윈도우 OS에서 DKOM 기반 루트킷 드라이버 탐지 기법

심재범*, 나중찬**
*과학기술연합대학원대학교 정보보호공학
**한국전자통신연구원
e-mail:puiw89@gmail.com

Detecting a Driver-hidden Rootkit using DKOM in Windows OS

Jae-bum Sim*, Jung-chan Na**
*Dept of Information Secutiry Engineering,
University of Science & Technology
**ETRI

요 약

많은 루트킷들은 사용자 또는 탐지 프로그램으로부터 탐지되지 않기 위해서 중요한 함수를 후킹하거나 DKOM 등의 기술을 이용한다. API를 후킹하여 반환되는 정보를 필터링하는 전통적인 루트킷과 달리 DKOM 루트킷은 이중 연결리스트로 구성된 커널 오브젝트의 링크를 직접 조작하여 루트킷 자신의 존재를 숨길 수 있으며 DKOM으로 은닉한 루트킷은 쉽게 탐지되지 않는다. 본 논문에서는 DKOM을 이용하여 은닉된 루트킷 드라이버를 탐지하기 위한 기술을 제안하고, 루트킷 탐지 능력을 검증한다.

1. 서론

루트킷은 사용자 또는 보안 프로그램으로부터 악성코드를 보호하고 은닉하기 위한 목적으로 제작된 프로그램이다. 기존의 루트킷은 탐지되지 않기 위해 주로 후킹을 하여 함수의 제어흐름(Control Flow)을 조작한다. 최근 루트킷은 DKOM(Direct Kernel Object Manipulation)[1] 기술을 이용하여 OS에서 변경 가능한 영역을 조작하여 루트킷의 프로세스뿐만 아니라 루트킷의 드라이버도 은닉시킬 수 있다. 특히 드라이버는 커널 레벨에서 임의의 코드를 실행할 수 있기 때문에 쉽게 탐지되지 않으며, 그 위험성이 매우 크다. DKOM 기반 루트킷 드라이버를 탐지하기 위한 방법으로는 메모리 덤프를 분석하는 방법과 메모리를 직접 조사하는 온라인 분석 방법이 있다. 메모리 덤프 분석 방법은 은닉된 드라이버를 정확하게 찾아낼 수 있지만, 시간이 오래 걸리며 실시간으로 분석할 수 없다는 단점이 있다. 온라인 분석 방법은 빠르게 은닉된 드라이버를 찾아낼 수 있지만 메모리 덤프 방법에 비해 정확도가 떨어지는 단점이 있다.

본 논문에서는 커널 메모리 영역을 직접 조사하여 DKOM 기반 루트킷 드라이버를 빠르고 정확하게 탐지하기 위한 방법으로 메모리 스캐닝과 윈도우 레지스트리 검색 방법을 제안한다.

이를 위해서 본 논문의 구성은 다음과 같다. 2장에서는 DKOM 기반 루트킷 드라이버 탐지와 관련된 연구에 대해 소개한다. 3장에서는 본 논문이 제안하는 메모리 스캐닝 방법과 레지스트리 검색 기법에 대해 기술한다. 4장에서는 제안된 기법에 대한 검증과 평가를 하고, 마지막으로 5장에서 결론을 내린다.

2. 관련 연구

2.1 루트킷의 은닉 방법

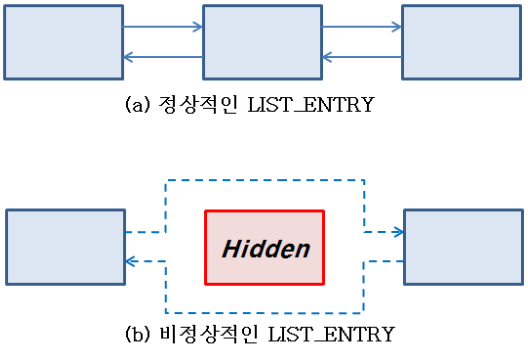
DKOM은 Jamie Butler에 의해 처음 소개된 루트킷 기술로 커널 오브젝트를 직접 조작하여 후킹없이 루트킷의 프로세스, 스레드 또는 드라이버를 숨길 수 있게 된다. DKOM은 후킹을 이용하는 방법보다 탐지하기 어렵다.

Windows OS에서 로드되어있는 각 드라이버에 관련된 정보는 DriverObject 구조체에 저장된다. DriverObject의 구조는 (그림 1)[2]과 같다. 각 DriverObject들은 이중 연

```
+0x000 Type             : Int2B
+0x002 Size             : Int2B
+0x004 DeviceObject     : Ptr32 _DEVICE_OBJECT
+0x008 Flags            : Uint4B
+0x00c DriverStart      : Ptr32 Void
+0x010 DriverSize       : Uint4B
+0x014 DriverSection    : Ptr32 Void
+0x018 DriverExtension  : Ptr32 _DRIVER_EXTENSION
+0x01c DriverName       : _UNICODE_STRING
+0x024 HardwareDatabase : Ptr32 _UNICODE_STRING
+0x028 FastIoDispatch   : Ptr32 _FAST_IO_DISPATCH
+0x02c DriverInit       : Ptr32     long
+0x030 DriverStartIo    : Ptr32     void
+0x034 DriverUnload     : Ptr32     void
+0x038 MajorFunction    : [28] Ptr32     long
```

(그림 1) DriverObject 구조체

결리스트 LIST_ENTRY로 연결되어있다. Windows OS는 이중 연결리스트를 순회하면서 실행되어있는 프로세스 목록이나 로드된 드라이버 목록을 출력한다. DKOM 기반 루트킷은 자신의 프로세스나 드라이버를 은닉하기 위해 자신의 커널 오브젝트로 연결된 이중 연결리스트의 링크를 끊는다. (그림 2)는 정상적인 이중 연결리스트와 루트킷에 의해 링크가 조작된 이중 연결리스트를 나타낸다.



(그림 2) 정상적인 링크(a)와 비정상적인 링크(b)

2.2 루트킷의 탐지 기법

DKOM 기반 루트킷 드라이버를 탐지하는 방법으로는 커널 메모리 영역을 덤프하여 조사하는 방법[7]과 메모리를 직접 조사하는 방법이 있다. 오픈소스 메모리 포렌식 도구인 Volatility[11]는 메모리 덤프 방식을 이용한다. Volatility는 은닉한 드라이버를 정확하게 찾을 수 있지만 시간이 오래 걸리며 실시간으로 찾아낼 수 없다는 단점이 있다. 온라인 분석 방법[3,4,5]을 이용하면 빠르게 루트킷을 탐지할 수 있다. 이러한 도구들로는 GMER[12], Rootkit Unhooker[13], Rootkit Buster[14], IceSword[15]가 있다. 온라인 분석 방법은 루트킷을 빠르게 탐지할 수 있지만 정확도가 많이 떨어지는 단점이 있다.

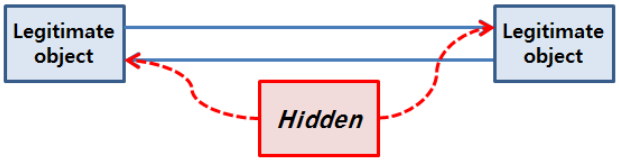
3. DKOM 기반 루트킷 드라이버 탐지기법

DKOM 루트킷의 탐지를 위해서 먼저 메모리 스캐닝을 진행한 후 윈도우 레지스트리로부터 얻어온 드라이버 정보를 비교한다. 메모리 스캐닝 과정에서는 이중 연결리스트의 링크가 유효한지를 검사한다.

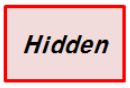
3.1 이중 연결리스트 무결성 검사

프로세스 또는 드라이버를 나타내는 커널 오브젝트의 이중 연결리스트는 루트킷에 의한 조작이 없을 경우 환형 이중 연결리스트(Circular doubly-linked list)의 형태가 나타난다. 하지만 루트킷에 의해 이중 연결리스트가 조작되었을 경우에는 환형 이중 연결리스트가 변경된다. 커널 오브젝트의 이중 연결리스트 LIST_ENTRY의 링크가 올바른지 확인하기 위해 포워드 링크(Forward link)와 백워드 링크(Backward link)를 분석한다. 포워드 링크와 백워드 링크가 조작된 경우를 아래와 같이 3가지로 분류하였다.

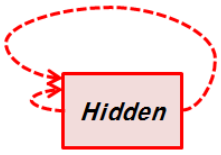
1) 포워드 링크와 백워드 링크가 다른 오브젝트를 가리키는 경우
2) 포워드 링크와 백워드 링크가 NULL이거나 부정확한 주소를 가리키는 경우
3) 포워드 링크와 백워드 링크가 자기 자신을 가리키는 경우



(그림 3) 이중 연결리스트의 링크가 다른 오브젝트를 가리키는 경우



(그림 4) 이중 연결리스트의 링크가 NULL이거나 유효하지 않은 주소를 가리키는 경우



(그림 5) 이중 연결리스트의 링크가 자기 자신을 가리키는 경우

이러한 경우를 검사하기 위해 두 가지 비교를 수행한다. 첫 번째로 (LIST_ENTRY -> FLINK) -> BLINK가 LIST_ENTRY의 주소와 같은지 확인한다. (LIST_ENTRY -> FLINK) -> BLINK가 LIST_ENTRY의 주소와 같지 않다면 그 LIST_ENTRY는 루트킷의 것이라고 볼 수 있다. 이 비교를 통해 1)와 2)를 동시에 확인 할 수 있다. 비슷하게 (LIST_ENTRY -> BLINK) -> FLINK가 LIST_ENTRY의 주소와 같은지 확인할 수 있다. 두 번째로 3)을 확인하기 위해 LIST_ENTRY -> FLINK와 LIST_ENTRY의 주소가 같은지 확인한다. 만약 두 개가 같다면, 역시 루트킷이라고 판별한다. 같은 방법으로 반대 방향도 확인한다. (그림 6)은 이중 연결리스트 무결성 검사를 수행하는 pseudocode를 나타낸다. Integrity_check1은 첫 번째 연결리스트 무결성 검사를 나타내고 Integrity_check2는 두 번째 연결리스트 무결성 검사를 나타낸다.

3.2 메모리 스캐닝

DriverObject의 모든 목록을 출력하는 API는 따로 존재하지 않기 때문에 커널 오브젝트를 분석하기 위해 OS 디버거를 이용해야 한다. Microsoft에서 제공하는 Windbg[10]를 이용하여 커널 오브젝트를 분석하였고, Virtual PC 위에서 구동되는 윈도우XP를 디버깅하였다.

DriverObject는 각 드라이버의 타입, 크기, 이름 등 여러 가지 정보를 나타낸다. 그 중 오프셋 +0x14 위치에는

```
BOOLEAN function integrity_check1
SET IsRootkit to false;
IF ((LIST_ENTRY -> FLINK) -> BLINK) isn't equal to LIST_ENTRY
    SET IsRootkit to true;
IF ((LIST_ENTRY -> BLINK) -> FLINK) isn't equal to LIST_ENTRY
    SET IsRootkit to true;
RETURN with IsRootkit;



BOOLEAN function integrity_check2
SET IsRootkit to false;
IF (LIST_ENTRY -> FLINK) equals LIST_ENTRY
    SET IsRootkit to true;
IF ((LIST_ENTRY -> BLINK) -> FLINK) equals LIST_ENTRY
    SET IsRootkit to true;
RETURN with IsRootkit;
```

(그림 6) 연결리스트의 링크를 검사하는 pseudocode

이중 연결리스트를 담고 있는 DriverSection 구조체의 주소가 존재한다. DriverSection은 문서화 되어있지 않은 구조체이며, 이를 분석하여 확인한 내용은 (그림 7)와 같다.

```
+0x000 blink            : Ptr32 _DriverSection
+0x004 flink            : Ptr32 _DriverSection
+0x008 Address1         : Ptr32
+0x00c Unknown          : Uint32
+0x010 Unknown          : Uint32
+0x014 Unknown          : Uint32
+0x018 BaseofPE         : Ptr32
+0x01c DriverInit       : Ptr32
+0x020 SizeofPE         : Uint32
+0x024 Unknown          : Uint32
+0x028 DriverPath       : PUNICODE_STRING32
+0x02c Unknown          : Uint32
+0x030 PDriverName      : PUNICODE_STRING32
+0x034 Unknown          : Uint32
+0x038 Unknown          : Uint32
+0x03c Unknown          : Uint32
+0x040 Unknown          : Uint32
+0x044 Unknown          : Uint32
+0x048 DriverName       : UNICODE_STRING
      ...
```

(그림 7) DriverSection 구조체

DriverSection 구조체에서는 이중 연결리스트 뿐만 아니라 드라이버 파일의 크기, 위치, 드라이버 이름 등의 정보를 확인할 수 있다.

추가적인 분석을 통해서 모든 DriverSection의 -0x04 위치에는 항상 MmLd(0x4d6d4c64) 문자열이 포함되어 있는 것을 확인할 수 있었다. MmLd 문자열이 존재하는 메모리 주소의 마지막 자리는 항상 0x4 또는 0xC이다. 또한 DriverSection 구조체는 항상 커널 메모리 영역 0x85000000에서 0x87000000 사이에 존재하였고, 메모리 스캐닝을 할 때는 0x81000000에서 0x8A000000의 범위에서 DriverSection 구조체를 검색하였다. 위의 방법으로 루트킷이 없는 시스템에서 모든 드라이버를 검색할 수 있다.

커널 메모리 영역에서 발견된 모든 DriverSection 구조체에 대해서는 LIST_ENTRY의 링크가 정확한지 확인하기 위해 이중 연결리스트 무결성 검사를 진행한다. 추가적으로 메모리 스캐닝 결과와 이중 연결리스트를 모두 순회하였을 때의 결과를 비교한다. 메모리 스캐닝에서는 확인되지만 이중 연결리스트를 모두 순회하였을 때 나타나지 않는 오브젝트는 루트킷이라고 간주한다.

메모리 스캐닝 방법을 이용하여 빠르게 메모리를 검색하여 링크의 무결성을 확인할 수 있고, DKOM 기반 루트킷 드라이버를 탐지하는 것이 가능하다. 하지만 일부 루트킷의 경우 DriverSection의 MmLd 매직넘버 값을 변경한다. 이런 루트킷은 메모리 스캐닝으로 확인하는 것이 불가능하다.

3.3 레지스트리 검색

메모리 스캐닝 방법의 단점은 경우에 따라서 탐지되지 않는 루트킷이 존재한다는 것이다. MmLd 매직넘버를 이용해서 메모리 스캐닝을 수행하기 때문에 루트킷이 매직넘버를 변경한다면 메모리 스캐닝으로 확인할 수 없다. 이 단점을 보완하기 위해서 로드되는 모든 드라이버들에 대한 정보를 레지스트리로부터 읽어온다. Windows OS는 부팅과정에서 다음 레지스트리 키의 서브키들을 읽어서 우선순위에 따라 드라이버를 로드한다.

\HKLM\SYSTEM\CurrentControlSet\Service\

루트킷의 행위 중에 하나는 OS에서 은닉되어있는 상태를 지속하는 것이다. 매번 부팅할 때마다 루트킷 드라이버가 로드되기 위해서 루트킷은 레지스트리 하이브에 자신의 존재를 알려야만 한다. 몇몇 루트킷들은 레지스트리에서 자신의 존재를 숨기기 위해 RegOpenKey, RegEnumKey, ZwOpenKey, ZwEnumerateKey와 같은 API를 후킹하기도 한다. 지금까지 알려진 루트킷이 레지스트리에서 자신을 숨기기 위해 사용할 수 있는 기술은 API를 후킹하는 것이다. 그러므로 우리가 후킹되지 않은, 안전한 API를 사용한다면 모든 드라이버 정보를 확인 가능하다. 안전하게 레지스트리를 검색하기 위해 [9]을 참조하였다.

\HKLM\SYSTEM\CurrentControlSet\Service\[드라이버 이름] 위치에는 로딩 우선순위, 드라이버의 파일 위치 등 로드되는 모든 드라이버에 대한 정보를 나타낸다. 다음은 레지스트리에서 확인할 수 있는 정보이다.

1) DriverName
2) ErrorControl
3) ImagePath
4) Start
5) Type

레지스트리에서 확인할 수 있는 드라이버에 관한 정보가 메모리 스캔을 통해서도 발견 되는지 비교하였다. 만약 어떤 드라이버에 대한 정보가 레지스트리에는 존재하지만, 메모리 스캐닝을 통해 그와 같은 정보를 가지고 있는 DriverSection 구조체를 발견하지 못한다면 그 드라이버는 루트킷의 것이라고 판단한다.

4. 실험 및 평가

DKOM 기반의 루트킷 드라이버에 대한 탐지 기법을 테스트하기 위해서 Microsoft Virtual PC에서 구동되는

Windows XP SP3를 이용하였다. 테스트를 위해 사용한 루트킷으로는 Fu_Rootkit, Unreal[6], BadRKDemo, RKUnhooker's test rootkit을 이용하였다. 또한 제안된 탐지 기법과 비교하기 위해 많이 알려진 안티 루트킷 프로그램 3종을 이용하였다. 실험 결과는 (표 1)과 같다.

제안된 루트킷 탐지 기법은 4가지 루트킷 샘플을 모두 탐지해낼 수 있었다. Fu_rootkit, BadRKDemo의 경우 메모리 스캐닝 방법만으로도 탐지해낼 수 있었다. Unreal, RKUnhooker's Test rootkit은 메모리 스캐닝으로는 탐지되지 않았다. 하지만 레지스트리 검색 방법으로 모든 샘플 루트킷을 찾아낼 수 있었다. 다른 안티 루트킷 프로그램의 탐지 결과는 표에 나타난 것과 같다. 특히 GMER의 경우, RKUnhooker's Test rootkit을 탐지해내긴 했지만 이는 루트킷의 다른 행위를 찾아낸 것이고 드라이버를 은닉한 행위는 찾아내지 못했다.

5. 결론

루트킷은 사용자 또는 보안 프로그램으로부터 탐지되지 않기 위해 다양한 기술을 사용한다. 그 중 DKOM은 드라이버 커널 오브젝트의 이중 연결리스트의 링크를 직접 조작하여 안티 루트킷 프로그램이 루트킷의 드라이버를 탐지하지 못하게 하는 기술이다. 은닉된 DKOM 루트킷 드라이버를 탐지하는 방법은 많이 알려지지 않았다. 본 논문은 이런 유형의 루트킷을 탐지해내기 위해 OS 커널 메모리 영역을 스캔하여 모든 드라이버 관련 커널 오브젝트를 검색하고 각 오브젝트의 이중 연결리스트를 검사하여 링크가 정확한지 확인하였다. 추가로 레지스트리에서 확인할 수 있는 정보와 비교하여 드라이버 은닉 루트킷을 판별하는 방법을 제안하였다. 제안된 방법은 온라인 분석을 수행하기 때문에 메모리 덤프를 이용한 방법보다 더 빠르게 탐지할 수 있다.

Detector / Rootkit	Proposed technique	GMER	Rootkit Unhooker	Rootkit Buster	IceSword
Fu Rootkit	○	○	○	○	○
Unreal	○	○	○	○	X
BadRKDemo	○	○	○	X	X
RKUnhooker's test rootkit	○	△	X	X	X

(표 1) 탐지 능력 비교

참고문헌

[1] Greg Hoglund and Jamie Butler Author, Rootkit : Subverting the Windows Kernel, Addison Wesley, 2005, pp. 199 - 242

[2] MSDN, http://msdn.microsoft.com

[3] Woei-Jiunn Tsaur, Yuh-Chen Chen, and Being-Yu Tsai, "A New Windows Driver-Hidden Rootkit Based on Direct Kernel Object Manipulation" Lecture Notes in Computer Science, Volume 5574, 2009, pp 202-213

[4] Woei-Jiunn Tsaur and Yuh-Chen Chen, "Exploring Rootkit Detectors' Vulnerabilities Using a New Windows Hidden Driver Based Rootkit" IEEE International Conference for Social Computing, 20 10 : 842-848

[5] Woei-Jiunn Tsaur and Lo-Yao Yeh, "Identifying Rootkit Infections Using a New Windows Hidden-driver-based Rootkit" WorldComp'14

[6] Carl Jongsma, Undetectable Rootkits - Hide & Seek with your system, http://www.beskerming.com/downloadable/reports/The%20Undetectable%20Rootkit.pdf

[7] A Schuster, "Searching for processes and threads in Microsoft Windows memory dumps" 2006 6th DFRWS

[8] Mariusz Burdach, "Finding Digital Evidence In Physical Memory" Blackhat 2006

[9] SSDT 무력화에 관한 연구, http://www.hackerschool.org/HS_Boards/data/Lib_kernel/AntiSSDTHooking.pdf

[10] Windbg, http://msdn.microsoft.com/en-us/windows/hardware/hh852365

[11] RootkitBuster V5.0, http://downloadcenter.trendmicro.com/index.php?regs=NABU&clk=latest&clkval=355&lang_loc=1

[12] GMER, http://www.gmer.net

[13] Rootkit Unhooker, http://www.antirootkit.com/software/RootKit-Unhooker.htm

[14] Volatility, https://code.google.com/p/volatility/

[15] IceSword, http://icesword.en.softonic.com/