API 순차적 특징을 이용한 악성코드 변종 분류 기법

한경수[1], 김인경[2], 임을규[3]

Malware Family Classification Method using API Sequential Characteristic

Kyoung-Soo Han[1], In-Kyoung Kim[2], Eul-Gyu Im[3]

요 약

공격자들은 금전적인 이득을 목적으로 악성코드를 제작하여 유포시킴으로써 사용자들의 컴퓨터를 감염시키고 ID 및 패스워드, 메일 주소, 휴대폰 번호, 금융 정보와 같은 각종 개인정보들을 감염된 사용자들의 컴퓨터로부터 유출시킨다. 또한 감염된 사용자들의 컴퓨터를 봇넷으로 구성하여 DDoS 공격, 스팸 발송 등의 공격을 수행한다. 악성코드 자동 생성 도구는 공격자들 사이에 보편화되어 신종 및 변종 악성코드를 쉽게 생성할 수 있게 해준다. 이로 인해 국내외에서 새롭게 발견되는 악성코드의 수는 지속적으로 증가하는 추세이다. 그러나 악성코드에 대한 대응 방법은 아직까지 대부분 악성코드를 수동으로 분석하고 시그니처를 생성하는 방법으로 대응하고 있어 악성코드의 전파 속도에 뒤처지고 있다. 이에 따라 악성코드에 의한 직접적인 피해 및 2차적인 피해는 급속도로 확산되고 있으므로 악성코드에 대한 신속한 분석과 대응이 필요하다. 본 논문에서는 악성코드에 대한 정적 분석을 수행하여 얻을 수 있는 API 리스트의 순차적 특징을 이용하여 보다 빠른 악성코드 변종 분류 방법을 제안하고, 각종 악성코드 샘플을 대상으로 수행한 실험 및 결과에 대하여 기술한다.

핵심어 : 악성코드, 악성코드 변종 분류

Abstract

Malware is generated to gain profit by attackers, and it infects many user's computers. As a result, attackers can acquire private information such as ID, password, e-mail address, cell-phone number and banking account from infected machines. Moreover, infected machines are used for cyber-attacks such as DDoS attack, spam, and so on. The number of new malware discovered everyday is increasing continuously because the automated tools allow attackers to generate the new malware or their variants easily. However, new countermeasures against the malware are invented at lower rate than the propagation rate of the malware. Therefore, rapid malware analysis method is required in order to mitigate the infection rate and secondary damage to the users. In this paper, we proposed a malware variants classification method using sequential characteristics of API list, and described an experiment result with some malware samples.

Keywords : Malware, Malware Variants Classification

접수일(2011년02월06일), 심사의뢰일(2011년02월07일), 심사완료일(1차:2011년02월16일, 2차:2011년02월28일)

게재일(2011년04월30일)

[1]133-791 서울시 성동구 행당1동 17 한양대학교 전자컴퓨터통신공학과.
email: 1hanasun@hanyang.ac.kr

[2]133-791 서울시 성동구 행당1동 17 한양대학교 전자컴퓨터통신공학과.
email: blackangel@hanyang.ac.kr

[3](교신저자) 133-791 서울시 성동구 행당1동 17 한양대학교 컴퓨터공학부 교수.
email: imeg@hanyang.ac.kr

1. 서론

전 세계적으로 컴퓨터와 인터넷의 보급이 확산됨에 따라 정보 네트워크의 구축이 더욱 광범위해지고 있다. 그러나 이와 더불어 바이러스나 웜, 트로이목마와 같은 악성코드 역시 증가하고 있으며, 사용자들의 컴퓨터를 감염시켜서 각종 정보를 유출하거나 또 다른 시스템을 공격하는데 사용하는 등 그 부작용도 꾸준히 증가하고 있다. 또한 이렇게 악성코드에 감염된 사용자들의 컴퓨터가 연결되어 거대한 봇넷(Botnet)을 형성하고 더욱 광범위한 사이버 공격에 이용된다[1-2]. 공격자들은 금전적인 이득을 목적으로 악성코드를 제작하여 유포시킴으로써 사용자들의 컴퓨터를 감염시킨다. 이때 공격자들은 보편화된 악성코드 자동 생성 도구와 기존의 악성코드를 이용하여 손쉽게 신종 및 변종 악성코드를 제작하기 때문에 이로 인해 매일 새롭게 발견되는 악성코드의 수와 종류는 기하급수적으로 증가하고 있다. 그러나 악성코드에 대한 대응 방법은 대부분 수동으로 악성코드를 분석하고 시그니처를 생성함으로써 대응하고 있어 악성코드의 전파 속도에 뒤처지고 있다. 이에 따라 악성코드에 의한 직접적인 피해 및 2차적인 피해는 급속도로 확산되고 있으므로 악성코드에 대한 신속한 대응이 필요하다.

본 논문에서는 악성코드의 변종을 빠르게 분류하기 위해 정적 분석을 통해 얻을 수 있는 API 순차적 특징을 이용한 시스템을 제안한다. 이는 악성코드가 윈도우(Windows)에서 실행되기 위해서 실행 가능한 파일 포맷인 PE(Portable Executable) 형식을 가지고 있기 때문에 PE 파일에 대한 정적 분석을 수행하여 악성코드에 삽입된 API 리스트를 분리하고, 해당 API 리스트에 나타난 순차적 특징을 비교하는 알고리즘을 통해 유사도를 계산함으로써 어떤 악성코드의 변종인지 분류하는 방법이다. 실험에서는 트로이목마 악성코드의 세부 분류인 Trojan-DDoS와 Trojan-Spy 샘플들을 대상으로 API 순차적 특징의 유사도를 비교하여 기술한다.

본 논문의 구성은 다음과 같다. 2장에서는 API와 악성코드의 분석 방법과 관련된 배경지식에 대하여 설명하고, 3장에서는 악성코드의 탐지 및 분류 방법에 대한 관련 연구를 기술한다. 4장에서는 API 리스트의 순차적 특징을 이용하여 악성코드 변종을 분류하는 기법을 제안하고, 5장에서는 제안한 기법을 바탕으로 실험한 결과를 기술한다. 마지막으로 6장에서는 결론과 향후 연구 방향에 대하여 제시하고자 한다.

2. 배경지식

2.1. API

API(Application Programming Interfaces)란 응용프로그램에서 사용할 수 있도록 운영체제나 프로그래밍 언어가 제공하는 기능을 제어할 수 있도록 만든 인터페이스를 의미하며, 다른 응용프로그램과 상호작용을 하게 해주는 것이다[3-4]. 또한 윈도우에서도 마찬가지로 응용프로그램을 위한

함수들을 Windows API로 제공된다. Windows API들은 동적 라이브러리(DLL)에 포함되어 윈도우 프로그래밍시 사용할 수 있게 해주며, 사용자 모드와 커널 모드에서 동작한다. 특히 커널 모드에서 동작하는 API는 Native API라 부른다[5]. [표 1]은 윈도우 사용자 모드의 주요 DLL과 포함된 API의 예를 나타낸 것이다.

[표 1] 윈도우 사용자 모드의 주요 DLL과 API의 예
[Table 1] The Example of DLLs and APIs in User Mode of Windows

	설명	예
KERNEL32.DLL	윈도우 커널이 제공하는 모든 작업 처리 메모리 관리, 파일 입출력, 작업 관리 등	LoadLibraryA, GetCurrentProcess, ExitProcess, TerminateProcess, GetFileType, CreateFileA, WriteFile 등
USER32.DLL	사용자 인터페이스 모든 윈도우 조작 로직 처리	MessageBoxA, CreateWindowExA, SetCapture, SendMessageA 등
GDI32.DLL	그래픽 장치 인터페이스 화면 및 프린터에 텍스트와 그래픽 출력	DeleteDC , SetTextColor, GetWindowOrgEx, GetTextMetricsA, GetTextExtentPointA 등

2.2. API 리스트 추출 방법

2.2.1. 커널 후킹

커널 후킹을 이용한 API 리스트 추출 방법으로는 SSDT(System Service Descriptor Table) 후킹이나 IDT(Interrupt Descriptor Table) 후킹 등이 존재하며, 주로 Native API에 대한 정보를 얻기 위해 사용된다. SSDT 후킹은 커널 모드에서 서비스를 받기 위해 필요한 테이블 내 함수의 메모리 주소를 변경하거나 테이블 자체를 프로그램 내부 메모리로 리다이렉트함으로써 사용된 Native API의 리스트를 추출할 수 있다. IDT 후킹은 인터럽트를 처리하는데 사용되는 테이블을 변경한다. 이때 IDT 내의 인터럽트 처리 수로를 변경하여 특정 인터럽트를 후킹함으로써 Native API 정보를 추출한다[5-7].

2.2.2. IAT 분석

PE(Portable Executable) 파일 포맷은 윈도우 운영체제 로더가 실행 코드를 관리하는데 필요한 정보를 캡슐화한 데이터 구조이다. 이는 링크를 위한 동적 라이브러리 참조와 API Export 및 Import 테이블, 리소스 관리 데이터, 스레드 로컬 저장 데이터를 포함한다. 일반적으로 EXE 파일을 실행하게 되면 Windows 운영체제 로더가 파일의 구조를 분석하고 메모리에 로드하여 프로그램의 진입점으로 들어가게 하며, 로드하는 동안 파일 내부의 Import 정보를 통해 필요한 DLL도

찾아 메모리에 로드한다. PE 파일에는 .text, .data, .rdata, .idata, .edata 등의 섹션들이 포함되는데, 그중에서도 .idata 섹션에는 다른 DLL을 통해 사용할 수 있는 API들의 테이블에 관한 정보가 포함된다. .idata는 IMAGE_IMPORT_DESCRIPTOR들로 이루어져 있고, 이는 IMAGE_IMPORT_BY_NAME을 가리키며, 이곳에 사용될 API 이름과 실행되면서 할당되는 주소가 저장된다. 이러한 API 엔트리 포인트 배열을 IAT(Import Address Table)이라 한다[6][8]. 결국 응용프로그램이 API를 이용하려면 해당 API의 주소를 알아야 하며, 이를 저장하기 위해서는 IAT(Import Address Tale)를 이용하는 것이다. 따라서 PE 파일을 분석함으로써 IAT를 찾고, IAT로부터 포함된 API 리스트를 추출할 수 있다. 본 논문에서는 이 방법을 사용하여 악성코드의 API 리스트를 추출한다.

3. 관련연구

3.1. 악성코드 분석 방법

악성코드를 탐지하고 분류하기 위한 분석 방법으로는 행위를 분석하는 동적 분석 방법과 코드를 분석하는 정적 분석 방법이 존재한다. 동적 분석은 분석환경 내에서 악성코드를 실행시킴으로써 수행되는 악성 행위 및 위험요소를 실시간으로 모니터링하고 추적하는 방법이다. 정적 분석은 악성코드를 실행하지 않고 구성 요소들의 연관성 및 호출 관계 등을 분석함으로써 악성코드의 구조와 삽입된 DLL(Dynamic Link Library) 등을 파악할 수 있으며, 소프트웨어 역공학(Reverse Engineering)에서 주로 사용된다[9-10].

3.2. 동적 분석 기반의 탐지 및 분류 방법

동적 분석을 기반으로 하는 악성코드 탐지 기법은 악성코드의 행위에 대한 특징이 필요하다.

[10]에서는 실행 압축의 우회기법을 이용한 악성코드에 대하여 행위 기반의 탐지 패턴을 생성하고, 신종 및 변종 악성코드를 탐지하는 방법을 제안하였다. 제안한 방법은 시스템의 중요 자원에 접근하려는 API를 대상으로 동적 분석과 정적 분석을 수행하여 악성코드의 행위 및 순차 패턴의 통합 과정을 통해 악성코드를 동적으로 탐지하기 위한 시그니처를 생성하는 방법을 제안하였다.

[11]에서는 그래프 마이닝(Graph Mining)과 개념 분석(Concept Analysis)을 이용하여 특정 악성코드 셋트에서 나타날 수 있는 행위에 대한 특징들을 자동으로 추출하는 방법을 제안하였다. 제안한 방법은 행위가 유사한 악성코드들의 하위 셋트(Subset)로 분류하고, 각 해당 악성코드를 대표할 수 있는 악성 행위에 대한 그래프를 구성하여 클러스터링함으로써 악성코드 패밀리에 대한 함축적 행위(Significant Behavior)로 종합하는 방법이다.

[12]에서는 에뮬레이터를 기반으로 악성코드의 행위를 모니터링하여 자동으로 시스템 콜 인자와 반환 값, 오류 상태 등의 주요 속성을 기록하는 “API Capture”를 개발하였다. [13]에서는

Metamorphic Generator로부터 생성된 악성코드를 탐지하기 위해 에뮬레이터를 이용한 동적 모니터링을 기반으로 악성코드에서의 API 콜을 추적하고 빈도수를 측정하여 CAPI(Critical API)을 추출함으로써 시그니처를 생성한 다음 통계학적 방법으로 CAPI 비율간의 차이를 계산하고 악성코드를 분류하는 방법을 제안하였다.

3.3. 정적 분석 기반의 탐지 및 분류 방법

악성코드를 특정 시그니처로 정의하기 위해 악성코드를 분석하는 방법 중의 하나로 플로우차트(Flow Chart) 혹은 플로우 그래프(Flow Graph)를 이용하는 방법이 존재한다. [14]에서는 바이너리를 분석하여 도출된 함수의 호출 흐름을 시그니처로 정의하고, 이를 이용하여 악성코드를 탐지하는 방법을 제안하였다. 이는 악성코드에서 시스템 콜의 호출 관계를 분석하여 콜 그래프(Call Graph)로 나타내고, 프로세스/메모리/소켓 등 32개의 시스템 오브젝트에 대한 Read/Write/Open/Close의 네 가지 행위를 하는 API들로 그룹화하여 코드 그래프(Code Graph)로 단순화한 후 유사도를 측정함으로써 악성코드를 탐지하는 방법이다. [15]에서는 컨트롤 플로우 그래프(Control Flow Graph)를 이용하여 문자열(String) 형태의 플로우 그래프 시그니처를 생성하고 유사도를 계산함으로써 악성코드의 변종을 탐지할 수 있는 시스템을 제안하였다. [16]에서는 스크립트 악성코드를 대상으로 각 코드의 기능을 파악하여 컨트롤 플로우에 따른 그래프와 변수의 사용 여부 등 코드의 의존도에 따라 의존도 그래프(Dependency Graph)로 나타내고, Hybrid Genetic Algorithm을 이용함으로써 서브그래프의 유사도를 측정하여 악성코드를 탐지하는 방법을 제안하였다.

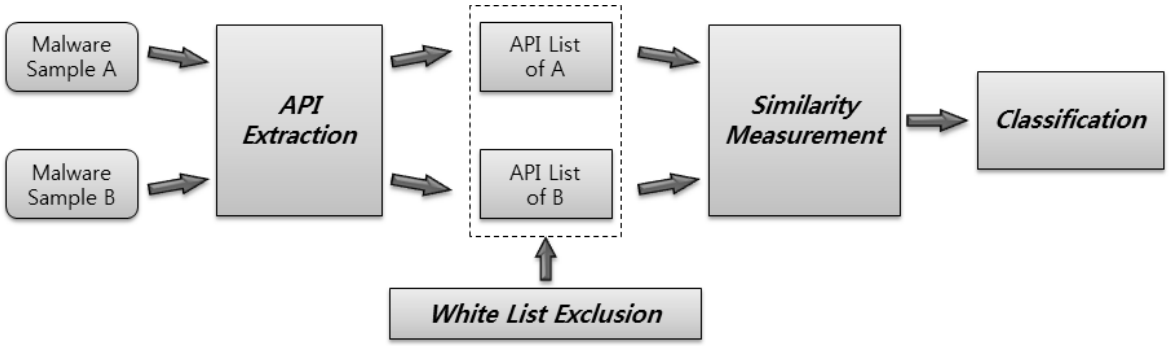
[17]에서는 Metamorphic 악성코드의 시스템콜 및 라이브러리 함수 호출을 기반으로 특정 악성 행위를 나타낼 수 있는 코드 조각에 대한 의미(Semantics)나 기능(Functionality)을 특징화함으로써 패턴을 생성하는 방법과 유사도 계산을 통한 코드 패턴 매칭 방법을 제안하였다. [18]에서는 PE 파일의 정적 분석을 기반으로 2개의 파일을 비교하기 위해 함수 리스트를 벡터로 나타내고, 함수 내의 명령어 빈도를 기준으로 내림차순 정렬시킨 후 코사인 유사도 분석(Cosine Similarity Analysis)을 통해 유사도를 측정하는 방법을 제안하였다.

[19]에서는 ClamAV[20]를 확장시켜 빠른 탐지와 메모리를 적게 사용하는 것에 초점을 맞추고, 이를 위해서 Feed- Forward Bloom Filter(FFBF)를 사용하였다. 이는 전체 파일을 w 길이의 슬라이딩 윈도우로 스캔하고, 스캔된 내용에 대하여 블룸필터를 통해 비트 벡터(Bit Vector)를 생성한 후 데이터베이스에 존재하는 기존 악성코드의 비트 벡터와 비교하여 동일한 값이 포함되어 있을 경우 악성코드를 탐지하는 방법이다.

4. 제안하는 방법

4.1. 개요

본 논문에서는 PE 파일이 실행될 때 필요한 DLL 및 API 정보를 가진 IAT(Import Address Table)로부터 API 리스트를 추출하고, API의 종류 및 순차적 특징을 이용하여 악성코드를 분류하는 방법을 제안한다. 전체적인 방법의 흐름은 [그림 1]과 같다.



[그림 1] 제안하는 방법의 전체 흐름도

[Fig. 1] The Overview of Proposed Method

정적 분석을 통해 추출된 API 리스트로부터 유사도를 측정하여 두 악성코드의 유사성, 즉 동일한 악성코드의 변종 여부를 탐지하는 방법은 다음과 같은 단계를 수행한다.

■ 단계 1.

악성코드들로부터 API 리스트를 추출한다. 일반적으로 프로그램은 내부에 IAT를 포함하여 호출할 API 리스트를 관리한다. 제안한 방법은 IAT로부터 해당 악성코드의 API를 추출하는 방법을 사용한다.

■ 단계 2.

악성코드가 아닌 정상적인 프로그램으로부터 자주 포함되는 API의 리스트를 DLL 별로 저장하여 화이트리스트(White List)를 생성한다. 그리고 악성코드로부터 추출된 API 리스트에서 화이트리스트의 API를 제거한다.

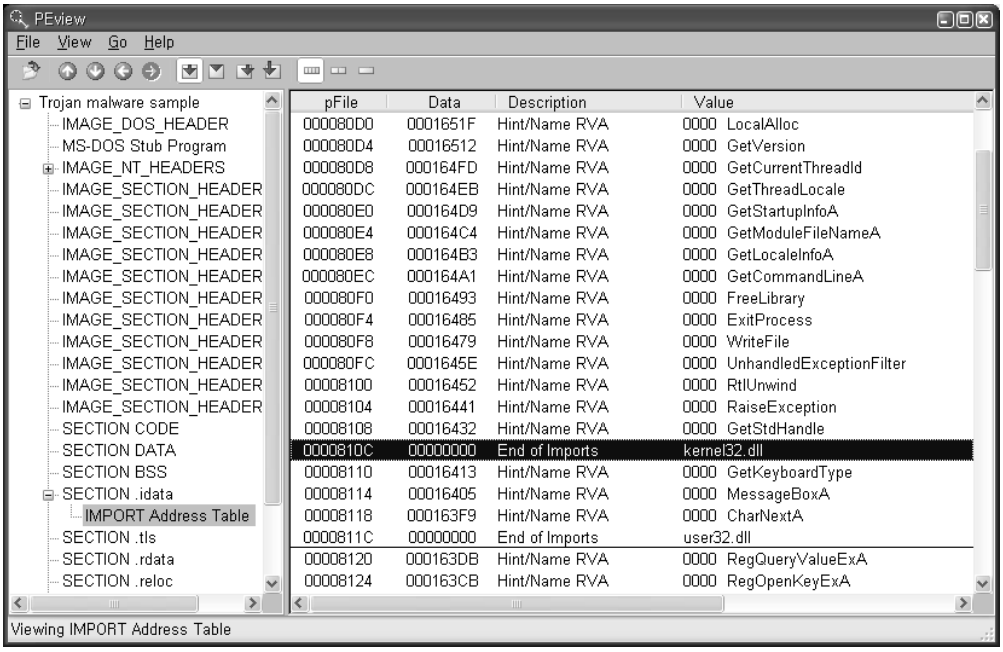
■ 단계 3.

화이트리스트의 API가 제거된 악성코드의 API 리스트와 비교 대상이 되는 API 리스트에서

동일하게 포함된 API를 분리한다. 즉, 양쪽 API 리스트에서 동일하게 포함된 API의 순서를 특징으로 하여 두 악성코드의 유사성을 계산한다.

4.2. API 추출

본 논문에서 악성코드로부터 API 리스트를 추출하는 방법은 2장의 배경지식에서 언급한 바와 같이 정적 분석을 통해 IAT를 찾고 내부에 포함된 API 이름들을 추출한다. 최근에는 이러한 방법을 자동으로 찾아서 보여주는 PE 분석 도구들이 존재하며, [그림 2]는 PEview라는 PE 분석 도구를 이용하여 IAT에 포함된 API를 조회한 결과이다.



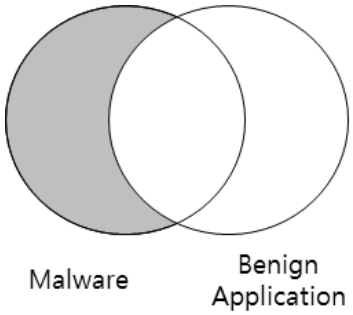
[그림 2] PEview를 이용한 IAT 조회

[Fig. 2] IAT Check using PEview

4.3. 화이트리스트 생성

본 논문에서 제안하는 방법은 유사한 기능을 가진 프로그램, 즉 악성코드 변종의 경우 악성코드가 실행될 때 호출하여 사용하는 API의 종류나 IAT에 포함된 해당 API들의 순서가 유사할 것이라는 가정에서 출발한다. 그러나 악성코드를 실행함에 있어서 호출되는 API들은 정상적인 응용프로그램에서도 호출될 수 있다. [그림 3]에 나타낸 것처럼 악성코드나 정상 응용프로그램의 기능이 비슷할 경우 실행을 위해 호출되기 위한 API들이 IAT에 동일하게 포함될 수 있다. 예를 들면, [표 2]와 같이 정상 응용프로그램 Groove와 악성코드 Trojan-Spy.Zapchast.f 및 Trojan-DDoS.Smurf.c에

는 GetProcAddress, GetCurrentProcess, TerminateProcess가 IAT에 공통적으로 포함되어 있는 것을 볼 수 있다. 따라서 제안하는 방법에서는 이러한 공통적인 특성으로 인한 오판(False Positive)을 줄이기 위해 여러 가지 정상 프로그램에서 자주 사용되는 API들을 화이트리스트로 작성함으로써 [그림 4]와 같이 필터링에 이용하고, 비교 대상이 되는 악성코드 API 리스트 사이의 유사도를 계산한다.



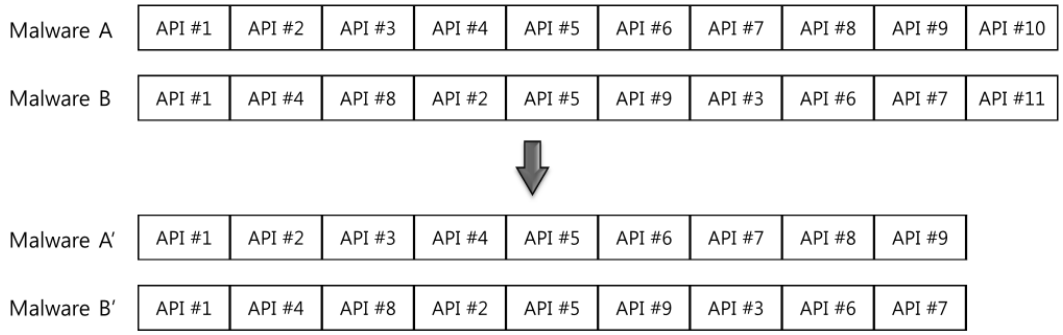
[그림 3] 악성코드와 정상 응용프로그램에서 사용되는 API
[Fig. 3] APIs used in Malware and Benign Application

[표 2] 정상 응용프로그램과 악성코드의 IAT에 포함된 API의 예
[Table 2] The Example APIs Included in Benign Application and Malware

<table>
<tr><th></th><th>정상 응용프로그램(Groove)</th><th>Trojan-Spy.Zapchast.f</th><th>Trojan-DDoS.Smurf.c</th></tr>
<tr><td rowspan="4">APIs</td><td colspan="2">DeleteCriticalSection, InitializeCriticalSection,
CloseHandle, GetCurrentThreadId</td><td rowspan="2">CreateFileA,
CreateFileMappingA,
GetCurrentProcessId,
GetEnvironmentVariableA,
(중략)
MapViewOfFile,
UnmapViewOfFile</td></tr>
<tr><td rowspan="2">CharNextW, CharPrevW,
CoCreateInstance, Sleep,
CreateProcess,
VirtualProtect,
GetCommandLineW,
GetCurrentProcessId,
GetModuleFileNameW,
(중략)
SetUnhandledExceptionFilter,
UnregisterClassA,
WaitForSingleObject</td><td>CallNextHookEx,
CreateWindowExA,
DefWindowProcA,
EnterCriticalSection,
(중략)
TlsAlloc, TlsFree,
TlsGetValue, TlsSetValue,
UnhookWindowsHookEx</td></tr>
<tr><td colspan="2">ExitProcess, FlushFileBuffers, FreeEnvironmentStringsA,
FreeEnvironmentStringsW, GetACP, GetCommandLineA,
(중략)
LoadLibraryA, MultiByteToWideChar, RtlUnwind,
SetFilePointer, SetHandleCount, SetStdHandle,
VirtualAlloc, VirtualFree, WideCharToMultiByte,
WriteFile</td></tr>
<tr><td colspan="3">GetProcAddress, GetCurrentProcess, TerminateProcess</td></tr>
</table>

4.4. 유사도 계산

악성코드의 IAT에 포함된 API 리스트에서 화이트리스트 API들을 제거하고 나면 비교 대상이 되는 두 개 악성코드의 API 유사도 계산을 위해 API들의 교집합을 추출한다. 본 논문에서는 단순히 교집합을 추출하여 유사도를 계산하는 것이 목적이 아니라 해당 API가 나타는 순차적 특징을 이용하여 유사성 계산하는 방법을 제안하는 것이므로, [그림 4]와 같이 기존의 순서를 유지하여 교집합을 추출한다.



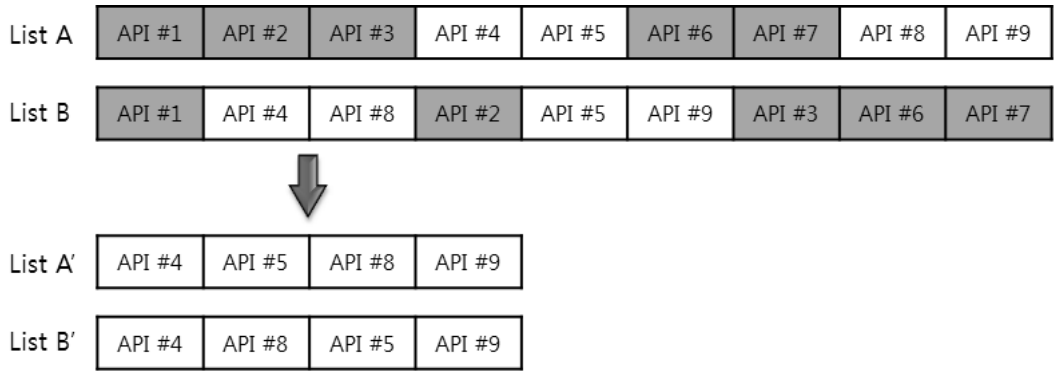
[그림 4] 두 개의 악성코드 API 리스트에서 교집합 추출

[Fig. 4] The Intersection Extraction from 2 Malware API lists.

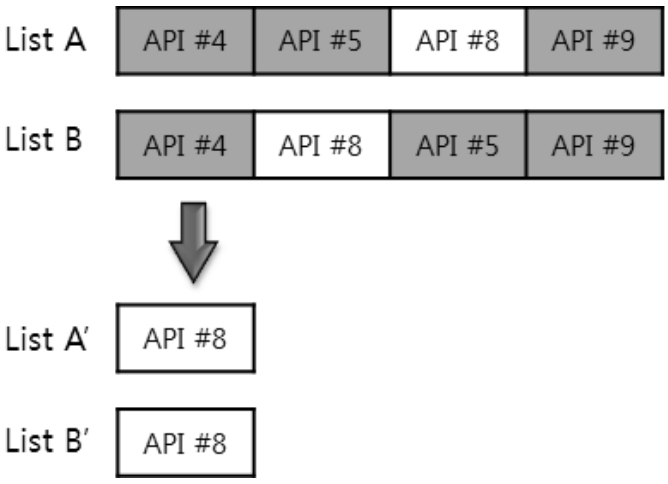
다음으로 추출된 교집합으로부터 순차적 특징을 유지하여 원본 API 리스트의 부분집합 리스트를 생성하며, 이를 위한 알고리즘은 다음과 같다.

```
Algorithm: SumOfSameSequence
1.  SumOfSameSequence(String List A, B)
2.  i, j, pre_i, pre_j, next_i, next_j ← 0
3.  new String List A', B'
4.  if( A.length == 1)
5.      return 0
6.  end if
7.
8.  while( i < A.length )
9.      while( j < B.length)
10.         if( A[i] == B[j] )
11.             i ← i+1
12.             pre_i ← i
13.             j ← j+1
14.             while( pre_j < j)
15.                 B'[next_j] ← B[pre_j]
16.                 next_j ← next_j+1
17.                 pre_j ← pre_j+1
18.             end while
19.             pre_j ← j
20.             go to line 4
21.         end if
22.         else
23.             j ← j+1
24.         end else
25.     end while
26.     A'[next_i] ← A[i]
27.     i ← i+1
28.     j ← pre_j
29.     next_i ← next_i+1
30. end while
31.
32. count ← A.length -A'.length
33.
34. if(count == 1)
35.     return SumOfSameSequence(A',B')
36. end if
37. else
38.     return count + SumOfSameSequence(A',B')
39. end else
```

제안한 알고리즘에서는 먼저 두 개의 악성코드 API 리스트의 순차적 특성을 파악하기 위해 리스트 A를 기준으로 동일한 순서로 나타나는 API들을 원본 리스트에서 제거한다. [그림 4]에서 보인 API 리스트를 그 예로 설명하면, 동일한 순서로 나타나는 API들을 리스트 A와 리스트 B에서 모두 제거하고 그 개수를 카운트한다. API#4와 API#5는 API#3이후로 나타났지만 리스트 B에서는 그보다 이전에 나타났으므로 리스트에 남게 되고 그 다음 순서로 나타난 API#6과 API#7을 리스트에서 제거하게 된다. 새로운 리스트 A'과 B'에 제거되지 않는 API를 기록하는 과정으로 표현하면 다음과 같다. [그림 5]에서는 동일한 순서로 나타난 API를 제거한 리스트 A', B'를 생성하고, [그림 6]과 같이 생성된 리스트에 동일한 알고리즘을 재귀적으로 적용하여 전체 교집합 리스트에서 동일한 순서로 나타난 API의 총 합을 계산한다.



[그림 5] API 리스트에 알고리즘을 적용한 결과
[Fig. 5] Applied Result of Algorithm for API Lists



[그림 6] 재귀적으로 알고리즘을 적용한 결과
[Fig. 6] Recursively Applied Result of Algorithm

제안한 알고리즘을 통해 동일한 순서를 가진 API 하위 집합의 합을 계산한 후, 하위 집합의 크기와 전체 API 리스트의 합집합과의 비율을 계산함으로써 유사도를 판단한다. 이를 유사도 계산 수식으로 표현하면 수식 (1)과 같다.

$$P_{(A,B)} = \frac{n(\Sigma S(A,B))}{n(A \cup B)} \quad (1)$$

※ X : API List Set / n(X) : # of API List Set X / S(X,Y) : Same Sequence Subset of X,Y

제안한 방법은 단순히 교집합으로 유사도를 계산하는 것이 아니라, 동일하게 나타나는 API의 순차적 특징을 반영함으로써 더 정확한 유사도 계산 결과를 기대할 수 있다. 또한 부분적인 합을 이용하지만 LCSS(Longest Common Sub-Sequence)가 아닌 동일한 순서로 나타나는 API를 전부 추출하여 유사도 계산에 사용하기 때문에 API의 순서 변경으로 생성된 악성코드의 변종을 동일한 악성코드로 분류할 수 있다.

5. 실험 및 결과

5.1. 실험 환경 및 데이터

본 논문에서 제안한 방법 및 알고리즘을 JAVA 개발환경인 Eclipse를 이용하여 구현하였다. 또한 실험을 위해서는 VX Heavens[21]으로부터 악성코드 샘플을 선택적으로 수집하였다. 수집한 악성코드는 윈도우 상에서 실행될 수 있는 트로이목마(Trojan)이며, 기능적으로 분류했을 때 각각 Trojan-DDoS 125개와 Trojan-Spy 420개이다. 각 악성코드 샘플의 진단명은 Kaspersky 안티바이러스의 진단명을 따른다. 수집한 악성코드 샘플들로부터 API 리스트를 추출하고, 구현한 프로그램을 통해 화이트리스트를 적용한 API 리스트의 순차적 특징에 대하여 유사도를 계산하였다.

5.2. 화이트리스트

본 논문에서 제안한 방법은 보다 정확한 유사도 계산 및 오버헤드 감소를 위해 정상 응용프로그램들에서 나타날 수 있는 API 리스트를 정리하여 화이트리스트로 작성한다. [표 3]은 실험에 앞서 화이트리스트를 추출하기 위해 사용한 정상 프로그램들과 이를 통해 작성된 화이트리스트이다. 이때 화이트리스트 내에 존재하는 API는 정상 응용프로그램들에서 5번 이상 IAT에 포함된 것들만 필터링하였다.

[표 3] 정상 응용프로그램 기반의 화이트리스트 작성
[Table 3] White-list Extraction based on Benign Applications

정상 프로그램	화이트리스트
Mspaint, NateOn, Notepad, Hwp, Calcul, Explorer, iTunes, uTorrent, Excel, Groove, AlZip, AlFtp, Everything, Acrobat	BitBlt, CloseHandle, CoCreateInstance, CoTaskMemFree, DeleteCriticalSection, DeleteDC, DeleteObject, EnableWindow, EnterCriticalSection, exit, FindClose, FreeLibrary, GetACP, GetClientRect, GetCurrentProcess, GetCurrentProcessId, GetCurrentThreadId, GetCursorPos, GetDC, GetDeviceCaps, GetFocus, GetLastError, GetModuleFileNameW, GetModuleHandleA, GetModuleHandleW, GetParent, GetProcAddress, GetProcessHeap, GetSubMenu, GetSystemMenu, GetSystemMetrics, GetSystemTimeAsFileTime, GetTickCount, GetVersionExW, GlobalAlloc, GlobalFree, GlobalLock, GlobalUnlock, HeapAlloc, HeapFree, InitializeCriticalSection, InterlockedCompareExchange, InterlockedDecrement, InterlockedExchange, InterlockedIncrement, InvalidateRect, IsWindowVisible, LeaveCriticalSection, LoadLibraryA, LoadLibraryW, LocalAlloc, LocalFree, lstrlenW, memcpy, MessageBoxW, MulDiv, MultiByteToWideChar, PostQuitMessage, QueryPerformanceCounter, RaiseException, ReadFile, RegCloseKey, RegOpenKeyExW, RegQueryValueExW, ReleaseDC, ScreenToClient, SelectObject, SetActiveWindow, SetCursor, SetErrorMode, SetForegroundWindow, SetMapMode, SetUnhandledExceptionFilter, SetViewportExtEx, SetWindowPos, Sleep, TerminateProcess, UnhandledExceptionFilter, UpdateWindow, WideCharToMultiByte, WriteFile, SetLastError, VirtualAlloc, VirtualFree, GetCommandLineW

5.3. 악성코드 유사도 계산 결과

실험에서는 알고리즘을 구현한 프로그램을 이용하여 Trojan-DDoS와 Trojan-Spy 각각의 샘플들에 대하여 유사도를 계산하였다. 실험 방법은 같은 패밀리 내에 포함되는 악성코드 샘플들의 유사도를 계산하고, 다른 패밀리 내의 악성코드 샘플들 사이의 유사도를 계산한다.

[표 4]는 Trojan-Spy의 각 Zapchast, GhostSpy, PCSpy 패밀리 샘플들에 대하여 유사도를 계산한 결과를 나타낸 것이다. 같은 패밀리 내에 포함된 샘플들 사이의 유사도는 약 0.4 이상이며, 같은 Trojan-Spy라 하더라도 다른 패밀리 내에 포함된 샘플들 사이의 유사도는 그 이하로 나타나는 것을 확인할 수 있다.

[표 4] Trojan-Spy 악성코드 샘플들 사이의 유사도
[Table 4] Similarity between Trojan-Spy Samples

			Trojan-Spy								
			.Zapchast				.GhostSpy		..PCSpy		
			.b	.f	.g	.i	.52	.40	.b	.c	.d
Trojan -Spy	.Zapchast	.b	**1.000**	**0.414**	**0.414**	**0.414**	0.171	0.207	0.071	0.060	0.065
		.f	**0.414**	**1.000**	**0.414**	**0.414**	0.161	0.232	0.057	0.070	0.068
		.g	**0.414**	**0.414**	**1.000**	**0.414**	0.161	0.232	0.057	0.070	0.068
		.i	**0.414**	**0.414**	**0.414**	**1.000**	0.161	0.232	0.057	0.070	0.068
	.GhostSpy	.52	0.171	0.161	0.161	0.161	**1.000**	**0.488**	0.194	0.172	0.177
		.40	0.207	0.232	0.232	0.232	**0.488**	**1.000**	0.187	0.111	0.115
	..PCSpy	.b	0.071	0.057	0.057	0.057	0.194	0.187	**1.000**	**0.404**	**0.414**
		.c	0.060	0.070	0.070	0.070	0.172	0.111	**0.404**	**1.000**	**0.965**
		.d	0.065	0.068	0.068	0.068	0.177	0.115	**0.414**	**0.965**	**1.000**

[표 5]는 Trojan-DDoS의 각 Desex, Delf, Boxed 패밀리 샘플들에 대한 유사도 계산 결과를 나타낸 것이다. Desex 및 Boxed의 경우 높은 유사도 계산 값이 나타났으나, Delf의 경우 그 값이 비교적 작게 나타났다. 이는 악성코드 제작자가 해당 악성코드의 변종을 제작하면서 동일한 기능을 수행하는 다른 API를 사용하도록 악성코드를 업그레이드 한 것으로 판단된다.

[표 5] Trojan-DDoS 악성코드 샘플들 사이의 유사도
[Table 5] Similarity between Trojan-DDoS Samples

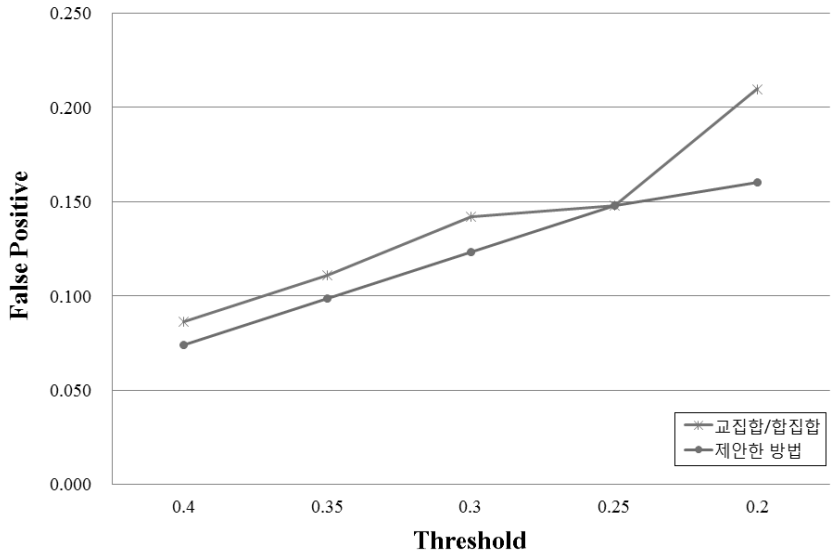
			Trojan-DDoS					
			.Desex		.Delf		.Boxed	
			.a	.b	.e	.i	.a	.j
Troja n-DD oS	.Desex	.a	**1.000**	**0.625**	0.118	0.032	0.042	0.036
		.b	**0.625**	**1.000**	0.056	0.000	0.041	0.071
	.Delf	.e	0.118	0.056	**1.000**	***0.278***	0.130	0.120
		.i	0.032	0.000	***0.278***	**1.000**	0.117	0.120
	.Boxed	.a	0.042	0.041	0.130	0.117	**1.000**	**0.811**
		.j	0.036	0.071	0.120	0.120	**0.811**	**1.000**

[표 6]은 Trojan-Spy와 Trojan-DDoS 사이의 각 악성코드 샘플들에 대한 유사도 계산 결과이다. Trojan-DDoS.Delf를 제외하고 같은 패밀리 내에 포함되는 악성코드 샘플들 사이의 유사도는 0.4 이상을 유지하고 클래스 및 패밀리가 다른 경우 유사도 값이 더 작다. 여기서 Trojan-Spy.GhostSpy와 Trojan-DDoS.Delf, 그리고 Trojan-Spy.Zapchast와 Trojan-DDoS.Boxed 사이의 유사도가 높게 나타난 것을 볼 수 있다. 이는 클래스와 패밀리가 다르게 분류되어 있더라도 파일 및 레지스트리 조작, 프로세스 생성 또는 종료와 같이 악성코드가 감염된 시스템을 조작하는데 필요한 API가 유사하게 포함되었기 때문이라고 할 수 있다.

[표 6] Trojan-Spy 및 Trojan-DDoS 악성코드 샘플들 사이의 유사도
[Table 6] Similarity between Trojan-Spy and Trojan-DDoS Samples

			Trojan-Spy						Trojan-DDoS					
			.Zapchast		.GhostSpy		.PCSpy		.Desex		.Delf		.Boxed	
			.g	.i	.52	.40	.b	.c	.a	.b	.e	.i	.a	.j
Trojan -Spy	.Zapchast	.g	**1.000**	**0.414**	0.161	0.232	0.057	0.070	0.068	0.087	0.190	0.110	***0.328***	***0.295***
		.i	**0.414**	**1.000**	0.161	0.232	0.057	0.070	0.068	0.087	0.190	0.110	***0.328***	***0.295***
	.GhostSpy	.52	0.161	0.161	**1.000**	**0.488**	0.194	0.172	0.045	0.056	0.214	***0.400***	0.092	0.103
		.40	0.232	0.232	**0.488**	**1.000**	0.187	0.111	0.091	0.089	***0.396***	***0.342***	0.100	0.093
	.PCSpy	.b	0.057	0.057	0.194	0.187	**1.000**	**0.404**	0.032	0.016	0.140	0.173	0.050	0.048
		.c	0.070	0.070	0.172	0.111	**0.404**	**1.000**	0.014	0.014	0.069	0.150	0.036	0.041
Trojan -DDoS	.Desex	.a	0.068	0.068	0.045	0.091	0.032	0.014	**1.000**	**0.625**	0.118	0.032	0.042	0.036
		.b	0.087	0.087	0.056	0.089	0.016	0.014	**0.625**	**1.000**	0.056	0.000	0.041	0.071
	.Delf	.e	0.190	0.190	0.214	***0.396***	0.140	0.069	0.118	0.056	**1.000**	***0.278***	0.130	0.120
		.i	0.110	0.110	***0.400***	***0.342***	0.173	0.150	0.032	0.000	***0.278***	**1.000**	0.117	0.120
	.Boxed	.a	***0.328***	***0.328***	0.092	0.100	0.050	0.036	0.042	0.041	0.130	0.117	**1.000**	**0.811**
		.j	***0.295***	***0.295***	0.103	0.093	0.048	0.041	0.036	0.071	0.120	0.120	**0.811**	**1.000**

[그림 7]은 악성코드의 IAT에 포함된 API들을 단순히 합집합에 대한 교집합 비율로 유사도를 계산한 결과와, 본 논문에서 제안한 방법을 통해 유사도를 계산한 결과에 대하여 임계치(Threshold) 설정 값에 따라 발생하는 긍정 오류(False Positive) 비율을 비교한 그래프이다. 임계치를 0.4로 설정할 경우 제안한 방법에서 나타나는 긍정 오류 비율은 0.074로, 이는 합집합에 대한 교집합 비율로 계산했을 경우보다 낮게 나타나는 것을 확인할 수 있다.



[그림 7] False Positive 비율 비교
[Fig. 7] False Positive Rate

5. 결론 및 향후 연구

본 논문에서는 악성코드 PE 파일의 IAT에 포함된 API 리스트의 순차적 특징을 이용하여 악성코드 변종을 분류할 수 있는 방법을 제안하였다. 제안한 방법은 정상 응용프로그램들에서 나타날 수 있는 API 리스트를 화이트리스트로 정리하고, 악성코드 간의 유사도를 계산할 때 이를 제거함으로써 오버헤드 감소 및 보다 정확한 유사도를 계산한다. 제안한 방법 및 유사도 계산 알고리즘을 구현하고, 수집한 Trojan-Spy 및 Trojan-DDoS 악성코드 샘플들에 대하여 실험을 진행하였다. 그 결과 같은 패밀리 내에 포함되는 샘플들 사이의 유사도는 약 0.4 이상의 유사도를 보였으며, 임계치를 0.4~0.2 사이로 설정했을 때의 긍정 오류 비율을 계산한 결과는 0.074~0.160 사이의 값을 보였다. 그러나 모든 클래스의 악성코드가 아닌 2가지 클래스의 트로이목마 악성코드 샘플을 대상으로 실험을 진행하였기 때문에 향후 다양한 클래스의 악성코드 샘플들을 대상으로 제안한 방법을 적용하여 알고리즘을 보완함으로써 보다 개선된 악성코드 분류 방법을 연구하고자 한다.

참고문헌

[1] D. Barroso, “Botnets-The Silent Threat”, ENSIA Position Paper, no. 3, pp. 1-9, November 2007.

[2] 한경수, 임광혁, 임을규, “허니넷을 이용한 P2P 기반 Storm 봇넷의 트래픽 분석”, 정보보호학회논문지, 제19권 제4호, pp. 51-61, 2009년 8월.

[3] Charles Petzold, Programming Microsoft Windows 5th Edition, Microsoft Press, 2002.

[4] 강태우, 조재익, 정만현, 문종섭, “API call의 단계별 복합분석을 통한 악성코드 탐지”, 정보보호학회논문지, 제17권 제6호, pp. 89-98, 2007년 12월.

[5] 권오철, 배성재, 조재익, 문종섭, “Native API 빈도 기반의 퍼지 군집화를 이용한 악성코드 재그룹화 기법연구”, 정보보호학회논문지, 제18권 제6호, pp. 115-127, 2008년 12월.

[6] Greg Hoglund, James Butler, Rootkits: Subverting the Windows Kernel, Pearson Education, Inc, 2006.

[7] 김완경, 소우영, “윈도우 XP 커널 기반 API 후킹 탐지 도구 설계 및 개발”, 보안공학연구논문지, 제7권 제4호, pp. 385-397, 2010년 8월.

[8] 김성우, 해킹 파괴의 광학 개정판, 와이미디어, 2006.

[9] 한경수, 신윤호, 임을규, “스팸메일로 전파되는 악성코드의 분석 및 대응 프레임워크”, 보안공학연구논문지, 제7권 제4호, pp. 363-383, 2010년 8월.

[10] 박남열, 김용민, 노봉남, “우회기법을 이용하는 악성코드 행위기반 탐지 방법”, 정보보호학회논문지, 제16권 제3호, pp. 17-28, 2006년 6월.

[11] M. Fredrikson, S. Jha, M. Christodorescu, R. Sailer, and X. Yan, “Synthesizing Near-Optimal Malware Specifications from Suspicious Behaviors”, Proceedings of the 2010 IEEE Symposium on Security and Privacy, pp. 45-60, November 2010.

[12] Q. Miao, Y. Wang, Y. Cao, X. Zhang, and Z. Liu, “APICapture - a Tool for Monitoring the Behavior of Malware”, Proceedings of the 3rd International Conference on Advanced Computer Theory and Engineering, pp. 390-394, August 2010

[13] V. P. Nair, H. Jain, Y. K. Golecha, M. S. Gaur, and V. Laxmi, “MEDUSA: MEtamorphic malware Dynamic analysis Using Signature from API”, Proceedings of the 3rd International Conference on Security of Information and Networks, September 2010.

[14] J. Lee, K. Jeong, and H. Lee, “Detecting Metamorphic Malwares using Code Graphs”, Proceedings of the 2010 ACM Symposium on Applied Computing, pp. 1970-1977, March 2010.

[15] S. Cesare, and Y. Xiang, “A Fast Flowgraph Based Classification System for Packed and Polymorphic Malware on the Endhost”, Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications, pp. 721-728, April 2010.

[16] K. Kim, and B. Moon, “Malware Detection based on Dependency Graph using Hybrid Genetic Algorithm”, Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 1211-1218, July 2010.

[17] Q. Zhang, and D. S. Reeves, “MetaAware: Identifying Metamorphic Malware”, Proceedings of the 23rd Annual Computer Security Applications Conference, pp. 411-420, September 2007.

[18] A. Karnik, S. Goswami, and R. Guha, “Detecting Obfuscated Viruses Using Cosine Similarity Analysis”, Proceedings of the 1th Asia International Conference on Modelling & Simulation, pp. 165-170, March 2007.

[19] SK. Cha, I. Moraru, J. Jang, J. Truelove, D. Brumley, and D. G. Andersen, “SplitScreen: Enabling Efficient, Distributed Malware Detection”, Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, April 2010.

[20] ClamAV, http://www.clamav.net/

[21] VX Heavens, http://vx.netlux.org/

저자 소개

한경수 (Kyoung-Soo Han)

2008년 상지대학교 컴퓨터정보공학부 학사
2010년 한양대학교 전자컴퓨터통신공학과 석사
2011년 4월 현재 한양대학교 전자컴퓨터통신공학과 박사과정
관심분야 : 악성코드 분석, 네트워크 보안, 정보보호

김인경 (In-Kyoung Kim)

2010년 한양대학교 컴퓨터공학부 학사
2011년 4월 현재 한양대학교 전자컴퓨터통신공학과 석사과정
관심분야 : 악성코드 분석, 네트워크 보안, 정보보호

임을규 (Eul-Gyu Im)

1992년 서울대학교 컴퓨터공학과 학사
1994년 서울대학교 컴퓨터공학과 석사
2002년 University of Southern California 컴퓨터과학과 박사
2011년 4월 현재 한양대학교 컴퓨터공학부 조교수
관심분야 : 네트워크 보안, 악성 프로그램 분석, RFID 보안, SCADA 보안