프로파일 기반 악성 로더 공격탐지 및 필터링 기법

윤이중* · 김요식*

요 약

소프트웨어를 대상으로 하는 다양한 공격방법이 등장하고 있는 가운데 컴퓨터 소프트웨어에 대한 불법 조작 및 변조 등의 위협이 증가하고 있다. 특히, 온라인상에서 동작하는 어플리케이션 클라이언트를 대상으로 악의적인 로더 프로그램을 이용하여 프로그램의 코드를 조작하고, 흐름을 변조하여 정상적인 동작을 방해하는 행위가 날로 늘어나고 있다. 본 논문에서는 악의적인 용도로 사용되는 로더가 가지는 패턴을 분석하여 시그너처를 생성하고, 변형된 패턴을 탐지할 수 있고 시그너처 기법을 보완한 프로파일 기반의 탐지 기법을 제시한다.

Profile based Malicious Loader Attack Detection and Filtering Method

E-Joong Yoon* · Yosik Kim*

ABSTRACT

Recently, illegal manipulation and forgery threats on computer softwares are increasing. Specially, forge the code of program and disrupt normal operation using a malicious loader program against the Internet application client. In this paper, we first analyze and generate signatures of malicious loader detection. And, we propose a method to secure the application client based on profiling which can detect and filter out abnormal malicious loader requests.

Key words : Malicious Loader, Profiling, Software Protection

* 국가보안기술연구소

1. 서 론

역공학 기술이 진보함에 따라 컴퓨터 소프트웨어에 대한 불법 조작 및 변조 등의 위협이 증가하고 있으며, 인터넷에 공개된 단순한 도구를 이용하여 손쉽게 크래킹을 할 수 있게 되었다. 역공학을 이용한 해킹은 기존의 네트워크, 호스트 시스템을 대상으로 하는 것이 아니라 온라인상에서 동작하는 클라이언트 프로그램에 대한 데이터, 코드를 변조하여 해킹하는 것이다.

일반적으로 클라이언트 프로그램은 사용자의 시스템 상에서 동작하여 다른 시스템 보다 중요 정보의 노출 등이 쉽게 발생할 수 있다. 트로이목마, 바이러스, 스파이웨어 등의 프로그램은 불특정 사용자를 공격대상으로 정하고, 사용자 시스템에 사용자의 의지와는 상관없이 특정 목적을 가지고 동작하는 반면, 핵(Hack)이라 불리는 로더를 이용하여 클라이언트 프로그램을 대상으로 공격하는 방법은 공격 주체가 해당 시스템의 사용자라는 면에서 다르게 해석될 수 있다.

로더란 비교적 작은 코드 사이즈를 가지며 다른 프로그램을 실행시키고 실행 시킨 프로그램을 메모리상에서 데이터나 프로그램 흐름을 바꿀 수 있는 프로그램을 의미하며 이미 오래전부터 게임 데이터를 조작하는 트레이너라는 이름으로 많이 알려져 왔으며 최근에는 온라인 게임 클라이언트 뿐만 아니라 웹하드 클라이언트 등을 공격하는 도구로 많이 사용되고 있는 추세이다[1-5].

본 논문에서는 로더의 바이너리 데이터를 분석하고, 로더가 사용하는 Win32 API를 후킹하여 API와 파라미터의 사용형태를 분석하는 프로파일링 기법을 사용한다. 시험을 통해 분석한 결과 본 논문에서 제시한 프로파일 식별자에 의한 탐지가 매우 효과적이었으며, 사용자 시스템에서 실행되는 프로세스, 대상 클라이언트에 접근하는 로더가 사용하는 API를 분석하여 공격을 탐지 할 수 있었다.

본 논문의 구성은 2장에서 소프트웨어 보안 방법에 대한 관련 연구를 분석하고, 3장에서는 제안된 시스템 구조를 설명하고, 4장에서 제안된 시스템에 대한 시험 및 평가에 대해 설명하고, 5장에서 결론을 맺는다.

2. 관련 연구

악성 로더는 바이러스, 트로이목마, 스파이웨어 등과 같이 Win32 API를 사용하지만, 그 공격 주체가 시스템을 사용하는 사용자라는 면에서 일부에서도 많이 다루어지지 않고 있다. 안티바이러스와 시스템의 프로세스에 대해 감시하고 통제하는 Process Guard와 같은 프로그램의 목적은 시스템 상에서 동작하는 스파이웨어, 바이러스, 백도어 등이 사용자 시스템에 접근하여 사용자 동의 없이 실행되는 프로그램의 동작을 방지하고 제거하는데 목적이 있다[6]. 하지만, 로더는 시스템을 사용하는 사용자가 그 공격 주체인 동시에 사용자 동의하에 실행되기 때문에 의미와 목적이 다르다고 할 수 있다.

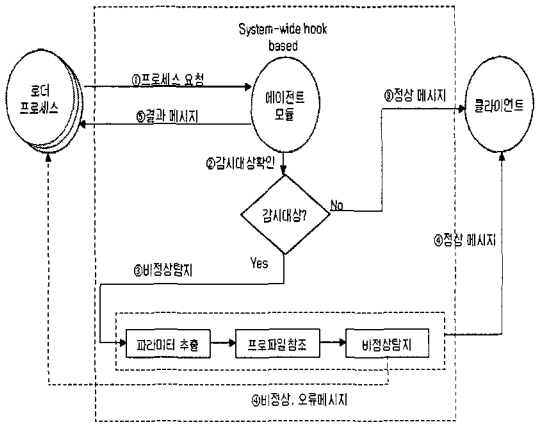
현재에도 소프트웨어의 보호를 위해 많은 연구들이 진행되고 있는데, Min Chen은 소프트웨어 보호방법의 일환으로 불법적인 소프트웨어의 사용을 막기 위해 저작권, 특허, 라이센스의 법률적인 제도와 하드웨어기반의 동글, 제한적 설치, 히든 시리얼 번호를 이용한 기술적인 보호방법에 대한 기법을 제시하였으며 각 기법에 따른 장점과 단점을 설명하였다[7]. 하지만 이런 기법들은 오래전부터 활용되어 왔음에도 불구하고 소프트웨어를 보호하기 위한 대안으로 자리 잡지 못하고 있다. 본 논문에서는 복사 방지에 주안점을 두지 않고 설치된 프로그램에 대한 오남용을 막기 위해 실시간으로 이상상태를 탐지하여 필터링하는 방법을 제한하고 프로파일 레코드의 키로 이용되는 프로파일 식별자를 이용한 탐지 방법을 적용하였다.

Horne, Matheson은 소프트웨어 변조를 방지하는 기술로 Self-Checking 기법을 제안하였다[8]. Selt-Checking 기법은 Testers와 Correctors로 구성된 컴포넌트를 소스코드와 바이너리 코드에 임베디드 시키는 형태를 가지며 일정 시간의 간격을 두고 프로그램의 무결성을 검사하여 변조가 발생한 것을 탐지하게 된다. 하지만, 임베디드된 코드는 프로그램의 원래 수행 속도에 영향을 줄 수 있으며, 소스코드와 바이너리 코드내에 임베디드된 컴포넌트의 기능은 역공학을 통해 무력화 시킬 수 있어 컴포넌트 자체를 보호하는 것은 어려운 문제이다. 본 논문에서는 별도의 에이전트로 구성되어 소스코드 변경 없이도 적용이 가능하며 API훅을 통해 API의 파라미터를 분석하여 보호대상 프로세스로의 악의적인 접근을 차단하고 접근규칙을 생성하는 방법으로 API 프로파일링 기법을 이용하여 정상적인 접근에 대한 학습과정을 통해 자동화된 규칙을 생성함으로써 접근규칙을 효율적으로 생성할 수 있도록 구성하였다. 또한, 실시간으로 로더 동작 상태를 탐지하여 필터링하는 방법을 제안하는 것으로 수행 중인 프로세스의 실행 파일의 임포트 어드레스 테이블(Import Address Table)을 참조하여 API 리스트를 구하고 프로파일 레코드의 키로 이용되는 프로파일 식별자를 이용한 탐지방법에서 좋은 탐지효과를 가지게 된다.

3. 프로파일 기반 악성 로더 보안

3.1 제안된 시스템 구조

악성 로더에 대한 프로파일링을 수행하기 위해서는 사용자 시스템에서 동작하는 프로세스들에 대한 모니터링이 가능해야 하며, 각 프로세스가 사용하는 API 정보에 대해 접근이 가능해야 한다. 이러한 요구사항을 만족하기 위한 시스템 구조로는 프로세스 모니터링 에이전트, 해당 프로세스의 실행 파일로부터 IAT에 대한 정보를 얻는 에이전트로 구성되며 시스템 와이드 훅(System-wide Hook)을 이용하여 실시간 동작 및 이상상태 탐지를 가능하게 된다.



(그림 1) 제안된 시스템 구조

(그림 1)에서와 같이 윈도우 시스템에서 동작하는 프로세스들은 시스템 와이드 훅을 이용하는 에이전트에 의해 모니터링 된다. 먼저 에이전트는 시스템 와이드 훅을 이용하여 프로세스들이 사용하는 API들에 대해 모니터링하며 API의 파라미터내용도 함께 파싱 및 분석하게 된다. 검사대상이 아닐 경우 프로세스가 사용하는 API와 대상 클라이언트로의 접근에 대해서는 허용한다.

검사대상에 해당하는 경우 API에 대한 파라미터를 추출함과 동시에 해당 프로세스의 경로를 파악하여 PE(Portable Executable) 이미지상의 임포트 어드레스 테이블을 추출한다. 파라미터와 임포트 어드레스 테이블 추출은 다음 절에서 자세히 소개된다. 요청에 대한 파싱이 완료되면 프로파일 자료구조를 참조하여 매칭되는 것이 있으면 비정상탐지를 통해 파라미터에 대한 특성검사를 수행하게 되고, 매칭이 되지 않으면 정상 행위로 판단하여 정상메시지를 전달하게 된다.

3.2 로더 프로파일링

로더 프로파일은 사전에 로더 프로세스의 구조를 파악하여 이를 데이터베이스화 하게 된다. 먼저 프로파일러는 자료 수집 단계로 시스템 동작중인 프로세스들에 대해 PE 이미지 상의 임포트 어드레스 테이블을 추출하여 의심가능성이 있는 API들에 대한 리스트를 (그림 2)와 같이 획득한다.

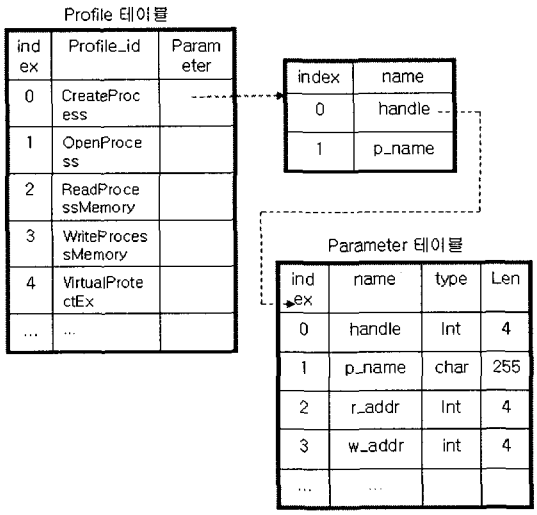
```
Dump of file xxxx.exe
File Type: EXECUTABLE IMAGE
  Section contains the following imports:

    KERNEL32.DLL
                403000 Import Address Table
                     0 Import Name Table
                     0 time date stamp
                     0 Index of first forwarder reference
                     0  VirtualProtectEx
                     0  TerminateProcess
                     0  ReadProcessMemory
                     0  OpenProcess
                     0  IsDebuggerPresent
                     0  GetProcAddress
                     0  GetModuleHandleA
                     0  GetExitCodeProcess
                     0  GetCommandLineA
                     0  ExitProcess
                     0  CreateFileA
                     0  CloseHandle
                     0  WriteProcessMemory
...
```

(그림 2) PE 이미지상의 IAT

수집된 자료인 API 리스트를 분석하여 데이터베이스에 저장된 악성 로더가 사용하는 API 형태가 비슷하다고 판단되면 감시 대상 프로세스로 등록하여 로더가 사용하는 API를 모니터링하게 된다.

로더 프로파일러는 (그림 1)에서의 일반적인 에이전트 형태와 유사하며, 프로세스들에 대한 요청 데이터를 분석하여 (그림 3)에서와 같이 프로파일 레코드를 생성하고, 생성된 레코드를 파일에 저장함으로써 정상 프로파일 데이터를 생성하게 된다.



(그림 3) 프로파일 레코드의 구성

CreateProcess, WriteProcessMemory 등의 API의 프로토타입을 참조하여 감시 파라미터를 사전에 정의하였으며, 각 API에서 모니터링이 되는 주요 파라미터로는 프로세스 핸들, 프로세스 이름, 대상 프로세스로의 어드레스, 임의 데이터를 쓰기 위한 어드레스 등이 있으며 프로파일 ID에 따라 필요 파라미터가 적용된다. 프로파일러는 로더가 사용하는 API를 모니터링하여 API명과 파라미터를 자동으로 수집하게 되며 수집된 데이터를 토대로 정책 테이블을 만들고, 이후에 출연하는 악성 로더에 대응하여 동작한다. 본 논문에서는 보호대상 프로세스로의 임의 접근을 막기 위해 악성 로더에 대한 시그니처 데이터베이스와 새로운 로더에 대한 자료수집과 융통성 있는 탐지 및 방어를 위해 프로파일링 기법을 함께 사용하였다.

3.3 로더 탐지 방법

본 논문에서 제시하는 탐지 알고리즘은 두 가지로 구성된다. 첫 번째는 시그니처 기반에 의한 탐지 방법, 두 번째는 프로파일링 식별자에 의한 탐지방법이다.

3.3.1 시그니처 기반의 탐지 방법

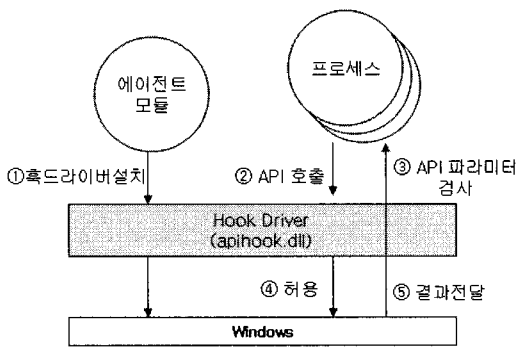
시그니처 기반의 탐지방법은 대부분의 로더가 Win32/PE 포맷을 가지는 실행파일 이미지를 가진다는 점에서 착안하였으며 적용 방법으로는 PE 이미지의 프로그램 엔트리 포인트(Program Entry Point)를 기점으로 하여 일부 바이너리 데이터를 비교하는 방법과 PE 헤더정보를 파싱하고 분석하여 기 등록된 악성 로더와 동일한 패턴인지를 판독하여 탐지하는 방법으로 나누어진다. 이 방법은 로더가 실행함과 동시에 특별한 동작이 없는 상태에서도 로더의 일부 바이너리 데이터와 시그니처 데이터베이스상의 데이터와 비교하여 실제 로더가 임의 동작을 하기 전에 탐지가 가능하다. 시그니처 데이터베이스는 (그림 4)와 같은 형태를 가지며 PE 이미지의 IMAGE_OPTIONAL_HEADER 구조체의 DWORD AddressOfEntryPoint 필드를 참조하여 위치를 구할 수 있다.

```
[freebox.exe]
signature = 60 BE 15 90 40 00 8D BE EB 7F FF FF 57
83 CD FF EB 10 90 90 90 90 90 90 8A 06 46 88 07 47
01 DB 75 07 8B 1E 83 EE FC 11 DB 72 ED B8 01 00
00 00 01 DB 75 07 8B 1E 83 EE FC 11 DB 11 C0 01
DB 73 EF 75 09 8B 1E 83 EE FC 11 DB 73 E4 31 C9
83 E8 ...
[fdx-pop.exe]
signature = 60 BE ?? 90 40 00 8D BE ?? ?? FF FF 57
83 CD FF EB 10 90 90 90 90 90 90 8A 06 46 88 07 47
01 DB 75 07 8B 1E 83 EE FC 11 DB 72 ED B8 01 00
00 00 01 DB 75 07 8B 1E 83 EE FC 11 DB 11 C0 01
DB 73 EF 75 09 8B 1E 83 EE FC 11 DB 73 E4 31 C9
83 E8 03 ...
[freehack.exe]
signature = E8 ?? ?? ?? 00 31 ED 55 89 E5 81 EC ??
00 00 00 8D BD ?? FF FF FF B9 ?? 00 00 00 ?? ?? ??
?? ?? ?? ?? ?? ?? 00 ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ...
```

(그림 4) 로더 시그니처 데이터

3.3.2 프로파일링 식별자에 의한 탐지 방법

프로파일링 식별자에 의한 탐지방법은 로더가 API를 사용하여 보호 대상 프로세스로의 접근이 이루어짐과 동시에 로더에 대한 정보수집 및 탐지가 가능한 방법이다. 이 방법은 시스템 와이드 훅(System-wide hook)이 선행 되어야 하며 감시 대상 API에 대해 API 후킹 루틴이 필요하다. 등록된 API 리스트들은 저마다 별도의 후킹된 API를 가지며 로더가 시스템내의 API를 호출하기 전에 매핑된 API를 호출하게 된다. 매핑된 API는 로더가 호출한 API와 파라미터를 분석, 즉 접근 대상 프로세스 이름, 접근 대상 프로세의 읽을 어드레스와 새로 작성할 어드레스에 대한 내용을 파싱하여 기 등록된 데이터베이스에서 검색 및 비교하여 해당 API 처리 루틴의 실행여부를 판단하게 된다. 이미 등록되어 있지 않은 레코드는 새롭게 데이터베이스에 등록되고, 파라미터 비교후 접근이 불허한 API의 경우 프로세스를 강제 종료 또는 시스템 API의 원래 기능을 무력화 시키게 된다. (그림 5)는 시스템 와이드 훅을 기반으로 한 API 필터링 동작 원리를 나타낸다.



(그림 5) API 필터링

에이전트 모듈은 시스템 와이드 훅의 기본 요구사항을 충족시키기 위해여 DLL로 구현하였으며, 훅 콜백 프로시져를 시스템상에서 훅된 프로세스 주소 공간에 설치하여 실행된다. 시스템 와이드 훅을 설치하면 운영체제는 DLL을 프로세스 각각의 주소 공간에 매핑하게 되어 실행되는 프로세스

들에서 사용되는 감시대상 API 들에 대한 호출 여부와 파라미터를 필터링하여 검사하게 된다. (그림 6)에서는 에이전트 모듈에 대한 로그를 보여주고 있는데, freehack.exe가 보호 대상 프로세스로 접근하여 메모리에 대한 읽기는 가능하지만, Write ProcessMemory를 수행하여 보호 대상 프로세스에 대해 메모리의 임의 변경을 탐지하고 막는 것을 보여준다.

```
agent:C:\WINDOWS\Explorer.EXE:_myReadProces
sMemory, hProcess:658 lpBaseAddress:20ad0,
lpBuffer:e4f910, nSize:56
agent:E:\test\loaders\freebox.exe:_myReadProcess
Memory, hProcess:48, lpBaseAddress:403020,
lpBuffer:0x0040317F, nSize:4
agent:E:\test\loaders\freebox.exe:failed to access
,_myWriteProcessMemory,hProcess:48,
lpBaseAddress:403020, lpBuffer:0x00403020, nSize:4
agent:C:\WINDOWS\Explorer.EXE:_myReadProces
sMemory, hProcess:7ec, lpBaseAddress:7ffd7008,
lpBuffer:e4f85c, nSize:4
```

(그림 6) 에이전트의 탐지 로그

4. 시험 및 평가

본 논문에서는 구현된 에이전트와 프로파일러는 WindowsXP에서 구현되었으며, 인터넷에서 수집한 악성 로더와 실험을 위해 구현된 프로그램을 대상으로 시험이 수행되었다. 프로파일러는 프로파일 레코드를 생성하기 위해 실시간으로 동작되는 시스템상의 프로세스들의 시작과 끝을 감지하고 새로운 프로세스가 시작되면 API 후킹을 통해 레코드를 생성한다. 본 논문에서는 인터넷에서 수집한 악성 로더를 실행함과 동시에 프로파일링을 수행하여, 논문에서 제시한 에이전트 기반의 필터링 시스템에서 유지해야 하는 프로파일 레코드의 수를 확인함으로써 구현된 시스템의 효용성을 확인한다.

실험에 사용된 악성 로더들은 현재 인터넷에서 사이트에 유포되어 있고, 실제 온라인 소프트웨어의 클라이언트를 공격대상으로 하고 있는 프로그램들이다. 악성 로더 샘플 대부분이 비슷한 형태를 가지며 로더가 사용하는 API의 대부분은 윈도우 시스템 DLL인 Kernel32.dll, User32.dll에 주로 구현되어 있었다. 분석해본 결과 <표 1>와 같이 중복된 API를 사용하는 것을 알 수 있었다.

〈표 1〉 악성 로더의 API 사용 비교

악성 로더	주요 Win32 API
freebox.exe	SuspendThread, **CreateProcess,** GetModuleFileName, ResumeThread, **VirtualProtectEx, WriteProcessMemory, ReadProcessMemory,** GetCommandLine
fdx-pop.exe	OpenProcess, GetModuleHandle, GetCommandLine, **CreateProcess, WriteProcessMemory**
freehack.exe	**VirtualProtectEx,** TerminateProcess, **ReadProcessMemory, OpenProcess,** GetProcAddress, **WriteProcessMemory**
asx-ds2.exe	FindResourceA, GetModuleHandleA, LoadResource, **OpenProcess,** SizeofResource, **WriteProcessMemory**
Speeder XP.exe	**OpenProcess, WriteProcessMemory**

시그니처 기반의 탐지 알고리즘의 경우 기 등록된 악성 로더에 대한 탐지에 높은 탐지율을 보였으나, 새로운 바이너리 형태나 PE 패킹 도구를 이용하여 팩된 로더를 분석 및 탐지 하지 못하는 문제점이 발생하였다[9]. 반면, 프로파일 기반의 시스템 와이드 훅을 이용한 탐지 알고리즘의 경우 로더 자체가 별도의 패킹 도구를 이용하여 팩되었어도 팩된 코드의 특성상 디스크상의 PE 이미지에 대한 분석이 어렵지만, 메모리상에서는 팩된 코드가 모두 언패킹(Unpacking)되어 실행되기 때문에 API 후킹을 이용하여 모니터링 및 자료 수집이 가능하였다[10, 11]. 접근 대상 프로세스를 등록 해놓은 후 타 프로세스로부터의 접근을 API선에서 필터링

하였을 경우 높은 안정성을 볼 수 있었다. 만일 사용자가 디버거를 통해 보호 대상 프로세스를 디버깅하고자 하였을 경우 디버거에 의해 대상 프로세스의 메모리가 변경될 수 있는데, 이 부분은 디버거에 대한 시그니처를 별도로 등록하여 프로파일 식별방법과 병행하여 해결할 수 있었다. 실험결과 시그니처를 획득하는 알고리즘은 시스템에 큰 영향을 주지 않지만, 시스템 와이드 훅의 경우는 시스템이 수행하여야 하는 메시지의 처리 과정을 증가시키므로 전체 시스템의 프로세스에 영향을 미치므로 시스템의 성능을 저하시킬 수 있는 점을 고려하여 시스템 프로세스를 제외하고 *:\test\loaders 내에서 동작하는 프로세스에 대해서만 후킹을 하도록 하였다.

로더 기능과 구현 방법에 따라 API 사용이 달라질 수 있으며 자동 제작 로더를 사용하여 로더를 제작하였을 경우 로더가 사용하는 API에 대한 변화는 없고 파라미터상의 오프셋만 변경된 경우가 일부 있었다.

수집한 20여개의 악성 로더 중 일부는 실제 타깃이 되는 프로그램이 설치되어 있거나 실행중일 경우만 동작하는 경우가 발생하였으며, 로더의 동작을 파악하기 위해 로더가 해당 프로세스를 찾는 방법에 따라 임의 타깃 프로그램을 구현하여 실험 대상으로 사용하였다. 예를 들어 User32.dll내의 FindWindowA, FindWindowW를 사용하여 프로세스의 클래스 이름이나 윈도우 이름을 이용하여 타깃 프로그램의 프로세스 ID를 얻어오는 로더의 경우 실험을 위해 윈도우 캡션 이름을 로더가 검색하는 이름으로 구현하여 실험을 진행하였으며 20여개의 로더 중 일부는 시험 대상 프로그램의 구성요소가 없어 실험을 진행 할 수가 없었다. 실험 가능한 로더를 대상으로 프로파일 레코드를 생성하였는데, 7개의 악성로더에서 생성된 레코드 수는 <표 2>에서와 같은 결과를 얻을 수 있었다.

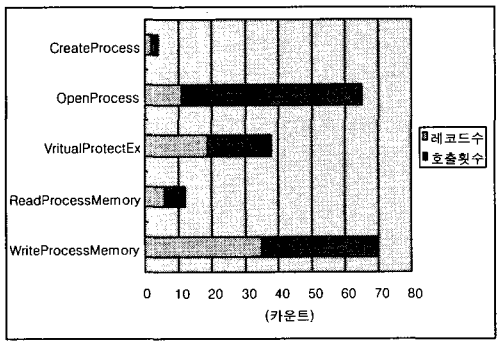
CreateProcess, OpenProcess의 경우 파라미터 사용에 있어 동일한 실행파일 또는 프로세스에 접근하는 것에 대한 레코드 수는 중복횟수로 판단하였고, VirtualProtectEx, ReadProcessMemory, WriteProcessMemory 등의 일부 API는 동일한 프로세스로의 접근일지라도 접근하는 메모리 위치가 다를 경우 보호 대상 프로세스에 대한 접근 및 변경이 발생 할때 마다 별도의 레코드로 생성하였다.

〈표 2〉 프로파일 레코드 수

로더명	API	호출 횟수	레코드수	비율 (%)
fs_fn.exe	OpenProcess	19	2	10.50
	VirtualProtectEx	11	11	100
	WriteProcessMemory	11	11	100
freebox.exe	CreateProcess	1	1	100
	ReadProcessMemory	1	1	100
	WriteProcessMemory	1	1	100
fdx-pop.exe	VirtualProtectEx	1	1	100
	OpenProcess	21	3	14.28
	VirtualProtectEx	6	6	100
	ReadProcessMemory	5	5	100
	WriteProcessMemory	5	5	100
	OpenProcess	2	2	100
freehack.exe	WriteProcessMemory	2	2	100
	CreateProcess	1	1	100
asx-ds2.exe	VirtualProtectEx	1	1	100
	WriteProcessMemory	1	1	100
Speeder XP.exe	OpenProcess	5	1	20
	WriteProcessMemory	12	12	100
FTrainer.exe	OpenProcess	7	3	42.85
	WriteProcessMemory	3	3	100

로더별로 사용되는 API에 대한 호출 빈도수를 측정해본 결과 (그림 7)과 같음을 알 수 있었으며 대체로 로더 구현에 사용되는 API의 주류를 파악할 수 있게 되었다. 로더를 이용하여 대상 프로세스를 실행하는 대신 OpenProcess를 이용하여 기존에 실행중인 프로세스에 접근이 많이 발생하였으며, WriteProcessMemory 함수의 사용이 많은 이

유는 대부분 악성 로더가 한번 오픈한 프로세스에 대해 데이터를 읽어올 필요가 없고 이미 알고 있는 오프셋에 원하는 데이터를 쓰기 때문이다.



(그림 7) 주요 API 호출 빈도 수

5. 결 론

본 논문에서는 악성 로더를 이용하여 어플리케이션에 대한 해킹 위협에 대한 방어를 할 수 있는 프로파일 기반의 보안 방법을 제시하고, 이에 대한 구현 및 시험을 수행하였다. 최근까지도 안티바이러스나 안티스파이웨어 프로그램에서 악성로더에 대해서는 탐지 및 치료 대상으로 포함시키지 않고 있다. 다만 로더 제작에 있어 특정 코드를 삽입한 대상에 한하여 안티바이러스 프로그램 일부에서 탐지를 하고 있지만, 시스템에서 허용된 Win32 API만을 사용할 경우 정상 프로그램과 분류할 방법이 없는 실정이다.

제시된 프로파일 기반의 악성 로더 탐지 기법은 별도의 독립적 에이전트로 구성이 가능하며, DLL로 구현하여 어플리케이션 개발시 직접적으로 적용이 가능하도록 설계 및 구현되었다. 일부 안티바이러스 프로그램에서 사용하는 시그니처 기반의 탐지 방법은 새로운 로더에 대한 즉각적인 탐지가 불가능하다는 단점을 가지고 있다. 제시된 프로파일 식별자에 의한 탐지 방법을 병행하게 되면 기존 탐지 대상이 되는 로더 뿐만 아니라 새로 수집된 정보를 기반에 대해 별도의 정책을 적용하여 보호 대상 어플리케이션의 실행파일 이미지가 변경되어 조작할 오프셋이 변경될 경우 자동으로 메모리 오프셋을 적용 갱신하여 주는 dUP와 같은 자동 로더 생성기에 의해 생성된 신규 로더에 대한 접근 및 탐지에 효과적이라고 할 수 있다[12].

향후 연구에서는 성능향상을 위해 시스템 성능 저하를 발생시키는 시스템 와이드 훅에 대한 대안을 마련하고, 팩된 로더에 대한 정보를 수집 및 자동 언패킹 모듈을 분석 및 구현함으로써 성능을 개선하는 연구를 수행할 것이다.

참 고 문 헌

[1] Loader, http://biw.rult.at/coding/loader.htm
[2] CheatEngine, http://blog.naver.com/sjhw1209?Redirect=Log&logNo=10006279503
[3] XCheater, http://www.xcheater.com/cheattools.aspx
[4] HACK ProFeSsioAl, http://cafe.daum.net/HACKProFeSsionAl
[5] starmap0, http://cafe.daum.net/starmap0
[6] ProcessGuard, http://www.diamondcs.com.au/processguard/
[7] Min Chen, "Software Product Protection", 2001.
[8] Bill Horne, Lesley Matheson, Casey Sheehan, Robert E. Tarjan, "Dynamic Self-Checking Techniques for Improved Tamper Resistance", Digital Rights Management Workshop, 2001
[9] Packer&Unpackers, http://www.woodmann.com/crackz/Packers.htm
[10] Predator, "Unpacking : A Generic Approach, Including IT Rebuilding", 2001.
[11] McCodEMaN, "A Manual Unpacking Essay

: unUPX v1.06", 2000.

[12] diablo2oo2's Universal Patcher, http://programmerstools.org/node/97

[13] Shub-Nigurrath, ThunderPwr, "Cracking with Loaders : Theory, General Approach and a Framework", CodeBreakers Journal, 2005.

[14] Hide Debugger, http://www.exetools.com/forum/ showthread.php?t=5299&page=1

[15] OllyDbg, http://www.ollydbg.de/

[16] Stone. "In Memory Patching : Three Appro-Aches(how to introduce breakpoints in an automated debugger and other marvels)", 1997.

[17] MSDN Library, http://msdn.microsoft.com/library/

[18] P.C. van Oorschot, "Revisiting Software Protection", ISC 2003.

[19] A. Main, and P.C. van Oorschot, "Software Protection and Application Security : Understanding the Battleground", International Course on State of the Art and Evol-ution of Computer Security and Industrial Cryptography, 2003.

[20] Woodmann, http://www.woodmann.com/forum/index.php

[21] BiW Reversing, http://www.reversing.be/

윤 이 중

2002년 충남대학교 컴퓨터공학과(박사)

1988년~2000년 한국전자통신연구원(ETRI)

2001년~현재 국가보안기술연구소

김 요 식

2004년~현재 국가보안기술연구소