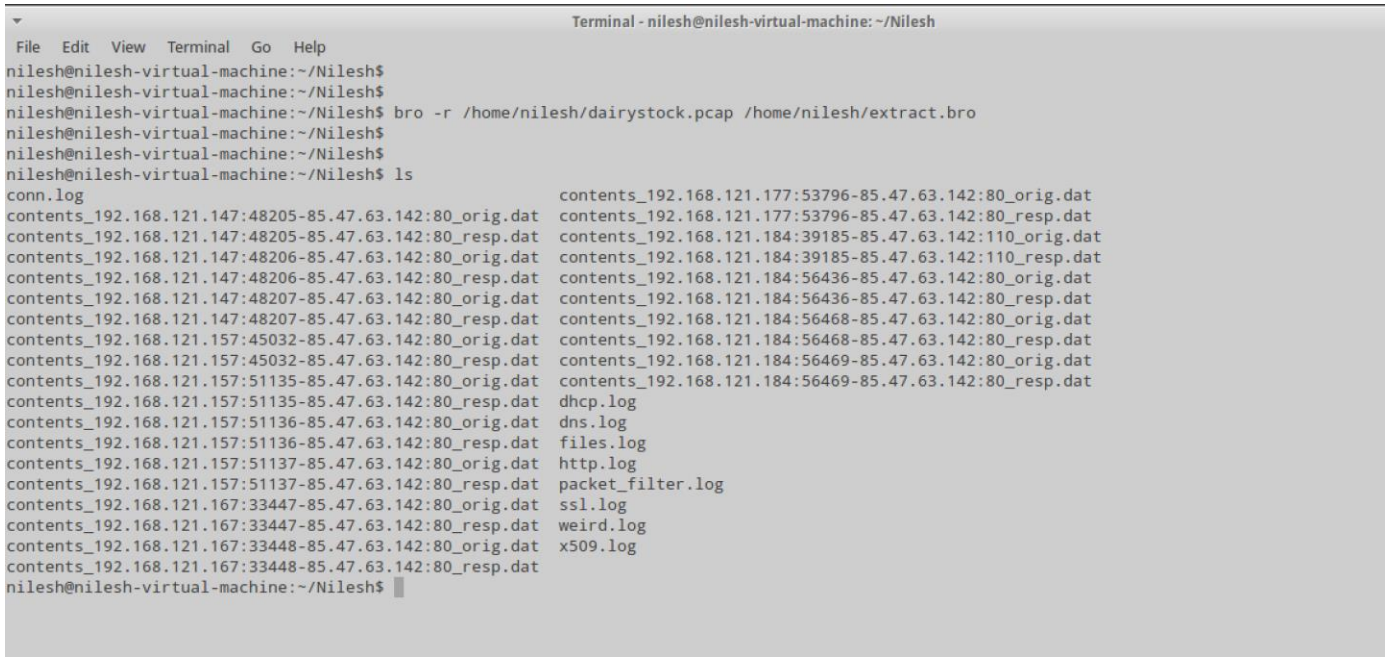


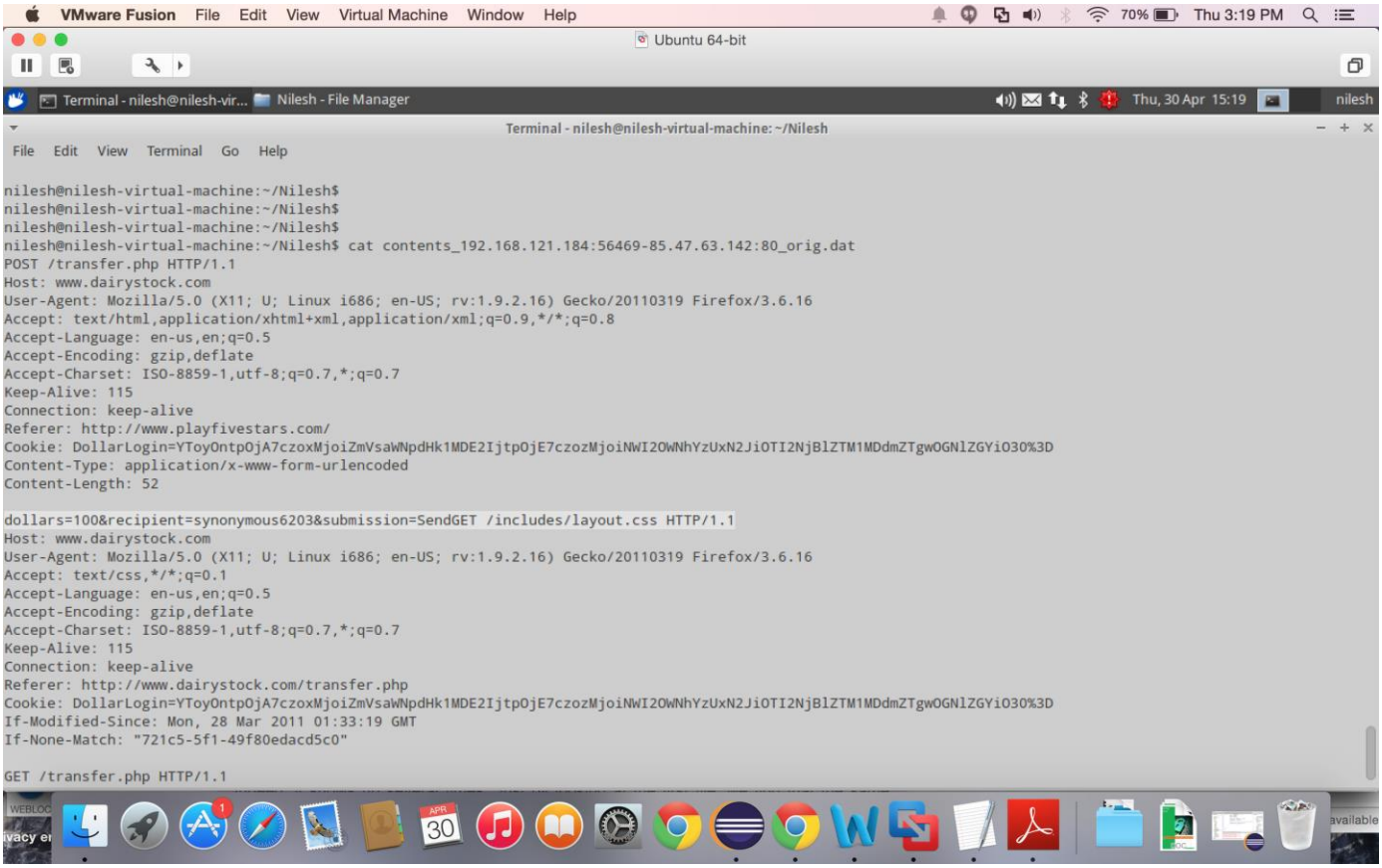
After extracting http log file, we are trying to check, who send the POST request from dairystock.com and send some data file over it.

Now, we have id.orig-h (sender ID) from http log. Now we are creating a script extract.bro in which we are writing IPs, which are generated in previous results.

Now, after generating script, we are again extracting logs from pcap file using origin ID. By doing this, we will get dat file for each corresponding IP in the script. And the connection that involves sending transfer.php, we will have there IP and port number. Now, if we check the file with that IP and port number, we will see the transaction details inside it.


Terminal - nilesh@nilesh-virtual-machine: ~
File Edit View Terminal Go Help
@load base/protocols/http
event connection_established(c: connection)
{
if ((cidorig_h == 192.168.121.147 ||
cidorig_h == 192.168.121.157 ||
cidorig_h == 192.168.121.167 ||
cidorig_h == 192.168.121.177 ||
cidorig_h == 192.168.121.184) &&
cidresp_h == 85.47.63.142)
{
c$extract_orig = T;
c$extract_resp = T;
}
}


Terminal - nilesh@nilesh-virtual-machine: ~/Nilesh
File Edit View Terminal Go Help
nilesh@nilesh-virtual-machine:~/Nilesh$
nilesh@nilesh-virtual-machine:~/Nilesh$
nilesh@nilesh-virtual-machine:~/Nilesh$ bro -r /home/nilesh/dairystock.pcap /home/nilesh/extract.bro
nilesh@nilesh-virtual-machine:~/Nilesh$
nilesh@nilesh-virtual-machine:~/Nilesh$
nilesh@nilesh-virtual-machine:~/Nilesh$ ls
conn.log
contents_192.168.121.147:48205-85.47.63.142:80_orig.dat
contents_192.168.121.147:48205-85.47.63.142:80_resp.dat
contents_192.168.121.147:48206-85.47.63.142:80_orig.dat
contents_192.168.121.147:48206-85.47.63.142:80_resp.dat
contents_192.168.121.147:48207-85.47.63.142:80_orig.dat
contents_192.168.121.147:48207-85.47.63.142:80_resp.dat
contents_192.168.121.157:45032-85.47.63.142:80_orig.dat
contents_192.168.121.157:45032-85.47.63.142:80_resp.dat
contents_192.168.121.157:51135-85.47.63.142:80_orig.dat
contents_192.168.121.157:51135-85.47.63.142:80_resp.dat
contents_192.168.121.157:51136-85.47.63.142:80_orig.dat
contents_192.168.121.157:51136-85.47.63.142:80_resp.dat
contents_192.168.121.157:51137-85.47.63.142:80_orig.dat
contents_192.168.121.157:51137-85.47.63.142:80_resp.dat
contents_192.168.121.167:33447-85.47.63.142:80_orig.dat
contents_192.168.121.167:33447-85.47.63.142:80_resp.dat
contents_192.168.121.167:33448-85.47.63.142:80_orig.dat
contents_192.168.121.167:33448-85.47.63.142:80_resp.dat
contents_192.168.121.177:53796-85.47.63.142:80_orig.dat
contents_192.168.121.177:53796-85.47.63.142:80_resp.dat
contents_192.168.121.184:39185-85.47.63.142:110_orig.dat
contents_192.168.121.184:39185-85.47.63.142:110_resp.dat
contents_192.168.121.184:56436-85.47.63.142:80_orig.dat
contents_192.168.121.184:56436-85.47.63.142:80_resp.dat
contents_192.168.121.184:56468-85.47.63.142:80_orig.dat
contents_192.168.121.184:56468-85.47.63.142:80_resp.dat
contents_192.168.121.184:56469-85.47.63.142:80_orig.dat
contents_192.168.121.184:56469-85.47.63.142:80_resp.dat
dhcp.log
dns.log
files.log
http.log
packet_filter.log
ssl.log
weird.log
x509.log
nilesh@nilesh-virtual-machine:~/Nilesh$


VMware Fusion File Edit View Virtual Machine Window Help
70% Thu 3:19 PM
Ubuntu 64-bit
Terminal - nilesh@nilesh-vir... Nilesh - File Manager
Thu, 30 Apr 15:19 nilesh
Terminal - nilesh@nilesh-virtual-machine: ~/Nilesh
File Edit View Terminal Go Help
nilesh@nilesh-virtual-machine:~/Nilesh$
nilesh@nilesh-virtual-machine:~/Nilesh$
nilesh@nilesh-virtual-machine:~/Nilesh$
nilesh@nilesh-virtual-machine:~/Nilesh$ cat contents_192.168.121.184:56469-85.47.63.142:80_orig.dat
POST /transfer.php HTTP/1.1
Host: www.dairystock.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.16) Gecko/20110319 Firefox/3.6.16
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Referer: http://www.playfivestars.com/
Cookie: DollarLogin=YToyOntpOjA7czoxMjoiZmVsaWNpdHk1MDE2IjtpOjE7czozMjoiNWI2OWNhYzUxN2JiOTI2NjBlZTM1MDdmZTgwOGNlZGYiO30%3D
Content-Type: application/x-www-form-urlencoded
Content-Length: 52
dollars=100&recipient=synonymous6203&submission=SendGET /includes/layout.css HTTP/1.1
Host: www.dairystock.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.16) Gecko/20110319 Firefox/3.6.16
Accept: text/css,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Referer: http://www.dairystock.com/transfer.php
Cookie: DollarLogin=YToyOntpOjA7czoxMjoiZmVsaWNpdHk1MDE2IjtpOjE7czozMjoiNWI2OWNhYzUxN2JiOTI2NjBlZTM1MDdmZTgwOGNlZGYiO30%3D
If-Modified-Since: Mon, 28 Mar 2011 01:33:19 GMT
If-None-Match: "721c5-5f1-49f80edacd5c0"
GET /transfer.php HTTP/1.1

==========

Ans. 1-F: http://www.bro.org/bro-workshop-2011/solutions/advanced-http/index.html

Exercise: Advanced HTTP Traffic Analysis:

Part 1: Building a Sidejacking Detector:

This part of exercise demonstrates us how to create a sidejacking detector. Sidejacking is when someone steals your cookies and uses it to impersonate you on a particular website. The tutorial gives the example of Brooke using Brody's cookies to impersonate him on Twitter. Network operators and incident responders clearly would like to find out when their users' security and privacy is compromised. How do we detect it?

At a very basic level, we have to monitor each session cookie and observe whether it appears in more than one user context. That is, if one IP address uses a cookie previously used by another IP address, we have likely witnessed a sidejacking incident.

The second thought is that of session cookie reuse where we reuse the same cookie on different browsers.

Exercise1: It asks us to add the cookie information to the http log file, since it is not added by default. We create a .bro file to do this.

```
  GNU nano 2.2.6                File: sidejack-detect.bro

module HTTP;

export {
    redef record Info += {
        # Extend this record with an optional, logged field named "cookie".
        cookie: string &log &optional;
    };
}


# Track the cookie value inside HTTP.
event http_header(c: connection, is_orig: bool, name: string, value: string)
    {
    if ( ! is_orig )
        return;

    # Write the cookie value into the HTTP state of the connection.
    if ( name == "COOKIE" )
        c$http$cookie = value;
    }
```

Then Used following command:

bro –r twitter.pcap sidejack-detect.bro

Now, the cookie field has been added to the log file.

```
-virtual-machine:~/sidejack$ grep "cookie" http.log
 uid      id.orig_h       id.orig_p       id.resp_h       id.resp_
epth      method  host    uri     referrer        user_agent      r
 response_body_len       status_code     status_msg      info_cod
;         filename        tags    username        password        p
ids       orig_mime_types resp_fuids      resp_mime_types cookie
-virtual-machine:~/sidejack$ cat http.log
```

Next exercise is to check whether the cookie has been seen with different user agents.
We write the below .bro file.

```
module HTTP;

export {
    redef enum Notice::Type += {
        ## Cookie reuse by a different user agent
        SessionCookieReuse
    };

    # We track the cookie inside the HTTP state of the connection.
    redef record Info += {
        cookie: string &log &optional;
    };
}

const twitter_cookie_keys = set("_twitter_sess", "auth_token");

# Map cookies to user agents.
global cookies: table[string] of string;
```

It keeps track of which cookie is using what browser and as soon as the same cookie uses a different browser, it detects it.
Used following commands:
nano solution.bro

bro –r twitter.pcap solution.bro

cat notice.log

```
#empty_field    (empty)
#unset_field    -
#path   notice
#open   2015-04-15-09-34-05
#fields ts      uid     id.orig_h       id.orig_p       id.resp_h       id.res
p       fuid    file_mime_type  file_desc       proto   note    msg     sub
rc      dst     p       n       peer_descr      actions suppress_for    droppe
emote_location.country_code     remote_location.region  remote_location.city
emote_location.latitude remote_location.longitude
#types  time    string  addr    port    addr    port    string  string  string
num     enum    string  string  addr    addr    port    count   string  set[en
]       interval        bool    string  string  string  double  double
1320257580.095086       COcH1123JafADEbB53      192.150.187.147 51742   199.59
49.198  80      -       -       -       tcp     HTTP::SessionCookieReuse
eused Twitter session via cookie _twitter_sess=BAh7DzoVaW5fbmV3X3VzZXJfZmxvdzA
```

Part 2: Building a Facebook Chat Analyzer:

This part tells us to recover the transcript from a Facebook chat. In first part of this exercise, we create a data structure ChatMessage and write a function for it.

```
## Reassemble the HTTP body of replies and look for Facebook chat messages.
event http_body(c: connection, is_orig: bool, data: string, size: count)
    {
    # Only consider chat messages for now.
    if (/^for \(;;\);\{\"t\":\"msg\".*text\":\"/ !in data)
        return;

    local msg = parse_fb_message (data);

    local i: Info;
    i$timestamp = msg$timestamp;
    i$chat_from = msg$from;
    i$chat_to = msg$to;
    i$chat_msg = msg$text;

    Log::write(Facebook::LOG, i);
    }
```

Running this .bro file is the next part.

bro –r url.pcap facebook-solution.bro

cat facebook.log

```
#path   facebook
#open   2015-04-15-09-53-29
#fields timestamp       chat_from       chat_to chat_msg
#types  string  string  string  string
1303218454567   Mondo Cheeze    Udder Kaos      So I need the URL, dude.  What is it?
1303218465938   Udder Kaos      Mondo Cheeze    the URL?
1303218474259   Mondo Cheeze    Udder Kaos      Yeah for the secret image
```

```
1303218454567   Mondo Cheeze    Udder Kaos      So I need the URL, dude.  What is it?
1303218465938   Udder Kaos      Mondo Cheeze    the URL?
1303218474259   Mondo Cheeze    Udder Kaos      Yeah for the secret image
1303218481721   Udder Kaos      Mondo Cheeze    ok lemme see
1303218495626   Mondo Cheeze    Udder Kaos      Someone could be sniffing this conversation, be s
ure to send it safely
1303218503972   Udder Kaos      Mondo Cheeze    ?
1303218570782   Mondo Cheeze    Udder Kaos      Cmon we talked about this.  Encrypt it with Wonde
rCipher-92 and send me the Base64 encoding of the hex.  Usual key.
1303218587568   Udder Kaos      Mondo Cheeze    'k.  So here it is:
1303218595067   Udder Kaos      Mondo Cheeze    NmQwMDJjZDdhZTdlYmYxNTc5MGVjZDc1YTYxNDk1OGE0ZTRhY
jAzOTVi
1303218618252   Mondo Cheeze    Udder Kaos      What's the IV
1303218624712   Udder Kaos      Mondo Cheeze    huh?
1303218637197   Mondo Cheeze    Udder Kaos      Initialization vector, you maroon.  WC-92 is a st
ream cipher, you know
1303218667601   Udder Kaos      Mondo Cheeze    oh yeah.  I used my birthday, all as one number.
1303218685436   Udder Kaos      Mondo Cheeze    you *do* remember it, right?
1303218700515   Mondo Cheeze    Udder Kaos      yeah your an April Fool, not hard to remember
1303218710402   Udder Kaos      Mondo Cheeze    heh
1303218718486   Mondo Cheeze    Udder Kaos      K gimme a sec to decrypt then.
1303218733463   Mondo Cheeze    Udder Kaos      Hey idiot this isn't the secret, it's Google's ho
me page.
1303218745028   Udder Kaos      Mondo Cheeze    whoops hang on, blew my cut&paste
1303218767633   Udder Kaos      Mondo Cheeze    okay, here's the right one:
1303218776922   Udder Kaos      Mondo Cheeze    NmQwMDJjZDdhZTdlYmYwMDY3MGRjZDdlYjA1NDlhODQ0ZjA1Y
mEyNDRm
1303218800303   Mondo Cheeze    Udder Kaos      And?
1303218807537   Udder Kaos      Mondo Cheeze    and what
1303218815022   Mondo Cheeze    Udder Kaos      What's the IV
1303218824330   Udder Kaos      Mondo Cheeze    huh?
```

In this way, we have used Bro to extract the transcript of the Facebook chat.

Extra credit:

Can you reconstruct the encrypted URL?

Yes, we can. using the same IV twice is not safe.

We define some variables below:
B64D:= function that decodes a Base64 encoded string
P1:= "http://www.google.com"
P2:= (unknown)
C1:= B64D (NmQwM...zOTVi)
C2:= B64D (NmQwM...mNDRm)
$\oplus$:= XOR operation
If R is the keystream used,
$C1 \oplus C2 = (P1 \oplus R) \oplus (P2 \oplus R) = P1 \oplus P2$
Which is same as
$P2 = C1 \oplus C2 \oplus P1$
We can find the value of P2 by plugging in the values of C1, C2 and P1. In the above example it is http://bit.ly/hbdairy

---

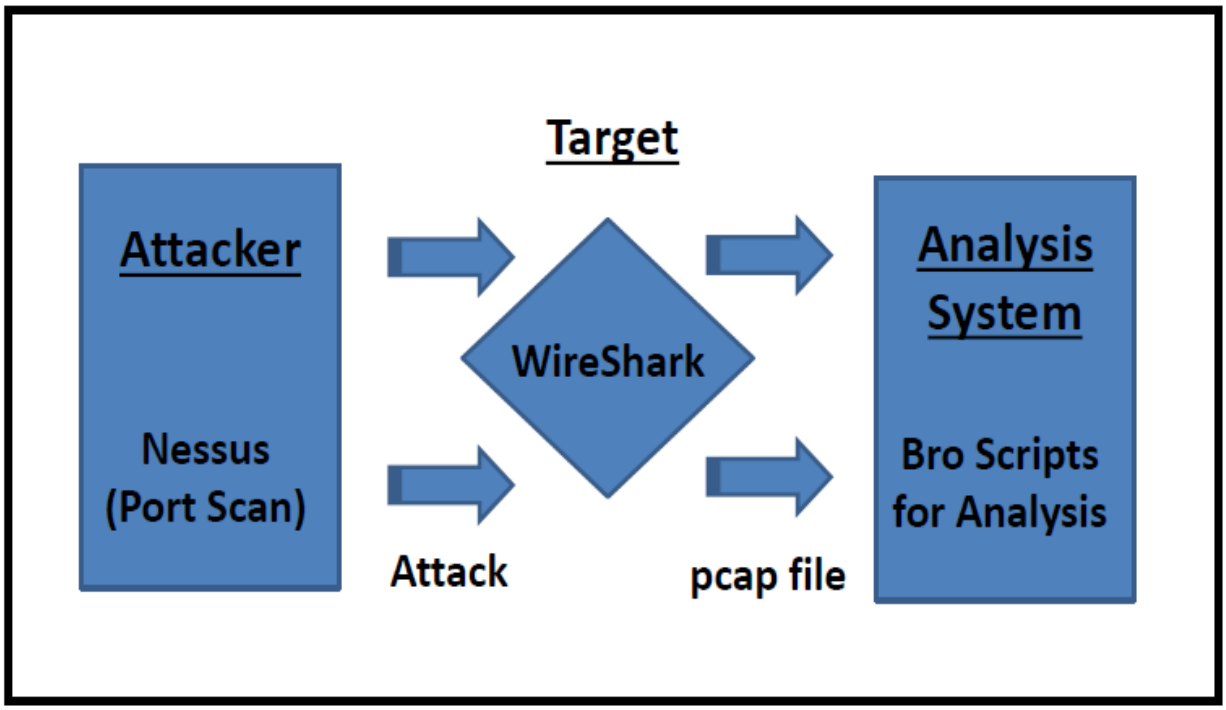
==========

Ans. 2:

2.Port Scan Attack Using Nessus

Most attacks start with a recognition phase, where an attacker looks for attack vectors in one or several victim systems. Port scanning is arguably the most widely used technique by both worms and human attackers to probe for vulnerabilities in Internet systems. In our project we attempt to address and implement the problem of how to efficiently and reliably port scans can be detected without to many false positives.

The below figure briefly explains what we are going to implement in our project, the main agenda is going to be to detect the port scan attack that we will perform using the penetration testing tool called Nessus and Wireshark by running a bro script on security onion.



The below section gives us a brief understanding of functionality of each of the components mentioned above.

Port Scan Attack

Port scanning can be defined as hostile Internet searches for open doors or ports, through which intruders gain access to computers. This technique consist of sending a message to a port and listening for an answer. The received response indicates the port status and can be helpful in determining a host's operating system and other information relevant to launching a future attack.

Ports are like little doors on our systems. Most packets leaving the machine come out of a certain door. They are destined for another door on another system. There are two different protocols that use ports: TCP and UDP.

An attacker launches a port scan to see what ports are open, with a listening service, on the machine. A port scan attack, therefore, occurs when an attacker sends packets to your machine, varying the destination port. The attacker can use this to find out what services are running and to get a pretty good idea of the operating system it has.

Hackers can compromise the vulnerable hosts and can either take over their resources or use them as tools for future attacks. With so many different protocols and countless implementations of each for different platforms, the launch of an effective attack often begins with a separate process of identifying potential victims.

Port scans can be classified into **three basic types** based on the pattern of target destinations and ports the scan explores, and they are:

Vertical Scans :The vertical scan is a port scan that targets several destination ports on a single host. Naively executed, this scan is among the easiest to detect because only local (singlehost) detection mechanisms are required.
Horizontal Scans:A horizontal scan is a port scan that targets the same port on several hosts. Most often the attacker is aware of a particular vulnerability and wishes to find susceptible machines. One would expect to see many horizontal scans for a particular port immediately following the publicizing of a vulnerability on that port.
Block Scans:Some attackers combine vertical and horizontal scanning styles into large sweeps of the address-port space.

Nessus

The penetration testing tool we have chosen is Nessus. Nessus is a remote security scanning tool, which scans a computer and raises an alert if it discovers any vulnerabilities that malicious hackers could use to gain access to any computer you have connected to a network. It does this by running over 1200 checks on a given computer, testing to see if any of these attacks could be used to break into the computer or otherwise harm it. If you are an administrator in charge of any computer (or group of computers) connected to the internet, Nessus is a great tool help keep their domains free of the easy vulnerabilities that hackers and viruses commonly look to exploit.

Why we used Nessus??
Unlike other scanners, Nessus does not make assumptions about your server configuration (such as assuming that port 80 must be the only web server) that can cause other scanners to miss real vulnerabilities.
Nessus is very extensible, providing a scripting language for you to write tests specific to your system once you become more familiar with the tool. It is also provides a plug-in interface, and many free plug-ins are available from the Nessus plug-in site. These plugs are often specific to detecting a common virus or vulnerability.
Up to date information about new vulnerabilities and attacks. The Nessus team updates the list of what vulnerabilities to check for on a daily basis in order to minimize the window between an exploit appearing in the wild, and you being able to detect it with Nessus.

Wireshark

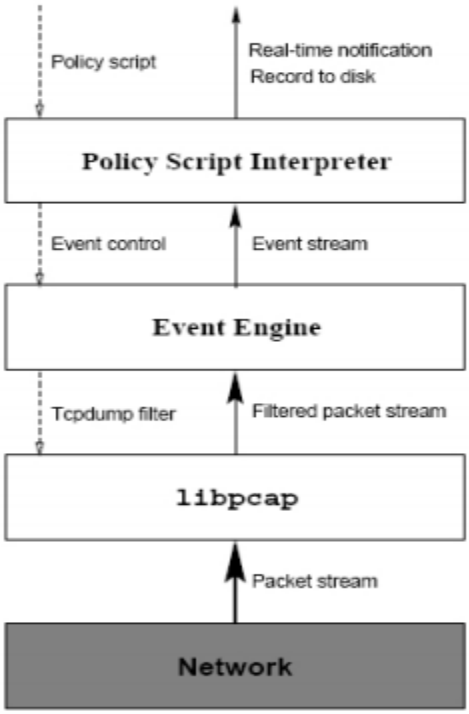
Wireshark, a network analysis tool formerly known as Ethereal, captures packets in real time and display them in human-readable format. Wireshark includes filters, color-coding and other features that let you dig deep into network traffic and inspect individual packets. We will use Wireshark to capture the the traffic while the port scan by Nessus is running and then the resulting **.pcap file** will be analyzed using bro script on security onion.

Bro

Bro is an intrusion detection system. Bro was intentionally a stand alone system for detecting network intruders in real time. Bro is open source software, written in C and is capable of deep & stateful packet inspection at very high speed. Bro IDS is a powerful real-time network traffic analyzing tool.

Bro - Internal Architecture:

Bro internal architecture consists roughly of the following elements.
1. libpcap
2. an event engine and
3. a policy script interpreter.



Network: Bro needs a physical network connection to get a copy of the network traffic it will analyze. This is normally done by the use of port mirroring functionality in switches/routers or a TAP device. **libpcap** :In order to get traffic data from the physical network, a so-called Application Program Interface (API) is needed. Bro uses libpcap. libpcap is a C/C++ library for network traffic capture. This is the abstract layer between the physical network medium and the operating system. With the use of libpcap Bro filters traffic that is sent to Bro Event Engine.
Event Engine :The filtered network data packages from libpcap are then fed into the next level; the Event engine. This event engine tries to reassemble all the network traffic it gets to known events/patterns as high as possible in the OSI ISO Model.
Policy Script Interpreter :The policy script interpreter processes events from the event engine. For every event handled to the policy script interpreter, it performs the following steps:
1. Look up the corresponding event handler's (semi-)compiled code/script
2. Bind the value(s) of the event(s) to the argument of the handler
3. Interpret the actual event code/script

Bro Log Files:
conn.log — consists of the complete connection log during Bro's run time. The file consists of twenty columns (timestamp, connection ID (unique), source IP, dest IP, source port, dest port etc.).
loaded_scripts.log — shows Bro scripts (*.bro) that were loaded during Bro startup.
notice_policy.bro — shows the current Bro Notice policy.
communication.log — logs for Bro's internal communication between remote and central instances, clusters etc.
conn-summary.log — generated when Bro is terminated. Post processing connection summaries

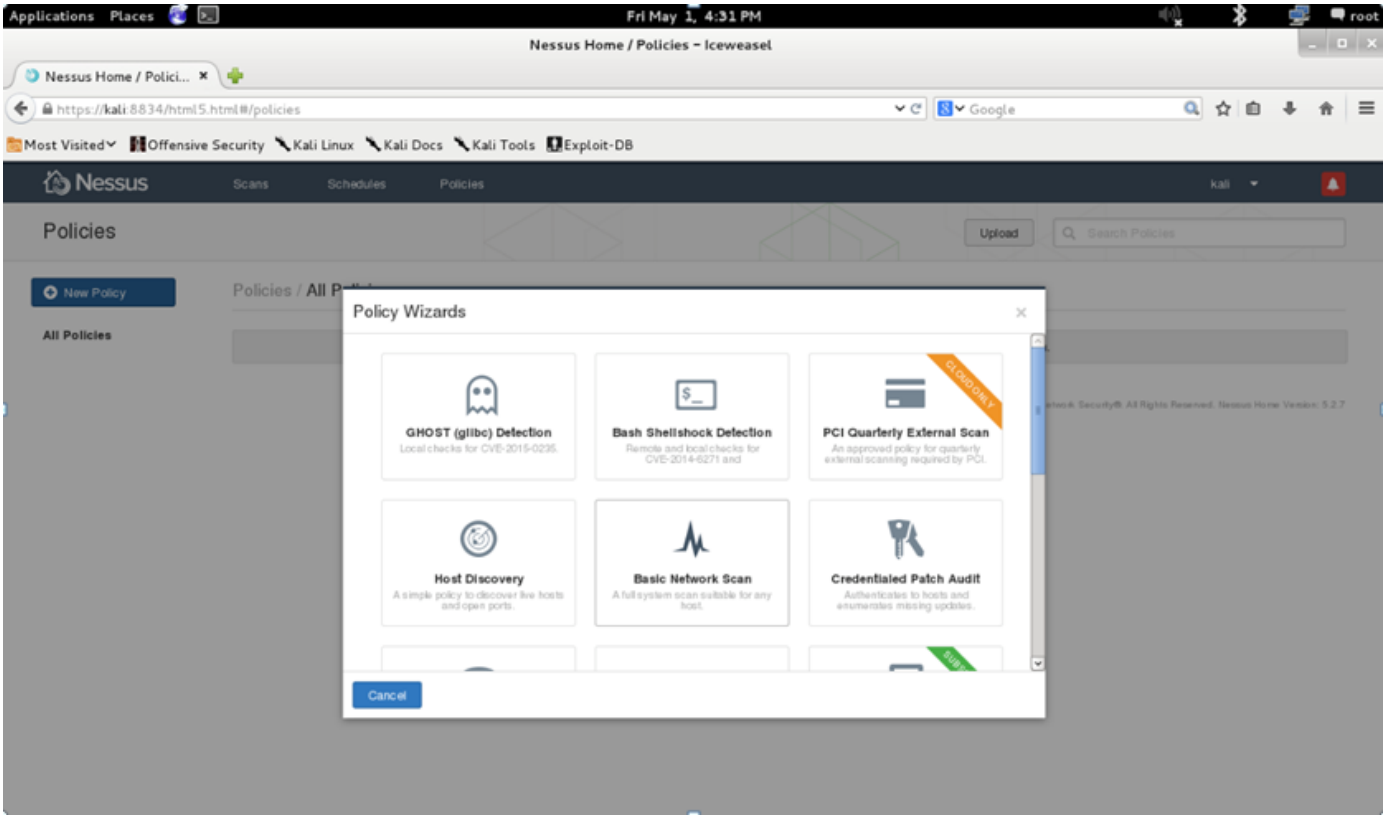
known_hosts.log — hosts that have performed complete TCP handshake
notice.log — notices that Bro rises
reporter.log — internal messages/warnings/errors for troubleshooting.

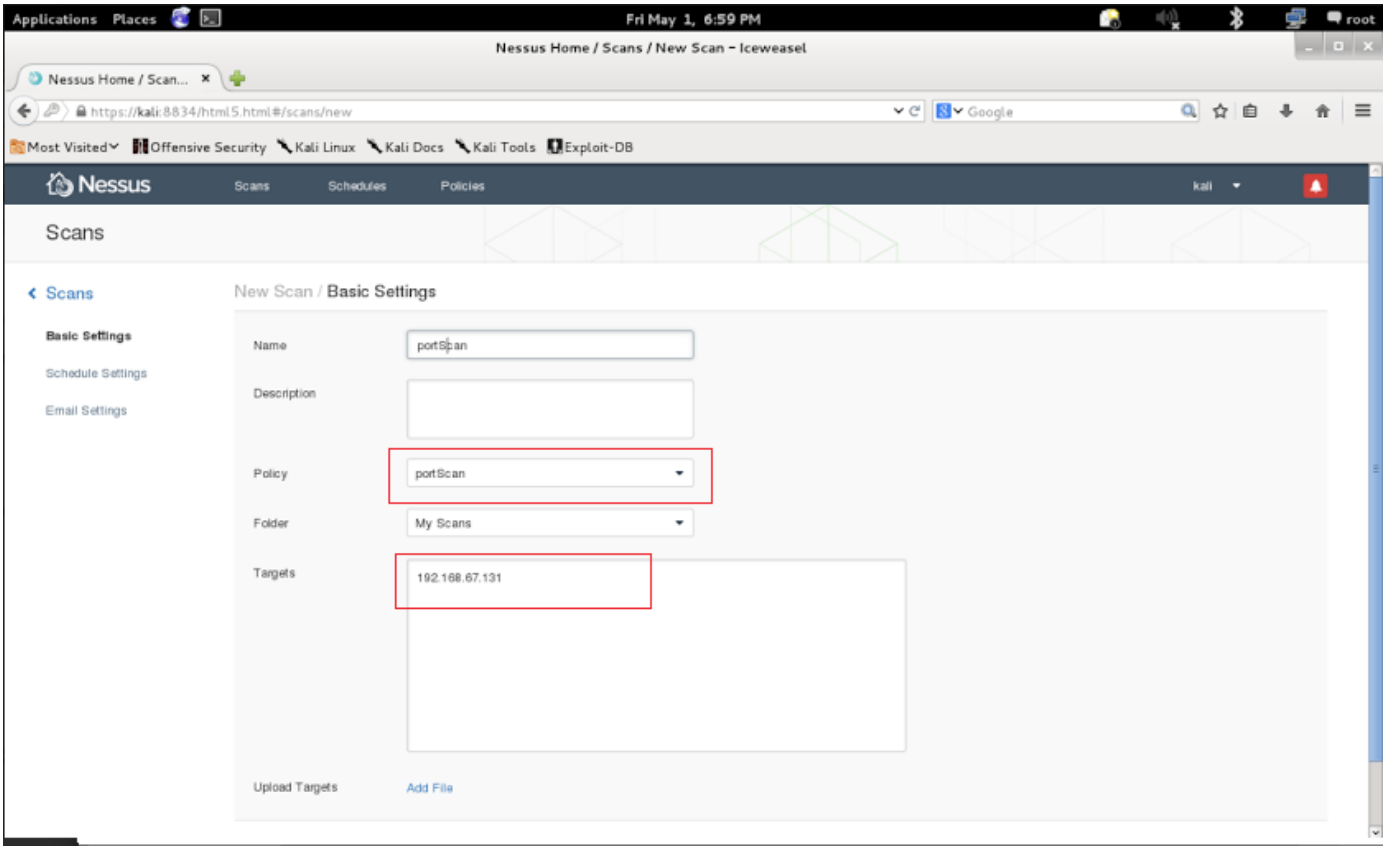
Implementation

1.Nessus

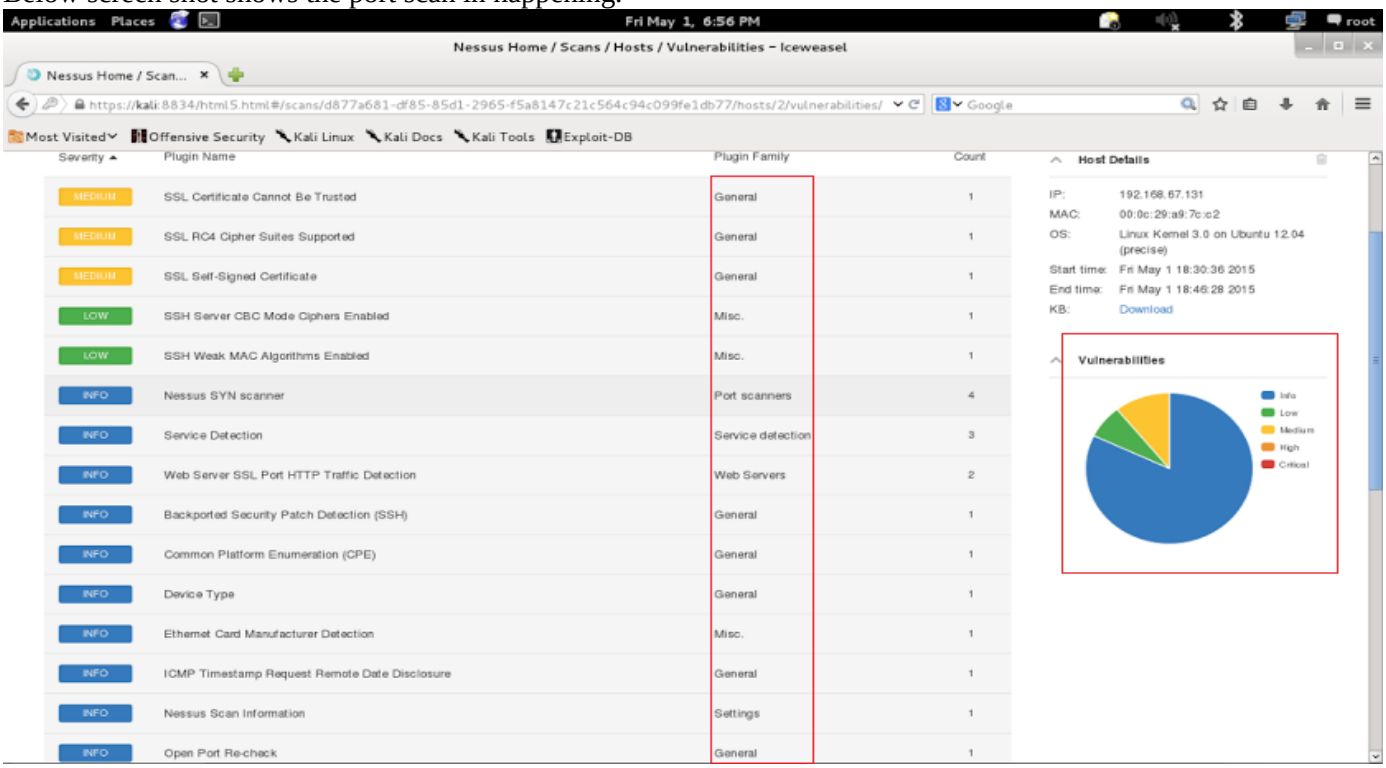
The attacker uses Nessus penetration testing tool to implement the Port Scan attack. Below screen shots show how this is performed.

After Starting Nessus we will create a policy by running the port scan and giving the ip address of the system we are trying to find the vulnerability.



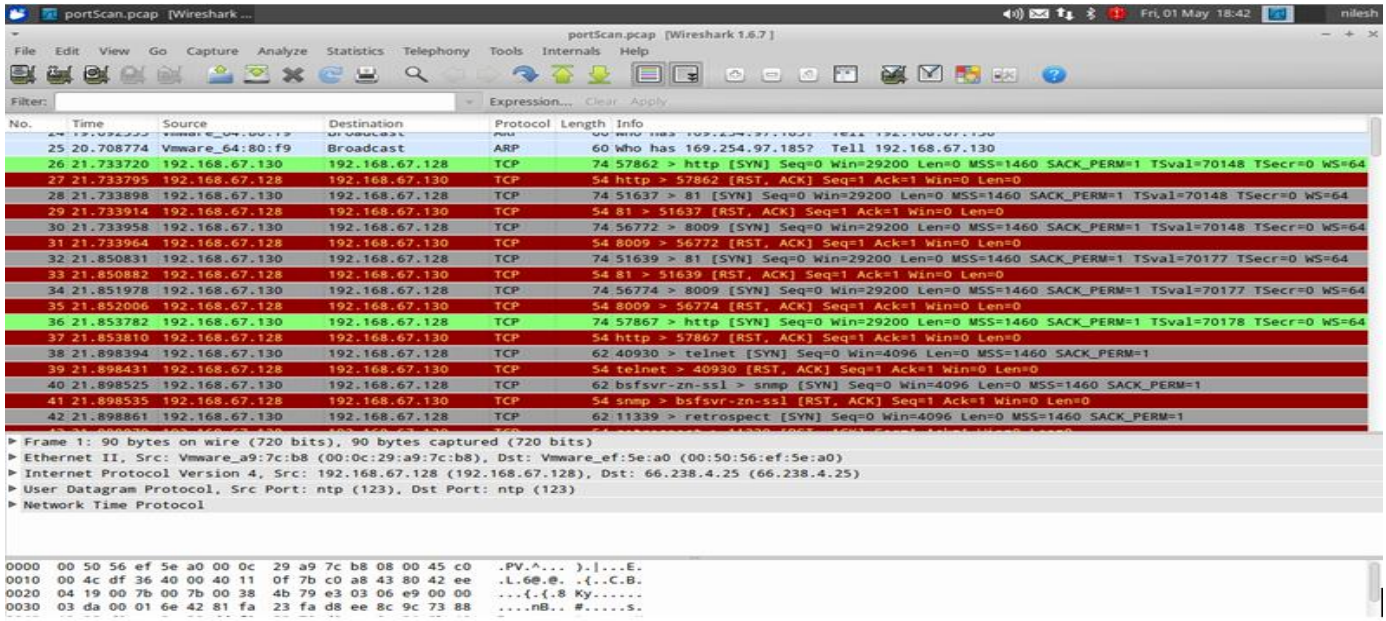


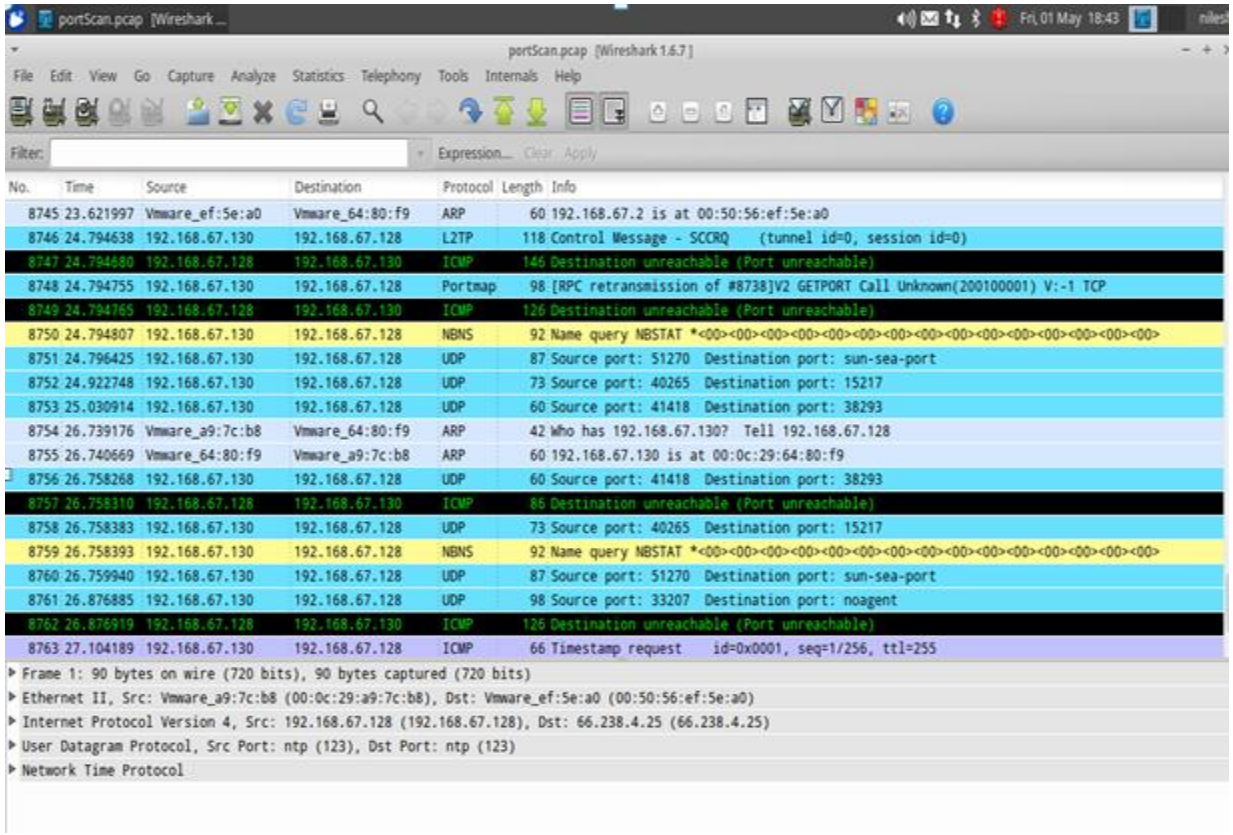
Below screen shot shows the port scan in happening.



2.WireShark

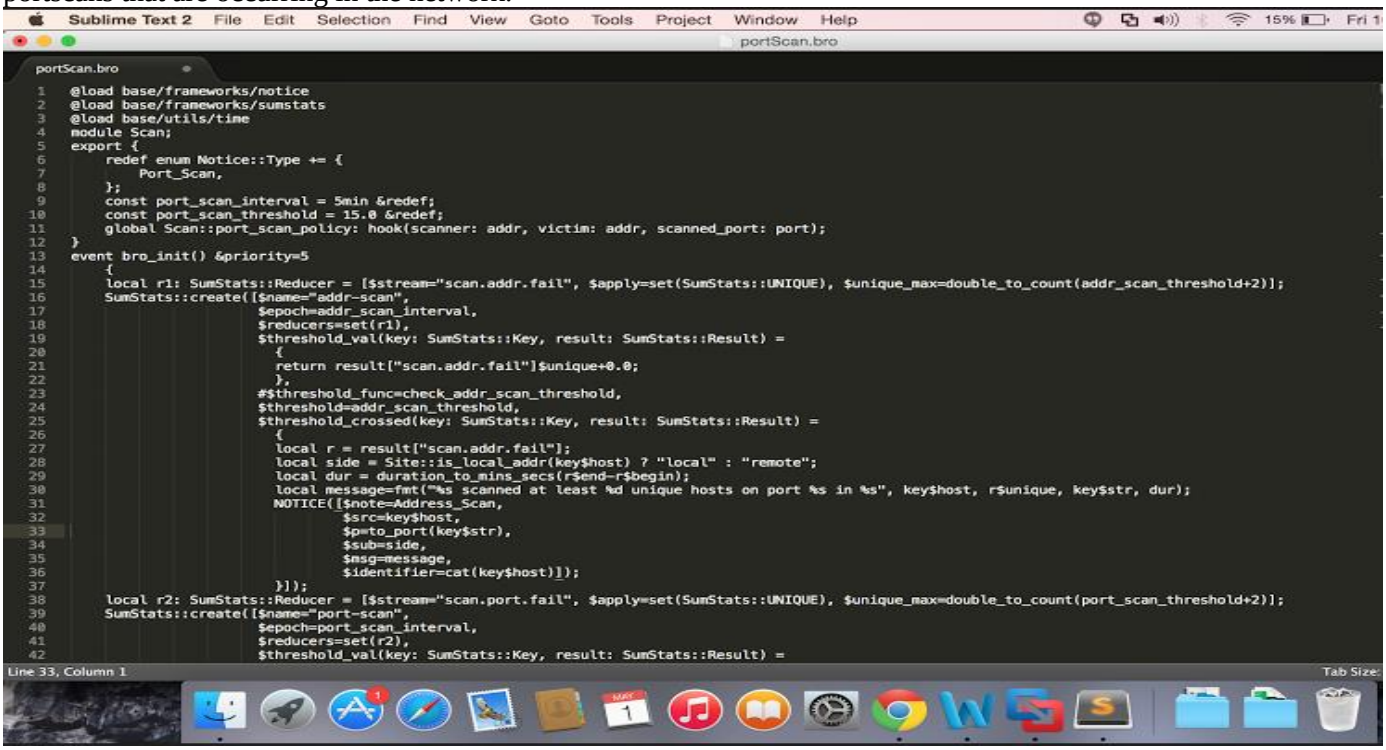
Now we are going to start the capture on the Wireshark to capture the network activity on the victims system. This is going to produce the .pcap file which we are going to use for bro analysis, checking the tcpdump in Wireshark for port scan.

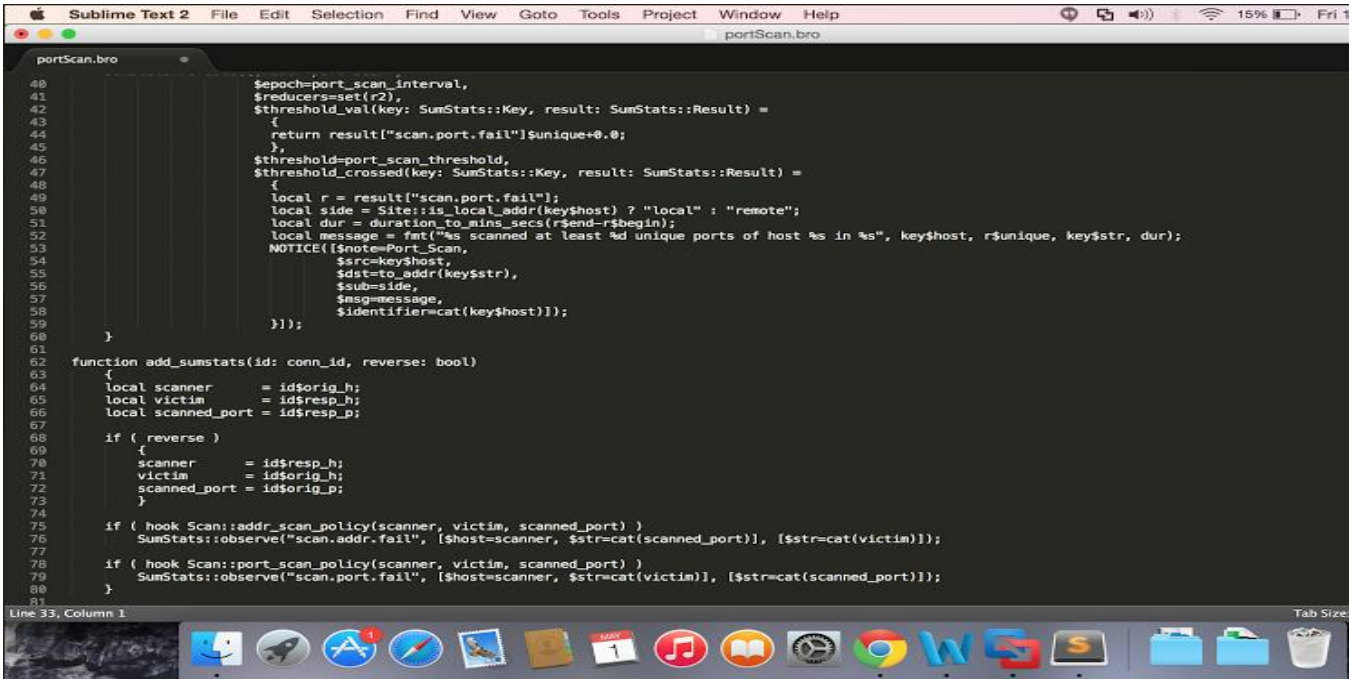




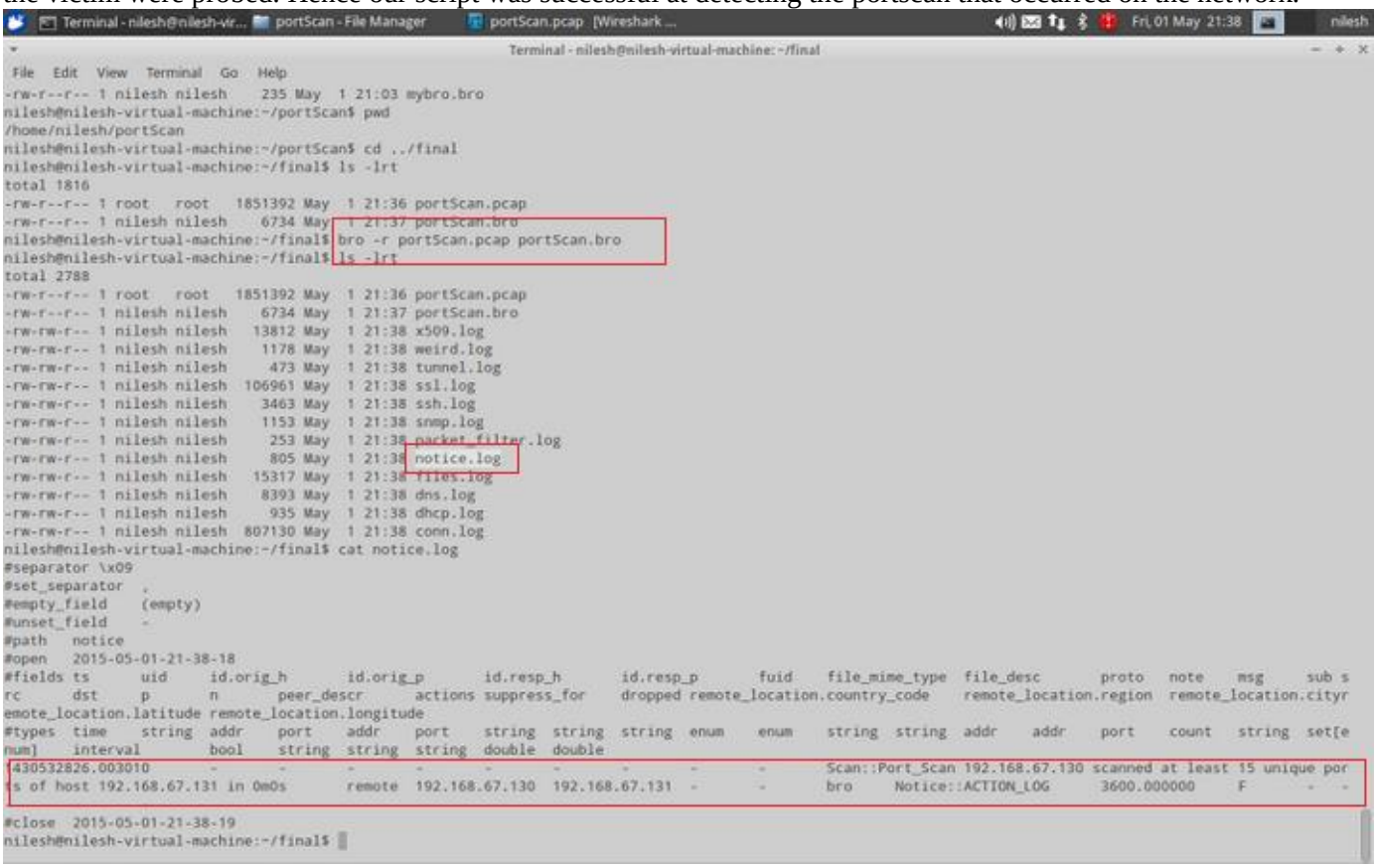
3. Bro

So here we have written a script that detects for the portscans occurring on the network, and it creates an entry in the notice.log file. Below screenshot shows the code we have implemented. This code keeps track of the number of portscans that are occurring in the network.





Below screenshot shows that after running our bro script it made an analysis on the .pcap file to find that 15 ports of the victim were probed. Hence our script was successful at detecting the portscan that occurred on the network.



Conclusion

Port Scanning means that a system has targeted one system, connecting to one port, then another, until it has scanned a range of ports. Bro labels unusual or exceptional events as weird, they include a wide range of events, including malfunctioned connections, attackers attempts to confuse or evade being detected, malfunctioning hardware, or misconfigured system or network devices such as routers and offcourse attacks like port scans. Some weird events might be an attack or some anomalies. The better we know our system the more likely we are going to be to detect the events to be genuine or an attack.

So as future work of our project we could next implement where weird activities will be detected and an alarm will be raised , care should be taken that genuine events are not mistaken to be attack attempts hence reducing false positives. We could write policies that will differentiate attacks from regular genuine events. Every system is vulnerable to port scanning. Before a system is placed online, port scan should be performed against the system. If a system is compromised, attacker may try to open more ports on the system so they can more easily exploit the port weaknesses.

To secure the unauthorized access on a port, when a request is coming on the port, we can first check whether the host is allowed for the port access. If there is no access, then it cannot send the request to port.

3. Password Security

Ans. 3-A: **Describe the format of the file (e.g., userid: hashed password: etc...)**

crack01:mrmOpRH5FyJAo:526:531::/home/crack01:/bin/bash

Each field is separated with a ":" colon character, and are as follows:

- **crack01→**Username, up to 8 characters. Case-sensitive, usually all lowercase
- **mrmOpRH5FyJAo→**in the password field. Passwords are stored in the ``/etc/shadow" file.
- **526→**Numeric user id. This is assigned by the ``adduser" script. Unix uses this field, plus the following group field, to identify which files belong to the user.
- **531→**Numeric group id. Red Hat uses group id's in a fairly unique manner for enhanced file security. Usually the group id will match the user id.
- **:" ":→**Full name of user. I'm not sure what the maximum length for this field is, but try to keep it reasonable (under 30 characters).
- **home →**User's home directory. Usually /home/username (eg. /home/smithj). All user's personal files, web pages, mail forwarding, etc. will be stored here.
- **:/bin/bash →**User's "shell account". Often set to ``/bin/bash" to provide access to the bash shell (my personal favorite shell).

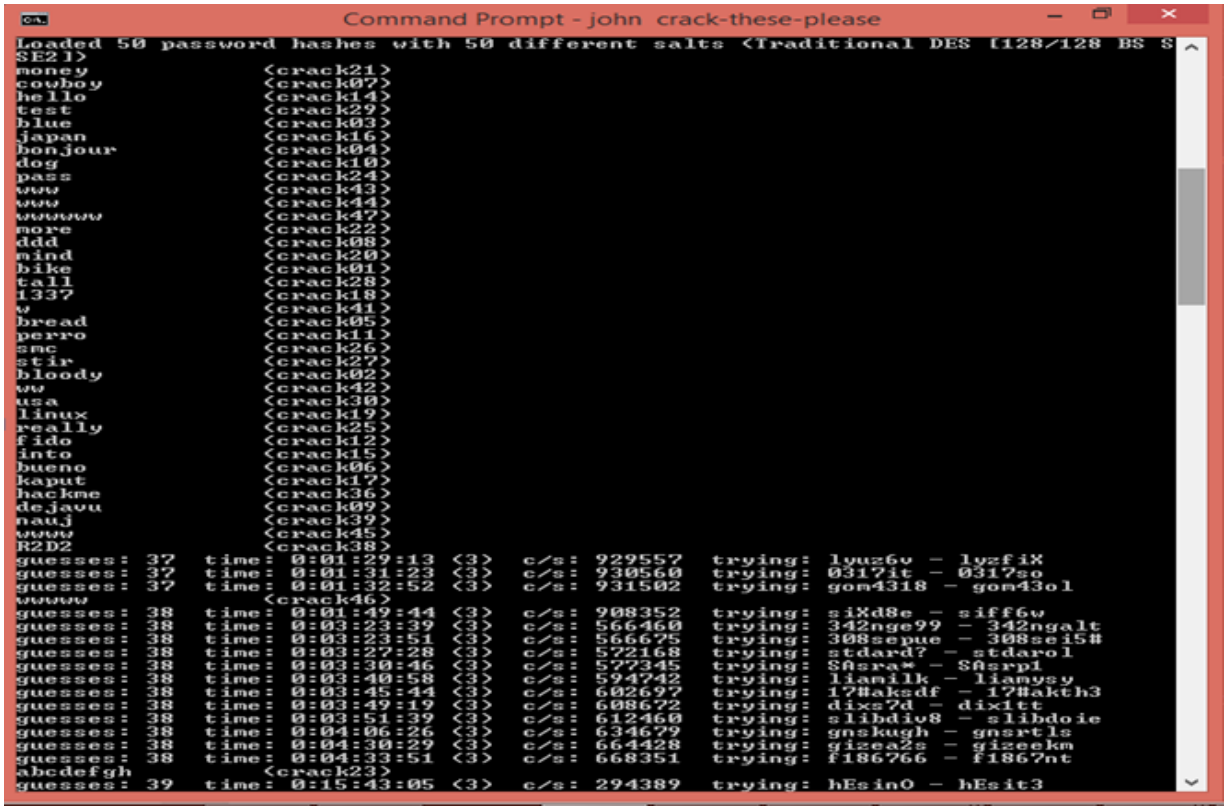
"crack-these-please"

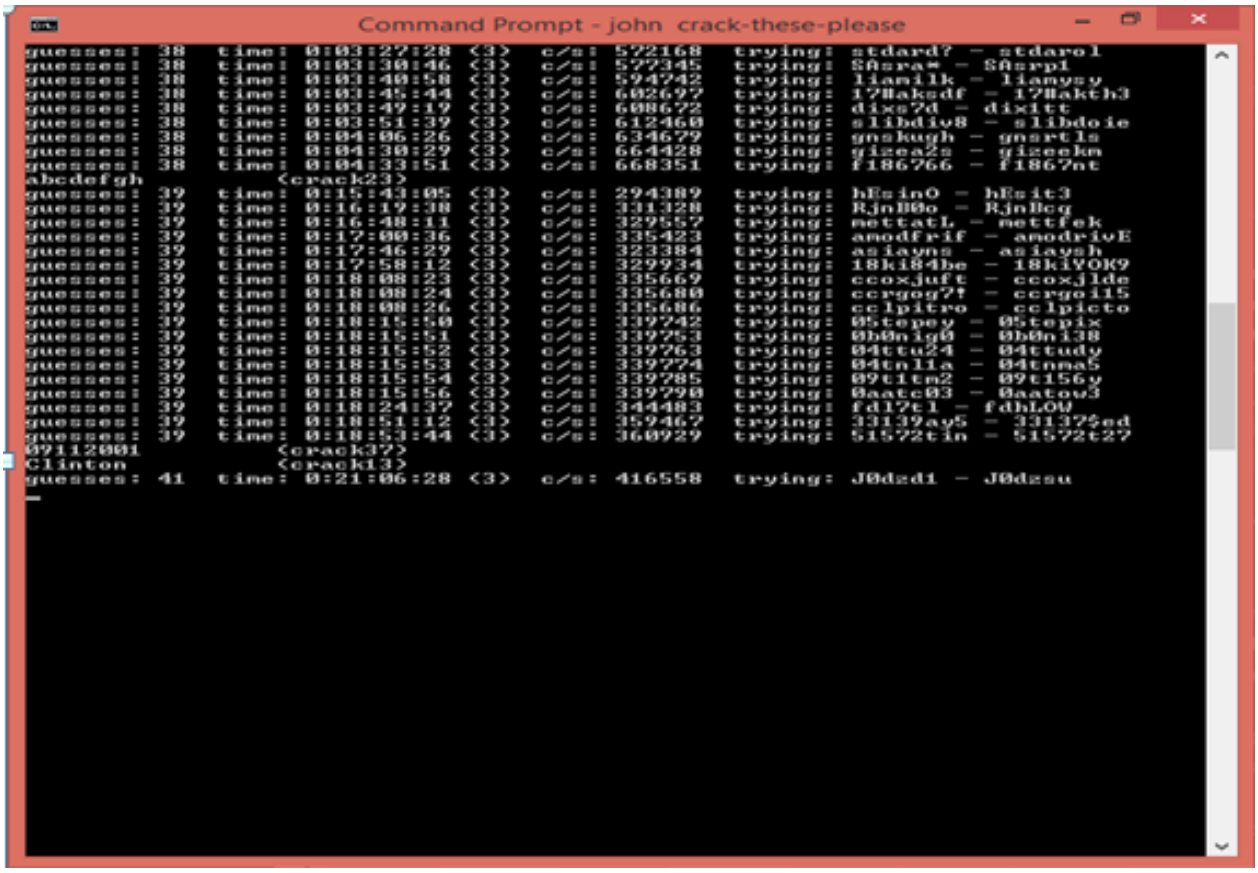
After installing john-the-ripper and placing the crack-these-please into the run folder of john, ran the command

john crack-these-please

1.crack01:**bike**	14.crack14:**hello**	27.crack27:**stir**	40.crack28:**tall**
2.crack02:**bloody**	15.crack15:**into**	28.crack29:**test**	41.crack47:**wwwwww**
3.crack03:**blue**	16.crack16:**japan**	29.crack30:**usa**	
4.crack04:**bonjour**	17. crack17:**kaput**	30.crack36:**hackme**	
5.crack05:**bread**	18.crack18:**1337**	31.crack37:**09112001**	
6.crack06:**bueno**	19.crack19:**linux**	32.crack38:**R2D2**	
7.crack07:**cowboy**	20.crack20:**mind**	33.crack39:**nauj**	
8.crack08:**ddd**	21. crack21:**money**	34.crack41:**w**	
9.crack09:**dejavu**	22. crack22:**more**	35.crack42:**ww**	
10.crack10:**dog**	23.crack23:**abcdefgh**	36.crack43:**www**	
11.crack11:**perro**	24.crack24:**pass**	37.crack44:**www**	
12.crack12:**fido**	25.crack25:**really**	38.crack45:**wwww**	
13.crack13:**Clinton**	26.crack26:**smc**	39.crack46:**wwwww**	

Below are the screenshots of the same:


Command Prompt - john crack-these-please
Loaded 50 password hashes with 50 different salts (Traditional DES [128/128 BS S
SE2])
money (crack21)
cowboy (crack07)
hello (crack14)
test (crack29)
blue (crack03)
japan (crack16)
bonjour (crack04)
dog (crack10)
pass (crack24)
www (crack43)
www (crack44)
wwwwww (crack47)
more (crack22)
ddd (crack08)
mind (crack20)
bike (crack01)
tall (crack28)
1337 (crack18)
w (crack41)
bread (crack05)
perro (crack11)
smc (crack26)
stir (crack27)
bloody (crack02)
ww (crack42)
usa (crack30)
linux (crack19)
really (crack25)
fido (crack12)
into (crack15)
bueno (crack06)
kaput (crack17)
hackme (crack36)
dejavu (crack09)
nauj (crack39)
wwww (crack45)
R2D2 (crack38)
guesses: 37 time: 0:01:29:13 (3) c/s: 929557 trying: lyuz6v - lyzfiX
guesses: 37 time: 0:01:31:23 (3) c/s: 930560 trying: 0317it - 0317so
guesses: 37 time: 0:01:32:52 (3) c/s: 931502 trying: gom4318 - gom43o1
wwwww (crack46)
guesses: 38 time: 0:01:49:44 (3) c/s: 908352 trying: siXd8e - siff6v
guesses: 38 time: 0:03:23:39 (3) c/s: 566460 trying: 342nge99 - 342ngalt
guesses: 38 time: 0:03:23:51 (3) c/s: 566675 trying: 308sepue - 308sei5#
guesses: 38 time: 0:03:27:28 (3) c/s: 572168 trying: stdard? - stdarol
guesses: 38 time: 0:03:30:46 (3) c/s: 577345 trying: $Asra* - $Asrp1
guesses: 38 time: 0:03:40:58 (3) c/s: 594742 trying: liamilk - liamysy
guesses: 38 time: 0:03:45:44 (3) c/s: 602697 trying: 17#aksdf - 17#akth3
guesses: 38 time: 0:03:49:19 (3) c/s: 608672 trying: dixs7d - dix1tt
guesses: 38 time: 0:03:51:39 (3) c/s: 612460 trying: slibdiv8 - slibdoie
guesses: 38 time: 0:04:06:26 (3) c/s: 634679 trying: gnskugh - gnsrtls
guesses: 38 time: 0:04:30:29 (3) c/s: 664428 trying: gizea2s - gizeekm
guesses: 38 time: 0:04:33:51 (3) c/s: 668351 trying: f186766 - f1867nt
abcdefgh (crack23)
guesses: 39 time: 0:15:43:05 (3) c/s: 294389 trying: hEsinO - hEsit3

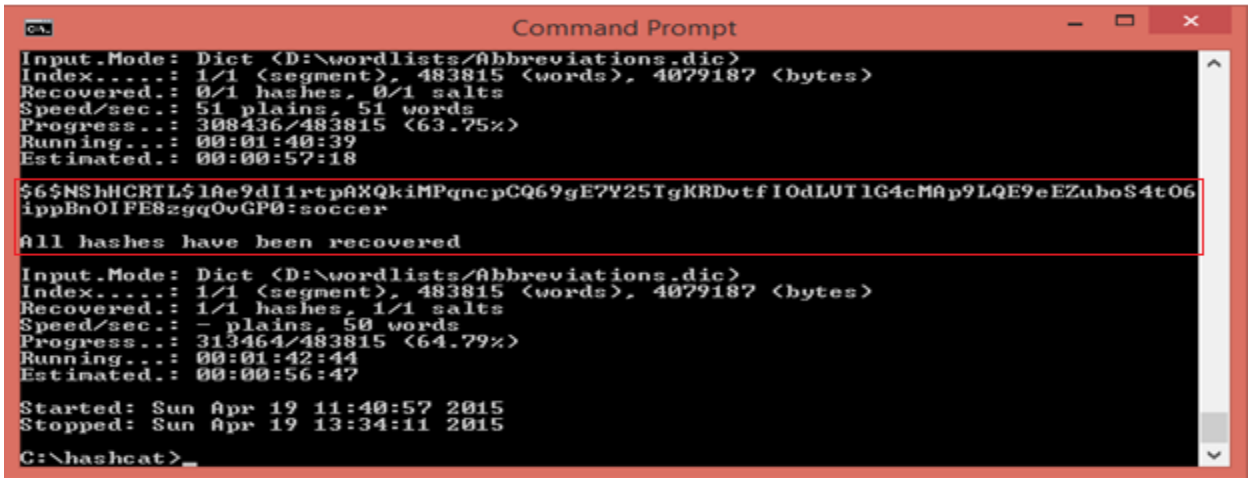

Command Prompt - john crack-these-please
guesses: 38 time: 0:03:27:28 (3) c/s: 572168 trying: stdard? - stdarol
guesses: 38 time: 0:03:30:46 (3) c/s: 577345 trying: $Asra* - $Asrp1
guesses: 38 time: 0:03:40:58 (3) c/s: 594742 trying: liamilk - liamysy
guesses: 38 time: 0:03:45:44 (3) c/s: 602697 trying: 17#aksdf - 17#akth3
guesses: 38 time: 0:03:49:19 (3) c/s: 608672 trying: dixs7d - dix1tt
guesses: 38 time: 0:03:51:39 (3) c/s: 612460 trying: slibdiv8 - slibdoie
guesses: 38 time: 0:04:06:26 (3) c/s: 634679 trying: gnskugh - gnsrtls
guesses: 38 time: 0:04:30:29 (3) c/s: 664428 trying: gizea2s - gizeekm
guesses: 38 time: 0:04:33:51 (3) c/s: 668351 trying: f186766 - f1867nt
abcdefgh (crack23)
guesses: 39 time: 0:15:43:05 (3) c/s: 294389 trying: hEsinO - hEsit3
guesses: 39 time: 0:16:19:38 (3) c/s: 311128 trying: RjnB0o - RjnBcq
guesses: 39 time: 0:16:48:11 (3) c/s: 329557 trying: mettatL - mettfek
guesses: 39 time: 0:17:00:36 (3) c/s: 335423 trying: amodfrif - amodrivE
guesses: 39 time: 0:17:46:29 (3) c/s: 323384 trying: asiayna - asiaysh
guesses: 39 time: 0:17:58:12 (3) c/s: 329934 trying: 18ki84be - 18kiYOK9
guesses: 39 time: 0:18:08:23 (3) c/s: 335669 trying: ccoxjuft - ccoxjlde
guesses: 39 time: 0:18:08:24 (3) c/s: 335680 trying: ccrgog7? - ccrgoi15
guesses: 39 time: 0:18:08:26 (3) c/s: 335680 trying: cclpitro - cclpicto
guesses: 39 time: 0:18:15:50 (3) c/s: 339742 trying: 05tepey - 05tepix
guesses: 39 time: 0:18:15:51 (3) c/s: 339753 trying: 0b0n1g0 - 0b0n138
guesses: 39 time: 0:18:15:52 (3) c/s: 339763 trying: 04ttu24 - 04ttudy
guesses: 39 time: 0:18:15:53 (3) c/s: 339774 trying: 04tn11a - 04tnna5
guesses: 39 time: 0:18:15:54 (3) c/s: 339785 trying: 09t1tm2 - 09t156y
guesses: 39 time: 0:18:15:56 (3) c/s: 339790 trying: 0aatc03 - 0aatov3
guesses: 39 time: 0:18:24:37 (3) c/s: 344483 trying: fd17e1 - fdhLOV
guesses: 39 time: 0:18:51:12 (3) c/s: 359467 trying: 33139ay5 - 33137$sd
guesses: 39 time: 0:18:53:44 (3) c/s: 360929 trying: 51572tin - 51572t27
09112001 (crack37)
Clinton (crack13)
guesses: 41 time: 0:21:06:28 (3) c/s: 416558 trying: J0dzd1 - J0dzsu

==========

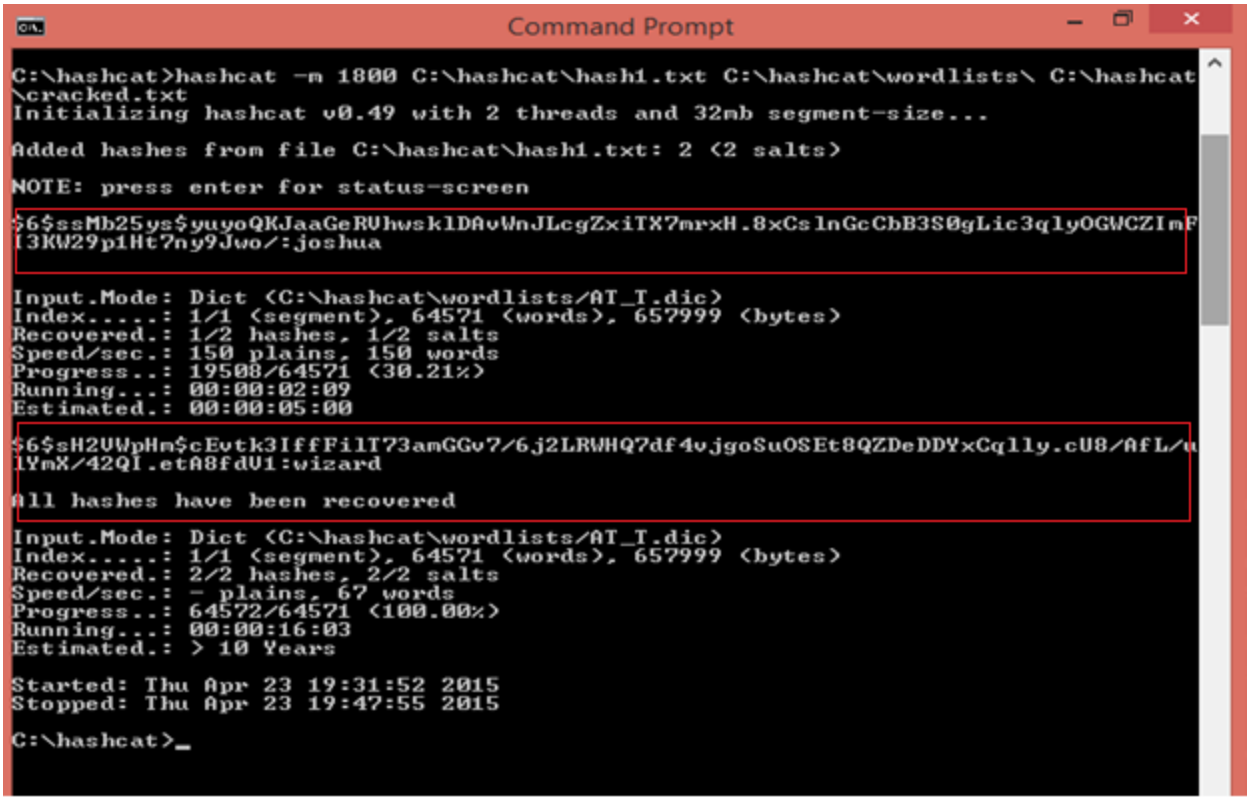
Ans. 3-B:

The previous file was an old password file as it used TripleDES for password hashing, a broken scheme that does not provide good security.These hashes are created with sha512crypt, which is a password-based hashing tool, where 6 is the type of passwrod hash (SHA512 many rounds), the next string between $ $ is the salt, and the remaining string is the hash. Use John The Ripper (or Hashcat) to break these four passwords.

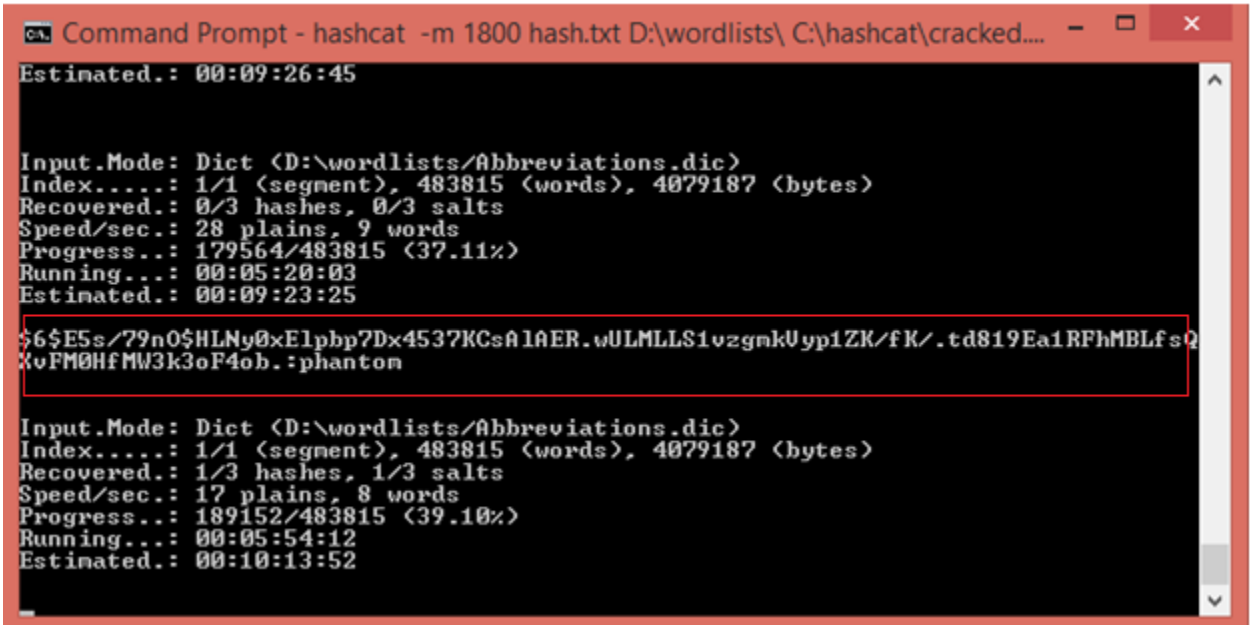
- 6NShHCRTL$lAe9dI1rtpAXQkiMPqncpCQ69gE7Y25TgKRDvtfIOdLVTlG4cMAp9LQE9eEZuboS4tO6ippBnOIFE8zgqOvGP0-->**soccer**



- 6ssMb25ys$yuyoQKJaaGeRVhwsklDAvWnJLcgZxiTX7mrxH.8xCslnGcCbB3S0gLic3qlyOGWCZImFI3KW29p1Ht7ny9Jwo/-->**joshua**
- 6sH2VWpHm$cEvtk3IffFilT73amGGv7/6j2LRWHQ7df4vjgoSuOSEt8QZDeDDYxCqlly.cU8/AfL/ulYmX/42QI.etA8fdV1-->**wizard**



- 6E5s/79nO$HLNy0xElpbp7Dx4537KCsAlAER.wULMLLS1vzgmkVyp1ZK/fK/.td819Ea1RFhMBLfsQXvFM0HfMW3k3oF4ob-->**phantom**



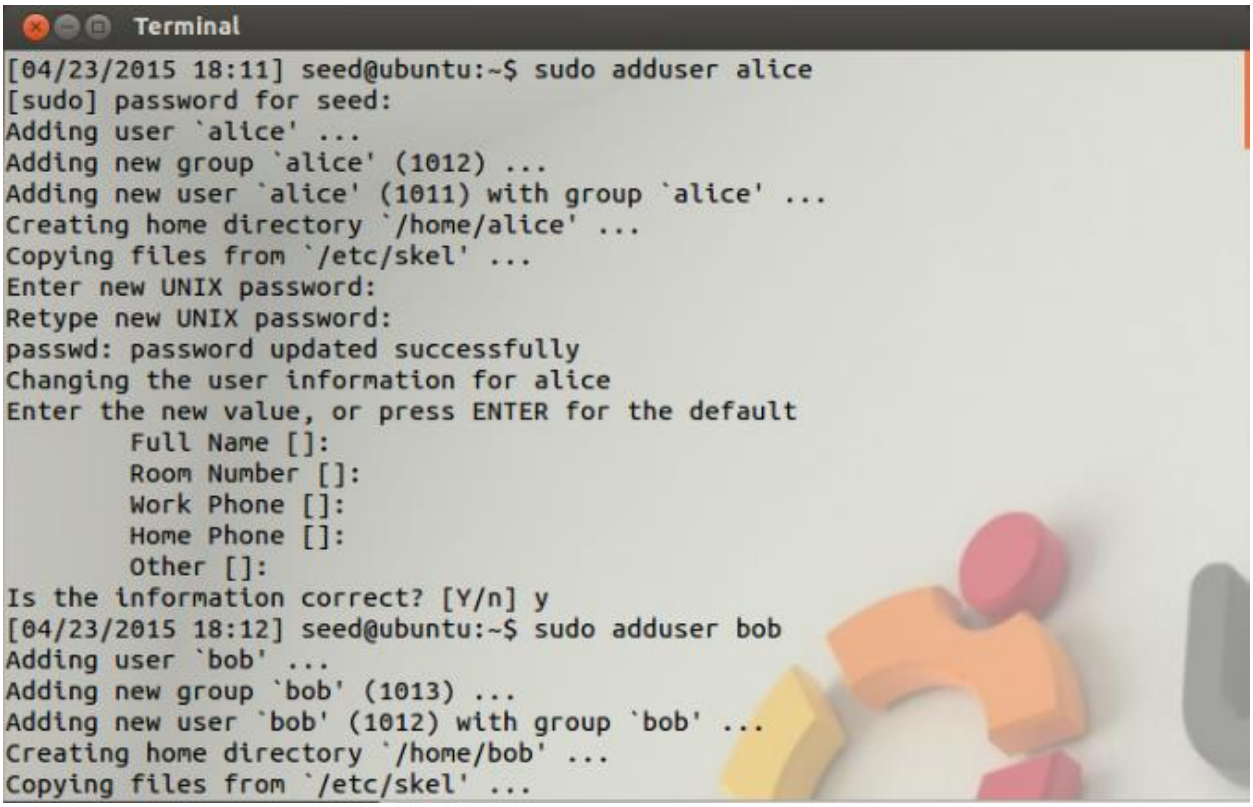
==========

Ans. 3-C:

Repeating these instructions, add 10 more users with 10 different passwords of your choosing and copy the last 10 lines of the shadow file to crack1.hash. You can use dictionary words, only letters, only symbols, four-character passwords, 10-character passwords----you don't need to crack them all, just leave them running overnight as a maximum---and summarize your impressions on password cracking.

Created 10 users as listed below with their respective passwords:

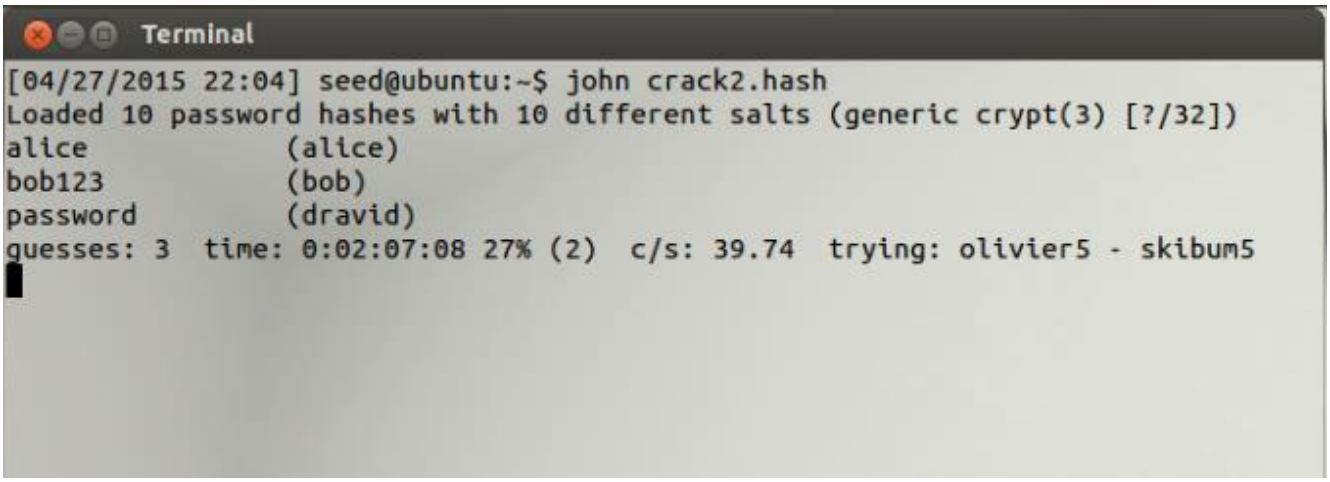
Username	Password
alice	alice
bob	bob123
dravid	password
mario	Super1390
king	kruskal
reena	$$reena12
tony	$bat#1#
rajesh	!123
kapil	20dev&&
sony	@@sony.!



After copying the last 10 line of the /etc/shadow file into crack2.hash, started john for password cracking.

Left it running overnight and in this span of time the only pass words it could crack were as shown in the screenshot below

Username	**Password**
alice	alice
bob	bob123
dravid	password



So the observation we have made is few passwords couldn't be decoded may be because they are protected by a strong password like a password with a combination of letters(Uppercase & Lowercase), digits(0,1…9) also with special characters. Such kind of passwords become very difficult to decode even using tools like john.

And on the other hand we could decode few passwords since these passwords are easy to decrypt as we can see it is very easy for john to guess these passwords as it is a commonly occurring word in all common dictionaries and can be easily guessed by dictionary based attacks. And it is not strong enough to resist hacking. This password would have been stronger if it had a mix of digits, letters and special characters which could have made it very difficult for john to guess the password.