**Scripting Project 3 – Due 5/4 12:00 PM CST**

You may only use the code that I’ve covered in class to date and your code structure must match mine. The sample output video is the definitive word on what to do. Your script must look and run like mine. Remember that your script must run to be graded, so it’s better to fully finish a function rather than halfway complete them all. If your script doesn’t run because you don’t have a variable path to whatever Windows desktop we use, and we have to fix it, you will be docked 25% of your total points. You have done everything required for this project in previous tutorials and homework exercises. Late submissions will not be graded.

The file alert\_full\_short.zip will be on the grading desktop prior to executing your script. The goal of your script will be to parse that file into a more usable file and provide some reporting on that data.

The script will unpack the zip file on the desktop and move the contents to the student directory Desktop\last name\first name. Mine would be Desktop\Burkman\Jim.

Start with Project 3 Starter.ps1 that has the array of major descriptors. Be sure to put your code in this order:

Array of major descriptors

Menu choice 1 function

Menu choice 2 function

Menu choice 3 function

Menu choice 4 function

Main code body:

Name strings

Active desktop path

Student directory

Data file check

Menu

1. Parse alert data

2. Major descriptors

3. Classifications

4. Clean up and exit

**Name strings**: make a static variable MY\_NAME and set it to your first and last name (e.g. ‘Jim Burkman’). Split that variable to make a variable for your last name and for your first name. Use these variables to make whatever paths you need. Do not hardcode your name anywhere else in the script.

**Active desktop path**: move the working directory to the desktop of the current user. Consider testing your script on the MSIS Lab image or another Windows computer before submission.

**Student directory**: actively check for the last name directory on the desktop. If it doesn’t exist then make the student path.

**Data file check**: actively check to see if alert\_full\_short.zip is on the desktop. If it is not, tell the user to put the file there and to restart the script. Exit the script. If the file is there, unzip the contents to your student folder. Use -Force. Specify the -DestinationPath as the user desktop so only the file gets unpacked, rather than a folder with the file inside.

**Menu:** Use a multiline string for the menu then IF/ELSEIF/ELSE for the options. Catch invalid options and have the user try again.

**Parse alert data (Menu Option 1)**: You’ve done this before so I’ll offer much less guidance this time around. Write the header, which consists of Date, Time, Priority, Classification, Description, Packet Type, Source IP, Source Port, Destination IP, and Destination Port. Write the corresponding data. See the demo video for details on how the data must look.

For packet type, use a flag when you get the line with the date, time, ip information. Set the flag to 1. Then on your next elseif if the flag = 1 grab the packet type and set the flag back to 0. This will avoid the packet types on the datagram dumps.

**Major descriptors (Menu Option 2):** The goal of this function is to allow the user to enter one or more characters of a major descriptor to show some statistics about alerts related to that major descriptor. The major descriptor is part of the capitalized start of the description. We will match user input with the tuple of major descriptors. If multiple descriptors are returned then we will ask the user to try again. If no major descriptors are returned we will inform the user then have them try again. If only one major descriptor is returned we will read the clean data file and collect all the full descriptions that match into a list. This list must only have unique values. We will tell the user the number of matches and wait for input to show those matches. Sort the result set before showing results. The user may type exit or EXIT to return to the main menu. See the demo video for output and formatting. Your results must match mine exactly, including the variable “------“ under the descriptor name when outputting values.

**Classifications:** The purpose of this function is to create a unique list of classifications and keep track of how many lines of data match that classification and what packet type was used. We will keep a running total of those packet instances. So your array is an array of arrays, where the first element of each array is the classification and the next element is initially three digits 0,0,0. Therefore if the array was named $dog, the first classification would be $dog[0]. That would look some like “words 000”. $dog[0][0] would look like “words”. $dog[0][0][0] would get the first digit, $dog[0][0][1] would get the second digit, and $dog[0][0][2] would get the third digit. [][][1] is the total of TCP packets for that classification, [][] [2] UDP packets, [][] [3] ICMP packets.

You can use [math]::Round($dog,2). Tabs are `t. See the demo video for the output. Yours must match exactly.

**Clean up and exit:** This function zips up the contents of the student folder into a zip file on the desktop named lastname\_firstname.zip (e.g. Burkman\_Jim.zip) then removes the student directory and all its contents. Archive all the contents of a folder like $some\_path + "\\*". You can set the destination path using a variable like $some\_path + \parrot" to get a zip file named parrot on that path.

The guidance here is made in good faith by reviewing my solutions. Errors are likely. All questions about the veracity of these instructions must be posted to the appropriate discussion forum. My answers will only be posted in that same forum. Use the Python Project 1 solution as a template for flow. It actually translates almost line for line to PowerShell.