Lab: Splunk and Log Analysis

Using Splunk at **http:// asecuritysite.com:8000** determine the following. You will be allocated a login.

Demo: https://youtu.be/Q7J_Fg_4zVI

Requirement	Answer
What is the start date of the log?	
How many log events are in d:\\buttercup\\mailsv\\secure.log:	
How many log events are in d:\\buttercup\\www1\\access.log:	
How many log events are in d:\\buttercup\\www1\\secure.log:	
How many log events are in d:\\buttercup\\www2\\access.log:	
What is the first username in the security log that gave an incorrect password (Hint: failed password | reverse)	
What is the first IP address in the security log that gave an incorrect password (Hint: failed password | reverse)	
Refer to the Splunk analysis. How many accesses were accessed by a "Chrome" browser and a "GET" method request (Hint - "chrome" AND method=GET)	
How many accesses were accessed by a "Chrome" browser or a "GET" method request (Hint - "chrome" OR method=GET)	
How many accesses were accessed by a "Chrome" browser and a "POST" method request (Hint - "chrome" OR method=POST)	
How many accesses were access by a "Chrome" browser or a "POST" method request (Hint - "chrome" OR method=POST)	
When was the peak accesses by a "Chrome" browser or a "POST" method request (Hint - "chrome" OR method=POST)	
How many accesses are there from a Safari browser (Hint: "safari"):	
How many accesses are there from a Chrome browser (Hint: "chrome"):	
How many accesses are there from a Mozilla browser (Hint: "mozilla"):	
On what day is there most activity in the secure logs (Hint: sourcetype=secure*):	
For the access.log from www1, which is the most popular HTTP response value (Hint - source="d:\\buttercup\\www1\\access.log"| top limit=5 status)	

For the access.log from www1, which is the second most popular HTTP response value (Hint = source="d:\\buttercup\\www1\\access.log"| top limit=5 status)	
For the access.log from www1, which is the most popular IP address for accesses (Hint - source="d:\\buttercup\\www1\\access.log"| top limit=5 clientip)	
For the access.log from www1, which is the second most popular IP address for accesses (Hint - source="d:\\buttercup\\www1\\access.log"| top limit=5 clientip)	
For the access.log from www1, which is the most popular action (Hint - source="d:\\buttercup\\www1\\access.log"| top limit=5 action):	
Refer to the Splunk analysis. For the access.log from www1, which is the second most popular action (source="d:\\buttercup\\www1\\access.log"| top limit=5 action)	
For the access.log from www1, estimate the number of iPad accesses (Hint - source="d:\\buttercup\\www1\\access.log" ipad)	
For the access.log from www1, what is the top refer domain (Hint - source=" d:\\buttercup\\www1\\access.log " | top limit=20 referer)	
Which is the first time for a refer from google.com (Hint - source="d:\\buttercup\\www1\\access.log" referer="http://www.google.com" | reverse)	
Which is the IP address of the client which is first referred from google.com (source="d:\\buttercup\\www1\\access.log" referer="http://www.google.com" | reverse)	
Which IP address connects successfully access signals.zip (Hint - signals.zip status=200)	
Refer to the Splunk analysis for secure*.log. How many failed password attempts were there from 194.8.74.23 (Hint - sourcetype=secure* 194.8.74.23 failed)	
Refer to the Splunk analysis for secure*.log. What day of the week had the most failed password attempts from 194.8.74.23 (Hint - sourcetype=secure* 194.8.74.23 failed)	
Refer to the Splunk analysis for access*.log. What day had the most successful purchases (Hint - action=purchase status=200)	
Refer to the Splunk analysis for access*.log. What day had the fewest purchases (Hint - action=purchase status=200)	
Refer to the Splunk analysis for access*.log. What day had the most purchases which were not successfully processed (Hint - action=purchase status!=200)	
Refer to the Splunk analysis for access*.log. How many STRATEGY games have been successfully purchased (Hint - categoryId=STRATEGY action=purchase status=200)	
Refer to the Splunk analysis for access*.log. Which file access always produces a 404 return message	
Refer to the Splunk analysis for access*.log. Which file access always produces a 404 return message: anna_nicole.html, productscreen.html, numa.html, cart.do or oldlink	
Refer to the Splunk analysis for access*.log. How many ARCADE games have been successfully purchased (Hint - categoryId=ARCADE action=purchase status=200)	

Refer to the Splunk analysis for access*.log. How many TEE games have been successfully purchased? (Hint - categoryId=TEE action=purchase status=200)	
Refer to the Splunk analysis for access*.log. How many SIMULATION games have been successfully purchased? (Hint - categoryId=TEE action=purchase status=200)	
Refer to the Splunk analysis for access*.log. How many SHOOTER games have been successfully purchased? (Hint - categoryId=SHOOTER action=purchase status=200)	
Refer to the Splunk analysis for secure*.log. What day of the week had the least failed password attempts from 194.8.74.23? (Hint - failed password 194.8.74.23)	
Refer to the Splunk analysis for access*.log. For an HTTP GET request, which is the most popular return code (Hint - sourcetype="access*" method="GET"| top limit=20 status)	
Refer to the Splunk analysis for access*.log. For an HTTP GET request, which is the 2nd most popular return code (Hint - sourcetype="access*" method="GET"| top limit=20 status)	

B Regular Expression Searches

We can use regular expressions to find information. For example, to find the number of accesses from an IP address which starts with "182.", we can use:

```
get | regex _raw="182\.\d{1,3}\.\d{1,3}\.\d{1,3}"
```

Determine the number of accesses for GET from any address which begins with 182:

The security team search for an address that is ending with .22, and do a search with:

```
get | regex _raw="\d{1,3}.\d{1,3}.\d{1,3}.22"
```

But it picks up logs which do not include addresses with .22 at the end. What is the problem with the request, and how would you modify the request:

You are told that there's accesses to a file which ends in "a.html". Using a regular expression, such as:

```
get | regex _raw="[a]+\.html"
```

outline three HTML files which end with the characters 'a', or an 'e', and have '.html' as an extension:

A simple domain name check is:

```
get | regex _raw="[a-zA-Z\.]+\.(com|net|uk)"
```

If we now try:

```
get | regex _raw="[a-zA-Z0-9\-\.]+\.(com|org|net|mil|edu|COM|ORG|NET|MIL|EDU|UK)"
```

We return additional domain. Outline which ones have been added:

We can search for times using regular expressions, such as:

```
get | regex _raw="[0-9]{2}\:22\:[0-9]{2}"
```

How many GET requests where there at 22 minutes past the hour:

How many GET requests were made at 14 seconds past the minute:

C Investigation

The incident response team wants a report of unusual HTTP requests. Produce a report of the HTTP response codes, and what they identify. Which ones could be malicious?

The security team would like a report on the most popular user names that have failed on the security logs:

The company are worried about the sales of some of their games. Which game category has the least amount of sales, and which is the best seller?

The Web design department have been told that there are missing files on the Web site. Investigate the files/pages that are missing on the site:

You have been asked to investigate accesses to the file named passwords.pdf in the Buttercup Games Splunk trace. Investigate any accesses related to it, and outline any possible significant evidence of malicious activity related to these accesses.

D Analysing Web logs

Within Ubuntu install Logstalgia from:

```
sudo apt-get install  logstalgia
```

You can then analyse one of your Apache log files. Go into:

```
/var/log/apache2
```

and then run:

```
/var/log/apache2$ logstalgia access.log
```

What do you observe from the visualisation?

Download the following file and analyse it in logstalgia:

http://asecuritysite.com/log/log01.zip

E Log parsing

In this example we will create a Python script to analyse the log file downloaded in the previous section. On Ubuntu (or Kali) create a Python file which will use a regular expression to parse an Apache Web log:

```
regex = '([(\d\.)]+) - - \[(.*?)\] "(.*?)" (\d+)'

import sys
import re

if (len(sys.argv)>1):
     file=sys.argv[1]

count200=0

for line in open(file):
     try:
          inp=re.match(regex, line).groups()
          status=inp[3]
          if (status=="200"): count200=count200+1
     except:
          continue
print count200
```

Run the Python program with:

```
python log.py log01.log
```

where log01.log is the name of your log file, and log.py is the name of your Python program.

Run the program, and observe the output:

Now modify the program so that it picks-off the other HTTP response codes that are contained in the file.

F Test

Now perform the following test:

http://asecuritysite.com/tests/tests?sortby=siem

Appendix A

Bad logins:

```
alert tcp any 21 -> any any (msg:"FTP Bad login"; content:"530 User "; nocase; flow:from_server,established; sid:491;
rev:5;)
```

Detecting email addresses:

```
alert tcp any any <> any 25 (pcre:"/[a-zA-Z0-9._%+-]+@[a-zA-Z0-9._%+-]/"; \
msg:"Email in message";sid:9000000;rev:1;)
```

Detect DNS:

```
alert udp any any -> any 53 (msg: "DNS"; sid:10000;)
```

File types:

```
alert tcp any any -> any any (content:"GIF89a"; msg:"GIF";sid:10000)
alert tcp any any -> any any (content:"%PDF"; msg:"PDF";sid:10001)
alert tcp any any -> any any (content:"|89 50 4E 47|"; msg:"PNG";sid:10002)
alert tcp any any -> any any (content:"|50 4B 03 04|"; msg:"ZIP";sid:10003)
```

Telnet login:

```
alert tcp any any <> any 23 (flags:S; msg:"Telnet Login";sid:9000005;rev:1;)
```

Port scan:

```
preprocessor sfportscan:\
        proto { all } \
        scan_type { all } \
        sense_level { high } \
        logfile { portscan.log }
```

DoS on Web server:

```
alert tcp any any -> any 80 (msg:"DOS flood denial of service attempt";flow:to_server; \
detection_filter:track by_dst,  count 60, seconds 60; \
sid:25101; rev:1;)
```

Stealth scans:

```
alert tcp any any -> any any (msg:"SYN FIN Scan"; flags: SF;sid:9000000;)
alert tcp any any -> any any (msg:"FIN Scan"; flags: F;sid:9000001;)
alert tcp any any -> any any (msg:"NULL Scan"; flags: 0;sid:9000002;)
alert tcp any any -> any any (msg:"XMAS Scan"; flags: FPU;sid:9000003;)
alert tcp any any -> any any (msg:"Full XMAS Scan"; flags: SRAFPU;sid:9000004;)
alert tcp any any -> any any (msg:"URG Scan"; flags: U;sid:9000005;)
alert tcp any any -> any any (msg:"URG FIN Scan"; flags: FU;sid:9000006;)
alert tcp any any -> any any (msg:"PUSH FIN Scan"; flags: FP;sid:9000007;)
alert tcp any any -> any any (msg:"URG PUSH Scan"; flags: PU;sid:9000008;)
alert tcp any any -> any any (flags: A; ack: 0; msg:"NMAP TCP ping!";sid:9000009;)
```

ping sweep:

```
alert icmp any any -> any any (msg:"ICMP Packet found";sid:9000000;)
alert icmp any any -> any any (itype: 0; msg: "ICMP Echo Reply";sid:9000001;)
alert icmp any any -> any any (itype: 3; msg: "ICMP Destination Unreachable";sid:9000002;)
alert icmp any any -> any any (itype: 4; msg: "ICMP Source Quench Message received";sid:9000003;)
alert icmp any any -> any any (itype: 5; msg: "ICMP Redirect message";sid:9000004;)
alert icmp any any -> any any (itype: 8; msg: "ICMP Echo Request";sid:9000005;)
alert icmp any any -> any any (itype: 11; msg: "ICMP Time Exceeded";sid:9000006;)
```

Note you may have to add the following for the stream analysis:

```
preprocessor stream5_global: track_tcp yes, \
   track_udp yes, \
   track_icmp no, \
   max_tcp 262144, \
   max_udp 131072, \
   max_active_responses 2, \
   min_response_seconds 5
preprocessor stream5_tcp: policy windows, detect_anomalies, require_3whs 180, \
   overlap_limit 10, small_segments 3 bytes 150, timeout 180, \
    ports client 21 22 23 25 42 53 70 79 109 110 111 113 119 135 136 137 139 143 \
        161 445 513 514 587 593 691 1433 1521 1741 2100 3306 6070 6665 6666 6667 6668 6669 \
        7000 8181 32770 32771 32772 32773 32774 32775 32776 32777 32778 32779, \
    ports both 80 81 82 83 84 85 86 87 88 89 90 110 311 383 443 465 563 591 593 631 636 901 989 992 993 994 995 1220 1414 1830 2301 2381 2809 \
       3037 3057 3128 3443 3702 4343 4848 5250 6080 6988 7907 7000 7001 7144 7145 7510 7802 7777 7779 \
       7801 7900 7901 7902 7903 7904 7905 7906 7908 7909 7910 7911 7912 7913 7914 7915 7916 \
        7917 7918 7919 7920 8000 8008 8014 8028 8080 8085 8088 8090 8118 8123 8180 8222 8243 8280 8300 8500 8800 8888 8899 9000 9060 9080 9090 \
        9091 9443 9999 10000 11371 34443 34444 41080 50000 50002 55555
preprocessor stream5_udp: timeout 180
```