Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.31, NO.4, Aug. 2021 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2021.31.4.661

메모리 분석 우회 기법과 커널 변조 탐지 연구

이 한 얼,[1†] 김 휘 강[2‡]
[1,2]고려대학교 (대학원생, 교수)

A study on Memory Analysis Bypass Technique and Kernel Tampering Detection

Haneol Lee,[1†] Huy Kang Kim[2‡]
[1,2]Korea University (Graduate student, Professor)

요 약

커널을 변조하는 루트킷과 같은 악성코드가 만약 메모리 분석을 회피하기 위한 메커니즘을 추가하게 될 경우, 분석이 어려워지거나 불가능하게 되면서 분석가의 판단에 악영향을 미칠 수 있다. 따라서 향후 고도화된 커널 변조를 통해 탐지를 우회하는 루트킷과 같은 악성코드에 선제적으로 대응하고자 한다. 이를 위해 공격자의 관점에서 윈도우 커널에서 사용되는 주요 구조체를 분석하고, 커널 객체를 변조할 수 있는 방법을 적용하여 메모리 덤프 파일에 변조를 진행하였다. 변조 결과 널리 사용되는 메모리 분석 도구에서 탐지가 되지 않는 것을 실험을 통해 확인하였다. 이후 분석가의 관점에서 변조 저항성의 개념을 사용하여 변조를 탐지할 수 있는 소프트웨어 형태로 만들어 기존 메모리 분석 도구에서 탐지되지 않는 영역에 대해 탐지 가능함을 보인다. 본 연구를 통해 선제적으로 커널 영역에 대해 변조를 시도하고 정밀 분석이 가능하도록 인사이트를 도출하였다는 데 의의가 있다 판단된다. 하지만 정밀 분석을 위한 소프트웨어 구현에 있어 필요한 탐지 규칙을 수동으로 생성해야 한다는 한계점이 존재한다.

ABSTRACT

Malware, such as a rootkit that modifies the kernel, can adversely affect the analyst's judgment, making the analysis difficult or impossible if a mechanism to evade memory analysis is added. Therefore, we plan to preemptively respond to malware such as rootkits that bypass detection through advanced kernel modulation in the future. To this end, the main structure used in the Windows kernel was analyzed from the attacker's point of view, and a method capable of modulating the kernel object was applied to modulate the memory dump file. The result of tampering is confirmed through experimentation that it cannot be detected by memory analysis tool widely used worldwide. Then, from the analyst's point of view, using the concept of tamper resistance, it is made in the form of software that can detect tampering and shows that it is possible to detect areas that are not detected by existing memory analysis tools. Through this study, it is judged that it is meaningful in that it preemptively attempted to modulate the kernel area and derived insights to enable precise analysis. However, there is a limitation in that the necessary detection rules need to be manually created in software implementation for precise analysis.

Keywords: Kernel Tampering, Bypass Technique, Detection Methodology, Memory Analysis, Malware

Received(05. 03. 2021), Modified(06. 24. 2021), Accepted(06. 24. 2021)

† 주저자, roytravel@korea.ac.kr
‡ 교신저자, cenda@korea.ac.kr(Corresponding author)

I. 서 론

루트킷은 크게 동작하는 영역에 따라 커널 모드 루트킷과 유저 모드 루트킷으로 나뉜다. 일반적으로 루트킷은 커널 모드에서 동작하는 커널 모드 루트킷을 의미한다. 커널 모드 루트킷은 운영체제의 핵심 부분에서 동작하기 때문에 자기 자신을 은닉시키거나 커널 영역의 다른 객체들을 감시 또는 변조시킬 수 있다. 때문에 커널 모드 루트킷은 유입되어 동작하고 있다는 사실조차 탐지하기 어려운 것이 특징이다.

유저 영역에서 동작하는 루트킷의 경우 주로 SSDT(System Service Descriptor Table) 후킹을 통해 프로세스 은닉과 같은 동작을 진행한다. 하지만 SSDT는 프로세스에 독립적이고 커널 영역에서 전역적으로 사용되기에 특별한 변조를 가하지 않으면 모든 프로세스가 같은 SSDT를 가지고 있다. 때문에 다른 프로세스가 가진 SSDT와의 비교 분석을 통해 쉽게 탐지할 수 있다. 반면 커널 영역에서 동작하는 루트킷의 경우 DKOM(Direct Kernel Object Manipulation) 기법[1]을 사용하여 커널 객체를 직접적으로 변조한다. 이는 SSDT에 기술된 함수 주소를 변조하여 다른 함수를 실행하게 만드는 유저 모드 루트킷과 달리 근본적으로 변조 방법이 더 강하기에 탐지가 어렵다. 하지만 커널 루트킷은 시스템에서 곧바로 동작하여 시스템에 영향을 미치는 일반적인 악성코드와는 달리 커널 영역에서 동작하므로 시스템 재부팅을 통해 커널 영역에 적재되어야 한다. 또 근본적으로 윈도우 보호 메커니즘인 커널 패치 보호(kernel patch protection)에 의해 드라이버 서명이 되어 있는 커널 모드 소프트웨어만 커널 영역을 변조할 수 있기 때문에 시스템 침투에 있어 일반적인 악성코드에 비해 어려운 편에 속한다. SCADA와 같은 제어시스템이나 발전소와 같이 가용성이 중시되며 국가의 산업과 경제에 밀접한 영향을 주는 곳의 경우 아직도 오래된 운영체제를 사용하는 곳이 있다. 오래된 운영체제는 커널 패치 보호 기법이 없거나 드라이버 서명 없이도 루트킷이 유입되고 실행될 수 있다. 따라서 만약 이런 민감하고 취약한 시스템에 대해 루트킷이 고도화된 탐지 회피 기법을 적용했다면, 이에 대한 대응 방안이 부재하여 국가 경제의 타격이나 국가가 재산으로 취급하는 시설들에 대한 기밀이 유출될 수 있다. 따라서 이에 미리 대비하여 고도화된 탐지 회피 기법을 분석할 수 있는 연구가 필요하다.

기존에 루트킷이 취할 수 있는 방법에 대한 연구를 조사한 결과, 대부분 커널에 대한 국소적인 변조나, 탐지 확률이 높아질 수 있는 불필요한 부분의 변조 방법이 있었고 각각의 한계점들이 모두 존재 하는 것을 확인하였다. 하지만 능동적으로 공격자의 관점에서 메모리 분석을 우회하기 위한 테크닉과 정교하게 변조된 루트킷 탐지에 관한 연구가 미비한 것을 확인하였다. 따라서 본 논문에서는 기존 연구에서 미비했던 고도화된 커널 변조 테크닉에 대해 설명하고 이의 대응 방안에 대한 방법론을 적용한 커널 변조 탐지 소프트웨어를 제시한다. 앞서 언급된 시나리오와 같은 상황에 대해 선제적으로 대응하기 위해 크게 공격자와 분석가의 관점으로 나누어 연구를 진행하였다. 공격자의 관점에서는 분석가의 탐지를 우회할 수 있는 기법을 설명한다. 이를 위해 실제로 커널에서 객체를 찾는 방법을 조사하였고, 윈도우 커널 분석을 통해 루트킷에 대한 탐지를 우회할 수 있는 대부분의 주요 커널 구조체들을 찾아 정리하였다. 이후 찾은 커널 구조체들을 기반으로 우회할 수 있는 PoC를 제작하여 기존의 널리 사용되는 메모리 분석 도구에서 탐지하지 못하는 것을, 실험을 통해 확인하였다. 이후 분석가의 관점에서 분석 우회 메커니즘이 추가된 루트킷을 탐지할 수 있는 방법론을 적용하여 소프트웨어로 구현하였고, 기존에 탐지하지 못하던 소프트웨어들과 달리 회피 메커니즘이 추가된 루트킷을 탐지할 수 있음을 확인하였다.

연구 진행에 사용한 대상 시스템과 프로그램은 윈도우 7 커널 64bit, Volatility 2.6을 사용하였다. 이후 윈도우 커널에서 사용하는 주요 구조체를 전반적으로 분석하고, 이를 우회할 수 있는 방법으로 메모리 덤프 도구인 DumpIt에 대해 DLL 인젝션과 인라인 후킹을 통해 수집 중인 메모리 덤프의 커널 영역을 변조한다. 이후 변조 저항성의 개념을 이용하여 변조되어 은닉된 커널 객체를 다시 탐지할 수 있는 탐지 방법론을 설명한다. 마지막으로 탐지 방법론을 기반으로 소프트웨어로 만든 뒤, 보편적으로 사용되는 메모리 분석 도구인 Volatility에서 탐지하지 못하는 변조 영역을 탐지할 수 있음을 보인다.

1장에서 서론을 설명하고, 2장에서 관련 연구를 설명한다. 3장에서는 전반적인 이론적 배경을 설명하고, 4절에서 실험 설계를 설명한다. 5장에서는 실험 결과를 설명하고, 6장에서는 연구의 의의와 한계점을 언급한 뒤 7장에서는 결론을 이야기한다.

II. 관련 연구

본 절에서는 커널 변조, 메모리 수집 방해, 메모리 분석 방해의 세 가지 관점에서 이루어진 연구를 소개하며 각 연구가 지니는 단점에 관해 설명한다.

2.1 Kernel Modification

프로세스 카빙을 방해하기 위해 POOL_HEADER 구조체의 PoolTag 변수를 변조하는 방법과 DISPATCHER_HEADER 구조체 내에 있는 운영체제 타입과 비트를 나타내는 시그니처 값을 변조함으로써 분석을 실패로 이끄는 연구가 진행된 바 있다[16]. 하지만 이는 분석 자체를 실패로 이끌기 때문에 분석가로 하여금 의심을 사게 되고 후속 조치를 통해 분석이 가능하다는 것과, 구체적인 커널 객체를 은폐 시킬 수 없다는 단점이 존재한다.

2.2 Anti Memory Dump

메모리 분석을 위해 선행되어야 할 메모리 수집에 있어 메모리 수집 도구는 항상 같은 이름을 가진다는 것에 기반하여 메모리 수집 도구가 확인되면 해당 프로세스를 종료시키는 방법과 수집을 하는 데 필요한 커널 드라이버에 대한 설치를 막는 방안에 관한 연구가 진행된 바 있다[17]. 하지만 마찬가지로 수집 도구의 이름을 바꾸는 간단한 후속조치를 통해 대응 가능하며, 드라이버 설치를 막기 위해 만든 드라이버 자신을 은폐할 수 없다는 것과, 32bit에서만 동작한다는 단점이 존재한다. 메모리 수집에서 많이 사용되는 함수인 MmGetPhysicalMemoryRanges라는 메모리를 나열하는 함수의 동작에 관여하여 온전한 물리 메모리 수집을 방해하는 방법이 연구된 바 있다[2]. 해당 함수는 MmPhysicalMemoryBlock라는 변수에서 물리 메모리 나열 범위 정보를 얻어오며, 이와 관련된 구조체를 분석하고 물리 메모리 나열 범위를 조작하여 유저가 원하는 특정 범위의 페이지를 덤프가 되지 않게 하는 방법이다. 하지만 이의 경우 페이지 단위로 메모리를 덤프하기 때문에 페이지 내에 정밀 변조에 필요한 필수 영역 이외의 부분이 손실된다는 단점이 존재한다.

2.3 Anti Memory Analysis

미니 필터 드라이버를 이용하여 메모리 수집 도구에서 사용하는 함수인 NtWriteFile가 호출된 경우, 인라인 후킹을 통해 수집될 때 여러 커널 객체들을 변조하는 연구가 진행된 바 있다[17]. 이 연구 역시 공격을 위해 만들었던 드라이버 자체를 은닉시킬 수 없다는 단점이 존재한다. 일반적으로 특정 커널 객체를 변조하는 방식과는 대조적으로 수집된 메모리에 인공의 커널 객체를 만들어 삽입하여 분석을 방해하는 연구도 진행된 바 있다[18]. 이의 경우에도 마찬가지로 인공의 커널 객체를 통해 분석에 혼란을 주기는 하나 결과적으로 적절한 시간만 투입된다면 분석가는 분석이 가능하다는 한계점이 있다. 기존의 메모리 분석기법을 설명하고, 안티 메모리 분석에 있어 변조가 불가능한 데이터 필드를 이용하여 메모리를 분석하는 새로운 대응 기법을 제안하는 연구가 진행된 바 있다[1]. 하지만 공격에 사용될 수 있는 구체적인 주요 커널 구조체와 변수에 대한 예시나 탐지할 수 있는 구체적인 방법론에 대한 예시가 부재하다는 단점이 있다. 관련 연구 조사 결과 대부분의 경우 커널 객체를 변조 및 삽입이 가능하지만 후속조치를 통해 분석이 가능하다는 것과, 변조의 흔적을 은폐시킬 수 없다는 점이다. 때문에 고도화된 공격에 대응하기 위해서는 커널 객체를 변조 및 삽입을 하더라도 분석가가 알아차릴 수 없어야 하며, 변조에 사용된 흔적을 은폐시킬 수 있는 방안에 대한 연구가 필요하다.

III. 윈도우 커널과 메모리 분석의 이론적 배경

3.1 메모리 분석 우회 방법론과 특징

공격자는 메모리 분석을 우회하기 위해 방법은 크게 시간 축을 기준으로 수집 방해와 분석 방해로 나뉜다. 수집 방해의 경우 분석가가 메모리 분석을 위해 대상 시스템의 메모리를 수집하는 과정을 방해하는 것으로, 분석가가 덤프 파일을 생성하지 못하도록 하는 것이다. 분석 방해의 경우 분석가가 메모리 분석을 위해 대상 시스템의 메모리 수집을 완료한 이후 분석을 진행할 때 방해하는 것으로, 메모리에 불필요한 인공의 커널 객체를 만들어 넣거나[18], 또는 정밀하게 커널 객체를 변조하여 분석을 어렵게 또는 불가능하게 만든다. 공격자는 메모리 수집 전, 중, 후에 따라 분석가의 메모리 수집 및 분석을 방해할 수

있다. 하지만 수집 도중에 개입하여 메모리 덤프를 생성하지 못하도록 방해하면 분석가는 즉각적으로 시스템 내부에 이상이 있음을 알아차리고 추가적인 후속 조치를 통해 메모리 덤프를 수집할 수 있게 된다. 또한 분석 과정에 개입하여 인공의 커널 객체를 만드는 방법을 사용 할 경우 분석가의 분석에 혼란을 주어 분석 시간을 지속시킬 수는 있다. 하지만 결과적으로 분석 시간을 지연시킬뿐, 인위적으로 생성된 커널 객체가 있음을 탐지해 낼 수 있다. 따라서 공격자는 분석가가 수집 과정과 분석 과정 모두에서 알아챌 수 없도록, 메모리 수집을 하고 분석을 한다고 하더라도 변조된 흔적을 분석가가 탐지해낼 수 없도록 하여야 한다. 연구 목적 또한 기존의 존재하는 방법으로 탐지할 수 없는 공격 기법에 대해, 이를 다시 탐지하는 방법을 제시하는 것이다. 따라서 메모리 분석을 진행하더라도 변조된 흔적을 탐지할 수 없도록 하고자 한다.

3.2 윈도우 커널과 메모리 구조

윈도우는 가상 메모리 시스템을 사용하기 때문에 메모리를 분석하기 위해서는 가상주소에서 실제 물리주소로의 변환이 필요하다. 가상 메모리 시스템의 구조는 운영체제의 비트 수(32/64)에 따라 각각 최대 4GB, 16EB의 크기를 가지며, 크게 커널 영역과 유저 영역으로 나뉜다. 커널 영역은 운영체제인 커널과 드라이버가 사용하며 접근 가능한 공간이며, 유저 영역은 사용자 프로세스나 DLL이 사용하며 접근 가능한 공간이다. 각각의 프로세스는 자신만의 가상주소 공간을 가지고 있으며, 커널은 필요시 프로세스와 같은 커널 객체의 가상주소를 물리주소로 변환[19]하여 작업을 수행한다. 주소 변환은 메모리 관리 장치인 MMU(Memory Management Unit)에 의해 수행된다. 주소 변환 메커니즘은 공격자의 관점에서 정교한 커널 객체 변조를 위해 알아야 하며 분석가의 관점에서 변조한 커널 객체를 탐지하기 위해 공통으로 알아야 할 사항이다. 따라서 이를 활용하기 위해 주소 변환 메커니즘을 직접 구현하는 과정이 필요하고, 구현을 위해 DTB(Directory Table Base)와 Intel Architecture에 대한 이해가 필요하다.

3.3 Directory Table Base (DTB)

DTB는 윈도우에서 실행되는 프로세스가 독립적인 주소 공간을 가지기 위해 사용된다. DTB는 프로세스별로 독립적인 주소 공간을 보장하기 위해 프로세스마다 다른 값을 가지는 것이 특징이다. 또한 DTB는 Context Switching시 CR3 레지스터에 로드되어, 프로세스에 존재하는 쓰레드마다 접근하여 주소 변환에 사용할 수 있다. 또한 프로세스를 관리하는 EPROCESS 구조체 내의 KPROCESS 구조체의 DirectoryTableBase 변수에도 저장되어 있어 필요시 접근하여 주소 변환에 사용할 수 있다. 커널 객체가 사용하는 커널 주소 공간을 물리주소로 변환하기 위해서는 커널에서만 사용하는 일종의 전역 DTB가 필요하며 이는 "Idle" 프로세스의 EPROCESS 구조체 내의 KPROCESS 구조체의 DirectoryTableBase 변수로부터 얻을 수 있다[19].

3.4 Intel Architecture

인텔 아키텍처는 32/64bit에 따라 Intel IA-32와 IA-32e로 나뉜다. 연구에서는 Intel IA-32e를 기준으로 하였으며 이후 이를 기준으로 설명한다.

3.4.1 Address Translation Mechanism

인텔 아키텍처에서 메모리 주소는 가상주소, 선형주소, 물리주소가 있다. 가상주소에서 물리주소로의 변환은 크게 Segmentation과 Paging 과정을 거친다. Segmentation 과정 이후 선형 주소로 변환되며, 변환된 선형 주소는 Paging 과정을 거쳐 물리 주소에 접근하게 된다. 과정을 도식화한 것은 아래 [Fig. 1]과 같다[21].

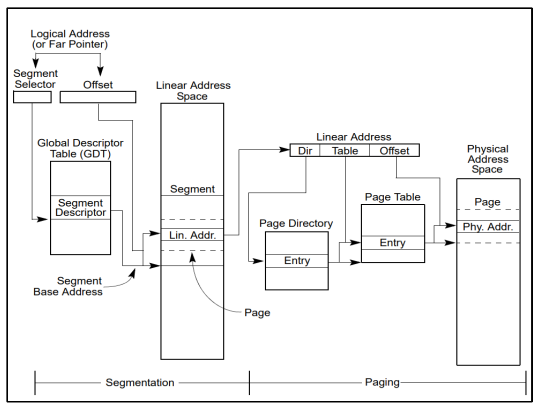

Fig. 1. Address translation process

물리주소로의 변환을 위해 Segmentation 과정 이후 CR3 레지스터를 참조하여 DTB를 가져온다. 가져온 DTB를 사용하여 변환 테이블의 시작 주소를 얻고, 변환 테이블의 엔트리 인덱스를 계산하며 다음 변환 테이블의 물리주소를 얻는다. 이렇게 가상주소 영역에서 각 테이블의 엔트리 인덱스를 나타내는 부분을 페이지 테이블의 엔트리에 대입하여 계속 참조하면 찾고자 하는 커널 객체가 위치한 물리주소를 얻을 수 있다.

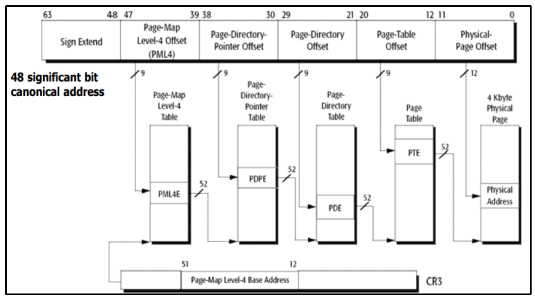

Fig. 2. Physical address calculation process

선형 주소는 비트 단위로 총 5단계로 나뉘어 있다. Page Map Level 4 Table을 참조하여 해당 Table의 엔트리가 위치하는 주소를 찾고 엔트리가 가리키는 포인터를 통해 Page Directory Pointer Table로 이동한다. 이후 동일한 과정을 거쳐 Page Directory Table, Page Table, Physical Page를 찾은 다음 마지막으로 실제 물리 주소가 위치하는 주소인 Physical Address를 얻게 된다[20]. 이 과정은 PAGE의 크기인 4KB, 2MB, 1GB에 따라 변환 과정에 PAE(Physical Address Extension)와 같이 고려해야 할 추가적인 요소가 있으나 전반적인 메커니즘은 동일하다.

3.5 윈도우 커널 구조

3.5.1 커널 전역변수

윈도우에서 사용하는 대부분의 주요 커널 구조체를 찾기 위해서는 먼저 커널 전역변수 정보가 존재하는 KDDEBUGGER_DATA64 구조체에 접근해야 한다. 이는 문자열 “KDBG”를 기준으로 시그니처 스캔을 통해 찾을 수 있다. 이후 PsActiveProcessHead, PsLoadedModuleList 변수를 통해 프로세스 목록이나 모듈 목록을 구할 수 있고, 또한 KernBase, KiProcessBlock와 같은 주요 커널 변수와 프로세서 제어와 관련된 정보를 가진 KPCR 구조체 등을 찾을 수 있다[1].

3.5.2 프로세스 구조체

윈도우에서 프로세스 관리는 EPROCESS 구조체를 통해 이뤄진다. EPROCESS 구조체의 시작은 KDDEBUGGER_DATA64 구조체의 커널 변수 PsActiveProcssHead를 참조하여 얻을 수 있다. EPROCESS 구조체는 프로세스마다 각각 존재하며, 프로세스가 실행될 때 가장 먼저 생성된다. EPROCESS 구조체 내에 프로세스, 쓰레드, DLL 등의 주요 정보들이 담겨 있으며 내부의 KPROCESS 구조체를 참조하여 DTB 또한 얻을 수 있다.

3.5.3 로드된 모듈 목록

로드된 모듈은 커널 영역에서 커널 드라이버, 사용자 영역에서 DLL을 의미한다. 로드된 모듈은 KDDEBUGGER_DATA64 구조체의 커널 변수인 PsLoadedModuleList를 참조하여 구할 수 있다. 정확히는 LDR_DATA_TABLE_ENTRY 구조체 내에 로드된 모듈에 대한 정보가 정의되어 있고, 커널 변수 PsLoadedModuleList가 이를 가리키고 있다. 따라서 이를 가리키는 링크드 리스트를 참조하면 커널에 적재된 모듈 리스트를 얻을 수 있다.

3.5.4 메모리 풀

메모리 풀은 윈도우가 메모리를 관리하기 위해 시스템 초기화 당시 만들어지는 영역이다. 풀은 POOL_DESCRIPTOR 구조체에 의해 정의되며, 정의된 풀 영역은 POOL_HEADER 구조체에 의해 관리된다. 풀의 종류는 2가지로 Paged Pool, Non-Paged Pool이 있다. Paged Pool의 경우 실제 메모리에서 제거되어 디스크에 페이징 되어 저장되는 풀이며, Non-Paged Pool의 경우 디스크에 저장되지 않고 항상 실제 메모리에 상주하여 액세스할 수 있는 풀을 의미한다. 풀에 데이터를 할당할 때는 ExAllocatePoolWith 함수를 사용하고, 할당되는 데이터를 청크라 부른다. 청크들은 POOL_HEADER 구조체로 관리된다. 할당된 청크마다 POOL_HEAD

ER 구조체를 가지며, 구조체 내에는 메모리 풀의 크기나 종류를 식별할 수 있는 태그가 있다. 태그는 프로세스, 쓰레드, 네트워크, 드라이버 등의 커널 객체에 따라 각기 다른 4바이트의 아스키 문자가 지정되며, 메모리 분석에서 풀 태그 스캔 방식을 통해 POOL_HEADER 구조체를 찾는다.

3.5.5 커널 구조체 분석 방법론

커널 영역에서 커널 객체를 찾는 방법은 크게 링크드 리스트를 탐색하는 방법과 커널 구조체를 카빙하는 방법 둘로 나뉘며, 커널 구조체를 카빙 하는 방법은 다시 구조체 내 필드 값을 이용한 카빙 방법과 풀 헤더를 이용한 카빙 방식으로 나뉜다[1]. 링크드 리스트 탐색의 경우 LIST_ENTRY 구조체 내에 존재하는 Flink와 Blink 변수가 가진 메모리 주소를 따라가면서 커널 객체를 얻는 방법이다. 대표적으로 윈도우 작업 관리자가 이와 같은 방식으로 구현되어 있다. 메모리 주소를 따라가기에 커널 구조체 카빙 방식보다 탐색 속도가 빠르다는 장점이 있다. 반면 커널 구조체 카빙 방식은 커널에서 사용되는 구조체가 운영체제 버전별로 동일한 구조를 가진다는 점을 이용한 방법이다. 풀 헤더를 이용한 카빙 방식은 POOL_HEADER 구조체 내에 존재하는 풀 태그 값을 시그니처로 활용하여 커널 객체를 찾는 것이다. 커널 구조체 카빙 방식의 경우 메모리 주소를 따라가는 것이 아닌, 찾고자 하는 영역에 대한 카빙 방식이기에 상대적으로 탐색 속도가 느린 것이 특징이다.

3.5.6 루트킷 특징

가장 널리 알려진 루트킷의 동작 방식은 프로세스를 은닉하는 것이다. 커널 영역 또는 유저 영역을 변조하여 은닉하고자 하는 프로세스의 앞, 뒤에 존재하는 프로세스의 링크를 변조하는 방식이며 아래 [Fig. 3]과 같다.

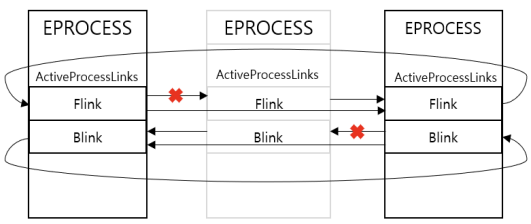

Fig. 3. Linked List Traversal Process

EPROCESS 구조체의 ActiveProcessLinks 변수는 LIST_ENTRY 구조체를 가리키고 있다. LIST_ENTRY 구조체는 위 그림과 같이 Flink와 Blink의 두 변수를 가진다. Flink는 다음 프로세스의 주소값을 얻기 위해 사용되며, Blink는 이전 프로세스의 주소값을 얻기 위해 사용된다. 이와 같은 특성을 활용하여 은닉하고자 하는 프로세스의 이전 프로세스의 Flink를, 은닉하고자 하는 프로세스의 다음 프로세스를 가리키도록 링크를 변조한다. 이후 은닉하고자 하는 프로세스의 다음 프로세스의 Blink를, 은닉하고자 하는 프로세스의 이전 프로세스의 Blink를 가리키도록 변조하면 결과적으로 프로세스 목록상에서 사라지게 되어 일반적인 방법으로는 프로세스를 찾을 수 없게 되고, 메모리 분석을 통해서만 찾을 수 있게 된다.

IV. 실험 설계

실험을 위한 전체 아키텍처 구성은 [Fig. 4]와 같다. 크게 공격자의 관점과 분석가의 관점에서 나누어 진행된다. 메모리 분석을 위해 분석가는 메모리 수집을 진행하고 이후 메모리 분석의 과정을 거친다. 공격자는 메모리를 수집하는 단계에서 수집되고 있는 데이터에 포함된 커널 영역의 데이터 변조를 진행한다. 이때, 공격자가 진행할 수 있는 시나리오는 아래 [Table 1]과 같다. 시나리오 1의 경우 만약 공격자가 메모리 수집을 방해하면 분석가는 즉각적으로 시스템 내부에 이상이 있음을 알아차리고 후속 조치를 통해 메모리 덤프를 수집할 수 있게 되어 공격에 실패하게 된다. 시나리오 2의 경우 메모리 수집을 통한 메모리 덤프 파일을 생성 후, 덤프 파일을 변조하는 경우 생성 시각에 계산된 해시값과 변조된 이후의 해시값 비교를 통해 탐지할 수 있다. 시나리오 3의 경우 인공의 커널 객체를 삽입하는 방법이다. 하지만 이 또한 분석 시간을 지연시킬 뿐 커널 객체와 연결된 다른 객체와의 관계 분석을 통해 탐지할 수 있다. 시나리오 4의 경우 변조하고자 하는 커널 객체를 가리키는 정보와 해당 커널 객체와 관련된 정보에 대한 정교한 변조를 진행할 경우 실제 정밀 분석을 진행하더라도 탐지가 어려워진다. 이는 본질적으로 연산의 기본 단위인 비트에 대한 정보를 변조하기 때문이다. DKOM(Direct Kernel Object Manipulation)이 이러한 특징을 이용하는 공격 기법이다. 하지만 기존에 알려진 DKOM은 커널 영역을 직접적으로

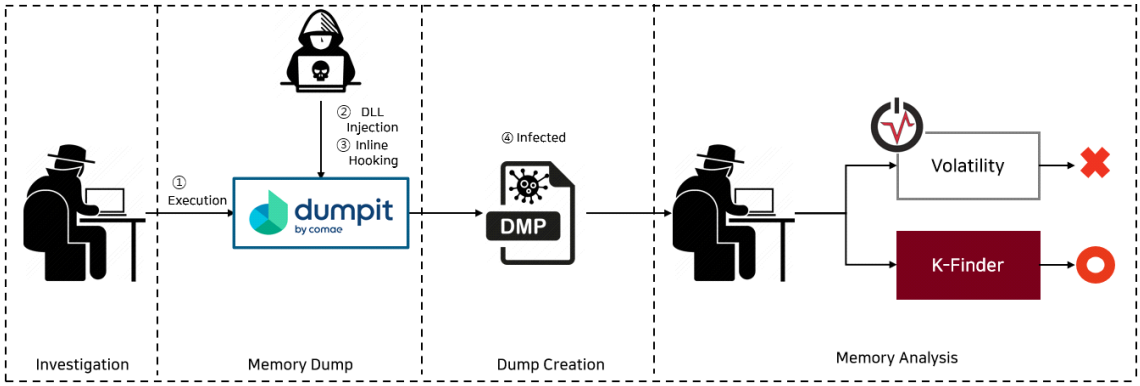

Fig. 4. Overall process on modifying memory dump and analyzing it correctly

변조하지만, 커널 객체들에 대한 교차 분석 과정을 통해 탐지할 수 있다. 예를 들어 EPROCESS 구조체의 ActiveProcessHead의 링크를 변조하여 프로세스를 은닉시킨 경우, 프로세스를 찾을 수 있는 또 다른 방법인 EPROCESS 구조체 내에 존재하는 SessionProcessLink 변수의 링크 정보와 ActiveProcessHead 변수의 링크 정보 간의 교차 검증을 통해 탐지하거나 프로세스 정보를 나타내는 POOL_HEADER 구조체 영역을 찾음으로써 은폐된 프로세스를 찾을 수 있다. 본 논문에서는 DKOM에 대한 기존의 탐지 방안을 무력화시키기 위해 커널을 이루는 주요 구조체들에 대해 분석과 정리를 진행하고, 교차 검증을 진행할 수 있는 커널 구조체와 변수를 찾아 공격 벡터로 삼는다. 이후 시나리오 4와 같이 메모리 수집 과정에 개입하여 메모리 수집 도구를 후킹하고 커널 객체에 해당하는 데이터에 대해 변조를 진행한 다음, 실제 널리 사용되는 메모리 분석 도구에서 탐지되지 않음을 보인다. 이후 이를 탐지할 수 있는 소프트웨어를 개발하여 고도화된 공격에도 불구하고 탐지 가능함을 보이고, 객체 변조 탐지의 본질적 한계를 설명한다.

4.1 커널 구조체 분석

모든 커널 구조체는 변조 대상이 될 수 있다. 실험을 위해 추출한 주요 커널 구조체와, 구조체 내부 주요 변수는 아래 [Table 2]와 같다. 분석 대상을 기준으로 크게 OS, 프로세스, DLL/쓰레드, 네트워크, 인젝션, 드라이버, 레지스트리, 콜백, 파일로 나누었다. 분석 대상별 사용되는 주요 커널 구조체를 나타내었으며, 커널 구조체별 주요 변수들을 정리하였다. 또한 비고에는 주요 변수가 가지는 의미를 표현하거나 해당 커널 구조체가 가지는 풀 태그 값을 정리하였다.

4.2 메모리 분석 도구 분석

메모리 분석 도구는 가장 널리 사용되고 있는 Volatility를 기준으로 분석을 진행하였다. Volatility는 파이썬으로 작성된 오픈소스 프레임워크로 메모리 분석에 있어 일반적으로 널리 사용된다. 윈도우를 기준으로 약 100여 개의 메모리 분석 모듈을 사용할

Table 1. Possible scenario lists for applying proposed attack method

Type	Point of attack	Attack method	Detection method
Scenario 1	Before/during/after memory collection	Interfering with the creation of a memory dump file	Dump with follow-up
Scenario 2	After collecting memory	Tampering after creating a memory dump file	Detection through hash comparison
Scenario 3	During/after memory collection	Insert kernel object into memory dump file	Kernel object association analysis
Scenario 4	Collecting memory	Memory dump file advanced tampering	-

Table 2. Type of mainly used kernel structure to compromise the kernel

Section	Kernel Structure	Main variable		Remark
OS	KDDEBUGGER_DATA64 (KDBG)	PsActiveProcessHead		Process list
		PsLoadedModuleList		Module list
		PspCidTable		Process list
		KiProcessorBlock		Global variable
		KernBase		Global variable
	DBKGD_DEBUG_DATA_HEADER64	OwnerTag		Kernel signature
	KPRCB	Major Version	Minor Version	Kernel version
	KPCR	KdVersionBlock		debugging version
		SelfPcr		KPCR structure
	DBGKD_GET_VERSION64	KernBase		Global Variable
		PsLoadedModuleList		Module list
		DebuggerDataList		KDBG structure
PROCESS	POOL_HEADER	PoolTag		Proã(50 72 6F E3)
	OBJECT_HEADER	TypeIndex		Object type
	EPROCESS	ActiveProcessLinks		Process list
		SessionProcessLinks		Process list
		Win32Process		Process(Self)
		ExitTime		Process exit time
		ObjectTable		Process list
		ImageFileName		Process name
		UniqueProcessId		Process ID
		InheritedFromUniqueProcessId		Parent process ID
	KPROCESS	DirectoryTableBase		CR3 Register
		ProcessListEntry		Process list
		ThreadListHead		Thread list
	PEB	ImageBaseAddress		Process ImageBase
		ldr		PEB_LDR_DATA
	HANDLE_TABLE	QuotaProcess		Process list
		UniqueProcessId		Process ID
		HandleTableList		Process list
	tagPROCESSINFO	Process		Process list
DLL/ THREAD	PEB_LDR_DATA	InLoadOrderModuleList		DLL list
		InMemoryOrderModuleList		DLL list
		InInitializationOrderModuleList		DLL list
	RTL_USER_PROCESS _PARAMETERS	DllPath		DLL path
		ImagePathName		DLL name
		CommandLine		Command line
	ETHREAD	Tcb	ThreadListEntry	Thread list
	KTHREAD	Teb		Thread Information
	LDR_DATA_TABLE_ENTRY	DllBase		MmLd(4D 6D 4C 64)
		FullDllName	BaseDllName	
	tagTHREADINFO	pETHREAD	ppi	Thread
NETWORK	TCP_ENDPOINT	AddrInfo		TcpE(54 63 70 45)
		LocalPort	RemotePort	
	TCP_LISTENER	LocalAddr	Port	TcpL(54 63 70 4C)
	UDP_ENDPOINT	LocalAddr	Port	UdpA(55 64 70 41)
INJECTION	MM_AVL_TABLE	BalancedRoot		Page attribute
	MM_ADDRESS_NODE	Tag	RightChild	Page attribute
DRIVER	DRIVER_OBJECT	DriverName	DriverStart	Driver(44 72 69 F6)
REGISTRY	CMHIVE	Hive	HiveList	CM10(43 4D 31 30)
CALLBACK	NOTIFICATION_PACKET	ListEntry		IoFs(49 6F 46 73)
	SHUTDOWN_PACKET	Entry		IoSh(49 6F 53 68)
	GENERIC_CALLBACK	Callback		Cbrb(43 62 72 62)
FILE	FILE_OBJECT	Filename		File(46 69 6C E5)

수 있고, 실험을 위해 주로 사용되는 모듈 목록을 추출하여 공격 대상으로 선정하였다.

4.3 메모리 수집 도구 후킹

메모리 수집 도구의 경우 메모리 수집 도구 중 수집 방법이 비교적 간단해서 주로 사용되는 DumpIt을 공격 대상으로 선정하였다. 이후 정적 분석을 진행한 결과, DumpIt에서 메모리 수집을 위해 DeviceIoControl 함수를 통해 드라이버 파일과 통신하는 것을 확인하였다. 이를 통해 DumpIt에 대해 DLL Injection을 진행하여 인라인 후킹을 통해 내부 드라이버 파일과 통신하는 DeviceIoControl 함수를 후킹하고 생성 중인 덤프 파일이 가지고 있는 커널 영역에 대한 변조를 진행하였다.

4.4 탐지 방법론

커널 변조를 탐지하는 기존 방법은 변조된 영역을 가리키거나 참조하는 다른 커널 구조체 변수와의 상호의존적 특성을 이용하여 교차 검증을 진행하여 은닉되거나 변조된 커널 객체가 있는지 탐지하는 방식이다. 하지만 교차 검증을 이용한 방법은 교차 검증에 사용되는 다른 영역을 마찬가지로 변조할 경우 탐지할 수 없게 되기에 여전히 본질적으로 취약한 구조이다. 따라서 교차 검증에 사용되는 주요 영역이 전부 변조되더라도 탐지하는 방법을 제시하고자 한다.

커널에서 객체를 찾는 방법은 주소값 정보를 가진 링크드 리스트를 파싱하거나, 커널 구조체를 카빙 하는 방법으로 나뉜다. 커널 구조체를 카빙 하는 방법은 다시 구조체 내 필드 값을 이용한 카빙 방식과 풀 헤더를 이용한 카빙 방식으로 나뉜다. 링크드 리스트를 파싱하여 커널 객체를 찾는 방법의 경우 커널 객체를 가리키는 모든 주소값이 변조되면 더 이상 찾을 수 없다는 한계가 존재한다. 풀 헤더를 이용한 카빙 방법 또한 마찬가지로 풀 헤더에 사용된 풀 태그 값을 전부 찾아 변조할 경우 더 이상 커널 객체를 찾을 수 없게 된다. 따라서 고려 가능한 방법은 구조체 내 필드 값을 이용한 카빙 방식이다. 이는 특정 구조체는 일정한 크기를 가지며 구조체 내의 변수가 가지는 값이 고정되거나 일정한 범위를 가진다는 점을 이용한 것이다. 이와 같은 방법론의 적용 전제는 구조체 내의 변수 중 고정되거나 일정한 범위를 가지는 변수를 변조할 경우 쉽게 탐지할 수 있기에, 고도화된 커널 변조를 위해서는 변조 저항성이 큰 변수는 공격하지 않는 것이다. 변조 저항성이 큰 변수의 예시 중 하나로는, 여러 구조체 내의 변수에 저장된 주소값이다. 주소값에 대해 현재 64bit 중 48bit만 사용된다는 점을 이용하여 나머지 변하지 않는 16bit 고정값인 “FF FF”를 기준으로 만들고 이외의 추가적인 여러 개의 고정값을 And로 결합하여 탐지 규칙을 만든다면, 은폐되거나 변조된 커널 객체를 찾을 수 있게 된다. 변조 저항성이 큰 변수를 찾기 위해 MSDN에서 공개한 구조체 정의를 참고하고 정의되지 않은 부분은 휴리스틱을 통해 값의 범위를 파악하였다. 이를 통해 여러 개의 변수를 찾고 이들의 조합으로 다양한 규칙을 생성할 수 있으며, 찾은 변조 저항성이 큰 변수의 값을 아래 〔Table 3〕에 정리하였다.

Table 3. Rules to detect advanced kernel tampering

Section	Standard structure	Index	Value	Rule
PROCESS	EPROCESS	A. 〔0x00:0x08〕	“03 00 58 00 00 00 00 00“	((A | B) & (C & D)) & (!E & !F)
		B. 〔0x00:0x08〕	“03 00 58 00 01 00 00 00”	
		C. 〔0x0D:0x0F〕	“FF FF“	
		D. 〔0x16:0x18〕	“FF FF”	
		E. -〔0x3C:0x38〕	“50 72 6F E3”	
		F. -〔0x5C:0x58〕	“50 72 6F E3”	
NETWORK	TCP_ENDPOINT	G. 〔0x00:0x10〕	“00 00 00 00 00 00 00 00 02 00 00 00 00 00 00 00”	G & H & I & J
		H. 〔0x16:0x18〕	“FF FF”	
		I. 〔0x104:0x108〕	“04 00 00 00”	
		J. 〔0x23E:0x240〕	“FF FF”	
FILE	FILE_OBJECT	K. 〔0x00:0x08〕	“05 00 D8 00 00 00 00 00”	(K & L & M) & (!N | !O)
		L. -〔0x60:0x58〕	“00 04 00 00 80 01 00 00“	
		M. -〔0x18:0x15〕	“1C 00 0C”	
		N. -〔0x6F:0x6C〕	“00 15 02”	
		O. -〔0x6C:0x68〕	“46 69 6C E5”	

프로세스, 네트워크, 파일과 같이 분석에 주로 사용되는 대표적인 커널 객체를 대상으로 규칙을 생성하였다. 예를 들어 변조 및 은폐된 프로세스를 찾기 위해서는 EPROCESS 구조체를 기준으로 8바이트 값이 "03 00 58 00 00 00 00 00"이거나 "03 00 5 8 00 01 00 00 00"이어야 한다. 이는 운영체제 비트 수를 나타내는 값으로 모든 프로세스가 객체가 동일 값을 가진다. 또한 0xD 바이트 이후와 0x16 바이트 이후는 "FF FF"가 되어야 하며 이는 가상 메모리를 사용하는 윈도우의 특성상 고정되어 변하지 않는 값이다. 마지막으로 -0x3C 또는 -0x5C 이후의 값인 "50 72 6F E3"은 프로세스를 의미하는 풀 태그 값으로 마찬가지로 변하지 않는 값이다. 변하면 의미가 바뀌어 쉽게 탐지되는 값을 변조 저항성이 큰 변수라 하며 이를 조합하여 변조 및 은폐된 객체를 찾을 수 있다.

V. 실험 결과

5.1 변조 우회 결과

Windows 7 64bit 커널에 대해 가상환경을 구축하였다. 이후 〔Table 2〕에 정리된 주요 커널 구조체를 공격 대상으로 삼고, 메모리 수집 도구를 후킹 하여 해당 구조체를 변경한 결과는 아래 〔Table 4〕와 같다. 영역을 크게 OS, Process, DLL/Thread/Handle, Network, Injection, Driver, File, Registry, Callback, Rootkit으로 나누었고 구분된 영역을 분석하는 Volatility의 모듈과 이에 대한 우회 방법을 적용한 결과이다. 비고의 표시는 우회 결과를 나타내는 것으로 Bypass Method에 기술된 방법으로 변조하여 Volatility 모듈에서 탐지할 수 없음을 확인하였다. 〔Table 4〕에 표시되지 않은 모듈 또한 이와 같은 방법을 적용하여 모두 변조 가능하다.

5.2 탐지 소프트웨어 구현

〔Table 3〕에 기술된 규칙을 조합하여 은폐된 객체를 찾기 위한 소프트웨어를 구현한 결과는 〔Fig. 7〕, 〔Fig. 8〕, 〔Fig. 9〕와 같다. 이를 위해 가상 환경의 메모리를 수집/변조한 덤프 파일을 사용하였으며, 크게 프로세스, 네트워크, 파일의 3개의 영역에 대해 변조되고 은닉된 커널 객체를 탐지할 수 있도록

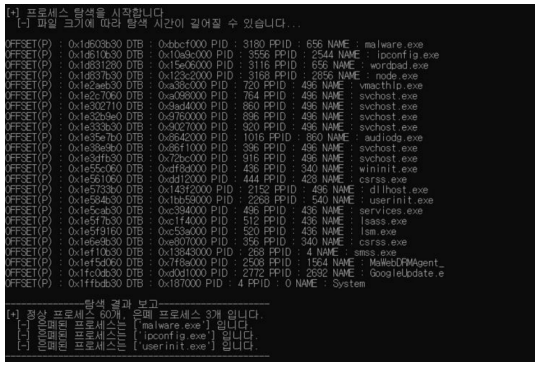

Fig. 5. K-Finder: Procscan

〔Table 3〕의 규칙 K, L, M, N을 조합하였고, 은폐된 파일 객체를 찾는 탐지 도구로 의도적으로 은폐한 파일을 찾을 수 있음을 확인하였다.

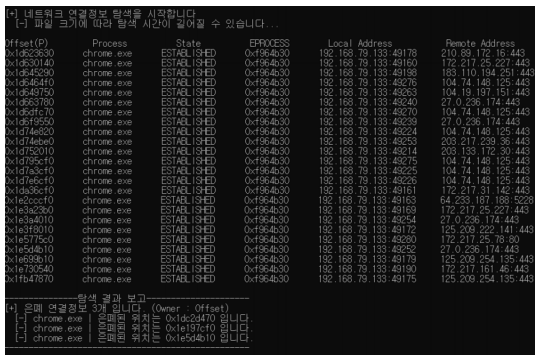

Fig. 6. K-Finder: Netscan

〔Table 3〕의 규칙 G, H, I, J를 조합하였고, 은폐된 네트워크 연결 정보를 찾는 탐지 도구로 의도적으로 은폐한 연결정보를 찾을 수 있음을 확인하였다.

Fig. 7. K-Finder: Filescan

〔Table 3〕의 규칙 A, B, C, D, E를 조합하였고, 공격자의 입장에서 의도적으로 은폐한 프로세스를 찾을 수 있음을 확인하였다.

Table 4. Attack method to bypass existing memory analysis and the result of bypassing

Section	Module	Bypass method	T/F
OS	imageinfo	Modify the "KDBG" signature and DTB of Idle process	T
	kdbgscan	Modify the "KDBG" signature	T
	kpcrscan	Modify the KdVersionBlock variable of KPCR	T
PROCESS	pslist	Modify the PsActiveProcessHead variable of the KDDEBUGGER_DATA64	T
	psscan	Modify the pool tag Proã(50 72 6F E3) of the POOL_HEADER	T
	pstree	Modify the InheritedFromUniqueProcessId variable in the EPROCESS	T
DLL/ THREAD/ HANDLE	dlllist	Modify the PEB_LDR_DATA address and LDR_DATA_TABLE_ETNRY variables InLoadOrderLinks, InMemoryOrderLinks, InInitializationOrderLinks	T
	dlldump	Modify the PEB_LDR_DATA address and LDR_DATA_TABLE_ETNRY variables InLoadOrderLinks, InMemoryOrderLinks, InInitializationOrderLinks	T
	handles	Modify the ObjectTable variable of EPROCESS and dlllist	T
	privs	Modify the Token variable of the EPROCESS	T
	vadinfo	Modify the RightChild variable of the MMADDRESS_NODE	T
	vaddump	Modify the RightChild variable of the MMADDRESS_NODE	T
	moddump	Modify the Dllbase variable of the LDR_DATA_TABLE_ENTRY	T
	ldrmodules	Modify the Dllbase variable of LDR_DATA_TABLE_ENTRY and RightChild variable of MM_ADDRESS_NODE	T
NETWORK	netscan	Modify the pool tag TcpL(4C 70 63 54) of the TCP_LISTENER Modify the pool tag TcpE(45 70 63 54) of the TCP_ENDPOINT Modify the pool tag UdpA(41 70 64 55) of the UDP_ENDPOINT	T
INJECTION	malfind	Modify the Tag variable of the MMADDRESS_NODE	T
DRIVER	modules	Modify the PsLoadedModuleList variable of the KDDEBUGGER_DATA64	T
	drvierscan	Modify the pool tag Driö (44 72 69 F6) of the DRIVER_OBJECT	T
FILE	filescan	Modify the pool tag Filå(46 69 6C E5) of the FILE_OBJECT	T
REGISTRY	hivescan	Modify the pool tag CM10(30 31 4D 43) of the CMHIVE and Modify the registry startup signature E0 BE E0 BE	T
	hivelist	Modify the pool tag CM10(30 31 4D 43) of the CMHIVE and Modify the registry startup signature E0 BE E0 BE	T
	userassist	Modify the pool tag CM10(30 31 4D 43) of the CMHIVE	T
CALLBACK	callbacks	Modify the pool tag IoFs(73 46 6F 49) of the NOTIFICATION_PACKET Modify the pool tag IoSh(68 53 6F 49) of the SHUTDOWN_PACKET Modify the pool tag Cbrb(62 72 62 43) of the GENERIC_CALLBACK	T
ROOTKIT	psxview	Modify the EPROCESS variables ActiveProcessLinks, SessionProcessLinks, Win32Process, ExitTime and OBJECT_HEADER's TypeIndex variable and POOL_HEADER's pool tag Proã(50 72 6F E3)	T
	procdump	Modify the ImageBaseAddress variable of PEB sturcture and DTB	T
	apihooks	Modify the starting address of the LDR_DATA_TABLE_ENTRY structure	T
	modscan	LDR_DATA_TABLE_ETNRY	T

구현하였다. 규칙의 조합을 통해 기존의 Volatility에서 탐지가 되지 않던, 공격자의 관점에서 커널 변조를 진행한 커널 객체를 탐지할 수 있음을 확인하였다. 〔Table 3〕과 같은 메커니즘을 적용하면 모든 변조된 커널 객체에 대해 탐지할 수 있는 범용 탐지 소프트웨어를 만들 수 있다.

VI. Discussion and Limitation

공격자가 의도적으로 변조하여 망가뜨린 메모리 덤프에서 의미 있는 데이터를 찾는 방법을 연구하는 것은, 분석가의 더 정확한 판단을 통해 추가적인 피해를 막을 수 있다는 관점에서 의미가 있다 생각된다. 또한 기존 연구들의 단점이었던 커널 객체를 변조/삽입 가능하지만 후속조치를 통해 분석이 가능하다는 것과, 변조의 흔적을 은폐시킬 수 없었다는 단점이 있었다. 하지만 제시한 공격 방법론을 사용하여 커널 영역에 대한 정밀 변조를 통해 분석가가 변조 되었다는 사실을 알아차리기 매우 어렵게 만들었다. 또한 메모리 수집 과정에 변조하기에 변조를 위한 도구의 흔적을 은폐시킬 수 있게 됨으로써 기존의 단점을 극복하였다는 점에 있어 의미 있는 연구라 생각된다.

커널 변조 공격 방법은 커널 객체를 변조하는 데 여러 커널 변수들이 존재하기 때문에 이를 조합한다면 무수히 많은 방법이 존재한다. 본 연구는 그러한 방법 중 공격자의 관점에서 커널을 변조하되 분석 회피가 가능하고, 메모리 수집 과정에서 커널 영역을 변조함으로써 공격에 가장 효율적이라 판단되는 공격 벡터와 공격 방법을 선제적으로 연구하였다는 것에 의의가 있다 생각된다. 주요 기여 포인트를 정리하면 다음과 같다.

- 기존 연구들에서 후속 조치를 통해 분석이 가능한 것과, 변조를 위해 사용한 흔적을 은폐시킬 수 없는 단점을 개선
- 침해사고 분석가가 놓칠 수 있는 커널 영역에 대한 변조와 은폐된 객체에 대해 정밀한 분석이 가능하도록 인사이트 도출
- 윈도우 커널에서 사용하는 주요 구조체에 대한 요약 정리를 통한 인사이트 도출
- 선제적인 공격 방법을 연구하고 이에 대응 가능한 탐지 기법을 적용한 소프트웨어를 제시함으로써 한 단계 업그레이드된 보안 생태계 구축

또한 탐지 소프트웨어 구현에 있어 사용되는 탐지 규칙을 휴리스틱을 통해 수동으로 생성하기에 시간 효율이 떨어진다는 단점이 있고 이에 따라 수동이 아닌 자동으로 생성할 수 있는 탐지 규칙 생성 자동화와 관련된 연구가 필요로 한다. 이는 머신러닝을 활용하여 다수의 메모리 덤프 파일에 대해 변조 저항성이 큰 변수를 자동으로 탐지하는 모델을 생성함으로써 보완시킬 수 있을 것이라 판단된다.

VII. 결 론

실험에서 공격자의 관점에서 주요 커널 구조체를 정리하고, 메모리 수집 과정에 개입하여 인라인 후킹과 DLL Injection을 통해 링크드 리스트 변조와 풀 태그 변조를 진행하였다. 실험 결과 메모리 분석을 위해 널리 사용되는 Volatility에서 우회되어 탐지할 수 없음을 보였고, 변조 저항성에 기반한 탐지 소프트웨어를 만들어 변조되어 은닉된 커널 객체를 탐지할 수 있음을 보였다. 특히 메모리 분석 우회를 통해 분석가의 판단에 영향을 미칠 수 있음을 보였고, 이는 메모리에 대한 정밀 분석의 필요성에 대한 인식을 제고시킨 것이다. 또한 변조 영역 탐지를 통해 분석가의 올바른 판단을 가능하게 하는 구체적인 방법론을 제시하였다는 점에서 연구의 의의가 있다 판단된다. 연구에 사용된 Windows 7 64bit와 Volatility 2.6는 향후 Windows 10과 Volatility 3을 대상으로 한 추가 연구가 필요로 할 것이다.

References

〔1〕 Kyung-ho Lee, "A Countermeasure Mechanism for Anti-memory Forensics based on Process Control Region Analysis in Windows Environment," thesis paper, Chonnam National University, Aug. 2015.

〔2〕 Dae-woon Kim, "Anti-dump technique using PHYSICAL_MEMORY_RUN memory structure manipulation," thesis paper, Chonnam National University, Feb. 2017.

〔3〕 Jae-Myung Kim, Dong-Hwi Lee, and Kui-nam Kim, "The study of response model & mechanism against windows

kernel compromises," Journal of information and security, 6(3), pp. 1-12, Sep. 2006.
[4] Hyun-Joong Woo, "A study of technique to detect and intercept Windows rootkits," thesis paper, Kyunggi University, Feb. 2005.
[5] Seok-joo Kim, "A Study of Effective Rootkit-Detection base on Windows System," thesis paper, Konkuk University, Aug. 2008.
[6] Ji-sung Han and Sang-jin Lee, "The Windows Physical Memory Dump Explorer for Live Forensics," Journal of the Korea Institute of Information Security & Cryptology, 21(2), pp. 71-82, Apr. 2011.
[7] Min-seok Bang, "Design and Implementation of Detection Tool for Hidden File by Memory Falsification," thesis paper, Dongguk University, Feb. 2013.
[8] Seok-young Jang, "A method for checking execution of rootkit using Windows kernel information," thesis paper, Ajou University, Aug. 2013.
[9] Young-Bok Kang, Hyun-uk Hwang, Ki-bom Kim, Ki-wook Sohn, and Bong-nam Noh, "Physical Memory Analysis Technology for Malware Detection," Joural of the Korea Institute of Information Security & Cryptology, 24(1), pp. 39-44. Feb. 2014.
[10] Dong-eun Shin, "A Study of Detection Method for Heap Memory Resident Malware using Table of Essential Callee Code," thesis paper, Soongsil University, Feb. 2016.
[11] Tsaur, Woei-Jiunn, and Yuh-Chen Chen, "Exploring Rootkit detectors' vulnerabilities using a new windows hidden driver based Rootkit," 2010 IEEE Second International Conference on Social Computing, pp. 842-848, Aug. 2010.
[12] Cui, Weidong, et al, "Tracking rootkit footprints with a practical memory analysis system," 21st {USENIX} Security Symposium ({USENIX} Security 12). pp. 601-615, Aug. 2012.
[13] Quynh, Nguyen Anh, and Yoshiyasu Takefuji, "Towards a tamper-resistant kernel rootkit detector," Proceedings of the 2007 ACM symposium on Applied computing. pp. 276-283, Mar. 2007.
[14] Dija, S., et al, "Forensic reconstruction of executables from Windows 7 physical memory," 2016 IEEE International Conference on Computational Intelligence and Computing Research, pp. 1-5, Dec. 2016.
[15] Stüttgen, Johannes, and Michael Cohen, "Anti-forensic resilient memory acquisition," Digital investigation vol. 10, pp. 105-115, Aug. 2013.
[16] Takahiro Haruyama and Hiroshi Suzuki, "One-byte Modification for breaking Memory Forensic Analysis," Black Hat Europe, Mar. 2012.
[17] Luka Milkovic, "Defeating Windows memory forensics," Future Soldier Exhibition & Conference, Sep. 2012.
[18] Jake Williams and Alissa Torres, "ADD - Complicating Memory Forensics Trough Memory Disarray," ShmooCon, Dec. 2014.
[19] Github, "user:F-INSIGHT", https://github.com/F-INSIGHT/Slides/tree/master/(111217) #FITALK - Windows System Structure.pdf, Dec. 2011.
[20] Google, "IA-32 Intel 32/64-bit Architecture", https://cs.hac.ac.il/staff/martin/Micro_Modern/slide03.pdf, 2012.
[21] Google, "Intel 64 and IA-32", https://www.intel.co.kr/content/www/kr/ko/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.html, Sep. 2016.

〈저자소개〉

이 한 얼 (Haneol Lee) 학생회원
2020년 3월~현재: 고려대학교 정보보호대학원 정보보호학과 석사과정
〈관심분야〉 컴퓨터 비전, 자연어 처리, 신호 처리

김 휘 강 (Huy Kang Kim) 종신회원
1998년 2월: KAIST 산업경영학과 학사
2000년 2월: KAIST 산업공학과 석사
2009년 2월: KAIST 산업 및 시스템공학과 박사
2004년 5월~2010년 2월: 엔씨소프트 정보보안실장. Technical Director
2010년 3월~2014년 12월: 고려대학교 정보보호대학원 조교수
2015년 1월~2020년 2월: 고려대학교 정보보호대학원 부교수
2020년 3월~현재: 고려대학교 정보보호대학원 교수
〈관심분야〉 온라인게임 보안, 자동차 보안, 침입탐지시스템, 네트워크 보안