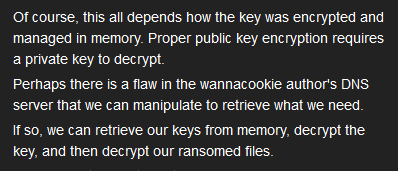
Objective--Recover Alabaster’s Password (Part 3)

# Searching for a solution in the memory dump

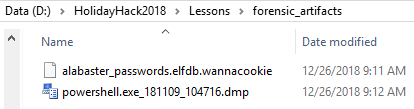
Code analysis has taught us that we need a key ($Byte\_key) in order to decrypt Alabaster’s files that WannaCookie encrypted with AES. However, that key was deleted from memory. The malware encrypted the key using the server’s public key and sent it to the server. The code did not clear/erase the encrypted version of the key from memory. If we can find the encrypted version of $Byte\_key, $Pub\_key\_encrypted\_Key, and the companion private key to the server’s public key we can recover $Byte\_key. This is what Shinny Upatree is telling us to do.  


A close up of a logo

Description automatically generated

Although the malware deleted the key we need ($Byte\_key), it encrypted it with the server’s public key and sent it to the server. Since the server has the private key that matches the public key, it can decrypt $Byte\_key and save it for safekeeping. Farther along in the code (line 245, then 235), you can see where the server will return the unencrypted key to the malware once the ransom has been paid.

# Alabaster’s zip file

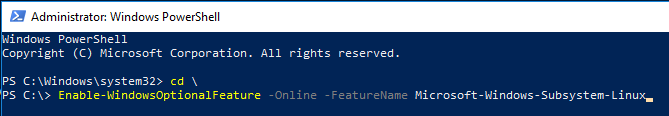
Once we download the zip file from Alabaster, we see that it contains the encrypted version of his password database (alabaster\_passwords.elfdb.wannacookie) and the dump of the memory from the WannaCookie process on his computer (powershell.exe\_181109\_104716.dmp).  


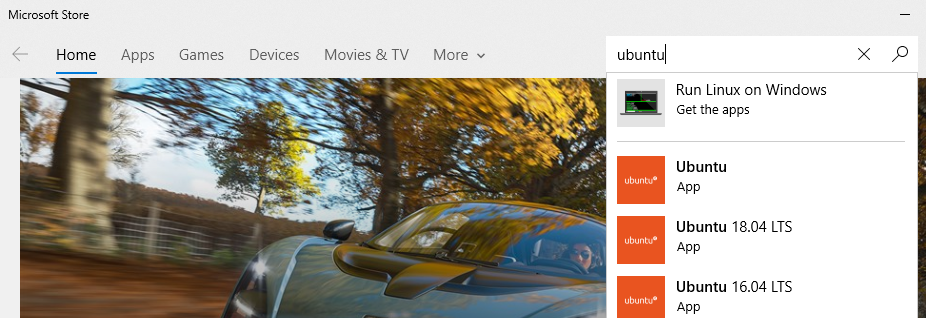
Sure enough, the encrypted database had an elfdb extension and WannaCookie appended its extension.

Chris Davis’ powerdump.py script works fine in an Ubuntu VM. [In the talk](https://www.youtube.com/watch?v=wd12XRq2DNk) he uses Windows 10 and the Windows Subsystem for Linux (WSL). It is really nice to switch back and forth between Windows and Linux command shells in Windows but be careful. In a recent Sacred Cash Cow Tipping Contest (2017?) at Black Hills Information Security, they escaped antivirus detection by jumping to WSL and executing malware there. I see no problems with using WSL on a protected machine, though.

# Installing Linux on Windows 10

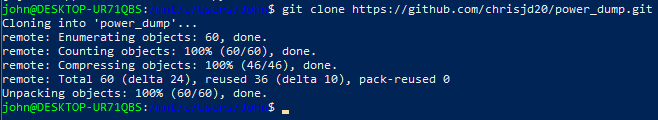
This [link is to an article by Microsoft](https://docs.microsoft.com/en-us/windows/wsl/install-win10) with instructions on installing Linux on Windows 10, or WSL as Microsoft calls it. You have the choice of several different distributions. For this lab I chose Ubuntu 18.04 LTS.

The first step is to execute a PowerShell command as Administrator.  


Then you go to the Microsoft Store and choose your version. Even Kali is available.  


There are more steps after that, but they are not difficult and are well documented.

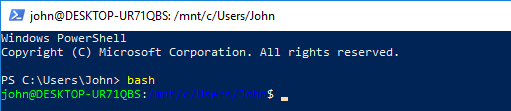
# Installing Power\_dump

This is a [link to the Git Hub repository](https://github.com/chrisjd20/power_dump) for Chris’ software. The installation is easy.  
git clone https://github.com/chrisjd20/power\_dump.git  


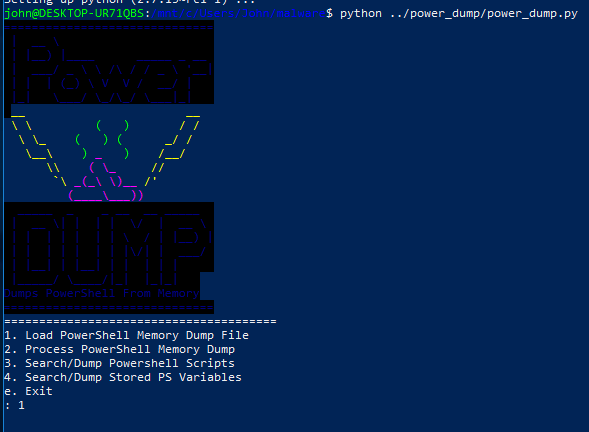
The default installation of Python on the distribution I used is python3, and power\_dump.py did not run well for me in python3. We can add version 2 of Python easily.  
sudo apt install python

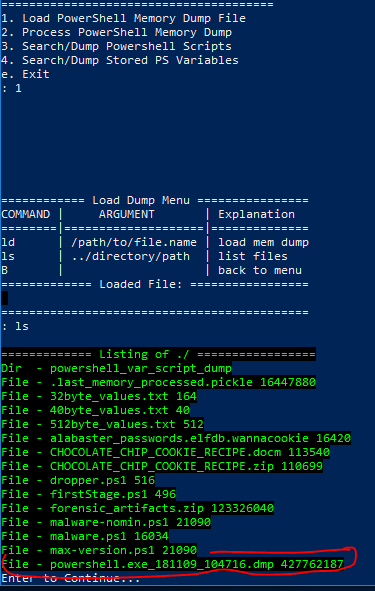
Before we run power\_dump.py, let’s recall the variable table we made before. We will need to know the content type and length of the variables we are searching for. We can find hex strings (or strings of hex) but we won’t be able to find byte arrays (or binary) with this tool.

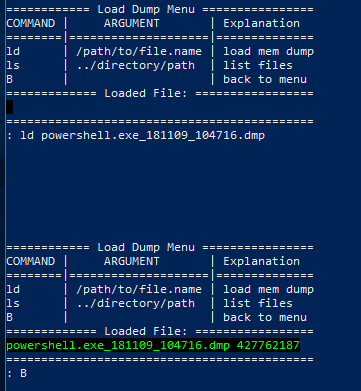
|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **type** | **length** | **purpose** |
| $pub\_key | byte array | 865 | server's public key |
| $Byte\_key | byte array | 16 | AES key for encrypting files |
| $Hex\_key | hex | 32 | $Byte\_key converted to hex |
| $Key\_Hash | string of hex | 40 | SHA-1 of $Byte\_key AES key |
| $Pub\_key\_encrypted\_Key | string of hex | 512 | $b\_k key encrypted with server's public cert |
| $cookie\_id | array of strings | 16 | $cookie\_id[15] is a string len 20, the rest are empty |
| $date\_time | string | 39 | date and time |
| $future\_cookies | array of strings | variable | file paths to be encrypted, all \*.elfdb files |

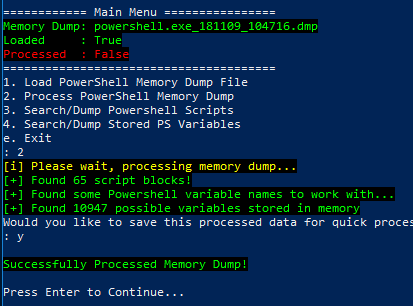
We drop into BASH from PowerShell, as Chris did.  


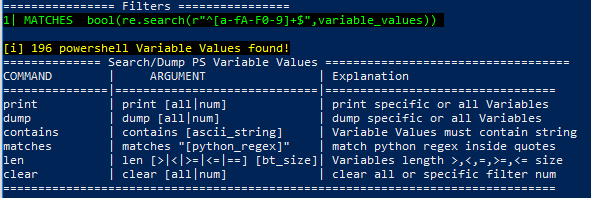
Then we start Power\_dump. On my machine, power\_dump.py is in ~/power\_dump/, and my malware is in ~/malware. With the current working directory set to ~/malware, we can start Power\_dump with   
python ../power\_dump/power\_dump.py  

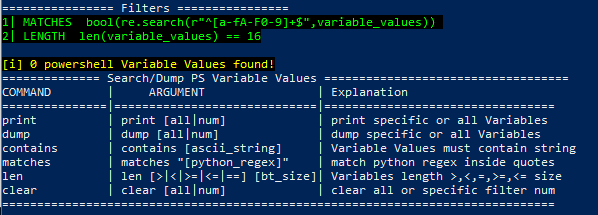

Now we are ready to start. From here on the procedure follows Chris’ talk very closely.  


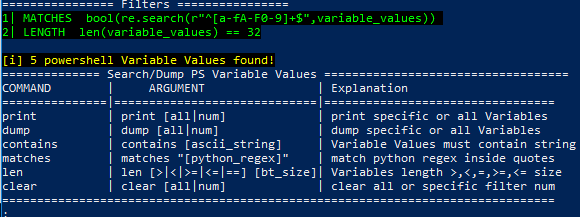
Just to check, we make sure that we are in the correct directory and the dump is available.  


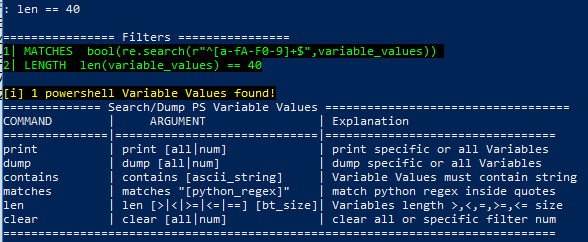
We load the dump file that Alabaster gave us.  


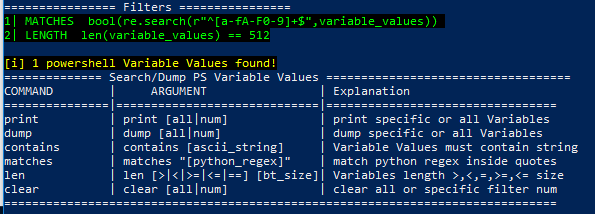
We process it and save the processed version.  


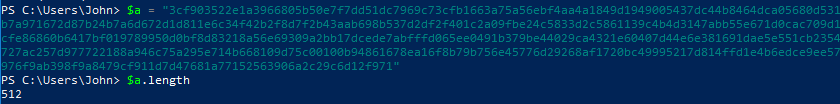
First search was with the hex string regex that Chris used. It finds 196 possible values.  
matches “^[a-fA-F0-9]+$”  


Narrow the field by adding a search for length 16. Only $cookie\_id could match here. The type of $Byte\_key is byte array (binary), so strings won’t find it. We found nothing at all.  


Next up is len == 32. Before we can enter that, we have to clear the old len==16 line.  
clear 2 The length filter is number 2 in the screenshot; if we forget to clear it we will be looking for len == 16 and len == 32 at the same time.  
This could match $Hex\_key, but that variable was cleared. We do find five strings that match; dump them and save to 32byte\_alues.txt. Note: Remember to change the file name so that the next search does not overwrite it.  


A search for length 40 finds one string. Chances are, that is the SHA-1 hash of the key, $Key\_Hash. It may not help us but save it as 40byte-values.txt.  


Finally, a search for len == 512 finds one string. Most likely it is the encrypted version of our key, $Pub\_key\_encrypted\_Key. This is what we were looking for. I saved it as 512byte-values.txt

Back in PowerShell we find that the string is indeed 512 bytes long. 

We have the encrypted key now, although we can only prove that by decrypting it to get the key.

# Hand in

We have the encrypted key, $Pub\_key\_encrypted\_Key, from memory. If we can find the server’s private key, we can decrypt it. One line in Shinny Upatree’s discussion may be critical, *“Perhaps there is a flaw in the wannacookie author's DNS server that we can manipulate to retrieve what we need.”*

1. Get the server, erhaetfanu.com, to give you the private key. The line from the malware that grabbed the public key may prove helpful.