IT 360 Written Report

By: Maya Gopinathan and Matt Brendel

Introduction:

Our tool is a Windows PowerShell script that gathers system-level information (hardware information, running processes, I/O devices, and system logs) and outputs it to a text file which is then encrypted using AES-256. Furthermore, the tool provides a decryption mode to decrypt the protected output file.

The issue we solved with our tool is inefficient forensic data gathering. With our tool, users just have to run the script to gather forensics information. Users don't need to have in-depth knowledge about PowerShell to gather data on Windows systems.

Technical Implementation:

We used PowerShell to code our tool. Our tool is intended for Windows devices, so PowerShell seemed like the best choice of language. Since PowerShell does not require any external binaries to run, this makes it the perfect language to use for creating forensics tools because it has a very low risk of contaminating evidence.

We used AES-256 to encrypt the file because it's widely used and extremely secure.

PBKDF2 with 100,000 iterations was used to derive the AES encryption key because it provides resistance against brute-force and dictionary attacks.

A 16-byte secure random salt was generated for each encryption operation to ensure that identical passwords do not produce identical keys.

CryptoStream was used to stream data through the AES encryptor/decryptor, allowing the tool to handle large files efficiently.

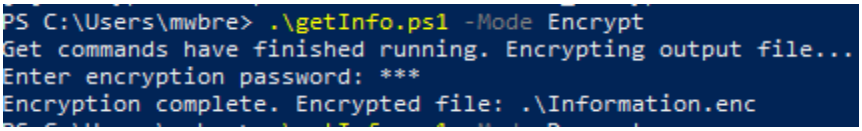
Error handling was used by implementing try/catch blocks to detect incorrect passwords, corrupted files, and read/write issues. If decryption fails, the tool deletes any partially written plaintext to prevent data leakage.

We only used "Get" commands when dealing with data on the Windows system. This was so the data would remain unaltered, which is important for the forensic process.

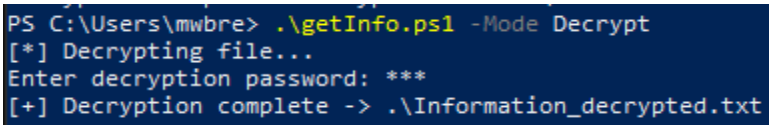
After converting the SecureString password to plaintext, the memory buffer was wiped to prevent password remnants from remaining in RAM.

Results:

Our tool results in an encrypted or decrypted text file that contains information about the Windows system's hardware information, running processes, I/O devices, and system logs.

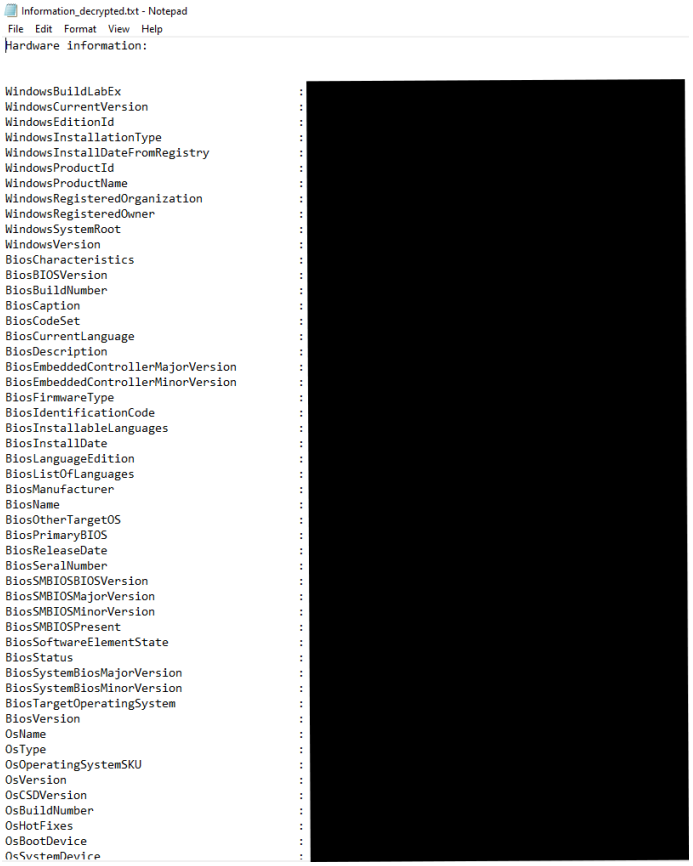


Using the "Encrypt" mode, the script outputs "Information.enc" which cannot be viewed or opened because it is encrypted. This mode's average run-time is around 25 seconds.



Using the "Decrypt" mode, the script decrypts "Information.enc" after the user inputs the correct password. The original encrypted file is deleted from the system, and "Information_decrypted.txt" is now available. This mode takes around three seconds to run, so it is much faster than the encrypt mode.

Hardware information section of Information_decrypted.txt:



Running processes section of Information_decrypted.txt:

```
Running Processes:
Adobe Desktop Service 6.8.0.821
AdobeIPCBroker 7.3.0.41
AdobeNotificationClient 6.0.0.1
```

I/O Devices section of Information_decrypted.txt:

```
I/O Devices:

Status     Class          FriendlyName                                                  InstanceId
------     -----          ------------                                                  ----------
OK         Volume         Volume                                                        STORAGE\...
OK         USB            USB Root Hub (USB 3.0)                                        USB\ROOT...
OK         HIDClass       Logitech G HUB Virtual Keyboard                               LGHUBDEV...
OK         System         Motherboard resources                                         ACPI\PNP...
OK         System         PCI standard host CPU bridge                                  PCI\VEN_...
OK         HDC            Standard SATA AHCI Controller                                 PCI\VEN_...
OK         Ports          Printer Port (LPT1)                                           ACPI\PNP...
OK         AudioEndpoint  Headset Earphone (Wireless Controller)                        SWD\MMDE...
```

System Logs section of Information_decrypted.txt:

```
System Logs:

   Index Time          EntryType   Source                 InstanceID Message
   ----- ----          ---------   ------                 ---------- -------
   91397 Dec 02 12:57  Information
   91396 Dec 02 12:51  Warning
   91395 Dec 02 12:44  Warning
   91394 Dec 02 12:00  Information
   91393 Dec 02 11:30  Information
   91392 Dec 02 11:29  Information
   91391 Dec 02 11:29  Information
   91390 Dec 02 11:29  Information
```

Lessons Learned & Conclusion:

<u>What Worked and What Didn't:</u>

We originally had the get commands output to a CSV file. After the first get command was run, we ran into an error where process objects didn't have the same properties as the computerInfo objects, so it wasn't able to append properly.

```
Export-Csv : Cannot append CSV content to the following file: .\Information.csv. The appended object does not have a property that corresponds to the following column: WindowsBuildLabEx. To
continue with mismatched properties, add the -Force parameter, and then retry the command.
At line:3 char:32
+ ... rsionInfo | Export-Csv -Path ".\Information.csv" -NoTypeInformation - ...
+                 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidData: (WindowsBuildLabEx:String) [Export-Csv], InvalidOperationException
    + FullyQualifiedErrorId : CannotAppendCsvWithMismatchedPropertyNames,Microsoft.PowerShell.Commands.ExportCsvCommand
```

Rather than using the –Force parameter and continuing with mismatched properties in the CSV file, we switched to using a text file instead.

Originally, we didn't use try-catch blocks for any of the commands. We then ran into an error with the Get-Process command as some processes didn't have file version information.

```
Get-Process : Cannot enumerate the file version information of the "WUDFHost" process.
At line:5 char:1
+ Get-Process -FileVersionInfo | Out-File -FilePath .\Information.txt - ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : PermissionDenied: (System.Diagnostics.Process (WUDFHost):Process) [Get-Process], ProcessCommandException
    + FullyQualifiedErrorId : CouldnotEnumerateFileVer,Microsoft.PowerShell.Commands.GetProcessCommand

Get-Process : Cannot enumerate the file version information of the "WUDFHost" process.
At line:5 char:1
+ Get-Process -FileVersionInfo | Out-File -FilePath .\Information.txt - ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : PermissionDenied: (System.Diagnostics.Process (WUDFHost):Process) [Get-Process], ProcessCommandException
    + FullyQualifiedErrorId : CouldnotEnumerateFileVer,Microsoft.PowerShell.Commands.GetProcessCommand

Get-Process : Cannot enumerate the file version information of the "WUDFHost" process.
At line:5 char:1
+ Get-Process -FileVersionInfo | Out-File -FilePath .\Information.txt - ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : PermissionDenied: (System.Diagnostics.Process (WUDFHost):Process) [Get-Process], ProcessCommandException
    + FullyQualifiedErrorId : CouldnotEnumerateFileVer,Microsoft.PowerShell.Commands.GetProcessCommand
```

After implementing a try-catch block for the Get-Process command, we then ran into an error where the process name wouldn't print when the command ran.

```
ProductVersion   FileVersion      FileName
--------------   -----------      --------
10,01,01,2052    10,01,01,2052    C:\Program Files\WindowsApps\AdvancedMicroDevicesInc-2.AMDRadeonSoftware_10.24.30026.0_x64__0a9344xs7nr4m\radeonsoftware\amdow.exe
 has no version information
25.5.27.0        25.5.27.0        C:\Windows\System32\amdpmfserviceuser.exe
 has no version information
10,01,01,2052    10,01,01,2052    C:\Program Files\WindowsApps\AdvancedMicroDevicesInc-2.AMDRadeonSoftware_10.24.30026.0_x64__0a9344xs7nr4m\radeonsoftware\AMDRSServ.exe
10,01,01,2052    10,01,01,2052    C:\Program Files\WindowsApps\AdvancedMicroDevicesInc-2.AMDRadeonSoftware_10.24.30026.0_x64__0a9344xs7nr4m\radeonsoftware\AMDRSSrcExt.exe
                                  C:\Windows\SystemApps\MicrosoftWindows.Client.CBS_cw5n1h2txyewy\AppActions.exe
10.0.26100.7019  10.0.26100.70... C:\Windows\system32\ApplicationFrameHost.exe
 has no version information
16.0.19328.20010 16.0.19328.20010 C:\Program Files\Common Files\Microsoft Shared\ClickToRun\AppVShNotify.exe
 has no version information
 has no version information
10.0.26100.1     10.0.26100.1 ... C:\Windows\system32\backgroundTaskHost.exe
 has no version information
 has no version information
142.0.7444.163   142.0.7444.163   C:\Program Files\Google\Chrome\Application\chrome.exe
142.0.7444.163   142.0.7444.163   C:\Program Files\Google\Chrome\Application\chrome.exe
```

After some trial and error, we figured out how to get the command to work. For each process, we checked if there was a path. If so, we used that to retrieve the version information which was then output to the file. If there wasn't a path, we added the process name to the file with a note that there wasn't an executable path.

Future Additions:

Deeper forensic coverage could be added to the tool. This could include memory capture, browser artifacts, and the network state at the time of collection.

Hashes could be generated and stored for the final encrypted file for integrity verification. Furthermore, digital signatures could be used to prove that the evidence came from this tool.

Automatically log the user of the tool and what system it was running on into a signed log file.