Global Information Assurance Certification Paper

Putting it all together through Automation

GIAC (GCFE) Gold Certification

Author: Kenneth Ray, Kenneth.Ray@MKCorp.com

Advisor: Rob VandenBrink

Accepted: April 17, 2019

Abstract

Most problems faced in Information Security are typically time sensitive. For Forensic Engineers and Analysts, it's not the problem related to when a Forensic Analyst has a local physical drive in hand; rather the problem is how quickly they can obtain forensic evidence to support or disprove data exfiltration, exploitation, or infection when the system is not locally accessible. Most times this requires remote collection and, in some cases, covert data collection. This paper will explain methods to automate collection using scripts and functions formatted mostly in PowerShell to accomplish goals. This paper will include the heavily commented version of the Yet Another Forensic Tool(YAFORTO). Where possible, prerequisites will be identified to alleviate failures already discovered during the development and testing of the scripts and functions.

1. Introduction

With most investigations, the Forensic Analyst is attempting to ascertain whether a certain event has occurred. This could be linked to time, a specific user, or even both. The initial steps for discovery are rather simple and follow the three major associations with an event: the suspicion that an event occurred, forming a hypothesis of what has occurred, and proving or disproving that hypothesis with evidence. The intent of this paper is to give the reader a way to quickly do some forensic analysis through the use of tools and commands. A common desired action is to leverage the ability of tools through automation and scripting.

Having the expertise and knowledge to find the evidence is either: the cornerstone components of the Forensic Analyst or the cornerstone components of a forensic investigation. For a Forensic Analyst starting out in the field, it's harder to remember all the locations to look. This is evident especially if the Forensic Analyst's primary job is something other than forensics. In some cases, the analyst would have to reeducate themselves or use some other reference such as the SANS.org Windows Forensic poster by Rob Lee (Lee, 2018) to gain more insight or even perform an investigation.

As stated, most Forensic Analysts in areas other than law enforcement are more commonly engaged in other security-related activities in their day to day operations. They may only have performed a forensic investigation at such a limited frequency, they are unsure of their own findings or the tools that can be used. If there were at least a way to perform a quick look at a system without having to relearn the locations and specific details it would be beneficial. The ability to automate the process of gathering the preliminary information that would be needed to form a hypothesis would be an improvement. Yet another improvement would be the ability to gather evidence remotely and covertly. These needs are the main reason the program was designed that will be discussed in this paper.

A key component of automation is reliance and availability of tools to be used in automation. This fact is the reason specific code and tools were used within the script. A

worthwhile goal, one used within the script, are tools that support command line switches and can be integrated within the chosen scripting language. Using the Windows native scripting language of PowerShell was also a conscious choice over using Perl, Python, Ruby on Rails, or even C. The choice was due to its widespread installation natively to the largest set of forensic targets.

As the code is discussed, and code snippets will be reviewed in this paper, it will be noticed that the code is highly commented. This again was intentional. While familiar with other languages such as Perl, Python, and C, the developer is less familiar with the commands associated with PowerShell. Using heavy commented code also makes maintenance and understanding of the code a little easier for everyone involved. Understanding the basics of programming in almost any other language a reader should have no problem with following the program to its completion and can make adjustments that would fit into their environment.

Gathering a full disk image and a full dump of memory requires not only a lot of time but a lot of knowledge of tools. In the heat of an Information Security Incident, the time to grab a full image from a remote computer and a memory dump will often be prioritized by may not always be required. The attitude of 'better safe than sorry' applies to this scenario, but urgency also plays a large factor. Take, for instance, a scenario of data exfiltration. If the scenario is a matter of showing evidence that data exfiltration occurred after the fact, this is less urgent than using forensic information to determine where a file is being exfiltrated in real time.

This paper and the accompanying program were designed to address the urgency assosicated with most Information Security Incidents. Using this tool and associated indicators will give an investigator a larger understanding of whether more time and resources are needed for the investigation.

YAFORTO was designed to be modular. As already stated it is highly documented to give even a person not knowledgeable in PowerShell an understanding of what is happening.

1.1. Perquisites

The script relies heavily on the use of the WinRM service and PowerShell commands: Invoke-Command, New-PsDrive, and Remove-PsDrive. These commands will fail if WinRM service access is denied to the investigator running the script. If winrm is off, it should be enabled before running YAFORTO. To do this remotely, by changing GPO settings or possibly on the computer directly ("WinRM QuickConfig, HowTo Enable via GPO or Remotely on All Servers" 2019).

As is always good practice, creating a login ID that has proper access to all target systems is a desirable prerequisite. That ID should also never be allowed to interactively login and be disabled at all times, except when an investigation using the elevated privileges are needed. It is common to have a login that is used for the sole purpose of Security Incident Response. That ID is monitored for activity to document its use and has privileges allowing for unfettered access to both network, infrastructure and endpoints.

Access to a form of administrative privileges for the system and a sizeable drive to record the data retrieved remotely is important for the success of YAFORTO. Running the script in an administrative session on the source computer is also important. Using an administrative session, or one configured with needed rights, on the source computer will allow certain conditions to be overcome.

1.2. Known limitations

While YAFORTO has been tested, it also has limited error checking capabilities in the current version of code. Where a catastrophic error preventing execution may occur, rudimentary error checking was put in place. Another consideration not captured in the current version is MD5 checksum validation of files. This function is planned to be added in later versions. Of consideration, too is the limitation related to the memory image that is captured by YAFORTO. The current version doesn't take advantage of any tools to analyze it.

2. Main Functions

The name of Yet Another Forensic Tool (YAFORTO) was chosen to show that there are several other scripts and tools available with similar functions as this one. Within the YAFORTO, there are three major functions.

The first is the collection function, using both a memory and a data collection program to gather data which can then be searched. Because most of the forensic evidence can be obtained using a specific subset of the entire disk, and acquisition of this data is far less time consuming than taking a full dump of the disk image, it was decided to use this method as the choice of retrieval. A triage image that includes the relevant files for the initial evaluation of a forensic hypothesis is done by a program called Cylr. This program was developed by Alan Orlikoski and distributed on his GitHub page (Orlikoski, 2018).

The second function is the analysis function. After data is collected and moved to the analysis computer, the analyst will want to run tools to reduce the inspection surface to only those related to the time or user the analyst cares about. The function that uses the RegRipper program (keydet89,2019) inside YAFORTO can be expanded past the initial version. YAFORTO can also be expanded to include options like the forensic memory tool, Volatility ("The Volatility Foundation - Open Source Memory Forensics", 2019).

The last function of YAFORTO, reporting, is currently still under development. PowerShell variables are assigned to the data, but exporting that data into a report has not yet been developed. Like the previous function of analysis, the reporting of the data is intended to be used in a timeline. Future versions will include a single report that takes all the information and exports it to a CSV format.

2.1. Information retrieval.

The first focus of forensics is reliable data retrieval. After all, an examination using data that can be questioned does not produce undisputable facts. Bad data can also affect the initial hypothesis the examiner may make. How that information is gathered and what information supports a hypothesis are the only true factors for an examiner. Like most of the functionality in the YAFORTO script, the acquisition using Cylr (Orlikoski, 2018), is contained in a function called "cylr_pull".

Before this function is used there are some preliminary details that must be obtained. The first information needed is the computer name being targeted. The next statement is the credentials needed to gain access. If the person running the script has submitted them on the command line the script checks using this assignment to the variables.

```
$Mycomputer_name = $args[0]
$MySecureCreds = $args[1]
```

Once this preliminary information is obtained, the Cylr_pull function, as well as other parts of the script, will use these variables to obtain or parse information.

2.1.1. Cylr_pull function

The main function of gathering information is to actually get the information in the fastest time without mistakes. While a full disk image is the best source of evidence, it is often not needed to initially investigate. Using an image that has the forensic details important for a first glance by the examiner is usually the best. As mentioned the first function that should be run is the function cylr_pull. The function creates a PSDrive, copies the cylr.exe to the drive, runs it, copies the zip file back to the local directory, and deletes both the exe and the zip file from the remote. Here is the code:

```
Function cylr_pull {
#########################################################
## cylr_pull function uses the cylr.exe file defined inside the function as
variable $myClyr_exe
```

```
## create a drive so we can copy the information, Note you must have the
ability to map to a Admin share.
#  uses the Mysecurecreds as the credentials
## variables:
#            uses Mycomputer_name previously defined
#            $myClyr_exe =location of Cylr executable
#            Uses $MySecurecreds as credentials
#############################################################
$myClyr_exe = "c:\temp\CyLR.exe"
New-PSDrive -Name cylr -PSProvider FileSystem -Root \\$Mycomputer_name\c$  -
Credential $MySecureCreds
#changes to the newly created psdrive. not really necessary but will generate
an error if it cant be done.
Write-host "checking drive by Switching to it"
cd cylr:
Write-host "copying $myClyr_exe to the remote c:\temp directory"
Copy-Item -path $myClyr_exe cylr:\temp
Write-host "Invoking the cylr.exe executable using the credentials provided"
Invoke-Command -ComputerName $Mycomputer_name -ScriptBlock {cmd.exe /C
"C:\temp\cylr.exe -od c:\temp" } -Credential $MySecureCreds
#notify the end user what happened.
Write-host "if successful the $Mycomputer_name.zip file will be in the temp
directory on the remote host`n we will move it to the local c:\temp"
Copy-Item cylr:\temp\$Mycomputer_name.zip C:\temp
Write-host "cleaning up the remote host by removing the data and the
executable"
Remove-Item cylr:\temp\$Mycomputer_name.zip
Remove-Item cylr:\Temp\cylr.exe
#change back to the local C drive. because we checked to see if we could CD to
the PSdrive we need to go back
# to C: so we can remove the drive.
C:
# now remove the Drive we created
Remove-PSDrive cylr
#-----------------------------------------------------------
# End of Cylar_pull function
#-----------------------------------------------------------

}
```

2.1.2. Detail of cylr_pull function

The function takes a number of steps to complete. The initial step is to connect to the remote computer using the PSDrive command from PowerShell. The drive created is called cylr. The process uses the existence of the administrative share c$. The script also assumes there is a directory off the root of C: called 'temp'. If this is an issue for the target environment, then it is suggested adding a statement inside the function that will check for the existence of the directory using the PowerShell native command of 'Child-Item'. This can be expanded to creating it using a variety of methods if the directory doesn't exist.

The next step in the function changes to the drive. The action is to validate the drive connection was set up. Again, this can be edited to add error checking. After changing to the newly created drive and changing to its alias, the function copies the cylr.exe file to the drive. The script executes the Cylr command with switches that will save a Zip file to the local C:\temp directory. This is executed using the cmd.exe /c command which will exit upon running the single command. One thing of note is that the out directory indicated by the switch 'od' must be an accessible location on the local machine.

After execution of the above steps, there will be a zip file that needs to be copied back to the investigating machine. The next step will remove both the zip file and the cylr.exe file and remove the PSdrive called 'cylr' so the next part of the script can be executed. The function will do this and delete the zip file locally. Of consideration is the fact the function is changing slack space and manipulating the disk of the remote computer. Running this script should be done with full documentation of time where this occurred. This record can be then used in case the slack space and other forensic fragments are affected.

2.1.3. MEMTriage function

The MEMTriage Function takes a full dump of memory from the remote system. Just like the cylr_pull function, the results of the execution are saved on the local remote system and then moved over to the local system that is running the script. For ease of use and because there have been issues during the testing of the script on different Windows operating systems, the main MEMTriage function has two memory dump command choices, Winpmem and Dumpit. To modularize the function into understandable code both choices are encapsulated into functions. One called 'runDumpit' which will run the dumpit executable and one called 'runWinpmem' which will run the winpmem executable. The person running the script will have the option to pick either.

2.1.4. runDumpit function

The person executing the script will be prompted to choose either 'w' for winpmem or 'd' for dumpit. The question is set into a loop where if they type anything other than these two choices they will be asked to choose again.

Both functions will create a named PSdrive similar to the application being run, either winpmem or dumpit. Like the other PSdrive commands in previous functions, this will be mapped to the administrative share of c$ and the directory of 'temp' must be present. Both create a zip file that will then be moved from the remote machine to the machine running the script.

```
Function runDumpit {
###############################################################################
# Dumpit section
###############################################################################
New-PSDrive -Name Dumpit -PSProvider FileSystem -Root \\$Mycomputer_name\c$  -
Credential $MySecureCreds
#changes to the newly created psdrive. not really necessary but will generate
an error if it cant be done.
Write-host "checking drive by Switching to it"
cd Dumpit:
Write-host "copying the Dumpit.exe from c:\temp to the remote c:\temp
directory."
Copy-Item -path C:\temp\Dumpit.exe dumpit:\temp
#note: that on some systems dumpit.exe will generate an error about not being
able to install. This is not a scripting
#error of this script.
#notify user what we are doing
Write-host "Invoking the dumpit.exe executable using the credentials provided.
This will take a while, be patient!"
#invoke the command on the remote system
Invoke-Command -ComputerName $Mycomputer_name -ScriptBlock {cmd.exe /C
"C:\temp\dumpit.exe /Q /O c:\temp\$Mycomputer_name.raw" } -Credential
$MySecureCreds

#-----------
#notify the end user what happened.
Write-host "If successful the .raw file will be in the temp directory on the
remote host`n we will move it to the local c:\temp`n Depending on the size this
also may take a while."
#dumpit creates a .raw with no name instead of the expected name because the
$MyComputer_name is not a local variable for the invoke command.
#No matter, we will copy it to c:\temp\ with the computer name.
Copy-Item Dumpit:\temp\.raw C:\temp\$Mycomputer_name.raw
#tell the user our actions
Write-host "cleaning up the remote host by removing the data and the
executable"
#remove the .raw file on the remote computer.
Remove-Item Dumpit:\temp\.raw
#remove the dumpit executable

Remove-Item Dumpit:\Temp\dumpit.exe
#change back to the local C drive. because we checked to see if we could CD to
the PSdrive we need to go back
```

```
# to C: so we can remove the drive.
C:
# now remove the Drive we created
Remove-PSDrive dumpit
#-----------------end of Dumpit section ----------------------------------#
}
```

2.1.5. Firewall information gathering.

The next information to gather will be the firewall rules. We are obtaining this information to show the state of the target when the information was gathered. This can be used in conjunction with other information found. Typically changing firewall rules would be done by a user to allow access. Obtaining Firewall rules can be done in a variety of ways, including using Powershell Commands. The chosen method for the script is using Netsh on the command line and passing the information to a hash. By creating a function, we can then assign the output to a variable to be reported upon. The original code for this function was retrieved from the original author, Stephane vanGulik on the site http://powershelldistrict.com/netsh-advfirewall-powershell/. The original was changed for use within the YAFORTO script. The original code was changed by adding verbose comments. Within the script the function is called FirewallRules and the call is captured in the variable $TRGFirewallRules. Because it doesn't have relevance in this version of the script other than being a placeholder for further development, it is suggested to reference the original code or the commented code on the YAFORTO GitHub Page (manta0101, 2018).

2.2. Information filtering

The second function of the YAFORTO script is to take the triage image that was created and copied locally and parse it with tools to reduce the amount of data that the Forensic analyst needs to spend reviewing. The Registry is the largest dataset of forensic information and just reducing the amount of noise to a specific timeframe will reduce the amount of time the analyst needs to Complete the assessment.

There are different methods by which to do this and the script has two, one of which is there as a placeholder. The method uses the reg load function and the PsDrive function to allow for traversal of the different keys. This method does not require a secondary tool to be installed or loaded. It does, however, require more knowledge of the registry and understanding of time stamps. Later in development, the script will be modified to offer an option to the Forensic Analyst.

The other method in the script which was developed further takes advantage of the RegRipper distribution (keydet89, 2019). The reason RegRipper was chosen is that it already has plugins that can filter the results of the registry into readable chunks and can also be added to the script without having to re-code the nuances of time and date stamps.

2.2.1. Time Zone Filtering

To reduce the amount of data, an interval of interest for the investigation needs to be defined. A start point and an endpoint need to be formally set. The initial interval of interest is supplied by the forensic examiner prior to the execution of the time zone filtering functions. The RegRipper plugins also display information in UTC which is convenient for our purposes. YARFORTO is meant to be executed remotely, so conversion may need to occur to produce the timeline we need to review. For this, YAFORTO uses two functions, timezonetarget, and time-convertUTC. The first function, timezonetarget uses a list of time zones for time conversion. The outcome is then fed into the time-convertUTC function as the time zone to convert into UTC.

YAFORTO uses a simple numbered heading scheme and information of the available time zones and use the C function called GetTimeZonebyID. This is the quickest way to convert. YAFORTO uses the list of time zones provided by ThomasThomas (ThomasThomas). This was incorporated in YAFORTO as a function that encloses the two functions mentioned above and the outcome is then assigned to the variable '$TimeZoneChoice'. TimeZoneChoice is then examined using a switch statement to apply the correct TimeZoneByID text to the variable ChosenTimeZone which is then passed to the TimeConvertUTC function that converts it to UTC. The

outcome of the Functions are assigned to two variables: minUTC and maxUTC, which are used throughout YARFORTO to filter results.

2.2.2. User filtering

YAFORTO reads the triage image "Users" directory to present available users. The Forensic Examiner is asked to pick the user being investigated to allow the RegRipper portion of the YAFORTO script to load the Ntuser.DAT file for the appropriate user. Later in development, this may become a simple question instead which allows the Forensic Examiner to choose whether to narrow the information to a single user or to cast a wider net of all users.

2.2.3. Registry filtering

YAFORTO uses the two variables minUTC and maxUTC from the time zone filtering with the the RegRipper application developed by keydet89 (keydet89,2019). The RegRipper plugin, regtime, has an output that requires conversion to be useful with the two variables. Conversion uses a search and replace method, parsing the date to be converted and then converting it into a filetime string. The filetime string is then compared to minUTC and maxUTC to reduce the entries to those of interest.

Just like many of the other functions that use external commands, the rip_reg function has a couple of variables internally to the function and through testing the formatting of the variables is required to allow the function to work. The first of these is the location of the rip.exe file, which is assigned to the $myRegripperEXE.

The rip.exe program is passed the variables that are built using the variable $myworkingDirectory combined with the known locations of the SAM, SYSTEM, And SOFTWARE registry hives. Each Hive is referenced by a variable with a similar name. This is then added to another array called $registryFiles in preparation for the execution of the rip.exe, RegRipper executable. The foreach loop assigns the variable $registryFile and then runs "&cmd /c '$myRegRipperlocation -r $registeryFile -p regtime'" and places that data into a variable called $out.

When this is completed we take the information in $out and assign it to $out2. The original used a different method regtime_tln to parse a different string. In the original code that parsing is still part of the line where the variable is assigned but commented out to not be executed against the $out Variable. The $out2 variable is then passed into a loop and iterated using $KeyString. It's parsed using a regex expression to assign the outcome to the array label $DataSplit.LastWriteTime. Then the actual key as written is assigned to the array label of $DataSplit.Key. Because of the parsing format of the regtime plugin, the $DataSplit.LastWriteTime has to be converted. This is done with a replacement switch and then passed to get-date and the C function .toFiletime() and reassigned to $test.

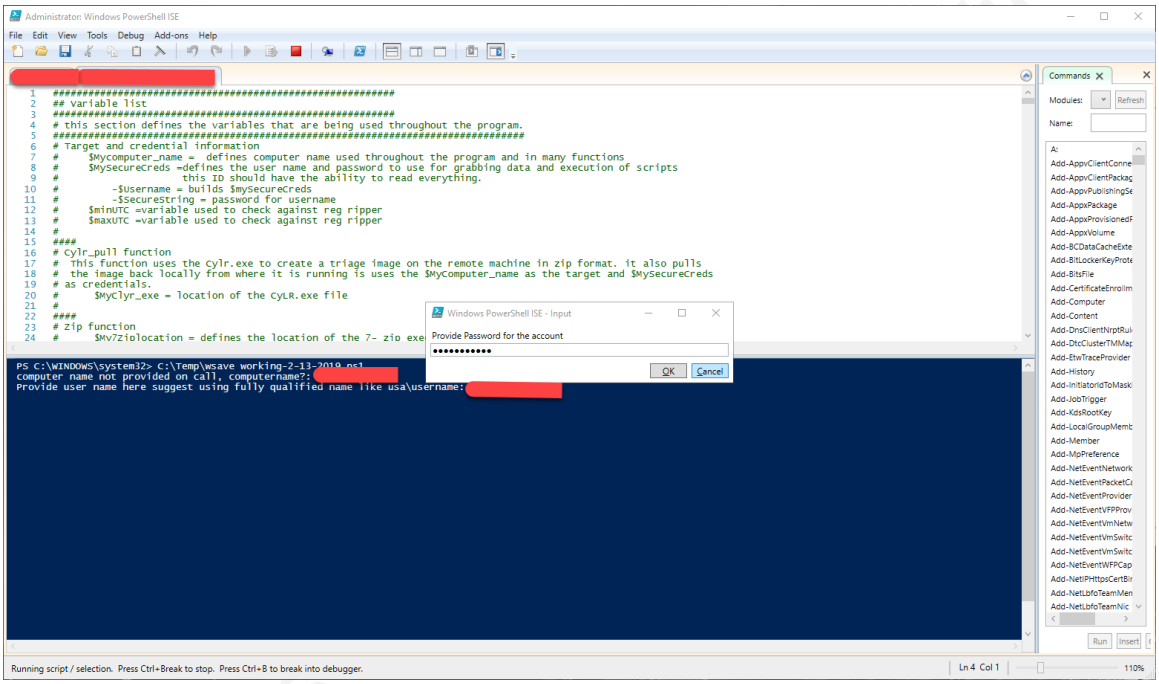
$test is checked against the $minUTC and $maxUTC variables to see if it falls between them. If it does, then we assign the array label $DataSplit.ActionTKN with the string Change, and replace any spaces in the $DataSplit.Key. The current version of the re_rip function only outputs the original entry, the action taken, last write time, and the original key to the screen. Planned in a future version is to place this information into another array that is able to be used in a report.

The rip_reg function can also be expanded on to parse any of the Plugins available from the RegRipper program by copying the entire foreach loop. The reason for leaving the $out2 variable was to accommodate the output that is different from each plugin without having to rewrite the code completely. It could be stated that the foreach loop is a function within the function. The current version is also being expanded to include those plugins that hold the most value for the Examiner to view in a report.

2.3. An example run of YAFORTO

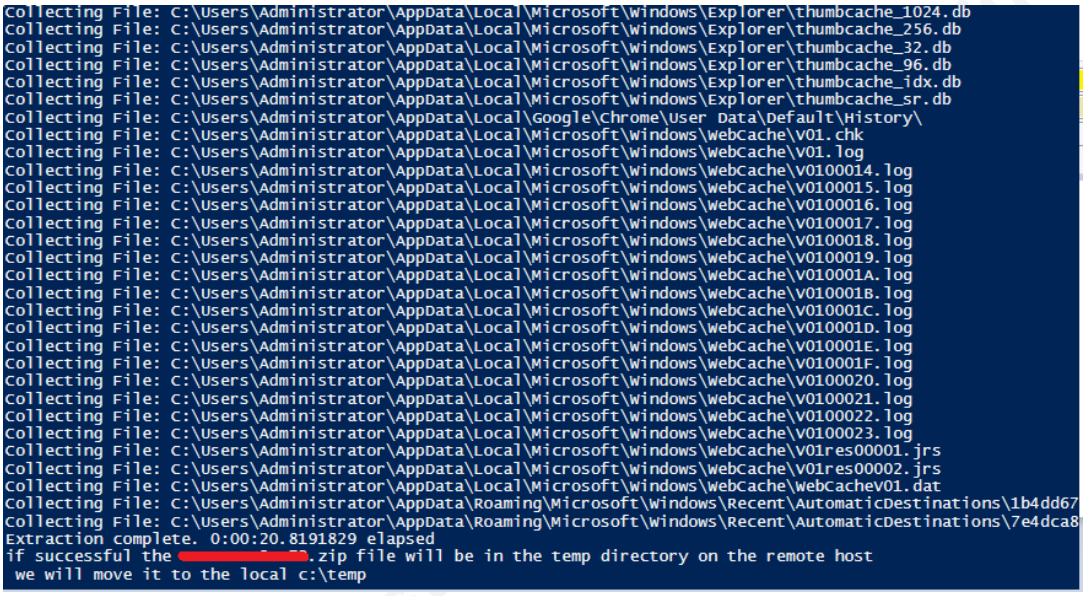
Here is an example run of YAFORTO to demonstrate what the examiner sees.

As the analyst enters the program, the computer name and the username to use to perform the investigation is requested.

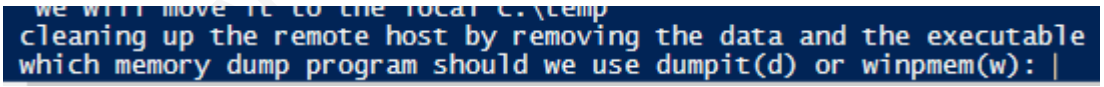


After supplying credentials and the target computer, the analyst is asked the timezone to be used with the two times the examiner will supply later.

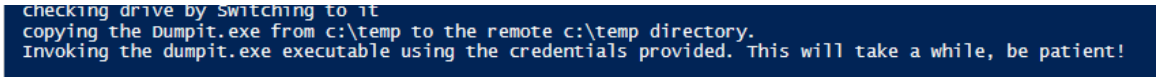
After supplying the time zone, YAFORTO will begin execution of the collection of data using Cylr (Orlikoski. 2018), and notify the user of the steps being taken. The zip file created by YAFORTO will be named after the target computer, and moved from the remote to the local computer.

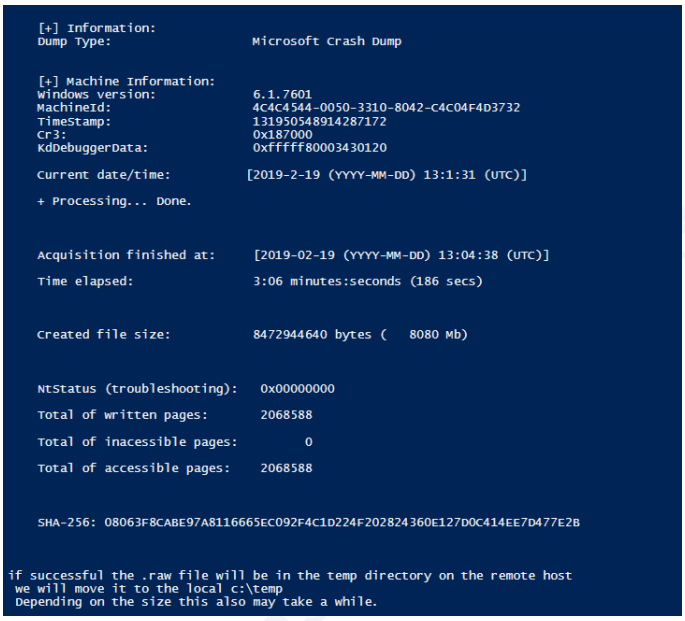


The next collection point is the memory collection. Here the analyst is asked which program to use.



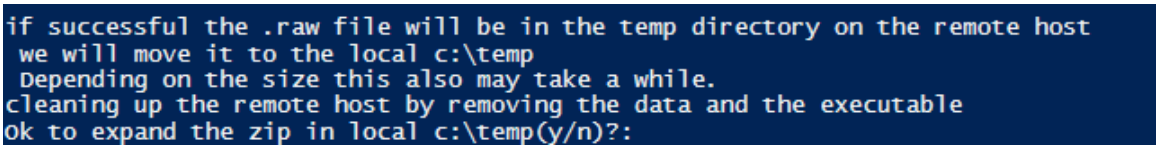
Upon selection, the appropriate memory program will grab the memory place it in a file, move that file over to the analyst's computer, then delete the remote executable and the remote file. The output of the command will remain untouched on the local machine until it is finished. Then it will be copied to the Forensic Analyst's machine.



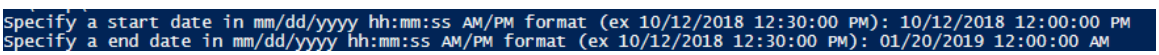


Future versions of the YAFORTO program will give the option of copying directly to a shared drive that is mapped at login. The process will also check the SHA-256 checksum for accuracy or tampering.

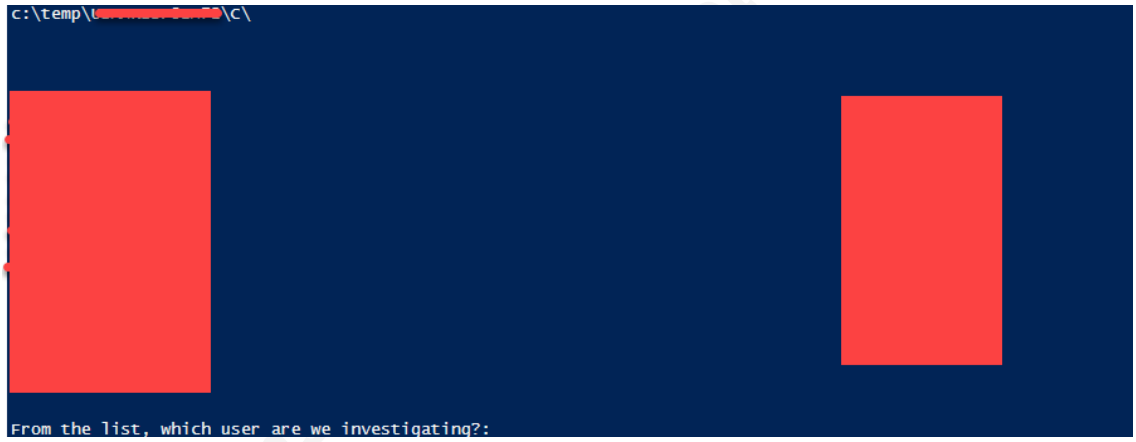
After all the files are copied to the source machine, the next function 'UnZipIT' will uncompress the .zip file for the machine in question. The analyst is asked if expanding to the local C:\temp directory is ok. If not the Analyst is then asked to specify a full path of an existing directory. This is assigned to a variable used in other functions inside YAFORTO.



Once the .zip file is expanded, the analyst will be asked a specific date range. Of importance to this query is the realization of the timezone previously selected. The date and time zone will be converted using the time zone to UTC so it can be matched with the output of other functions.



The analyst is then presented with the list of directories from c:\Users from the uncompressed file and asked which user is being investigated. This is used by the reg_rip function inside YAFORTO to parse the NTUSER.DAT file of the specific user.



The next step uses the information provided to output the registry files into the $TRGRegistry array using a loop and the RegRipper (keydet89,2019) executable.

This is the end of the program, the variables for $TRGRegistry and all other variables in the program should be available for use in the command line. Future versions of the program will take these variables and place them in a report.

2.4. Future development

To accommodate the limited time to produce at least a functioning script that could be documented for use in triage, there were a couple of things on the development roadmap that did not make it into the current version. The information below details those items that are being considered for inclusion into the next version.

2.4.1. Full disk imaging and MD5 Hash.

Full disk imaging can also be automated to some extent. Though not released in the current version of YAFORTO, full disk imaging is also in development. The plan is

to use either dd for Windows ("chrysocome.net") or the native dd for Linux ("dd"). Preliminary testing of the dd for Windows ("chrysocome.net") version lends itself well to automation through command line switches and will make itself into the next release. MD5 hash checking of output from YAFORTO for all the files, and a report showing validation will also make its way into the next version of YAFORTO.

2.4.2. Graphical printout

The ability to output the data in a graphical format is also beneficial to the forensic analyst so that it can be delivered to higher level stakeholders. To facilitate the future planning of this, the program is designed to conform all the data points into arrays with labels. These arrays can then be exported using the native 'Export-CSV' command of PowerShell and ingested into other programs. As further development of this script is done, the developer will publish the array data and consolidate arrays where feasible or where they logically fit together. The rip_reg function is a prime candidate for this activity and will probably be the first converted when time permits.

Also not in time for this release, the ability to report the data in a graphical interface, similar to ProcDot (Wojner,2019) which represents Malware execution. It may come down to rewriting the current script into a Python script and taking advantage of Tkinter ("Page",2019) and maybe Matplotlib ("Matplotlib: Python plotting — Matplotlib 3.0.2 documentation" 2019) could be used. A potential approach would be to present a graphic showing each action taken in chronological order with arrows for direction.

2.4.3. Other Desirable options

The YAFORTO script was developed to address a need for remote collection and evaluation of forensic data. The first version was a proof of concept to see what was possible. The later versions of the script will have a common goal of trying to present the Forensic Analyst a timeline that they can then use to report on suspicious activity. This

assumption was designed into the script to allow for further development by the way of commenting excessively and keeping the scripting simple. The code can be cannibalized by anyone to fit their needs. The code and execution of the code is provided 'as is' with no warranty nor liability implied or otherwise considered.

Another option discovered in exploring the design of the code was using alternate execution paths to avoid using local disk space and potentially corrupting slack space on the disk. The developer is exploring this. As functions are developed to be used inside the script, they will also be posted on the GitHub repository for YAFORTO (Manta01, 2018).

3. Conclusion

Not everyone can be a Forensic Analyst and the time it takes to learn this craft competently is out of reach for some individuals. The YAFORTO script is a good starting point to address this learning gap, allowing anyone with the appropriate permissions to collect data, which could be used by a forensics team to perform a Forensic Analysis. The current version of the script is currently being used in a global company to obtain forensic triage images remotely in reaction to Incident Response. Further development for use with Security Operations Center personnel is the goal and is on the road map for YAFORTO.

A by-product of the design of the script is that those familiar with other programming languages, or even those with PowerShell expertise can cannibalize the existing script for their use. The current script is over 1000 lines long and functions, for the most part, are at the top of the script. This paper was a by-product of the development of the script. In the appendix, you will see the current version of the script. It is encouraged to take a look at it and feel free to use it as is or cannibalize it.

Kenneth Ray, Kenneth.Ray@mkcorp.com

References

Chrysocome.net. (n.d.). Retrieved February 16, 2019, from http://www.chrysocome.net/dd

Dd. (n.d.). Retrieved February 16, 2019, from https://ss64.com/bash/dd.html

keydet89. “keydet89/RegRipper2.8.” GitHub, 28 Jan. 2019, github.com/keydet89/RegRipper2.8.

Manta0101. (2018, December 06). Manta0101/yaforto. Retrieved December 20, 2018, from https://github.com/manta0101/yaforto

"Matplotlib: Python Plotting — Matplotlib 3.0.2 Documentation." Matplotlib: Python Plotting - Matplotlib 2.2.3 Documentation, Retrieved February 13, 2019, from matplotlib.org

Orlikoski. (2018, December 31). Orlikoski/CyLR. Retrieved January 16, 2019, from https://github.com/orlikoski/CyLR

“Page.” TcpCommunication - Python Wiki, Retrieved February 13, 2019, from https://wiki.python.org/moin/TkInter

“The Volatility Foundation - Open Source Memory Forensics.” The Volatility Foundation - Open Source Memory Forensics, Retrieved February 15, 2019, from www.volatilityfoundation.org/.

Thomas. (n.d.). List of Timezone ID's for use with FindTimeZoneById() in C#? Retrieved December 05, 2018, from https://stackoverflow.com/questions/7908343/list-of-timezone-ids-for-use-with-findtimezonebyid-in-c

“Page.” TcpCommunication - Python Wiki, wiki.python.org/moin/TkInter.

"WinRM QuickConfig, HowTo Enable via GPO or Remotely on All Servers." PC & Network Downloads - PCWDLD.com, 24 Jan. 2019, www.pcwdld.com/winrm-quickconfig-remotely-configure-and-enable.

Wojner, Christian. "ProcDOT's Home." ProcDOT's Home, Retrieved February 13, 2019, from http://www.procdot.com

Appendix

YAFORTO script

The current version of this script is maintained at

https://github.com/manta0101/yaforto

```
#############################################################
## Variable list
#############################################################
# this section defines the variables that are being used throughout the
program.
#############################################################################
#
# Target and credential information
#     $Mycomputer_name =  defines computer name used throughout the program and
in many functions
#     $MySecureCreds =defines the user name and password to use for grabbing
data and execution of scripts
#                     this ID should have the ability to read everything.
#         -$Username = builds $mySecureCreds
#         -$SecureString = password for username
#     $minUTC =variable used to check against reg ripper
#     $maxUTC =variable used to check against reg ripper
#
####
# Cylr_pull function
#  This function uses the Cylr.exe to create a triage image on the remote
machine in zip format. it also pulls
#  the image back locally from where it is running is uses the $MyComputer_name
as the target and $MySecureCreds
# as credentials.
#      $MyClyr_exe = location of the CyLR.exe file
#
####
# Zip function
#      $My7Ziplocation = defines the location of the 7- zip executable does not
have a prompt
#                        because it should be the same for your organization.
#           -(reuse) $computer_name to unzip the file zip file and set a
working directory
#      $MyexandDir = where the zip file is expanded to c:\temp is the default
#      $myTempDecision = used in the question of expand directory.
#      $myworkingDirectory = "$MyexandDir\$computer_name" so we dont have to
keep hardcoding the path of the
#                             directory location.
####
# Firewall rules Function
# this function will grab the remote firewall rules it depends on native
windows command netsh.exe being
# on the remote system.original code Retrieved from
http://powershelldistrict.com/netsh-advfirewall-powershell
# and modified to fit remote collection and only show enabled rules.
#      $RuleNae = original function could accept a rule name.Left for posterity
#      $Rules = this is an array to hold the command run on the remote computer
to capture all the firewall
#               rules.
#      $return= an array to used to return data
#      $HAsh = an ordered hash used to parse the returned informatio from
$rules
#      $Rule = temporary variable for a foreach loop
#      $remote_FWrules = an array to hold the enabled rules.
```

```
#
####
# Investigation answers Section
#      $UserInvest = User being investigated. This will be used to load the
registry files and do searches that are based
#                    on the user.Prior to this an enumeration of the expanded
zip file c\users\ will be presented to the
#                    investigator. *Note* due to this not being passed back
correctly. It has been moved outside the
#
####
#
# ------------------------------------------------------------------------
---
# end of Variable list
# ------------------------------------------------------------------------
---

#############################################################################
###
#############################################################################
###
# MAIN
#############################################################################
###

#gather module
#############################################################################
#
##### prompts for user name, machine name and times
#############################################################################

# Declare Variables for script if they are not supplied at the command line
then prompt user
#########################################################
$Mycomputer_name = $args[0]
$MySecureCreds = $args[1]

# Check to see if Computer name was supplied

# if it wasn't then prompt for it
if ($Mycomputer_name -eq $null) {
$Mycomputer_name=Read-Host -prompt 'computer name not provided on call,
computername?'
}
#check to see if secureCreds were passed.
# if not then prompt and set them
if ($MySecureCreds -eq $null) {

#prompt user for a fully qualified user name
$Username = Read-Host -prompt 'Provide user name here suggest using a fully
qualified name like usa\username'

#prompt the user for the password while also still keeping it secret
#and store it as a securestring.
$SecureString = Read-Host -prompt 'Provide Password for the account' -
AsSecureString

#convert the string into the MySecureCreds variable.
$MySecureCreds = New-Object -TypeName System.Management.Automation.PSCredential
-ArgumentList $Username,$SecureString
}

#################################################################
### get Time frame
#################################################################
```

```
##get time and date and convert to UTC for functions  need to be
## defined before the functions are called.
#################################################################
##get time and date and convert to UTC for functions
$getRange =""
#$getRange =Read-host "11/12/2018 07:20:00 AM"
$getRange= Get-Date $getRange
$chosenTImeZone =""
$targetUTC =""
function timezonetarget{#this is a list of timezones available to the
FindtimezonebyID. retreived from
#https://stackoverflow.com/questions/7908343/list-of-timezone-ids-for-use-with-
findtimezonebyid-in-c
# retreived 12-5-18 2018
#formated by me to include in a script so i can convert the target timezone to
UTC.  the user can choose  a number from the 97 available.
Clear
#first Line
Write-host "what is the target timezone? Choose a number."
Write-host "-------------------------------------------"
write-host "1)" -ForegroundColor Red -NoNewline;  Write-host "Dateline Standard
Time" -NoNewline;
Write-host " 2)" -ForegroundColor Red -NoNewline; Write-host "UTC-11" -
NoNewline ;
write-host " 3)" -ForegroundColor Red -NoNewline; Write-host "Samoa Standard
Time"-NoNewline;
Write-host " 4)" -ForegroundColor Red -NoNewline; Write-Host "Hawaiian Standard
Time" -NoNewline
Write-host " 5)" -ForegroundColor Red -NoNewline; Write-Host "Alaskan Standard
Time" -NoNewline
write-host " 6)" -ForegroundColor Red -NoNewline; Write-host "Pacific Standard
Time (Mexico) "
#2nd line
Write-host " 7)" -ForegroundColor Red -NoNewline; Write-Host "Pacific Standard
Time"-NoNewline;
Write-Host " 8)" -ForegroundColor Red -NoNewline; Write-Host "US Mountain
Standard Time "-NoNewline;
Write-Host " 9)" -ForegroundColor Red -NoNewline; Write-Host "Mountain Standard
Time (Mexico) " -NoNewline;
Write-Host " 10)"-ForegroundColor Red -NoNewline; Write-Host "Mountain Standard
Time "-NoNewline;
Write-Host " 11)"-ForegroundColor Red -NoNewline; Write-Host "Central America
Standard Time "
#3rd Line
Write-Host " 12)"-ForegroundColor Red -NoNewline;Write-Host "Central Standard
Time "-NoNewline;
Write-Host " 13)"-ForegroundColor Red -NoNewline;Write-Host "Central Standard
Time (Mexico) "-NoNewline;
Write-Host " 14)"-ForegroundColor Red -NoNewline;Write-Host "Canada Central
Standard Time "-NoNewline;
Write-Host " 15)"-ForegroundColor Red -NoNewline;Write-Host "SA Pacific
Standard Time "-NoNewline;
Write-Host " 16)"-ForegroundColor Red -NoNewline;Write-Host "Eastern Standard
Time"

#4th line
Write-Host " 17)"-ForegroundColor Red -NoNewline;Write-Host "US Eastern
Standard Time"-NoNewline;
Write-Host " 18)"-ForegroundColor Red -NoNewline;Write-Host "Venezuela Standard
Time"-NoNewline;
Write-Host " 19)"-ForegroundColor Red -NoNewline;Write-Host "Paraguay Standard
Time"-NoNewline;
Write-Host " 20)"-ForegroundColor Red -NoNewline;Write-Host "Atlantic Standard
Time"-NoNewline;
Write-Host " 21)"-ForegroundColor Red -NoNewline;Write-Host "Central Brazilian
Standard Time"
#5th line
```

```
Write-Host " 22)"-ForegroundColor Red -NoNewline;Write-Host "SA Western
Standard Time "-NoNewline;
Write-Host " 23)"-ForegroundColor Red -NoNewline;Write-Host "Pacific SA
Standard Time "-NoNewline;
Write-Host " 24)"-ForegroundColor Red -NoNewline;Write-Host "Newfoundland
Standard Time "-NoNewline;
Write-Host " 25)"-ForegroundColor Red -NoNewline;Write-Host "E. South America
Standard Time "-NoNewline;
Write-Host " 26)"-ForegroundColor Red -NoNewline;Write-Host "Argentina Standard
Time"
#6th line
Write-Host " 27)"-ForegroundColor Red -NoNewline;Write-Host "SA Eastern
Standard Time "-NoNewline;
Write-Host " 28)"-ForegroundColor Red -NoNewline;Write-Host "Greenland Standard
Time "-NoNewline;
Write-Host " 29)"-ForegroundColor Red -NoNewline;Write-Host "Montevideo
Standard Time "-NoNewline;
Write-Host " 30)"-ForegroundColor Red -NoNewline;Write-Host "UTC-02  "-
NoNewline;
Write-Host " 31)"-ForegroundColor Red -NoNewline;Write-Host "Mid-Atlantic
Standard Time "-NoNewline;
Write-Host " 32)"-ForegroundColor Red -NoNewline;Write-Host "Azores Standard
Time"


#seventh line
Write-Host " 33)"-ForegroundColor Red -NoNewline;Write-Host "Cape Verde
Standard Time "-NoNewline;
Write-Host " 34)"-ForegroundColor Red -NoNewline;Write-Host "Morocco Standard
Time "-NoNewline;
Write-Host " 35)"-ForegroundColor Red -NoNewline;Write-Host "UTC  "-NoNewline;
Write-Host " 36)"-ForegroundColor Red -NoNewline;Write-Host "GMT Standard Time
"-NoNewline;
Write-Host " 37)"-ForegroundColor Red -NoNewline;Write-Host "Greenwich Standard
Time "-NoNewline;
Write-Host " 38)"-ForegroundColor Red -NoNewline;Write-Host "W. Europe Standard
Time"

#8th line
Write-Host " 39)"-ForegroundColor Red -NoNewline;Write-Host "Central Europe
Standard Time "-NoNewline;
Write-Host " 40)"-ForegroundColor Red -NoNewline;Write-Host "Romance Standard
Time "-NoNewline;
Write-Host " 41)"-ForegroundColor Red -NoNewline;Write-Host "Central European
Standard Time "-NoNewline;
Write-Host " 42)"-ForegroundColor Red -NoNewline;Write-Host "W. Central Africa
Standard Time  "-NoNewline;
Write-Host " 43)"-ForegroundColor Red -NoNewline;Write-Host "Namibia Standard
Time"

#9th line
Write-Host " 44)"-ForegroundColor Red -NoNewline;Write-Host "Jordan Standard
Time  "-NoNewline;
Write-Host " 45)"-ForegroundColor Red -NoNewline;Write-Host "GTB Standard Time
"-NoNewline;
Write-Host " 46)"-ForegroundColor Red -NoNewline;Write-Host "Middle East
Standard Time  "-NoNewline;
Write-Host " 47)"-ForegroundColor Red -NoNewline;Write-Host "Egypt Standard
Time  "-NoNewline;
Write-Host " 48)"-ForegroundColor Red -NoNewline;Write-Host "Syria Standard
Time  "-NoNewline;
Write-Host " 49)"-ForegroundColor Red -NoNewline;Write-Host "South Africa
Standard Time "

#10th line
Write-Host " 50)"-ForegroundColor Red -NoNewline;Write-Host "FLE Standard Time
"-NoNewline;
```

```
Write-Host " 51)"-ForegroundColor Red -NoNewline;Write-Host "Israel Standard
Time "-NoNewline;
Write-Host " 52)"-ForegroundColor Red -NoNewline;Write-Host "E. Europe Standard
Time "-NoNewline;
Write-Host " 53)"-ForegroundColor Red -NoNewline;Write-Host "Arabic Standard
Time "-NoNewline;
Write-Host " 54)"-ForegroundColor Red -NoNewline;Write-Host "Arab Standard Time
"-NoNewline;
Write-Host " 55)"-ForegroundColor Red -NoNewline;Write-Host "Russian Standard
Time"

#12th
Write-Host " 56)"-ForegroundColor Red -NoNewline;Write-Host "E. Africa Standard
Time "-NoNewline;
Write-Host " 57)"-ForegroundColor Red -NoNewline;Write-Host "Iran Standard Time
"-NoNewline;
Write-Host " 58)"-ForegroundColor Red -NoNewline;Write-Host "Arabian Standard
Time "-NoNewline;
Write-Host " 59)"-ForegroundColor Red -NoNewline;Write-Host "Azerbaijan
Standard Time "-NoNewline;
Write-Host " 60)"-ForegroundColor Red -NoNewline;Write-Host "Mauritius Standard
Time "-NoNewline;
Write-Host " 61)"-ForegroundColor Red -NoNewline;Write-Host "Georgian Standard
Time"

#13th
Write-Host " 62)"-ForegroundColor Red -NoNewline;Write-Host "Caucasus Standard
Time "-NoNewline;
Write-Host " 63)"-ForegroundColor Red -NoNewline;Write-Host "Afghanistan
Standard Time "-NoNewline;
Write-Host " 64)"-ForegroundColor Red -NoNewline;Write-Host "Ekaterinburg
Standard Time "-NoNewline;
Write-Host " 65)"-ForegroundColor Red -NoNewline;Write-Host "Pakistan Standard
Time "-NoNewline;
Write-Host " 66)"-ForegroundColor Red -NoNewline;Write-Host "West Asia Standard
Time"-NoNewline;
Write-Host " 67)"-ForegroundColor Red -NoNewline;Write-Host "India Standard
Time"

#14th
Write-Host " 68)"-ForegroundColor Red -NoNewline;Write-Host "Sri Lanka Standard
Time "-NoNewline;
Write-Host " 69)"-ForegroundColor Red -NoNewline;Write-Host "Nepal Standard
Time "-NoNewline;
Write-Host " 70)"-ForegroundColor Red -NoNewline;Write-Host "Central Asia
Standard Time "-NoNewline;
Write-Host " 71)"-ForegroundColor Red -NoNewline;Write-Host "Bangladesh
Standard Time "-NoNewline;
Write-Host " 72)"-ForegroundColor Red -NoNewline;Write-Host "N. Central Asia
Standard Time "-NoNewline;
Write-Host " 73)"-ForegroundColor Red -NoNewline;Write-Host "Myanmar Standard
Time"

#15th
Write-Host " 74)"-ForegroundColor Red -NoNewline;Write-Host "SE Asia Standard
Time "-NoNewline;
Write-Host " 75)"-ForegroundColor Red -NoNewline;Write-Host "North Asia
Standard Time "-NoNewline;
Write-Host " 76)"-ForegroundColor Red -NoNewline;Write-Host "China Standard
Time "-NoNewline;
Write-Host " 77)"-ForegroundColor Red -NoNewline;Write-Host "North Asia East
Standard Time "-NoNewline;
Write-Host " 78)"-ForegroundColor Red -NoNewline;Write-Host "Singapore Standard
Time "-NoNewline;
Write-Host " 79)"-ForegroundColor Red -NoNewline;Write-Host "W. Australia
Standard Time"
```

```
#16th
Write-Host " 80)"-ForegroundColor Red -NoNewline;Write-Host "Taipei Standard
Time "-NoNewline;
Write-Host " 81)"-ForegroundColor Red -NoNewline;Write-Host "Ulaanbaatar
Standard Time "-NoNewline;
Write-Host " 82)"-ForegroundColor Red -NoNewline;Write-Host "Tokyo Standard
Time "-NoNewline;
Write-Host " 83)"-ForegroundColor Red -NoNewline;Write-Host "Korea Standard
Time "-NoNewline;
Write-Host " 84)"-ForegroundColor Red -NoNewline;Write-Host "Yakutsk Standard
Time "-NoNewline;
Write-Host " 85)"-ForegroundColor Red -NoNewline;Write-Host "Cen. Australia
Standard Time"

#17th
Write-Host " 86)"-ForegroundColor Red -NoNewline;Write-Host "AUS Central
Standard Time "-NoNewline;
Write-Host " 87)"-ForegroundColor Red -NoNewline;Write-Host "E. Australia
Standard Time "-NoNewline;
Write-Host " 88)"-ForegroundColor Red -NoNewline;Write-Host "AUS Eastern
Standard Time "-NoNewline;
Write-Host " 89)"-ForegroundColor Red -NoNewline;Write-Host "West Pacific
Standard Time "-NoNewline;
Write-Host " 90)"-ForegroundColor Red -NoNewline;Write-Host "Tasmania Standard
Time"

#18th
Write-Host " 91)"-ForegroundColor Red -NoNewline;Write-Host "Vladivostok
Standard Time "-NoNewline;
Write-Host " 92)"-ForegroundColor Red -NoNewline;Write-Host "Central Pacific
Standard Time"-NoNewline;
Write-Host " 93)"-ForegroundColor Red -NoNewline;Write-Host "New Zealand
Standard Time "-NoNewline;
Write-Host " 94)"-ForegroundColor Red -NoNewline;Write-Host "UTC+12 95)Fiji
Standard Time "-NoNewline;
Write-Host " 96)"-ForegroundColor Red -NoNewline;Write-Host "Kamchatka Standard
Time "-NoNewline;
Write-Host " 97)"-ForegroundColor Red -NoNewline;Write-Host "Tonga Standard
Time"
$timeZoneChoice =Read-host "pick your number"

$chosenTImeZone = Switch($timeZoneChoice){
1 {"Dateline Standard Time";break }
2 {"UTC-11";break }
3 {"Samoa Standard Time";break }
4 {"Hawaiian Standard Time";break }
5 {"Alaskan Standard Time";break }
6 {"Pacific Standard Time (Mexico)";break }
7 {"Pacific Standard Time";break }
8 {"US Mountain Standard Time";break }
9 {"Mountain Standard Time (Mexico)";break }
10 {"Mountain Standard Time";break }
11 {"Central America Standard Time";break }
12 {"Central Standard Time";break }
13 {"Central Standard Time (Mexico)";break }
14 {"Canada Central Standard Time";break }
15 {"SA Pacific Standard Time";break }
16 {"Eastern Standard Time";break }
17 {"US Eastern Standard Time";break }
18 {"Venezuela Standard Time";break }
19 {"Paraguay Standard Time";break }
20 {"Atlantic Standard Time";break }
21 {"Central Brazilian Standard Time";break }
22 {"SA Western Standard Time";break }
23 {"Pacific SA Standard Time";break }
24 {"Newfoundland Standard Time";break}
25 {"E. South America Standard Time";break }
```

```
26 {"Argentina Standard Time";break }
27 {"SA Eastern Standard Time";break }
28 {"Greenland Standard Time";break }
29 {"Montevideo Standard Time";break }
30 {"UTC-02";break }
31 {"Mid-Atlantic Standard Time";break }
32 {"Azores Standard Time";break }
33 {"Cape Verde Standard Time";break }
34 {"Morocco Standard Time";break }
35 {"UTC";break }
36 {"GMT Standard Time";break }
37 {"Greenwich Standard Time";break}
38 {"W. Europe Standard Time";break}
39 {"Central Europe Standard Time";break}
40 {"Romance Standard Time";break }
41 {"Central European Standard Time";break}
42 {"W. Central Africa Standard Time"; break}
43 {"Namibia Standard Time";break }
44 {"Jordan Standard Time";break }
45 {"GTB Standard Time";break }
46 {"Middle East Standard Time";break }
47 {"Egypt Standard Time";break }
48 {"Syria Standard Time";break }
49 {"South Africa Standard Time";break }
50 {"FLE Standard Time";break }
51 {"Israel Standard Time";break }
52 {"E. Europe Standard Time";break }
53 {"Arabic Standard Time";break }
54 {"Arab Standard Time";break }
55 {"Russian Standard Time";break }
56 {"E. Africa Standard Time";break }
57 {"Iran Standard Time";break }
58 {"Arabian Standard Time";break }
59 {"Azerbaijan Standard Time";break }
60 {"Mauritius Standard Time";break }
61 {"Georgian Standard Time";break }
62 {"Caucasus Standard Time";break }
63 {"Afghanistan Standard Time";break }
64 {"Ekaterinburg Standard Time";break }
65 {"Pakistan Standard Time ";break }
66 {"West Asia Standard Time";break }
67 {"India Standard Time";break }
68 {"Sri Lanka Standard Time";break }
69 {"Nepal Standard Time";break }
70 {"Central Asia Standard Time";break }
71 {"Bangladesh Standard Time";break }
72 {"N. Central Asia Standard Time";break }
73 {"Myanmar Standard Time";break }
74 {"SE Asia Standard Time";break }
75 {"North Asia Standard Time";break }
76 {"China Standard Time";break }
77 {"North Asia East Standard Time";break }
78 {"Singapore Standard Time";break }
79 {"W. Australia Standard Time";break }
80 {"Taipei Standard Time";break }
81 {"Ulaanbaatar Standard Time";break }
82 {"Tokyo Standard Time";break }
83 {"Korea Standard Time";break }
84 {"Yakutsk Standard Time";break }
85 {"Cen. Australia Standard Time";break }
86 {"AUS Central Standard Time";break }
87 {"E. Australia Standard Time";break }
88 {"AUS Eastern Standard Time";break }
89 {"West Pacific Standard Time";break }
90 {"Tasmania Standard Time";break }
91 {"Vladivostok Standard Time";break }
92 {"Central Pacific Standard Time";break }
```

```
93 {"New Zealand Standard Time";break }
94 {"UTC+12";break }
95 {"Fiji Standard Time ";break }
96 {"Kamchatka Standard Time";break }
97 {"Tonga Standard Time";break }
}
return $chosenTImeZone

}
#for some reason the return variable is just not working as expected.
# to compensate I am creating a temporary variable.

#set up the timezone choice to be used in the Time conversion.
$chosenTImeZone += timezonetarget
function time-convertUTC{
Param(

        [parameter(position=1)]
        $targetTImeZone,
        [parameter(position=0)]
        $getRange

        )

$time = get-date -date $getRange
###testing variables###
#Write-host "original"
#$time
#Pause
####################
$oFromTimeZone= $targetTImeZone
$fromTimeZone = $oFromTimeZone
$oToTimeZone = "UTC"
$toTimeZone = $oToTimeZone
#$time.ToUniversalTime($oFromTimeZone)
#$utc = [System.TimeZoneInfo]::ConvertTimeToUtc($time, $oFromTimeZone)
#“2014-07-29T03:15:00”
$oFromTimeZone = [System.TimeZoneInfo]::FindSystemTimeZoneById($fromTimeZone)
  $oToTimeZone = [System.TimeZoneInfo]::FindSystemTimeZoneById($toTimeZone)
  $utc = [System.TimeZoneInfo]::ConvertTimeToUtc($time, $oFromTimeZone)
  $UTCTime = [System.TimeZoneInfo]::ConvertTime($utc, $oToTimeZone)
 #$oFromTimeZone
 #$oToTimeZone
 #pause
 #$utc
 #pause
 $UTCTime
 }

Function query_time{ #start of Function
 #set a temporary target global Variable for the UTC conversion.
 # note that the function needs a += to work correctly
 $targetUTC =""
 #read from host a target start date and time
 $getminRange = read-host "Specify a start date in mm/dd/yyyy hh:mm:ss AM/PM
format (ex 10/12/2018 12:30:00 PM)"
 #convert it to something we can use to do date math
 [double]$min = (Get-date "$getminRange").ToFileTime()
 #we need to do a seperate conversion for UTC. Note that it was defined in the
$chosenTimeZone Variable.
 $tempMinUTC = (Get-Date "$getminRange")
 ##reset the temp variable.
 $targetUTC = ""
 $targetUTC += (time-convertUTC $tempMinUTC $chosenTImeZone)
 $minUTC = (Get-date "$targetUTC").ToFileTime()
```

```
 $getMaxRange = read-host "Specify a end date in mm/dd/yyyy hh:mm:ss AM/PM
format (ex 10/12/2018 12:30:00 PM)"
 [double]$max = (Get-date "$getMaxRange").ToFileTime()
 $targetUTC =""
 $tempMaxUTC = (Get-Date "$getMaxRange")
 $targetUTC += (time-convertUTC $tempMaxUTC $chosenTImeZone)
 $maxUTC = (Get-date "$targetUTC").ToFileTime()

 $minUTC
 $maxUTC
 } ##End of function
 ##################################################End Functon
function rip_reg {
#Local Variables
#       $mySAMRegWork= Uses $myworkingDirectory + "windows\system32\config\SAM"
to access the SAM file
#       $mySYSTEMRegWork   = Uses $myworkingDirectory +
"windows\system32\config\SYSTEM" to access the SYSTEM HIVE
#       $mySECURITYRegWork = Uses$myworkingDirectory +
"windows\system32\config\SECURITY" to access the SECURITY hive
#       $mySOFTWARERegWork = Uses$myworkingDirectory +
"windows\system32\config\SOFTWARE" to access the SOFTWARE hive
#       $myUSERRegWork     = Uses $myworkingDirectory+"Users\"+
$UserInvest+"\NTUSER.DAT"
#       $registryFiles    = array for the above so we can iterate through each
collectively
#       $myRegRipperLocation = location of the Regripper command rip.exe
#       $registryFile = Temporary variable for the loop of
#       $out = array to hold the reg ripper data
#       $dataSplit = ordered hash to hold the results of both keys and
##########
$mySAMRegWork= $myworkingDirectory + "windows\system32\config\SAM"
$mySYSTEMRegWork= $myworkingDirectory + "windows\system32\config\SYSTEM"
$mySECURITYRegWork= $myworkingDirectory + "windows\system32\config\SECURITY"
$mySOFTWARERegWork= $myworkingDirectory + "windows\system32\config\SOFTWARE"
$myUSERRegWork = $myworkingDirectory+"Users\"+$UserInvest+"\NTUSER.DAT"

#create an Array to sift through that includes all the Registry files. IF one
is missing just add it above and then below
$registryFiles =
$mySAMRegWork,$mySYSTEMRegWork,$mySECURITYRegWork,$mySOFTWARERegWork,$myUSERReg
Work
#name of executable
$myRegripperEXE = "rip.exe"
#location of the executabe *need this for the Change directory command to make
the plugins work
$myRegRipperDir = "C:\Temp\RegRipper2.8-master\"
# combination of both the exe and directory used to execute the command.
$myRegRipperLocation = $myRegRipperDir+$myRegripperEXE

$minUTC
$maxUTC
write-host " should be two numbers above this line. if not break out"
Pause
#
$returnhash = New-Object -TypeNamePSobject
$returnObje
#################################################################
#looking for entries with the regtime plugin of regripper
#################################################################
# for the plugins to work you need to be in the directory where the Regripper
rip.exe is located.
#################################################################################
##################
& cd $myRegRipperDir
```

```
########################################
### check for timeframe on all files
########################################
Foreach ($registryFile in $registryFiles) {
$out = @()
Write-host "looking for matching timeframe entries for $registryFile.`n You can
safely ignore the error for CMD.exe : Launching regtime plugin ...`n Its an
expected handling issue with the PowerShell command line execution"
$out = & cmd /c "$myRegRipperLocation -r $registryFile -p regtime"

$DAtaSplit =[ordered]@{}
$out2 =$out #|Select-String "Key:"



foreach ($KeyString in $out2) {
#match first part of string (should be write time)
if ($KeyString -match '^\S+\s+\S+\s+\S+\s+\S+\s+\S+\s+'){
$Datasplit.LastWriteTime =$Matches[0]
}
#Match second part of the string Should be the Key
#
if ($keystring -match '\s{2,}\S+'){
$DAtaSplit.Key = $Matches[0]
}
#$Datasplit.LastWriteTime, $Datasplit.key  =$KeyString -split '\s{3,}'
#replace because the format is 'Fri Nov 16 18:48:52 2018 Z' we need to convert
so we can run a comparison
#$DAtaSplit.LastWriteTime
#because of the output of the regripper command we need to convert it to the
corrected date.
$test= $DAtaSplit.LastWriteTime -replace "^...\s" -replace "Z" -replace
"Jan.","01/" -replace "Feb.","02/" -replace "Mar.","03/" -replace "Apr.","04/"
-replace "May.","05/" -replace "Jun.","06/" -replace "Jul.","07/" -replace
"Aug.","08/" -replace "Sep.","09/" -replace "Oct.","10/" -replace "Nov.","11/"
-replace "Dec.", "12/" -replace "\d\d:\d\d:\d\d","/" -replace " "
#$test
$test = (get-date $test).toFileTime()
#$test to see if it matches the Max and min UTC variables defined in the
Investigation section

if (($test -ge $minUTC) -and ($test -le $maxUTC)) {
$DAtaSplit.ActionTKN = "Change,"
$DAtaSplit.Key = $DAtaSplit.Key -replace "\s+"
Write-host $registryFile "," $DAtaSplit.ActionTKN
$DAtaSplit.LastWriteTime","$DAtaSplit.Key
                    }
}

 }
 ###################################################################
 #check for deletions
 ###################################################################
 Foreach ($registryFile in $registryFiles) {
$out = @()
Write-host "looking for deleted entries for $registryFile.`n You can safely
ignore the error for CMD.exe : Launching del ...`n Its an expected handling
issue with the PowerShell command line execution"
$out = & cmd /c "$myRegRipperLocation -r $registryFile -p del"

$DAtaSplit =[ordered]@{}
$out2 =$out |Select-String "Key:"



foreach ($KeyString in $out2) {
#splitting the string into the two separate parts.
```

```
$Datasplit.key, $Datasplit.LastWriteTime =$KeyString -split 'LW:'
#replace because the format is 'Fri Nov 16 18:48:52 2018 Z' we need to convert
so we can run a comparison

$test= $DAtaSplit.LastWriteTime -replace "^...\s" -replace ".Z|Z" -replace
"Jan.","01/" -replace "Feb.","02/" -replace "Mar.","03/" -replace "Apr.","04/"
-replace "May.","05/" -replace "Jun.","06/" -replace "Jul.","07/" -replace
"Aug.","08/" -replace "Sep.","09/" -replace "Oct.","10/" -replace "Nov.","11/"
-replace "Dec.", "12/" -replace "\d\d:\d\d:\d\d","/" -replace " "
$test = (get-date $test).toFileTime()
if (($test -ge $minUTC) -and ($test -le $maxUTC)) {
$DAtaSplit.ActionTKN = "Delete,"
$KeyString
Write-host  $registryFile "," $DAtaSplit.ActionTKN $DAtaSplit.LastWriteTime ","
$DAtaSplit.Key
                    }
}

 }
 ### Changing directory back to the parent so other Change directory commands
operate correctliy.
 ################################################

 cd ..
}
################################################End Function


 ## end testing

###end of user investigation section

Function TRiageImage{

## triage image grab section
## define the functions for pulling the triage image
##
Function cylr_pull {
#########################################################
## cylr_pull function uses the cylr.exe file defined inside the function as
variable $myClyr_exe
## create a drive so we can copy the information, Note you must have the
ability to map to a Admin share.
#  uses the Mysecurecreds as the credentials
## variables:
#             uses Mycomputer_name previously defined
#             $myClyr_exe =location of Cylr executable
#             Uses $MySecurecreds as credentials
#########################################################
$myClyr_exe = "c:\temp\CyLR.exe"
New-PSDrive -Name cylr -PSProvider FileSystem -Root \\$Mycomputer_name\c$  -
Credential $MySecureCreds
#changes to the newly created psdrive. not really necessary but will generate
an error if it cant be done.
Write-host "checking drive by Switching to it"
cd cylr:
Write-host "copying $myClyr_exe to the remote c:\temp directory"
Copy-Item -path $myClyr_exe cylr:\temp
Write-host "Invoking the cylr.exe executable using the credentials provided"
Invoke-Command -ComputerName $Mycomputer_name -ScriptBlock {cmd.exe /C
"C:\temp\cylr.exe -od c:\temp" } -Credential $MySecureCreds
#notify the end user what happened.
Write-host "if successful the $Mycomputer_name.zip file will be in the temp
directory on the remote host`n we will move it to the local c:\temp"
Copy-Item cylr:\temp\$Mycomputer_name.zip C:\temp
Write-host "cleaning up the remote host by removing the data and the
executable"
```

```
Remove-Item cylr:\temp\$Mycomputer_name.zip
Remove-Item cylr:\Temp\cylr.exe
#change back to the local C drive. because we checked to see if we could CD to
the PSdrive we need to go back
# to C: so we can remove the drive.
c:
# now remove the Drive we created
Remove-PSDrive cylr
#-----------------------------------------------------
# End of Cylar_pull function
#-----------------------------------------------------

}

# -------------------- end of function definitions for section---------------

#execute our function needed to pull an image
cylr_pull
}
#####################################################End Function
Function UnZipIT {
#########################################################
# zip section
#########################################################
# the only function is to unzip the file we just transferred
# using 7-zip is the easiest.
#declare variable
# where is the current version of 7 zip (using the default install location)
#defatult is $My7Ziplocation ="C:\Program Files\7-Zip\7z.exe"
$My7Ziplocation ="C:\Program Files\7-Zip\7z.exe"
# ask the user if we can expand the Zip file locally at c:\temp
$MyexandDir = "c:\temp"

 $myTempDecision = read-host "Ok to expand the zip in local c:\temp(y/n)?"
 $myTempDecision
if ($myTempDecision -ne "y" -and $myTempDecision -ne "n"){
 do
 { Write-Host "you must answer y or n"
 $myTempDecision = Read-Host "y or n?"
 if ($myTempDecision -eq 'y' -or $myTempDecision -eq 'n') { break}

 } while ($myTempDecision -ne 'y' -or $myTempDecision -ne 'n')
 }
 if ($myTempDecision -eq "y") {
 Write-host "using $MyexandDir"} else { $MyexandDir = read-host "supply the
path"}
 #create a directory to drop everything while notifying the user.
 Write-Host "Creating the $Mycomputer_name directory under $MyexandDir"
 #create our working directory.
 New-Item -ItemType directory -Path $MyexandDir\$Mycomputer_name
 #assign a new variable to be used later inside other scripts
 $myworkingDirectory = "$MyexandDir\$Mycomputer_name"

 #run the 7zip command 'x' equals expand all the directories, -o is the output
and the rest is self evident
 #& $My7Ziplocation X -y -o"$MyexandDir\$computer_name"
C:\temp\$Mycomputer_name.zip
 & $My7Ziplocation X -y -o"$myworkingDirectory" C:\temp\$Mycomputer_name.zip
 #-----------------------------------------------------
 # end of zip section
 #-----------------------------------------------------
 return $myworkingDirectory
 }
#####################################################End Function
Function FirewallRules{
 ##########################################################################
 # firewall rules section
```

```
 ##########################################################################
 # Retrieved from http://powershelldistrict.com/netsh-advfirewall-powershell/
 # 10/30/18
 # modified for use and commented by me to understand it.
 # start of function
 # original function was Get-NetshFireWallrule. This one will use the invoke
command with:
 # Invoke-Command -ComputerName $Mycomputer_name -ScriptBlock {cmd.exe /C
"C:\temp\cylr.exe -od c:\temp" } -Credential $MySecureCreds
 ###
 # first set the remote collection
 # create  a function that gets remote firewall rules. we are using the global
credentials $mySecureCreds
 Function Get-remoteFireWallrule  {
    # accept a string Parameter called Rulename
    Param(
        [String]$RuleName
    )
  #original code will accept a firewall rule name.or just give all
  ##if statement to show if a rule name was passed. If not(else), just give all
  ## if ($RuleName){
  ##      $Rules = netsh advfirewall firewall show rule name="$ruleName"
  ##  }else{
  ##      $Rules = netsh advfirewall firewall show rule name="all"
  ##  }

  # modified variable to remotely capture all the rules. we parse this later to
return only enabled rules
  $Rules = Invoke-Command -ComputerName $Mycomputer_name -ScriptBlock {cmd.exe
/C "netsh advfirewall firewall show rule name= `"all`"" } -Credential
$MySecureCreds
       #set an Array called Return.
    $return = @()
    #set a ordered Hash
     $HAsh = [Ordered]@{}
     #loop through the $rule Variable and match using Regex
        foreach ($Rule in $Rules){
            #if a specific rulename was not requested
            if ($Rule -match '^Rule Name:\s+(?<RuleName>.+$)'){
              $Hash.RuleName = $Matches.RuleName
            }Else{

              if ($Rule -notmatch "----------------------------------------
-------------------------"){
                switch -Regex ($Rule){
                #look for rulename and map it to Rulename
                  '^Rule Name:\s+(?<RuleName>.*$)'{$Hash.RuleName =
$MAtches.RuleName;Break}
                  '^Enabled:\s+(?<Enabled>.*$)'{$Hash.Enabled =
$MAtches.Enabled;Break}
                  '^Direction:\s+(?<Direction>.*$)'{$Hash.Direction =
$MAtches.Direction;Break}
                  '^Profiles:\s+(?<Profiles>.*$)'{$Hash.Profiles =
$MAtches.Profiles;Break}
                  '^Grouping:\s+(?<Grouping>.*$)'{$Hash.Grouping =
$MAtches.Grouping;Break}
                  '^LocalIP:\s+(?<LocalIP>.*$)'{$Hash.LocalIP =
$MAtches.LocalIP;Break}
                  '^RemoteIP:\s+(?<RemoteIP>.*$)'{$Hash.RemoteIP =
$MAtches.RemoteIP;Break}
                  '^Protocol:\s+(?<Protocol>.*$)'{$Hash.Protocol =
$MAtches.Protocol;Break}
                  '^LocalPort:\s+(?<LocalPort>.*$)'{$Hash.LocalPort =
$MAtches.LocalPort;Break}
                  '^RemotePort:\s+(?<RemotePort>.*$)'{$Hash.RemotePort =
$MAtches.RemotePort;Break}
```

```
                    '^Edge
traversal:\s+(?<Edge_traversal>.*$)'{$Hash.Edge_traversal =
$MAtches.Edge_traversal;$obj = New-Object psobject -Property $Hash;$return +=
$obj;Break}
                    default {Break}
                }


              }
            }

        }
    # returns(uncomment the one you want.)
    #return only those items that are enabled Note:uncomment the select
statement
    #if you only want to grab rulename and Enabled status.
    return $return|where-object{$_.Enabled -eq 'Yes'}#| select-object Rulename,
Enabled
   # the other option is to return everything regardless this was the original
code
   #return $return
   }

   #set a new table to hold the enabled rules (calls the Get-
remoteFireWallrule)
   $remote_FWrules = Get-remoteFireWallrule

   return $remote_FWrules

#----------------------end of Firewall rules section----------------------
 }
################################################End Function
Function MEMtriage{
#############################################################################
##memory grab section
#############################################################################
## first we will define the two types of memory aquisition programs as a
#  function so the user can have a choice of either.
## Other options can be added as a function using the same format really
###
# function to use dumpit against the $mycomputer_name
#---------------------------------------------------------------------------
 Function runDumpit {
#############################################################################
# Dumpit section
#############################################################################
New-PSDrive -Name Dumpit -PSProvider FileSystem -Root \\$Mycomputer_name\c$  -
Credential $MySecureCreds
#changes to the newly created psdrive. not really necessary but will generate
an error if it cant be done.
Write-host "checking drive by Switching to it"
cd Dumpit:
Write-host "copying the Dumpit.exe from c:\temp to the remote c:\temp
directory."
Copy-Item -path C:\temp\Dumpit.exe dumpit:\temp
#note: that on some systems dumpit.exe will generate an error about not being
able to install. This is not a scripting
#error of this script.
#notify user what we are doing
Write-host "Invoking the dumpit.exe executable using the credentials provided.
This will take a while, be patient!"
#invoke the command on the remote system
Invoke-Command -ComputerName $Mycomputer_name -ScriptBlock {cmd.exe /C
"C:\temp\dumpit.exe /Q /O c:\temp\$Mycomputer_name.raw" } -Credential
$MySecureCreds

#testing commands
```

```
#get-childitem dumpit:\temp
#Pause
#-----------
#notify the end user what happened.
Write-host "if successful the .raw file will be in the temp directory on the
remote host`n we will move it to the local c:\temp`n Depending on the size this
also may take a while."
#dumpit creates a .raw with no name instead of the expected name because the
$MyComputer_name is not a local Variable for the invoke command.
#No matter, we will copy it to c:\temp\ with the computername.
Copy-Item Dumpit:\temp\.raw C:\temp\$Mycomputer_name.raw
#tell the user our actions
Write-host "cleaning up the remote host by removing the data and the
executable"
#remove the .raw file on the remote computer.
Remove-Item Dumpit:\temp\.raw
#remove the dumpit executable

Remove-Item Dumpit:\Temp\dumpit.exe
#change back to the local C drive. because we checked to see if we could CD to
the PSdrive we need to go back
# to C: so we can remove the drive.
c:
# now remove the Drive we created
Remove-PSDrive dumpit
#-----------------end of Dumpit section ----------------------------------#
}
#-------------------------------------------------------------------------
###
# function to use winpmem against the $mycomputer_name
#-------------------------------------------------------------------------
 Function runWinpmem {
#########################################################################
#Start of Wpmem section (alternative to  Dumpit)
#########################################################################
# you can either comment out this part of the script and use
# let user know what is happening '
Write-Host "creating the connection to copy winpmem to the remote"
#create a new PS drive for winpmem
New-PSDrive -Name winpmem -PSProvider FileSystem -Root \\$Mycomputer_name\c$  -
Credential $MySecureCreds
#check to see if the drive worked by changing to it.
# future-put a try here and capture the error codes
cd winpmem:
#copy the winpmem executable to the temp drive.
Copy-Item C:\temp\winpmem_1.6.2.exe winpmem:\temp\winpmem.exe
#tell the end user what we are doing.
Write-Host "starting winpmem dump on remote machine $Mycomputer_name"
#execute the wpmem command remotely using the cmd.exe windows
Invoke-Command -ComputerName $Mycomputer_name -ScriptBlock {cmd.exe /C
"C:\temp\winpmem.exe $Mycomputer_name.raw" } -Credential $MySecureCreds
#tell the user we are moving the raw file locally
Write-Host "copying $Mycomputer_name.raw to local computer`n this may take a
while"
#copy the item locally
Copy-Item winpmem:\temp\$Mycomputer_name.raw C:\Temp
#cleanup time
write-host "doing some clean up"
#remove the winpmem file we created on the remote machine
Remove-Item winpmem:\temp\winpmem.exe
#because we are on the psdrive we created we need to get off it so we can
delete it.
c:
#remove the drive.
Remove-PSDrive winpmem

}
```

```
#-------------------------------------------------------------------------
# -------------------- end of function definitions for section--------------

# ask the user which one they are using. Winpmem or dumpit.
 $my_memorydumper = Read-Host "which memory dump program should we use
dumpit(d) or winpmem(w)"
#enter a do.. while loop to prompt for the correct response.. when met we
execute either runDumpit or runWinpmem
  do {
#check if d was supplied if so run dumpit
   if ($my_memorydumper -eq 'd') {
    #run the dumpit function.
    runDumpit
    #without this break statement the process will go into an infinite loop
    break}
#check if w was supplied if so run winpmem
    elseif ($my_memorydumper -eq 'w') {
     #run the Winpmem function
     runWinpmem
     #without this break statement the process will go into an infinite loop
     break}
# otherwise keep asking for the correct response
   else {
   $my_memorydumper = Read-Host "you need to supply either w or d"
   }
   }
#the while test, if neither d nor w it should redo the same over and over.
   while ($my_memorydumper -ne 'd' -or $my_memorydumper -ne 'w')
#-----------------------end of memory grab section-------------------------
}
#####################################################End Function
Function Investigation {
#############################################################################
# user and time frame section
#############################################################################
# in this section we determine who we are investigating and the timeframe
# define the user we may want to look at using the $myworkingDirectory. This
# is where we unzipped the Cylr triage image.
###
#user section
 Write-host "here are the users as defined in the triage images"
#use the variable where we unzipped the triage image($myworkingDirectory) and
read c\User selecting only
#the name and using format-wide to display
#note: $myworkingDirectory is set in unZip Section.
 Get-ChildItem $myworkingDirectory\C\Users |format-wide -property Name
 $UserInvest = Read-Host "From the list, which user are we investigating?"
###
#timeframe section
 $getminRange = read-host "Specify a start date in mm/dd/yyyy hh:mm:ss AM/PM
format (ex 10/12/2018 12:30:00 PM)"
 [double]$min = (Get-date "$getminRange").ToFileTime()

 $getMaxRange = read-host "Specify a end date in mm/dd/yyyy hh:mm:ss AM/PM
format (ex 10/12/2018 12:30:00 PM)"
 [double]$max = (Get-date "$getMaxRange").ToFileTime()
#--------------------End of user and time section-------------------------
}
#####################################################End Functon
Function rip_reg {
$ErrorActionPreference = "silentlycontinue"
#dependencies
#############
#function requires:
# -$userInvest,
# -$myworkingDirectory,
# -$minUTC,
```

```
# -maxUTC
# -myworkingDirectory
#
# be defined
############
#function start
####
##declare variables to work with inside the function.
$mySAMRegWork= $myworkingDirectory + "windows\system32\config\SAM"
$mySYSTEMRegWork= $myworkingDirectory + "windows\system32\config\SYSTEM"
$mySECURITYRegWork= $myworkingDirectory + "windows\system32\config\SECURITY"
$mySOFTWARERegWork= $myworkingDirectory + "windows\system32\config\SOFTWARE"
$myUSERRegWork = $myworkingDirectory+"Users\"+$UserInvest+"\NTUSER.DAT"

#create an Array to sift through that includes all the Registry files. IF one
is missing just add it above and then below
$registryFiles =
$mySAMRegWork,$mySYSTEMRegWork,$mySECURITYRegWork,$mySOFTWARERegWork,$myUSERReg
Work
#name of executable
$myRegripperEXE = "rip.exe"
#location of the executabe *need this for the Change directory command to make
the plugins work
$myRegRipperDir = "C:\Temp\RegRipper2.8-master\"
# combination of both the exe and directory used to execute the command.
$myRegRipperLocation = $myRegRipperDir+$myRegripperEXE
###
cd $myRegRipperDir

Foreach ($registryFile in $registryFiles)
{

##### new code
#create a hashtable to store the information
Write-host "running comparison on $registryFile"
$rip_out = & cmd /c $myRegRipperLocation -r $registryFile -p regtime

#$rip_out.output = & C:\Temp\RegRipper2.8-master\rip.exe -r
"C:\Temp\USAMKBDP3BM72\c\windows\system32\config\SYSTEM" -p regtime

$out2 = @{}
    foreach ($test in $rip_out) {
#testing value
$test
#$test = "Tue Jul 14 04:45:41 2009Z       CMI-CreateHive{0297523D-E529-4E42-
8BE7-E1AABC063C84}\Policy\PolRevision"

#from the testing value pull the Time. Because we are going to replace it with
nothing in the out strig to parse the date.
$stringTime = [regex]::Match($test,'\d\d:\d\d:\d\d').captures.groups[0].value
#parse the outString removing unnecessary information and changing the three
letter
$out = $test -replace "^...\s" -replace "Z|Z" -replace "Jan.","01/" -replace
"Feb.","02/" -replace "Mar.","03/" -replace "Apr.","04/" -replace "May.","05/"
-replace "Jun.","06/" -replace "Jul.","07/" -replace "Aug.","08/" -replace
"Sep.","09/" -replace "Oct.","10/" -replace "Nov.","11/"  -replace "Dec.",
"12/" -replace "\d\d:\d\d:\d\d","/"  -replace "\s/\s","/"
$out

# grab the date from the Test string
$TimeStringDate =
[regex]::Match($out,'^.*?\s\.*?\s*\s').captures.groups[0].value
# because we are looking for date in MM/DD/YYYY,HH:MM:SS we convert both
strings back to the right format. changing one or many spaces to a comma
$TimeconvertString = $TimeStringDate + $stringtime -replace "\s{1,20}","," -
replace "/","/"
#write-host "timeconvertstring"
```

```
$TimeconvertString

#we convert our newly formed string into a datetime and then into filetime to
compare against our Max and min filetimes
# we use the get-date because it looks to work on the ones we care about.
$timeString2 = get-date("$TimeconvertString")
##testing variables
$timeSTring2

#this one worked on a string if we defined it in testing but doesn't work on
the loop"

#$timeString2 = [datetime]::ParseExact($TimeconvertString
,"MM/dd/yyyy,hh:mm:ss",$null)
#convert the date string to filetime to compare with out range.
$timeString2 = $timeString2.ToFileTime()
#
 #       $timeString2
  #      $minUTC
  #      $maxUTC

  #Pause
    #usingan if-and statement we test to see if the time is between
        if (($timeString2 -ge $minUTC) -and ($timeString2 -le $maxUTC)){
            #testing variables
            #Write-Host "out"
            $out
            ########
            # we parse the original test variable for the registry entry and
add this
            #to the regstring column of our hash table
            $out2.regString += $test -replace '^.*?\s\.*?\s*\s'
            #we output the new time format for later use if needed.
            $out2.date += $TimeconvertString
            #
            #$timeString2
            #$minUTC
            #$maxUTC
            #testing variables
            ###
            #write-host "testing varaible"
            #$test
            #Write-Host "out2"
            #$out2
            #Write-host "string-time"
            #$stringTime
            #Write-host "timestring2"
            #$TimeString2
            #Write-Host "TimeStringDate"
            #$TimeStringDate
            #write-host "TimeconvertString"
            #$TimeconvertString
            #pause

#########
 #end of if statement.
            }
 #end of foreach function
 }

 }
## end of test foreach function
return $out2
}
##################################################End Function
#UnZipIT
#$UserInvest= "cusumanov"
```

```
Function getFileDateszip{
###Function to check for files written to the Triage Image that meet the time
frame" Uses the previously
##defined varaibles to open the zip file and list the files that meet. Note
that the first couple of lines always error
## will work on getting it better. note this relies on UnzipIT function running
first as it uses some of the variables.
#########################################################
##create a ordered hash to be used to store data
$datedFile = [ordered]@{}

#check the zip file for the dates in question using the list function of the
command line.
#note: this is how to get around the creation date changing when the file is
unzipped.
#set a variable for the list
$myFiledateCHecker= & $My7Ziplocation l "$MyexandDir\$Mycomputer_name.zip"

#check each entry and parse the date, and file name
foreach ($myFile in $myFiledateCHecker) {
if ($myFile -match '^\d+-\d+-\d+\s+\d+:\d+:\d+') {$datedFile.Date =
$Matches[0]}
#not sure if we need this but kept to make sure
#if ($myfile -match 'C\\.*') {$datedFile.Fullname = $Matches[0]}
#set a test variable to the date we just got
$test = $datedFile.Date
#convert it so we can do some date math against it
$test = (get-date $test).toFileTime()
#test if it matches the min and max we set before.
if (($test -ge $min) -and ($test -le $max)) { $myfile}

}
### end of this function#################################################
}

#########################################################################
## registry load section
#########################################################################
# this section will load the registry files we just got from the remote system.
#
# example of how to load a registry file. in this case we use the windows
navite command reg load
# to set the file up then we use a new-PSDrive to allow access.
# Syntax:
#
#    reg load "hklm\brownda" "c:\temp\c\Users\brownda\NTUSER.DAT"
#    New-PSDrive <name> -PSProvider Registry -Root hklm\brownda
###
Function load_registry{
Write-host "loading registry files using 'reg load'"
#load the registry file for the user we are investigating and..
reg load HKLM\$UserInvest
"C:\Temp\$myComputer_name\C\users\$UserInvest\NTUSER.DAT"
# then use new-psdrive to map it to its name
New-PSDrive -Name $userInvest  -PSProvider Registry -Root HKLM\$UserInvest
#load the other Registry files
#set a variable to allow loads of the different registry files
$registry_load = $myComputer_name +"_Sam"
#load the registry file
reg load  HKLM\$registry_load
"C:\temp\$myComputer_name\C\Windows\System32\config\SAM"
#create a PSDrive
New-PSDrive -name $registry_load -PSProvider Registry -Root HKLM\$registry_load
#reset the value to system
$registry_load = $myComputer_name +"_System"
#load the registry file
```

```
reg load  HKLM\$registry_load
"C:\temp\$myComputer_name\C\Windows\System32\config\SYSTEM"
# Create a PsDrive
New-PSDrive -name $registry_load -PSProvider Registry -Root HKLM\$registry_load
#


write-host "beginning to look for files that were touched between the date."


}
################################################################################
##################################
###note this section needs to be run after the Zip file has been expanded in
order to grab the right information
Write-host "here are the users as defined in the triage images"
#use the variable where we unzipped the triage image($myworkingDirectory) and
read c\User selecting only
#the name and using format-wide to display
#note: $myworkingDirectory is set in unZip Section.
################################################################################
##################################

##capture
 TRiageImage
MEMtriage
# expand
 UnZipIT
 ####
 #query time to look for.
 query_time
 $MyexandDir = "c:\temp"
 $myworkingDirectory = "$MyexandDir\$Mycomputer_name\C\"
$myworkingDirectory

 ## collect
 $TRGFirewwallRules = ""
 $TRGFirewwallRules = FirewallRules
 $TRGRegistry =""
 #testing

 ######################
 Get-ChildItem $myworkingDirectory\Users |format-wide -property Name
 ######################
 #Get-ChildItem $myworkngDir\Users |format-wide -property Name
 #we use this to gather the NTuser.DAT file
 $UserInvest = Read-Host "From the list, which user are we investigating?"
 #now we have the user, we can run the Rip_reg function and assign the variable
to TRGRegistry
 $TRGRegistry = rip_reg
 #now we export the values from $TRGRegistry to a CSV using add-content.
note:future, fix export-csv to work with  this.
 #clear the screen
 Clear
 #tell the user what we are doing
 Write-host "writing the found registry entries to
C:\Temp\registryTimeline.csv"
 #write the values to a CSV
 $TRGRegistry|ForEach-Object{Add-Content -Path C:\temp\registryTimeline.csv -
Value $_}

 getFileDatesZip
```