Lateral Movement — SCM and DLL Hijacking Primer

Dwight Hohnstein · Follow
Published in Posts By SpecterOps Team Members · 9 min read · Apr 18, 2019

Summary

As Defenders increase in maturity, the more they are able to leverage built-in utilities against attackers. PowerShell has a tremendous amount of verbose logging and introspection, service creation and manipulation is often flagged, and permanent WMI event subscriptions are caught by both Windows EventID 5861 and Sysmon. Dynamic Link Library (DLL) hijacking has been used traditionally for persistence, privilege escalation, and execution. In this article I'll examine two DLL hijacks that occur on most versions of Windows for the purpose of lateral movement, including discovery methodology, detections, and example code.

. . .

Detailed Description

The Service Control Manager (SCM) governs all aspects of running services installed on a Windows Computer. Historically, the Service Control Manager has been abused by attackers to escalate their privilege locally on a machine or to create new services on target machines for persistence or lateral movement.

Instead of creating new services, attackers can move laterally using the SCM by copying specifically crafted Dynamic Link Library (DLL) files to trusted directories and restarting services remotely. This is possible because these services call LoadLibrary on libraries not present in the specified path. This allows an attacker to place their crafted library in the trusted directories Windows attempts to load the library from and leverage the service to move laterally.

Two such services can be leveraged by attackers, the IKEEXT and SessionEnv service, as they call LoadLibrary on files that do not exist within C:\Windows\System32\ by default. An attacker can place their malicious logic within the PROCESS_ATTACH block of their library and restart the aforementioned services to gain code execution on a remote machine. The IKEEXT attempts to load C:\Windows\System32\wlbsctrl.dll when started, while the SessionEnv service attempts to load TSMSISrv.dll and TSVIPSrv.dll from the System32 directory.

. . .

Proof of Concept Code

https://github.com/djhohnstein/wlbsctrl_poc

https://github.com/djhohnstein/TSMSISrv_poc

. . .

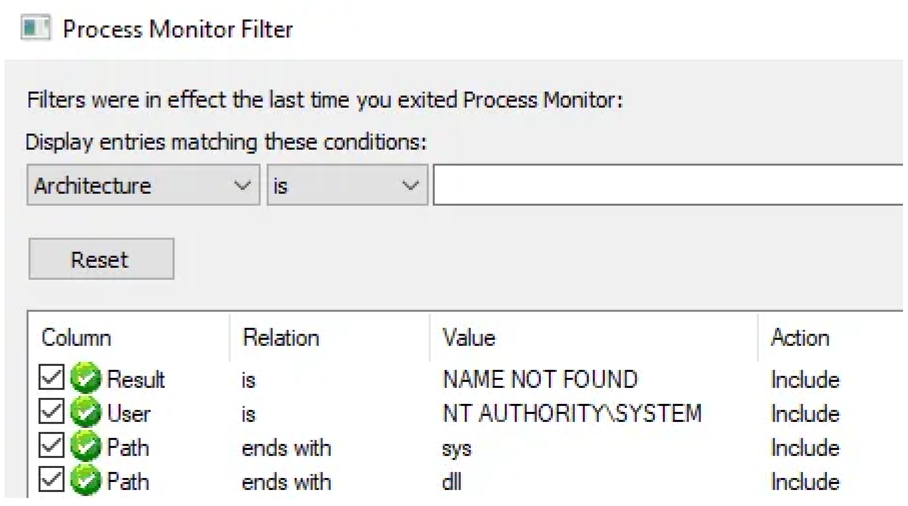
Methodology

To find services that contained DLL hijacks, I first deployed the most common versions of Windows that are still under support: Windows 7, Windows 10, Server 2008, Server 2012 and Server 2016.

Next, I gathered a list of all services installed on these machines by default using the Get-Service command-let. Taking the intersect of these data sets, a list of roughly 90 common services was generated.

With the initial service information compiled, I installed Procmon and applied a filter that searches for:

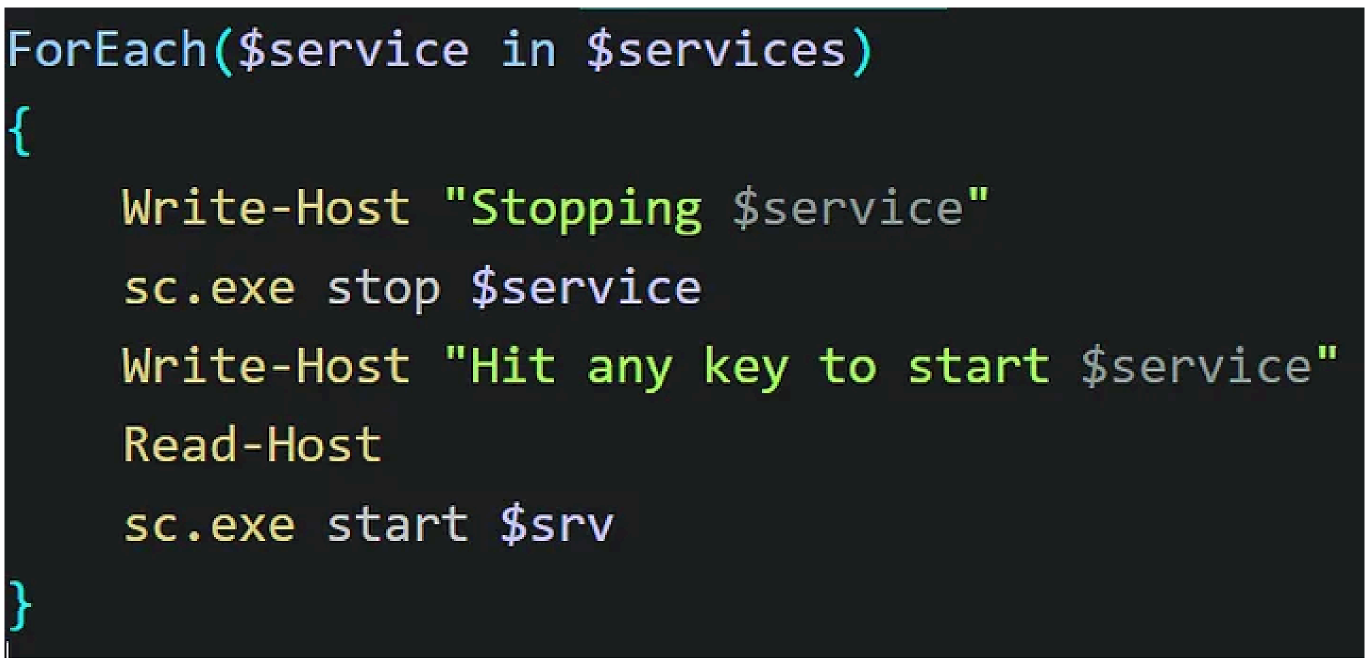
1. The result is "NAME NOT FOUND".
2. The path ends in "sys" or "dll".
3. The user or integrity level is SYSTEM.


Process Monitor Filter
Filters were in effect the last time you exited Process Monitor:
Display entries matching these conditions:
Architecture
is
Reset

Column	Relation	Value	Action
✓ Result	is	NAME NOT FOUND	Include
✓ User	is	NT AUTHORITY\SYSTEM	Include
✓ Path	ends with	sys	Include
✓ Path	ends with	dll	Include

The Process Monitor filter used.

To monitor these services in a fine-grained manner, I needed some degree of control over their start and stop times in order to correlate the information I saw in Procmon with the service being started. My solution was to use a simple PowerShell for-loop such that I could clear my event log between each service restart, and trigger each restart with a keystroke.



```
ForEach($service in $services)
{
    Write-Host "Stopping $service"
    sc.exe stop $service
    Write-Host "Hit any key to start $service"
    Read-Host
    sc.exe start $srv
}
```

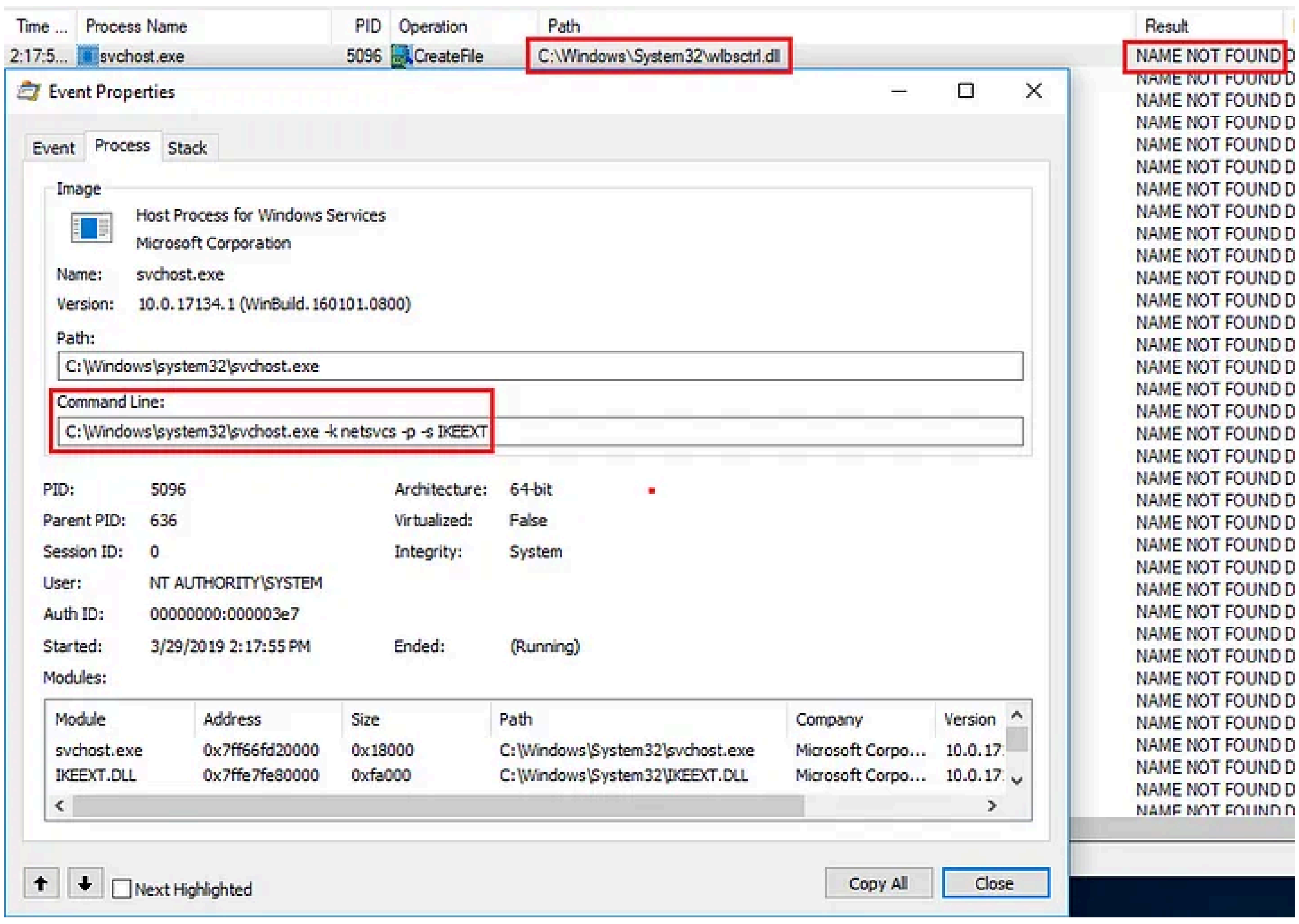
Simple PowerShell script to stop and start services. Gist can be found here.

During this process of stopping services, clearing Process Monitor logs, and restarting the services, I documented each service that attempted to search for a DLL that wasn't present on disk. Using this new data set of potentially vulnerable services across the Windows product line, two met the criteria I set when starting this endeavor: IKEEXT and SessionEnv.

. . .

IKEEXT Analysis and Exploitation

IKEEXT hosts the Internet Key Exchange (IKE) and Authenticated Internet Protocol (AuthIP) keying modules. When the service is started, it searches for the file wlbsctrl.dll. This is the first indicator that the service is potentially abusable.



```
Time ...   Process Name   PID   Operation    Path                               Result
2:17:5...  svchost.exe    5096  CreateFile   C:\Windows\System32\wlbsctrl.dll   NAME NOT FOUND
```

```
Event Properties

Event   Process   Stack

Image
    Host Process for Windows Services
    Microsoft Corporation
Name:      svchost.exe
Version:   10.0.17134.1 (WinBuild.160101.0800)
Path:
C:\Windows\system32\svchost.exe
Command Line:
C:\Windows\system32\svchost.exe -k netsvcs -p -s IKEEXT

PID:         5096                     Architecture:   64-bit
Parent PID:  636                      Virtualized:    False
Session ID:  0                        Integrity:      System
User:        NT AUTHORITY\SYSTEM
Auth ID:     00000000:000003e7
Started:     3/29/2019 2:17:55 PM     Ended:          (Running)
Modules:

Module        Address          Size      Path                              Company          Version
svchost.exe   0x7ff66fd20000   0x18000   C:\Windows\System32\svchost.exe   Microsoft Corpo...  10.0.17
IKEEXT.DLL    0x7ffe7fe80000   0xfa000   C:\Windows\System32\IKEEXT.DLL    Microsoft Corpo...  10.0.17

Next Highlighted                                              Copy All   Close
```

svchost.exe starts the IKEEXT service, which then queries for the wlbsctrl.dll file.

Viewing the stack tab of the event properties we can see exactly how svchost.exe was called and what files may be attempting to call LoadLibrary on the wlbsctrl.dll file. Doing so reveals that IKEEXT.DLL is directly above svchost.exe in the stack frame and is a perfect candidate for further analysis.

IKEEXT.DLL appears to be responsible for the svchost.exe kickoff, indicating it may be responsible for the LoadLibrary call.

I then threw this file into a Ghidra to begin analyzing the IKEEXT.dll. The first place one looks when searching for these LoadLibrary calls is in the PE's import table. This table defines function dependencies in other Portable Executables (PEs) on disk. Unfortunately the wlbsctrl.dll was not referenced in the import table, and upon reflection this is to be expected. The reason being is that if it were referenced, we should have expected to see svchost.exe searching not only C:\Windows\System32\, but every directory in the PATH environment variable as well (defined in the load library specification here).

The wlbsctrl.dll is not specified by IKEEXT.DLL's expanded import table.

At this point, two potential next steps are searching for all references to the LoadLibrary call or search the PE for the “wlbsctrl.dll” string. I went with the latter as it likely would have fewer results, and luckily enough, only one hit was returned.

Searching for the wlbsctrl.dll string

When clicking the search result from above I’m navigated to the data segment that defines the string. Right clicking this address and select References > Show References to Address, I’m pointed to a singular function at 0x180005ea0. Jumping to this function we see that after variable declarations, the first function call is to LoadLibraryExW with a path-relative reference to wlbsctrl.dll. From our Process Monitor logs above, we know that this function is called at some point during service startup. As such, no more analysis is needed to build a working proof of concept.

The function calls LoadLibraryExW as its first function call.

To leverage this service, simply place the crafted DLL that performs actions on PROCESS_ATTACH in the same folder as IKEEXT.dll (C:\Windows\System32\ by default). Then, use the service control manager binary (sc.exe) to restart the service.

Example command set to leverage the DLL hijack in IKEEXT.

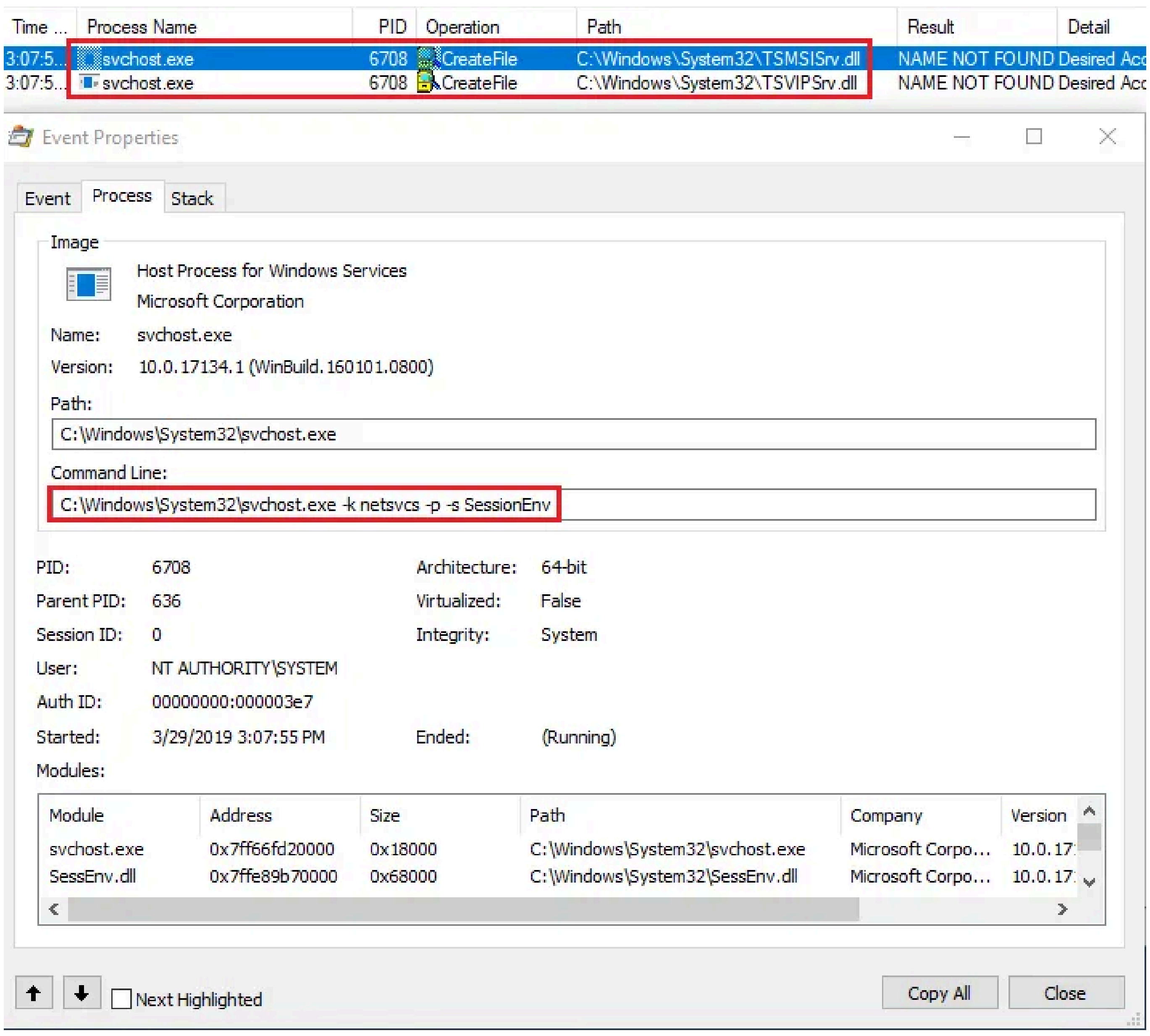
Video demo of leveraging the IKEEXT service remotely to add a new user called "demo" to DC01

- Editor's note: It was brought to my attention @FuzzySec discussed the same hijack in his article here. I hope this article provides guidance and context in addition to, not instead of, his excellent work.

. . .

SessionEnv Analysis and Exploitation

Applying the same methodology from IKEEXT to the SessionEnv service, we are greeted with two DLLs being queried by svchost.exe: TSMSISrv.dll and TSVIPSrv.dll.



```
Time ...  Process Name   PID   Operation   Path                                Result          Detail
3:07:5... svchost.exe    6708  CreateFile  C:\Windows\System32\TSMSISrv.dll    NAME NOT FOUND  Desired Acc
3:07:5... svchost.exe    6708  CreateFile  C:\Windows\System32\TSVIPSrv.dll    NAME NOT FOUND  Desired Acc
```

```
Event Properties
Event | Process | Stack

Image
    Host Process for Windows Services
    Microsoft Corporation
Name:     svchost.exe
Version:  10.0.17134.1 (WinBuild.160101.0800)
Path:
C:\Windows\System32\svchost.exe
Command Line:
C:\Windows\System32\svchost.exe -k netsvcs -p -s SessionEnv

PID:         6708                       Architecture:  64-bit
Parent PID:  636                        Virtualized:   False
Session ID:  0                          Integrity:     System
User:        NT AUTHORITY\SYSTEM
Auth ID:     00000000:000003e7
Started:     3/29/2019 3:07:55 PM       Ended:         (Running)
Modules:

Module        Address          Size     Path                               Company            Version
svchost.exe   0x7ff66fd20000   0x18000  C:\Windows\System32\svchost.exe    Microsoft Corpo...  10.0.17
SessEnv.dll   0x7ffe89b70000   0x68000  C:\Windows\System32\SessEnv.dll    Microsoft Corpo...  10.0.17

Next Highlighted                                              Copy All   Close
```

Process Monitor showing svchost.exe requesting the aforementioned files.

Once again, these files aren't searched for recursively using the DLL search order specification; rather, they must be called directly by name somewhere within the calling library, SessEnv.dll. Using the string reference method as above we are dropped into a singular function that references both TSMSISrv.dll and TSVIPSrv.dll by name.

The only function that references the affected files by name.

Highlighted above is this uVar6 variable, which is most likely the return code of the function I've labeled "FunctionRequiringDLL1". Notably this function is called on each of the suspected vulnerable DLL hijack locations. At this point, we know nothing about this function other than it takes four arguments, one of which is a DLL.

The decompiled version of FunctionRequiringDLL1.

Analyzing the decompiled function call, it's clear that the function performs the following:

1. Creates a buffer of size 264 to hold the file path.
2. Expands the buffer passed in the second argument of the function to be populated with the environment variable, which in our case is %SYSTEMROOT%.
3. Calls LoadLibraryExW on the result, so long as the function call succeeds in populating the buffer.

Thus, to leverage this service to execute code, we need to place a DLL in %SYSTEMROOT%\System32\ with a name of TSMSISrv.dll or TSVIPSrv.dll that performs our actions on PROCESS_ATTACH. Then use the Service Control Manager to stop and start the service to have the service load your crafted DLL.

Example commands to leverage the service remotely.

Video demo of leveraging the SessionEnv service remotely to add a new user called "demo" to DC01

Operational Caveat

One caveat of this technique is that at times, other processes besides the process started by the service may occasionally load your planted DLL. If this happens you **will not** be able to delete your file. You can attempt to call a remote free on the library, but this will crash the remote process. You could attempt to swap the contents out in memory and replace all functions with a call to FreeLibrary, but this will also crash the process. The only way to remove the file is to move it to another location such as the APPDATA folder and restart the machine. After restart, no process should have a handle on the file and it should be removable.

. . .

Host-based Mitigations and Detections

On Windows hosts Event ID 7036 will trigger each time a service is stopped and started; however, these events are no longer supported in Windows workstations starting from version 8.0. To enable auditing of these services we'll need to configure the Security Access Control List to enable auditing of all service functions. The steps to do so are as follows:

1. Check the current security descriptor of the service using sdshow.
2. Copy the value returned from sdshow and append the "(AU;SAFA;RPWPDTCCLC;;;WD)" ACE.
3. Set the descriptor of the service using sdset and the value created from above.

Setting the additional ACE to the IKEEXT service.

A full breakdown of the ACE and subsequent data sources, including events and descriptions of those events have been provided by Roberto Rodriguez. You can find his detailed detection guidance in the document provided here.

Starting in Windows 10, Microsoft introduced Exploit Guard which allows you to "manage and reduce the attack surface of apps used by your employees." Exploit Guard introduced several new event streams including one for non-Microsoft-signed binary loads. If one were to apply this event (Event ID 11) to svchost.exe, they would be able to extract both the PID and path to the non-Microsoft-signed PE that would be loaded. While this is sure to cause noise at first, tuning this data source could prove to be a potent detection. A full list of Exploit Guard's events can be found here.

Event ID 11 which shows the unsigned payload being loaded into svchost.exe

Another detection source for Windows 10 and up also includes the Audit File Share service, which can be enabled to track file creation events on network shares (Event ID 5140 for share access, 5145 for objects accessed). This is important as the crux of the hijacks detailed here (and many others) require the placing of a DLL inside the trusted System32 directory.

Network Detections

Network-based detections should monitor the negotiation of RPC UUID {367ABB81–9844–35F1-AD32–98F038001003} over port 135. This UUID denotes the Service Control Manager. This opens a new named pipe, SVCCTL, for the client to communicate with over port 445 in order to control the desired service.

Negotiation between endpoints for the Service Control Manager.

The SCM RPC methods invoked for starting and stopping a service are shown below.

The SCM RPC methods invoked when starting a service.

The SCM RPC methods invoked when stopping a service.

Practicing Detections with Mordor

Mordor is an open-source tool created by Roberto Rodriguez and Jose Luis Rodriguez to help aggregate and recreate malicious events in a modular format. Mordor creates recordings of various TTPs by logging each event created during the execution of the attack in a digestible format. This allows defenders to create detections based on both the execution and context surrounding an attack without having to recreate it in a lab every time. Roberto Rodriguez was gracious enough to create a Mordor recording for this method of lateral movement which can be downloaded at this link.

. . .

Conclusion

While I detail only two services in this article, there are several more that exist across the current Windows ecosystem. Moreover, if one wanted to avoid executing their logic within the PROCESS_ATTACH block, they could instead find the first function call referenced from the LoadLibrary call and attach their malicious logic within that function stub. This is useful if you wanted to create a .NET DLL and use the Unmanaged Exports nuget library. The discovery of these additional services and function calls is left as an exercise for the reader.

. . .

Additional Resources

- https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-scmr/705b624a-13de-43cc-b8a2-99573da3635f
- https://liberty-shell.com/sec/2019/03/12/dll-hijacking/
- http://www.fuzzysecurity.com/tutorials/16.html
- https://www.nuget.org/packages/UnmanagedExports
- https://www.crowdstrike.com/blog/in-depth-analysis-of-the-ccleaner-backdoor-stage-2-dropper-and-its-payload/

Microsoft Windows Lateral Movement Dll Hijacking Services

--

Written by Dwight Hohnstein

173 Followers · Writer for Posts By SpecterOps Team Members

IBM X-Force Red Team Operator

Follow