Scripts.

I run many of these scripts every day. This is because some of these groups change memberships frequently. For the more intense scripts, I make sure I run them at night.

I. Active Inactive Accounts

*Get-ADUser -Filter {Enabled -eq $false} | FT samAccountName | export-csv C:\Data\InactiveAccounts.csv -NoTypeInformation*

*OR*

*Search-ADAccount -AccountDisabled | select samAccountName | export-csv C:\Data\InactiveAccounts.csv -NoTypeInformation*

II. Gather all Service Accounts

For this, it depends on how your organization organizes service accounts. Are they in a service account OU? Are they named in a consistent manner? If either one of these, that makes it easier to gather them into a list

*Get-aduser –Filter \* -searchbase “OU=service, DC=you” | select samAccountName | export-csv C:\Data\ServiceAccounts.csv -NoTypeInformation*

*OR*

*Get-aduser -filter {name LIKE “svc\*”} | select samAccountName | export-csv C:\Data\ServiceAccounts.csv -NoTypeInformation*

If your service accounts are spread around, it will take a lot of digging for you to ID all of them so you can monitor them. It also could make this script less useful.

III. Last Reboot Time & VI. Long Time Logged on Users

For this script, which I run at night as I've mentioned, I wrapped the commands into a python script so I can perform them against multiple machines and then send the results to my SIEM. You could adjust the script to drop to a CSV if you like. It sends the results in CEF format which despite being ArcSight, most SIEMs understand.

*import subprocess, os, sys, re, time, argparse, socket  
from datetime import datetime, date  
  
parser = argparse.ArgumentParser('Options to run')  
parser.add\_argument('-ip',dest='ip',action='store',default=None,help='IP of SmartConnector')  
parser.add\_argument('-v',dest='verb',action='store',default=0,help='Verbosity Level: 0 - None, 1 - Minimum, 2 - Maximum')  
parser.add\_argument('-o',dest='outlocation',action='store',default='s',help='Output destination: s - SIEM via SmartConnector, f - local file in XXXX, b - both')  
  
args = parser.parse\_args()  
  
if args.ip != None:  
  
    def syslog(message, level=5, facility=3, host='localhost', port=8514):  
  
            sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
            data = '<%d>%s' % (level + facility\*8, message)  
            sock.sendto(data, (host, port))  
            sock.close()  
  
#  
print "Getting time"  
endtime = time.strftime("%b %d %Y %H%M%S",time.localtime())  
  
date\_dict = {'01':'JAN', '02':'FEB', '03':'MAR','04':'APR','05':'MAY','06':'JUN','07':'JUL','08':'AUG','09':'SEP','10':'OCT','11':'NOV','12':'DEC'}  
  
print "Setting basic parameters"  
verbosity = int(args.verb)  
  
dest = args.ip  
Header = 'CEF:0'  
devicevendor = 'HomeGrown Script'  
deviceproduct = 'Logged On Users'  
deviceversion = '1.0'  
  
#machines = ['  ']  
#['']  
# [' ']  
print "calling powershell script to get list of PCs"  
subprocess.call('powershell -ExecutionPolicy Unrestricted C:\Python-projects\Loggedon-users\get-pcs.ps1')    
# comment out the above line when testing.  This prevents massive hits to AD.  
print "opening text file where list of PCs were stored"  
machines = open('C:\Python-projects\Loggedon-users\pcs.txt','r')  
print "starting loop to check if which PC is online"  
onmachines = []  
offmachines = []  
count=1  
for word in machines:  
    #if len(word) < 10: continue  
    if re.search(r'DNSHostName|#',word): continue  
    # if count > 35: break   # uncomment this line to test changes to the script.  Keeps the script from running for more than x number of machines  
    PC = word.lstrip('"').rstrip(',"\n')  
    #print PC  
    echo\_request = subprocess.Popen(["ping", "-n", "1",PC],stdout=subprocess.PIPE,stderr=subprocess.PIPE)  
    out, error = echo\_request.communicate()  
    out = out.strip()  
    error = error.strip()  
    # print out  
    # print error  
    if re.search(r'Ping request could not find host',out):   
        print PC + " could not be found" #continue  
        offmachines.append(PC)  
    elif re.search(r'Request timed out',out):   
        print PC + " could not be reached" #continue  
        offmachines.append(PC)  
    elif re.search(r'Reply from',out):   
        onmachines.append(PC)  
        print PC + " is alive"  
    count += 1  
  
for PC in offmachines:  
    CEF = Header +"|"+ devicevendor +"|"+ deviceproduct +"|"+ deviceversion +"|0|Update Logged on Users|0| end=" + endtime + " shost=" + PC + " alive=no"  
    if args.ip != None: syslog(CEF, host=dest)  
    elif args.ip == None: print CEF  
#  
  
#print onmachines  
print "starting loop to check if anyone is logged on to alive machines and who it is"  
results = {}  
  
  
for PC in onmachines:  
    print PC  
    doscommand = "wmic /node:\"" + PC + "\" OS Get LastBootUpTime"  
    result1 = subprocess.Popen(doscommand,stdout=subprocess.PIPE,stderr=subprocess.PIPE)  
    boot, error = result1.communicate()  
    if len(boot)>10:      
        boot = boot.strip('LastBootUpTime\n')  
        t1 = boot.strip()  
        #print t1  
        Month = date\_dict[t1[4:6]]  
        boot = Month + " " + t1[6:8] + " " + t1[0:4] + " " + t1[8:14]   #t1[4:6] + "-" + t1[6:8] + "-" + t1[0:4]  
    else: boot = "Could not get boot time"  
    result2 = subprocess.Popen(["C:\Python-projects\Loggedon-users\psloggedon", "-l", "\\\\" + PC],stdout=subprocess.PIPE,stderr=subprocess.PIPE)  
    out, error = result2.communicate()  
    out = out.strip()  
    error = error.strip()  
    #print "Here is the result: \n\n" + result  
    info = 1  
    if re.search(r'Error opening ', out): pass# print "Could not open the registry"  
    elif re.search(r'No one is logged on locally', out): pass# print "There is no one currently logged into this machine"  
    else:   
        info = 2  
        if not re.search(r'([0-9]{1,2}/[0-9]{1,2}/[0-9]{1,4})\s([[0-9]{1,2}:[0-9]{1,2}:[0-9]{1,2}\s[A,P]M)',out):  
            findthings = re.split(r'\S+(\\\w+.\w+)', out)  
            #print "Nothing in out, here is findthings" + str(findthings)  
            time\_diff = 9999  
            getuser = False  
            for thing in findthings:  
                #print thing  
                if re.search(r'Users logged on locally:',thing): getuser = True  
                if getuser == True:  
                    if re.search(r'\\',thing):  
                        username = thing.strip('\\')  
                        #print username  
        if re.search(r'([0-9]{1,2}/[0-9]{1,2}/[0-9]{1,4})\s([[0-9]{1,2}:[0-9]{1,2}:[0-9]{1,2}\s[A,P]M)',out):  
            findthings = re.split(r'([0-9]{1,2}/[0-9]{1,2}/[0-9]{1,4})\s([[0-9]{1,2}:[0-9]{1,2}:[0-9]{1,2}\s[A,P]M).+(\\\w+)', out)  
            logon\_date = datetime.strptime(findthings[1].strip(), '%m/%d/%Y')  
            now\_time = datetime.strptime(time.strftime('%m/%d/%Y', time.localtime()), '%m/%d/%Y')  
            #print logon\_date #findthings[1]  
            time\_diff = now\_time - logon\_date  
            username = findthings[3].strip('\\')  
    skip1 =0  
    if boot!= "Could not get boot time" and info==2:   
        print PC + " has last boot time of " + boot + " and " + username + " has been logged on for " + str(time\_diff)  
    elif boot== "Could not get boot time" and info ==2:   
        boot = "01-01-1969"  
        print PC + " has last boot time of " + boot + " and " + username + " has been logged on for " + str(time\_diff)  
    elif boot== "Could not get boot time" and info ==1:  
        boot = "01-01-1969"  
        time\_diff = 0  
        username = 'unknown'  
        print PC + " has last boot time of " + boot + " and no one is logged on or registry couldn't be opened."  
    elif boot!="Could not get boot time" and info ==1:  
        username = 'unknown'  
        time\_diff = 0  
        print PC + " has last boot time of " + boot + " and no one is logged on or registry couldn't be opened."  
    else:   
        skip1 = 1  
        print "Nothing worked, try again"  
    if skip1!=1:  
        CEF = Header +"|"+ devicevendor +"|"+ deviceproduct +"|"+ deviceversion +"|0|Update Logged on Users|0| end=" + endtime + " shost=" + PC + " cs1Label=Days Logged On cs1=" + str(time\_diff) + " suser=" + username + " deviceCustomDate1=" + boot + " deviceCustomDate1Label=Last Boot Time cs2Label=Alive cs2=yes"  
        if args.ip != None: syslog(CEF, host=dest)  
        elif args.ip == None: print CEF  
          
  
print "\nScript Complete"*

IV. Accounts never accessed / Accounts with never expiring passwords

Accounts never accessed:

*get-aduser -f {-not ( lastlogontimestamp -like "\*") -and (enabled -eq $true)} | export-csv C:\Data4Splunk\neverloggedon.csv -NoTypeInformation*

*Accounts with never expiring passwords:*

*Search-ADAccount –PasswordNeverExpires | export-csv C:\Data4Splunk\noexpirepassword.csv -NoTypeInformation*

V. File/Folder Monitoring

*get-aduser -filter { “all finance users” } | select samAccountName | export-csv C:\Data4Splunk\finance.csv -NoTypeInformation*

VI. (see above)

VII. Check to see if service is running

I rely on the SIEM to monitor for machines that have not been heard from, have the SIEM create a csv that this powershell script retrieves via SFTP. This powershell script processes that CSV, creates a new CSV of the results and uploads it via SFTP to a server (in this case a splunk server) for the SIEM to ingest.

*$file = "c:\Tools\Powershell\test\check\_machines.txt"  
$data = get-content $file  
  
$scpcommand = "c:\tools\pscp.exe -pw password root@server:/opt/splunk/var/run/splunk/unheardfrommachines.csv c:\Tools\Powershell\test\unheardfrommachines.csv"  
iex $scpcommand  
  
$hostfile = import-csv "c:\Tools\Powershell\test\unheardfrommachines.csv"   
  
# write-host $hostfile  
  
$hosts = $hostfile | foreach {$\_.Log\_Sources, $\_.Last\_Report\_Time}  
  
# write-host $hosts  
# write-host $hosts[1]  
  
# foreach ($line in $hosts)  
# {  
# if ($line -like "\*:\*") {write-host "This is a time " $line}  
# }  
  
# $length = $hosts.length  
  
# write-host $length  
  
$FileName = (Get-Date).tostring("MM-dd-yyyy-hhmmss")   
New-Item -itemType File -Path c:\Tools\Powershell\test -Name ($FileName + "-wls.txt")  
$outfile = $FileName + "-wls.txt"  
$filetosend = "c:\\Tools\\Powershell\\test\\" + $outfile  
  
#write-host $hosts | select Mode  
  
"Last\_Report\_Time,Log\_Sources,Status" | add-content $outfile  
  
ForEach ($line in $hosts)  
  
{  
  
# write-host $line  
if ($line)  
  
{  
# write-host $line  
if ($line -like "\*:\*")   
{  
# $get\_info = $line.split('\s\s')  
#write-host $line  
# write-host $get\_info  
# write-host $get\_info[2]  
#write-host $line " " $result1  
$line + "," + $result1 | add-content $outfile  
}  
else  
{  
$ruthere = test-connection $line -count 1 -erroraction 'silentlycontinue'  
if ($ruthere)  
{  
#Write-Host $line, " is alive on the network"  
$iswlsrunning = get-service -name wls -computername $line -erroraction 'silentlycontinue' | foreach {$\_.Status} #| select Status -hidetableheaders  
if ($iswlsrunning)  
{  
#write-host $iswlsrunning " is the status of WLS on " $line  
#"The status of WLS on " + $line, " is " + $iswlsrunning | add-content $outfile  
#$line + ",WLS is " + $iswlsrunning | add-content $outfile  
$result1 = $line + ",WLS is " + $iswlsrunning  
}  
else  
{  
#"WLS does not seem to be installed on " + $line | add-content $outfile  
#$line + ",WLS not installed" | add-content $outfile  
$result1 = $line + ",WLS not installed"  
}  
}  
else  
{  
#$line + ",Machine is offline" | add-content $outfile  
$result1 = $line + ",Machine is offline"  
}  
}  
}  
}  
  
$uploadresults = "c:\tools\pscp.exe password " + $filetosend + " root@server:/opt/splunk/etc/apps/search/lookups/resultsofwlscheck.csv"  
  
write-host $uploadresults  
  
iex $uploadresults*