Exploring Rootkit Detectors’ Vulnerabilities Using a New Windows Hidden Driver Based Rootkit

Woei-Jiunn Tsaur  
Department of Information Management  
Da-Yeh University  
Changhua, Taiwan, R.O.C.  
e-mail: wjtsaur@yahoo.com.tw

Yuh-Chen Chen  
Department of Information Management  
Da-Yeh University  
Changhua, Taiwan, R.O.C.

Abstract—More and more malware writers are taking advantage of rootkits to shield their illegal activities. Any computer security products that are not equipped with the anti-rootkit functionality may not identify this kind of threat. Thus, the role of a rootkit detector is becoming extremely important. Though much research has been focused on kernel data to develop schemes for finding malicious behaviors and undoubtedly they can effectively detect hooking based or virtual machine based rootkits in Linux or Windows, they cannot foresee what the result is when meeting unknown Windows DKOM (Direct Kernel Object Manipulation) based rootkits. In this paper, we develop a new Windows driver-hidden rootkit with five tricks based on DKOM, and have verified that it can successfully avoid a variety of well-known rootkit detectors. This paper spots the weaknesses of current detectors, and also discusses possible remedies and solution for detecting the proposed subtle rootkit. We expect that this research will contribute to the development of rootkit detection methods for Windows hidden driver based rootkits.

Keywords—system security; rootkit; malware; Windows; kernel mode

I. INTRODUCTION

In 2005, Sony-BMG used a rootkit to conceal the digital right management [1] software, which is aptly installed in consumers’ computers to prevent unauthorized copying. As a result, it lets the installed rootkit computers provide malware with proper shelters to be capable of escaping anti-virus detection easily. After this event was exposed by Mark Russinovich, RootkitRevealer [2] maker, the term “rootkit” drew the great attention of people around the world.

“Rootkit” represents stealth techniques which can hide information about computer resources furtively, and prevent itself from being discovered by system administrators. Rootkits made its debut at the end of 80’s. Roughly speaking, the evolution of rootkits can be divided into four phases. In the first phase, a rootkit can alter Unix’s log files to hide the presence of certain users. In the second phase, it not only alters log files but also replaces some important programs such as “ls”, “ps” or “netstat” to hide users’ activities. The main drawback of substituting programs is that it can be easily found by integrity checking programs such as Tripwire [3]. To overcome this drawback, the third phase is that rootkits aim to intercept user mode function requests and hide desired information in memory. It is not necessary to modify or replace files in this phase. In order to pursue higher stealth level, rootkits and detectors have transferred to the kernel mode in the fourth phase. The targeted operating system is also changed from the UNIX system to Microsoft Windows system.

Since Sony-BMG used a rootkit and an opening dedicated rootkits forum [4] was created, the topics related to rootkits have been widely discussed and many rootkits targeting Microsoft Windows have been available. In addition, more and more malware writers are integrating rootkits to shield their illegal activities [5]-[8]. Any computer security products that are not equipped with the anti-rootkit functionality may not identify this kind of threat. Thus, the role of a rootkit detector is becoming extremely important. It can be observed that most of the sophisticated kernel mode rootkits are implemented as driver-style programs to execute hiding tasks in Windows. Though much research [9]-[20] has been focused on kernel data to develop schemes for finding malicious behaviors and undoubtedly they can effectively detect hooking based or virtual machine based rootkits in the Linux or Windows system, they cannot foresee what the result is when meeting unknown Windows DKOM (Direct Kernel Object Manipulation) based rootkits. This paper aims to discuss detective measures to guard these avenues of attack by introducing a new set of tricks for implementing a driver-hidden rootkit using DKOM in the Windows system. As King et al.’s research process [12], in this paper we first assume the perspective of the attacker, who is trying to run malware and avoid detection. By assuming this perspective, we then hope to help defenders understand and defend against the threat posed by the proposed new type of driver-hidden rootkit.

To sum up, our contributions are threefold. First, we demonstrate the vulnerability of well-known Windows rootkit detectors by exploring the design and implementation of the proposed driver-hidden rootkit. Second, as stated in the literature [2], [21]-[23], rootkit is a stealth technology, and the intent with which this technology is used determines their malicious or otherwise legitimate purpose. The same technology used by rootkits is also used in security software such as firewalls and host-based intrusion prevention systems to extend the protection
of the operating system. Therefore, the stealth tricks of the proposed sophisticated driver-hidden rootkit can be a great inspiration to defenders who need to effectively strengthen the legitimate uses. Lastly, our efforts will be useful for stimulating detector developers to upgrade their products to enhance their techniques against such a new threat in the Windows system.

The remainder of this paper is organized as follows. Section II surveys current rootkit techniques. Section III presents the DKOM-based method for developing a new Windows driver-hidden rootkit. Section IV depicts the experimental results of testing the proposed rootkit’s stealth ability, and further discusses the reasons the six well-known detectors cannot detect it. Section V gives possible remedies and solution for detecting the proposed subtle rootkit. Finally, we conclude by presenting our plans for future work in Section VI.

II. RELATED WORK

A kernel-mode rootkit can hide computer resources essentially by two different techniques. One is hooking that intercepts the requests of accessing resources. The other is Direct Kernel Object Manipulation (DKOM) that manipulates the data used by operating systems to keep track of resources. NT Rootkit [24], the oldest Windows kernel mode rootkit, used table hooking to alter System Service Descriptor Table (SSDT) to hide processes, drivers, files, etc. Although it is a simple, stable and efficient method, it is easily detected by current rootkit detectors [25]. Hunt and Brubacker [26] introduced Detours, a library for intercepting arbitrary Win32 binary function. Later, this method was also applied by a rootkit, which replaces the first few instructions of a specific function with “jump” to point to the rootkit’s code instead of targeting system tables. The above-mentioned is named inline hooking. But VICE [27], a heuristic detector, is created to detect processes generating hooks no matter table hooking or inline hooking. In order to strengthen inline hooking, sophisticated rootkit writers combine a polymorphic technique [28] whose purpose is to generate different appearances of a piece of code. These appearances may look different but have the same functionality. On the other hand, Butler et al. [29] used DKOM to target EPROCESS, a kind of kernel data structures to record information related to a process, to alter an affiliated doubly linked list, and let the desired processes be hidden. When using DKOM, rootkit writers need to clearly understand the data structures in kernel, but it is more furtive than using hooking [2], [10], [22], [23]. The DKOM technique was first used in the FU rootkit and then used in FUTo to hide their drivers [2], [23]. In 2007, the DKOM-based Unreal rootkit was created and shown off that all of the famous detectors cannot detect it. However, at present several well-known detectors like GMER and Rootkit Unhooker are capable of effectively detecting the above-mentioned three driver-hidden rootkits.

III. PROPOSED NEW WINDOWS DRIVER-HIDDEN ROOTKIT

In this section, we will introduce unknown DKOM-based rootkit creation techniques which may be taken into consideration by rootkit detection researchers and be studied by the security community, in order to help defenders strengthen their detection techniques for identifying a wide variety of unknown Windows driver-hidden rootkits. In this paper, we use the DKOM technique to create a new driver-hidden rootkit with two reasons. One reason is that all of loaded drivers are stored into memory as data structure formats and DKOM specializes in altering data structure values. The other is that detecting DKOM is more difficult than detecting hooking [2], [10], [22], [23].

The following is to illustrate DKOM with an example. List_Entry data structure is employed by Microsoft Windows to maintain a string of the same kind of objects, such as Object Driver, Object EPROCESS, etc. For example, Fig. 1 shows that there are three objects named A, B and C respectively. Each one has a List_Entry data structure. A.FLINK points to B, B.FLINK points to C, C.BLINK points to B and B.BLINK points to A. If we want to hide B, we should let A.FLINK point to C and C.BLINK point to A, as shown in Fig. 2. This skill is straightforward but useful, and the operation process for DKOM is depicted in Algorithm 1.

Algorithm 1

<table>
<thead>
<tr>
<th>DKOM(Target_Object)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 {</td>
</tr>
<tr>
<td>2 (Target_Object -&gt; Blink) -&gt; Flink = Target_Object -&gt; Flink</td>
</tr>
<tr>
<td>3 (Target_Object -&gt; Flink) -&gt; Blink = Target_Object -&gt; Blink</td>
</tr>
<tr>
<td>4 }</td>
</tr>
</tbody>
</table>

Figure 1. An original doubly linked list to record the same kind of objects.

Figure 2. A doubly linked list after being modified to hide Object B.
Our contribution in this section does not highlight how to use DKOM to alter objects, but we disclose five places, some of which may not be known by anti-rootkit developers, to hide driver information. The proposed new Windows driver-hidden rootkit is composed of five tricks which will be presented in the following Items A-E, respectively.

A. Removing Object Drivers and Object Devices from Object Directory

In the internal of Windows kernel, the crucial part is objects. They contain all kinds of resources that are queried by kernel functions. The program of Windows object management is responsible to manage objects. All of objects are kept in a tree of Object Directory. As shown in Fig. 3, the Object Directory is established with 37 HashBuckets. Each one points to an Object Directory Entry whose Object member refers to an object and ChainLink member points to another Object Directory Entry. Most of Object Drivers have at least one Object Device pointing to themselves, so Object Drivers and Object Devices are needed to consider when a driver-hidden rootkit is created. We can explore the whole Object Directory to find the desired Object Drivers and Object Devices, and then apply DKOM to achieve the purpose of hiding. In the following, we first define notations, and then the operation process for removing Object Drivers and Object Devices from Object Directory is depicted in Algorithm 2.

![Figure 3. Relationship between an Object Directory and its member.](image)

**Notations definition**

- pObjectDirectory: a pointer to the Object Directory
- Target: an array storing target drivers’ names to hide
- FindObject(): a function finding the object. If the object is found, the function returns 1; otherwise, it returns 0.
- CheckObjectType(): a function checking the type of a given object. If a given object belongs to the type of Object Driver or Object Device, the function returns 1. If it belongs to the type of Object Directory, then the function returns 2.

**Algorithm 2**

```plaintext
Algo 2
HideMain(pObjectDirectory)
1 {  
2   for ( i = 0; i<=36; i++ ){  // 37 HashBuckets  
3       pTmp= pObjectDirectory + i*4  
4           j = CheckObjectType(pTmp)  
5           if ( j == 1 ){
6               HideDriver(pTmp)  
7           } else if( j == 2 ){
8               HideMain(pTmp)  
9           }  
10          if (Target -> ChainLink = = Null){  
11              exit  
12          } else{
13              pTmp = pTmp ->ChainLink  
14          }  
15      }
16  }

HideDriver(pTmp)
1 {  
2   i = FindObject(pTmp, Target)  
3   if ( i == 1 ){
4       ptmp = pTmp -> ChainLink  
5       ptmp1 = ptmp -> ChainLink  
6       pTmp -> ChainLink = ptmp1  
7   }
8  }
```

B. Removing Object Drivers from Driver Object_Type

Every object has a pointer to an Object_Type structure which defines the common properties of the same kind of objects. Each kind of objects has a dedicated Object_Type. For example, an Object Device and an Object Driver have different Object_Type. The TypeList member of an Object_Type maintains a string of Object_Creator_Info structures of the same kind of objects. Here the type of objects is referred to Object Driver. Fig. 4 shows the relationship between an Object Driver and an Object_Type. Taking Fig. 4 as an example, we first exploit our loaded rootkit driver to get its Object Driver 0, then move to the Object_Header of the Object Driver 0 to get the pointer to the Object_Type, and finally check its TypeList member to find the desired Object Drivers to hide. In the following, we first define notations, and then the operation process for removing Object Drivers from Driver’s Object_Type is described in Algorithm 3.
Notations definition

pObjectType : a pointer to the Object_Type.
Target : an array storing desired drivers’ names to hide
FindObject() : a function finding the object. If the object is
found, the function returns 1; otherwise, it returns 0.

Algorithm 3

Hide(pObjectType)
1 {
2    pTarget = pObjectType -> Flink
3    while(pTarget != pObjectType){
4        ptmp = pTarget -> Flink
5        i = FindObject (pTarget , Target)
6           if( i = = 1 ) {
7              (pTarget -> Blink) -> Flink = ptmp -> Flink
8              (pTarget -> Flink) -> Blink = ptmp -> Blink
9            pTarget = pTarget -> Flink
10        } 
11   }
12 }

C. Removing Object Devices from Device Object_Type

It is the same as the method described in Item B, except
for different kind of objects. First, we use installed rootkit
driver to get its Object Driver whose DeviceObject member
is a pointer to the Object Device. After obtaining the Object
Device, we can find an Object_Type as shown in Fig. 5.
Finally, we traverse its List_Entry to locate the desired
Object Devices to hide. The operation process for removing
Object Devices is the same as Algorithm 3 described in
Item B, except for the input argument which is a pointer to
an Object_Type obtained from Object Device.

D. Removing Drivers from PsLoadedModuleList

A PsLoadedModuleList data structure is not exported
by Windows Driver Kit (WDK). We find it from Fuizen, FU
rootkit [2], [23] writer, who modified it to hide drivers. Its
TypeList member has double links that point to neighbors
with the same structure as it. Fig. 6 shows a relationship
between an Object Driver and PsLoadedModuleList. Taking
Fig. 6 as an example, we first inspect our loaded rootkit
driver to get its Object Driver’s DriverSection, and further
receive a pointer to PsLoadedModuleList 0. Finally we
traverse PsLoadedModuleList 0 to search the desired drivers
to hide based on DKOM. The operation process for
removing drivers is the same as Algorithm 3 described in
Item B, except for the input argument which is a pointer to
a PsLoadedModuleList.

E. Altering Object Driver Appearance

We modify targeted Object Driver appearance to let it
look different as compared to a normal one, so that it is no
longer a driver format and thus escapes driver signature
based detectors to achieve the purpose of hiding. For
example, as shown in Fig. 7, if the value stored in the offset 0x000h of an Object Driver should be 0x04, then we can alter it with a random value to accomplish the purpose of stealth.

![Replacing this value with a random value](image)

Figure 7. Replacing the value of the Type member of an Object Driver with a random one.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

We have explained the principles of stealth in the proposed new Windows driver-hidden rootkit in Section III. In the following, we will demonstrate the experimental results. First, the experimental procedure is illustrated in Fig. 8.

![Experimental procedure for testing the stealth ability of the proposed DKOM-based driver-hidden rootkit](image)

Figure 8. Experimental procedure for testing the stealth ability of the proposed DKOM-based driver-hidden rootkit.

The proposed rootkit named hookzw.sys is a driver format, composed by Borland TASM5.0. We execute it on Windows XP SP2. According to the recommendations of the Antirrootkit website [30], we find kernel-level detectors DarkSpy, GMER, IceSword and Rootkit Unhooker all have a rating with five stars. Detectors RootkitBuster [31] and Tucan [32], developed by famous Trend Micro and Panda respectively, are not rated, but they also possess a driver-hidden detection capability. Moreover, as stated in the literature [16], [18], [19], [21]-[23], we can find DarkSpy, GMER, IceSword and Rootkit Unhooker are highly effective detectors for identifying DKOM-based systemic threats to kernel data. Therefore, in order to effectively test the stealth ability of the proposed DKOM-based driver-hidden rootkit, we choose the above-mentioned six rootkit detectors. When the proposed rootkit is installed but its stealth functionality is not invoked, the four detectors DarkSpy, GMER, IceSword and Rootkit Unhooker have listed it, respectively; however, the two detectors RootkitBuster and Tucan do not provide with displaying any driver information. When its stealth functionality has been invoked, all of six well-known detectors cannot detect the presence of the proposed rootkit.

In particular, although detectors GMER and Rootkit Unhooker can effectively detect the known sophisticated DKOM-based FU, FUTo and Unreal rootkits, they are still unable to find the proposed DKOM-based rootkit. We conclude that the reasons all of the six detectors cannot detect it should be the following two points. First, in Section III we have introduced the first four tricks to independently hide drivers by respectively modifying the List_Entry data structures of Object Directory, Object Driver, Object Device and PsLoadedModuleList. Some detectors do not completely check whether the four List_Entry data structures may be modified, and therefore the proposed first four tricks are to evade the heuristic based detection method. Second, the proposed fifth trick aims to modify targeted Object Driver appearance to let it look different as compared to a normal one, so that it is no longer a driver format and thus capable of escaping driver signature based detectors to achieve the purpose of hiding. In summary, in order to avoid both heuristic based and signature based detection methods, all of the proposed five tricks must be executed.

V. POSSIBLE REMEDIES AND SOLUTION

In this section, we will discuss possible remedies and solution for detecting the proposed new rootkit in order to help detector researchers strengthen their techniques against such a subtle rootkit. Current rootkits and detectors are almost always through drivers in the kernel mode to execute their tasks. Thus, such a way is also adopted in the proposed solutions for detecting the sophisticated driver-hidden rootkit. In Section III, the first four tricks are to independently hide drivers by respectively modifying the List_Entry structures of Object Directory, Object Driver, Object Device and PsLoadedModuleList. If a detector counts on these List_Entry structures, it is consequently unable to identify our proposed Windows driver-hidden rootkit, because the hidden driver has already disappeared from the modified doubly linked lists of the List_Entry structures and therefore it is impossible for a detector to easily find the hidden driver from these List_Entry structures. As Schuster’s research process [33], we can search for drivers from Microsoft Windows memory dumps. This approach does not depend on the List_Entry structures, and is consequently able to resist the attack of DKOM modification to provide trusted information. Therefore, we can be based on such an idea to get reliable information about drivers. In order to detect the proposed new rootkit, we can employ the cross view based
The function ZwQuerySystemInformation returns a list of loaded drivers from a set of PsLoadedModuleList Objects which are necessary targets for rootkits to manipulate for hiding desired drivers. We traverse the set of PsLoadedModuleList Objects to get the list of loaded drivers as the fake information. After gathering the original information and the fake information, the crow view based rootkit detection can be employed to detect the proposed sophisticated driver-hidden rootkit.

VI. CONCLUSION AND FUTURE WORK

In this paper, we list possible targets on which rootkit writers may focus, and the proposed new Windows driver-hidden rootkit with five tricks based on DKOM has successfully avoided the well-known rootkit detectors. To the best of our knowledge, there are no papers about such a kind of study to have been published. We affirm our research is valuable for rootkit detector researchers to discover the weaknesses of their detectors, and can be a great inspiration to defenders to effectively improve the current techniques of detecting unknown Windows driver-hidden rootkits. Furthermore, this study also inspires defenders to effectively reinforce the legitimate uses by the stealth tricks of the proposed rootkit. For example, the same stealth technology used by rootkits can also be used in security software such as firewalls and host-based intrusion prevention systems to extend the protection of the operating system.

With the experience in developing a rootkit, we are continuing our research on developing procedures for detecting a variety of Windows driver-hidden rootkits. It is expected that in the future our proposed detector will give developers a different aspect of detection so that they can combine their detection method with ours to form better solutions.

ACKNOWLEDGMENT

This work was supported by the National Science Council of Republic of China under contract numbers NSC 98-2219-E-212-001.

REFERENCES

[24] NT Rootkit, 


functions,” in Proc. the Third USENIX Windows NT Symposium, pp. 
135-143, 1999.

2010.


implication for intrusion detection,” in Proc. the IEEE International 


[33] A. Schuster, “Searching for processes and threads in Microsoft 
Windows memory dumps,” Digital Investigation: The International 