Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.25, NO.6, Dec. 2015 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2015.25.6.1385

윈도우즈 라이브러리로 위장한 Proxy DLL 악성코드 탐지기법에 대한 연구 : Winnti 사례를 중심으로

구 준 석,† 김 휘 강‡
고려대학교 정보보호대학원

A research on detection techniques of Proxy DLL malware disguised as a Windows library : Focus on the case of Winnti

JunSeok Koo,† Huy Kang Kim‡
School of Information Security, Korea University

요 약

Proxy DLL은 윈도우즈에서 DLL을 사용하는 정상적인 메커니즘이다. 악성코드들은 목표 시스템에 심어진 뒤에 감염을 위해 최소 한번은 실행되어야 하는데, 이를 위해 특정 악성코드들은 정상적인 윈도우즈 라이브러리로 위장하여 Proxy DLL 메커니즘을 이용한다. 이런 유형의 대표적인 공격 사례가 윈티(Winnti) 그룹이 제작한 악성코드들이다. 윈티 그룹은 카스퍼스키랩(Kaspersky Lab)에서 2011년 가을부터 연구하여 밝혀진 중국의 해킹 그룹으로, 온라인 비디오 게임 업계를 목표로 다년간에 걸쳐 음성적으로 활동하였고, 이 과정에서 다수의 악성코드들을 제작하여 온라인 게임사에 감염시켰다. 본 논문에서는 윈도우즈 라이브러리로 위장한 Proxy DLL의 기법을 윈티의 사례를 통해 알아보고, 이를 방어할 수 있는 방법을 연구하여 윈티의 악성코드를 대상으로 검증하였다.

ABSTRACT

The Proxy DLL is a mechanism using a normal characteristics of Windows. Specific malware is executed via this mechanism after intrusion into a system which is targeted. If a intrusion of malware is successful, malware should be executed at least once. For execution, malware is disguised as a Windows Library. The malware of Winnti group is a good case for this. Winnti is a group of Chinese hacking groups identified by research in the fall of 2011 at Kaspersky Lab. Winnti group activities was negatively over the years to target the online video game industry, in this process by making a number of malware infected the online gaming company. In this paper, we perform research on detection techniques of Proxy DLL malware which is disguised as a Windows library through Winnti group case. The experiments that are undertaken to target real malware of Winnti show reliability of detection techniques.

Keywords: Malware, Malicious Code, Anti Virus, Proxy DLL, Windows Library, Winnti, APT

I. 서 론

전통적인 APT(Advanced Persistent Threat)는 정부 혹은 군사시설이나 정치조직, 전력망, 화학공장, 사회 기반 시설 등이 주요 목표였었다. 이런 국가 주요 기반 시설을 목표로 하여 사이버 무기로 사용 되었던 대표적인 실례가, 외국에서는 2010년에 최초로 밝혀진 스턱스넷(Stuxnet)이었고, 국내에서는 2011년에 발생한 '농협 전산망 마비 사건' 이었다. 하지만 같은 해에 국내에서 발생한 '네이트 개인

Received(07. 08. 2015), Modified(09. 22. 2015), Accepted(09. 22. 2015)

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

정보 유출 사건'을 비추어 보면, 초창기 APT가 한 사회에 혼란을 초래하기 위한 목적에서, 이제는 상업적인 회사를 목표로 금전적인 이익을 얻기 위한 목적으로 점차 넓혀져 가고 있음을 알 수 있다. 이러한 흐름 속에서 윈티는 특정한 게임사들을 목표로 금전적인 이익을 얻는 것을 목적으로 한 해킹 그룹이었다. 이를 위해 윈티는 악성코드를 작성하였고 게임사들을 감염시켰다. 이 과정에서 불법적으로 얻은 정보를 이용하여 윈티는 다음과 같은 이득을 볼 것이라고 예상하고 있다[7].

첫째, 게임에서 통용되는 사이버머니를 불공정하게 축적하여 현실에서 통용되는 실제 돈으로 환전을 하여 금전적인 이득을 취한다.

둘째, 게임내의 취약점을 발견하기 위해 게임사의 서버에서 소스 코드를 불법적으로 훔쳐서 불공정한 사이버머니를 축적하는 것에 이용한다.

셋째, 게임사의 서버에서 훔친 소스 코드를 이용하여 해적판 게임 서버를 운용한다.

윈티의 APT공격으로 인해 국내 대형 온라인게임 회사들도 상당 수 피해를 입었었는데, 본 논문에서는 윈티가 제작한 악성코드의 최초 감염 방법을 알아보고 그와 유사한 공격에 대한 대응방안을 제시하고자 한다.

II. 관련 연구

악성코드 탐지 방법 중 하나인 시그니쳐 기반 탐지(signature-based detection) 기술은 대부분 AV(Anti-Virus) 제품에서 사용하고 있는 기술이다. 이 기술은 새로운 악성코드가 출현하면, 해당 악성코드를 수집 및 분석하여, 시그니쳐를 생성한 후, AV 제품의 데이터베이스에 시그니쳐를 추가하여 악성코드를 탐지하는 방식이다.

이 기술에서는 악성코드가 출현할 때마다 새로운 시그니쳐를 추가하여야 하는데, 이러한 특징 때문에 악성코드의 시그니쳐의 개수가 기하급수적으로 증가하는 문제점이 발생한다. 덧붙여 악성코드 제작자들은 이 기술을 우회하기 위해 악성코드의 변종을 계속적으로 개발함으로, 유사한 주요 기능을 가진 변종 악성코드들이 다수 출현하게 되었다[1,2].

이 단점을 보완코자, 악성코드의 행위를 기반으로 악성코드를 진단하는 기술이 많이 연구되었다. 하지만 이는 코드 가상화 같은 기술로 우회되고 있다[5].

III. Proxy DLL에 대한 이해

일반적인 경우 응용프로그램(.exe)은 필요한 라이브러리(.dll)를 직접 로드하여 사용한다.

Fig. 1은 이런 응용프로그램과 라이브러리 간의 관계를 도식화한 것이다.

하지만 Proxy DLL이 사용되면 응용프로그램은 Proxy DLL을 통하여 필요한 라이브러리의 기능들을 사용하게 된다. 이런 메커니즘 자체가 악성행위를 뜻하는 것은 아니다. 여러 경우 이런 메커니즘이 사용 되는데, 응용프로그램이 라이브러리 파일을 사용하기 전에 한시적으로 수행하고 싶은 작업이 있을 때 사용된다. 예를 들면, 응용프로그램을 종료하지 않고 라이브러리 파일을 업데이트하고 싶을 때, Proxy DLL에게 라이브러리 파일의 업데이트를 담당시켜, 라이브러리 파일을 언로드 한 후, 파일을 업데이트하고 다시 로드 하는 것이다. 이 경우 응용프로그램은 계속적으로 Proxy DLL을 로드하고 있기 때문에 종료할 필요도 또한 새롭게 코드를 수정하여 응용프로그램을 배포할 필요도 없게 된다.

Fig. 2는 Proxy DLL이 사용된 경우 응용프로그램, Proxy DLL, 라이브러리 간의 관계를 도식화한 것이다.

이런 메커니즘은 라이브러리 파일을 '동적 로딩'을 하여 사용하는 경우보다 '정적 로딩'을 하여 사용하는 경우 많이 사용된다. '동적 로딩'이란 응용프로그램이 실행 중 필요할 때 라이브러리를 로드하는 것이고, '정적 로딩'이란 응용프로그램 실행 시점에 필요한 라

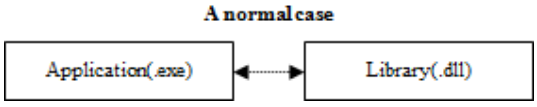

Fig. 1. Relationship between the application and the library in normal case.

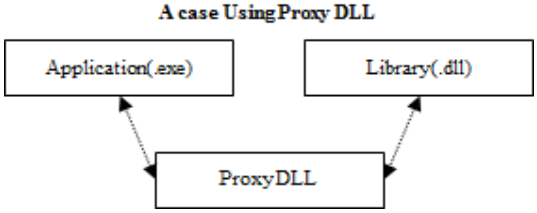

Fig. 2. Relationship among the application, the library and the Proxy DLL, when the Proxy DLL is used.

이브러리를 로드하는 것이다[8]. 동적 로딩은 '명시적(Explicit) 로딩'이라고도 하고, 정적 로딩은 '암시적(Implicit) 로딩'이라고도 한다.

동적 로딩을 사용하면 사용하려는 라이브러리 파일의 전체 경로를 지정하여 로드할 수 있기 때문에 Proxy DLL을 이용한 악성행위를 방어 할 수 있다. 또한 로드 전에 로드하려는 라이브러리에 대한 무결성 검사를 할 수 있고 이에 따른 오류 처리도 가능하다. 그러나 동적 로딩을 하려면 정적 로딩에 비해 개발자가 작성해야 하는 코드의 양이 증가하는 단점이 있다.

정적 로딩은 동적 로딩에 비해 훨씬 단순하게 사용 할 수 있다. 이는 응용프로그램 컴파일 시에 해당 라이브러리에 대한 정보가 실행파일 안에 저장되기 때문이다. 이런 특징으로 인해 응용프로그램 실행 시 정적 로딩을 할 라이브러리가 존재하지 않는다면 응용프로그램은 실행조차 되지 않게 된다. 모든 라이브러리에 대해 동적 로딩을 한다면 Proxy DLL과 같은 악성코드에 대한 방어율을 높일 수 있겠지만, 이런 경우 응용프로그램이 사용하는 라이브러리 내부에서도 다른 라이브러리를 사용하는 경우, 모두 동적 로딩을 하여야하기 때문에 이는 현실적으로 개발자에게 큰 부담을 안겨주게 된다.

정적 로딩을 사용하는 라이브러리의 정보는 실행파일 안에 저장되는데, 이런 경우 라이브러리에 대한 파일명만 저장되고 경로는 저장되지 않는다. 윈도우즈는 특정 경로를 순서대로 검색하여 라이브러리의 존재 유무를 검사하게 되는데, 만약 이 때 우선순위가 높은 경로에 동일 파일명을 가진 라이브러리가 존재한다면 이를 먼저 로드하게 된다.

다음은 응용프로그램이 라이브러리(dll)을 로드하려고 할 때, DLL이 존재하는 경로를 검색하는 순서이다[9].

1) 응용프로그램이 로드 된 디렉터리.
2) 시스템 디렉터리. GetSystemDirectory()를 이용하여 얻어오는 디렉터리
3) 16Bit 시스템 디렉터리.
4) 윈도우 디렉터리. GetWindowsDirectory()를 이용하여 얻어오는 디렉터리
5) 현재의 디렉터리
6) 'PATH' 환경 변수 디렉터리

정적 로딩은 로드하는 라이브러리 내에 응용프로그램이 사용하는 함수가 존재하지 않아도 응용프로그램은 실행이 되지 않는다. 이 역시 실행파일 내에 라이브러리의 파일명과 함께 사용하는 함수명도 저장되기 때문이다.

이런 윈도우즈의 특징으로 인해, 윈도우즈 라이브러리로 위장한 Proxy DLL은 파일명과 내부에 구현된 함수들의 함수명이 정상적인 윈도우즈 라이브러리와 동일하다. 또한 윈도우즈 라이브러리로 위장한 Proxy DLL은 사용자가 직접 실행하지 않더라도 다른 응용프로그램이 정상적인 라이브러리로 착각하고 실행해주게 된다. 특히 응용프로그램이 탐색기와 같이 사용이 빈번한 프로그램이라면 감염의 확률은 높아지게 된다.

IV. 윈티에 대한 이해

4.1 발견과 인지의 시작

2011년 가을, 대중적인 온라인 게임을 즐기는 사용자들의 컴퓨터에서 트로이 목마(Trojan)류의 악성코드가 발견되었다. 분석 결과, 이 악성코드는 온라인 게임사의 공식 업데이트 서버에서 정기 업데이트 통해 사용자 컴퓨터로 다운로드 된 것으로 밝혀졌다. 발견 초기에는 온라인 게임 공급자가 스파이웨어(Spyware)를 게임 사용자들에게 몰래 배포한 것으로 의심되었으나, 이는 온라인 게임사의 단순한 실수에 의하여 사용자들의 컴퓨터에 다운로드된 것으로, 이 악성코드 제작자들의 본래 목표는 컴퓨터 게임을 개발하고 배포하는 게임사였던 것으로 밝혀졌다[7].

해당 악성코드가 배포된 온라인 게임사의 요청에 의해 카스퍼스키랩이 분석한 결과, 온라인 게임사의 업데이트 서버에서 악성코드가 발견되었다. 이 악성코드들은 64-Bit Windows 환경에서 동작 되도록 작성된 DLL 라이브러리와 유효한 디지털 서명이 되어 있는 악성 드라이버 모듈이었다.

4.2 목표 게임사와 유출된 디지털 서명 소유 회사

윈티의 목표는 일반 사용자가 아닌 온라인 게임사들이었다. 목표가 된 온라인 게임사들의 대부분은 한국 기업이었고, 또한 감염된 것을 은폐하기 위하여 사용된 디지털 서명의 소유 회사 역시 다수가 한국의 IT 기업이었다. 이는 중국의 해킹 그룹들이 그들의 금전적인 이득을 위하여 한국의 게임사를 지속적으로 목표로 삼고 있는 것을 알 수 있는 부분이다. 이렇듯 특정 회사를 목표로 삼는 해킹 그룹과 그들이 개발한

악성코드들은 일반 시그니쳐 기반의 AV 제품으로는 진단이 거의 불가능하다. 불특정 다수에게 배포되는 악성코드와는 달리 특정 회사를 목표로 개발된 악성코드들은 그 샘플을 얻기도 힘들 뿐만 아니라 그 피해 역시 은밀하여 피해 사실 조차 인지하지 못하는 것이 대부분이기 때문이다.

Table. 1은 윈티의 공격을 받은 회사의 국가별 개수이다[7].

Table 1. The number of companies per country attacked by Winnti group.

Country	Count of Company
Korea	14
Japan	5
USA	2
China	2
Russia	2
Taiwan	2
Germany	2
Belarus	2
Brazil	1

Table. 2는 디지털 서명이 유출된 국가별 회사이다[7].

Table 2. List of companies whose digital certificates are stolen.

Country	Company
Korea	ESTsoft Corp.
Korea	Kog Co., Ltd
Korea	MGAME Corp.
Korea	Sesisoft
Korea	Neowiz
Korea/Philippines	LivePlex Corp.
Korea/Japan/USA	Wemade
Japan	Rosso Index KK
Japan	YNK Japan
China	Guangzhou YuanLuo
China	Fantasy Technology Corp

4.3 감염의 시작

윈티의 모든 감염은 DLL과 함께 시작한다. 윈티는 정상적인 윈도우즈 라이브러리를 가장한 'winmm.dll' 혹은 'apphelp.dll'를 이용하여 사용자의 컴퓨터를 감염 시킨다. 이 윈도우즈 라이브러리를 가장한 악성 DLL들은 응용프로그램이 정상적인 윈도우즈의 라이브러리인 'winmm.dll' 혹은 'apphelp.dll'을 로드 하려고 할 때, 먼저 로드된다. 그리고 응용프로그램에서는 정상적인 DLL이 로드된 것으로 착각하고 로드된 DLL의 익스포트(Export) 함수(Function)를 호출하지만, 실은 악성코드의 익스포트 함수가 실행되는 것이다

이는 응용프로그램에서 DLL을 로드 할 때, 윈도우즈가 DLL의 경로를 검색 하는 순서에 우선순위가 있다는 점과, DLL을 암시적 로딩 할 경우 DLL의 전체 경로 정보가 없다는 것을 이용 한 것이다.

DLL을 암시적 로드할 때에는, PE(Portable Executable) 구조로 되어 있는 실행 파일의 'Import Section'에 명시되어 있는 정보를 이용한다. 그런데 이 안에는 DLL이 존재하는 전체 경로가 저장되어 있는 것이 아니라 DLL의 파일명만 저장되어 있기 때문에, 앞서 설명한 경로의 순서대로 DLL이 존재하는 경로를 검색하는 과정을 거치게 된다. 윈티의 악성 DLL들은 이런 규칙을 이용하여 정상적인 윈도우즈 라이브러리보다 먼저 응용프로그램에 로드되게 된다.

정상적인 윈도우즈 라이브러리인 'winmm.dll' 혹은 'apphelp.dll'은 '%WINDIR%\System32'에 위치한다. 하지만 윈티의 악성 DLL들은 응용프로그램과 같은 경로에 위치하고 있다. 예를 들자면 윈도우즈의 탐색기인 'explorer.exe'는 '%WINDIR%'에 위치하고 있는데 윈티의 악성 DLL도 같은 경로에 위치함으로 정상적인 윈도우즈 라이브러리보다 먼저 로드된다.

4.4 감염의 방법

'explorer.exe'는 윈도우즈가 처음 부팅 될 때, 윈도우즈와 사용자간의 상호 통신을 위하여 실행되는 필수 프로그램이다. 또한 'explorer.exe'의 'Import Section'에는 'winmm.dll'의 정보가 저장되어있다. 결국 윈티의 악성 DLL이 'explorer.exe'와 같은 경로에 존재하게 되면, 윈도우즈의 필수 프로그램인

'explorer.exe'가 윈도우즈 부팅 시점에 실행되고, 'explorer.exe'의 'Import Section'에는 'winmm.dll'이 저장되어 있으므로, 'explorer.exe'가 실행되는 시점에 윈티의 악성 DLL이 로드되는 것이다.

Fig. 3은 'PEBrowse Professional'를 이용하여 'explorer.exe'의 'Import Section'을 확인한 것이다.

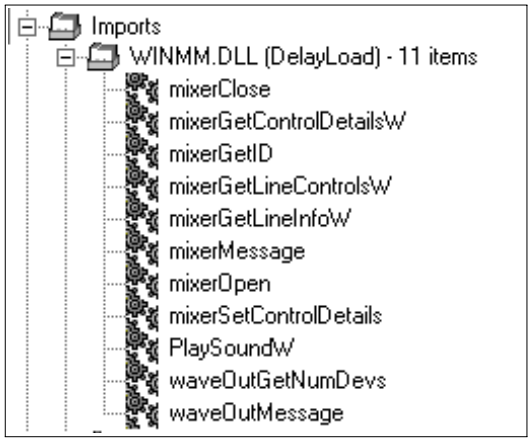

Fig. 3. The Import section information of 'explorer.exe' which embeds 'winmm.dll'

4.5 감염 후 응용 프로그램의 정상 동작

윈티의 악성 DLL에 'explorer.exe'에 감염이 된 이 후에도 'explorer.exe'는 정상 동작을 하게 된다. 원래 'explorer.exe'는 정상적인 윈도우즈 라이브러리가 로드 되지 않고 윈티의 악성 DLL이 로드되었기 때문에 정상적인 'winmm.dll'을 통해 윈도우즈에서 제공하는 기능을 사용할 수 없다. 이런 이유로 'explorer.exe'는 정상적인 동작을 할 수 없게 된다.

이런 점을 보완하기 위해서는 윈티의 악성 DLL 안에 해당 기능을 구현해주어야 한다. 하지만 악성코드에 정상적인 윈도우즈 라이브러리와 똑같은 기능을 구현 하는 것은 악성코드 입장에서 비효율적이다. 그래서 윈티의 악성 DLL은 자신이 로드되면서 정상적인 윈도우즈 라이브러리인 'winmm.dll'에 대해 명시적 로딩을 한다. 그리고 'explorer.exe'가 악성 DLL의 익스포트 함수를 호출하면 정상적인 'winmm.dll'의 익스포트 함수로 다시 포워딩 해주게 된다. 이렇게 되면 'explorer.exe'는 윈티의 악성 DLL에 감염 전과 똑같은 기능을 할 수 있게 된다. 이런 방식의 DLL을 'Proxy DLL'이라고 하며, 이런 기술적인 특징을 구현하기 위해 정상 'winmm.dll'와 윈티의 악성 DLL은 동일한 이름을 가진 익스포트 함수를 가지게 된다.

4.6 감염 후 응용 프로그램을 정상 동작시키기 위한 악성 'winmm.dll' 내의 코드

윈티에 감염된 이후에도 윈티의 악성 DLL을 로드한 응용프로그램은 정상동작하기 위해, 응용프로그램이 'winmm.dll'의 익스포트 함수를 호출 하였을 때, 윈티의 악성 DLL은 그 호출 흐름을 정상적인 'winmm.dll'로 전달해준다. 이의 대략적인 흐름은 다음과 같다.

1) 정상 윈도우즈 라이브러리를 로드하는 과정

윈티 악성 DLL은 'LoadLibraryA()'를 호출하여 '%WINDIR% \system32'에 위치한 'winmm'을 로드한다.

Fig. 4는 악성 'winmm.dll'이 정상 윈도우즈 라이브러리 'winmm'을 명시적 로딩 하는 코드를 IDA를 통해 확인 한 것이다.

```
bool __cdecl LoadLibraryAForProxy()
{
  CHAR LibFileName; // [sp+208h] [bp-104h]@1

  GetSystemDirectoryA(&LibFileName, 0x104u);
  lstrcatA(&LibFileName, "\\winmm");
  g_hModuleWinmm = LoadLibraryA(&LibFileName);
  return g_hModuleWinmm != 0;
}
```

Fig. 4. An example function code that shows Winnti's malicious DLL loads normal Windows library using dynamic loading.

2) 정상 윈도우즈 라이브러리의 함수 포인터 (Function Pointer)를 획득하는 과정

윈티 악성 DLL은 'GetProcAddress()'를 호출하여 얻고자 하는 'winmm'의 익스포트 함수 포인터를 얻어온다.

Fig. 5는 악성 'winmm.dll'이 정상적인 윈도우즈 라이브러리의 익스포트 함수 포인터를 얻어오는 코드를 IDA를 통해 확인 한 것이다.

```
FARPROC __stdcall GetProcAddressForProxy(LPCSTR lpProcName)
{
  int bResult; // eax@2
  CHAR v3; // [sp+0h] [bp-118h]@6
  FARPROC pfnFunc; // [sp+114h] [bp-4h]@4

  if ( !g_hModuleWinmm )
  {
    LOBYTE(bResult) = LoadLibraryAForProxy();
    if ( !bResult )
      ExitProcess(0xFFFFFFFFu);
  }
  pfnFunc = GetProcAddress(g_hModuleWinmm, lpProcName);
  if ( !pfnFunc )
  {
    if ( !((unsigned int)lpProcName >> 16) )
      wsprintfA(&v3, "%d", lpProcName);
    ExitProcess(0xFFFFFFFEu);
  }
  return pfnFunc;
}
```

Fig. 5. An example function code that shows Winnti's malicious DLL gets function pointer of normal Windows library.

3) 정상 윈도우즈 라이브러리의 함수를 호출하는 과정

응용프로그램에서 전달 받은 전달인자를 얻어온 함수 포인터에 넣고 호출해준다. 이 때 리턴 값은 정상적인 윈도우즈 라이브러리 함수가 리턴해 준 값을 사용하여 응용프로그램에게 다시 넘겨준다.

Fig. 6은 악성 'winmm.dll'이 정상 윈도우즈 라이브러리의 익스포트 함수를 호출하는 코드를 IDA를 통해 확인 한 것이다.

이런 과정을 통하여 윈티 악성 DLL을 로드한 응용프로그램은 평상시와 다름없이 정상 동작을 하게 된다.

4.7 감염 이후의 동작

윈티의 악성 DLL에 감염되면 악성 DLL은 다수의 DLL 라이브러리와 한 개의 .sys 커널 모드 드라이버를 생성하게 된다. 특히 커널 모드 드라이버는 64-Bit 윈도우즈 환경에서도 동작을 하는데, 이를 위해서 앞서 설명한 유출된 디지털 서명으로 서명이 되어있다. 64-Bit 윈도우즈에서는 디지털 서명이 되지 않은 커널 모드 드라이버는 실행이 되지 않는데, 이를 무력화하기 위하여 유출된 디지털 서명을 이용한 것이다. 이 커널 모드 드라이버는 일종의 루트킷으로 특정 네트워크의 주소에 대한 정보를 은폐시킨다. 이로 인해 유저 모드에서 실행되는 네트워크 모니터링 프로그램(예: 'netstat -a', 'TCPView')은 네트워크의 상태를 잘못 인식하게 된다. 윈티의 악성코드들은 이러한 모듈들을 이용하여 게임사 서버에 저장된 여러 정보를 특정 도메인 주소로 전송하게 된다.

V. 대응방안 및 실험

특정 소수를 목표로 개발된 윈티와 같은 악성코드들은 감염된 후에 생성된 파일들을 진단하는 것 보다 원천적으로 감염을 막는 것이 가장 효율적인 방법이다. 하지만 샘플을 구하기 힘든 윈티와 같은 경우는 시그니처 기반의 AV 제품으로는 도저히 진단하기 어렵다. 이런 이유로 본 논문에서는 정상적인 윈도우즈 라이브러리와의 비교를 진단 방법으로 제시한다. 이 방법은 윈도우즈 라이브러리를 가장한 악성 Proxy DLL에게 효과적이다.

이유는 첫째, 특정 집단을 목표로 하는 이른바 APT는 은밀하게 진행되기 때문에 그 공격을 주도하는 악성코드의 샘플을 수집하기 어렵기 때문에 분석된 악성코드를 기준으로 악성으로 판단하는 것 자체가 원천적으로 불가능하다. 만약 악성코드 샘플을 수집하여 분석 후에 대응을 한다면 이는 이미 피해가 발생 한 이후가 될 것이다. 이런 문제의 대안으로 정상적인 라이브러리 정보를 악성 판단의 기준으로 삼으면 아직 분석되지 않은 악성코드의 판별 기준을 세울 수 있다.

둘째, 윈티의 감염 방법과 같이 Proxy DLL 기법을 사용하는 악성코드들은 그들이 프록시 하려는 DLL과 일정 부분이 동일하다. 단발성의 악성코드들에 비해, 감염자의 시스템에서 가능한 오랫동안 검출되지 않고 활동해야 하는 APT 종류의 악성코드들은

```
BOOL __stdcall PlaySoundW(LPCWSTR pszSound, HMODULE hmod, DWORD fdwSound)
{
  FARPROC pfnFunc; // eax@1

  pfnFunc = GetProcAddressForProxy("PlaySoundW");
  return ((int (__stdcall *)(_DWORD, _DWORD, _DWORD))pfnFunc)(pszSound, hmod, fdwSound);
}
```

Fig. 6. An example function code that shows the Winnti's malicious DLL calls function of normal Windows library.

오랜 기간 동안 검출이 되지 않아야하기 때문에 시스템에 오류를 발생하는 일을 최대한 피하려 한다. 예를 들자면 Themida와 같은 패킹 기술을 적용하면 AV 제품에서 검출 될 수 있으며, Proxy 대상 DLL을 완벽하게 대체하지 못하면 시스템의 예상 할 수 없는 오류로 인하여 관리자의 주의를 끌 수 있다. 이는 특정 업체에 긴 시간을 투자하여 공격하는 APT의 특성상, 한번 침투 후에 짧은 시간 밖에 활동 하지 못한다면 공격자 입장에서 굉장히 비효율적이기 때문이다.

셋째, 윈티와 같이 특정 집단을 목표로 하는 APT는 대부분 거대 회사 혹은 단체 소유의 컴퓨터일 가능성이 높다. 그 중에서도 윈티와 같이 그 활동이 서버 시스템이 주요 목표일 가능성이 높다. 침입이 어려운 만큼 가장 가치 있는 정보에 접근해야하기 때문이다. 이런 시스템들은 일반 사용자의 컴퓨터와는 달리 설치되거나 삭제되는 응용 프로그램들이 굉장히 제한적이다. 특히 서버 시스템의 경우 처음 배치 시에 필요한 모든 응용프로그램을 설치하고 지속적으로 운용할 가능성이 크다. 운용 중에는 주기적으로 데이터 백업, 운영체제 및 사용하는 응용프로그램의 업데이트가 전부일 가능성이 높다. 이는 정상적인 응용프로그램 파일들의 변화가 적으며 충분히 관리 될 수 있음을 의미한다. 즉 이 논문에서 악성 판별의 기준을 정상적인 파일로 하고자 한 만큼 그 관리의 용이성이 보장되는 것이다.

5.1 임포트(Import)와 익스포트(Export) 함수

본 논문에서는 윈도우즈 라이브러리로 위장한 Proxy DLL을 판별하기 위하여, 정상적인 윈도우즈 라이브러리와 이를 위장한 Proxy DLL의 임포트 함수와 익스포트 함수를 비교한다.

PE 구조의 실행파일에서 보면 임포트 함수는 운영체제 혹은 다른 라이브러리가 제공하는 기능을 사용하겠다고 선언하는 것이고, 익스포트 함수는 반대로 다른 응용프로그램에게 제공하려는 기능을 선언한 것이다. 윈티 악성코드와 같은 Proxy DLL에서는 정상적인 윈도우 라이브러리와 동일한 익스포트 함수를 명시하여 다른 응용프로그램이 정상적으로 동작하게 하는 반면에, 실제 구현된 익스포트 함수의 기능은 정상적인 윈도우 라이브러리와는 다르기 때문에, 정상적인 윈도우 라이브러리의 임포트 함수와는 판이하게 다른 모습을 보인다.

이는 정상적인 윈도우 라이브러리를 악성코드 판별의 기준으로 삼을 수 있을뿐더러, 콜 시퀀스(Call Sequence)를 분석하기 위해 기계어를 분석하지 않아도 되므로 시스템의 자원 소모를 줄 일 수 있고 기

Table 3. File information of Winnti's malicious DLL and normal Windows library that is used for experiment. ('winmm.dll')

OS		SP	Version	Size	MD5
Winnti		-	-	270,336 Byte	50678ADEFC49735A4F236E06E83C089D
2000 x86		4	5.0.2161.00001	189,200 Byte	5B059A3D373301A5001C3BF1E6C580C4
XP	x86	3	5.1.2600.05512	167,936 Byte	56E7F186EEED2D09DD9B12D6D9A52304
	x64	2	5.2.3790.03959	310,784 Byte	46972CC7737808DF098DD2A763140FD0
2003 x86		No	5.2.3790.00000	161,792 Byte	96C050E78B45A4294AE62AC4FCDAADB0
		1	5.2.3790.01830	168,448 Byte	EF6FADF16E33E8B90EC32F0CC424F9F2
		2	5.2.3790.03959	165,888 Byte	3AE27700140A4F7DA16E127F6F10F3ED
Vista x86		No	6.0.6000.16386	193,024 Byte	3B5E50A380AE03249C9F60E5BB28EFCB
7	x86	1	6.1.7600.16385	194,048 Byte	26A634B2E0FD87F23541AD13A503CA72
	x64	1		217,600 Byte	EF2AE43BCD46ABB13FC3E5B2B1935C73
8.1	x86	No	6.3.9600.16384	128,576 Byte	8A606C90276DCAC67F3D45A0A235ECD6
	x64	No		123,936 Byte	6AA868B3C2A014AE76ECF53B667BF086
	x86	No	6.3.9600.17415	136,840 Byte	D3E5FBC4B4A87DB1036C431C90694D3B
	x64	No		126,056 Byte	33AE1B209D9BE2FC6835B8A35A889CEC

계어를 분석하지 않으면서도 32-Bit와 64-Bit 코드에서도 동일한 결과를 얻을 수 있다. 또한 이 비교는 정적 분석을 통해 진행되므로 가상의 시스템에서 진행해야 하는 동적 분석보다도 효율이 높다.

Table. 3은 실험에 사용된 윈티의 악성코드 'winmm.dll'과 정상적인 윈도우즈 라이브러리의 버전별 'winmm.dll'의 파일 정보이다.

5.2 익스포트(Export) 함수 비교

앞서 설명했듯이 익스포트 함수란 라이브러리가 다른 응용프로그램이나 라이브러리에게 자신의 기능을 제공 하는 것이다. 익스포트 함수는 하위 호환성을 위해 하위 버전에서 구현된 익스포트 함수명을 거의 수정하지 않고, 업데이트 시에는 새로운 익스포트 함수들이 추가 되는 것이 일반적이다.

Fig. 7은 윈도우즈의 실질적인 운영체제의 실행파일인 'ntoskrnl.exe'의 익스포트 함수명을 색으로 변환하여 시각화한 것이다. 그림에서 열은 윈도우즈 버전별 'ntoskrnl.exe'에 존재하는 익스포트 함수명을 색으로 표현하여 시각화를 한 것이고, 행은 버전별 익스포트 함수들의 존재 횟수를 뜻한다. 즉 행의 7은 7개의 버전에서 동일한 익스포트 함수명이 존재하는 것을 의미하고, 6은 6개의 버전에서 동일한 익스포트 함수명이 존재하는 것을 의미한다.

Fig. 7에서 알 수 있듯이 'ntoskrnl.exe'가 업데이트 되면서도 익스포트 함수는 특정한 패턴이 지속적으로 존재하는 것을 알 수 있다. 이는 앞서 설명했듯이 하위 호환성을 위해 이전 버전에 존재 했던 익스포트 함수를 삭제하지 않고 지속적으로 존재시키기 때문이다. 하지만 이는 익스포트 함수의 함수명, 전달인자, 리턴 값이 동일 한 것을 의미하는 것이지 함수 내부의 구현 자체도 동일 한 것을 의미하는 것은 아니다.

Proxy DLL은 정상적인 윈도우즈 라이브러리와 익스포트 함수명에 대해 높은 유사성이 있어야 정상적인 윈도우즈 라이브러리로 위장한 Proxy Dll이라고 할 수 있다.

위와 같은 특징을 이용하여, 윈티의 악성코드와 정상적인 윈도우즈 라이브러리를 비교하기 위해, 윈티의 악성코드 'winmm.dll'과 정상적인 윈도우즈 라이브러리 'winmm.dll'의 버전별 및 32-Bit / 64-Bit 별 익스포트 함수의 유사성을 비교하였다.

유사성을 비교해본 결과, 윈티의 악성 DLL은 정상적인 윈도우즈 라이브러리와 높은 유사성을 보여주었다.

Table. 4는 윈티의 악성 'winmm.dll'의 익스포트 함수명과 정상적인 윈도우즈 라이브러리 'winmm.dll'의 익스포트 함수명의 동일성을 측정 한 것이다.

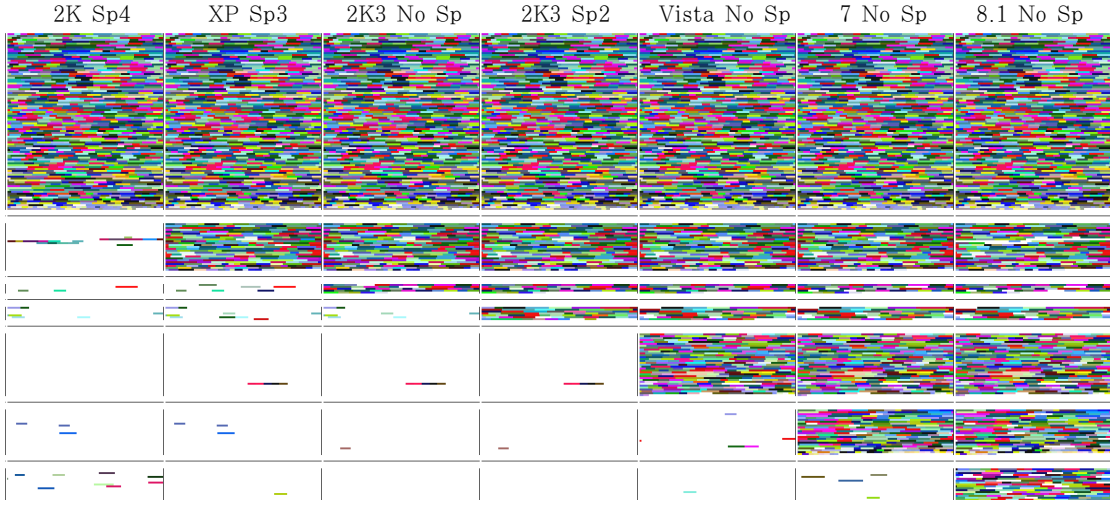

Fig. 7. Visualization of export function names and repeat count on various versions of 'ntoskrnl.exe'

Table 4. Similarity of Export functions on Winnti's malicious DLL and normal Windows library.

OS	Similarity
2000 SP4	99.50% [199 / 200]
XP x86 SP3	100.0% [210 / 210]
XP x64 SP2	98.99% [196 / 198]
2003 x86 No SP	100.0% [210 / 210]
2003 x86 SP1	100.0% [210 / 210]
2003 x86 SP2	100.0% [210 / 210]
Vista x86 No SP	100.0% [193 / 193]
7 x86 SP1	100.0% [193 / 193]
7 x64 SP1	100.0% [181 / 181]
8.1 x86 No SP 16385	100.0% [193 / 193]
8.1 x64 No SP 16385	100.0% [181 / 181]
8.1 x86 No SP 17415	100.0% [193 / 193]
8.1 x64 No SP 17415	100.0% [181 / 181]

5.3 임포트(Import) 함수 비교

임포트 함수란 응용프로그램이나 라이브러리가 다른 라이브러리로부터 그 기능을 제공 받는 것이다. 임포트 함수는 응용프로그램이나 라이브러리가 동일한 기능을 한다면 다른 라이브러리로부터 제공 받는 기능 또한 유사성이 존재한다.

Fig. 8은 윈도우즈의 실질적인 운영체제의 실행파일인 'ntoskrnl.exe'의 임포트 라이브러리명과 함수명을 색으로 변환하여 시각화한 것이다. 그림에서 열은 윈도우즈 버전별 'ntoskrnl.exe'에 존재하는 임포트 라이브러리명과 함수명을 색으로 표현하여 시각화를 한 것이고, 행은 버전별 라이브러리명과 함수명들의 존재 횟수를 뜻한다. 즉 행의 7은 7개의 버전에서 동일한 임포트 라이브러리명과 함수명이 존재하는 것을 의미하고, 6은 6개의 버전에서 동일한 임포트 라이브러리명과 함수명이 존재하는 것을 의미한다.

그림에서 알 수 있듯이 'ntoskrnl.exe'가 업데이트 되면서도 임포트 라이브러리와 함수명 역시 특정한 패턴이 지속적으로 존재하는 것을 알 수 있다. 이는 앞서 설명 했듯이 동일한 기능을 수행하기 위해선 동일한 라이브러리로부터 그 기능을 제공 받아야하기 때문이다. 응용프로그램이나 라이브러리가 가지고 있는 기능이 추가 된다면 다른 라이브러리로부터 제공받는 기능 역시 추가 될 수 있다.

윈도우즈 라이브러리로 위장한 Proxy DLL은 이 부분에 있어 정상적인 윈도우즈 라이브러리와는 판이하게 다르게 되는데, 그 이유는 외부로 보이는 것은 동일하나 내부적으로는 구현 목적이 완전히 다르기 때문이다.

위와 같은 특징을 이용하여, 윈티의 악성코드와 정상적인 윈도우즈 라이브러리를 비교하기 위해 윈티의

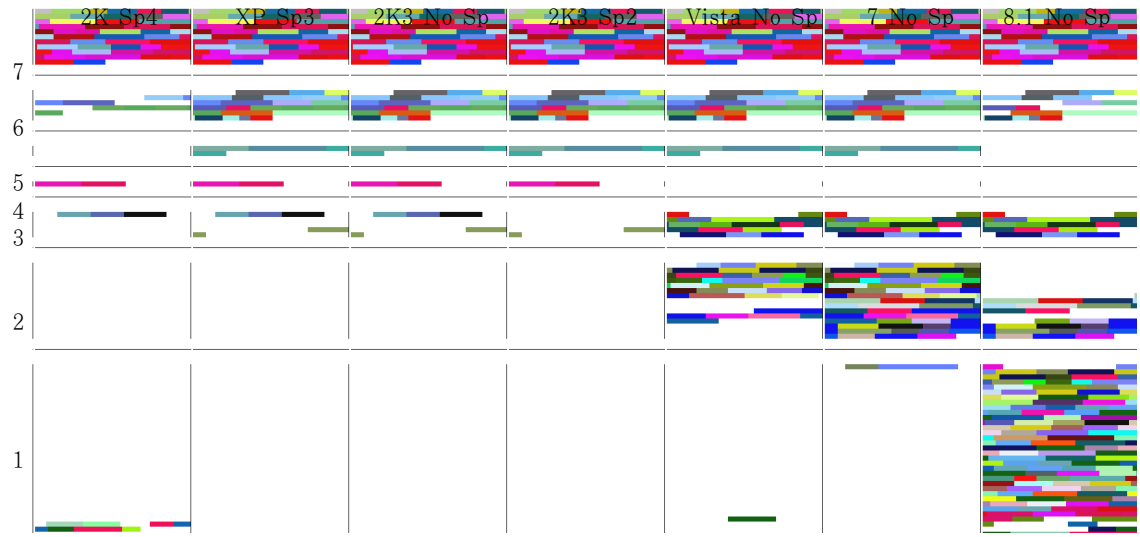

Fig. 8. Visualization of import libraries, function names and repeat count on various versions of 'ntoskrnl.exe'

악성코드 'winmm.dll'과 정상적인 윈도우즈 라이브러리 'winmm.dll'의 버전별 및 32-Bit / 64-Bit 별 임포트 라이브러리의 유사성을 비교하였다.

유사성을 비교해본 결과, 윈티의 악성 DLL은 정상적인 윈도우즈 라이브러리와 낮은 유사성을 보여주었다.

Table. 5는 윈티의 악성 'winmm.dll'의 임포트 라이브러리명 및 함수명과 정상적인 윈도우즈 라이브러리 'winmm.dll'의 임포트 라이브러리명 및 함수명의 동일성을 측정 한 것이다.

Table 5. Similarity of Import functions on Winnti's malicious DLL and normal Windows library.

OS	Similarity
2000 Sp4	7.91% [22 / 278]
XP x86 Sp3	8.27% [22 / 266]
XP x64 Sp2	8.53% [22 / 258]
2003 x86 No Sp	8.43% [22 / 261]
2003 x86 Sp1	8.30% [22 / 265]
2003 x86 Sp2	8.30% [22 / 265]
Vista x86 No Sp	12.69% [33 / 260]
7 x86 Sp1	10.98% [27 / 246]
7 x64 Sp1	10.92% [26 / 238]
8.1 x86 No Sp 16385	1.59% [6 / 377]
8.1 x64 No Sp 16385	1.08% [4 / 370]
8.1 x86 No Sp 17415	1.59% [6 / 377]
8.1 x64 No Sp 17415	1.08% [4 / 370]

5.4 실험 결론

정상적인 윈도우즈 라이브러리를 가장하는 악성코드의 경우, 외부 프로그램으로 제공하는 기능은 정상적인 윈도우즈 라이브러리와 외적으로 동일하여야 한다. 하지만 내부적으로는 기능의 목적이 다르기 때문에 구현이 동일 할 수 없다. 즉 악성코드가 그 기능을 구현하기 위해서 다른 라이브러리로부터 제공 받는 기능들은 정상적인 윈도우즈 라이브러리가 제공받는 기능들과는 상이하게 된다. 이는 라이브러리라고 하여도 모든 기능을 해당 라이브러리안에서 모두 구현 할 수 없고, 라이브러리 역시 다른 라이브러리로부터 기능을 제공 받아야하기 때문이다.

익스포트 함수 비교 실험에서는 악성코드가 외부적으로 정상적인 윈도우즈 라이브러리와 동일함을 확인하였다. 이는 최소 98.99%의 동일성에서 최대 100%의 동일성을 보여주고 있다.

하지만 임포트 함수 비교 실험에서는 악성코드가 내부적으로 정상적인 윈도우즈 라이브러리와 상이함을 확인 할 수 있다. 이 경우에는 최대 12.69%의 동일성에서 최소 1.08%의 동일성을 보여줌으로 정상적인 윈도우즈 라이브러리와 확연한 차이를 보여준다.

이는 정상적인 윈도우즈 라이브러리의 정보만 관리된다면 윈도우즈 라이브러리를 가장하여 침투하는 새로운 악성코드를 진단 할 수 있음을 의미한다. 그러기 위해 본 논문의 실험에서는 정상적인 윈도우즈 라이브러리를 위장하여 침투하는 윈티의 악성코드를 기반으로 정상적인 윈도우즈 라이브러리와의 차이점을 확인하였다.

Fig. 9는 윈티의 악성코드와 정상적인 윈도우즈 라이브러리 'winmm.dll'의 익스포트 함수명의 유사

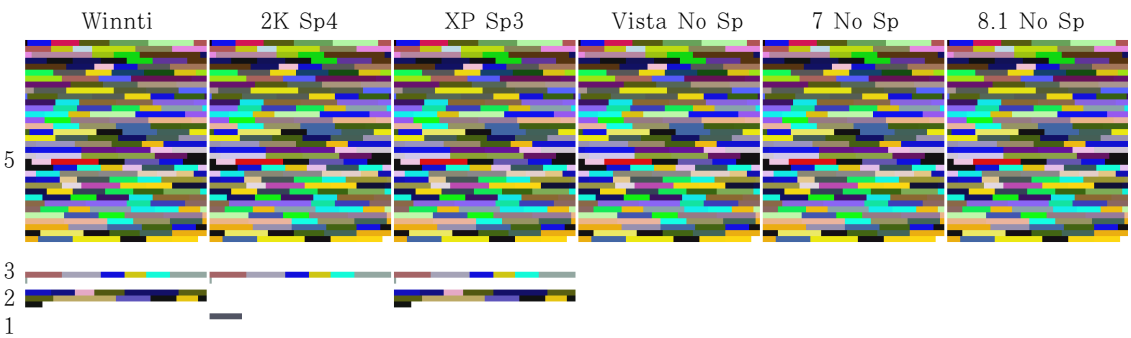

Fig. 9. Visualization of export function names and repeat count on Winnti's malicious DLL and normal Windows libraries

성을 시각화 한 것이다.

Fig. 10은 윈티의 악성코드와 정상적인 윈도우즈 라이브러리 'winmm.dll'의 임포트 라이브러리명과 함수명의 유사성을 시각화 한 것이다.

앞서 익스포트 함수 비교 결과인 Table. 4에서 보여 주듯이 악성코드와 정상적인 라이브러리가 상당히 유사한 것을 알 수 있다. 하지만 임포트 라이브러리와 함수 비교 결과인 Table. 5와 같이, 악성코드와 정상적인 라이브러리의 유사성은 거의 없는 것을 알 수 있다.

5.5 향후 유사 공격 전망과 대응방안

윈티의 악성코드를 분석한 후 추측하건데, 윈티 그룹이 'winmm.dll'을 타겟으로 위장한 이유는 윈도우즈 각 버전별 정상적인 'winmm.dll'의 변화가 거의 없는 것 이라는 점이다. 악성코드 제작자의 입장에서 생각해보면 자신들이 개발한 악성코드가 여러 버전의 윈도우즈에서 동작하길 원할 것이다. 하지만 그 목적을 달성하기 위해서, 악성코드를 윈도우즈 버전별로 동작하도록 여러 버전으로 제작하는 것은, 침투도 쉽지 않은 상황에서 해킹의 성공률을 더욱 떨어뜨리는 결과를 초래할 것이다. 앞서 설명 했듯이 정적 로딩을 이용하여 라이브러리를 로드하려면 파일명과 그 안에 구현된 익스포트 함수명이 정상적인 라이브러리와 동일하여야 한다. 하지만 윈도우즈 버전별로 익스포트 함수명이 계속 변하거나 추가되어 정상적인 라이브러리가 변화된다면, 정상적인 윈도우즈 라이브러리로 위장한 악성코드는 실행조차 되지 않을 가능성이 높다. 그러므로 이런 윈도우즈의 정상적인 라이브러리로 위장한 Proxy DLL을 대응하기 위해서는 윈도우즈 버전별로 변화가 거의 없는 시스템 라이브러리를 조사하고 이들의 익스포트 함수와 임포트 함수 정보를 지속적으로 관리한다면 이런 종류의 악성코드를 좀 더 수월하게 방어 할 수 있을 것이다. 이는 새롭게 개발된 악성코드를 AV 제품의 시그니처에 추가하는 시간이 원천적으로 필요하지 않게 함으로 해킹의 성공률을 낮출 수 있다.

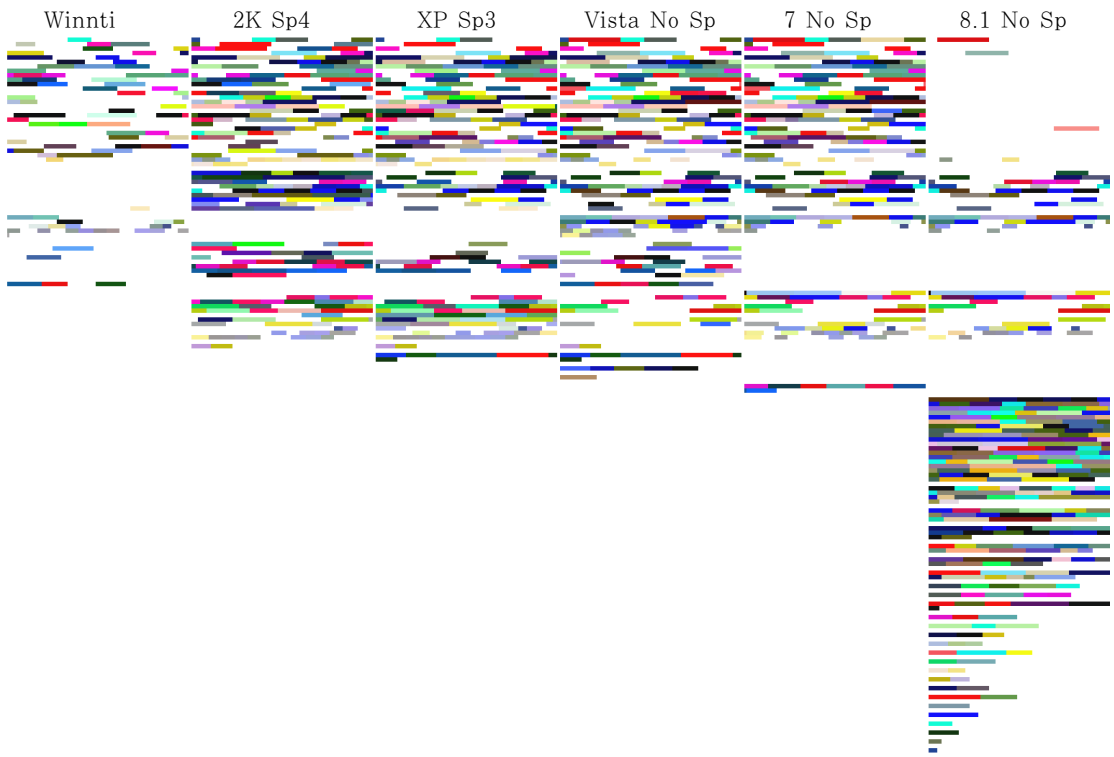

Fig. 10. Visualization of import library names and function names on Winnti's malicious DLL and normal Windows libraries

VI. 결 론

윈티와 같이 정상적인 윈도우즈 라이브러리를 가장하여 감염시키는 Proxy DLL 방식의 악성코드의 경우 정상적인 윈도우즈 라이브러리를 기준으로 악성코드를 판별 할 수 있다. 본 논문에서는 라이브러리가 외부에 자신의 기능을 제공하는 익스포트 함수와 다른 라이브러리의 기능을 제공 받는 임포트 함수의 정보를 사용하여 이런 악성코드 대응 방안을 제시하였다. 특히 제시한 방안은 사용자의 프로그램의 설치가 빈번한 일반 사용자의 컴퓨터보다 서버에서 활용되기 효율적이며 또한 APT 목표 역시 이런 서버를 목표로 삼기 때문에 적절한 방어 방법이 될 수 있다. 그리고 간단한 정적분석을 통해 익스포트와 임포트 정보를 얻어오기 때문에 기계어를 분석하는 호출 시퀀스 분석이나 가상머신에서 그 행위를 관찰하는 방법보다 훨씬 저비용으로 감지를 해낼 수 있는 장점이 있다. 향후에는 악성코드들이 정상적인 윈도우즈 라이브러리로 가장하기 용이한 윈도우즈 라이브러리의 종류와 그 이유를 연구하여 변종 악성코드 진단 등 좀 더 향상된 방법을 제시 할 것이다.

References

[1] Fanglu Guo, Peter Ferrie and Tzi-cker Chiueh, "A Study of the Packer Problem and Its Solutions," In 11th International Symposium on Recent Advances in Intrusion Detection, pp. 98-115, 2008

[2] Ilsun You and Kangbin Yim, "Malware Obfuscation Techniques: A Brief Survey," In International Conference on Broadband, Wireless Computing, Communication and Applications, IEEE Computer Society. pp. 297-300, 2010

[3] A. Moser, C. Kruegel and E. Kirda, "Exploring Multiple Execution Paths for Malware Analysis," In IEEE Symposium on Security and Privacy, pp. 231-245.A., 2007

[4] C. Xuan, J. Copeland and R. Beyah, "Toward Revealing Kernel Malware Behavior in Virtual Execution Environments," In 12th International Symposium on Recent Advances in Intrusion Detection, pp. 304-325., 2009

[5] M. Preda, "Code Obfuscation and Malware Detection by Abstract Interpretation," In Dipartimento di Informatica, 2010.

[6] Ahmed F.Shosha, Chen-Ching Liu and Pavel Gladyshev, "Evasion-Resistant Malware Signature Based on Profiling Kernel Data Structure Objects," 2012 7th International Conference on Risks and Security of Internet and Systems (CRiSIS), 2012.

[7] Kaspersky Lab Global Research and Analysis Team, "winnti - more than just a game," Kaspersky Lab, 2013

[8] Microsoft MSDN Dynamic-Link Library https://msdn.microsoft.com/en-us/library/windows/desktop/ms686912(v=vs.85).aspx

[9] Microsoft MSDN Dynamic-Link Library https://msdn.microsoft.com/en-us/library/windows/desktop/ms682586(v=vs.85).aspx#standard_search_order_for_desktop_applications

[10] Lee, Ho Dong, "Structure and Concept of Windows System Executable File," Hanbit Media, 2005

〈저자소개〉

구 준 석 (JunSeok Koo) 정회원
2013년 9월~현재: 고려대학교 정보보호대학원 정보보호학과 석사과정
〈관심분야〉 Windows Kernel Analysis, Reverse Engineering, Secure Coding

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