Dual-mode Kernel Rootkit Scan and Recovery with Process ID Brute-force

Ji-won Choi[1], Sang-jun Park[2], Seong-Jin Byeon[3], Dongho Kim[4]

[1,2,3] Dept. of Computer Science and Engineering
[4] Convergence Software Institute
Dongguk University, Korea
{[1]eternalklaus, [2]potter77777, [3]sjbyeon610, [4]dongho.kim}@dongguk.edu

Abstract Rootkit is a malware that attacks a system continuously by hiding files, processes, and registries in a system. DKOM (Direct Kernel Object Manipulation) is a process hiding technique that manipulates a kernel object. AL-DKOM(All Link – Direct Kernel Object Manipulation) is an extended version of DKOM that manipulates all the modifiable links in the kernel object. It is difficult to detect with existing tools. This paper designed and implemented a new AL-DKOM detection system using dual-mode operation and PIDB-based process scan that was not possible with existing List Walking method. In addition, we explain the recovery procedure of the infected system and flexible system management via process recovery that has not been tried before. We then show the performance of the new system by comparing and analyzing the effectiveness of the new system and existing rootkit scanning tools with real rootkit samples.

Keywords: Rootkit, PIDB, AL-DKOM, Dual-Mode

1. INTRODUCTION

Rootkit is a hidden malware that hides its existence in a system[1,2,3]. Such rootkit malwares are more dangerous than others since they may cause damage to the victims hidden inside of them avoiding detection by vaccine software.
According to McAfee Labs Threats Report 2015, the number of rootkit malwares is increasing and it was over 1.6 million during 4th quarter of 2015[4].

We designed a reliable and safe rootkit scan and recovery system. Next section describes previous work[5], on rootkits and rootkit detection techniques. It explains DKOM(Direct Kernel Object Manipulation) which is a process masking technique and introduces AL-DKOM(All Link - Direct Kernel Object Manipulation) as a new masking technique. Third section explains characteristics and limitations of existing rootkit detection techniques. It then suggests a new system that can overcome the limitations of the existing systems. Operation details and algorithm of it is introduced subsequently. The last section concludes the paper with summary and future work.

2. RELATED WORK

2.1. EPROCESS Kernel Structure

Every process running on Windows operating system produces a kernel object called EPROCESS[6]. This structure contains information such as process name and memory allocation and it is utilized throughout the life of the process. The EPROCESS structures of all the processes are connected as a circular doubly linked list, simply called as the *circular structure* in this paper.

Fig. 1. shows a part of the EPROCESS structure in Microsoft Windows 10. EPROCESS structures are slightly different depending on the specific Windows versions. The ActiveProcessLinks as a _LIST_ENTRY data type connects the EPROCESS structures to form the circular structure as shown in Fig. 1. `Flink` and `Blink` in the structure connect the previous and next processes of the current one, respectively as shown in Fig. 2.

```
kd> dt nt!_EPROCESS
   +0x000 Pcb              : _KPROCESS
   +0x0a8 ProcessLock      : _EX_PUSH_LOCK
   +0x0ac RundownProtect   : _EX_RUNDOWN_REF
   +0x0b0 VdmObjects       : Ptr32 Void
   +0x0b4 UniqueProcessId  : Ptr32 Void
   +0x0b8 ActiveProcessLinks : _LIST_ENTRY
   +0x0c0 Flags2           : Uint4B
   +0x0c0 JobNotReallyActive : Pos 0, 1 Bit
   +0x0c0 AccountingFolded : Pos 1, 1 Bit
```

Fig. 1. Structure of EPROCESS

```
   +0x0b8 ActiveProcessLinks : _LIST_ENTRY
      +0x000 Flink           : Ptr32
      +0x004 Blink           : Ptr32
```

Fig. 2. Structure of ActiveProcessLinks

Including ActiveProcessLinks, there are 17 different links that connect the EPROCESS structures in Windows 10. They form the circular structure that connect all the running processes as shown in Fig. 3. The process scanning tool in Windows traverses the circular structure with List Walking method.

2.2. DKOM (Direct Kernel Object Manipulation)

DKOM is one of the process masking techniques that hides a process by manipulating the links on the circular structure[7,8]. After locating EPROCESS of the target process, it disconnects Flink and Blink in the circular structure so that the EPROCESS is isolated from the structure and it is no longer found from the circular structure as shown in Fig. 4. As a result, the hidden process cannot be detected by process scanning tools such as Task Manager, Process Explorer, and etc.

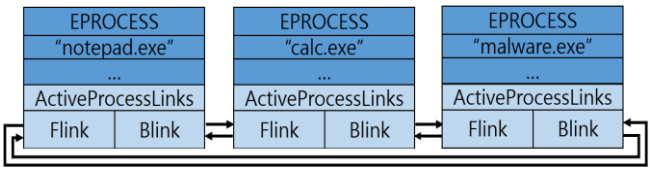

Fig. 3. Circular doubly linked list of working process

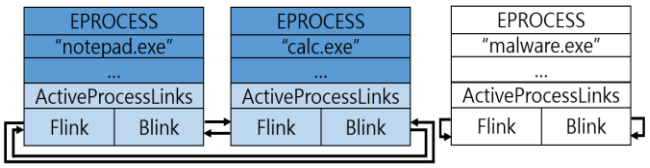

Fig. 4. EPROCESS structure with DKOM

DKOM manipulates Flink and Blink of ActiveProcessLinks because it is used by process scanners with List Walking method. Thus, a malware could be hidden from scanning tools by doing so. However, there are 17 different links including ActiveProcessLinks in the circular structure. Table 1. lists links of EPROCESS in Windows 10 with possibility of abuse by rootkits denoted with O/X. The links with O may by disconnected and used to mask processes whereas the links with X have low possibilities of use to hide since disconnecting them may cause critical system errors such as BSOD (Blue Screen of Death).

As the links listed on Table 1. connects EPROCESS structures, rootkits may be detected by traversing other links even if ActiveProcessLinks is disconnected. Thus, an improved rootkit that is harder to be detected could be produced only by disconnecting all the possible 13 links. The next section will discuss this new method in detail.

2.3. AL-DKOM (All Link – Direct Kernel Object Manipulation)

An improved rootkit that performs better masking may be designed by manipulating all the link structures that can be used for malicious purposes, instead of isolating ActiveProcessLinks only as done in DKOM. If one disconnects all 13 links that can be used for malicious purpose, someone could create a new rootkit that may be harder to detect and cure. We name the technique as AL-DKOM (All Link - Direct Kernel Object Manipulation). This is a new concept introduced in 2016[5], and the term AL-DKOM is newly used in this paper.

Table 1. Links in EPROCESS structure and vulnerabilities

WaitListHead	O	ActiveProcessLinks	O
ProfileListHead	O	SessionProcessLinks	O
ThreadListHead (0x2c)	O	JobLinks	O
ReadyListHead	O	ThreadListHead(0x194)	O
ProcessListEntry	X	WorkingSetExpansion Links	O
MmProcessLinks	X	ViewListHead	O
TimerResolution Link	X	SvmProcessDeviceList Head	O
SiloEntry	X	SharedCommitLinks	O
HandleTableList in ObjectTable	O		

2.4. Existing Rootkit Detection and Limitation

Existing rootkit detection tools applies List Walking method to multiple links. They find a running process by traversing ActiveProcessLinks and HandleTableList, then compare the results between the two. The process is determined as hidden if there is a difference between the two traversal results. For example, a hidden process would be detected from only one traversal, if ActiveProcessLinks is disconnected, but HandleTableList is still intact. Such an existing rootkit detection method that traverses links can only detect DKOM, but they cannot detect AL-DKOM, since it disconnects all the links.

2.4. Graceful Handling after Detection

The countermeasures for the detected rootkits of the existing systems are in two folds: one is just showing the list of the hidden processes to the user. The other is terminating the hidden processes. The former has a limitation since it just finds rootkits instead of processing them. The systems administrator may find rootkits, but they cannot terminate or delete the hidden processes. Tools that adopt the latter approach simply terminates the hidden processes instead of recovering them. This approach is rather an extreme and temporary solution that may cause unexpected errors or complications due to unnecessary termination of benign hidden processes.

The system we propose in this paper is designed to recover from infection flawlessly by overcoming the limitations from the existing rootkit detection methods and the countermeasures for the detected processes.

3. PROPOSED SYSTEM

3.1. Detection and Recovery Process of Proposed System

Proposed system is composed of two programs. One program runs in the user-mode and the other one runs in the kernel-mode[9] as a form of a device driver (.sys). Two programs detect and recover rootkits by communicating each other as shown in Fig. 5.

3.2.1 User-mode Program with PIDB

The user-mode program scans all the running processes using PIDB (Process ID Brute-force)[1]. As shown in Table 2, it accesses all the programs using PIDs to check existence of a process. If it determines a process exists, process information is conveyed to the kernel-mode program and let the user know as shown in Fig. 6.

PIDB scans all processes using process listing API that has been used by existing user-mode rootkit detection programs. We designed the proposed system so that the user-mode program with PIDB communicates with kernel-mode program to detect kernel-mode rootkit.

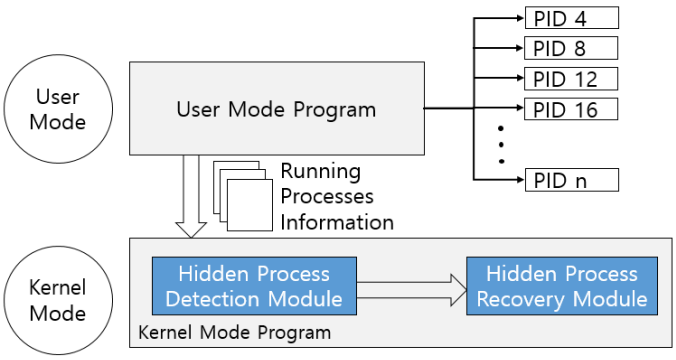

Fig. 5. Operation of the proposed system

Table 2. User-mode program: All process scan

```
1.  PID=0;
2.  while(PID<PID_MAX) {//scan all PID
3.  PID+=4; ; //increase PID
4.  hProc=NULL;
5.  hProc=OpenProcess(PROCESS_ALL_ACC
6.  ESS, true,PID); //get PID handle of process
7.  if (hProc==NULL) {
8.  continue; //if PID does not exist Continue.
9.  }
10. if(!WriteFile(hFile,&PID,sizeof(DWORD),&writ
11. e,
12. NULL)) {//try process delivery to kernel-mode
13. program
14. [...omit...] //exception handling when driver
15. call error
16. }
17. CloseHandle(hProc); //put handle finished
18. use
19. printf("Sending process information to device
20. driver.....\n\n");
21. }
```

```
C:\User\eternalklaus\Desktop>DKOM_REVEALER.exe
loading DKOM_REVEALER.sys...
Service started successfully
Driver installed successfully

pid[0524]=dse.exe
Sending process info to device driver...

pid[0596]=f-secure blacklight.exe
Sending process info to device driver...

pid[1200]=aswmbr.exe
Sending process info to device driver...

pid[1944]=DataExchangeHost.exe
Sending process info to device driver...

pid[2112]=ShellExperienceHost.exe
Sending process info to device driver...

pid[2460]=TDSSKiller.exe
Sending process info to device driver...
```

Fig. 6. User-mode program: All process scan

3.2.2 Kernel-mode program as a device driver

The second program runs in the kernel-mode as a device driver and performs rootkit detection and recovery[10, 11].The detection procedure uses System Base Detection Tech, which is a kind of Behavior Detection Tech that detects malicious codes by monitoring malicious behaviors in a system. Proposed system considers the act of modifying EPROCESS to hide a process as a malicious behavior.

Table 3. shows a part of the kernel-mode program. When the system runs, it receives information for all the running processes from the user-mode program. It checks whether a certain process is hidden, then performs recovery if the process is determined as hidden. It detects whether a process is hidden by examining the status of the EPROCESS kernel structure. It determines a process as hidden if it is abnormal, e.g., the link on the structure is pointing itself or the link is disconnected. It recovers a hidden process by reconstructing the damaged circular structures with reconnecting the disconnected links after locating the ActiveProcessLinks. It

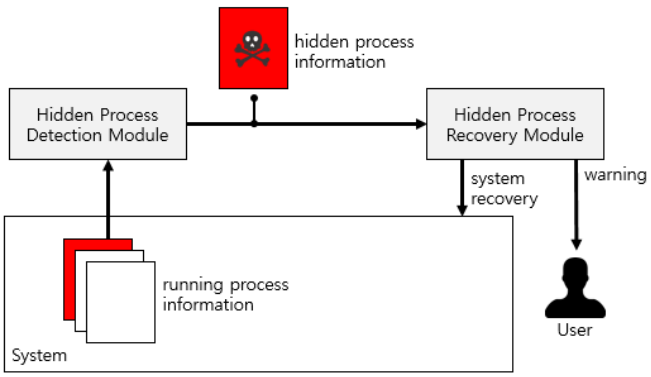

Fig. 7. Operation of kernel-mode program

then notifies the user with successful recovery of the process by displaying the information of the hidden process as a debug message.

It runs in the kernel-mode as it is necessary to access the kernel structure EPROCESS. We designed and built is as a device driver as it is possible to access and operate on kernel structures.

3.2.3 Merit of dual-mode operation

The proposed system overcomes the limitations of the existing systems that could only detect user-mode rootkits with PIDB, since it is capable of detecting kernel-mode rootkits by communicating with the kernel-mode program. It also is capable of detecting hidden processes with damaged EPROCESS structures. This is possible since the kernel-mode program can access the process objects directly through PID, instead of link traversal by List Walking method. As a result, hidden processes with AL-DKOM can be detected.

3.3. Comparison between Existing Rootkit Detection Tools and Proposed System

Among the existing tools, Gmer[12,13], Kaspersky tdsskiller[14,15], and AVAST Anti-Rootkit[16] are capable of detecting DKOM, but they cannot detect AL-DKOM.

F-Secure Blacklight[17,18] and IceSword[19,20] are capable of detecting both DKOM and AL-DKOM, but the detection entails terminating the hidden process and system reboot, which is an extreme and temporary measure. Besides, they only support old versions of Windows since the program update has been discontinued.

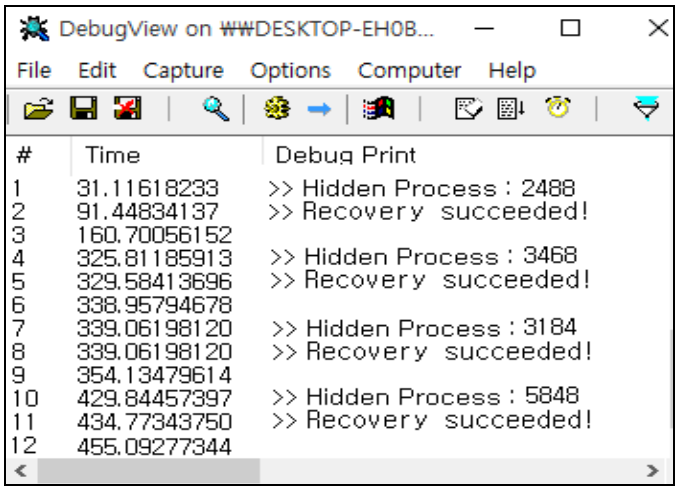

Fig. 8. Kernel-mode program:
Hidden process detection and recovery

Table 3. Kernel-mode program:
Hidden process detection and recovery

```
1.  // get EPROCESS object of the process
2.  NOW_EPROCESS = PsGetCurrentProcess();
3.
4.  // get ActiveProcessLinks address of the process
5.  NowListEntry =
6.  (PLIST_ENTRY)((DWORD)NOW_EPROCESS+
7.  0xb8);
8.
9.  // detects whether the process is hidden or not
10. Based on status of kernel objects in memory.
11. if((CurrListEntry->Flink->Blink==CurrListEntry->B
12. link)    || ((CurrListEntry->Flink==CurrListEntry)
13. && (CurrListEntry->Blink==CurrListEntry)))
14. __try {
15.
16. // recover the hidden process
17. CurrListEntry->Flink=NowListEntry->Flink;
18. CurrListEntry->Blink=NowListEntry->Flink->Blink;
19. NowListEntry->Flink->Blink=CurrListEntry;
20. NowListEntry->Flink=CurrListEntry;
21.
22. // print information of hidden process to Debug
23. Message.
24. HiddenPID = PsGetProcessId(NOW_EPROCESS);
25. DbgPrint(">> Hidden Process : %d \n",HiddenPID);
26. DbgPrint(">> Revocery. Success. \n");
27. }
28. __except{
29. [...omit...]
30. //exception handling when failsdriver call error
31. }
```

The proposed system, on the other hand, is not only capable of detecting both DKOM and AL-DKOM, but also can recover hidden processes by revealing them. The recovered processes may be scanned because they lose the masking capability of the rootkit. Flexible process management is possible because post-processing, e.g., checking process information, terminating the process, and etc., of the recovered processes may be done by the systems administrator.

3.4. Real Rootkit Malware Detection and Recovery

The proposed system detects hidden processes based on the status of the kernel structure EPROCESS. This is one of the behavior based detection[21] method and it does not need any signature file updates.

Table 4. Comparison with existing rootkit detection tools

Rootkit Detection Tool	DKOM Detection	AL-DKOM Detection	Recovery of Hidden process	Support Windows 10
Gmer	O	X	X	O
Kaspersky tdsskiller	O	X	X	O
AVAST Anti-rootkit	O	X	X	O
F-Secure Blacklight	O	O	X	X
IceSword	O	O	X	X
Proposed system	**O**	**O**	**O**	**O**

We verified the operation of the system with real rootkit malwares and listed representative malwares detected by the system on Table 5.

Table 5. Detection possible malware with proposed system

File Name	fu1.exe
MD5	CDE649535F655D0FEC152AC9A332A937
SHA1	0C6BF2E83DE2A840F87488E4B981F520102B9596
Avast	Win32:Fu [Trj]
AVG	BackDoor.Generic.YRX
File Name	VirusShare_8c056071ce56d2c25ea0dd1715c97c73
MD5	1E04D421CF11F48269FC4F3B8A717061
SHA1	0E4610CC93CCD6637F1CE1C518025AFD3E6325A0
Avast	Win32:Agent-DAW [Trj]
AVG	BackDoor.Generic9.KVB
File Name	VirusShare_0d4cc913ec32185de15ed25f87327375
MD5	C4DEEABA0FD575939ABE0FF5E358BCD4
SHA1	F290824BB90720D7393ACA2736689A3739E3FFEB
AVAST	Win32:Agent-DAW [Trj]
AVG	Exploit.MS04-011
File Name	DKOM_Downloader.exe
MD5	1C8FCE9A1CD0895649EF80E1EDB4066C
SHA1	4AC897BEFA78B6374FBC9D6A92BE39F5EFFBBE85
Symantec	Heur.AdvML.B
Qihoo-360	HEUR/QVM19.1.0000.Malware.Gen

4. CONCLUSIONS AND FUTURE WORK

We introduced a system that scans processes in user-mode, then detects rootkits and recovers in kernel-mode, with two main contributions. It utilizes PIDB for detecting kernel-mode rootkits, instead of user-mode rootkits only as done with the existing systems. It also focuses on recovery of revealing hidden rootkit processes, instead of simply terminating rootkits, so that the systems administrator may handle the situation with flexibility.

Proposed system supports rootkit detection and recovery on Windows. However, many rootkit malwares on Linux systems are found in these days. We intend to expand the work on Linux rootkits.

Acknowledgment

This research was supported by MISP (Ministry of Science, ICT & Future Planning) of Korean government, under the National Program for Excellence in Software (R7116-16-1014) grant supervised by IITP (Institute for Information & communications Technology Promotion).

References and Notes

1. Hoglund, Greg, Rootkits : subverting the Windows kernel, **(2006)**, pp. 12-66
2. Chris Ries, Inside windows Rootkits, VigilantMind, index-of.co.uk, **(2006).**
3. Jamie Butler, Peter Silberman, Raide, Rootkit analysis identification elimination, Black Hat USA, blackhat.com, **(2006).**
4. Intel Security, McAfee Labs Threats Report 2016, http://www.mcafee.com/uk/resources/reports/rp-quarterly-threats-aug-2015.pdf, **(2016).**
5. Jiwon Choi, Bongkyo Moon, Study on Detection Method and Development of the Kernel Mode Rootkit. Korea Information Processing Society Spring academic conference collection of dissertations, **(2016),** vol.23
6. Andrea Lanzi, Monirul Sharif, Wenke Lee, K-Tracer, A System for Extracting Kernel Malware Behavior, NDSS, pdfs.semanticscholar.org, **(2009).**
7. Elia Florio Florio, When Malware Meets Rootkits, Virus Bulletin, vxheaven.org, **(2005).**
8. Sherri Sparks, Jamie Butler, in *Black Hat Japan* Raising the bar for rootkit detection, blackhat.com, **(2005).**
9. Symantec, Windows Rootkit Overview, White paper, www.symantec.com , **(2006).**
10. Zhi Wang, Xuxian Jiang, Weidong Cui, Peng Ning, Countering kernel rootkits with lightweight hook protection, CCS '09 Proceedings of the 16th ACM conference on Computer and communications security, dl.acm.org, **(2009)**, pp.545-554
11. Ryan Riley, Xuxian Jiang, Dongyan Xu, Guest-Transparent Prevention of KernelRootkits with VMM-based Memory Shadowing, Recent Advances in Intrusion Detection - Volume 5230 of the series Lecture Notes in Computer Science, **(2008),** pp. 1-20
12. gmer, http://www.gmer.net/, **(2016).**
13. Thomas Martin Arnold, A comparative analysis of rootkit detection techniques, sceweb.sce.uhcl.edu, **(2011).**
14. Kaspersky, TDSSKiller, http://usa.kaspersky.com/downloads/ TDSSKiller, **(2016).**
15. Igor Korkin, Ivan Nesterov, Applying memory forensics to rootkit detection, arXiv preprint arXiv, arxiv.org, **(2015).**
16. raymond, AVAST Antirootkit, https://www.raymond.cc/blog/10-antirootkits-tested-to-detect-and-remove-a-hidden-rootkit/, **(2016).**
17. BleepingComputer, f-secure blacklight, http://www.bleeping.computer.com/tutorials/use-blacklight-to-remove-rootkits/, **(2016).**
18. A Baliga, L Iftode, XM Chen, Automated defense from rootkit attacks, academia.edu, **(2006).**
19. Softonic, IceSword, http://icesword.en.softonic.com/, **(2016).**
20. Woei-Jiunn Tsaur, Yuh-Chen Chen, Exploring Rootkit Detectors' Vulnerabilities Using a New Windows Hidden Driver Based Rootkit, Social Computing (SocialCom), **(2010).**
21. v3, System Base Detection Tech, http://v3.kcu.or.kr/secu_info_view.asp?list=/secu_info_list.asp&seq=9533&pageno=62&v_num=202, **(2007).**