랜섬웨어 Native API 정보 분석을 통한 탐지 방법 연구

장준영 김우재* 옥정윤** 임을규***

*한양대학교 컴퓨터 공학과

**한양대학교 정보보안학과

***한양대학교 컴퓨터소프트웨어학부

Research on ransomware detection through Native API information analysis

Jun-Young Chang Woo-Jae Kim* Jeong-Yun Oak** Eul-Gyu Im***

*Division of Computer Science, Hanyang University.

**Department of Information Security, Hanyang University

***Department of Computer and Software, Hanyang University

요 약

최근 파일을 암호화하는 악성코드인 랜섬웨어(ransomware)가 유행하고 있다. 비트코인(bitcoin)의 등장으로 해커는 손쉽게 사용자에게 금전을 요구할 수 있게 되었기 때문에 앞으로도 랜섬웨어는 꾸준히 유행할 것으로 보인다. 랜섬웨어가 파일을 암호화할 때 일반적인 프로그램과 마찬가지로 프로세스(process)를 생성하고, API(Application Programming Interface)를 호출한다. 본 논문에서는 프로세스 API 중에서 실제 시스템 영역에 접근하는 윈도우 Native API[1]들을 서열과 빈도수를 중심으로 분석하여 랜섬웨어의 특징을 추출, 분류한다. 결과적으로 파일의 프로세스 API 정보를 주었을 때 해당 파일이 랜섬웨어인지, 안전한 실행 파일인지 정확히 판별하는 것이 목표이다.

I. 서론

IT 기술은 시대가 거듭할수록 점점 발달 되고, 그 영향도 커지며 많은 분야에 적용되고 있다. 이에 따른 부작용들도 나타나고 있으며, 그 중 하나가 공격 대상 증가에 따른 사이버 공격의 증가이다. 과거 대부분의 사이버 공격의 경우, 개인의 해킹 기술을 과시하거나 호기심이 목적이지만 오늘날에는 금전적인 이득을 취하기 위한 수단으로 사이버 공격이 행해진다.

금전적 이득을 위한 대표적인 사이버 공격이 랜섬웨어이다. '몸값(ransom)'과 '소프트웨어(software)의 합성어인 랜섬웨어는 시스템 및 데이터를 암호화하여 사용할 수 없도록 만든 후, 이를 인질로 삼아 금전을 요구하는 악성 프로그램이다. 2005년에 본격적으로 알려지기 시작했지만, 이 시기에는 낮은 암호화 수준으로 복호화 방법을 통해 쉽게 데이터를 복구할 수 있었다. 하지만 2013년 거래가 쉬운 비트코인이 개발되어 직접적인 금품요구가 쉬워지고, 강력한 암호화 알고리즘을 이용한 랜섬웨어가 등장하면서 전 세계적으로 공공기관, 기업, 개인 PC 등을 향한 공격이 매년 늘어나는 추세이다. 이스트소프트에서 발표한 2016년 랜섬웨어 동향 결산[2]에 따르면 국내의 경우 2016년 한해 알약 백신을 통해 사전 차단된 랜섬웨어 공격이 총 397만 4,658건으로 적지 않은 수의 랜섬웨어 공격 시도가 발생한다는 것을 알 수 있다.

이처럼 매년 수가 급격히 증가하고 그 수법도 매우 고도화되는 랜섬웨어를 분석하고 그

특징을 도출해내려는 시도는 이전부터 있었다. 하지만 빠른 속도로 변종이 생성되기 때문에 이미 검출된 악성코드의 특징을 이용해 유사한 악성코드를 검출하는 방식으로 탐지하기는 어렵다[3]. 따라서 분석가들이 직접 로그(log) 파일을 살펴보며 랜섬웨어의 특징들을 찾아내야 한다. 이 과정에 Cuckoo sandbox[4]라는 분석 도구가 많이 사용되는데, 이는 가상머신 환경에서 직접 악성코드를 실행하며 발생하는 프로세스, 메모리, 네트워크 등을 동적 분석할 수 있게 해주는 시스템이다.

본 논문에서는 이 Cuckoo sandbox의 분석 결과로 생성된 로그 파일에서 프로세스 API, 그중에서 Native API에 집중해 실제로 랜섬웨어가 사용자의 컴퓨터를 암호화하는 과정에서 사용되는 API들의 상관관계를 서열과 빈도수를 중심으로 분석할 것이다.

II. 배경지식

2.1 랜섬웨어

랜섬웨어는 크게 사용자 화면을 장악하는 Lock Screen 랜섬웨어, 데이터를 암호화하는 Encryption 랜섬웨어, boot 영역을 훼손하는 Master Boot 랜섬웨어, 안드로이드 기기를 대상으로 하는 Mobile Device 랜섬웨어[5]로 나누어지며, 2017년 기준으로 우리나라에 보고된 랜섬웨어는 275종에 이른다[6]. 본 논문에서는 이 중에서도 2017년 5월 전 세계 90개국 이상에 대규모로 전파된 WannaCry(또는 WannaCrypt)[7] 랜섬웨어를 주요 대상으로 한다. 이 랜섬웨어는 마이크로소프트 윈도우 운영체제에서 파일 공유에 사용하는 서버 메시지 블록(Server Message Block)의 취약점을 이용한 Encryption 랜섬웨어로, 최신 윈도우 보안 패치를 적용하지 않아 보안이 취약한 PC로 전파되며 PC 내 다양한 문서파일과 압축파일, DB 파일 등을 암호화하여 사용할 수 없게 만든다. 이메일 첨부파일을 통해 유포되는 일반적인 랜섬웨어와 달리 인터넷 네트워크 접속만으로 감염되는 특징을 가진다.

2.2 Cuckoo sandbox

프로그램의 행위 분석은 크게 정적 분석과 동적 분석으로 나뉜다. 정적 분석은 실제 실행 없이 소프트웨어를 분석하는 방식으로, 코드 분석을 통해 프로그램의 실행 방향을 예측한다. 하지만 대부분의 악성 프로그램들은 난독 화 과정을 거치기 때문에 분석이 어렵다. 따라서 악성코드는 주로 동적 분석을 한다. 행위 자체를 탐지하기도 쉽고, 무엇보다 행동 과정을 실제로 지켜볼 수 있기 때문이다. 하지만 동적 분석을 시행하려면 실제로 악성코드를 작동시켜야 하기 때문에, 실험 장비가 감염될 가능성이 있다. 본 실험에서는 Cuckoo sandbox 오픈소스 악성코드 분석 시스템을 사용한다. 이 시스템에 악성 프로그램을 입력하면 Cuckoo sandbox가 관리하는 독립된 가상 환경에서 실제로 악성코드를 실행하고, 이때 발생하는 프로세스, 메모리, 네트워크 변화를 기록한다. 이 분석 결과 보고서는 JSON 등의 형식으로 저장되는데, 사용자는 이 분석 보고서만으로 안전한 환경에서 악성 프로그램을 살펴볼 수 있다.

2.3 Native API

프로그램은 실행 과정에서 프로세스를 생성하고, 다양한 API를 호출한다. 이 중에서 Native API는 일반적으로 사용자가 접근하지 못하는 OS의 기능들을 간접적으로 호출할 수 있는 도구로, 그 이름과 역할이 정해져 있다. 따라서 이전까지 없던 변종 프로그램이 등장하더라도 미리 알려진 Native API를 사용할 수밖에 없다. 본 논문에서는 랜섬웨어가 실행될 때 호출되는 프로세스 API를 추출, 그중에서 Native API만을 추려내어 변종에 대응할 수 있는 탐지 방법을 목표로 진행한다.

III 실험설계

3.1 샘플 분석 방식

총 두 가지 기준으로 API를 분석하는데, 호출된 모든 API를 서열화하여 순서대로 나열한 후 특정 그룹의 API 출현 정도 등을 조사하는 서열 중심 분석 N-gram 방식[8]과 어떤 종류의

API가 등장하고, 어떤 API가 빈번히 호출되는지를 빈도수 중심으로 분석하는 TF-IDF 방식[9]을 사용한다.

3.1.1 서열 중심 분석 (N-gram)

N-gram은 문서 간의 유사함을 찾기 위해 처음 제안된 기법으로, 연속한 N개의 단어나 텍스트를 하나의 하위 문자열로 나누어 통계학적으로 사용하는 방법을 말한다. N의 단위는 1bit, 1byte부터 하나의 단어나 음절까지 필요에 맞는 기준에 따라 자유롭게 선택할 수 있으며, N-gram으로 생성된 하위 문자열의 빈도수를 이용하여 문자의 유사도를 측정한다. 예를 들어 "랜섬웨어"라는 문자열에 2-gram을 적용하는 경우 "랜섬", "섬웨", "웨어"라는 3가지 하위 문자열이 생성되며, 각각 하위 문자열의 빈도수는 1이다.

본 논문에서는 각각의 Native API 이름을 그대로 사용하지 않고, Native API 이름을 각각 A1, B1 등과 같이 고유한 문자에 대응시켜, 이를 N-gram에 적용하였다.

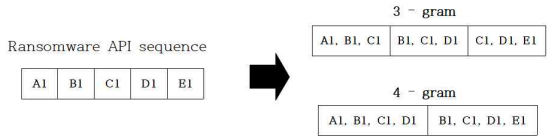

그림 1 서열 정렬 예시

N-gram의 경우 단순한 형태의 분석법이기 때문에, 결과를 가공할 필요가 있다. 가장 대표적으로 사용하는 방법이 의미가 없이 사용되는 조사나 연결어인 불용어를 제거하는 것이다. 이를 본 실험에 적용하여 일반 파일의 특징 서열을 불용어로 설정하여, 불용어를 제거한 경우와 아닌 경우를 나누어서 실험을 진행했다.

3.1.2 빈도수 중심 분석(TF-IDF)

TF-IDF는 문서에서 특정 단어가 얼마나 자주 나타나는지에 대한 정보(TF)와 특정 단어가 얼마나 많은 문서에서 등장하는지, 즉 얼마나 흔한 단어인지에 대한 정보의 역수(IDF)를 이용하여 문서와 단어 간의 관련성을 비교한다. 이를 계산하는 식은 분석 대상에 따라 다양하게 적용할 수 있다[10].

$$TF\text{-}IDF = TF * IDF \quad (1)$$

TF는 Term Frequency의 약자로, 한 문서 내에서 특정 단어가 얼마나 빈번히 등장하는지를 계산한 식이다. 단순히 전체 단어 개수로 특정 단어 개수를 나눈 값을 이용하거나, 문서 내 등장 여부만을 기록하는 Boolean 빈도 계산법, 또는 가중치를 적게 주고 싶을 때 로그를 취한 값을 사용할 수도 있다.

IDF는 Inverse Document Frequency의 약자이다. 특정 단어가 모든 문서에서 등장한다면 이 단어는 문서를 분류하는 특징이 부족하다고 할 수 있는데, 이런 사실을 이용해 전체 문서 중 특정 단어가 나타난 문서 수를 세어 이를 역수로 계산한다. TF 계산과 마찬가지로 주고 싶은 가중치에 따라 단순 개수 계산부터 로그 취한 값 계산까지 다양하게 적용할 수 있다.

본 논문에서는 이러한 특징 추출 방식을 프로그램과 API로 확장해서, 특정 API와 프로그램과의 연관성을 나타낼 수 있는 지표로 사용한다. 하지만 보통의 문서와 단어의 관계는 프로그램과 API의 관계와 차이점이 있다. 첫째, 본 논문에서 대상으로 삼는 Native API는 단어와는 다르게 그 수가 매우 제한적이다. 따라서 같은 API가 훨씬 빈번히 사용된다. 둘째, 악성코드는 탐지를 피하려고 의도적으로 불필요한 기능을 여러 번 수행해 본연의 목적을 숨기거나 전혀 불필요한 행동을 하기도 한다. 따라서 같은 기능을 수행하는 샘플이 더라도 전혀 다른 API를 추가로 사용할 수도 있다.

이러한 어려움 때문에 랜섬웨어를 다른 프로그램과 구별하기 위해 TF, IDF 어느 쪽에 가중치를 더 부여해야 하는지에 대한 정확한 기준을 세우기가 힘들다. 이 때문에 실험을 다양하게 설계하여 진행하였다. 수식을 선형적으로 또는 지수적으로 설정하기도 했고, 특정 요소를 아예 제외하기도 했다.

3.2. 유사도 측정 방식

샘플을 분석하여 나온 결과값과 분석하려는 대상의 유사도를 측정하는 방식에는 여러 가지

가 있지만 본 논문에서는 코사인 유사도(cosine similarity)를 사용하였다. 코사인 유사도는 두 벡터의 유사도를 측정하기 위한 간단한 공식이다. 이때 벡터의 크기가 아닌 방향성의 유사도를 판단한다. 코사인 유사도를 구하기 위한 공식은 다음과 같다.

$$\text{Similarity} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A^2 \sum_{i=1}^{n} B_i^2}} \qquad (2)$$

일반적으로 방향이 완전히 일치하면 1, 완전히 반대이면 -1의 값을 갖지만, TF-IDF처럼 값이 음수를 갖지 않는 경우는 0에서 1 사이의 범위의 값만을 가진다. 샘플들과 분석하려는 대상의 코사인 유사도를 계산하여 두 대상 사이의 유사도를 측정할 수 있다.

Ⅳ 실험내용

본 논문에서는 서열 중심 분석과 빈도수 중심 분석을 위해 Cuckoo sandbox를 활용하여, 'report.JSON' 파일을 생성하여 분석했다.

4.1 서열 중심 분석

서열 중심 분석을 하기 위해 500개의 랜섬웨어의 모든 Native API를 추출하여 고유한 문자표를 생성했다. 중복된 값은 삭제하고 하나의 Native API에 하나의 고유한 문자를 배정했다.

표 1 고유한 문자로 변환된 Native API

Native API	Unique code
NtQueryAttributesFile	A0
NtResumeThread	A1
NtOpenKeyEx	A2
RtlDecompressBuffer	A3
…	…
NtEnumerateValueKey	F6

총 57개의 Native API가 추출되었다. 이렇게 변환된 Native API 고유 문자를 하나의 N으로 하여, 4개의 고유 문자가 묶여 하위 문자열을 생성하는 4-gram으로 실험을 진행했다[11]. 유의미한 결과를 도출하기 위해 4-gram의 모든 하위 문자열의 빈도수가 1인 결과는 제외하였다. 위의 결과를 통해 나온 결과를 바탕으로, 랜섬웨어 특징 Native API 서열 찾기 위해 TF-IDF 식에서 TF 값만을 이용했다. 전체 빈도수를 TF를 통해 빈도수를 측정하지 않고, 각각의 파일에 대해서 TF를 적용하여 빈도수를 측정했다. 이는 똑같은 서열이 특정 파일에서 많이 측정되어 전체 결과값에 영향을 미칠 수 있기 때문이다. TF에 따라 랜섬웨어의 특징이 되는 서열이 결정되므로 적당한 크기를 가지며 랜섬웨어의 특징을 가지는 TF 값을 결정해야 한다. 그림 2 을 보면, TF 값을 0.01 이상의 TF 값만을 사용하는 경우 1760개의 서열이 특징 서열로 선택되기 때문에 올바르지 않다. 0.04 보다 큰 값으로 하는 경우 28개의 서열이 선택되지만, 랜섬웨어를 대표하는 특징이 부족했다. 실험을 통해 최적의 TF 값 0.038 이상을 이용하여 32개의 랜섬웨어 특징 서열을 도출했다.

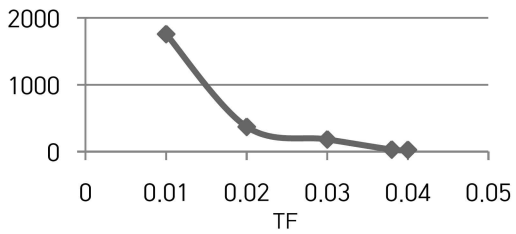

그림 2 TF에 따른 특징 서열의 개수 변화

이를 이용하여, 각각 랜섬웨어 파일 165개, 일반 파일 123개를 이용하여 실험을 진행했다.

4.1.1 랜섬웨어 특징 서열 이용

랜섬웨어 특징 서열만을 이용하여 실험을 진행하였다. 165개의 랜섬웨어 파일을 각각 65개의 랜섬웨어 파일 R1, 100개의 랜섬웨어 파일 R2로 나누고, 123개의 일반 파일을 77개의 일반 파일 N1, 46개의 일반 파일 N2로 나누어 실험을 진행하였다. R은 랜섬웨어 특징 서열을 가지고 있어 랜섬웨어로 분류된 것을 나타내고 'No R' 특징 서열이 없어서 일반 파일로 분류된 것이다. 실험 결과는 아래 그림 3이다.

.

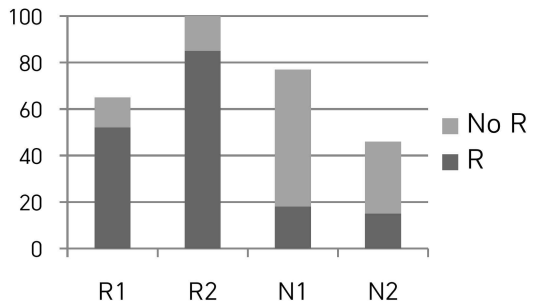

그림 3 랜섬웨어 특징 서열을 이용한 실험 결과

그림 3에서 알 수 있듯이 R1은 80%, R2는 85%의 확률로 랜섬웨어 파일을 랜섬웨어로 분류했다. 하지만 일반 파일의 경우 N1은 27%, N2는 32%의 확률로 일반 파일을 랜섬웨어로 분류했다.

4.1.2 일반 파일 특징 서열을 제거

랜섬웨어 특징 서열 32개에서 일반 파일 특징 서열 28개 중 중복되는 값을 제거하여, 18개의 특징 서열만을 가지고 실험을 진행했다. 실험 결과는 아래 그림 4와 같다.

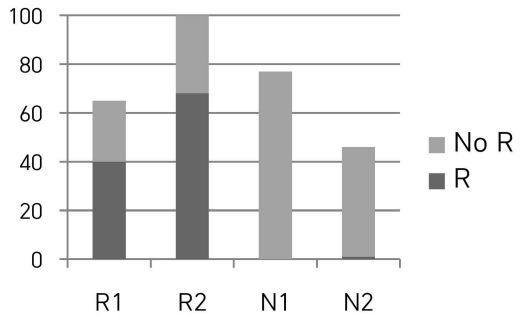

그림 4 일반 특징 서열을 제거한 실험 결과

랜섬웨어를 일반 파일로 분류하는 오류가 증가했지만, 일반 파일을 랜섬웨어로 분류하는 오류는 1% 내외로 측정된다.

4.2 빈도수 중심 분석

분석 대상 파일 종류에 따른 특징을 분석하기 위해, 유해하지 않은 실행 파일과 랜섬웨어가 아닌 악성코드, 랜섬웨어 샘플 각 400, 400, 800개를 대상으로 실험을 진행하였다. 여러 가지 고려해야 할 조건들이 혼합되어 있으므로 어떤 수식을 사용해야 가장 바람직한 결과를 도출해낼 수 있는지 알아내기가 힘들므로 여러 조건에서 실험을 진행했다.

실험 진행 순서는 다음과 같다.

(1) 각 분석 샘플의 Cuckoo sandbox 분석 보고서에서 Native API의 이름과 개수만을 추출한다.

(2) 샘플 별로 API별 TF 값을 계산하여 저장한다.

(3) API별 IDF 값을 계산하여 샘플별 TF-IDF 값을 저장한다.

(4) 샘플 전체에 대한 TF-IDF 평균을 구한다.

(5) 비교하려는 대상의 TF-IDF 값을 구한다.

(6) 두 TF-IDF 값으로 코사인 유사도를 측정한다.

이때 각 문서의 TF 값 계산, 샘플 전체의 IDF 값 계산에서 수식을 변화시킬 수 있다. 식의 항등원을 이용해서 특정 값을 아예 계산식에서 제외하거나, 선형적으로 계산하거나, 로그 취한 값으로 계산하여 실험을 진행하였다. 계산에 이용한 식은 다음과 같다.

$$TF = w / d \quad (3)$$

$$TF = \log(1 + w / d) \quad (4)$$

w를 문서 내 특정 API의 등장 횟수, d를 문서 내 전체 API의 수라고 했을 때 TF 값은 수식 (3)처럼 비율로 계산할 수도 있고, 이 값에 계산할 수도 있다. 이때 특정 샘플에서는 아예 계산하려는 API가 등장하지 않을 수 있으므로 수식(4)에서 로그 계산식에 1을 더해주었다.

$$IDF = 1 \quad (5)$$

$$IDF = \log(N / n) \quad (6)$$

$$IDF = N/n \quad (7)$$

IDF의 계산식도 TF와 유사하다. 전체 문서 수를 N, 특정 API가 등장하는 문서 수를 n이라고 했을 때 비율의 역수 또는 그것의 로그 값으로 계산할 수 있다. 또 아예 IDF의 값을 1로 지정해 TF 값 비교만으로도 유의미한 결과가 나오는지 실험을 진행했다.

분석하려는 대상은 그 개수가 하나이므로 IDF 값을 어떻게 부여해야 할지도 정해야 한다. IDF 값에 1을 부여하여 계산식에서 제외하거나, 샘플의 IDF와 같은 값을 사용하는 두 가지 방법을 채택했다.

동일한 샘플을 대상으로 위의 다각적 실험을 진행한 결과 랜섬웨어가 아닌 경우와 랜섬웨어인 경우 탐지율은 다음과 같았다. (ⅰ)는 검사하려는 파일의 IDF 값을 1로 준 경우이고, (ⅱ)는 샘플과 같은 IDF 값을 사용한 경우이다.

표 2 코사인 유사도 실험 결과

input / sample	수식 (3)	수식(4)
수식 (5)	(ⅰ)98.75%, 34.375% (ⅱ)98.75%, 34.375%	(ⅰ)98.75%, 41.5% (ⅱ)98.75%, 41.5%
수식 (6)	(ⅰ)90.5%, 58.25% (ⅱ)85.375%, 55.0%	(ⅰ)90.5%, 60.25% (ⅱ)85.125%, 56.875%
수식(7)	(ⅰ)97.75%, 48.625% (ⅱ)97.5%, 65.25%	(ⅰ)97.75%, 52.25% (ⅱ)97.625%, 68.25%

실험 결과로 보아 TF의 값은 로그를 취해 계산하고, IDF의 값은 선형적으로 계산하는 것이 랜섬웨어의 탐지율을 높일 수 있었다. 즉 랜섬웨어 탐지는 TF보다는 IDF에 더 의존적으로 나타났다. 수가 한정된 Native API라고 할지라도, 특정 API가 결과물에 등장하는지 아닌지의 정보가 중요하다고 판단할 수 있다. 또한 검사하는 파일의 IDF 값 설정에 따라서 결과가 크게 차이가 나는 것으로 보아, 해당 값을 적절하게 부여한다면 높은 탐지율을 얻을 수 있을 것으로 기대된다.

Ⅴ 결론

서열, 빈도수 두 가지 요소를 기준으로 악성코드를 탐지한 결과 랜섬웨어 탐지 면에서는 랜섬웨어의 특징 서열만 이용한 N-gram 분석 방식이 더 높은 탐지율을 보였고, 랜섬웨어가 아닌 파일을 탐지할 때는 TF-IDF 식을 사용한 방식이 높은 탐지율을 보였다. 따라서 이 두 방식을 혼합하여 사용한다면 높은 정확도로 파일을 랜섬웨어인지 아닌지 구별해낼 수 있을 것이다. 본 논문에서 사용한 프로그램 샘플은 한정적이지만, 좀 더 다양하고 많은 수의 샘플을 사용하여 특징을 추출한다면 더 높은 정확도로 파일을 분류해 낼 수 있을 것으로 기대된다.

[참고문헌]

[1] Native API, https://en.wikipedia.org/wiki/Native_API

[2] 2016년 랜섬웨어 동향 결산, https://www.estsecurity.com/

[3] 주정욱, “회피기법이 있는 악성코드 분석을 위한 샌드박스용 사용자 이벤트 발생기 설계”, 목포대학교 대학원, 2015

[4] Cuckoo Sandbox 공식 홈페이지, http://www.cuckoosandbox.org/

[5] No More Ransom, Types of Ransomware, https://www.nomoreransome.org/ransomware-qa.html

[6] 한국인터넷진흥원, 사이버 위협 동향 보고서, http://www.kiise.or.kr/academy/main

[7] 김소람, “최신 랜섬웨어에 대한 암호키 복구 방안 연구”, 국민대학교, 2017

[8] 심유진, “API 호출 시퀀스 기반 악성코드 탐지 및 분류 시스템”, 한양대학교, 2016

[9] Christopher D. Manning, Prabhakar Raghavan, "An Introduction to Information Retrieval", Cambridge University Press, pp. 117-128, April, 2009

[10] Amit Singhal, “Modern Information Retrieval: A Brief Overview”, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2001

[11] 권희준, 김선우, 임을규, “Multi N-gram을 이용한 악성코드 분류 시스템”, 보안공학연구논문지 제 9권 제 6호, pp. 531-542, 2012