ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

윈도우즈 운영체제 기반 커널 함수 보호 기법

A Protection Technique for Kernel Functions under the Windows Operating System

백 두 성[1] 편 기 현[1*]
Dusung Back Kihyun Pyun

요 약

오늘날 마이크로소프트사의 윈도우즈 운영체제는 가장 널리 사용되고 있고 인터넷 뱅킹, 게임 등 수많은 응용들에 널리 활용되고 있다. 윈도우즈 운영체제가 제공하는 커널 함수들은 실행되고 있는 임의의 프로세스들의 메모리 접근, 키보드 입출력 검사, 그래픽 출력 검사 등을 수행할 수 있기 때문에 많은 해킹 프로그램들이 이 기능들을 악용하여 메모리 해킹, 키보드 해킹, 불법적 게임 자동 사냥 도구 제작 등의 목적으로 악용하고 있는 실정이다. 기존 보안 방식은 커널 데이터 구조나 커널 함수의 시작 부분의 변형을 검사하는 방식으로 해킹 프로그램의 존재 여부를 판별한다. 본 논문에서는 기존 보안 방법의 문제점을 지적하고, 이를 해결할 수 있는 새로운 방식을 제안한다. 이 방식은 시스템 서비스 디스패쳐 코드를 변형하는 방식으로 기본 보안 방식의 문제점을 보완할 수 있다. 이 서비스 디스패쳐 코드를 해킹 프로그램이 활용하게 되면 기존 보안 프로그램이 해킹 행위를 검출하지 못한다. 따라서 커널 데이터 구조나 커널 함수의 시작 부분뿐만 아니라 디스패쳐 코드의 변형 또한 보안 프로그램에서 검출해야 해야 한다.

☞ 주제어 : 커널 함수 후킹, 시스템 보안

ABSTRACT

Recently the Microsoft Windows OS(operating system) is widely used for the internet banking, games etc. The kernel functions provided by the Windows OS can perform memory accesses, keyboard input/output inspection, and graphics output of any processes. Thus, many hacking programs utilizes those for memory hacking, keyboard hacking, and making illegal automation tools for game programs. Existing protection mechanisms make decisions for existence of hacking programs by inspecting some kernel data structures and the initial parts of kernel functions. In this paper, we point out drawbacks of existing methods and propose a new solution. Our method can remedy those by modifying the system service dispatcher code. If the dispatcher code is utilized by a hacking program, existing protection methods cannot detect illegal operations. Thus, we suggest that protection methods should investigate the modification of the dispatcher code as well as kernel data structures and the initial parts of kernel functions.

☞ keyword : kernel function hooking, system protection

1. 서 론

오늘날 마이크로소프트사의 윈도우즈 운영체제는 가장 널리 사용되고 있고 인터넷 뱅킹, 게임 등 수많은 응용들에 널리 활용되고 있다. 그러나 윈도우즈 운영체제는 개인용 컴퓨터로 출발한 운영체제이어서 보안 측면에서 태생적으로 다소 취약하다. 윈도우즈 운영체제에서 동작하는 악성 코드, 바이러스 등은 이미 널리 퍼져 있고 이들로부터 보호할 수 있는 소프트웨어 도구들이 널리 사용되고 있다.

윈도우즈 운영체제가 응용 프로그램들을 위해 제공하는 시스템 서비스들은 드라이버 등의 형태로 커널에 삽입될 수 있다. 커널에 삽입된 코드는 커널 모드(kernel mode)에서 동작하기 때문에 시스템 서비스 함수들을 비롯한 커널 데이터 구조에 임의로 접근할 수 있다. 이 커널 함수들은 실행되고 있는 임의의 프로세스들의 메모리 접근, 키보드 입출력 검사, 그래픽 출력 검사 등을 수행할 수 있기 때문에 많은 해킹 프로그램들이 이 기능들을 악용하여 메모리 해킹, 키보드 해킹, 불법적 게임 자동 사냥 도구 제작 등의 목적으로 악용하고 있는 실정이다 [1]. 따라서 보안 프로그램들도 허가 받지 않은 프로그램들이 시스템 서비스 함수들을 해킹에 이용하는 지를 검사한다. 반면 해킹 프로그램들은 보안 프로그램들이 수행하는 검사방법을 우회함으로써 보안 프로그램을 회피하는 시도를 한다.

[1] Dept. of Computer Science and Engineering, Chonbuk National Univ., 561-756, Korea.
[*] Corresponding author (khpyun@chonbuk.ac.kr)
[Received 19 April 2014, Reviewed 02 May 2014(R2 22 July 2014), Accepted 07 August 2014]

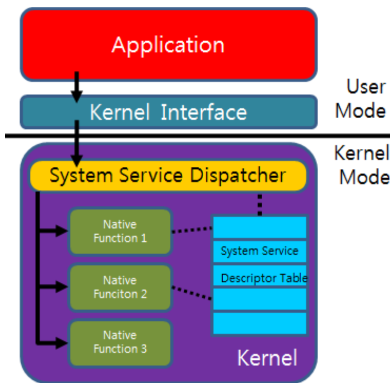
http://dx.doi.org/10.7472/jksii.2014.15.5.133

보안 프로그램은 해킹 프로그램이 불법적으로 커널 데이터 구조에 접근하거나 시스템 서비스 함수들을 호출하는 지를 검출할 수 있어야 한다.

본 논문에서는 불법적인 커널 함수 사용을 검출하는 기존 방법의 문제점을 지적하고, 이를 해결할 수 있는 새로운 방식을 제안한다.

2. 배경 지식 및 기존 방식

2.1 배경 지식



(그림 1) 시스템 서비스 처리 구조

(Figure 1) System service processing structure

그림 1은 윈도우즈 운영체제에서 시스템 서비스 처리를 위한 구조를 나타낸다[1][2][3]. 시스템 서비스는 보안이나 관리가 필요한 기능, 예들 들면 응용 프로그램들이 하드웨어를 접근하게 하거나 커널 관리 정보를 알려주는 기능을 제공하기 위한 것이다. 시스템 서비스는 사용자 모드가 아닌 커널 모드에서 동작해야 하는데, 이 역할은 커널 인터페이스 (kernel interface)가 담당한다. 응용 프로그램에서 요청하는 시스템 서비스는 ‘시스템 서비스 번호’로 구분되는데 커널 인터페이스는 시스템 서비스 번호를 eax 레지스터에 저장한 후 SYSENTER 명령어를 수행한 후 시스템 서비스 디스패쳐 (system service dispatcher) 루틴을 호출한다. SYSENTER 명령어는 CPU를 사용자 모드에서 커널 모드로 전환시킨다. 시스템 서비스 디스패쳐는 eax 레지스터에 저장된 시스템 서비스 번호를 색인으로 하여 시스템 서비스 디스크립터 테이블 (System Service Descriptor Table), 줄여서 SSDT 에 접근하여 그 값을 읽어 온다. 이 값은 해당 시스템 서비스를 수행하는 커널 함수의 시작 주소 값이다. 이 때 커널 함수를 Native 함수라고 부른다. 시스템 서비스 디스패쳐는 이 SSDT에서 읽은 Native 함수 시작 주소 값을 이용하여 해당 커널 함수를 호출하고 그 실행 결과 값을 응용 프로그램에게 돌려준다.

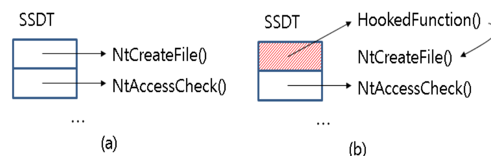
해킹 프로그램들은 이 시스템 서비스 구조에서 SSDT 데이터 구조내의 함수 시작 값을 조작하거나 Native 함수의 시작 코드의 일부분을 조작하는 방식을 취한다[1]. 전자는 SSDT의 원본 주소 값 대신 해킹 프로그램내의 자신의 코드를 대신 써 넣어 목표로 하는 응용 프로그램이 해당 커널 함수를 호출할 때 마다 원본 커널 함수가 아닌 해킹 프로그램의 함수가 호출되도록 만드는 방법이다. 후자는 Native 함수가 시작하는 코드의 시작 부분에 점프(jump) 코드를 삽입하여 해킹 프로그램내의 코드가 실행되도록 우회(detour) 시키는 후킹(hooking) 방법이다[1][4]. 편의를 위해 기존 방식의 개요를 2.1절과 2.2절에서 기술하였다.

보안 프로그램들은 해킹 프로그램들의 이러한 불법적 행위를 진단하기 위해 SSDT 테이블 내용이 원본 함수 주소 값이 아닌 다른 값이 저장되어 있는지를 검사하고 Native 함수 시작 코드에 점프 코드가 삽입되어 있지 않은지를 검사함으로써 해킹 시도가 이루어지는지를 판단하고 있다.

그러나 기존 보안 프로그램들은 시스템 서비스 디스패쳐에 해킹 프로그램들이 우회 방식을 사용할 경우 이를 탐지할 수 없다. 시스템 서비스 디스패쳐는 하나의 함수가 아니라 어셈블리어로 작성된 코드 조각 (code fragment)이다. 본 논문은 해킹 프로그램들이 이 시스템 서비스 디스패쳐 코드를 조작하여 기존 보안 프로그램들이 탐지하지 못할 수 있음을 뒤 절에서 보이고, 이를 해결할 수 있는 방법을 제시한다.

2.2 SSDT 후킹을 이용한 방식

시스템 서비스 디스패쳐가 Native 함수 시작 주소 값을 읽는 SSDT를 후킹 하는 방식의 한 예를 그림 2에 나타내었다.



(그림 2) SSDT 후킹을 이용한 방식 예

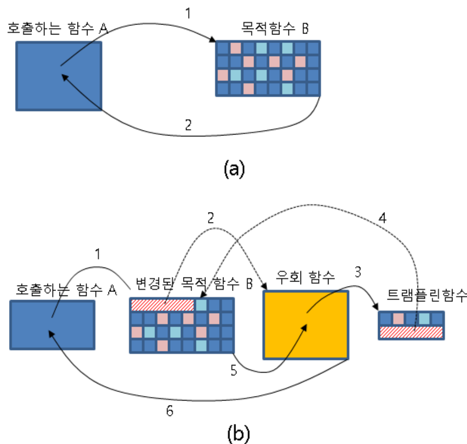
(Figure 2) An example of the SSDT hooking method

그림 2의 (a)는 SSDT의 원래 상태를 보여준다. 이 예에서 첫 번째 항목은 Native 함수 NtCreateFile() 함수의 시작 주소를 가리키고 있고 보안 프로그램이 이 함수를 보호하고자 한다고 가정하자. 이 경우 (b)의 빗금친 부분에 NtCreateFile()의 시작 주소가 아닌 보안 프로그램내의 함수 HookedFunction()의 시작 주소 값을 넣는다. 이 HookedFunction()은 NtCreateFile() 서비스를 요구한 프로세스가 정당한 경우에만 NtCreateFile() 함수를 수행한다.

해킹 프로그램의 경우도 보안 프로그램과 동일한 방식, 즉, SSDT 항목을 변경하는 방식을 이용할 수 있다. 따라서 보안 프로그램들은 SSDT내의 Native 함수 시작 주소 값들이 윈도우즈 커널 함수를 제대로 가리키고 있는지를 검사함으로써 해킹 여부를 확인한다.

2.2 Native 함수 시작 부분 후킹

윈도우즈 커널 함수는 바이너리 형태로 되어 있지만 그림 3에서 보는 바와 같이 목적함수가 시작하는 코드의 시작 부분에 점프(jump) 코드를 삽입하여 보안 프로그램내의 코드가 실행되도록 우회시킬 수 있다.



(그림 3) Native 함수 시작 부분 후킹 방법
(Figure 3) The hooking method of the starting code of the Native function

여기서 (a)는 목적함수가 변형되지 않은 상태를 보여주며 함수 A에서 Native 함수인 목적함수 B를 호출하면 함수 B가 실행 완료된 후 함수 A로 되돌아가는 것을 보여준다. (b)는 목적함수를 보호하기 위해 보안 프로그램이 후킹한 결과를 보여준다. 목적 함수 B의 시작되는 부분인 빗금친 부분을 점프 명령어로 대치하여 보안 프로그램내의 우회 함수로 이동하도록 하였다. 우회 함수에서는 이 커널 함수를 실행한 프로세스가 정당한 경우에만 트램플린(trampoline) 함수를 호출한다.

이 트램플린 함수는 변경된 목적 함수 B를 원본 목적 함수 B를 실행한 것과 동일하게 만들기 위한 것이다. 트램플린 함수를 만드는 방법은 변경된 목적 함수 B의 빗금친 부분에 원래 있었던 원본 코드 조각과 그 코드 바로 뒤로 점프하는 코드를 합치면 된다. 그 결과 트램플린 함수를 실행하면 원본 목적 함수를 실행한 것과 동일해 진다.

그림 3의 (b)에서 함수 A가 변경된 목적 함수 B를 호출한 뒤 다시 함수 A로 되돌아오는 원리를 살펴보자. 함수 A가 호출(call) 명령어를 실행하면 함수 A의 그 다음 명령어 주소가 실행 스택(execution stack)에 쌓인다(push). 나중에 리턴(return) 명령어를 실행하면 이 실행 스택의 최상위 값을 추출(pop)하여 프로그램 카운터 레지스터(program counter register) 값에 넣게 된다. 함수 A에서 호출 명령어를 실행하면 그 결과 프로그램 카운터 레지스터 값은 변경된 목적 함수 B의 시작 주소를 가리킨다. 함수 B의 이 부분은 점프 명령어로 변경되어 있어 이 점프 명령어를 실행하게 되고, 그 결과 프로그램 카운터 레지스터 값이 우회 함수의 시작 부분을 가리킨다. 우회 함수는 함수 B를 호출한 프로세스가 정당한 지를 확인한 후 정당하면 트램플린 함수 호출 명령어를 실행한다. 그 결과 실행 스택은 그림 4와 같게 된다. 트램플린 함수는 함수 B의 시작 부분을 실행하게 되고 최후에 결국 점프 명령어를 수행하여 함수 B의 나머지 부분을 실행한다. 함수 B는 결국 리턴 명령어를 실행하게 된다. 이 리턴 명령어를 실행한 결과로 갖게 되는 프로그램 카운터 레지스터 값은 실행 스택의 최상위 값이고, 그림 4에서 보듯이 우회 함수의 리턴 주소가 있으므로 그 주소로 돌아가게 된다. 또한 우회함수에서 마지막으로 리턴 명령어를 수행하면 그 시점에 실행스택의 최상위에는 함수 A의 리턴 주소가 있으므로 함수 A로 돌아가게 된다.

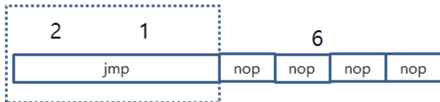
우회 함수의 리턴 주소
함수 A의 리턴 주소
...

(그림 4) 실행 스택의 내용
(Figure 4) The content of the execution stack

목적 함수 B에 점프 명령어를 삽입할 때 그림 5의 예에서 보는 바와 명령어 크기가 일치하지 않을 수 있다.

(그림 5) 점프 명령어 대상 코드 예
(Figure 5) An example of the target code for substituting instructions with a jump instruction

그림 5에서 점프 명령어 삽입에 5 바이트가 필요하다고 가정할 때 점프 명령어를 삽입하게 되면 “multiply” 명령어가 조각나는 문제가 발생한다. 이 경우 5 바이트가 아닌 ”multiply” 명령어 전체를 포함한 9 바이트를 트램플린 코드 작성에 사용한다. 또한 그림 6에서 보는 바와 같이 점프 명령어를 삽입한 뒤 남는 4 바이트 공간에는 아무 일도 하지 않고 CPU 싸이클만 소모하는 한 바이트 크기의 ”nop” 명령어를 삽입한다.



(그림 6) 점프 명령어 삽입 예
(Figure 6) An example of the insertion of a jump instruction

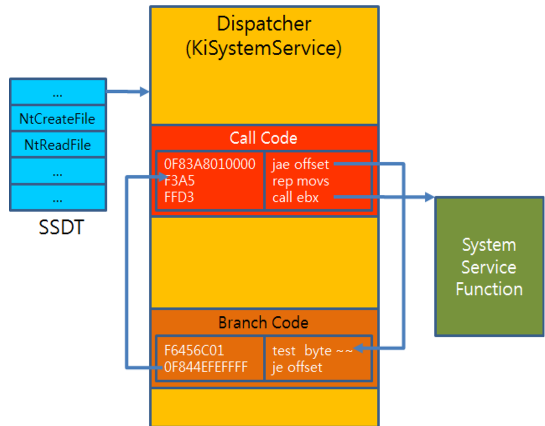
3. 제안하는 커널 함수 보안 방식

3.1 제안하는 커널 함수 후킹 방식

응용 프로그램이 시스템 서비스 요청을 한 후 커널 모드에서 동작하는 첫 번째 코드는 시스템 서비스 디스패쳐이다. 이 코드는 커널을 위한 여러 가지 작업을 수행한 이후에 결국 KiServiceSystem()이라 불리는 코드로 이동하게 되고, 이 코드에서 SSDT 내에 있는 해당 커널 함수 주소 값을 찾아 해당 함수를 호출하는 일을 담당한다.

본 논문은 시스템 서비스 디스패쳐 코드 내에서 해당 커널 함수 주소를 찾아 호출하는 실행 코드를 변경하여 커널 함수를 해킹 프로그램이 불법적으로 사용하는 지를 검사하는 방법을 제안하다.

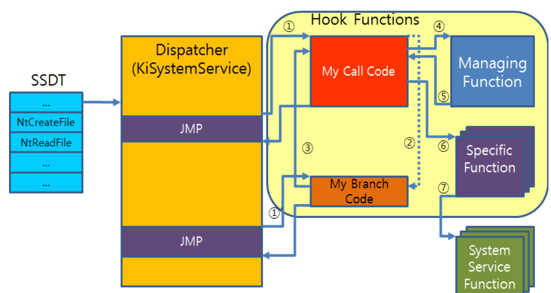
이와 같은 시도의 어려운 점은 시스템 서비스 디스패쳐 코드를 역공학(reverse engineering)을 통하여 분석하고 기존 코드 흐름에 전혀 영향을 주지 않으면서 모니터링 코드를 삽입해야 한다는 점에 있다. 본 논문은 이 기계어 코드를 분석하여 이 문제를 해결하였다. 이 디스패쳐 코드 부분은 윈도우즈 버전과 종류에 상관없이 모두 동일하기 때문에 윈도우즈 운영체제에 공통적으로 적용할 수 있다.



(그림 7) 원본 서비스 시스템 디스패쳐 코드
(Figure 7) Original system service dispatcher code

그림 7은 서비스 시스템 디스패쳐 코드를 나타낸다. 이 코드는 서비스 시스템 번호를 색인으로 하여 SSDT로부터 해당 커널 함수 시작 주소 값, 예들 들어 NtCreateFile 함수 시작 주소 값을 읽어 ebx 레지스터에 저장한 후 ‘call ebx’ 명령어를 수행한다. 이 명령어를 실행하게 되면 해당 커널 함수가 실행된다. 그런데 call ebx 명령어를 수행하기 이전에 ’Branch Code’로 이동할 수 있다. 그림 7에서 보는 바와 같이 ’Branch Code’에서는 ’test byte ~~’ 명령어를 수행한 이후에 결과에 따라 ’call ebx’ 이전 명령어인 ’rep moves’로 이동하거나 ’test byte’ 다음 명령어를 수행할 수 있다. 이와 같이 기계 명령어가 점프 명령어들을 포함하여 복잡하게 얽혀 있기 때문에 커널 동작 기능과 동일하면서 모니터링 기능을 시스템 디스패쳐 코드에 삽입하는 것은 간단하지 않다.

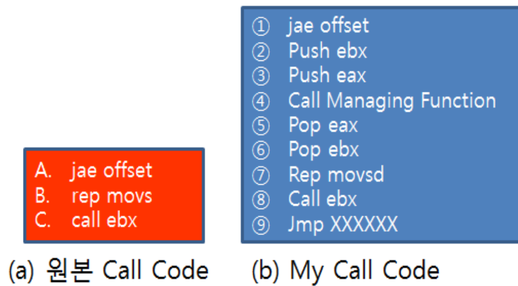
본 연구는 커널의 동작 기능에 영향을 주지 않고 동일한 기능을 수행하면서도 보안 기능을 수행할 수 있도록 그림 8에 나타낸 바와 같이 명령어 코드를 삽입하였다.



(그림 8) 제안된 시스템 서비스 디스패쳐 코드
(Figure 8) Proposed system dispatcher code

그림 8은 그림 7의 'Call Code'에 해당되는 부분에 7 바이트 크기의 'jmp x' 명령어로 대치되어 이 명령어를 실행하게 되면 x에 해당되는 주소값이 프로그램 카운터 레지스터로 들어가 그 주소부터 CPU의 다음 명령어 가져오기(fetch)가 일어나게 만든다. x 값은 보안 루틴의 시작 주소 값, 그림 8에서 'My Call Code'의 시작 주소 값을 넣는다.

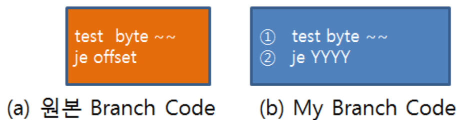
'My Call Code' 코드의 내용은 그림 9과 같다.



(그림 9) 'My Call Code' 코드 내용
(Figure 9) Code content of the 'My Call Code'

우선 eax와 ebx 레지스터 값을 실행 스택에 저장한다. 이 두 레지스터 값을 저장하는 이유는 보안 루틴을 수행하는 'Managing Function' 함수가 실행될 때 범용 레지스터인 이 두 레지스터 값을 변경할 수 있기 때문이다. 이 두 레지스터에는 각각 시스템 서비스 번호와 해당 커널 함수의 시작 주소가 들어있음을 상기하자. 'Managing Function'에서는 해당 커널 함수를 사용하는 프로세스가 정당한지를 검사하거나 모니터링 기능을 수행할 수 있다. 이 보안 함수를 수행한 뒤 다시 eax와 ebx 레지스터 값을 복원한 후 원본 'Call Code'의 뒤에 위치한 명령어 코드를 실행하도록 'jmp XXXXX' 명령어를 수행한다. 'My Call Code'는 결국 보안 루틴인 'Managing Function'을 수행한 후 해당 커널 함수가 수행되는 점을 제외하면 원본 실행 흐름과 동일하다.

'My Branch Code'의 경우 그림 10에 나타낸 바와 같다.



(그림 10) 'My Branch Code' 코드 내용
(Figure 10) Code content of the 'My Branch Code'

'je offset' 명령어 대신 'je YYYY'를 사용한다. 그 이유는 'offset'이 그림 7에서 'rep movs' 명령어가 저장된 위치를 나타내고 이 명령어는 그림 8에서 'My Call Code'로 옮겨졌기 때문이다. 이 때 YYYY는 그림 8의 'My Call Code'의 'Rep movs' 명령어가 저장된 주소 값을 의미한다.

그림 7와 그림 8을 비교하면 응용 프로그램에서 커널 함수를 호출할 때 'Managing function'을 추가적으로 실행되도록 만든 차이점을 제외하면 실행 흐름은 결국 동일하다.

3.2 제안하는 보안 방식 및 기존 보안 방식의 제한점

많은 상용 보안 프로그램들은 그들이 보호해야 될 인터넷뱅킹 프로그램이나 게임 등이 실행될 때 SSDT의 변형이나 커널 함수의 시작 부분이 변형되었는지를 검사한 후 사실로 판명되면 위험성을 알린 뒤 보호 대상 프로그램의 실행을 강제 종료 시키는 방식을 택한다.

만일 본 논문에서 제안한 서비스 디스패처 코드 수정을 통한 후킹 기법을 해킹 프로그램이 보안이 아닌 해킹의 용도로 악용할 가능성을 배제할 수 없다. 3.1절에서 제시한 본 논문의 커널 함수 후킹 방식은 보안과 모니터링 용도로 제안한 것이지만 'Managing Function'에 해당되는 곳에 보안이 아닌 해킹의 역할을 수행하는 코드를 작성하는 경우 기존의 보안 방식으로는 이 사실을 인지하지 못한다. 기존 보안 방식은 SSDT의 변형을 감지하거나 커널 함수의 시작 부분이 변형되었는지를 검사하지만 디스패쳐 코드 변형을 고려하지 않기 때문이다.

본 논문은 기존 보안 방식에다가 시스템 서비스 디스패쳐 코드 변형 살피기를 추가하는 것을 제안한다. 앞서 말한 상용 보안 프로그램들과 동일한 절차로 SSDT 변형과 커널 함수의 시작 부분 변형을 조사하고, 추가로 시스템 서비스 디스패쳐 코드도 변형되었는지를 조사하는 것이다. 이 경우 디스패쳐 코드 후킹을 해킹 프로그램이 악용하는 경우도 탐지해 낼 수 있다.

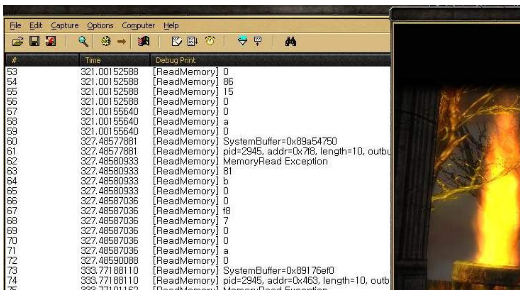
다행스럽게도 시스템 서비스 디스패쳐 코드는 윈도우즈 버전과 종류에 상관없이 모두 동일하기 때문에 윈도우즈 운영체제에 공통적으로 적용할 수 있다.

4. 실 험

대다수의 온라인 게임 프로그램들은 자동 사냥, 아이디 해킹 등을 방지하기 위해 상용 보안 모듈을 탑재하고

있다. 이 모듈들은 임의의 프로세스가 커널 함수를 이용하여 자신이 보안을 담당한 프로세스의 메모리 읽기 등 해킹 행위를 시도하면 보호해야할 프로세스를 종료시켜 보호한다. 예들 들면, 커널 디버거를 이용하면 특정 프로세스의 메모리 영역을 읽을 수 있지만, 온라인 게임 프로그램들의 보안 모듈은 이러한 행위를 허용하지 않는다.

본 논문에서 제시한 시스템 서비스 디스패쳐 변경 방식을 적용한 뒤 상용 보안 모듈이 탑재한 게임 프로세스를 실행시킨 결과 해당 게임 프로세스의 메모리 읽기 시도가 가능하였다. 실험은 윈도우 XP와 윈도우 7에서 수행해 보았으나 디스패쳐 코드가 동일하기 때문에 다른 윈도우 버전도 동작한다. 그림 11은 한 상용 온라인 게임 프로세스의 메모리를 커널 디버거를 이용하여 읽기가 성공한 예를 보여준다.



(그림 11) 커널 함수를 이용한 메모리 읽기 성공
(Figure 11) Success of memory read using the kernel function

이것은 기존 보안 방식이 SSDT나 커널 함수의 시작 부분의 변형을 탐지하지만 시스템 서비스 디스패쳐 변형을 파악하지 못해 본 실험에서 행한 커널 디버거의 메모리 읽기를 막지 못했지 때문이다. 만일 본 연구가 제안한 시스템 서비스 디스패쳐 코드 변형을 수행했다면 이러한 불법적인 메모리 접근을 탐지할 수 있었을 것이다.

5. 결 론

본 논문은 기존의 커널 함수 보안 방식이 SSDT나 커널 함수 시작 부분에 초점을 맞춘 것과 달리 시스템 서비스 디스패쳐 코드 부분을 활용하여 보안을 제공할 수 있는 방안을 제시하였다. 본 논문에서 제시한 방식을 통하여 상용 보안 프로그램들과도 충돌 없이 커널함수를 보안할 수 있다.

향후 연구로는 리눅스 등의 유닉스 기반 운영체제에 본 연구 방식을 적용해 보는 것이다. 유닉스 기반 운영체제에서 시스템 서비스 함수를 제공하는 방식이 윈도우즈 운영체제와 유사점이 많기 때문에 본 논문의 방식을 기반으로 확장할 수 있을 것으로 기대된다.

참 고 문 헌 (Reference)

[1] Greg Hoglund, Jamie Butler, “Rootkits: Subverting the Windows Kernel”, Chapter 4 - Chapter 5, Addison-Wesley, 2005

[2] ME Russinovich, DA Solomon, A Ionescu, “Windows Internals", Microsoft Press., 2012

[3] Jeffrey Richter, “Programming Applications for Microsoft Windows”, Microsoft Press., 1999

[4] Galen Hunt, Doug Brubacher, “Detours: Binary Interception of Win32 Functions”, Proceedings of the 3rd USENIX Windows NT Symposium, Vol. 3, pp. 14-23, 1999

◐ 저 자 소 개 ◑

백 두 성 (Dusung Back)
2006년 전북대학교 전자정보공학부 컴퓨터공학 졸업(학사)
2008년 전북대학교 전자정보공학부 컴퓨터공학 졸업(석사)
2009년 ~ 현재 전북대학교 컴퓨터공학부 박사과정
관심분야 : 센서네트워크, P2P 네트워크, 운영체제 보안
E-mail : whstar83@gmail.com

편 기 현 (Kihyun Pyun)
1995년 인하대학교 전자계산공학과 졸업(학사)
1997년 KAIST 전산학과 졸업(석사)
2002년 KAIST 전산학과 졸업(박사)
2003년 KAIST 전기및전자공학 박사후 연구원
2004년 ~현재 전북대학교 컴퓨터공학부 부교수
관심분야 : 차세대 인터넷 서비스, 유무선 네트워크, 시스템 소프트웨어, 모바일 소프트웨어
E-mail : khpyun@chonbuk.ac.kr