10/12/2018

High Level Documentation of Java Discovery Project

Matthew Tang
SUN LIFE FINANCIAL

Table of Contents

1. Linux, AIX, & Solaris Operating Systems

1.1 Script 1 – Find All Running Java Processes on a Server

1. The first step is to declare all the data structure (array) and read the file contents (keywords.txt) into an array.
2. Force remove temp files (temp.csv & pid.csv)
    - We force remove these temp files just in case the script gets interrupted and does not finish
3. Create temp files (temp.csv & pid.csv)
4. We call the function Get-Process, find the server name, and current date
5. Loop through each keywords
    - Use either command *pgrep* or *ps* and then filter output using *grep* for our keywords

      ***The above commands will give you all the processes that match your keyword (in the form of PIDs aka process IDs). There will be a list of PIDs for each keyword searched, ex: searching for "java" may return 10 PIDs (which are processes) and then a search for "jre" may return 5 PIDs. Note that some of these PIDs will be the same from keyword to keyword

    - Check if PID is found in pid.txt using *awk* command
        - If yes -> skip current PID and move onto next PID
        - If no -> Find all properties of the process

      (Server, Process Name, User that initiated the process, Path of running process, Set counter = 1, Find version if available, Command Line, Set First Discovered = Current date, Set Last Discovered = Current date)

        - Check if the process name, user, path, or command line is empty
            - If yes -> Add the below information to ERROR.txt and move to next PID

              (Server, Process Name, User, Path, Command Line, Current date)

            - If no -> Check if the path is an executable (should be an executable for almost 100% of them since they are all processes) and use the *version* command if it's an executable file
            - Check if there are any errors that happen when *version* command is used
                - If yes -> Set version = NONE
                - If no -> Continue
            - Combine all the fields to and store in a line variable

        - Check if the current process' command line is found in temp.csv using *awk* command
            - If yes -> Go to that row in the temp CSV file and increment counter by 1
            - If no -> Add line to temp.csv (this becomes a new row)
        - Add the process' PID to pid.csv to indicate that we already scanned this process
        - Keep going until all the PID's have been scanned for every keyword
6. Store all the lines found in temp.csv into the array processArray
7. Check if report exists
    - If No -> Create file, fill in the headers, and paste elements of processArray into report CSV
    - If Yes -> Loop through each element in processArray
        - Get the command line (last element in the comma separated list)
        - Check if command line exist in report CSV
            - If No -> Add the line of information into CSV (becomes new row)
            - If Yes -> Get the cumulative counter of the scanned process and add that to the counter found in the CSV report, Update the Last Discovered header to the current date

1.2 Script 2 – Find All Installed Java Applications on a Server

1. The first step is to declare all the data structure (array) and read the file contents (keywords.txt) into an array.
2. Force remove temp files (temp.txt)
    - We force remove these temp files just in case the script gets interrupted and does not finish
3. Find the current server and date
4. Call the function Get-Application which will do the following steps below:
5. Depending on the UNIX related system, there are two possible commands that will search the file system: *find* and *locate* (depending the OS, find or locate may not be available. However, at least one of the two will be present) Use one of the commands to search the file system for the keywords, store the output into temp.txt, and then extract its contents into an array called pathArray

    *Small note: if locate command is used, the script needs to run the following command: *updatedb* before *locate* command is used (locate works by searching through a database and *updated* updates the file system data base to the newest version)

6. Loop through each path
    - Check if path is a file
        - If yes -> Check if the last directory in the path contains java and does not contain a period (This is done to remove unnecessary files ex: /…/java.html)
            - If yes -> Find unique path properties required for the CSV and store in temp.txt
            - If no -> Move onto next path
        - If no -> Trim the path based on trimming principle (refer to section 5.2 in the Java Discovery Documentation for how it works). After trimming, only find the unique path properties and store if it is not found in the temp.txt. The duplicate check for data files uses the *awk* command

    *** Temp.txt format: Each line represents a comma separated list and an example is given below:
    Server,Path,Type,Version,LastAccessDate,FirstDiscovered,LastDiscovered
    Line: sv66548,/etc/pki/java,FOLDER,NONE, 2018-10-04=17:03:59, 11-23-2018=11:29:09, 11-23-2018=11:29:09

7. Each line of temp.txt is read and stored as an element in the array: applicationArray
8. Check if report exists
    - If No -> Create file, fill in the headers, and paste elements of applicationArray into CSV
    - If Yes -> Loop through each element in applicationArray
        - Get the path (The second element of each comma separated list)
        - Check if path exist in report using the *awk* command
            - If No -> Add the line of information into CSV (becomes new row)
            - If Yes -> Update the Last Discovered header to the current date

2. Microsoft Windows

2.1 Script 1 – Running Process Changes

On the Windows Operating system the same logic applies, however there are some minor changes and they will be listed below.

2.1.1 Command Differences

The commands for finding running processes are different:

- Linux, AIX, Solaris: either use *pgrep* or *ps* and then *grep* to filter
- Windows: *Get-WmiObject Win32_Process | Where-Object {$_.CommandLine –match <keyword> }*

There is a very clear difference between using these commands. For UNIX related systems, *pgrep* or *ps* then *grep* will look at all the properties of a running process (ex: user name, command line, etc.) to provide a match for the keyword. As long as one of the properties contains any of the keywords, the process will be "found". For Windows, a process can only be found if its command line contains a keyword. In other words, there is a slight possibility that in a Windows environment a java process can be missed if its command line does not contain one of the keywords.

2.1.2 Use of Objects

Since everything is an object in PowerShell, the properties of a process are stored in an object. For AIX, Solaris, and Linux, they are just individual variables that are overwritten for every process.

2.1.3 File Parsing & File Modifications

Linux, AIX, and Solaris have a command *awk* that can easily parse through data files and make modifications as necessary. It can search, match, and update headers via one command. For Windows, it is a little more complicated. It requires some more steps such as importing the CSV contents, looping through the contents and then seeing if a match is found. If found, modifications are made, if it isn't found and all the contents are looped through then the process is added. There are comments in the script that will explain the details of what is happening if further clarification is required. It is not too big of a deal, it is just different.

2.2 Script 2 – Installed Application Changes

2.2.1 Hard Drives & Network Drives

For Windows, there are often many drives presents such as C drive, E, F, and etc. There are also network drives found. Instead of searching from root '/' like in UNIX environments, on the Windows script it is necessary to identify all the drives first. Once that is done, the search loops through each available drive that is **NOT a network drive**.

2.2.2 Additional changes

2.1.2 and 2.1.3 both apply to Script #2 also. There are minor syntax differences between the different operating systems as well. The concepts explained above are the most important differences from one server type to another.

3. File Output

3.1 File Format – Linux, AIX, Solaris

For Script #1, the preferred file output format should be a CSV. However, there are some problems with having this specific file format. Sometimes in the command line there are commas which normally isn't a problem, but, when a user decides to open the file, Excel is not smart enough to distinguish between the commas used as a delimiter and leaving the commas alone in the command line. Instead, what actually happens is that Excel will split the command line up into different columns because it treats the commas in the command line as the delimiter instead of a normal comma. If a user opens this file, the columns will be mismatched and the formatting will be wrong. There is a solution to this, but it requires a couple of extra steps.

Instead of using a CSV format for running processes, a text file is used instead. The delimiter is set to a semicolon ";" instead of the normal comma ",". This is done to distinguish between the commas used normally to separate values and the commas found in the command line. **Only for script #1 for AIX, Linux, and Solaris** is the output a text file. The rest of them will output a CSV file. The steps below detail how to convert a text file with a semicolon delimiter to a CSV report.

1) Open a new workbook
2) Click *Data* in the top bar
3) Click *From Text*
4) Browse to the text file
5) Click *Delimited* & *My Data Has Headers*
6) Press next and then change delimiter to semicolon
7) Press finish and data should be organized