인 메모리 악성코드 인젝션 기술의 언 패킹기법

(Unpacking Technique for In-memory malware injection technique)

배성일*, 임을규**

(Seong Il Bae, Eul Gyu Im)

요 약

2018년 평창 동계 올림픽 개막식에서 출처를 알 수 없는 사이버공격이 발생하였다. 해당 공격에서 사용된 악성코드는 인 메모리 악성코드로 기존 악성코드와 은닉하는 장소가 다르며, 140개 이상의 은행, 통신, 정부 기관에서 발견될 정도로 빠르게 확산되고 있다. 인 메모리 악성코드는 전체 악성코드의 15%이상을 차지하며 매우 심각한 피해를 주고 있다. 비휘발성 저장장치로 알려진 하드디스크에 자신의 정보를 저장하는 것이 아닌 휘발성 저장장치 인 램의 특정 메모리영역인 프로세스에 삽입하여 악성행위를 일으키는 악성코드를 인 메모리 악성코드라고 지칭한다. 결과적으로 자신의 정보를 남기지 않아 메모리 탐지 도구를 우회하여 악성코드 분석가들의 분석을 어렵게 한다. 또한 현대 메모리는 갈수록 크기가 증가해 메모리 탐지 도구를 사용하여 메모리전체를 보기 힘들다. 따라서 본 논문에서는 인 메모리 악성코드인 Dorkbot과 Erger를 대상으로 IDA Pro 디버거를 통해 인젝션을 언 패킹하여 효율적으로 페이로드를 산출하는 방법을 제안한다.

■ 중심어 : 인젝션; 인 메모리 악성코드; 언 패킹; 아이다 프로

Abstract

At the opening ceremony of 2018 Winter Olympics in PyeongChang, an unknown cyber-attack occurred. The malicious code used in the attack is based on in-memory malware, which differs from other malicious code in its concealed location and is spreading rapidly to be found in more than 140 banks, telecommunications and government agencies. In-memory malware accounts for more than 15% of all malicious codes, and it does not store its own information in a non-volatile storage device such as a disk but resides in a RAM, a volatile storage device and penetrates into well-known processes (explorer.exe, iexplore.exe, javaw.exe). Such characteristics make it difficult to analyze it. The most recently released in-memory malicious code bypasses the endpoint protection and detection tools and hides from the user recognition. In this paper, we propose a method to efficiently extract the payload by unpacking injection through IDA Pro debugger for Dorkbot and Erger, which are in-memory malicious codes.

■ keywords : Injection ; In-memory Malware ; Unpacking ; IDA Pro.

Ⅰ. 서 론

IT기술의 급속한 발전으로 때와 장소에 관계없이 인터넷에 접속하여 원하는 일을 처리할 수 있을 뿐만 아니라, 날이 갈수록 빨라지는 유무선 네트워크 기술로 인해 각종 문서들을 쉽고 빠르게 다운로드할 수 있다. 그러나 장점이 있으면 단점도 있듯이, IT기술의 발전과 더불어 각종 컴퓨터 바이러스, 웜, 스파이웨어 등 악성코드들 또한 빠르게 진화하고 있다. 또한 악성행위를 수행하는 기술은 점점 더 복잡해지고 교활해졌고, 고도의 탐지 도구를 회피해 생존력을 높여가고 있다. AhnLab Security Emergency response Center(ASEC) 리포트[1]에 따르면 2018년 5가지보안 위협 전망 중 하나인 '문서 파일을 이용한 공격의 고도화와 인 메모리 공격'을 꼽으며 시스템에 존재하는 형태보다는 프로세스 메모리에 인젝션 되어 동작하는 방식이 증가할 것으로 예측하였다.

인 메모리 악성코드는 일반 악성코드와는 다르게 유해한 데이터가 하드디스크에 파일 상태로 존재하지 않고 메모리에 존재한다. 인 메모리 악성코드의 감염 경로로는 사용자를 유도해 악성 데이터에 감염시키는 사회 공학적 기법과 익스플로잇 도

* 준회원, 한양대학교 컴퓨터 소프트웨어 학과
** 정회원, 한양대학교 컴퓨터 소프트웨어 학부
이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. NRF-2016R1A2B4015254).
접수일자 : 2018년 09월 19일 게재확정일 : 2019년 01월 18일
수정일자 : 2018년 12월 05일
교신저자 : 임을규 e-mail : imeg@hanyang.ac.kr

구를 이용해 악성코드를 사용자의 네트워크로 전파시켜 감염시키는 Drive-by-download 공격 기법이 있다. 인 메모리 악성코드에 감염되었을 경우 악성 코드가 실행되면서 정상적인 프로세스처럼 보이는 익스플로러에 악성 페이로드가 주입된다. 그 후 합법적인 프로세스를 통해 악의적인 행위를 수행한다. 사용자는 악의적인 행위를 수행하는 프로세스를 정상적인 프로세스로 보기 때문에 감염된 사실을 알아채기 어렵다. 게다가 인 메모리 악성코드는 레지스트리 키를 변조하는 방식을 통해 지속성을 가지므로, 재부팅 시 사라지지 않고 감염된 환경에 머무를 수 있다.

본 논문은 인 메모리 악성코드인 Dorkbot과 Erger에서 사용되는 인젝션을 언 패킹하여 페이로드를 찾는 방법을 제안한다. 본 연구에서 서술하는 언 패킹이란 이미 인젝션이 진행된 프로세스에 특정 부분을 역으로 풀어내는 기법을 의미하며 논문의 구성은 다음과 같다. 1장에서 서론을, 2장 1절에서 본 연구를 수행하면서 참고한 관련 자료를 소개하고 2장 2절에서 실제 많이 사용되는 세 가지 인젝션 기법을 소개한다. 3장 1절에서는 악성코드분석에 사용된 환경을 소개하며, 3장 2절과 3절에서 Dorkbot과 Erger 악성코드에서 사용되는 인젝션 언 패킹 기법을 설명하여 악성행위의 주체인 페이로드를 산출하는 방법을 제안한다. 마지막으로 4장에서 본 논문의 향후 연구계획을 언급하며 끝으로 결론을 맺는다.

Ⅱ. 관련 연구

1. 인 메모리 악성코드 관련 연구

Jesse Smelcer는 인 메모리 악성코드의 역사를 통해 인 메모리 악성코드의 탄생과 진화과정을 설명하며, 세부적으로 인 메모리 악성코드가 메모리보호 탐지도구를 회피하는 기술을 설명하였다. 또한 악성코드와 파일리스 악성코드의 차이점을 설명하며 인 메모리 악성코드만 가지는 특징을 설명하였다. 끝으로 새로운 악성코드 탐지도구와 방어기술을 통해 증가하고 있는 파일리스 악성코드를 막아야한다고 주장하였다[2].

David Patten은 .NET프레임워크의 개발이 IT산업 발전에 긍정적인 영향을 주었지만, PowerShell 소프트웨어의 취약점으로 악성코드의 발전에도 긍정적인 영향을 주었다고 주장하였다. 또한 PowerShell 스크립트의 영향으로 인 메모리 악성코드는 화이트리스트 형태로 높은 관리권한을 갖게 될 수 있다고 설명하며, 결과적으로 인 메모리 악성코드의 발전에 영향을 주었다고 설명하였다[3].

Liam O'Murchu와 Fred P.Gutierrez는 최초의 인 메모리 악성코드인 Poweliks를 IDA pro 디버거를 이용해 상세히 디버깅하였다. 분석 내용으로는 Poweliks가 사용하는 인젝션 기법에서 자기 자신을 숨기는 방법 및 naming technique에 대하여 상세하게 분석하였다. 특히 C&C서버와 악성코드의 통신 과정에서 쓰이는 데이터를 분석하여 Poweliks가 사용하는 프로토콜을 설명하였다. 또한 Poweliks 인 메모리 악성코드가 가지는 지속성에 대해 설명하고 악성행위를 기술하였다[4].

Yang-seo Choi 등은 악성코드에서 주로 사용되는 패킹기법 및 암호화기법을 설명하였다. 패킹 및 암호화기술을 사용한 악성코드의 경우 PE 헤더의 구조가 정상파일과 다르기 때문에 정적 분석은 매우 힘들다고 주장하며 동적 분석을 하더라도 제약사항이 많다고 하였다. 해당 연구에서는 CV(Characteristic Vector)에 여덟 가지 요소를 정의하고 ED(Euclidean Distance)거리 측정 방식을 사용하여 패킹 파일 탐지 시스템을 제안하였다[5].

Kris Kendall은 실제 많은 피해를 입힌 악성코드를 분석하고, 분석한 내용을 바탕으로 해당 악성코드를 설명하였다. 특히 동적, 정적 분석에 사용되는 도구를 설명하며 경우에 따라 어떤 도구가 효과적인지 언급하였다. 또한 역공학에 사용되는 디스어셈블리와 디버깅도구인 IDA pro의 사용법을 설명하였다[6].

Iisun You와 Kangbin Yim은 공격자가 사용하는 난독화 기법들에 대해 설명하면서 안티바이러스 제품을 우회하는 공격기법에 대해 설명하였다. 난독화 기법과 더불어 악성코드에 적용되는 암호화, 다형화, 변형화 기법에 사용되는 Dead-code insertion, register reassignment, subroutine reordering, instruction substitution, code transposition등을 설명하였다[7].

Michael과 Sikorski는 악성코드분석에 필요한 다양한 기법과 분석도구를 설명하였다. 특히 안티디버깅기법과 안티 VM 기법을 설명하며 해당 안티기술들을 회피하는 방법을 설명하였다. 또한 여러 가지 인젝션기법을 설명하였으며, Shellcode 분석방법에 대해 설명하였다[8].

Emeric과 Nasi는 PE 인젝션 기법에 대하여 설명하고 인젝션에 사용되는 여러 가지 API를 소개하였다. 더 나아가 PE 인젝션을 c언어로 작성하여 함수와 사용되는 매개변수까지 자세하게 설명하였다[9].

Willems 등은 악성코드 분석 툴인 CWSandbox를 제작하였으며 해당 툴은 윈도우즈32 운영체제 기반으로 자동성, 효율성, 정확성을 만족하였다고 주장하였다. 해당 툴을 제작하면서 악성코드의 행동기반 분석을 진행하였다. 악성코드가 파일을 수정하거나 생성하는 방법, 윈도우즈 레지스트리를 변경하는 방법, 어떤 dll이 로드되는지를 설명하고 마지막으로 정보를 송수신하기 위한 네트워크 연결과정을 설명하였다[10].

Seong Il Bae 등은 인 메모리 악성코드가 주로 사용하는 인젝션기법에 대하여 설명하였으며, 인 메모리 악성코드인 Dorkbot을 분석하였다. Dorkbot은 이중구조의 인젝션기법을 사용하며 인젝션에 사용되는 기법을 IDA Pro 도구를 이용해

분석하였다[11].

앞서 언급한 바와 같이 악성코드와 인 메모리 악성코드분석에 다양한 관련 연구가 진행되었지만, 인 메모리 악성코드가 사용하는 인젝션 기법을 분석하고 해당 인젝션을 언 패킹하여 페이로드를 추출하는 방법을 제시하는 연구는 미비한 실정이다.

2. 인 메모리 악성코드 세 가지 인젝션방법

프로세스 인젝션이란 말 그대로 사용하고자 하는 악의적인 코드를 대상이 되는 프로세스에 삽입하고, 감염된 프로세스가 의도하지 않은 악성행위를 수행하는 것이다. 본 기술은 인 메모리 악성코드의 특성에 유사하며, 파일 없이 악성행위를 수행하기 최적화된 공격이다. 해당 수법은 정상적인 프로세스인 것처럼 위장해 화이트 리스팅 탐지 및 안티바이러스 도구를 회피한다. 본 기법은 윈도우즈에서 제공하는 API의 함수를 사용하여 실행할 수 있으며, 몇 개의 함수를 이용하면 누구나 쉽게 인젝션을 수행할 수 있다. 또한 이미 알려진 인젝션기법에 조금씩 변형된 기법까지 사용되며, 수많은 기법이 존재하여 악성코드 분석가들의 분석을 어렵게 만든다. 본 절에서는 인 메모리 악성코드에서 주로 사용되는 DLL 인젝션 기법, Process Hollowing, PE 인젝션 기법을 설명한다. 설명에 앞서 [표 1]은 인젝션기법에 주로 사용되는 API에 해당되는 함수와 해당 함수의 설명을 나타낸다.

표 1, 인젝션에 사용되는 주요 API

API	설명
VirtualAllocEx	가상 메모리 공간 할당
CreateProcess	특정 프로세스 생성
OpenProcess	로컬 프로세스에 접근
WriteProcessMemory/WriteVirtualMemory	특정 프로세스 메모리영역에 데이터 쓰기/ 가상메모리 영역에 데이터 쓰기
LoadLibrary	프로세스에서 dll을 로드
CreateRemoteThread	특정 프로세스의 가상 주소 공간에 원격 스레드 생성
ResumeThread	지정된 핸들을 가지는 스레드 실행
MapviewofSection/ UnMapviewofSection	가상 주소공간에 할당된 프로세스의 특정 세션에 메모리 매핑/언 매핑

인젝션 기법에는 크게 두 가지로 분류될 수 있으며 주입할 대상이 무엇이냐에 따라 DLL 인젝션과 Code인젝션으로 분류할 수 있다.

DLL 인젝션이란 프로세스가 필요한 DLL을 로드할 때 악의적인 DLL을 로드 하도록 하는 것이다. 해당 기법의 핵심 함수는 CreateRemoteThread()이며, OpenProcess()로 DLL이 삽입될 프로세스를 얻는다. 이어서 GetProcAddress()를 이용해 DLL을 로드하는 LoadLibrary()의 주소를 구한 뒤 마지막으로 할당된 위치에 WriteProcessMemory()를 이용해 DLL을 삽입하게 된다. 설명한 모든 함수들은 CreateRemoteThread()의 인자로 지정된다.

코드 인젝션이란 첫 번째로 OpenPorcess()를 이용해 주입할 대상 프로세스의 핸들을 얻는다. VirtualAllocEx()를 이용하여 프로세스 메모리공간에 삽입될 메모리영역을 확보한다. WriteProcessMemory()를 이용해 할당된 메모리에 데이터를 기록한다. CreateRemoteThread()를 이용해 프로세스에 스레드를 생성하고 마지막으로 ResumeThread()를 사용해 해당 스레드를 실행시킨다. DLL 인젝션과는 다르게 해당 인젝션 기법은 독립 실행코드를 삽입하기 때문에 코드영역과 데이터영역 분리하여 삽입한다. 따라서 삽입된 코드는 정확한 데이터영역을 참조해야한다. DLL 인젝션보다 구현이 어렵지만 DLL 인젝션의 탐지도구를 우회할 수 있어 공격자들은 코드 인젝션을 많이 사용한다.

가. DLL 인젝션 CreateRemoteThread()

해당 기법은 여러 가지 DLL 인젝션 기법 중 윈도우즈 제공함수인 CreateRemoteThread()를 이용한다는 측면에서 유연성이 좋은 인젝션기법이다. 공격자에 의해 생성된 스레드가 인젝션의 대상이 되는 DLL에 대하여 LoadLibrary함수를 실행한다. LoadLibrary함수는 로드되는 DLL의 DllMain함수를 실행하기 때문에 정상적인 프로세스처럼 보여 사용자들은 쉽게 알아차리지 못한다. [그림 1]은 CreateRemoteThread()의 인자와 인자가 실행하는 함수의 개요를 나타낸다.

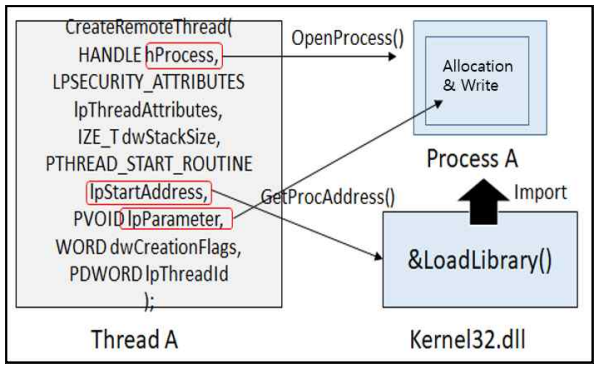

그림 1. DLL 인젝션 실행 개요

CreateRemoteThread함수의 첫 번째 인자는 OpenProcess함수를 이용하여 얻은 프로세스의 핸들을 나타내며, 4번째 매개변수인 lpStartAddress는 LoadLibrary()를 가리킨다. 정상적인 경우 스레드를 생성할 프로세스에서 LoadLibrary()를 가리

키지만, 윈도우즈는 각 프로세스의 Kernel32.dll의 이미지베이스 주소를 일정하게 관리하므로 Kerne32.dll의 함수를 가리켜도 문제가 되지 않는다. 본 과정까지 정상적으로 수행이 이루어지면 5번째 인자인 lpParameter를 이용해 프로세스 내의 스레드 위치를 할당하고 사용될 DLL을 쓴다. 메모리 영역 할당에는 VirtualAllocEx()가 사용되며, 이후 WriteProcessMemory()를 이용해 쓰고자하는 데이터를 메모리에 삽입한다. 마지막으로 생성된 스레드를 ResumeThrad()를 이용해 실행시키는 것으로 인젝션이 성공적으로 수행된다.

나. Process Hollowing

Process Hollowing은 프로세스 인젝션과 같이 프로세스에 프로세스 자체를 인젝션 하는 기법이다. 최근 악성코드에서 사용되는 흔한 기법으로 정지 상태로 만들어 놓은 정상적인 프로세스를 로드하여, 본래코드가 할당된 메모리영역을 악성코드가 속해있는 메모리영역으로 바꾸는 기술이다. 해당 기술은 세부적인 기술들도 매우 다양한 편이다. 기본적으로 정상 프로그램을 CreateProcess함수를 이용하여 실행시키는데 인자로 CREATE_SUSPENDED를 넣는다. 이후 메모리에 매핑되어 있는 각 섹션에 대하여 Unmapviewofsection함수를 이용하여 정상코드를 언 매핑하고 악성코드를 WriteProcessMemory함수로 다시 덮어쓴다. 해당 기술을 사용한다면 실제로는 악의적인 프로그램을 일반 사용자가 보기에 정상 프로그램인 것처럼 위장할 수 있다. 또한 본래 실행이 정상적으로 이뤄졌기 때문에 기존권한과 같은 권한을 갖고 있다. 따라서 메모리의 권한을 탐지하는 도구로는 탐지가 불가능하며, 디스크를 사용하지 않기 때문에 인 메모리 악성코드로 분류한다.

본 기술의 핵심은 CreateProcess함수를 이용해 정지 상태의 프로세스로 만들고 정확한 Image Base 주소를 알아내는 것이다. 그렇지 않으면 해당 프로세스 주소의 원본 바이너리를 메모리에서 언 매핑할 수 없다. 공격자는 Image Base 주소를 알아내기 위해 QueryProcessInformation함수를 이용하거나 GetThreadContext함수를 이용한다.

QueryProcessInformation함수를 호출하면 PEB(Process Environment Block)를 알 수 있다. PEB란 유저레벨에서 프로세스의 정보를 담는 구조체이다. 구조체에 선언되어 있는 변수 PVOID ImageBaseAddress를 이용하여 공격자는 Image Base 주소를 획득한다. 다른 또 한 가지 방법으로 GetThreadContext함수를 이용하여 PEB를 구할 수 있다. SUSPENDED상태로 생성된 프로세스의 경우 Context의 EBX에는 PEB의 주소가 들어가 있다. PEB주소에 0x08을 더하면 Image Base 주소를 나타내게 되며, 다른 프로세스의 메모리를 읽을 수 있는 ReadProcessMemory함수를 이용하여 프로세스의 Image Base 주소를 구할 수 있다. [그림 2]는 Process Hollowing의 실행개요를 나타낸다.

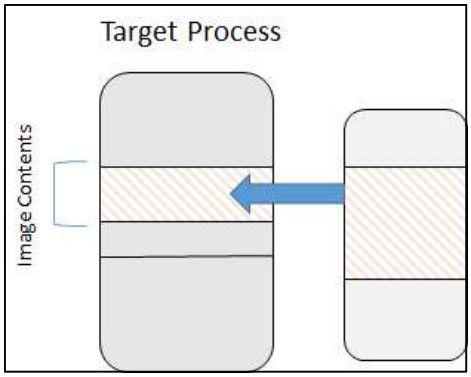

그림 2. Process Hollowing 실행개요

다. PE(Portable Executable) 인젝션

IAT(Import Address Table)과 BRT(Base Relocation Table)구조체가 PE 파일이 실행되면서 특정 메모리 영역에 생성된다. 프로세스 실행에 필요한 정보, 메모리, DLL의 주소는 PE 로더에 의해 IAT 테이블이 생성된다. BRT는 프로세스의 메모리공간에 PE 헤더의 정보중 하나인 Image Base 주소에 로드된다. 프로세스는 자신만의 고유한 메모리주소를 가지므로 이미 다른 프로세스가 해당 영역을 사용하고 있다면 PE 재배치 과정이 필요하며, 재배치과정에서 필요한 정보가 BRT의 정보이다. PE 인젝션의 핵심은 IAT정보와 BRT정보를 바꾸는 것이다. PE 인젝션은 OpenProcess함수와 VirtualAllocEx함수는 각각 삽입될 프로세스의 핸들을 구할 때와 삽입될 대상이 되는 프로세스에 메모리공간을 할당할 때 사용된다. 이때, 정확한 위치에 메모리를 할당하기 위해 BRT의 정보를 참조하여 절대주소를 알아낸 뒤 WriteProcessMemory함수를 사용하여 입력될 악성코드를 메모리에 위치시킨다. 이후 CreateRemoteThread함수를 호출하여 다른 프로세스에 스레드를 생성하여 RemoteThread를 통해 스레드를 실행시켜 악성행위를 수행하게 된다. [그림 3]은 PE 인젝션의 전체적인 과정을 나타낸다.

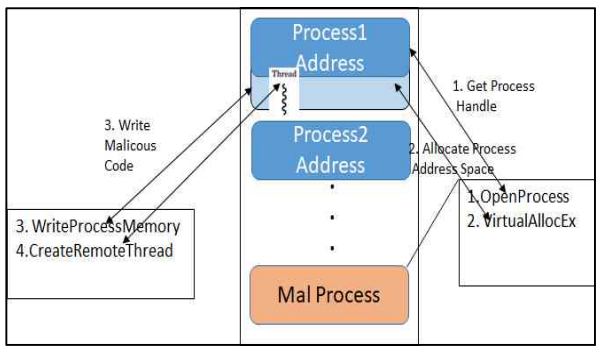

그림 3. PE 인젝션의 전체적인 흐름도

PE 인젝션 기법을 요약하여 설명하자면 임의의 프로세스의 메모리를 읽고 쓰거나 다른 영역에 있는 프로세스에 스레드를 생성하고 실행시키는 행위는 매우 어려운 일이지만, 윈도우즈 시스템API를 이용하면 누구나 쉽게 해당 기법을 수행할 수 있다는 장점을 이용한 인젝션 공격방법이다.

III. 인 메모리 악성코드 분석

1. 분석 환경과 도구

정적 분석방법과 동적 분석방법은 악성코드분석에서 가장 많이 쓰이는 방법론이며, 해당 연구에서 Dorkbot과 Erger악성코드를 분석하기 위해 두 가지 모두 사용하였다. 분석환경에는 가상환경을 사용하였으며 VirtualBot 5.2.8 버전의 Windows 7 64bit 이미지를 사용하였다. 또한 내부 네트워크를 이용하여 분석환경과 호스트환경을 독립적으로 분리시켜 안전하게 분석하였다. 디버깅도구로 IDA Pro를 사용했으며, IDA Pro란 디스어셈블러로 바이너리 파일을 어셈블리어로 재구성해주는 툴이다. 본 장에서는 해당 툴로 인 메모리악성코드인 Dorkbot과 Erger이 사용하는 인젝션기법을 언 패킹하여 페이로드를 산출하는 방법을 중점적으로 설명한다. 두 악성코드 모두 두 번의 인젝션을 사용하여 분석을 어렵게 함과 동시에 분석시간을 지연시킨다.

2. 인젝션 언 패킹, Dorkbot

Dorkbot은 PE 인젝션 기법과 Process Hollowing의 변형을 사용하므로써 총 두 번의 인젝션을 진행한다. [그림 4]는 Dorkbot을 실행한 프로세스 익스플로러의 정보를 나타내며 Dorkbot.exe가 iexplore.exe로 변형된 것을 볼 수 있다.

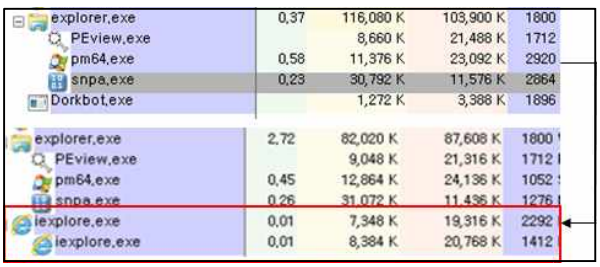

그림 4. DorkBot 실행 정보

익스플로러의 정보로 인젝션의 대상이 되는 프로세스는 iexplore로 유추할 수 있으며, Dorkbot의 본래 실행파일은 삭제된 후 레지스트리를 변경하는 것을 볼 수 있다.

IDA Pro로 Dorkbot의 바이너리파일을 연 뒤 인젝션에 많이 사용되는 API에 BP를 입력한 뒤 분석을 진행하면 WriteProcessMemory()가 호출되는 것을 확인할 수 있다. [그림 5]는 WirteProcessMemory()의 MSDN정보이며 첫 번째 인자인 hProcess는 삽입될 대상이 되는 프로세스의 핸들, 두 번째 인자는 쓰이는 데이터의 주소, 세 번째 인자는 실제 쓰이는 데이터의 위치, 네 번째 인자는 데이터의 크기를 나타낸다.

```
BOOL WINAPI WriteProcessMemory(
  _In_  HANDLE  hProcess,
  _In_  LPVOID  lpBaseAddress,
  _In_  LPCVOID lpBuffer,
  _In_  SIZE_T  nSize,
  _Out_ SIZE_T  *lpNumberOfBytesWritten
);
```

그림 5. MSDN의 함수 정보

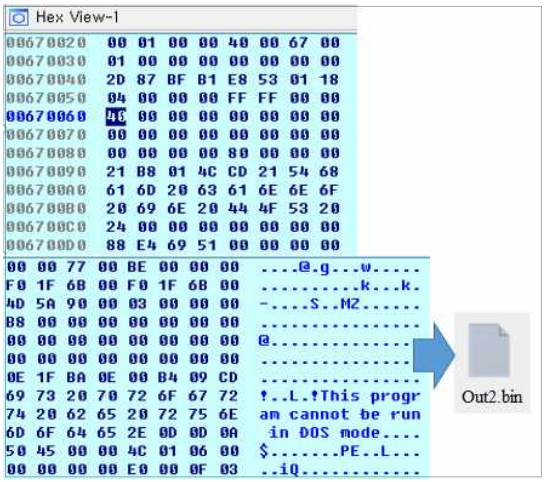

그림 6. Dorkbot의 WriteProcessMemory함수 정보

[그림 6]은 디버깅과정에서 호출되는 WriteProcessMemory 함수의 정보로, 0x670060주소에 PE 파일 형식의 데이터가 사용되는 것을 볼 수 있다. 쓰일 데이터의 위치는 0x00400000로 크기 400바이트이다. 추후 분석을 위해 해당 메모리영역을 덤프하여 Out2.bin파일로 저장하였다.

Out2.bin을 다시 IDA Pro로 실행시켜 인젝션에 많이 사용되는 API위주로 분석하다보면 NTMapViewofSection함수와 NTUnMapViewofSection함수가 호출되는 것을 확인할 수 있다. NTMapViewofSection함수는 10개의 인자를 가지며 본 연구에서 중요한 인자는 세 번째 인자로 해당 인자는 메모리 섹션에 스택포인터 정보를 갖고 있다. 따라서 스택의 값이 가리키는 값을 Hex View를 통해 관찰하면 한 번 더 PE 파일이 쓰이는 것을 확인할 수 있다. 2장 2절에서 언급한 인젝션기법의 변형으로 보이며 MapViewofSection함수와 UnMapViewofSection 함수를 사용하여 분석을 어렵게 만든다. [그림 7]은 Dorkbot이 호출하는 NTMapViewofSection의 정보를 나타낸다. [그림 7]에서 1번에 해당되는 그림은 NTMapViewofSection함수가 호출되는 것을 확인한 것이다. [그림 7]의 2번은 해당 함수의 첫

번째부터 세 번째 인자까지의 값을 나타내며 각각 프로세스 핸들, 프로세스가 외부에 있음을 나타내는 주소, 스택포인터를 나타낸다. [그림 7]의 3번은 NTMapViewofSection함수의 세 번째 인자인 00x18F360값이 가리키는 값은 2360000인 것을 나타내며 마지막으로 4번은 2360000의 데이터를 확인한 것이다. 결론적으로 다시 PE 파일 형식의 데이터가 사용되는 것을 확인할 수 있다.

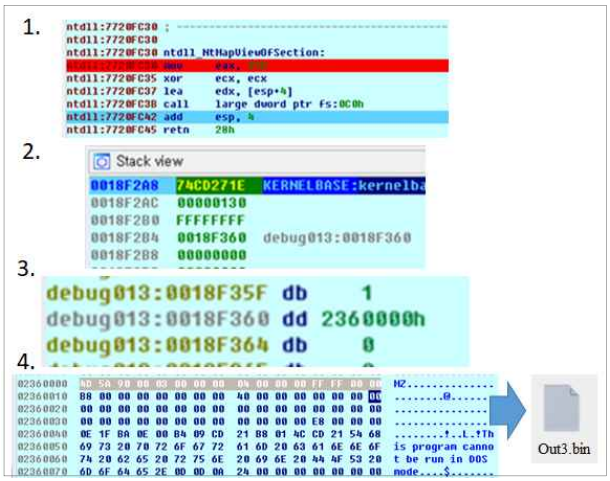

그림 7. Dorkbot의 NtMapViewofSection함수 정보

해당영역을 덤프하여 Out3.bin으로 저장하여 IDA Pro를 이용하여 실행을 시키면 암호화가 수행된 쉘 코드 영역과 복호화 알고리즘 영역으로 판단되는 부분을 확인할 수 있다. 복호화에는 Dorkbot이 사용하는 정확한 키값이 요구되며 실행 시 본 쉘 코드가 복호화 되면서 악성행위를 수행할 것으로 판단된다.

3. 인젝션 언 패킹, Erger

프로세스 익스플로러를 이용해 Erger악성코드의 실행을 모니터링하면 explorer.exe가 실행되는 것을 볼 수 있다. 해당 정보로 Erger악성코드는 인젝션을 수행한다는 것을 예상할 수 있으며 [그림 8]은 프로세스 익스플로러로 확인한 Erger악성코드 정보이다.

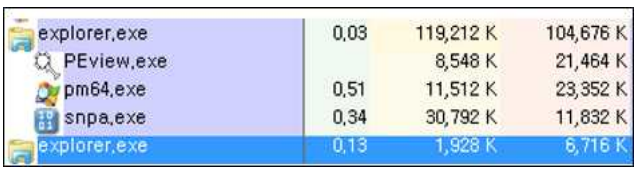

그림 8. 프로세스 익스플로러의 Erger악성코드

Erger악성코드는 Dorkbot과 마찬가지로 두 번의 인젝션을 수행하며 2장의 2절과 3절에서 언급한 인젝션기법의 변형으로 판단된다. 본 과제에서 수행한 Erger악성코드의 인젝션 언 패킹은 다음과 같은 과정이 필요하며, 첫 번째 단계로 CreateProcess함수의 핸들을 얻는 과정 두 번째로 처음 수행되는 인젝션 언 패킹, 두 번째로 실행되는 인젝션 언 패킹, 세 번째 단계로 페이로드를 추출하기 위한 파이선을 이용한 스크립트 작성이 필요하다.

첫 번째 단계에 해당되는 CreateProcess함수의 핸들 얻기는 프로세스의 읽기/쓰기에 필요한 권한 상승문제와 직결되어 있다. 따라서 Createfile함수의 세 번째 인자인 ShareMode의 적절한 변조작업이 필요하다. Erger악성코드의 경우 해당 값은 0으로 초기화되어 있으므로 0을 7로 대체함으로써 데이터를 읽거나 쓸 때 문제가 발생되지 않도록 한다.

처음 수행하는 인젝션 언 패킹에 대해서는 Dorkbot과 동일하게 WriteProcessMemory함수와 ResumeThread함수를 이용해 이루어진다. WriteProcessMemory함수의 세 번째 인자의 값을 stage01.bin으로 저장하여 추후 분석에 사용하였다.

두 번째 인젝션 언 패킹 방법은 대게 사용되는 인젝션 기법보다 다소 까다롭게 진행된다. 핵심이 되는 API는 NTmapviewofSection과 NTunmapviewofSection이다. 총 6번의 해당 함수들이 호출되며 매핑과 언 매핑을 반복하므로 어떤 섹션에 어떤 주소에서 매핑이 이루어지는지 정확하게 분석해야 한다. Erger악성코드의 매핑과 언 매핑에 해당되는 주소는 [표 2]와 같으며 위에서 아래로 호출순서를 나타낸다.

표 2. Erger악성코드 두 번째 인젝션 정보

Section & Address / API	Section_Handle	Remote_address	Local_address
Mapviewofsection	80	x	1E80000
Mapviewofsection	88	x	1BF0000
Unmapviewofsection	80	x	1E80000
Unmapviewofsection	88	x	1BF0000
Mapviewofsection	88	550000	1E80000
Mapviewofsection	7C	90000	3A0000

가장먼저 호출되는 섹션 80을 가지는 MapviewofSection함수는 1E80000에 해당되는 주소의 값을 매핑 시키고 이어서 88섹션을 가지는 함수는 1BF0000에 해당되는 주소의 값을 매핑 시킨다. 하지만 이어서 두 번의 UnMapviewofSection함수가 실행되는데, 각각 80과 88에 해당되는 섹션을 언 매핑 시킨다. 결론적으로 마지막 두 번의 매핑함수에 해당되는 정보인 88섹션의 550000주소와 7c섹션의 90000주소가 인젝션 되는 것을 확인할 수 있다. 550000주소의 데이터는 PE 파일 형식의 데이

터가 사용되며 90000주소는 현재 분석단계에서는 알 수 없는 정보가 매핑된다. 이후 분석을 위해 PE 파일 형식의 데이터를 stage02.bin으로 90000주소에 해당되는 데이터를 Dump.bin으로 저장하였다. stage02.bin을 IDA Pro디버깅도구를 이용해 실행시키면 [그림 9]와 같은 세도우 코드를 확인할 수 있다.

```
v1 = 0xF673FFFF;
v2 = 0xC000;
v3 = 0;
v4 = (_BYTE *)0x905B2;
do
{
  *v4 ^= v1;
  v5 = __ROL4__(v1, 8);
  ++v4;
  v1 = v5 + 1;
  ++v3;
  --v2;
}
while ( v2 );
v6 = (const WCHAR *)MEMORY[0x92C1C](0, v1);
memset((void *)(a1 - 64), 0, 0x240000u);
*(_DWORD *)(a1 - 68) = 68;
GetStartupInfoW((LPSTARTUPINFOW)(a1 - 68));
if ( *(_BYTE *)(a1 - 24) & 1 )
  v7 = *(unsigned __int16 *)(a1 - 20);
else
  v7 = 10;
v8 = v7;
v9 = GetModuleHandleW(0);
v10 = sub_101A5ED(v9, 0, v6, v8);
```

그림 9. stage02.bin의 세도우 코드

해당그림은 복호화를 수행하는 코드로 판단되며 XOR, Rotation, Increment이 수행되는 것을 알 수 있다. 특히 포인터 변수인 v4가 가리키는 0x905B2영역은 Dump.bin의 내부 영역으로 7c섹션의 값을 읽어와 복호화를 수행하는 것을 확인하였다. v1은 복호화에 필요한 키값의 주소이며, 이는 [표 2]의 핸들 7C를 갖는 영역으로 매핑 되었음을 유추할 수 있다. [그림 10]은 Erger악성코드에서 수행되는 인젝션 기법의 전체적인 개요를 나타낸다.

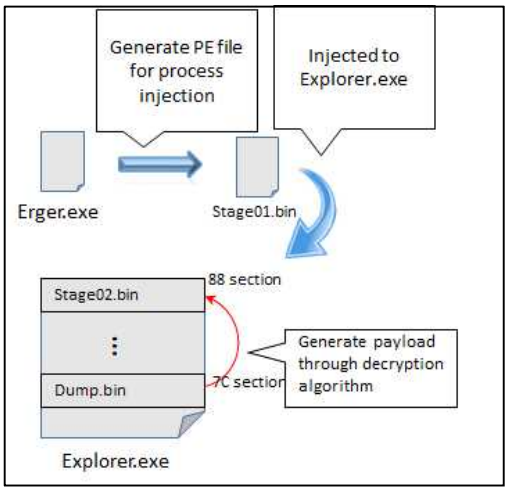

그림 10. Erger악성코드 실행 개요

[그림 11]은 [그림 9]의 세도우 코드에 해당되는 복호화알고리즘을 파이선 스크립트를 이용해 작성한 것이다. dump_905b2.bin영역의 데이터와 키를 unpacking_함수의 인자로 넣어 XOR, Rotation, Increment를 진행하여 얻은 평문을 stage3.exe로 저장하였다.

```
def main():
    with open("dump_905b2.bin","rb") as fp:
        data = fp.read()
        #print data
        pe_file = unpacking_(data,Key=0xf673ffff)
    with open("stage3.exe","wb") as f:
        f.write(pe_file)


main()
```

그림 11. 파이선을 이용한 복호화코드 중 일부

IDA Pro 디버깅도구를 이용하여 stage3파일을 확인하면 해당파일은 PE 파일 형식을 가지는 것을 알 수 있으며, 최종적으로 Erger악성코드가 수행하는 인젝션을 언 패킹한 결과 페이로드는 DLL인 것을 확인할 수 있다.

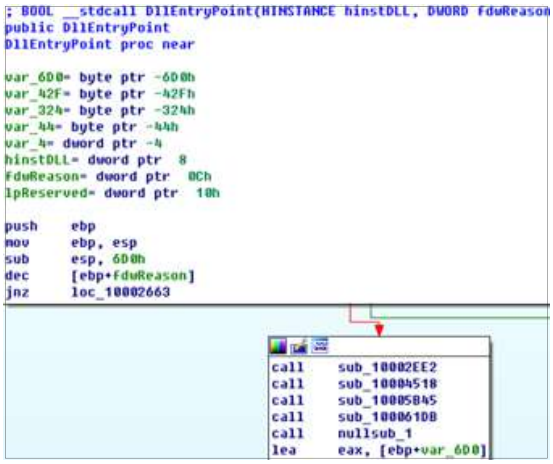

그림 12. 페이로드 dll 파일 추출

III. 결 론

본 논문에서는 인 메모리 악성코드인 Dorkbot과 Erger봇이 사용하는 인젝션 기법에 대해 설명하고, 디버깅도구를 이용해 상세히 분석하여 악성행위에 사용되는 페이로드를 찾는 방법을 제시하였다. 해당 방법은 실행과정을 추적하기 때문에 기존 백신 소프트웨어 및 탐지도구를 우회하는 악성코드에도 적용될 수 있다. 또한 제안하는 방법은 인젝션에 사용되는 특정 메모리 영역의 주소를 알 수 있기 때문에 현존하는 메모리 포렌식도구의 단점중 하나인 메모리의 모든 부분을 덤프하기 어렵다는 점을 극복할 수 있다. 향후연구로 Dorkbot과 Erger악성코드 이외에 더 많은 악성코드를 분석하여 인젝션에 주로 사용되는 API와 순서 및 행동정보를 활용하여 인 메모리 악성코드에서 악의적인 행위에 사용되는 메모리위치를 자동으로 탐지하는 프레임워크를 개발할 예정이다.

REFERENCES

[1] "AhnLab Security Emergency response Center Report, 2017 Q4 Cyber Threat Trend Report", (Jan, 2018), https://www.ahnlab.com/kr/site/securityinfo/asec/asecView.do?groupCode=VNI001&seq=27109, (Sep/18/2018).

[2] Jesse Smelcer. "The rise of Fileless malware". in *Partial Fulfillment of the Requirements for the Degree of Master of Science in Cybersecurity*, December 2017.

[3] David Patten. "The Evolution to Fileless Malware". East Carolina University, 2017.

[4] Liam O' Murchu and Fred P. Gutierrez. "The evolution of the fileless click-fraud malware Poweliks", *Symantec Security Response*, June 9, 2015.

[5] Yang-seo Choi, Ik-kyun Kim, Jin-tae Oh and Jae-cheol Ryou. PE File Header Analysis-Based Packed PE File Detection Technique (PHAD), *Computer Science and its* Applications, International Symposium on, pp. 28-31, Oct. 13, 2008.

[6] Kris, Kendall and McMillan, Chad.Practical "Malware Analysis". Black Hat Conference, USA ,p. 10, 2007.

[7] You, Ilsun and Yim, Kangbin. "Malware obfuscation techniques: A brief survey". Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on, IEEE, pp. 297-300, 2010.

[8] Sikorski, Michael and Honig, Andrew. "Practical malware analysis: the hands-on guide to dissecting malicious software", *Published by nostarch press*, 2012.

[9] Nasi and Emeric. "PE Injection Explained Advanced memory code injection technique". *Creative Commons Attribution-NonCommercial-NoDerivs*, volume 3, 2014.

[10] Willems, Carsten and Holz. "Toward automated dynamic malware analysis using cwsandbox. IEEE Security & Privacy", IEEE, volume 5, Issue 2, pages 32-39, 2007.

[11] Seong Il Bae and Eul Gyu Im. "Unpacking Technique for Process Injection Malware", 2018 Workshop on Dependable and Secure Computation, Aug, 16, 2018.

저 자 소 개

배성일(학생회원)

2018년 한양대학교 컴퓨터공학부 학사 졸업.
2018년 ~ 현재 한양대학교 컴퓨터소프트웨어학과 석사.
<주관심분야 : 역 공학, 악성코드 분석, 네트워크 보안, 컴퓨터 보안>

임을규(정회원)

1992년 서울대학교 컴퓨터공학과 학사 졸업.
1994년 서울대학교 컴퓨터공학과 석사 졸업.
2002년 University of Southern California 컴퓨터공학과 박사 졸업.
2005년 ~ 현재 한양대학교 컴퓨터소프트웨어학부 정교수
<주관심분야 : 악성코드 분석, 유무선 네트워크 보안, 컴퓨터 보안, 소프트웨어 보안, 취약점 탐지>