Journal of the Korea Academia-Industrial cooperation Society
Vol. 16, No. 10 pp. 7211-7218, 2015

http://dx.doi.org/10.5762/KAIS.2015.16.10.7211
ISSN 1975-4701 / eISSN 2288-4688

# API 통계 기반의 워드 클라우드를 이용한 악성코드 분석 기법

유성태[1], 오수현[1*]
[1]호서대학교 정보보호학과

# Malware Analysis Mechanism using the Word Cloud based on API Statistics

**Sung-Tae Yu[1], Soo-Hyun Oh[1*]**
**[1]Dept. of Information Security, Hoseo University**

**요 약** 악성코드는 하루 평균 수만 건 이상이 발생하고 있으며, 신종 악성코드의 수는 해마다 큰 폭으로 증가하고 있다. 악성코드를 탐지하는 방법은 시그니쳐 기반, API 흐름, 문자열 등을 이용한 다양한 기법이 존재하지만 대부분의 탐지 기법들은 악성코드를 우회하는 공격 기법으로 인해 신종 악성코드를 탐지하는데 한계가 있다. 따라서 신종 악성코드를 효율적으로 탐지하기 위한 연구가 많이 진행되고 있다. 그중 시각화 기법을 통한 연구가 최근 활발하게 이루어지고 있으며, 악성코드를 직관적으로 파악할 수 있으므로 대량의 악성코드를 효율적으로 탐지하고 분석할 수 있다는 장점이 있다. 본 논문에서는 악성코드와 정상파일에서 Native API 함수를 추출하고 해당 Native API가 악성코드에서 발생하는 확률에 따라서 F-measure 실험을 통해 가중치의 합을 결정하고, 최종적으로 가중치를 이용하여 워드 클라우드에서 텍스트의 크기로 표현되는 기법을 제안한다. 그리고 실험을 통해 악성코드와 정상파일에서 사용하는 Native API의 가중치에 따라서 악성코드를 판단할 수 있음을 보인다. 제안하는 방식은 워드 클라우드를 이용하여 Native API를 시각적으로 표현함으로써 파일의 악성 유무를 판단하고, 직관적으로 악성코드의 행위를 분석할 수 있다는 장점이 있다.

**Abstract** Tens of thousands of malicious codes are generated on average in a day. New types of malicious codes are surging each year. Diverse methods are used to detect such codes including those based on signature, API flow, strings, etc. But most of them are limited in detecting new malicious codes due to bypass techniques. Therefore, a lot of researches have been performed for more efficient detection of malicious codes. Of them, visualization technique is one of the most actively researched areas these days. Since the method enables more intuitive recognition of malicious codes, it is useful in detecting and examining a large number of malicious codes efficiently. In this paper, we analyze the relationships between malicious codes and Native API functions. Also, by applying the word cloud with text mining technique, major Native APIs of malicious codes are visualized to assess their maliciousness. The proposed malicious code analysis method would be helpful in intuitively probing behaviors of malware.

**Keywords :** Malware, Malware Analysis, Native API, Visualization, Word cloud

## 1. 서론

최근 들어 매우 다양하고 진화된 공격 방식을 사용하는 악성코드들이 빈번하게 발생하고 있다. 독일의 백신 테스트 연구소인 av-test에 따르면 현재까지 발생한 악성코드는 약 3억 4천만 개에 달하며, 하루 평균 47만개의 신종/변종 악성코드가 발생되는 것으로 나타났다[1]. 2014년에 발생한 대표적인 악성코드 유형을 살펴보면

"이 논문은 2014년도 호서대학교의 재원으로 학술연구비 지원을 받아 수행된 연구임"(2014-0025)
*Corresponding Author : Soo-Hyun Oh(Hoseo Univ.)
Tel: +82-41-540-5716 email: shoh@hoseo.edu
Received July 10, 2015 Revised (1st August 19, 2015, 2nd September 11, 2015)
Accepted October 8, 2015 Published October 31, 2015

개인정보 유출형, 금전 탈취형, 파일 암호화형으로 나눌 수 있으며, 특히 컴퓨터의 모든 파일들을 강제로 암호화하고 돈을 요구하는 새로운 악성코드인 랜섬 웨어가 최근에 발생했다. 과거 PC에 집중되어 있던 악성코드는 스마트폰으로 확대되고 있다. 안랩에 따르면 스마트폰 악성코드는 2014년 전년대비 14.2% 증가한 누적 총 143만 247개로 나타났다[2]. 증가하고 있는 악성코드에 대응하기 위해 다양한 연구가 진행되고 있으나 대량으로 발생하는 신종 악성코드와 변종 악성코드를 구분하는 데는 한계점이 존재한다. 일반적인 악성코드 탐지 및 분류 방법은 다음과 같다[3]. 첫째, 시그니쳐 기반의 패턴 매칭 기법으로, 일반적으로 문자열의 해쉬값을 비교해 진단하고 있으나 시**그니쳐가** 조금만 변형되어도 탐지가 불가능 하다는 단점이 있다. 또한 패킹이나 난독화 기법이 적용된 악성코드는 탐지가 매우 어렵다는 문제점이 있다. 둘째, 문자열 및 바이트 시퀀스 패턴 탐지 기법으로, 악성코드내의 문자열 및 바이트의 시퀀스만을 파악하여 분석할 수 있으나, 소스 코드의 변화에 매우 취약하다는 단점이 있다. 셋째, API 호출 관계를 나타낸 그래프를 비교하는 방법으로, 악성코드가 사용하는 API들 사이의 호출 관계를 도식화하여 변종을 구분해낼 수 있다. 그러나 코드의 변형에 따라 그래프가 쉽게 바뀔 수 있으며 그래프의 복잡성 때문에 직관적인 판단이 어렵다는 단점이 있다. 이러한 한계점을 극복하기 위해서 악성코드 탐지 및 분류에 관한 연구는 꾸준히 지속되고 있다. 본 논문에서는 정상 파일과 악성코드의 대표 Native API를 텍스트 마이닝의 워드 클라우드 기법을 적용하여 직관적으로 악성코드를 판단하고, 악성코드의 악성 행위를 예측할 수 있는 방법을 제안한다.

## 2. 관련 연구

### 2.1 악성코드 시각화 분석 기법

수많은 악성코드를 신종과 변종으로 분류할 수 있는 방법에 대한 연구는 과거부터 현재까지 활발하게 이루어지고 있다. 그러나 하루 평균 수십만건의 악성코드를 분석하여 신종과 변종으로 분류하는 작업은 매우 방대하여 큰 어려움이 따른다. 하지만 악성코드들의 특징을 시각적으로 표현함으로써 직관적으로 분석해 낼 수 있다면 기존의 어려움을 극복해 낼 수 있을 것이다. 따라서 악성코드의 시각화 분석 기법은 악성코드 분석가들에게 중요한 연구 주제가 되었다.

악성코드 분석 기법으로 많이 사용하는 CFG (Control Flow Graph)기법은 코드의 실행 흐름과 구조를 그래프로 시각적으로 표현한다[4]. 이렇게 표현한 CFG는 바이러스의 계놈지도와 같은 용도로 활용되어 바이러스의 변형을 탐지하는데 활용할 수 있다[5]. Fig.1은 [5]에서 실제 악성코드를 CFG로 표현한 그림으로 악성코드가 호출하는 함수들과 함수들 사이의 호출관계가 나타나 있다. 일한 악성코드 패밀리의 경우 CFG로 나타내면 변형 정도에 따라 매우 유사한 그래프를 얻을 수 있다.

Fig. 1. CFG, Virus Genome Map

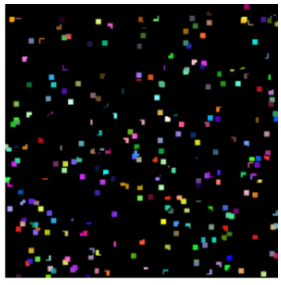

Fig. 2. Image Matrix

Fig.2는 이미지 매트릭스 기법으로 악성코드의 유사도를 계산하는 방식이다[6]. 이미지 매트릭스는 어셈블리 코드를 추출해 JMP 또는 RET와 같은 분기 명령어로 분할하고 분할된 블록으로 Opcode 명령어 시퀀스를 만든다. Opcode 명령어 시퀀스를 해쉬 함수를 이용하여 문자열을 만들고 박스의 위치와 컬러를 결정한다. 이렇게 만들어진 이미지 매트릭스는 데이터베이스에 저장되고 새로운 악성코드의 이미지 매트릭스에 대해 유사도를 계산하여 유사도가 임계값을 초과하면 특정 악성코드 패밀리의 변종으로 판단하고, 그렇지 않은 경우에는 알려지지 않은 악성코드로 간주한다.

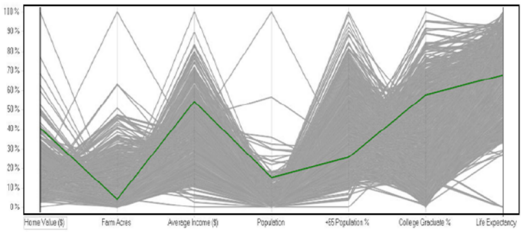

Fig. 3. An example of using Parallel Coordinates

Fig.3은 악성코드의 속성을 평행좌표계parallel coordinates)로 나타내어 악성코드를 분석하고 분류하는 기법의 예이다[7]. 이 방식에서는 백신 업체에서 제공하는 다양한 속성들을 정의하며, 속성에는 악성코드 종류, 플랫폼, 감염/설치 경로, 파일형태, 발견일, 대표적 증상 등이 있다.

정의한 속성에 순차적으로 이진형태의 값을 부여한 후, 수치화한 데이터를 평행좌표계로 나타낸 후 악성코드의 속성을 분석하고 분류한다. 기존 악성코드 시각화 분석 기법들의 장점과 단점은 Table 1과 같다.

Table 1. Analysis of related works

| | Feature | Weakness |
|---|---|---|
| CFG | • Visualization of function call relations<br>• Inference of malware behaviors | • Difficulty with decision of malware and variants |
| Image Matrix | • Decision of malware variants using the computation of similarity of image matrix | • Need a numerous malware DB<br>• • Difficulty with decision of malware behaviors |
| Parallel coordinates | • Analysis of malware using attributes | • Difficulty with decision of specific malware behaviors |

## 2.2 API 기반 악성코드 분석 기법

악성코드는 파일 생성, 프로세스 생성, 레지스트리 정보 등록, 통신 채널 생성 등 악성행위를 하기 위해 다양한 API를 호출한다. 따라서 API를 이용한 악성코드의 분석 기법은 매우 널리 알려져 있다. 하지만 최근에는 API 기반의 악성코드 분석 기법을 우회하기 위해 사용하지 않는 API를 추가하는 등 공격자들의 노력 또한 진화하고 있다.

Fig.4는 유전자 염기 서열 분석기법을 악성코드 분석에 적용한 연구 결과를 나타낸 것이다[3]. 이 방식에서는 실험 데이터로 1000개의 악성코드에서 공통적으로 1회 이상 호출되지만, 정상 파일에서 사용 빈도가 낮은 23개의 Native API를 추출해 'A','C','G','T'와 같이 Native API와 문자를 1대 1로 대응 시키는 방법을 사용하였으며, 다중 서열 정렬을 통해 같은 패밀리들의 시스템 콜 시퀀스를 추출한다. 유사도 비교에는 Smith-Waterman 알고리즘을 사용한다.

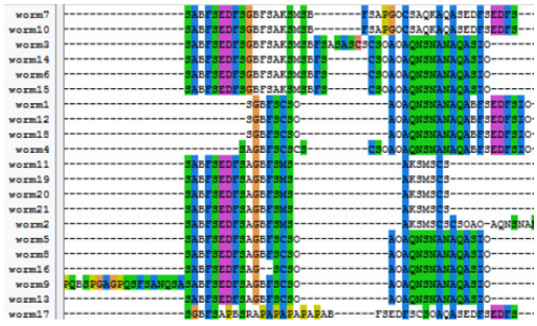

Fig. 4. System call sequences extracted using ClustalX

또한, PE 헤더에서 취약한 API의 종류들을 조사하고 정적분석 단계에서 취약한 API의 호출 빈도수를 분석하여, 동적분석 단계에서 프로세스, 포트, 레지스트리 등의 은닉 기능이 있는 Native API를 조사하는 방식이 있다[8]. Table 2는 [8]에서 프로세스, 포트, 레지스트리 등의 은닉 기능이 있는 Native API를 분류한 것이다. 이 방식에서는 조사한 값들을 판별함수에 넣고 사전에 정의한 취약한 API가 어느 정도로 나타는지를 계산하여 악성유무를 판별한다.

Table 2. Classification by concealing property of the Native API [8]

| function | Native API Call |
|---|---|
| Process Hiding | NtQuerySystemInformation |
| | NtQueryInformationFile |
| Hooking | NtOpenThread |
| | NtOpenProcess |
| Port Hiding | NtDeviceIoControlFile |
| Registry Hiding | NtEnumerateKey |
| | NtEnumerateValueKey |
| File/Directory Hiding | NtQueryDirectoryFile |
| Impersonation | NtImpersonateAnonymousToken |
| | NtImpersonateOfPort |
| Access Kernel Mode | NtSystemDebugControl |

### 2.3 텍스트 마이닝

텍스트 마이닝은 비정형화된 텍스트를 분석하여 해당 텍스트가 갖고 있는 패턴, 추세, 분포 등의 고급 정보들을 추출하여 가공하는 기술이다. 최근 들어 빅데이터가 보급되면서 대용량 텍스트 분석기술에 관한 관심이 증가하고 있으며 텍스트 마이닝 기술의 중요성이 강조되고 있다. 텍스트 마이닝은 기본적으로 비정형/반정형 데이터를 단순화된 모델로 나타낸다.

데이터 시각화 기법으로 많이 사용하는 워드 클라우드(word cloud)는 하나의 텍스트에 출현하는 단어의 빈도수에 비례하여 크기로 표출한 그래프이다. 이렇게 표현된 그래프는 경제적이고 효과적으로 분석 결과를 제공할 수 있다. 가장 인기있는 생성기로는 Wordle이 있으며, 공개된 웹상에서 누구나 쉽게 이용할 수 있다는 장점이 있다. 좀 더 전문적으로 나타내고자 한다면 빅데이터 분석 툴 R을 이용하여 워드 클라우드로 나타낼 수 있다.


Fig. 5. An example of a word cloud

## 3. 제안하는 악성코드 분석 기법

본 장에서는 악성코드를 탐지하고, 악성 행위를 직관적으로 판단하기 위해 악성코드와 Native API 함수들 사이의 관계를 분석하고, 텍스트 마이닝의 시각화 방법인 워드 클라우드를 적용하여 시각적으로 표현함으로써 파일의 악성 유무를 판단 할 수 있는 악성코드 분석 기법을 제안한다. 제안하는 분석기법은 API 리스트 추출, 가중치 적용, 시각화 등 총 3단계로 구성된다.

### 3.1 API 리스트 추출

API 리스트 추출 방법에는 바이너리 실행파일 자체에서 추출하는 방법과 동작 중 호출되는 API를 후킹하는 방법이 있다[9]. 바이너리 실행파일에서 API를 추출하기 위해서는 PE 파일을 분석하여 IAT(Import Address Table)에 포함된 API 리스트를 추출한다. 프로그램 동작 중 호출되는 API를 후킹하는 방법에는 대표적으로 SSDT(System Service Descriptor Table) 후킹과 IDT(Interrupt Descriptor Table) 후킹이 있다. SSDT 후킹은 커널에서 서비스를 받기 위해 필요한 테이블내 함수의 메모리 주소를 변경하거나 테이블을 리다이렉트 함으로써 API를 추출할 수 있고, IDT 후킹은 IDT내의 인터럽트 처리 경로를 변경하여 특정 인터럽트를 후킹함으로써 API를 추출할 수 있다. 최근 악성코드는 실제 사용하지 않는 API를 추가해 API 기반의 탐지 방법을 우회하는 노력을 하고 있다. 따라서 제안하는 분석 기법에서는 악성코드가 실제 동작하기 위해 호출하는 API만을 추출하기 위해 API 후킹 방법을 이용하여 악성코드가 사용하는 API에 대한 정보를 얻는다.

### 3.2 가중치 적용

악성코드는 악성 행위를 수행하기 위해 API 함수를 호출한다. 악성 행위를 위해 사용하는 API의 종류는 일반적으로 한정되어 있다. 따라서 악성코드가 사용하는 API를 알고 있다면 악성 행위를 보다 쉽게 파악할 수 있다. 본 논문에서 제안하는 기법에서는 악성코드별 대표 API와 일반적으로 사용하는 API에 대해 가중치를 차등적으로 적용한다. 가중치를 적용하기 위해서는 사전에 악성코드와 정상코드의 대표 Native API를 선정해야 한다. 대표 Native API를 선정하기 위한 실험 데이터 셋으로 200개 이상의 악성코드와 대중적으로 사용하는 20개 이상의 정상 파일을 대상으로 동적분석을 수행하여 Native API를 추출한다. 그리고 추출한 Native API별로 악성코드와 정상 파일에 대해 어느 정도의 확률로 존재하는지 확인한며, 확률 차이를 통해 얻은 결과는 악성코드의 대표 Native API의 Score로 사용한다. 해당 Score는 악성 유무와 정상 유무를 판단하기 위한 점수이며, F-measure 측정을 통해 얻은 임계값을 초과하면 악성으로 판단할 수 있다.

### 3.3 시각화

본 논문에서는 데이터 시각화 방법인 워드 클라우드 기능을 이용하여 정형화시킨 API를 시각화 한다. 워드

클라우드는 빈도수에 따라 단어의 크기를 결정한다. 가중치 적용 단계에서 결정된 Score는 API의 크기를 결정하는 요소로 파일에서 악성코드의 대표 Native API가 발견되면 워드 클라우드는 API를 크게 표시한다. 따라서 표시된 API를 통해 악성 유무를 판단할 수 있다. 제안하는 기법에서는 3단계로 악성코드의 API를 정형화하고 직관적으로 판단할 수 있도록 시각적으로 나타내는 방법을 사용하였다. 워드  클라우드로 나타낸 악성코드의 대표 Native API를 통해 악성 유무를 판단하고 악성 행위를 예측할 수 있다.

## 4. 악성코드 분석 실험 및 결과

### 4.1 Native API 추출

악성코드의 대표 Native API를 선정하기 위한 실험 데이터는 악성코드 가운데 가장 많이 발생하는 TROJAN과 WORM만을 대상으로 하였으며 샘플은 정상파일(주요프로그램) 20종류, VX-Heaven에서 제공하는 임의의 WORM 100종류, TROJAN 100종류를 대상으로 실험을 수행하였다. 실험 환경은 VMware-Windows XP SP2에서 수행되었으며, API Monitor v2.0 을 통해 Native API를 추출하였다.

**Table 3.** Native API lists
N:NORMAL, T:TROJAN, W:WORM

| API | N | T | W |
|---|---|---|---|
| NtCreateNamedPipeFile | 0% | 0% | 2% |
| NtPulseEvent | 0% | 4% | 8% |
| NtSetContextThread | 0% | 8% | 13% |
| NtGetContextThread | 0% | 9% | 13% |
| NtCancelTimer | 5% | 0% | 0% |
| NtOpenDirectoryObject | 5% | 3% | 4% |
| NtApphelpCacheControl | 10% | 0% | 0% |
| NtOpenKeyEx | 15% | 0% | 0% |
| NtRemoveIoCompletion | 15% | 5% | 4% |
| NtFlushInstructionCache | 15% | 29% | 19% |
| NtCreateProcessEx | 15% | 41% | 52% |
| NtQuerySection | 15% | 41% | 52% |
| NtReadVirtualMemory | 15% | 43% | 53% |
| NtQueryInformationJobObject | 15% | 44% | 54% |
| NtFlushBuffersFile | 20% | 0% | 2% |
| NtAllocateLocallyUniqueId | 25% | 23% | 6% |
| NtWriteVirtualMemory | 25% | 41% | 53% |
| NtDeleteKey | 35% | 3% | 5% |
| NtQueryFullAttributesFile | 40% | 1% | 0% |
| NtDeleteValueKey | 40% | 5% | 14% |
| NtQueryTimerResolution | 55% | 14% | 28% |
| NtAdjustPrivilegesToken | 60% | 11% | 26% |
| NtQueryVirtualMemory | 60% | 59% | 55% |
| NtCreateIoCompletion | 70% | 26% | 42% |
| NtNotifyChangeKey | 70% | 32% | 49% |
| NtQuerySystemTime | 80% | 31% | 52% |
| NtSetValueKey | 85% | 38% | 50% |
| NtQueryInformationThread | 90% | 29% | 32% |
| NtSetEvent | 95% | 64% | 50% |
| NtConnectPort | 100% | 64% | 66% |
| NtCreateMutant | 100% | 69% | 72% |
| NtOpenEvent | 100% | 71% | 75% |
| NtReadFile | 100% | 72% | 77% |
| NtDeviceIoControlFile | 100% | 77% | 90% |
| NtReleaseMutant | 100% | 80% | 81% |
| NtSetInformationFile | 100% | 82% | 89% |
| NtSetInformationProcess | 100% | 84% | 79% |
| NtAccessCheck | 100% | 89% | 86% |
| NtOpenProcessToken | 100% | 90% | 88% |
| NtWaitForSingleObject | 100% | 90% | 89% |
| NtQueryInformationFile | 100% | 92% | 92% |
| NtQueryInformationProcess | 100% | 93% | 95% |
| NtQueryValueKey | 100% | 94% | 96% |
| NtCreateFile | 100% | 94% | 99% |
| NtCreateSection | 100% | 95% | 91% |
| NtMapViewOfSection | 100% | 95% | 93% |
| NtUnmapViewOfSection | 100% | 95% | 93% |
| NtOpenFile | 100% | 96% | 97% |
| NtOpenKey | 100% | 96% | 97% |
| NtQuerySystemInformation | 100% | 96% | 97% |
| NtClose | 100% | 99% | 98% |
| NtSetInformationThread | 100% | 100% | 100% |

Table 3은 악성코드와 정상 파일에서 추출한 Native API 리스트이다. 추출한 Native API는 전체 127개이며, 일부 Native API는 정상 파일 보다 악성코드에서 확연히 많이 사용한다는 결과를 볼 수 있다.

### 4.2 가중치 적용

다음으로는 악성코드의 대표 Native API를 선정하고 Score를 결정하는 실험을 진행하였다. Table 2와 같이, 정상파일(N), 트로이목마(T), 웜(W)에 대해 어느 정도의 확률로 Native API가 발생하는지 확인하였다. Score는 식(1)과 같이 나타낼 수 있으며, 트로이목마의 발생 확률에서 정상파일의 발생 확률을 뺀 결과 또는 웜의 발생 확률에서 정상 파일을 뺀 결과 중에 최대값으로 구하며, 워드 클라우드에 적용할 Score는 Table 4와 같다.

$$Score = MAX(T\text{-}N \text{ or } W\text{-}N) \quad (1)$$

Table 4. Native API Score

| API | Score |
|---|---|
| NtQueryInformationJobObject | 39 |
| NtReadVirtualMemory | 38 |
| NtCreateProcessEx | 37 |
| NtQuerySection | 37 |
| NtWriteVirtualMemory | 28 |
| NtFlushInstructionCache | 14 |
| NtGetContextThread | 13 |
| NtSetContextThread | 13 |
| NtPulseEvent | 8 |
| NtRaiseHardError | 5 |
| NtAlertThread | 2 |
| NtSetSecurityObject | 3 |
| NtLockVirtualMemory | 4 |
| NtRestoreKey | 3 |
| NtCreateNamedPipeFile | 2 |
| NtFlushKey | 2 |
| NtSaveKey | 2 |
| NtSetSystemTime | 2 |

### 4.3 기준점수에 따른 탐지율과 오탐율

파일의 악성유무를 판단하기 위해 Table 4에서 정한 악성코드의 대표 Native API의 Score를 합산하여 나타내는 기준점수별로 임의의 파일에 대해서 탐지율과 오탐율을 구하고 F-measure 측정을 통해 악성 유무를 판단할 수 있는 기준점수를 정하는 실험을 수행하였다. 실험을 위한 데이터 셋으로 임의의 정상파일 20종류와 악성파일 20종류를 대상으로 Native API를 추출한다.

Table 5. Detection rate and False Detection rate according to a threshold

| Threshold | Detection Count | Detection rates | False Detection Count | False Detection Rate |
|---|---|---|---|---|
| 0 | 20 | 100% | 20 | 100% |
| 2 | | 0% | 1 | 5% |
| 3 | | 0% | 1 | 5% |
| 5 | 1 | 5% | 1 | 5% |
| 8 | 2 | 10% | | 0% |
| 12 | 2 | 10% | | 0% |
| 13 | 3 | 15% | | 0% |
| 14 | 6 | 30% | 4 | 20% |
| 17 | 1 | 5% | | 0% |
| 18 | 1 | 5% | | 0% |
| 20 | 2 | 10% | | 0% |
| 21 | 2 | 10% | | 0% |
| 25 | 3 | 15% | | 0% |
| 26 | 1 | 5% | | 0% |
| 28 | 9 | 45% | 4 | 20% |
| … | | | | |
| 81 | 6 | 30% | | 0% |
| 82 | 4 | 20% | | 0% |
| 83 | 4 | 20% | | 0% |
| 84 | 4 | 20% | | 0% |
| 85 | 5 | 25% | | 0% |
| 86 | 5 | 25% | | 0% |
| 87 | 9 | 45% | | 0% |
| 88 | 9 | 45% | | 0% |
| 89 | 11 | 55% | | 0% |
| 90 | 9 | 45% | | 0% |
| 91 | 8 | 40% | | 0% |
| 92 | 6 | 30% | | 0% |
| 93 | 5 | 25% | | 0% |
| 94 | 5 | 25% | | 0% |
| 95 | 5 | 25% | | 0% |
| 96 | 4 | 20% | | 0% |
| 97 | 4 | 20% | | 0% |
| 98 | 4 | 20% | | 0% |
| 99 | 4 | 20% | | 0% |
| 100 | 6 | 30% | | 0% |
| 101 | 13 | 65% | 2 | 10% |
| 102 | 13 | 65% | 2 | 10% |
| 103 | 20 | 100% | 3 | 15% |
| The rest is omitted. | | | | |

Table 5와 같이, 기준 점수가 103점에서 탐지율이 가장 높고 오탐율은 가장 낮은것으로 나타났다. F-measure는 실험의 정확도를 측정하기 위한 값으로 F-measure 측정치가 높다는 것은 정밀도와 재현율 모두가 상당히 크다는 것을 보장한다[10].

Table 6. Definition of the F-measure

| | | Predicted class (expectation) | |
|---|---|---|---|
| | | Positive | Negative |
| Actual class (observation) | Positive | TP | FN |
| | Negative | FP | TN |

$$\mathrm{Precision}(P) = \frac{TP}{TP+FP} \quad (2)$$

$$Recall(R) = \frac{TP}{TP+FN} \quad (3)$$

$$F-measure(F) = 2* \frac{\mathrm{Precision}*Recall}{\mathrm{Precision}+Recall} \quad (4)$$

Table 6은 F-measure의 정의를 나타낸다. F-measure 실험에서 정밀도(Precision)는 실험 결과가 참인 것 중에 실제 결과가 참인 비율을 나타내고, 재현률(Recall)은 실제 결과가 참인 것 중에 실험 결과가 참인 비율을 나타낸다. F-measure 측정치는 정밀도와 재현률의 조화평균을 나타낸다.

Table 7. F-measure according to threshold

| threshold | TP | FP | FN | P | R | F |
|---|---|---|---|---|---|---|
| 0 | 20 | 20 | 0 | 0.500 | 1.000 | 0.667 |
| 2 | | 1 | 20 | 0.000 | 0.000 | 0 |
| 3 | | 1 | 20 | 0.000 | 0.000 | 0 |
| 5 | 1 | 1 | 19 | 0.500 | 0.050 | 0.091 |
| 8 | 2 | | 18 | 1.000 | 0.100 | 0.182 |
| 12 | 2 | | 18 | 1.000 | 0.100 | 0.182 |
| 13 | 3 | | 17 | 1.000 | 0.150 | 0.261 |
| 14 | 6 | 4 | 14 | 0.600 | 0.300 | 0.400 |
| 17 | 1 | | 19 | 1.000 | 0.050 | 0.095 |
| 18 | 1 | | 19 | 1.000 | 0.050 | 0.095 |
| 20 | 2 | | 18 | 1.000 | 0.100 | 0.182 |
| 21 | 2 | | 18 | 1.000 | 0.100 | 0.182 |
| 25 | 3 | | 17 | 1.000 | 0.150 | 0.261 |
| threshold | TP | FP | FN | P | R | F |
| 26 | 1 | | 19 | 1.000 | 0.050 | 0.095 |
| 28 | 9 | 4 | 11 | 0.692 | 0.450 | 0.545 |
| ... | | | | | | |
| 81 | 6 | | 14 | 1.000 | 0.300 | 0.462 |
| 82 | 4 | | 16 | 1.000 | 0.200 | 0.333 |
| 83 | 4 | | 16 | 1.000 | 0.200 | 0.333 |
| 84 | 4 | | 16 | 1.000 | 0.200 | 0.333 |
| 85 | 5 | | 15 | 1.000 | 0.250 | 0.400 |
| 86 | 5 | | 15 | 1.000 | 0.250 | 0.400 |
| 87 | 9 | | 11 | 1.000 | 0.450 | 0.621 |
| 88 | 9 | | 11 | 1.000 | 0.450 | 0.621 |
| 89 | 11 | | 9 | 1.000 | 0.550 | 0.710 |
| 90 | 9 | | 11 | 1.000 | 0.450 | 0.621 |
| 91 | 8 | | 12 | 1.000 | 0.400 | 0.571 |
| 92 | 6 | | 14 | 1.000 | 0.300 | 0.462 |
| 93 | 5 | | 15 | 1.000 | 0.250 | 0.400 |
| 94 | 5 | | 15 | 1.000 | 0.250 | 0.400 |
| 95 | 5 | | 15 | 1.000 | 0.250 | 0.400 |
| 96 | 4 | | 16 | 1.000 | 0.200 | 0.333 |
| 97 | 4 | | 16 | 1.000 | 0.200 | 0.333 |
| 98 | 4 | | 16 | 1.000 | 0.200 | 0.333 |
| 99 | 4 | | 16 | 1.000 | 0.200 | 0.333 |
| 100 | 6 | | 14 | 1.000 | 0.300 | 0.462 |
| 101 | 13 | 2 | 7 | 0.867 | 0.650 | 0.743 |
| 102 | 13 | 2 | 7 | 0.867 | 0.650 | 0.743 |
| 103 | 20 | 3 | 0 | 0.870 | 1.000 | 0.930 |
| The rest is omitted. | | | | | | |

F-measure 측정치가 가장 높은 값은 0.93으로 기준점수 103점에서 정밀도(0.87)와 재현율(1.0) 모두 상당히 높다. 위 실험 결과를 통해서 높은 탐지율과 낮은 오탐율을 위한 기준점수로 103점을 설정할 수 있다.

### 4.4 시각화

시각화는 실제로 알지 못하는 임의의 파일에 대해서 제안하는 악성코드 분석 기법을 적용하여 워드 클라우드로 나타내고 악성 유무를 판단하는 실험으로, 임의의 2개의 파일에 대해서 Native API를 추출하고 가중치를 적용하여 워드 클라우드로 나타냈다. 파일로부터 추출된 Native API 가운데 악성코드에서 많이 사용하는 API가 발견되면 가중치에 따라서 크게 표현된다. 만약 가중치의 합이 악성코드 판단 기준점수보다 높다면 악성코드로 판단할 수 있다.

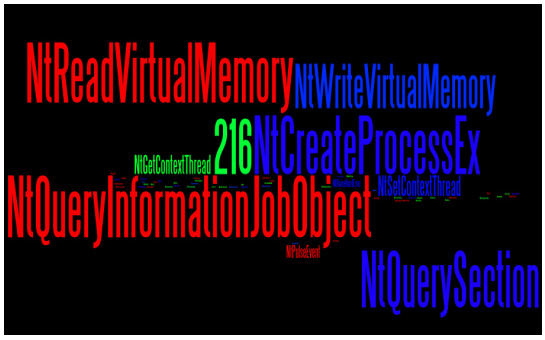


Fig. 6. Word cloud for malware

Fig.6에는 Worm.Win32.AutoTDSS.act 샘플을 대상으로 워드 클라우드로 나타냈다. 5개 이상의 Native API가 상대적으로 다른 Native API 보다 크게 나오는 것을 알 수 있다. 또한 216점으로 4.3절에서 F-measure 측정을 통해 얻은 기준점수 103점을 초과한다. 이는 Fig.6에 사용한 임의의 파일이 악성일 확률이 높다는 것을 알 수 있으며, VirusTotal을 통해 검사한 결과 54개 가운데 48개의 백신에서 탐지되었다.

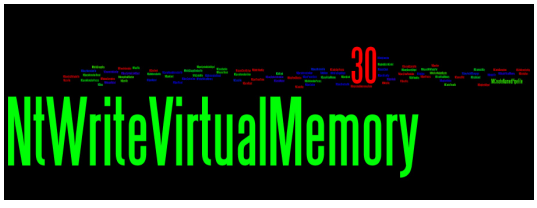


Fig. 7. Word cloud for Nomal file

Fig.7은 한글 2010을 워드 클라우드로 나타낸 것이다. 1개의 Native API가 나타났으며 30점으로 기준점수 103점 미만인 것으로 나타났다. 따라서 정상파일로 판단되었으며, VirusTotal을 통해 검사한 결과 54개의 백신에서 정상파일로 나타났다.

## 5. 결론

매일 수만 건의 신종 악성코드가 발생하고 있는 상황에서 악성코드 분석 기법에 대한 연구는 지속적으로 필요하다. 기존의 악성코드 분석 기법은 시그니쳐 기반, API 흐름, 문자열 등 다양한 방법이 존재하지만 직관적으로 악성코드를 판단하는 데는 한계점이 존재한다. 본 논문에서 악성 유무와 악성 행위를 직관적으로 파악하기 위한 방법으로 Native API 함수를 워드 클라우드로 표현하는 방법을 제안하였다. 실험을 통해 악성코드의 대표 Native API를 선정하고 해당 Native API가 악성코드와 정상코드에 대해 어느정도의 확률로 존재하는지를 계산하여 Score로 사용했다. Score는 워드 클라우드에서 악성코드의 대표 Native API를 크기로 나타내는 요소로써, Score 점수의 합은 악성유무를 판단할 수 있는 점수를 제공한다. 또한 악성과 정상의 판단을 위한 점수로 F-measure 측정을 통해 기준점수를 결정하였으며, 40개의 임의의 파일들에 대해서 탐지율 100%, 오탐율 15%로 나타났다. 비교적 높은 탐지율이 나타났지만 오탐율 또한 10%미만으로 낮추기 위한 노력이 필요하다. 제안하는 악성코드 분석 방법은 동적 분석 단계에 API를 분석할 때 적용하기에 적합한 분석 방법으로 더 많은 실험 데이터로 가중치를 적용한다면 기존의 결과보다 오탐율을 줄일 수 있을 것으로 기대한다. 하지만 본 논문에서 실험한 내용은 소량의 악성코드 샘플 및 기 진단된 악성코드를 대상으로 수행되어서 최신 악성코드에 적용하기에는 한계가 있을 수 있다. 따라서. 향후 연구로 단일 API 함수가 아닌 API의 시퀀스를 이용한 시각화에 대한 연구를 진행할 계획이다.

## References


[1] Tae-hyung Kim, "Security, IT industry trends", boannews, 2015, www.boannews.com

[2] Kyung-ho Son, "This year mobile security keyword, banking, payment, SMS phishing, IoT", ZDNetKorea, 2015, www.zdnet.co.kr

[3] Pauline KOH, “System call sequence based malware analysis”, pp. 4, Korea University, 2013.

[4] E. Carrera, Gergely Erdelyi, "Digital genome mapping - advanced binary malware analysis", Virus Bulletin Conference, 2004.

[5] won-hyuck choi, “Inference virus variants Using the Virus Genome”, Monthly CyberSecurity, 2005.

[6] Jae-Hyun Im, “Malware detection method using Visualization technique”, pp. 6, Hanyang University, 2014

[7] In-Soo Song, Dong-Hui Lee, Kui-Nam Kim, "A Study on Malicious Codes Crouping and Analysis Using Visualiztion", pp. 51-60, journal of information and security, 2010.

[8] Tae-woo Kang, Jae-ik cho, Man-hyun Chung, Jong-sub Moon, "Malware Detection Via Hybrid Analysis for API Calls", Journal of The Korea Institute of Information Security & Cryptology, Vol. 17, No. 6, pp. 89-98, 2007

[9] Jae-woo Park, Sung-tae Moon, Gi-Wook Son, In-Kyoung Kim, Kyoung-Soo Han, Eul-Gyu Im, Il-Gon Kim, “An Automatic Malware Classification System using String Lsit and APIs”, Journal of Security Engineering, Vol. 8, No. 5, pp. 611-626, 2011.

[10] Jae-ho Lee, Sangjin-Lee, "A Study on Unknown Malware Detection using Digital Forensic Techniques", Journal of The Korea Institute of Information Security & Cryptology, Vol. 24, No. 1, pp. 107-122, 2014.
DOI: http://dx.doi.org/10.13089/JKIISC.2014.24.1.107



**유 성 태(Sung-Tae Yu)** **[준회원]**

- 2014년 2월 : 호서대학교 정보보호학과 졸업(공학사)
- 2014년 3월 ~ 현재 : 호서대학교 정보보호학과 대학원 석사과정
- 2015년 8월 ~ 현재 : 한국인터넷진흥원 주임연구원

<관심분야>
악성코드, IoT 보안 프로토콜

**오 수 현(Soo-Hyun Oh)** **[정회원]**

- 1998년 2월 : 성균관대학교 전기전자 및 컴퓨터공학부 대학원 졸업 (공학석사)
- 2003년 8월 : 성균관대학교 전기전자 및 컴퓨터공학부 대학원 졸업 (공학박사)
- 2004년 3월 ~ 현재 : 호서대학교 정보보호학과 교수

<관심분야>
암호학. 네트워크 보안, IoT 보안