윈도우 환경에서의 메모리 해킹 방지 시스템 연구

김요식* · 윤영태* · 박상서*

요 약

역공학(Reverse Engineering) 기술이 진보함에 따라 컴퓨터 소프트웨어에 대한 불법 조작 및 변조 등의 위협이 증가하고 있으며, 인터넷에 공개된 단순한 도구를 이용하여 누구나 쉽게 크래킹(Cracking)을 할 수 있게 되었다. 자사의 소프트웨어를 위협으로부터 방어하고자 하는 제작사들의 노력과 이를 무력화시키고자 하는 소위 크래커들의 노력은 지금까지도 계속되고 있다. 이에 본 논문에서는 소프트웨어가 가지는 위협 모델과 크래킹 기술에 대해 분석 및 실험하고, 소프트웨어를 위협으로부터 보호하기 위한 윈도우 환경에서의 메모리 해킹 방지 시스템을 제안한다.

A Study on Memory Hacking Prevention System in Windows Environment

Yosik Kim* · Youngtae Yun* · Sangseo Park*

ABSTRACT

Recently, illegal manipulation and forgery threats on computer softwares are increasing due to the advances in reverse engineering techniques. Furthermore someone who has concerns about these area can crack the software by using the open-to-public simple tools on the internet. The software companies are struggling to defend their own softwares against threats, while the crackers are continuing to crack the softwares. In this paper, we first establish the generic software threat model and, analyze and experiment on the software cracks, before suggest a memory hacking prevention system in Microsoft Windows environment.

Key words : Memory Hacking, Packing, Unpacking, Reverse Engineering, Cracking

* 국가보안기술연구소

1. 서 론

인터넷의 활성화로 인해 소프트웨어 개발사들은 자사의 제품에 대한 저작권 보호를 위해 더 많은 시간과 비용을 들여야 했다. 반면 소위 "소프트웨어 해커"라 불리는 크래커들은 자신들의 노하우를 보다 쉽게 공유할 수 있게 되었으며 단순한 도구나 역공학을 이용한 크래킹을 통해 자신들의 요구에 맞도록 프로그램 코드를 수정하거나 무력화 시킬 수 있게 되었다[1, 2].

저작권을 보호하기 위한 기술로는 사용자에게 등록번호를 입력하도록 하고 올바른 등록번호 여부를 검사하는 방법과 소프트웨어에 보이지 않는 워터마크를 주입하는 방법, 하드웨어를 접목시켜 소프트웨어를 보호하는 기술들이 있다. 그러나 새롭게 고안된 알고리즘이나 기술은 고도화된 크래킹 기술에 그 생명 주기를 오래 가져가지 못하고 있는 실정이다.

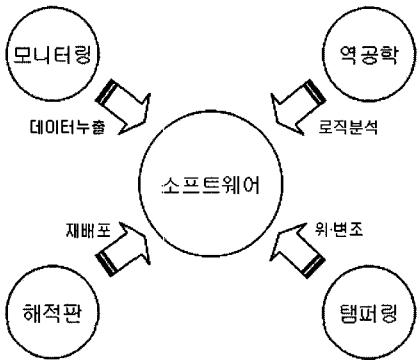
또한, 최근들어 오프라인에서 동작하는 패키지 소프트웨어가 점차 사라지고 대다수 소프트웨어들이 온라인상에서 운영됨에 따라 크래킹에 직접적으로 노출되고 있어 위·변조뿐 아니라 다양한 공격의 대상이 되고 있다.

이러한 문제점들을 해결하기 위해 본 논문에서는 2장에서 일반적으로 소프트웨어가 가지는 위협모델을 분석하여 문제점을 도출하고, 소프트웨어에 대한 크래킹 방법에 대한 예를 보이도록 한다. 3장에서는 소프트웨어를 위협으로부터 보호할 수 있는 대응책으로 윈도우 환경에서의 메모리 해킹 방지 시스템을 제안하고, 4장에서 결론 및 향후 방향을 제시한다.

2. 소프트웨어 크래킹 기술

2.1 위협 모델

대부분 소프트웨어는 사용자로부터 입력데이터를 입력 받아 정상적인 데이터 여부를 검사한 후 결과를 사용자에게 보여주고, 입력된 데이터와 내부 데이터를 통해 처리 로직에 따라 동작하는 형태의 루틴을 포함하고 있다. 소프트웨어가 처리하는 일련의 과정은 모니터링 및 역공학 등의 방법을 통하여 내부 비밀 데이터의 추출과 처리되는 로직을 임의 조작하여 프로그램의 흐름을 초기 제작 의도와 다르게 변경하여 사용할 수 있는 취약점을 가지고 있다.



(그림 1) 소프트웨어 위협 모델

디지털 정보인 소프트웨어는 임의 조작을 통해 원본과 동일한 복사본을 만들 수 있으며, 패키지 전체를 무작위로 재배포 할 수 있다.

컴퓨터상에서 동작하는 어떤 소프트웨어라도 다음과 같은 공격의 목표가 될 수 있다.

- 모니터링
- 역공학
- 소프트웨어 해적판(Software Piracy)
- 탬퍼링

2.2 크래킹 기술

크래킹 기술은 소프트웨어가 사용하는 방어기술에 따라 달라지며, 그에 따른 분석과 크래킹 기술이 달리 존재한다.

등록키(Registration Key), 다중 일련번호(Multiple Serials)와 같은 단순한 등록번호를 사용할 경우, 등록번호를 비교하는 코드상의 분기문을 수정하거나 디버깅을 통해 하드코딩된 등록번호를 알아내거나 등록번호를 생성하는 알고리즘을 분석하여 등록번호만을 전문적으로 생성하는 프로그램을 제작할 수 있다. 또한 패커(Packer)에 의해 압축·암호화되어 있는 코드도 메모리상에서 언패킹(Unpacking) 되어 실행된다는 취약점을 가지고 있어 매뉴얼 언패킹을 이용하면 패커 기능을 어렵지 않게 무력화 시킬 수 있다.

소프트웨어의 불법적인 복사를 방지하기 위한 기술로 컴퓨터의 I/O 포트에 작은 하드웨어를 부속시키는 방법인 동글(Dongle)에 대해서는 위조된 드라이버(Driver)를 제작하거나 별도의 에뮬레이터(Emulator)를 이용하여 동글없이도 소프트웨어를 동작할 수 있는 크래킹 방법이 사용되고 있다[3-5].

2.3 크래킹 방법

본 절에서는 단순한 내그 스크린(Nag Screen)을 무력화 시키는 방법과 UPX(The Ultimate Packer for eXecutables)로 팩된 프로그램에 대한 프로세스 패치(Process Patch) 방법에 대해 설명하고, 마지막으로 메모리 조작에 자주 사용되는 로더(Loader)를 이용한 방법에 대해 설명한다.

2.3.1 내그 스크린 무력화

내그 스크린이란 흔히 쉐어웨어(Shareware)에서 많이 사용하는 것으로 특별히 시간제한을 두거나 매번 프로그램을 수행할 때마다 구매를 유도하거나 홍보하는 다이얼로그 박스를 화면에 몇초간 출력하는 비교적 단순한 프로텍션 기술에 속한다. 본 논문에서는 Helios사의 TextPad ver4.7을 분석하였다.

일반적인 내그 스크린을 사용하는 프로그램의 경우 몇 초간 다이얼로그를 출력하기 위해 시간을 제어하는 GetSystemTime, GetLocalTime, GetTickCount와 같은 Win32 API를 사용한다. TextPad의 경우 (그림 2)와 같이 GetTickCount를 사용하기 때문에 GetTickCount를 호출하는 위치에 브레이크 포인트를 설정한다.

```
004055A3 . 8B01          MOV EAX,DWORD PTR DS:[ECX]
004055A5 . FF90 58010000 CALL DWORD PTR DS:[EAX+158]
004055AB . 85C0          TEST EAX,EAX
004055AD   74 24         JE SHORT TextPad.004055D3
004055AF . 8B8F B0000000 MOV ECX,DWORD PTR DS:[EDI+B0]
004055B5 . 6A 05         PUSH 5
004055B7 . E8 F2341000   CALL TextPad.00508AAE
004055BC . 8B87 B0000000 MOV EAX,DWORD PTR DS:[EDI+B0]
004055C2 . FF70 1C       PUSH DWORD PTR DS:[EAX+1C]
004055C5 . FFD3          CALL EBX
004055C7 . FF15 48C35300 CALL DWORD PTR DS:[<&KERNEL32.GetTickCou>
004055CD . 8987 AC000000 MOV DWORD PTR DS:[EDI+AC],EAX
```

(그림 2) TextPad.exe의 소스코드 일부

GetTictCount API는 내그 스크린이 시작되어 경과된 시간을 밀리초(millisecond) 단위로 리턴하므로 내그 스크린을 화면상에 출력하는 부분은 GetTicCount API가 호출되기 앞부분이라는 것과 내그 스크린 윈도우의 호출 여부를 결정하는 조건도 앞부분에서 계산된다는 것을 유추할 수 있다. 이는 디버거로도 확인할 수 있는데, 메모리 주소 0x004055AD 위치가 바로 내그 스크린의 호출 여부를 결정짓는 조건 분기문이 된다. 0x004055A5의 CALL문은 내그 스크린을 호출하는 구문이며, "TEST EAX, EAX"의 AND연산을 거쳐 0x004055AD의 JE 구문을 통해 등록버전의 경우 네그 스크린을 호출하지 않는 위치인 0x004055D3로 점프하도록 구현되어 있다.

여기서, 0x004055AD의 조건 분기 구문인 JE의 코드를 무조건 분기 명령인 JMP로 단지 1바

이트만을 수정하면 비등록버전의 경우라도 내그 스크린은 호출되지 않는다.

0x004055AD에 위치한 "74 24"의 OP코드(Operation Code)를 무조건 분기 명령인 JMP문을 적용하여 "EB 24"로 수정하면 간단하게 패치할 수 있다[10].

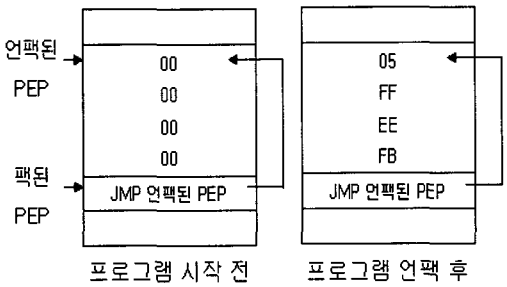
2.3.2 프로세스 패치

최근 인터넷을 통해 배포되는 프로그램들은 대부분 프로텍션 기능의 일환으로 패커를 사용하거나 프로그램 바이너리에 대한 역공학을 방지하기 위해 디버거 자동 감지 기능을 포함하고 있다. 프로세스 패치란 팩(Pack)되어 있는 프로그램의 언패킹이나 암호화 코드를 제거하는 것이 불가능할 경우 실행시의 탬퍼할 루틴을 찾아 영구적으로 패치할 수 있는 기술을 말한다[2].

실험을 위해 앞서 언급했던 TextPad의 실행파일인 TextPad.exe를 UPX를 이용하여 (그림 3)과 같이 팩하였다.

```
C:\Program Files\TextPad 4>\upx120w\upx TextPad.exe
   Ultimate Packer for eXecutables
   Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001
   UPX 1.20w Markus F.X.J. Oberhumer & Laszlo Molnar May 23rd 2001
   File size Ratio Format Name
   -------------------- ------ ----------- -----------
   1900544 -> 756224 39.79% win32/pe TextPad.exe
   Packed 1 file.
```

(그림 3) UPX 실행 화면



(그림 4) 패커 동작 형태

패커를 통해 압축/암호화된 코드는 메모리 상에서 프로그램을 수행하기 위해 언팩 과정을 수행하게 된다. (그림 4)는 패커가 메모리상에 팩된 코드를 언팩하기 전과 후의 상황을 도식화한 것으로 언팩 과정을 거치면서 패킹된 프로그램 엔트리 포인트(Program Entry Point)에서 언팩된 엔트리 포인트로 분기하는 코드의 모습을 나타낸다.

(그림 4)의 과정을 디버거를 통하여 살펴보면 프로그램이 실행되기 전의 메모리 내용인 0x004BD302의 코드가 "00 00 00…"으로 구성되어 있는 것을 확인할 수 있다.

```
005E2BC0   83C3 04        ADD EBX,4
005E2BC3 ^ EB D8          JMP SHORT textpad .005E2B9D
005E2BC5   FF96 508C1E00  CALL DWORD PTR DS:[ESI+1E8C50]
005E2BCB   61             POPAD
005E2BCC - E9 31A7EDFF    JMP textpad .004BD302
005E2BD1   0000           ADD BYTE PTR DS:[EAX],AL
005E2BD3   0000           ADD BYTE PTR DS:[EAX],AL
```

(그림 5) 패커의 마지막 부분

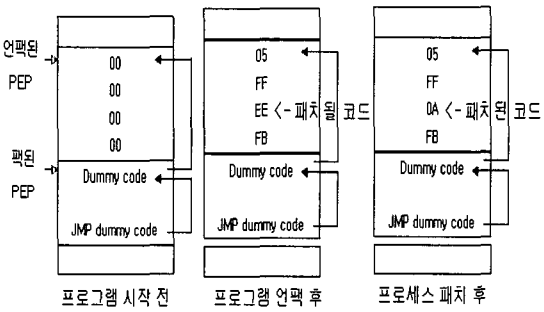
(그림 5)는 압축·암호화된 TextPad.exe 파일을 디버거를 이용하여 패커의 마지막 부분에 원래의 TextPad.exe가 가지는 프로그램 엔트리 포인트인 0x004BD302로 점프하는 구문이 포함되어 있는 것을 나타낸다. 이는 UPX를 통해 압축·암호화를 수행하였더라도, 프로그램 원래의 엔트리 포인트와 앞서 1바이트 패치를 통해 내그 스크린을 무력화시킨 코드의 메모리상의 위치에는 변화가 없다는 것을 알 수 있다.

(그림 6)은 전체적인 프로세스 패치 절차 과정을 나타낸다.

(그림 6)의 Dummy code는 TextPad.exe내의 섹션과 섹션사이의 슬랙 공간(Slack Space)나 "0"으로 채워진 일정 공간에 다음과 같은 코드로 작성된다.

```
push eax
mov eax,404ED0h
mov byte ptr [eax],0EBh
pop eax
jmp 004A038E
```

위의 코드는 0x404ED0의 위치에 해당하는 코드를 JMP문의 OP코드인 0xEB로 수정하고 언팩된 후의 엔트리 포인트인 0x004A038E로 점프하도록 하는 코드이다.

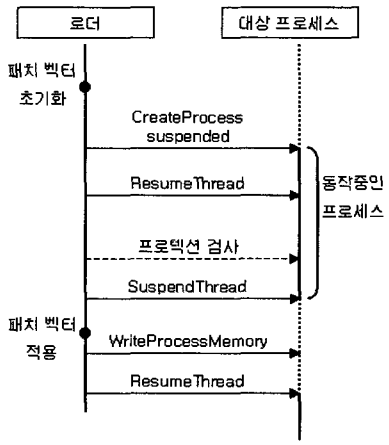


(그림 6) 프로세스 패치 절차

프로세스 패치 절차에 따라 UPX가 가지는 루틴을 패치하면 앞서 실험과 동일하게 TextPad의 나그 스크린을 화면상에 보이지 않도록 하는 동일한 결과를 얻을 수 있다. 이와 같은 과정이 가능한 이유는 먼저 UPX가 가지는 섹션 자체가 실행이 기능한 영역이기 때문이며, UPX를 통해 압축·암호화되어 있는 프로그램이더라도 프로그램이 실행되는 시점에서는 압축·암호화되어 있는 코드 전체가 언팩 과정을 통해 압축해제 및 복호화 되기 때문이다.

2.3.3 로더를 이용한 패치

로더를 이용한 패치는 로더라는 별도의 프로그램을 제작하여 대상 프로그램이 가지는 메모리영역의 권한과 코드를 수정하는 방법이다. 아직까지도 각종 소프트웨어에 적용가능한 방법이며 국·내외에서 최근 웹 하드 클라이언트와 게임 소프트웨어가 그 대상이 되고 있다. 로더를 이용한 기본적인 패치 절차는 (그림 7)과 같다.



(그림 7) 로더 동작 절차

패치 벡터 초기화는 대상 프로그램이 정해지면 디버거 등을 통해 대상 프로그램을 분석하여 패치하고자 하는 주소 위치와 변경할 데이터 등을 초기화하는 과정이다. 초기화 과정이 종료된 후 로더는 (그림 7)과 같이 CreateProcess API를 이용하여 대상 프로그램을 직접 호출한 후 WriteProcessMemory API의 파라미터에 패치 벡터를 적용하여 메모리를 조작한다. 로더는 이미 실행중인 프로세스에 대해서도 메모리 조작이 가능한데, <표 1>과 유사한 형태의 Win32 API를 사용한다.

〈표 1〉 로더제작에 자주 사용되는 API

Win32 API	기 능
GetWindowThreadProcessId	대상 프로세스에 대한 ID를 리턴
OpenProcess	대상 프로세스 개방
VirtualProtectEx	패치하고자하는 주소의 권한을 변경
WriteProcessMemory	패치하고자하는 주소에 패치 벡터를 적용

2.4 크래킹 도구

소프트웨어가 가지고 있는 방어 기능을 무력화시키기 위해서는 분석 및 크래킹 도구 없이는 불가능하다. 이러한 용도로 사용되는 도구는 크게 역공학 도구와 시스템 모니터링 도구로 구분할 수 있다.

2.4.1 역공학 도구

역공학 도구는 프로그램 내부의 로직을 분석하는데 사용된다. 도구의 기능을 이용하여 프로세스 내부 로직을 분석하고 취약점과 익스플로잇(Exploit)을 제작할 수 있다.

역공학 용도로 사용되는 도구로는 프로그램의 정적 로직에 대한 분석을 위한 역어셈블러(Disassembler)와 디컴파일러(Decompiler)가 있으며 프로그램이 실행되는 동안의 상태에 대한 분석을 위한 도구인 디버거(Debugger)가 있다.

(1) 역어셈블러/디컴파일러

역어셈블러는 컴파일된 코드를 분석하여 어셈블리 코드와 대응되는 코드를 생성, 하위레벨의 소스코드를 생성하는데 사용된다.

C/C++의 바이너리 파일에 대한 디컴파일만으로는 완벽한 원시 코드를 추출하기가 어렵지만, 자바의 바이너리 파일에 대한 디컴파일은 1:1 매핑에 가깝게 얻을 수 있다. 윈도우 환경에서 사용되는 대표적인 역어셈블러는 W32Dasm으로 바이너리 코드에서 내부 프로그램 구조와 유용한 정보를 추출하는 것이 가능하다.

W32Dasm은 다음과 같은 윈도우상의 역어셈블러/디버거 기능을 가진다.

- 16 · 32비트 윈도우 프로그램에 대한 역어 셈블 지원
- MMX 명령에 대한 역어셈블 지원
- 익스포트, 임포트, 메뉴, 다이얼로그, 텍스트 참조 정보 디스플레이
- 32비트 프로그램을 위한 통합 디버거

(2) 디버거

디버거는 프로세스를 에뮬레이트하면서 동작하는데, 마치 프로세스와 직접 상호 작용하는 것처럼 디버거 내에서 프로그램을 실행시키며 런타임 실행과 메모리 및 레지스터의 내용, 브레이크 포인트(Break Point) 등을 설정하고 트레이스(Trace)할 수 있는 도구이다. 디버거에는 크게 2가지의 종류가 있는데, 응용프로그램 수준의 디버거와 시스템 수준의 디버거가 있다. 응용프로그램 수준 디버거는 OS와 디버깅하는 프로그램 사이에서 동작하며 시스템 레벨 디버거는 프로세서와 OS사이에서 동작한다. 시스템 레벨 디버거의 경우는 OS 자체의 디버깅까지 가능하다.

다음은 디버거에서 자주 사용되는 기능들이다.

- Step over, Step into와 같은 소스 구문 에 각각에 대한 실행
- 브레이크 포인트를 통한 지정한 위치까지의 프로그램 실행
- 와치(Watch)기능을 통한 디버깅중 변수 값 출력
- 프로그램 디버깅중 변수값 변경
- 런타임시 메모리와 레지스터 값에 대한 모니터링과 변경
- 역어셈블
- 스택(Stack) 모니터링

Compuware사의 소프트아이스(SoftIce)는 시스템 수준의 디버거로 싱글 또는 듀얼 머신상에서 디바이스 드라이버(Device Driver) 디버깅이 가능한 커널 모드 디버거(Kernel-mode Debugger)이다. 이 도구는 기존 전통적인 윈도우 SDK/DDL의 한계를 넘어 강력한 기능으로 디버깅 시

간을 현저하게 줄였으며, 다양한 윈도우 프로그램의 문제에 대해 이해·진단하기가 쉽고 시스템에 대해 전반적인 내용을 볼 수 있도록 지원한다.

그 밖에 많이 사용되는 도구로는 DataRescue사의 IDA Pro, 공개로 제공되는 OllyDBG 등이 있다[6-8].

2.4.2 시스템 모니터링 도구

모니터링 도구는 프로그램이 동작하는 동안 프로그램이 접근하는 파일이나 레지스트리의 상태 및 정보를 제공하여 주는 도구로 Sysinternals사의 FileMon과 RegMon이 대표적이다.

Filemon은 실시간으로 시스템상의 시스템 파일 동작에 대해 디스플레이에 해 줄뿐 아니라 윈도우의 동작을 모니터할 수 있으며, 응용 프로그램이 사용하는 파일, DLL, 또는 시스템이나 응용 프로그램의 문제점으로 인한 단서를 찾을 수 있도록 강력한 기능을 제공해 준다. 또한 모든 파일이 오픈, 읽고, 쓰고, 삭제되는 행위에 대한 결과를 타임 스탬프 형태로 확인할 수 있다.

Regmon은 윈도우 시스템의 레지스트리 모니터링 유틸리티로 특정 응용프로그램이 레지스트리에 접근하여 읽고 쓰는 상황을 실시간으로 모니터링 할 수 있도록 지원한다[9].

2.4.3 기타 도구

바이너리 파일에 대한 최종 패치 작업은 주로 Hex 에디터를 이용한다. Hex 에디터는 일반 Text 에디터와 마찬가지로, 바이너리 파일에 대한 편집 및 복사, 붙여넣기 등을 지원하며 바이너리 형태의 실행파일의 코드를 수정하여 패치 된 최종의 바이너리를 저장할 수 있다. 많이 사용되는 에디터로는 BreakePoint사의 Hex Workshop, IDM사의 Ultra Editor가 있다.

3. 윈도우 환경에서의 메모리 해킹 방지 시스템

3.1 시스템 개요

악성 프로그램의 일종인 로더는 Kernel32.dll, User32.dll 등의 시스템 DLL에서 제공하는 Win32 API를 이용하여 보호대상 프로세스에 대한 메모리를 조작하는 프로그램으로 <표 2>와 같은 비슷한 패턴의 Win32 API를 사용한다.

〈표 2〉 악성 프로그램의 Win32 API 사용 비교

프로그램	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

(그림 8)은 freehack.exe를 역어셈블한 코드 일부로 freehack.exe는 User32.dll내의 FindWindow를 이용하여 접근 대상 프로세스의 핸들(Handle)을 얻어오고 GetWindowThreadProcessId에 의해 프로세스의 ID를 획득한 후, OpenProcess, VirtualProtectEx, WriteProcessMemory, CloseHandle의 API를 호출하여 대상 프로세스가 가지는 특정 위치의 메모리를 조작하게 된다.

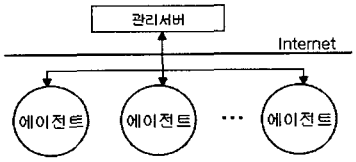
```
0040129B  /$ 55             PUSH EBP
0040129C  |. 8BEC           MOV EBP,ESP
0040129E  |. FF75 0C        PUSH DWORD PTR SS:[EBP+C]
004012A1  |. FF75 08        PUSH DWORD PTR SS:[EBP+8]
004012A4  |. E8 D50D0000    CALL <JMP.&USER32.FindWindowA>
004012A9  |. 0BC0           OR EAX,EAX
004012AB  |. 74 6E          JE SHORT fs0929_f.0040131B
004012AD  |. A3 5D414000    MOV DWORD PTR DS:[40415D],EAX
004012B2  |. 68 65414000    PUSH fs0929_f.00404165
004012B7  |. FF35 5D414000  PUSH DWORD PTR DS:[40415D]
004012BD  |. E8 E00D0000    CALL <JMP.&USER32.GetWindowThreadProcessId>
004012C2  |. FF35 65414000  PUSH DWORD PTR DS:[404165]
004012C8  |. 6A 00          PUSH 0
004012CA  |. 68 FF0F1F00    PUSH 1F0FFF
004012CF  |. E8 400E0000    CALL <JMP.&KERNEL32.OpenProcess>
004012D4  |. A3 61414000    MOV DWORD PTR DS:[404161],EAX
004012D9  |. 68 69414000    PUSH fs0929_f.00404169
004012DE  |. 6A 40          PUSH 40
004012E0  |. FF75 18        PUSH DWORD PTR SS:[EBP+18]
004012E3  |. FF75 10        PUSH DWORD PTR SS:[EBP+10]
004012E6  |. FF35 61414000  PUSH DWORD PTR DS:[404161]
004012EC  |. E8 350E0000    CALL <JMP.&KERNEL32.VirtualProtectEx>
004012F1  |. 6A 00          PUSH 0
004012F3  |. FF75 18        PUSH DWORD PTR SS:[EBP+18]
004012F6  |. FF75 14        PUSH DWORD PTR SS:[EBP+14]
004012F9  |. FF75 10        PUSH DWORD PTR SS:[EBP+10]
004012FC  |. FF35 61414000  PUSH DWORD PTR DS:[404161]
00401302  |. E8 250E0000    CALL <JMP.&KERNEL32.WriteProcessMemory>
00401307  |. FF35 61414000  PUSH DWORD PTR DS:[404161]
0040130D  |. E8 D20D0000    CALL <JMP.&KERNEL32.CloseHandle>
```

(그림 8) 악성 프로그램 코드 일부

이런 위협으로부터 소프트웨어를 보호하기 위한 대응책으로 본 논문에서는 <표 2>와 같은 악성 프로그램의 Win32 API 사용 패턴과 실행 모듈에 대한 해쉬 검증, 그리고 시스템 와이드 훅(System Wide Hook)을 이용하여 클라이언트 시스템상에서 메모리를 조작하는 악성 소프트웨어를 모니터하고 임의 조작을 방지하는 시스템을 제안하고자 한다.

3.2 시스템 구성

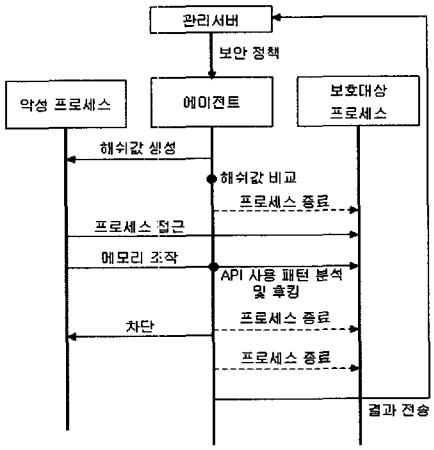
시스템은 (그림 9)와 같이 클라이언트상에서 보호대상 프로세스에 대한 디버깅과 메모리 조작 행위를 방지하며, 의심가는 행위에 대한 모니터링 데이터를 수집하는 에이전트와, 에이전트로부터 데이터를 전송받아 저장 관리하며 보안 정책을 설정하는 서버로 구성된다.



(그림 9) 시스템 구성도

시스템 주요 기능은 다음과 같다

- Win32 API 사용 패턴을 이용한 악성 프로그램 동작 감지 기능
- 악성 프로그램의 실행모듈에 대한 해쉬값 생성 및 악성 프로그램 탐지 기능
- 시스템 와이드 훅을 이용한 메모리 조작 방지 기능
- 보고된 탐지 패턴 정보를 기반으로 보안 정책 생성 및 적용 기능



(그림 10) 시스템 동작 절차

시스템은 (그림 10)과 같이 서비스 시작전 서버로부터 보안 정책을 다운받고, 실행되는 새로운 프로세스의 실행을 감지한다. 이와 동시에 프로그램 모듈에 대한 해쉬값을 구한 뒤 관리 서버로부터 수신된 악성 프로그램에 대한 해쉬값 리스트와 비교하여 악성 프로그램으로 등재된 프로그램인지 여부를 검사한다. 이때 악성 프로그램으로 판명이 나면 보호 대상 프로세스를 종료한다. 악성 프로그램이 아닌 것으로 판단된 경우에는 보호대상 프로그램으로의 접근 및 사용하는 Win32 API 패턴을 분석한다. 만약 메모리 조작으로 판단될 경우에는 보호대상 프로세스를 종료하거나 악성 프로그램의 메모리조작을 차단

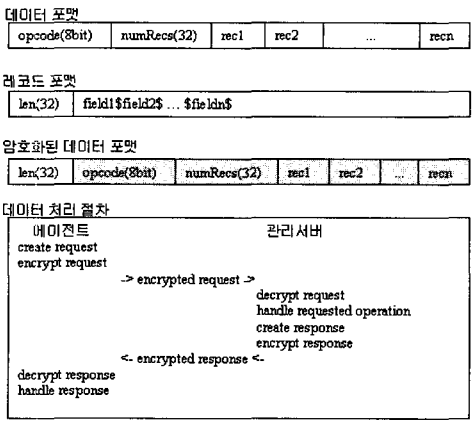
하고 일련의 과정에 대한 모니터링 결과를 서버로 전달한다.

시스템 수행에 적용되는 보안 정책은 다음과 같다.

- 프로그램에 대한 해쉬값 생성시 적용되는 해쉬 알고리즘
- 에이전트와 서버와의 통신에 적용되는 세션 암호화 알고리즘
- 차단 대상 프로그램 해쉬 목록
- 차단 대상 프로그램 탐지시 보호 대상 프로세스 바로 종료 여부
- 차단 대상 프로그램 탐지시 대상 프로그램 종료 시도 여부
- 에이전트 무결성 검사 여부
- 로그 저장 방법

3.3 통신 프로토콜

에이전트와 관리서버간에 송·수신 되는 모든 데이터는 기본적으로 암호화되어 전송이 이루어진다. (그림 11)은 프로토콜에 적용되는 데이터 포맷, 레코드 포맷 암호화된 데이터의 포맷을 나타낸다. 처리되는 데이터의 분리자는 "$"를 사용하며 특정 필드의 값이 정의되어 있지 않았을 경우 분리자를 "$$"와 같이 연속해서 표시한다.



(그림 11) 오퍼레이션 프로토콜

3.4 에이전트

클라이언트 시스템에서 프로세스들의 행위를 실시간으로 모니터한다. 구성 모듈과 각 모듈의 기능을 요약하면 <표 3>과 같다.

〈표 3〉 에이전트 모듈 구성과 기능

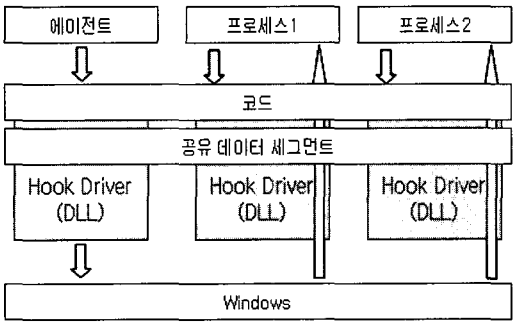
구 성	기 능
해쉬생성 모듈	MD5를 이용한 실행 파일에 대한 해쉬값 생성(h=md5[.exe]||md5[.dll])
조작에 대한 탐지 및 방지 모듈	Win32 API 패턴기반의 프로세스에 대한 악의적 접근 및 메모리 조작 탐지·차단
분석 방지 모듈	보호대상 프로그램에 대한 디버거, 치터 등의 동작을 탐지
전송 모듈	수집한 데이터를 서버로 전송 (hash||process_name||caption_string||class_name||api_pattern)

에이전트는 보호대상 프로세스와는 별개의 프로그램으로 동작할 수 있으며 또는 DLL로 제공되어 보호대상 프로세스가 동적으로 사용할 수도 있다. 에이전트에 대한 전반적인 동작 절차는 다음과 같은 순서로 이루어진다.

① 보안정책을 초기화한다.
② 실행 프로그램에 대한 해쉬값을 생성한다.
③ 해쉬값 비교후 차단대상 프로그램인지를 검사한다.
④ 차단대상 프로그램일 경우 프로세스에 대한 정보를 관리서버로 전송하고 보호대상 프로세스를 종료한다.
⑤ 보호 대상 프로세스로의 접근과 디버거 동작을 탐지한다. 디버거가 동작이 탐지되면 디버거가 가지는 프로세스 정보를 관리서버로 전송하고 보호대상 프로세스를 종료한다.
⑥ 사용하는 API 패턴과 파라미터를 분석한다.
⑦ 보호 대상 프로세스에 대한 메모리 조작 여부를 검사한다.

⑧ 메모리 조작이 탐지되면 시스템 와이드 훅을 이용하여 메모리 조작에 사용되는 Win32 API를 차단하고 프로세스 정보를 관리서버로 전송하며 보호 대상 프로세스를 종료한다.

보호대상 프로세스에 대한 메모리 조작 판단 여부를 검사하기위해 사용되는 후킹은 시스템 와이드 후킹의 기본 요구 사항을 충족시키기 위해 보통 DLL로 구현된다. 후킹은 기본적으로 후킹 콜백 프로시져가 시스템 상에서 후킹된 각각의 프로세스의 주소 공간에서 실행되는데, 에이전트는 로더 프로세스가 접근하는 프로세스가 보호대상 프로세스인지를 판단한 후, 로더가 호출하는 Win32 API에 대해 모니터한다. 메모리 조작이 행하여진다고 판단할 경우 시스템 와이드 훅을 이용하여 로더가 사용하는 Win32 API 기능을 무력화하여 메모리 조작을 사전에 방지한다. 악의적인 프로세스의 동작에 대한 모니터링을 하는 모듈은 (그림 12)와 같은 프로세스 후킹 구조를 가진다.



(그림 12) 프로세스 후킹 구조

디컴파일러와 디버거의 동작을 탐지하는 모듈은 다양한 종류의 분석 도구에 대해 계속 적인 패턴 관리가 필요하다. 분석 도구의 탐지는 (그림 13)과 같이 분석 도구가 사용하는 고유 캡션 스트링(Caption String)과 실행파일 이름을 비교하거나 분석 도구의 사용으로 인해 발생하는 특정 인터럽트(Interrupt)등을 모니터하여 탐지한다.

분석 도구를 탐지하기 위한 방법은 다음과 같다.

- FindWindow API를 이용한 분석도구의 윈도우 정보 비교
- CreateToolhelp32Snapshot, Process32First/Next API를 이용하여 분석도구 프로세스가 사용하는 힙, 모듈, 쓰레드 정보 비교
- SetUnhandledExceptionFilter API를 이용한 예외처리 핸들러 이용
- IsDebuggerPresent API를 이용한 디버거 동작 여부 감지
- 분석 도구가 사용하는 고유 메모리 영역 및 파일 정보 비교

프로세스명을 비교하는 방법은 기존에 실행되고 있는 프로세스명을 얻어오거나, 새로 실행되는 프로세스명을 별도로 관리하고 필터링하여 실행여부를 결정짓는다. 관리 대상 프로세스나 인가되지 않은 새로운 프로세스의 경우 보안정책 설정에 따라 실행중인 프로세스 자체를 종료시킬 수 있다.

```
/Title = "API-Log v1.2 by M.o.D. [F2F]"
|Class = "#32770"
WFindWindowA

/Title = "VxD Monitor"
|Class = "VxDMonClass"
WFindWindowA

/Title = "File Monitor"
|Class = "FileMonClass"
WFindWindowA

/Title = "Registry Monitor"
|Class = "RegmonClass"
WFindWindowA

/Title = "URSoft W32Dasm Ver 8.93
   Program Disassembler/Debugger"
|Class = "OWL_Window"
WFindWindowA

/Title = "TRW2000 for Windows 9x"
|Class = "#32770"
WFindWindowA

/Title = "OllyDbg"
|Class = "OLLYDBG"
WFindWindowA
```

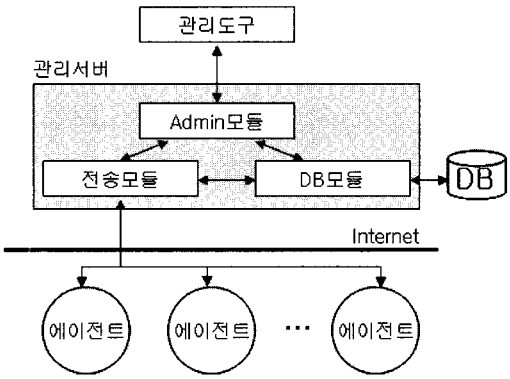
(그림 13) 분석 도구들의 윈도우 정보

전송 모듈은 클라이언트 시스템 상에서 에이전트가 수집한 데이터들에 대한 전송을 담당한다. 전송되는 데이터는 아래와 같다.

- 프로그램 실행 모듈 해쉬값
- 실행 모듈 및 프로세스 이름
- 프로그램의 캡션 스트링 및 클래스 이름
- Win32 API 사용 데이터

3.5 관리 서버

서버는 에이전트를 업데이트하거나 탐지된 데이터를 바탕으로 새로운 보안 관리 정책을 생성하여 개별 에이전트에 배포하는 역할을 수행한다. 서버 관리자는 메모리 조작과 같은 악의적인 접근에 대해 에이전트로부터 보고된 데이터에 대한 이벤트 로그 및 위험도를 바탕으로 악성 프로세스 여부를 판단한다. 악성 프로세스로 판단된 경우 이를 데이터베이스에 저장하고 동시에 에이전트에게 해당 패턴 정보 및 보안 정책을 배포하여 추후 동일한 악성 프로세스가 재실행되었을 경우 프로세스를 강제로 종료하도록 하여 메모리 조작을 사전에 방지하도록 한다. 관리서버 구조는 (그림 14)와 같다.



(그림 14) 관리서버 운영 구조

Admin모듈은 전송모듈 및 DB모듈을 초기화하고 구동시키며 GUI형태로 제공되는 관리도구와 연동하여 데이터베이스에 저장된 내용에 대한 통계 및 내역을 조회할 수도 있고, 새로 부가된 정책을 적용할 수도 있다. 전송모듈은 에이전트들로부터 수집된 데이터를 수신받거나 새로운 보안 정책을 에이전트에게 전송하는 역할을 하며, DB 모듈은 수집된 데이터를 저장 관리하는 기능을 수행한다.

4. 결 론

실험결과 다수의 소프트웨어들이 단순한 프로텍션 모델을 가지고 있으며, 프로텍션 모듈 자체도 보호되지 않고 있음을 확인하였다. 즉, 완성도 높은 소프트웨어 개발과 더불어 보다 안전하게 지적재산권을 보호하기 위한 노력이 절실하게 요구되고 있다.

이러한 문제점을 해결하기 위하여 본 논문에서는 소프트웨어 프로텍션과 크래킹 기술에 분석하였으며 대응책으로 윈도우 환경에서의 메모리 해킹 방지 시스템을 제안하였다. 제안된 시스템은 관리 서버에서 설정한 보안 정책에 따라 에이전트가 운영되며 에이전트에서는 보안 정책을 기반으로 프로그램 실행 모듈에 대한 해쉬값을 비교하고, Win32 API 사용 패턴과 파라미터를 분석하여 프로세스들에 대한 동작을 감시하고 탐지한다. 시스템은 모든 윈도우 플랫폼에 적용가능하며, 특히 온라인상에서 서비스 중인 게임 소프트웨어나 일정 금액을 납부하고 사용하는 웹 하드 클라이언트 프로그램에 대한 메모리 조작으로 인한 불법적인 사이버머니 및 마일리지 취득에 대해 현실적으로 대응할 수 있을 것으로 예상된다.

향후 방향으로는 구체적인 시스템 운영 및 보안 정책을 수립하고 보안 정책을 적용한 기본 시스템 구현과 실험이 필요하다.

참 고 문 헌

[1] Woodmann, http://www.woodmann.com/forum/index.php.
[2] BiW Reversing, http://www.reversing.be/
[3] Aladdin, http://www.aladdin.com/.
[4] Brain Studio, http://www.brstudio.com/index.html.
[5] Crackmes, http://www.crackmes.de/.
[6] Compuware, http://www.compuware. com.
[7] OllyDbg, http://www.ollydbg.de/.
[8] DataRescue, http://www.datarescue.com/.
[9] Sysinternals, http://www.sysinternals.com.
[10] Le4rN TO Cr4cK, http://learn2crack.com.
[11] pudn.com, "HookAPI source code", http://www.codeproject.com/system/Paladin.asp.
[12] William Zhu, Clark Thomborson, and Fei-Yue Wang, "A Survey of Software Watermarking", LNCS 3495, 2005.
[13] P. C. van Oorschot, "Revisiting Software Protection", ISC 2003.
[14] Min Chen, "Software Product Protection", 2001.
[15] A. Main and P. C. van Oorschot, "Software Protection and Application Security : Understanding the Battleground", International Course on State of the Art and Evolution of Computer Security and Industrial Cryptography, 2003.
[16] Shub Nigurrath, "Writing Loaders for DLLs : Theory and Techniques", CodeBreakers Journal 2005.
[17] Crashtest, tutorial #1, 2nd version, http://207.218.156.34/krcbar/beginner/97.txt.
[18] Stone, "In memory patching : three Approaches(how to introduce breakpoints in an automated debugger and other marvels)", 1997.
[19] UPX(The Ultimate Packer for eXecutables), http://www.upx.org/.
[20] cRACKER's nOTES, "Commercial Protection Systems : SalesAgent", http://www.learn-2crack.com/p09.html.
[21] McCodEMaN, "A Manual Unpacking Essay : unUPX v1.06", 2000.
[22] Predator, "Unpacking : a generic approach, including IT rebuilding", 2001.

김 요 식

2004년~현재 국가보안기술연구소 연구원

윤 영 태

1995년 충남대학교 컴퓨터과학과(학사)
1997년 현대전자 정보시스템 사업본부
1999년 충남대학교 컴퓨터과학과(이학석사)
1999년~현재 국가보안기술연구소 선임연구원

박 상 서

1991년 중앙대학교 전자계산학과(공학사)
1993년 중앙대학교대학원 전자 계산학과(공학석사)
1996년 중앙대학교대학원 컴퓨터 공학과(공학박사)
1996년~1998년 국방정보체계연구소 선임연구원
1998년~1999년 국방과학연구소 선임연구원
2000년~현재 국가보안기술연구소 선임연구원