



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

석사학위논문

API 호출 시퀀스 기반
악성코드 탐지 및 분류 시스템

Malware Detection And Classification System
based on API Call Sequence

심 유 진

한양대학교 대학원

2016년 8월

석사학위논문

API 호출 시퀀스 기반
악성코드 탐지 및 분류 시스템
Malware Detection And Classification System
based on API Call Sequence

지도교수 임 을 규

이 논문을 공학 석사학위논문으로 제출합니다.

2016년 8월

한양대학교 대학원

컴퓨터·소프트웨어학과

심 유 진



이 논문을 심유진의 석사학위 논문으로 인준함

2016년 8월

심사 위원장 강 수 용(인)

심사 위원 임 을 규(인)

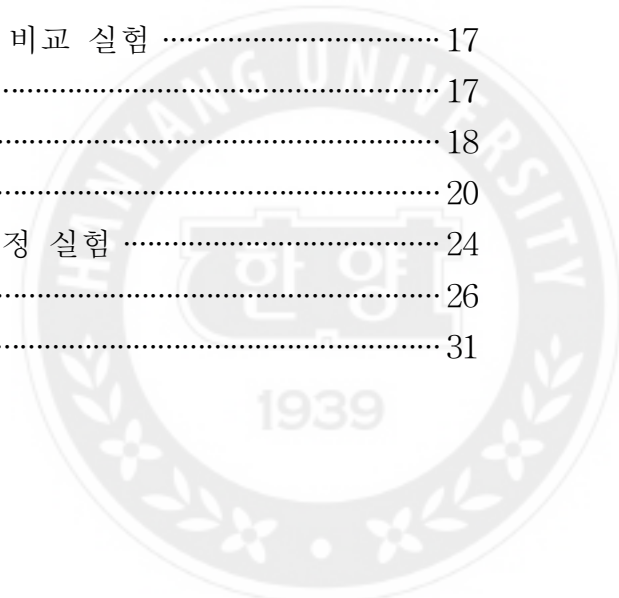
심사 위원 차 재 혁(인)

한양대학교 대학원



차 례

국 문 요 약	vi
제 1 장 서 론	1
제 1 절 연구의 필요성	1
제 2 장 연구 내용	3
제 3 장 배경 지식 및 관련 연구	4
제 1 절 API	4
제 2 절 Cuckoo sandbox	5
제 3 절 Smith Waterman Algorithm	6
제 4 절 관련 연구 동향	7
제 4 장 악성코드 유사도 측정 기법	9
제 1 절 N-gram	9
제 2 절 Cosine Similarity	11
제 3 절 Overlap Coefficient	12
제 4 절 Jaccard Index	13
제 5 절 Sorensen-Dice	14
제 6 절 파라미터	15
제 7 절 고려 사항	16
제 5 장 악성코드 유사도 측정 기법 분석 및 비교 실험	17
제 1 절 악성코드 유사도 측정 기법 분석	17
제 2 절 최적해 N을 찾는 실험	18
제 3 절 프로세스 관련 실험	20
제 4 절 N의 값에 따른 수행 시간 변화 측정 실험	24
제 5 절 알고리즘 비교 실험	26
제 6 절 유사도 계산식 비교 실험	31



제 6 장 악성코드 분석 시스템	40
제 1 절 시스템 흐름도	40
제 2 절 모듈별 설명	41
제 7 장 결론 및 향후 연구	46
제 8 장 참 고 문 헌	47



그림 차례

[그림 1] 최근 10년간 악성코드 증가 추이 (AV-TEST, http://www.av-test.org/)	1
[그림 2] 서열 정렬 예시	6
[그림 3] N-gram 슬라이딩 윈도우에 따라 추출되는 서브스트링	9
[그림 4] 악성 API 호출 시퀀스 코드화	17
[그림 5] 최적해 실험에 사용된 트레이스의 예	18
[그림 6] N값에 따른 수행 시간 변화	19
[그림 7] 모든 프로세스의 의한 API 호출 시퀀스 유사도 측정	22
[그림 8] 일반적인 프로세스의 의한 API 호출 시퀀스 유사도 측정	23
[그림 9] 일반적이지 않은 프로세스의 의한 API 호출 시퀀스 유사도 측정	24
[그림 10] 최적해 실험에 사용된 트레이스의 예	25
[그림 11] N값에 따른 수행 시간 변화	26
[그림 12] 알고리즘 비교 실험을 위한 실험 구성도	27
[그림 13] N-gram 패밀리별 유사도 측정 실험 결과	28
[그림 14] Smith-Waterman 패밀리별 유사도 측정 실험 결과	29

[그림 15] API 개수에 따른 수행 시간 측정 실험 결과	30
[그림 16] Cosine similarity를 이용한 실험 결과	31
[그림 17] Overlap coefficient를 이용한 실험 결과	32
[그림 18] Jaccard Index를 이용한 실험 결과	33
[그림 19] Sorensen-Dice를 이용한 실험 결과	34
[그림 20] Cosine & Overlap Similarity 결과	36
[그림 21] 유사도 계산식 가중치 실험 1	37
[그림 22] 유사도 계산식 가중치 실험 2	38
[그림 23] 유사도 계산식 가중치 실험 3	39
[그림 24] 악성코드 분석 시스템 구성도	40
[그림 25] 인스트럭션 데이터 해쉬값 변환 및 이미지 변환	41
[그림 26] 패밀리별 대표 이미지와 입력한 악성코드 이미지간의 유사도 결과	42
[그림 27] Json 파일 예시	43
[그림 28] 행위 분석 결과 예시	44
[그림 29] 파라미터 분석 결과 예시	44
[그림 30] 유사도 분석 결과 예시	45

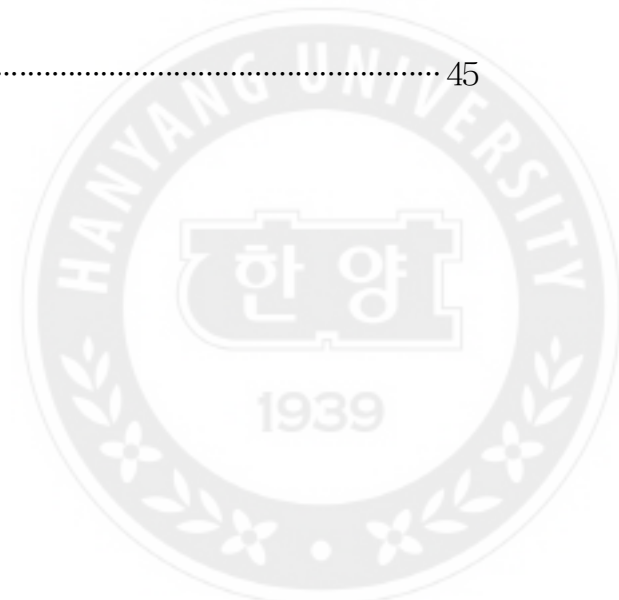


표 차례

[표 1] 공통 프로세스	20
[표 2] 알고리즘 비교 실험 입력 데이터 셋	27
[표 3] 알고리즘 속도 비교 입력 데이터 셋	29



국 문 요 약

인터넷은 시대가 거듭할수록 점점 발달되면서 그 영향력도 매우 커지고 있다. 이에 따라 부작용들도 나타나고 있는 가운데, 그 중 하나가 악성코드에 의한 피해 사례이다. 이러한 피해 사례 증가의 의미는 변종 악성코드들의 수가 급증하는 것과 관계가 있다. AV-TEST[1]의 2015년 조사 자료에 의하면 매일 220,000여개의 악성코드가 지속적으로 발견되고 있으며, 종류 또한 다양해지고 있다고 한다. 이와 같이, 악성코드의 수가 증가하고 있는 이유 중 하나는 악성코드 제작자가 광범위한 인터넷 보급을 이용하여 경제적, 금전적으로 이득을 취할 수 있기 때문이다. 초기의 악성코드 제작자의 목적은 자신의 해킹 기술 과시 및 호기심으로 시작하였다. 그러나 현재의 악성코드 제작자의 목적은 경제적, 금전적으로 이득을 취하는 것이 목적이 되었다.

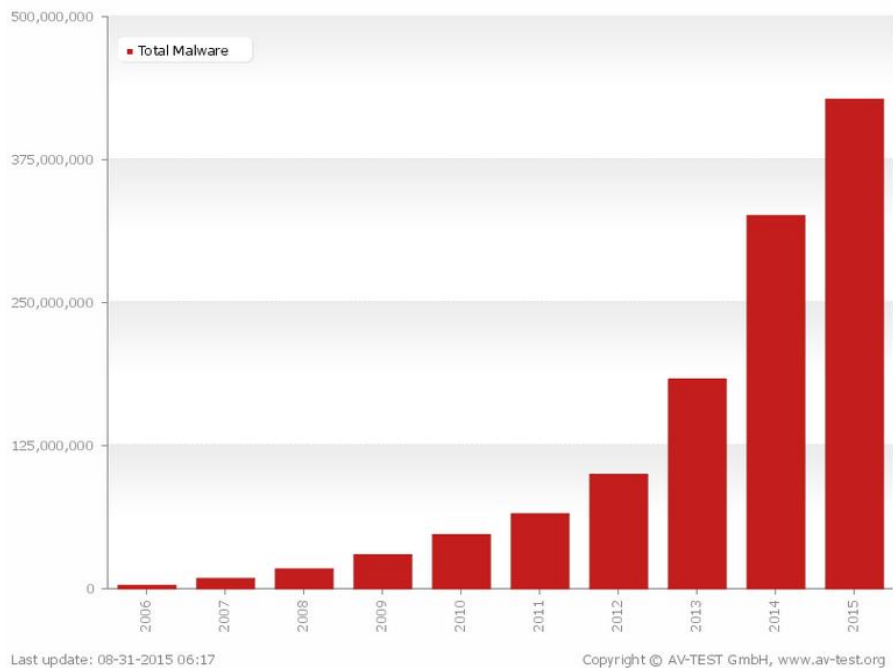
매년 수가 급격히 증가하고 그 수법도 매우 고도화되는 악성코드에 대항하기 위해 여러 악성코드 탐지 기법, 분류 기법이 연구되어 왔다. 본 논문에서는 악성코드 탐지 및 분류를 위하여 동적 분석 기법 중 악성코드의 API 호출 시퀀스 기반으로 분석하는 기법을 소개한다. 분석 방법으로는 N-gram 기법을 이용하여 API 호출 시퀀스의 서브 API 호출 시퀀스 빈도수를 추출하고, 추출된 빈도수를 이용하여 네 가지 유사도 계산식으로 악성코드 간의 유사도 값을 측정한다. 측정된 유사도 결과를 토대로 네 가지 유사도 계산식 비교분석하며, 변종 악성코드에 대한 악성코드 패밀리 분류 가능성을 분석한다.



제 1 장 서 론

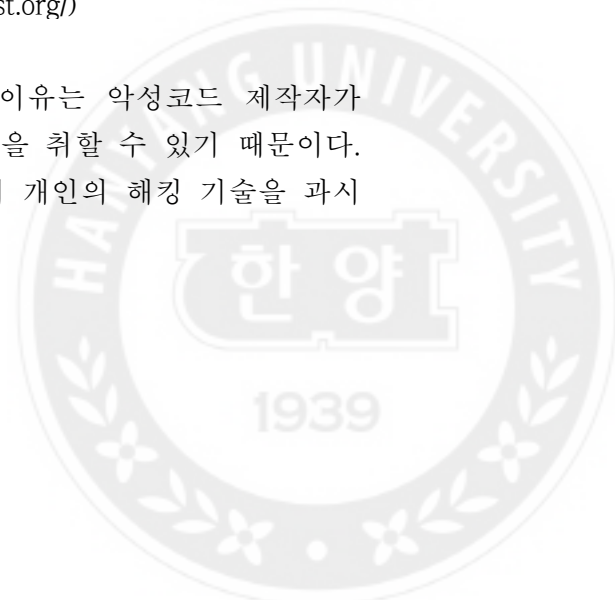
제 1 절 연구의 필요성

모리스 웜이 처음 등장한 이래로 매해 꾸준히 악성코드의 수는 증가하고 있다. The AV-TEST 기관의 조사에 의하면 매일 220,000여개의 악성코드가 계속적으로 발견되고 있다고 한다. 아래 [그림 1]은 AV-TEST기관에서 보고하고 있는 최근 10년간 악성코드 증가 추이를 나타낸다.



[그림 2] 최근 10년간 악성코드 증가 추이
(AV-TEST, <http://www.av-test.org/>)

이러한 증가 추세가 계속적으로 이어지는 이유는 악성코드 제작자가 바로 PC의 악성코드 감염으로 인한 경제적 이득을 취할 수 있기 때문이다. 처음 악성코드가 등장하였을 때는 감염의 목적이 개인의 해킹 기술을 과시



하거나 단순 호기심에 의한 제작이 대부분이었다. 그러나 현재는 블랙마켓이 형성되고 여러 범죄조직이 연루되어 금전적 이득을 취하기 위한 수단으로 악성코드가 이용된다. 또한, 최근 해외에서 극성을 부린 랜섬웨어라는 악성코드는 PC에 존재하는 임의 문서자료에 암호를 걸어 놓고 돈을 요구하는 식의 공격을 수행하였다. 암호를 풀려면 400달러나 400유로에 해당하는 비트코인(온라인 가상화폐)을 3일 안에 지급해야 한다는 협박 메시지도 포함됐다. 은행과 생명보험사, 증권사 등 8개 금융사에 설치된 PC 20여대에서 랜섬웨어인 크립토락커(cryptolocker)가 동시에 발견됐다고 한다.

이와 같이 매년 수가 급격히 증가하고 그 수법도 매우 고도화되는 악성코드에 대응하기 위해 기존에는 여러 악성코드 탐지 기법이나 분류 기법이 연구되어왔다. 악성코드 분석 기법은 크게 동적 기법과 정적 기법으로 구분할 수 있다. 본 논문에서는 동적으로 악성코드 분석을 수행하여 악성코드 탐지 및 분류하기 위한 기법을 소개하며, 소개한 기법을 이용하여 실험 및 분석한다.



제 2 장 연구 내용

매년 수가 급격히 증가하고 그 수법도 매우 고도화되는 악성코드에 대응하기 위해 기존에는 여러 악성코드 탐지 기법이나 분류 기법이 연구되어왔다.

본 논문에서는 대량의 악성코드 분류를 위해 효율적인 기법을 분석하고 악성코드 분석가가 악성코드 분석 시 도움을 주는 분석 시스템을 제안한다.

제안하는 기법은 기존에 사용하는 악성코드 동적 분석 도구인 Cuckoo sandbox[2]를 이용하여 동적 분석 정보를 추출한다. 악성코드 분류를 위해 추출된 정보에서 API 및 파라미터를 분석하여 유용한 정보를 분석한다. 이러한 정보를 이용하여 악성코드간의 유사도를 측정하고, 행위 정보를 분석한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 기술 및 연구에 대해 설명한다. 3장에서는 본 연구의 악성코드간의 유사도 측정 방법에 대해 설명하고 4장에서는 앞에서 설명한 기법을 이용하여 유사도 측정 기법 분석 및 비교한 연구 내용을 설명하고, 5장에서 개발한 시스템에 대하여 소개한다. 6장에서는 본 논문에서 개발한 시스템을 통해 악성코드간의 유사도 측정 실험 및 결과를 설명하며, 마지막 7장에서는 본 연구의 결론 및 향후 연구에 대한 내용으로 논문에 대한 결론을 짓는다.



제 3 장 배경 지식 및 관련 연구

제 1 절 API

API는 Application Programming Interface의 약자로 응용 프로그램에서 사용할 수 있도록, 운영 체제나 프로그래밍 언어가 제공하는 기능을 제어할 수 있게 만든 인터페이스를 뜻한다. 운영체제나 C, C++, Pascal 등과 같은 언어로 응용 프로그램을 만들 때, 윈도우를 만들고 파일을 여는 것과 같은 처리를 할 수 있도록 1,000여 개 이상의 함수로 구성되어 있다. API는 프로그래머를 위한 운영체제나 프로그램의 인터페이스로서 사용자와 직접 대하게 되는 그래픽 사용자 인터페이스나 명령형 인터페이스와 뚜렷한 차이가 있다. 또한, API는 응용 프로그램이 운영체제나 데이터베이스 관리 시스템과 같은 시스템 프로그램과 통신할 때 사용되는 언어나 메시지 형식을 가지며, API는 프로그램 내에서 실행을 위해 특정 서브루틴에 연결을 제공하는 함수를 호출하는 것으로 구현된다. 그러므로 하나의 API는 함수의 호출에 의해 요청되는 작업을 수행하기 위해 이미 존재하거나 또는 연결되어야 하는 몇 개의 프로그램 모듈이나 루틴을 가진다.



제 2 절 Cuckoo sandbox

Cuckoo sandbox는 악성코드 분석 시스템이다. 즉, 의심스러운 파일을 Cuckoo sandbox에 입력하게 되면 독립된 환경 내에서 악성코드를 실행하며, 악성코드가 실행되는 동안 무슨 행위를 했는지 분석하고 그 결과를 사용자에게 제공한다.

Cuckoo sandbox 분석 결과에는 악성코드가 실행되는 동안에 생성되는 프로세스 및 해당 프로세스가 호출하는 API, 파라미터 정보를 추출하여 로그에 기록한다. 또한, 파일을 생성 및 삭제와 같은 특정 이벤트가 발생할 경우에도 로그에 기록하게 된다.

본 논문에서는 Cuckoo sandbox를 이용하여 악성코드를 동적으로 분석하고, 분석 결과를 이용하여 연구를 진행한다.

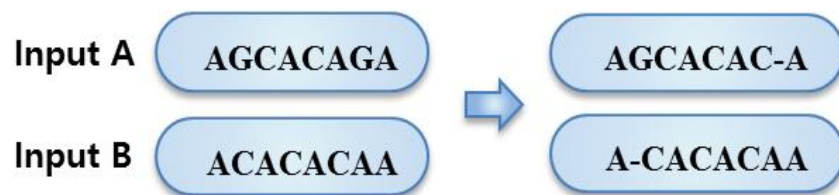


제 3 절 Smith-Waterman Algorithm

T F Smith & M S Waterman이 제안한 Smith-Waterman 알고리즘[3]은 2개의 DNA 또는 단백질서열 간의 유사한 패턴을 찾기 위한 서열정리를 수행하기 위한 동적 프로그래밍 알고리즘이다.[10] 이 알고리즘은 1981년 T.Smith 와 M. Waterman에 의해서 제안되었다.

이 알고리즘은 2단계로 이루어졌다. 첫 번째는 유사성 행렬 점수를 계산하는 것이고 두 번째는 유사성 행렬을 역추적해서 최적의 정렬을 탐색하는 것이다.

2개 서열이 A, B의 경우 2차원 배열을 동적으로 구하여 배치점수가 높은 부분 배치를 찾는다. 그리고 부분배치가 높은 부분을 기점을 통하여 A와 B를 정렬하면 다음과 [그림 2]와 같다.



[그림 3] 서열 정렬 예시



제 4 절 관련 연구 동향

악성코드를 탐지하고 분류하기 위한 분석 방법으로는 행위를 분석하는 동적 분석 방법과 코드를 분석하는 정적 분석 방법이 있다. 동적 분석은 가상환경에서 악성코드를 실행하여 실시간으로 모니터링하고 모니터링된 결과를 분석하는 방법이다. 정적 분석은 악성코드를 실행하지 않고 바이너리 자체의 구성 요소들이나 인스트럭션의 호출 관계 등을 분석하고 DLL 등을 파악하여 분석한다. 정적 분석 방법은 주로 리버싱에서 사용된다.

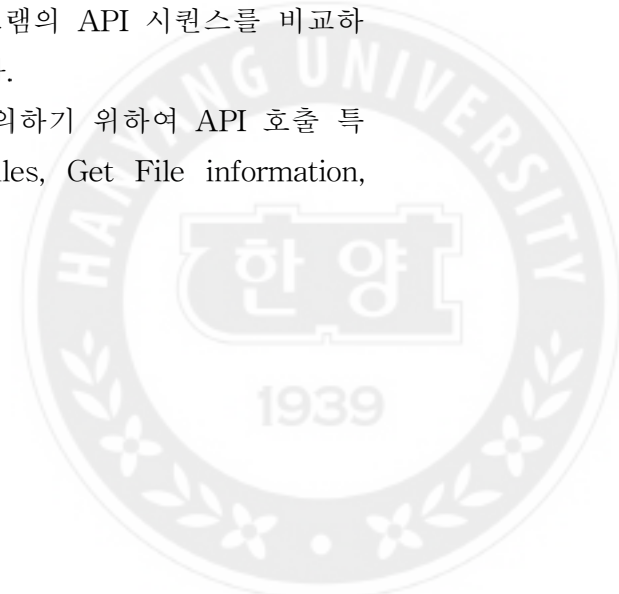
동적 분석으로 악성코드 탐지 기법의 가장 중요한 점은 악성코드 행위에 대한 정의이다. 악성코드 행위에 대한 정의를 위한 관련 논문들은 아래와 같다.

K. Rieck et al.[4]는 악성코드의 행위에 대하여 자동 분류를 위한 학습 기반으로 접근하는 연구를 진행하였다. 다양한 악성코드 샘플로부터 행위를 추출하기 위하여 4가지 단계별로 연구를 진행하였으며, 4가지 단계로는 Data Acquiring, Behavior Monitoring, Feature Extraction, Learning and Classification이 있다.

Yilun Wu et al [5]는 네트워크 악성코드 분석 시스템을 개발하였으며, 분석 시스템에 사용되는 분석 방법으로 동적 바이너리 분석과 동적 테인트 분석 기법을 이용 하였다.네트워크 악성코드의 행위인 자기 복제, 자기 번식 행위에 대한 API 및 파라미터를 정의하여 실제 시스템의 가상환경에서 실행하여 모니터링 후 이러한 행위가 나타나는지 분석하였다.

LIU Wu, et al.[6]는 악성 행위를 수행하는 API 시퀀스들을 Behavior Operation Set으로 크게 File Action, Process Actions, Network Actions, Registry Actions 네 가지로 분류하였고, 악성행위 및 행위에 대한 위험도를 식으로 표현하였다. 이 식을 이용하여 프로그램의 API 시퀀스를 비교하여 악성코드를 탐지하고 분류하는 연구를 하였다.

Mamoun Alazab, et al.[7]은 악성 행위를 정의하기 위하여 API 호출 특징을 Search Files to Infect, Copy, Delete Files, Get File information,



Move Files, Read/Write Files, Changes File attributes와 같이 6가지 항목으로 분류하였으며, 악성 행위 분석에는 API 호출의 빈도수, 시퀀스 패턴이 고려되었다. 이를 바탕으로 실험한 결과, 일반적인 악성코드에서는 Search files to infect와 Read/Write Files가 가장 많은 것으로 나타났다.

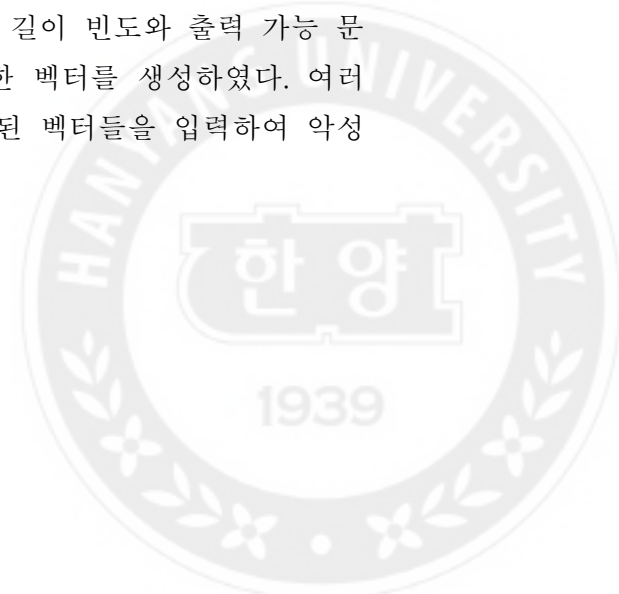
악성코드 행위 정의를 이용하여 악성코드 분류를 위한 연구는 아래와 같다.

Liangboonprakong et al.[8]은 바이너리 파일로부터 특성을 추출하여 N-gram과 여러 classification model을 이용하여 악성코드를 분류하였다. 유사도 분석 시 부하를 줄이기 위하여 Feature를 추출 한다. 이 후 악성코드에서 중요한 역할을 하는 Feature를 제외한 다른 Feature들을 비교대상에서 제거한다. 추출한 특성들을 Decision Tree, Artificial Neural Network (ANN), Support Vector Machine (SVM)) 알고리즘을 이용하여 악성코드를 분류하는 연구를 진행하였다.

Zhong, Y et al.[9]은 바이너리 파일 내 함수들의 특징을 추출고, 이를 기준으로 악성코드를 분류하였다. 함수의 특성으로는 함수 내부의 API 호출 정보, 함수에서 사용한 문자열 정보, 함수의 basic block이 있다. N-gram을 이용하여 각 함수들의 opcode의 빈도수를 추출하였고, one-class Support Vector Machines(SVMs) 알고리즘에 추출된 특성들을 적용하여 악성코드를 분류하였다.

Kang, B et al.[10]은 악성코드 분류 시 부하를 줄이기 위한 Major Block Comparison(MBC) 방식을 제안하였다. 이 방법은 악성코드 패밀리의 특성을 반영하는 부분만을 추출하여 바이너리 파일을 함수 단위로 나누고, 각 함수를 다시 block으로 나눈 후 다른 함수를 호출하는 block을 major block으로 인식한다. 이 후 major block만을 이용한 N-gram 기반 유사도 비교를 통해 악성코드를 분류하여 이를 증명하는 연구를 진행하였다.

Islam, R et al.[11]은 바이너리 파일에서 함수 길이 빈도와 출력 가능 문자열 정보를 추출하여 바이너리의 특성을 반영한 벡터를 생성하였다. 여러 머신 러닝 알고리즘을 제공하는 WEKA에 생성된 벡터들을 입력하여 악성코드들을 분류하는 연구를 진행하였다.

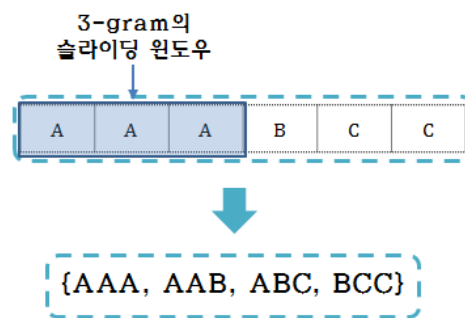


제 4 장 악성코드 유사도 측정 기법

제 1 절 N-gram

N-gram[12]은 컴퓨터 언어학에서는 텍스트 또는 음성을 주어진 순서에 따라 N개의 묶음으로 나누어 문맥 등을 분석하는데 쓰인다. 여기서 N은 사용자의 기준에 따라 자유롭게 결정할 수 있다.

이 기법은 대표적인 확률적 언어 모델의 하나로써, N개의 연쇄 단어를 확률적으로 표현하여 다음 단어를 확률적으로 예측하는데 사용된다. 기본적으로 [그림 3]과 같이 특정 길이의 슬라이딩 윈도우가 이동하면서 서브스트링을 추출한다.



[그림 4] N-gram 슬라이딩 윈도우에 따라 추출되는 서브스트링

예를 들어 “Malware”라는 문자열이 존재하는 경우, 1byte를 기준으로 3-gram을 적용한다면, “Mal”, “alw”, “lwa”, “war”, “are” 라는 5가지의 서브스트링들이 각각 빈도수 1로 생성된다. N-gram을 이용한 악성코드 탐지 기법은 이러한 하위 문자열들의 빈도수를 시그니처로 사용하여 탐지한다. 프로그램 전체를 이진 바이너리 파일로 표현한 후, N에 따라 크기 N의 하위 바이너리 문자열을 추출한 것이 시그니처로 사용된다.

본 논문에서는 각각의 API 이름을 사용하는 것이 아니라, API 이름을 각각 AA, AB 등과 같이 유니크한 코드로 맵핑한 API Code를 이용하여 시퀀스를 구성하고, 이를 N-gram에 적용하였다. 예를 들어 “AABAAC” 라

는 API 코드 시퀀스가 존재할 경우, N을 4로 정의하면 “AABA”, “BAAC”라는 서브스트링으로 나누어지며 각각 1의 빈도수를 가지게 된다. 그 결과 API 코드 시퀀스에 4-gram을 적용한 결과는 [(AABA:1), (BAAC:1)] 이라는 벡터로 추출된다. 이러한 결과 벡터는 악성코드 유사도 측정 시 사용된다.



제 2 절 Cosine Similairity

Cosine Similarity[13]는 데이터마이닝의 거리 측정 알고리즘으로 널리 사용된다. 주어진 두벡터의 벡터 스칼라 곱을 통해 벡터간의 거리를 계산하는 방법이다. 결과 값으로는 0~1사이 값이 나온다. 결과 값이 1이 나오는 경우는 두 샘플이 사용되는 API의 빈도수가 같으며, N의 값이 클 경우, API의 순서도 유사하다고 결론 내릴 수 있다. 본 연구에서 사용되는 Cosine Similarity 계산식은 아래 수식과 같다.

$$Similarity = \cos(\theta) = \frac{A \cdot B}{||A|| ||B||} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$

수식 1. Cosine Similarity

Cosine Similarity 계산 방법은 위에서 예로 들었던 API 코드 시퀀스의 4-gram 결과 값인 [(AABA:1), (BAAC:1)]를 하나의 벡터로 취급한다. 만약 다른 API 코드 시퀀스의 4-gram 결과 값이 [(AABA:1), (BAAC:2)]일 경우, 다음과 같이 두 시퀀스 간의 Cosine Simlarity 공식을 이용하여 유사도를 측정할 수 있다. 분모는 각각의 벡터 값의 원소들을 제곱하여 더한 값에 대하여 제곱근을 계산한다. 위 예제 데이터를 이용한다면, 가 되는 것이다. 분자는 두 벡터 값의 원소들 중 같은 서브스트링이 존재한다면 그 빈도수를 곱하여 더한 값이다. 즉, A벡터와 B벡터 중 같은 값인 “AABABA”의 곱인 1*1 이며, 나머지 두 원소는 같지 않으므로 계산하지 않는다. 따라서 두 개의 예제 시퀀스 사이의 Cosine Similarity 측정값은 (약 0.633)이 된다.



제 3 절 Overlap Coefficient

Overlap Coefficient[14]는 두 집합사이의 중첩을 측정하는 Jaccard 인덱스와 관련된 기법이며, Overlap Coefficient는 두 집합의 합집합 부분의 크기를 두 집합 중 크기가 작은 것으로 나눈 값으로 결과 값이 정의된다. Overlap Coefficient의 결과 값으로는 0~1 사이의 값이 나오며, 결과 값이 1이 나오는 경우에는 두 샘플이 포함관계에 있다고 정의할 수 있다. 본 연구에서 사용되는 Overlap Coefficient의 계산식은 아래 수식과 같다.

$$Similarity = \frac{|A \cap B|}{\min(|A|, |B|)}$$

수식 2. Overlap Coefficient

Overlap Coefficient 계산 방법은 위에서 예로 들었던 API 코드 시퀀스의 4-gram 결과 값인 [(AABA:1), (BAAC:1)] 와 다른 API 코드 시퀀스의 4-gram 결과 값 [(AABA:1), (BAAC:2)]에 대한 두 시퀀스 간의 Overlap Coefficient 측정은 다음과 같다. 분모는 두 시퀀스 각각의 길이 중 더 작은 길이가 분모가 된다. 위 예제 데이터를 이용한다면, 첫 번째 벡터가 두 번째 벡터보다 길이가 작기 때문에 첫 번째의 벡터의 길이가 분모가 된다. 분자는 두 벡터 값의 원소들 중 같은 서브스트링이 존재한다면 그 중 작은 빈도수를 제공하여 더한 값이다. 따라서 두 개의 예제 시퀀스 사이의 Overlap Coefficient 측정값은 1이 된다.



제 4 절 Jaccard Index

Jaccard similarity coefficient라고 잘 알려진 Jaccard index[15]는 Paul Jaccard가 제안한 유사성 계산 알고리즘이다. 주로 샘플들의 유사성과 다양성을 비록하기 위해 사용되는 통계이다. 기본적으로 Jaccard similarity 측정 방법은 두 샘플 집합의 교집합을 합집합으로 나눈 것으로 정의한다. Jaccard index의 결과 값으로는 0~1 사이의 값이 나오며, 결과 값이 1이 나오는 경우에는 두 샘플이 포함관계에 있다고 정의할 수 있다. 본 연구에서 사용되는 Jaccard index의 계산식은 아래 수식과 같다.

$$Similarity = \frac{|A \cap B|}{|A \cup B|}$$

수식 3. Jaccard Index

Jaccard index 계산 방법은 위에서 예로 들었던 API 코드 시퀀스의 4-gram 결과 값인 [(AABA:1), (BAAC:1)] 와 다른 API 코드 시퀀스의 4-gram 결과 값이 [(AABA:1), (BAAC:2)]에 대한 두 시퀀스 간의 Jaccard Index 측정은 다음과 같다. 두 벡터의 합집합의 길이가 분모가 된다. 분자는 두 벡터 값의 원소들 중 같은 서브스트링이 존재한다면 그 중 작은 빈도수를 제공하여 더한 값이다. 따라서 두 개의 예제 시퀀스 사이의 Jaccard Index 측정값은 약 0.6325이 된다.



제 5 절 Sorensen-Dice

Sorensen-Dice[16]는 보통 Sorensen index 또는 Dice's coefficient라고 불린다. Sorensen-Dice 는 Jaccard index와 형태가 매우 비슷하다. Sorensen-Dice의 결과 값으로는 0~1 사이의 값이 나오며, 결과 값이 1이 나오는 경우에는 두 샘플이 포함관계에 있다고 정의할 수 있다. 본 연구에서 사용되는 Sorensen-Dice의 계산식은 아래 수식과 같다.

$$Similarity = \frac{2|A \cap B|}{|A| + |B|}$$

수식 4. Sorensen-Dice

Sorensen-Dice 계산 방법은 위에서 예로 들었던 API 코드 시퀀스의 4-gram 결과 값인 [(AABA:1), (BAAC:1)] 와 다른 API 코드 시퀀스의 4-gram 결과 값이 [(AABA:1), (BAAC:2)]에 대한 두 시퀀스 간의 Jaccard Index 측정은 다음과 같다. 두 벡터의 길이를 합한 값이 분모가 된다. 분자는 두 벡터 값의 원소들 중 같은 서브스트링이 존재한다면 그 중 작은 빈도수를 제공하여 더한 후 2배한 값이다. 따라서 두 개의 예제 시퀀스 사이의 Sorensen-Dice 측정값은 약 0.775이 된다.



제 6 절 파라미터

API 이름뿐만 아니라 호출 시 사용된 API의 파라미터를 활용한 악성코드 분석 방법론을 연구하였다. 파라미터의 유형에 따라 악성행위에 가능성을 조사하는 연구가 진행하였다. 먼저, API가 사용하는 파라미터의 유형들을 분석하기 위하여 특정 행위를 하는 악성코드 패밀리를 선정하고, 파라미터를 추출하여 이를 비교하였다. 그러나 악성코드가 특정 행위를 위해서 호출하는 특정 파라미터에 대해서는 명확한 특징을 정의할 수 없었다. 따라서 이에 대한 대책으로는 16개 패밀리별 파라미터에 대해 먼저 조사한 후, 이를 바탕으로 입력된 악성코드의 파라미터가 16개 패밀리에서 나타나는 파라미터와 얼마나 유사한지를 측정하는 연구로 진행하였다.



제 7 절 고려 사항 : 반복 패턴 제거

기본적으로 서열 정렬 기법의 성능은 분석 대상의 길이에 많은 영향을 받으며 N-gram과 비교하여 매우 오랜 시간이 소요된다. 그 이유는 행렬 구성에 요구되는 시간이 많기 때문이다. 따라서 행렬 구성에 요구되는 시간을 단축하기 위하여 행렬의 크기 자체를 축소하는 방법이 고려될 수 있다. 이를 위해 행렬 크기와 대응되는 분석 대상 서열들의 길이를 단축할 수 있다. 이 때 본래 서열이 가지던 정보가 손실될 수 있으나, 서열의 길이를 감소시키더라도 분석 결과로서 추출되는 두 서열 간 유사도의 정확성은 최대한 유지되어야 한다. 이러한 서열 길이 단축을 위하여 반복 패턴 제거를 적용할 수 있다.

본 연구과제에서는 하나의 서열 안에서 2번 이상 연속되어 나타나는 부분을 가리켜 ‘반복 패턴’이라 정의한다. 예를 들어, 서열 “ACACACTA”의 경우, “AC”가 3번 연속으로 반복되어 나타나는데 이러한 반복 패턴을 제거하면 서열 “ACTA”가 추출된다.

이와 같이 반복 패턴 제거를 통한 접근은 두 서열 간의 유사한 부분 중 반복되는 부분을 제거하여도 유사도 측면에서는 큰 손실이 없다는 것에서 비롯된다. 또한 프로그램이 반복문을 수행하는 경우가 빈번하기 때문에 API 호출 리스트 역시 반복되는 부분이 많으며, 이러한 반복 패턴을 제거함으로써 매우 높은 성능 향상을 기대할 수 있다.



제 5 장 악성코드 유사도 측정 기법 분석 및 비교

본 장에서는 악성코드 분류에 앞선 선행 연구로서 유사도 측정 기법을 분석하고 기존의 유사도 측정 기법과 비교한다.

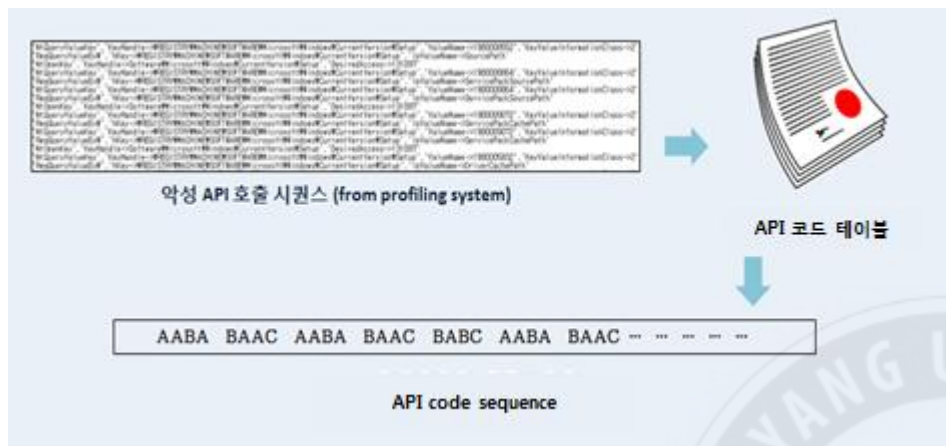
제 1 절 악성코드 유사도 측정 기법 분석

1. 유사도 측정 기법을 이용한 실험 과정

본 논문에서 제안하는 N-gram 알고리즘을 이용하여 악성코드의 실행에 따라 저장된 트레이스를 기반으로 API 호출 시퀀스를 추출하고, 각 API의 이름을 코드화하였다. 그리고 코드화된 API 호출 시퀀스를 이용하여 N-gram의 각 빈도수를 추출하여 유사도 측정에 사용하였다.

2. 코드 값 변환

본 실험에서 악성코드 동적 분석으로 추출한 각각의 API 이름을 그대로 사용하는 것이 아니라, API 코드 테이블을 참조하여 유니크한 코드로 맵핑한 후 시퀀스를 구성한다. 아래의 [그림 4]는 위 과정을 요약한 그림이다.



[그림 5] 악성 API 호출 시퀀스 코드화

제 2 절 최적해 N을 찾는 실험

본 논문에서 제안하는 N-gram 알고리즘은 N 값에 따라 API 호출 시퀀스 순서, API 호출 빈도수에 영향을 미친다. 따라서 N 값의 변화에 따른 측정 시간 및 유사도 측정값을 비교하여 최적해 N을 도출하는 실험하였다.

1. 입력 데이터 셋

본 실험에서 사용되는 데이터는 악성코드 프로파일링 시스템에서 제공한 악성코드 실행 트레이스 100개이며, 이 데이터들을 이용하여 실험하였다. 이때 악성코드 이름은 해시 값으로 이루어져 있고, 악성코드 패밀리 변종 관계는 주어지지 않았다. 또한, [그림 5]와 같이 트레이스 파일의 구성은 시간, PID(Process ID), 프로세스 경로와 이름, 호출된 API 정보 등을 포함하며, 약 10MB 내외의 크기이다.

```
"1409897572","624","672","C:\\WINDOWS\\system32\\csrss.exe","2","NtCreateFile",  
"FileHandle->0x000006BC","FileName->\\Device\\HarddiskVolume1\\WINDOWS\\system32\\ntdll.dll",  
"DesiredAccess->2148532352","FileAttributes->0","ShareAccess->1","CreateDisposition->1",  
"CreateOptions->96"  
:  
:
```

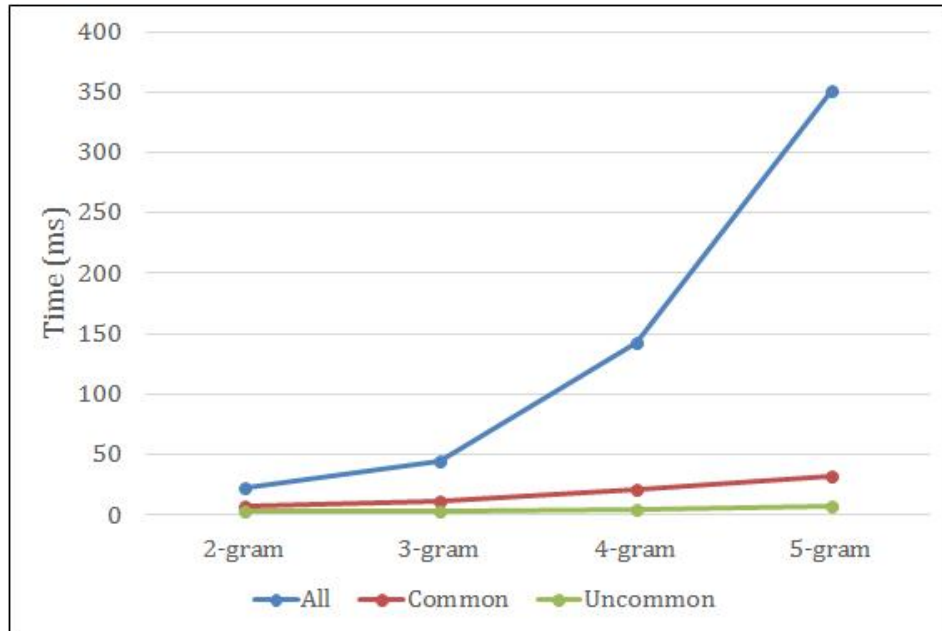
(7.31MB)

[그림 6] 최적해 실험에 사용된 트레이스의 예

2. 실험 결과

[그림 6]은 N의 크기에 따른 유사도 측정 시간 변화를 나타낸 것이다. N의 크기가 클수록 N-gram에 대한 Cosine Similarity를 계산하는 시간이 점차 증가하는 것을 확인할 수 있다.





[그림 7] N값에 따른 수행 시간 변화

3. 실험 결과

[그림 6]의 결과 그래프를 보면 알 수 있듯이, N의 값이 증가할수록 수행 시간이 급격하게 증가하는 것을 알 수 있다. 따라서 N의 값에 따라 유사도 측정 결과 값이 차이가 존재할 경우, 유사도 측정 정확도와 수행 시간 사이의 적절한 트레이드오프가 필요하다.



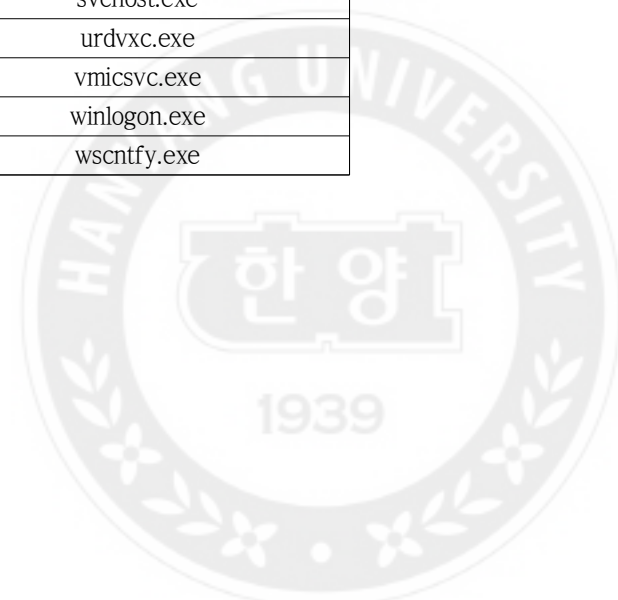
제 3 절 프로세스 관련 실험

입력 데이터인 악성코드 트레이스에서 악성코드의 영향을 받은 프로세스 뿐만 아니라, 운영체제에서 일반적으로 실행되는 프로세스가 호출하는 API 도 포함되어 있다. 따라서 다음과 같이 프로세스를 3가지로 나누어 실험하였다.

- o All of API Sequence
 - 모든 프로세스에 의한 API 호출 시퀀스
- o API Sequence called by Common Processes
 - 운영체제에서 일반적으로 실행되는 프로세스
 - [표 1]에 나타낸 16개 프로세스에 의해 호출된 API
- o API Sequence called by Uncommon Processes
 - Common Processes를 제외한 나머지 프로세스에 의해 호출된 API

No.	Path	Process
1	C:\Program Files\Java\jre6\bin\	java.exe
2	C:\WINDOWS\	explorer.exe
3	C:\WINDOWS\system32\	alg.exe
4		cmd.exe
5		conime.exe
6		csrss.exe
7		ctfmon.exe
8		lsass.exe
9		ntvdm.exe
10		services.exe
11		spoolsv.exe
12		svchost.exe
13		urdrvcs.exe
14		vmicsvc.exe
15		winlogon.exe
16		wscntfy.exe

[표 1] 공통 프로세스



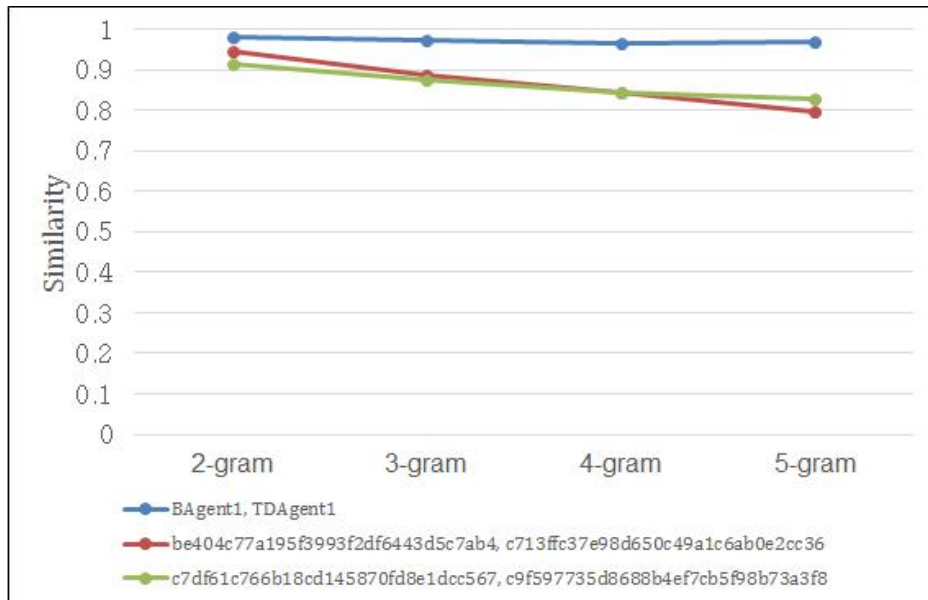
1. 입력 데이터 셋

본 실험에서 사용되는 데이터는 앞의 최적해 n 을 찾는 실험에서 사용되는 데이터와 동일하다.

2. 실험 결과

가. All of API Sequence

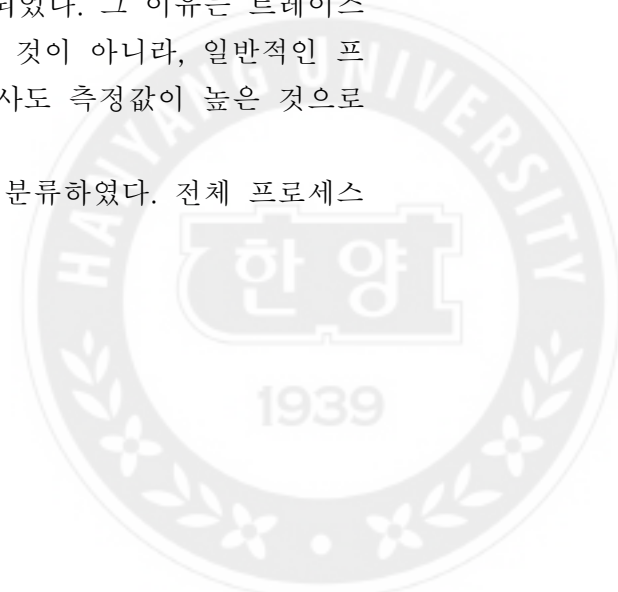
[그림 7]악성코드 실행에 의해 기록된 트레이스에서 모든 API 호출을 이용하여 시퀀스를 생성하고 유사도를 측정한 결과의 일부를 나타낸 것이다.



[그림 8] 모든 프로세스의 의한 API 호출 시퀀스 유사도 측정

첫 번째 악성코드 샘플 BAgent1과 TDAgent1은 서로 다른 패밀리의 샘플들임에도 불구하고, 유사도가 매우 높게 계산되었다. 그 이유는 트레이스에 악성코드의 실행에 의한 API 호출만 기록된 것이 아니라, 일반적인 프로세스에 의한 API 호출도 기록되기 때문에 유사도 측정값이 높은 것으로 판단된다.

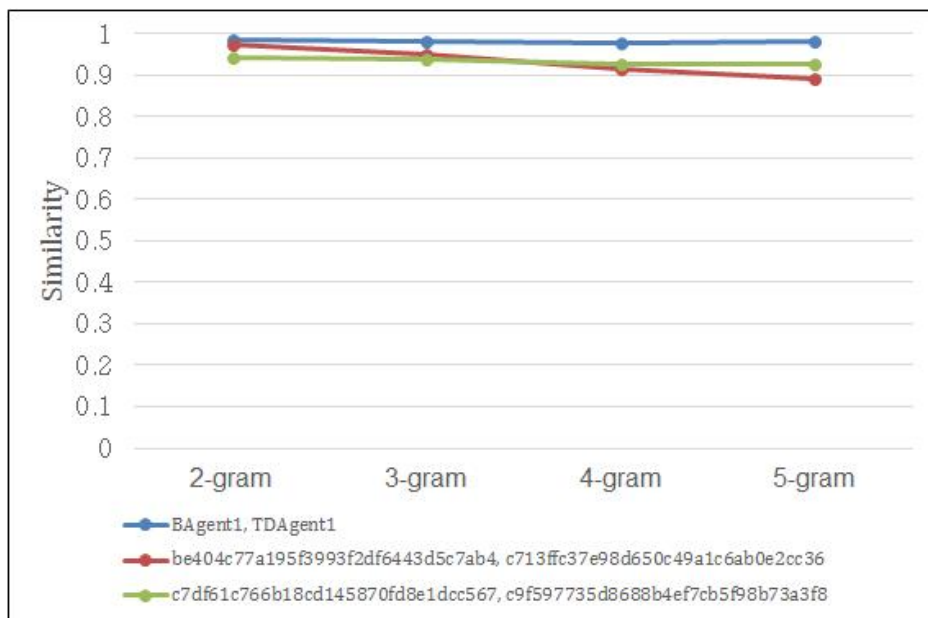
이에 따라 각 프로세스에 의해 호출된 API를 분류하였다. 전체 프로세스



에 대하여 운영체제에서 일반적으로 실행되는 프로세스와 그렇지 않은 프로세스로 구분하여 유사도를 측정하였다.

나. API Sequence called by Common Processes

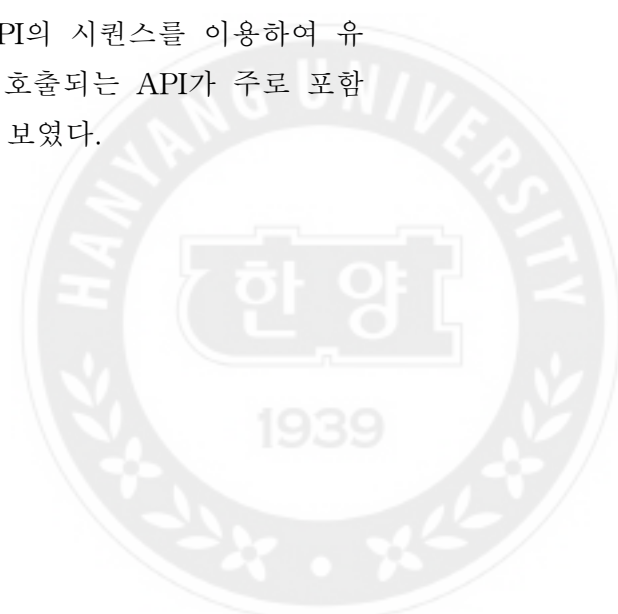
[그림 8]는 공통 프로세스에 의해 호출된 API의 시퀀스를 이용하여 유사도를 측정한 결과이다. 이는 svchost.exe csrss.exe 등과 같이 악성코드에 의해서 생성된 서브 프로세스들이 호출하는 API들이 대부분 유사하기 때문에 유사도가 높은 것으로 판단된다.

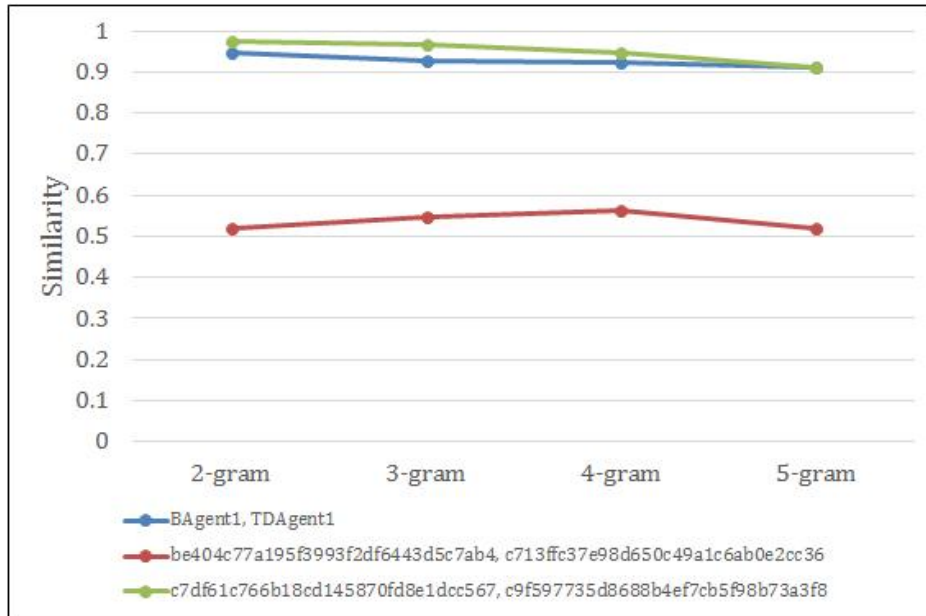


[그림 9] 일반적인 프로세스의 의한 API 호출 시퀀스 유사도 측정

다. API Sequence called by Uncommon Processes

[그림 9]은 비공통 프로세스에 의해 호출된 API의 시퀀스를 이용하여 유사도를 측정한 결과이다. 악성코드에 의해 직접 호출되는 API가 주로 포함되어 있기 때문에 유사도 수치에 확실한 차이를 보였다.



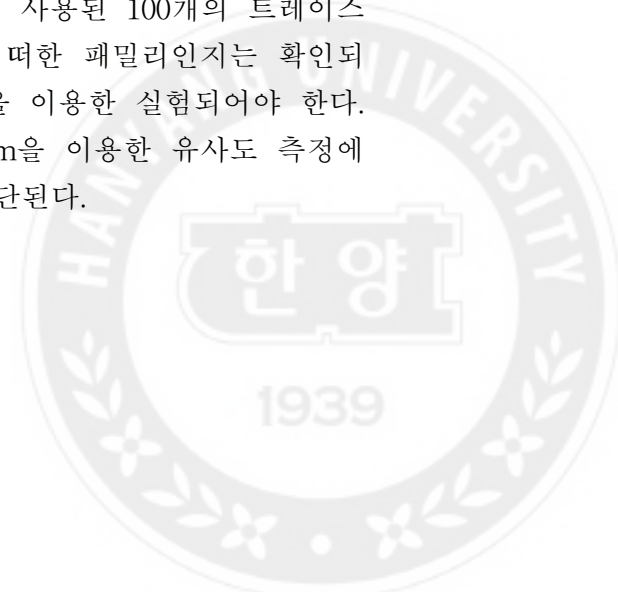


[그림 10] 일반적이지 않은 프로세스의 의한 API 호출 시퀀스 유사도 측정

결론적으로, 첫 번째 BAgent1와 TDAgent1의 유사도, 그리고 세 번째 c7df~와 c9f5~의 경우는 악성코드 패밀리 변종 여부와 상관없이 행위 자체가 유사한 악성코드 샘플들이고, 두 번째 be40~와 c713 사이 유사도의 경우는 행위 유사도가 상대적으로 낮은 악성코드 샘플임을 판단할 수 있다.

3. 결론

먼저 N의 값에 따라 유사도 값의 차이가 크지 않기 때문에 정확도는 N의 값에 영향을 받지 않는다고 할 수 있다. 그리고 앞의 세 가지의 경우로 나누어 수행한 프로세스 관련 실험을 통하여 트레이스에서 공통된 프로세스들을 제외하고, 그외의 프로세스를 이용하여 유사도 측정하는 방법이 더 정확한 것으로 판단된다. 그러나 현재 실험에서 사용된 100개의 트레이스 파일들에 대하여, 어떠한 악성코드인지 혹은 어떠한 패밀리인지는 확인되지 않았기 때문에 더욱 명확히 분류된 샘플들을 이용한 실험되어야 한다. 또한, API 시퀀스의 테이블을 작성하여 N-gram을 이용한 유사도 측정에 반영하면 보다 정확한 결과가 도출될 것으로 판단된다.



제 4 절 N의 값에 따른 수행 시간 변화 측정 실험

1. 입력 데이터 셋

본 실험에서 사용되는 데이터는 악성코드 프로파일링 시스템에서 제공한 악성코드 실행 트레이스 100개이며, 이 데이터들을 이용하여 실험하였다. 이때 악성코드 이름은 해시 값으로 이루어져 있고, 악성코드 패밀리 변종 관계는 주어지지 않았다. 또한, [그림 10]와 같이 트레이스 파일의 구성은 시간, PID(Process ID), 프로세스 경로와 이름, 호출된 API 정보 등을 포함하며, 약 10MB 내외의 크기이다.

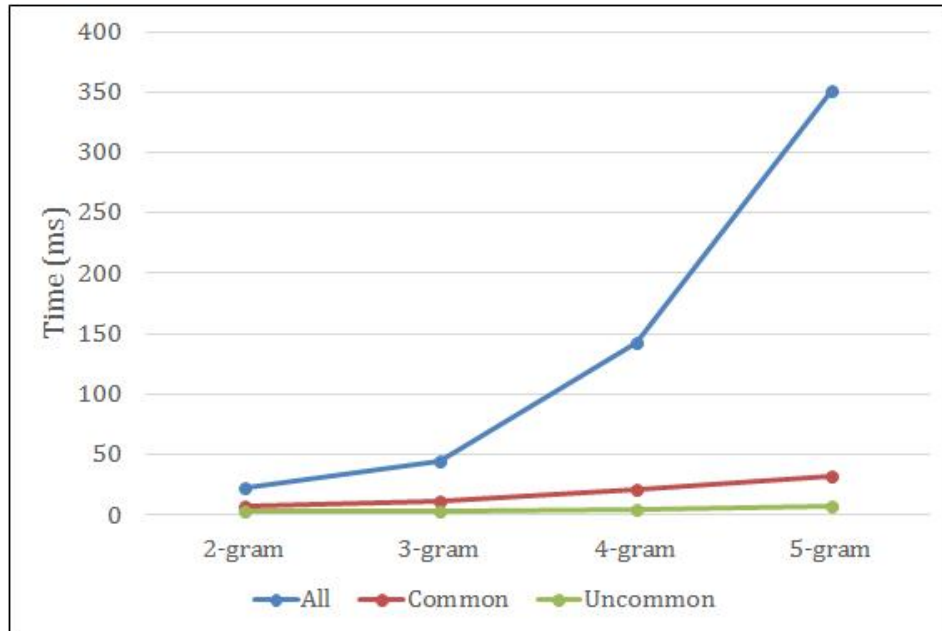
```
"1409897572", "624", "672", "C:\WINDOWS\system32\csrss.exe", "2", "NtCreateFile",  
"FileHandle->0x000006BC", "FileName->\Device\HarddiskVolume1\WINDOWS\system32\ntdll.dll",  
"DesiredAccess->2148532352", "FileAttributes->0", "ShareAccess->1", "CreateDisposition->1",  
"CreateOptions->96"  
:  
:  
:  
(7.31MB)
```

[그림 11] 최적해 실험에 사용된 트레이스의 예

2. 실험 결과

[그림 11]은 N의 크기에 따른 유사도 측정 시간 변화를 나타낸 것이다. N의 크기가 클수록 N-gram에 대한 Cosine Similarity를 계산하는 시간이 점차 증가하는 것을 확인할 수 있다.





[그림 12] N값에 따른 수행 시간 변화

3. 실험 결과

[그림 11]의 결과 그래프를 보면 알 수 있듯이, N의 값이 증가할수록 수행 시간이 급격하게 증가하는 것을 알 수 있다. 따라서 N의 값에 따라 유사도 측정 결과 값이 차이가 존재할 경우, 유사도 측정 정확도와 수행 시간 사이의 적절한 트레이드오프가 필요하다.

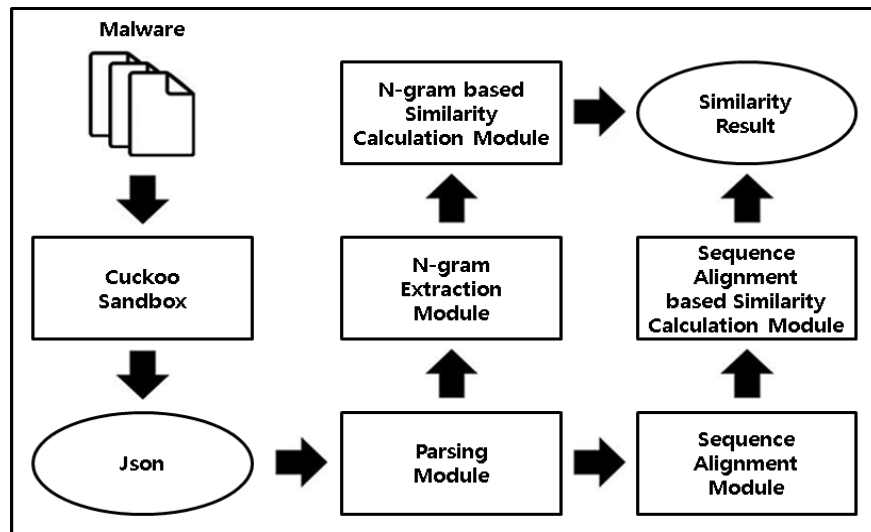


제 5 절 알고리즘 비교 실험

먼저, N-gram과 서열 정렬 기법 중 하나인 Smith-Waterman의 비교 실험을 하였다. N-gram은 API 호출 순서 및 API 호출 빈도수를 모두 반영하여 유사도 값을 측정지만, Smith-Waterman은 API 호출 순서만을 반영하여 유사도 값을 측정한다. 알고리즘 성능 비교 실험은 다음과 같은 2가지 방법으로 실험하였다.

1. 알고리즘 비교 실험 구성도

N-gram과 서열 정렬 기법 중 하나인 Smith-Waterman의 비교 실험을 위한 실험 구성도는 아래 [그림 12]와 같다.

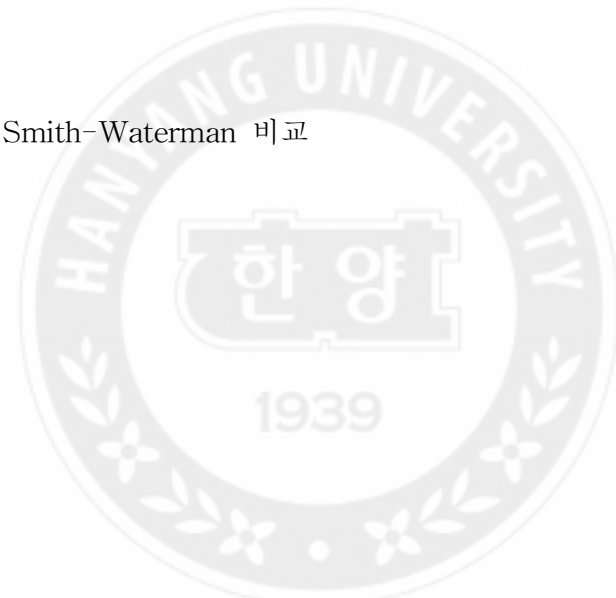


[그림 13] 알고리즘 비교 실험을 위한 실험 구성도

가. 패밀리별 유사도 측정 실험

- 패밀리가 명확한 샘플을 이용하여 N-gram, Smith-Waterman 비교

나. API 개수에 따른 수행 시간 측정 실험



- 입력 데이터의 크기에 따른 수행 시간 비교 진행

2. 패밀리별 유사도 측정 실험 입력 데이터 셋

본 실험에서 사용되는 데이터는 패밀리 10개이며, 각 패밀리별 15개의 샘플로 총 150개를 이용하여 실험을 진행하였다. 입력 데이터 셋은 아래 [표 2]와 같다.

[표 2] 알고리즘 비교 실험 입력 데이터 셋

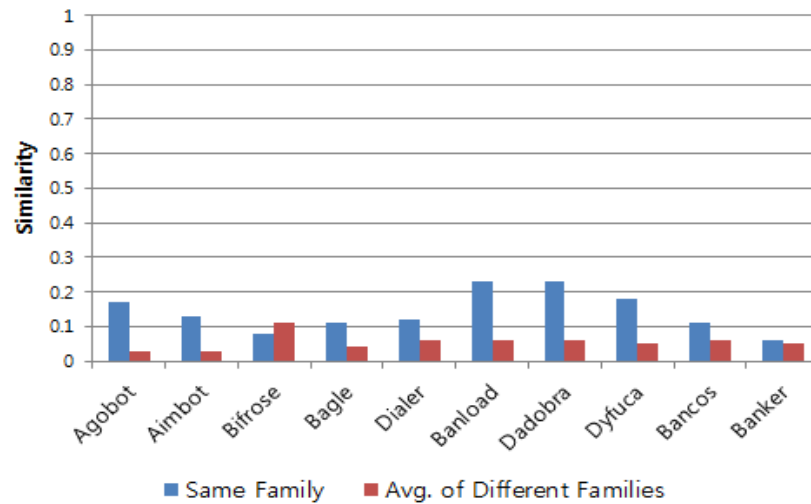
Malware Samples	
Agobot	Banload
Aimbot	Dadobra
Bifrose	Dyfuca
Bagle	Bancos
Dialer	Bannker

3. 실험 결과

가. N-gram

N-gram의 패밀리별 유사도 측정 실험 결과는 아래 [그림 13]와 같다.

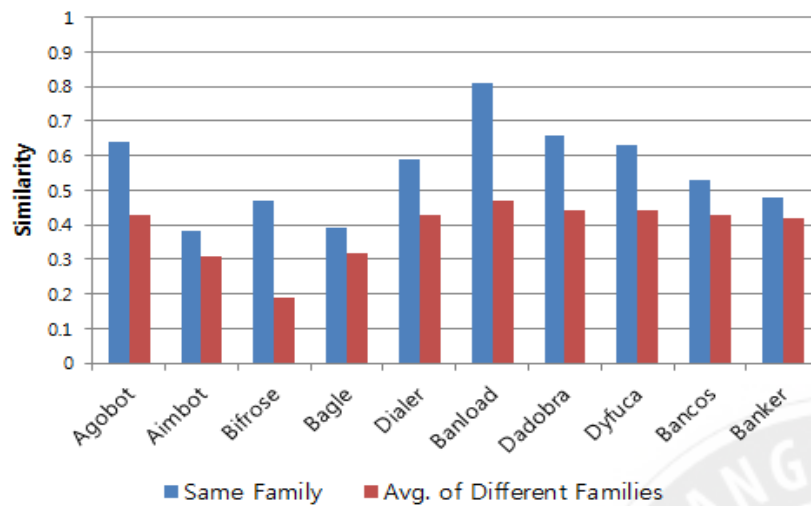




[그림 14] N-gram 패밀리별 유사도 측정 실험 결과

나. Smith-Waterman

Smith-Waterman의 패밀리별 유사도 측정 실험 결과는 아래 [그림 14]과 같다.



[그림 15] Smith-Waterman 패밀리별 유사도 측정 실험 결과



4. API 개수에 따른 수행 시간 측정 실험 입력 데이터 셋

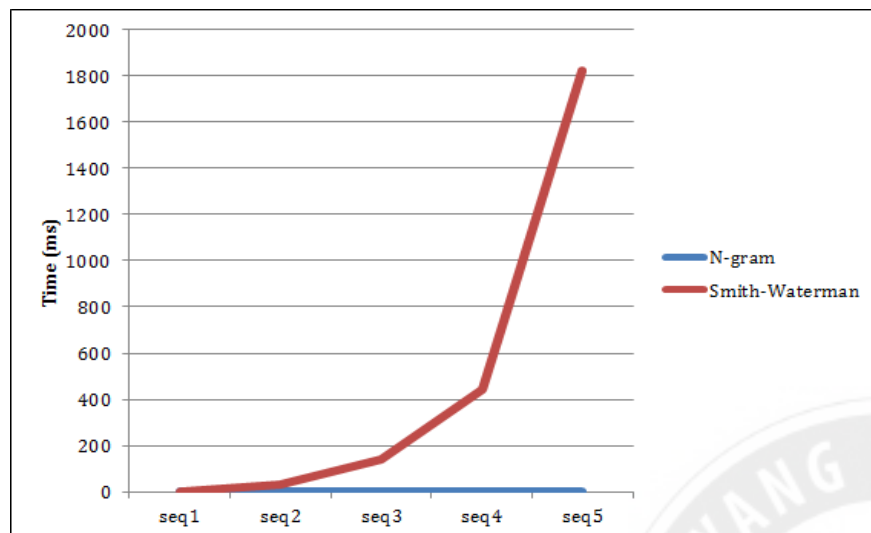
본 실험에서 사용되는 데이터는 API 개수를 1000개부터 1000개씩 늘려 총 5개의 샘플에 대하여 실험을 진행하였다. 입력 데이터 셋은 [표 3]과 같다.

[표 3] 알고리즘 속도 비교 입력 데이터 셋

Sequence	API 개수
Seq 1.	1000
Seq 2.	2000
Seq 3.	3000
Seq 4.	4000
Seq 5.	5000

5. 실험 결과

API 개수에 따른 N-gram과 Smith-Waterman의 수행 시간 측정 실험 결과는 아래 [그림 15]와 같다.



[그림 16] API 개수에 따른 수행 시간 측정 실험 결과

6. 결론

유사도 측정 실험 결과에서는 N-gram 에서는 10개의 악성코드 패밀리에 대하여 같은 패밀리 내의 악성코드 평균 유사도 값과 다른 패밀리와의 평균 유사도 값의 차이가 높게 나타났다. 마찬가지로 Smith-Waterman에서도 악성코드 패밀리 9개에 대하여 같은 패밀리 내의 악성코드 평균 유사도 값과 다른 패밀리와의 평균 유사도 값의 차이가 높게 나타났다. 하지만 Bifrose 패밀리의 경우 Smith-Waterman으로 유사도 측정 실험 결과 같은 패밀리 내의 악성코드 평균 유사도 값보다 다른 패밀리와의 유사도 평균 값이 높게 나타났다. 이 이유는 Bifrose 패밀리의 공통적인 API 시퀀스를 찾지 못하여 나타난 것으로 판단된다.

수행 시간 측정 실험 결과 N-gram은 API 개수가 늘어나더라도 시간 복잡도가 상대적으로 매우 낮으며, 그에 반해 Smith-Waterman의 경우 상대적으로 시간 복잡도가 매우 높다는 것을 확인하였다. 결과적으로는 악성코드 분류 성능에 대해서는 큰 차이가 없으나, 대량의 악성코드를 분석 및 분류할 경우 N-gram이 시간복잡도가 낮기 때문에 더 적합하다고 판단된다.



제 6 절 유사도 계산식 비교 실험

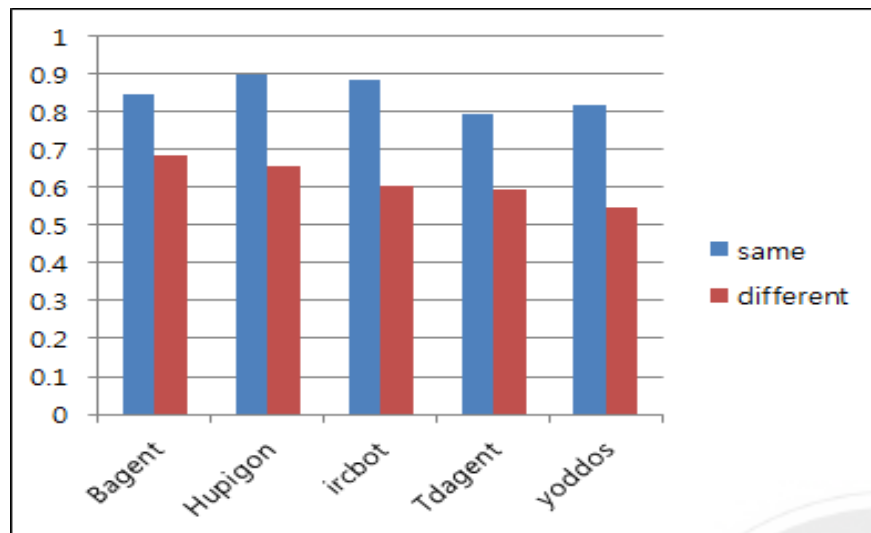
본 실험에서는 N-gram을 기반으로 빈도수를 추출하고, 추출된 빈도수를 앞에서 설명한 Cosine similariy, Overlap coefficient, Jaccard Index, Sorensen-Dice 총 4 가지의 유사도 계산식을 이용하여 비교하였다.

1. 데이터 셋

본 실험에서 사용한 데이터 셋은 5개의 패밀리어며, 샘플 수는 각 패밀리 별 20개씩 총 100개이다.

2. Cosine similarity

먼저, Cosine similarity를 이용하여 실험을 진행하였다. 실험 결과는 아래 [그림 16]와 같다.



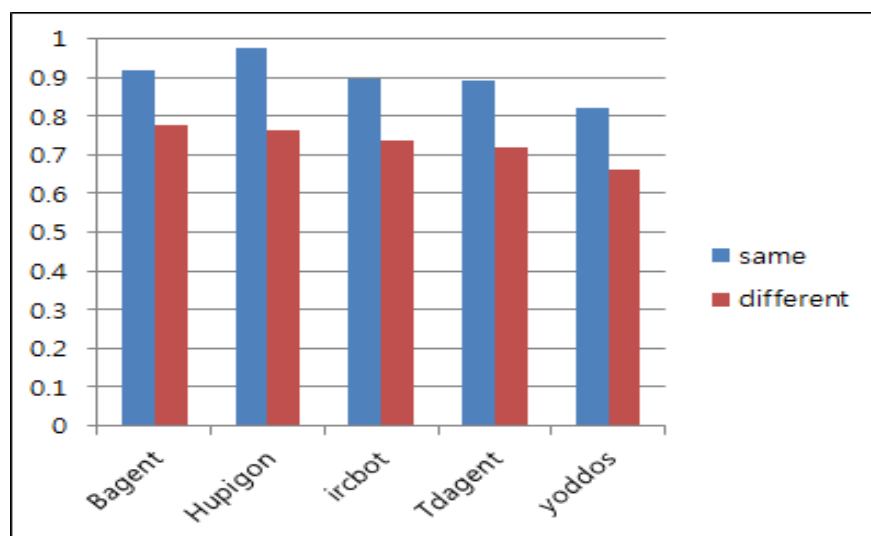
[그림 17] Cosine similarity를 이용한 실험 결과

위 [그림 16]에서 알 수 있듯이, 같은 패밀리의 악성코드 간의 평균 유사

도 값은 0.85로 나타났으며, 다른 패밀리와의 평균 유사도 값은 약 0.62로 나타났다.

3. Overlap coefficient

Overlap coefficient를 이용하여 실험을 진행하였다. 실험 결과는 아래 [그림 17]과 같다.



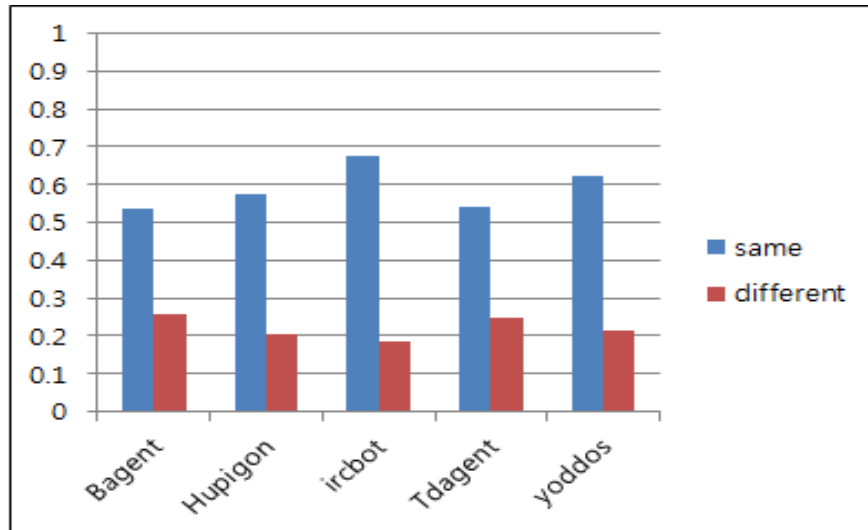
[그림 18] Overlap coefficient를 이용한 실험 결과

위 [그림 17]에서 알 수 있듯이, 같은 패밀리의 악성코드 간의 평균 유사도 값은 0.90로 나타났으며, 다른 패밀리와의 평균 유사도 값은 약 0.73로 나타났다.

4. Jaccard Index

Jaccard Index를 이용하여 실험을 진행하였다. 실험 결과는 아래 [그림 18]과 같다.





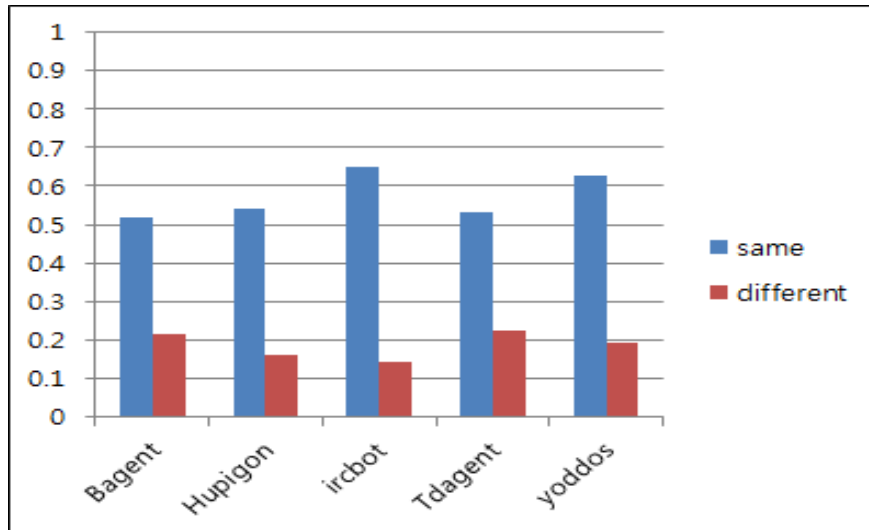
[그림 19] Jaccard Index를 이용한 실험 결과

위 [그림 18]에서 알 수 있듯이, 같은 패밀리의 악성코드 간의 평균 유사도 값은 0.59로 나타났으며, 다른 패밀리와의 평균 유사도 값은 약 0.22로 나타났다.

5. Sorensen-Dice

Sorensen-Dice를 이용하여 실험을 진행하였다. 실험 결과는 아래 [그림 19]과 같다.





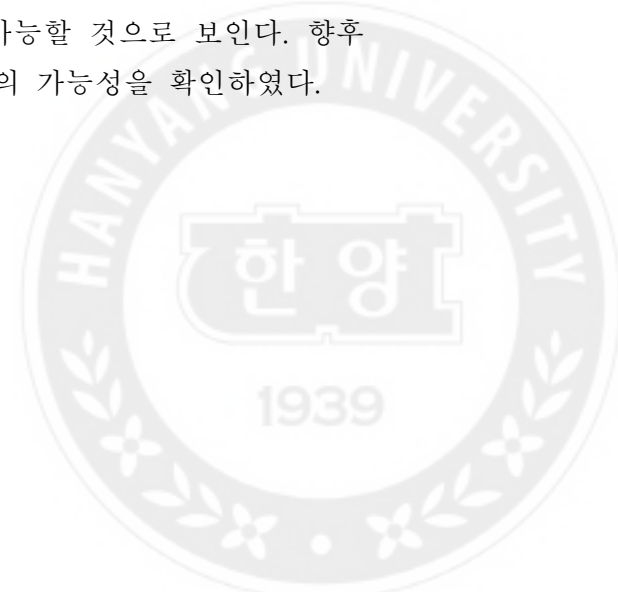
[그림 20] Sorensen-Dice를 이용한 실험 결과

위 [그림 19]에서 알 수 있듯이, 같은 패밀리의 악성코드 간의 평균 유사도 값은 0.57로 나타났으며, 다른 패밀리와의 평균 유사도 값은 약 0.19로 나타났다.

6. 결론

Cosine similarity와 Overlap coefficient의 실험결과로는 유사도 값이 전체적으로 높게 나타났지만, 같은 패밀리간의 평균 유사도 값과 다른 패밀리와의 평균 유사도 차이가 크지 않다. 반면, Jaccard Index와 Sorensen-DICE의 실험 결과 유사도 값이 전체적으로 낮게 나타났지만, 같은 패밀리간의 평균 유사도 값과 다른 패밀리와의 평균 유사도 값의 차이가 이전의 2가지 방법에 비해 크게 나타났다.

이러한 실험 결과를 바탕으로 각 유사도 계산식별 임계치를 설정해 악성코드를 분류할 경우, 악성코드 패밀리 분류가 가능할 것으로 보인다. 향후 악성코드 패밀리 분류에 대한 각 유사도 계산식의 가능성을 확인하였다.

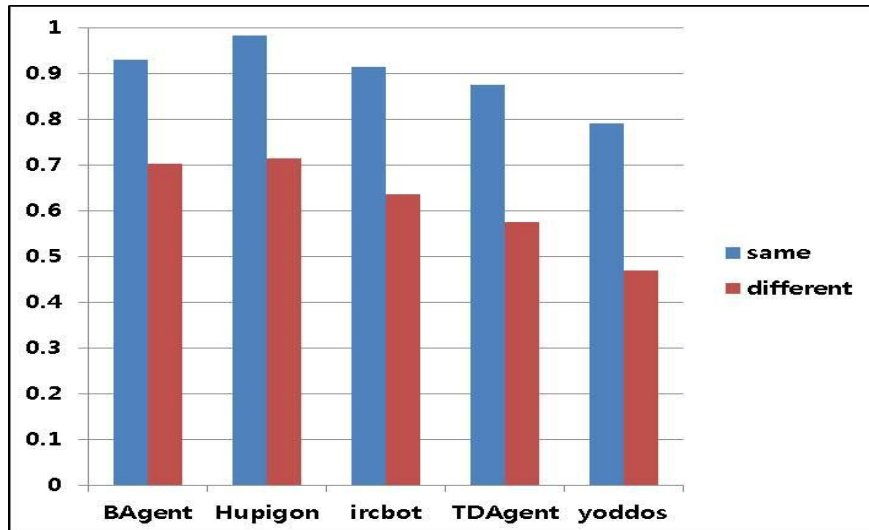


7. 추가 가중치 실험

이 실험은 이전의 Cosine Similarity 실험 결과와 Overlap Similarity 실험 결과를 분석하여 나타난 각각의 유사도 계산식에 단점을 보완하기 위하여 수행되었다. Cosine Similarity의 경우, 두 개의 샘플이 포함관계이지만 API 시퀀스 길이 차이가 매우 크면 유사도 결과 값이 매우 낮게 나타났다. 반면 Overlap Similarity에 경우, 두 개 샘플이 포함관계라면 API 시퀀스의 길이 차이에 상관없이 유사도 결과 값이 1이 나왔다. Overlap Similarity에서는 Cosine Similarity에서 표현하지 못하는 포함관계를 해결할 수 있다. 그러나 같은 패밀리간의 유사도 평균값과 다른 패밀리와의 유사도 평균값의 차이는 Overlap Similarity보다 Cosine Similarity가 더 높았다. 이러한 각각의 장단점을 이용하여 두 개의 유사도 계산식을 같이 이용하는 실험을 수행하였다.

실험 방법으로는 먼저 Overlap Similarity를 이용하여 포함관계인지 판단한다. 만약 유사도 결과 값이 0.8 이상일 경우, 해당 유사도 값을 그대로 사용한다. 만약 그렇지 않을 경우 포함관계가 아닌 것으로 판단하고 Cosine Similarity를 측정한다. 아래 [그림 20]는 두 개의 유사도 계산식을 같이 이용한 실험 결과이다.



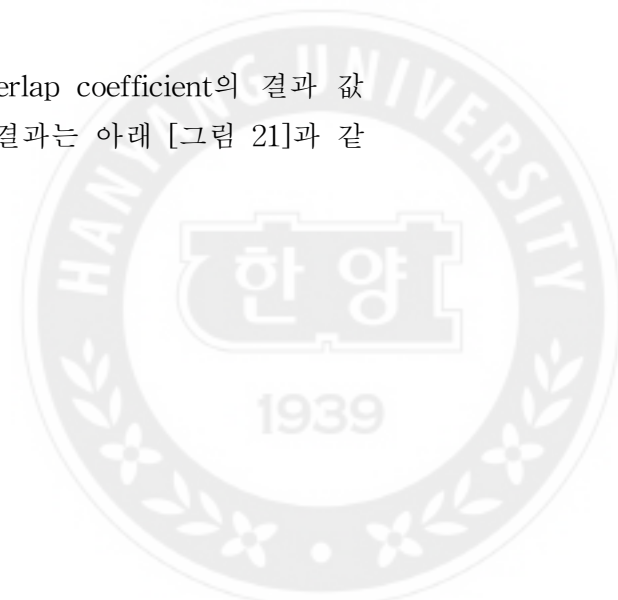


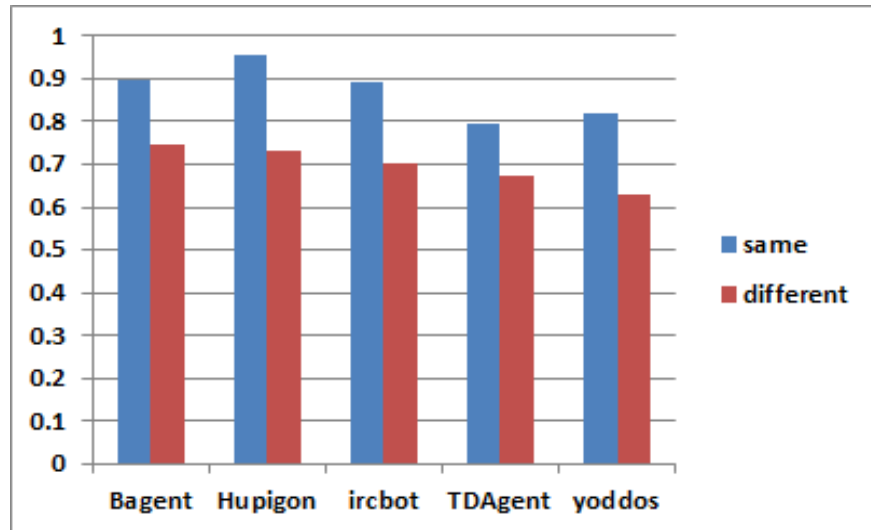
[그림 21] Cosine & Overlap Similarity 결과

실험 결과로 Cosine Similarity의 장점인 같은 패밀리간의 유사도 평균 값과 다른 패밀리와의 유사도 평균값의 차이가 크지 않다는 것과 Overlap Similarity의 장점인 포함관계의 표현을 모두 반영할 수 있었다. 그러나 두 가지의 유사도 계산식 결과값이 따로 적용되기 때문에 최종 결과 값의 의미를 정확하게 정의할 수 없다는 단점이 있다. 이후 연구에서는 두 가지 계산식에 대하여 가중치를 부여하여, 두 가지 계산식을 모두 사용하여 한 가지의 유사도 결과 값을 도출하는 실험이 필요하다. 따라서 두 가지 계산식에 대한 가중치는 검증 실험을 통하여 선정하며, 선정된 가중치를 이용하여 추가 실험을 진행하였다.

추가로 진행한 실험은 총 3가지의 방법으로 진행하였다. 유사도 계산식은 Cosine similarity와 Overlap coefficient를 모두 사용하였으며, 각각의 유사도 계산식에 가중치를 두어 유사도 계산 결과가 0 ~ 1 사이로 나오도록 정의하여 실험하였다.

먼저 Cosine similarity의 결과 값에 0.3, Overlap coefficient의 결과 값에 0.7 가중치를 두어 실험을 진행하였다. 실험 결과는 아래 [그림 21]과 같다.





[그림 22] 유사도 계산식 가중치 실험 1

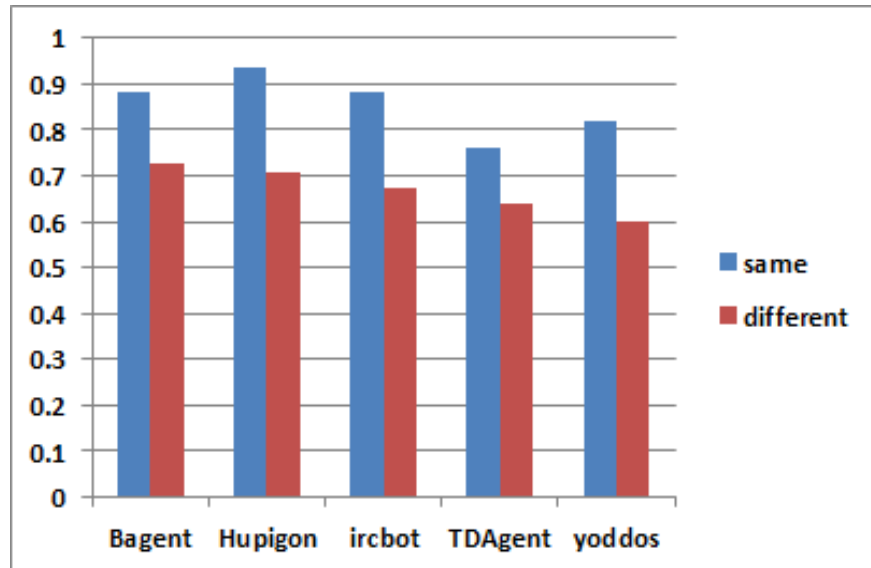
위 [그림 21]의 유사도 계산식은 다음과 같다.

$$similarity = (Cosine\ similarity \times 30\%) + (Overlap\ coefficient \times 70\%)$$

수식 5 Cosine similairy 30% & Overlap coefficient 70%

다음은 Cosine similarity의 결과 값에 0.5, Overlap coefficient의 결과 값에 0.5 가중치를 두어 실험을 진행하였다.





[그림 23] 유사도 계산식 가중치 실험 2

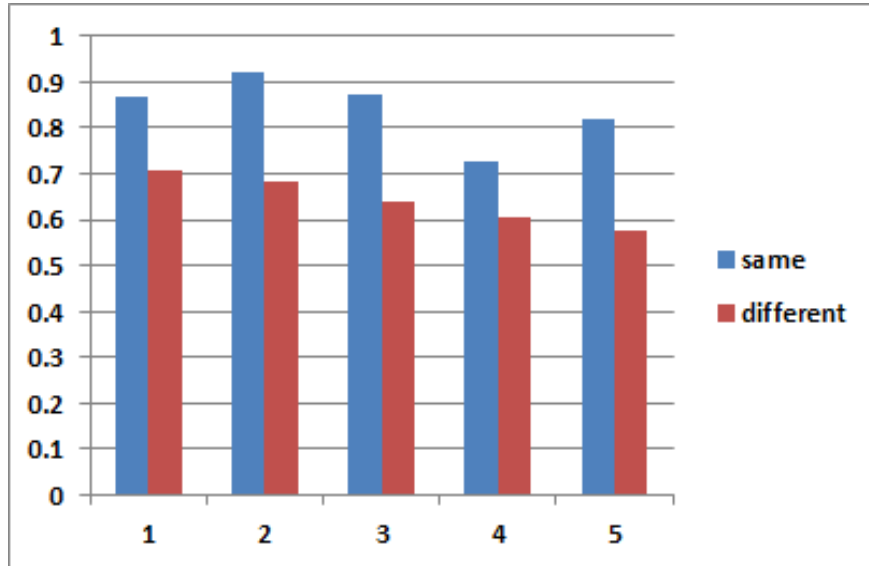
위 [그림 22]의 유사도 계산식은 다음과 같다.

$$similarity = (Cosine\ similarity \times 50\%) + (Overlap\ coefficient \times 50\%)$$

수식 6 Cosine similairy 50% & Overlap coefficient 50%

마지막으로 Cosine similarity의 결과 값에 0.5, Overlap coefficient의 결과 값에 0.5 가중치를 두어 실험을 진행하였다.





[그림 24] 유사도 계산식 가중치 실험 3

위 [그림 23]의 유사도 계산식은 다음과 같다.

$$similarity = (Cosine\ similarity \times 70\%) + (Overlap\ coefficient \times 30\%)$$

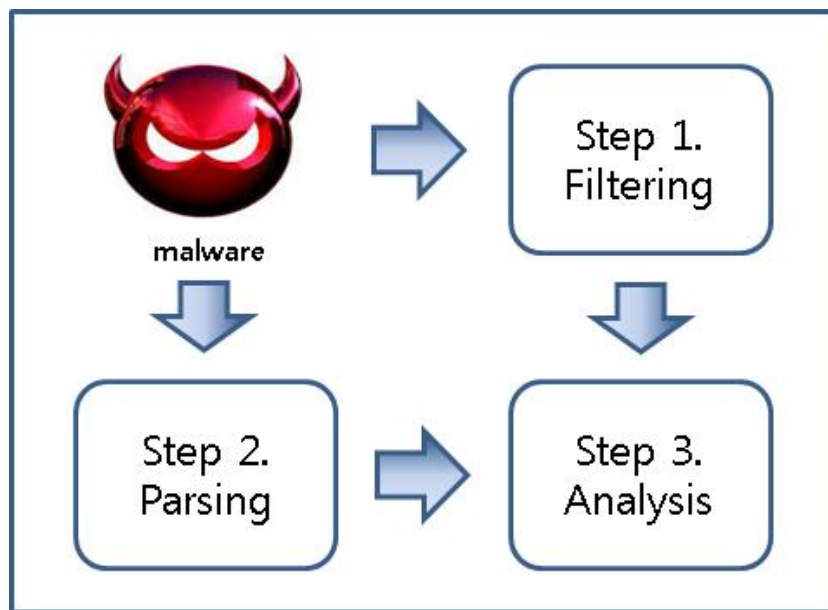
수식 7 Cosine similairy 70% & Overlap coefficient 30%

각각의 유사도 계산식에 가중치를 두어 실험한 결과 Cosine similarity에 가중치를 0.7, Overlap similarity에 가중치를 0.3 두는 것이 같은 패밀리 내의 평균 유사도 값과 다른 패밀리와의 평균 유사도 값 차이가 가장 높게 나타났다.



제 6 장 악성코드 분석 시스템

본 연구에 개발한 악성코드 분석 시스템에 대하여 설명한다. 악성코드 분석 시스템의 구성도는 아래 [그림 24]과 같다.



[그림 25] 악성코드 분석 시스템 구성도

악성코드 분석 시스템의 실행 순서는 다음과 같다.

o Step1. Filterig: 이 단계에서는 핀툴을 사용하여 악성코드를 동적 분석하고 인스트럭션 정보를 추출한 후 이를 이미지매트릭스로 변환한다. 그 후, 기존에 저장되어 있는 패밀리별 대표 이미지들과 입력된 악성코드의 이미지간의 유사도를 측정한다. 또한, 측정된 유사도 값이 임계치 이상인 패밀리 정보를 저장한다. 이는 Analysis단계에서 모든 악성코드 패밀리와 비교하지 않도록 Filtering한 정보이다.



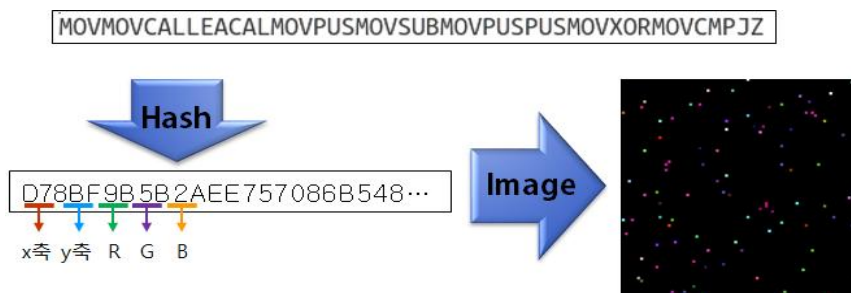
o Step2. Parsing: 이 단계에서는 Cuckoo sandbox를 이용하여 악성코드를 분석하고 API, parameter, 이벤트 정보 등을 Json파일에 저장한다.

o Step3. Analysis: 마지막으로 이 단계에서는 Json파일을 통해 행위 분석, 파라미터 분석, 악성코드 간의 유사도 분석을 한다.

제 1 절 모듈별 설명

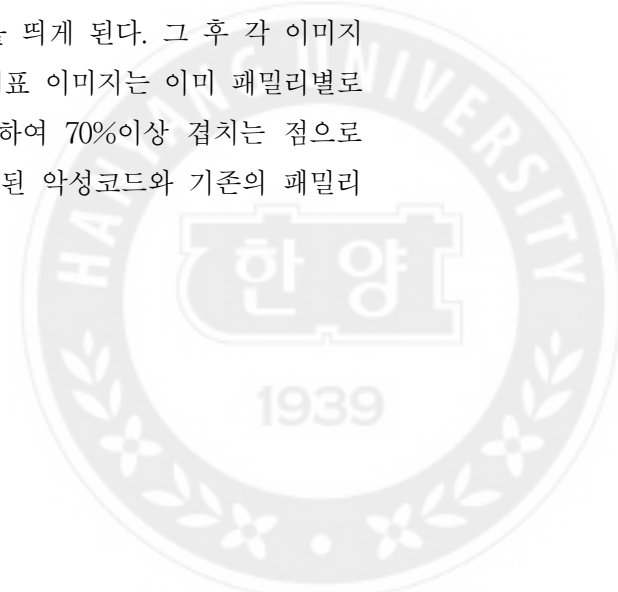
1. Filtering 단계

필터링 단계에서는 먼저 악성코드를 PIN tools을 이용하여 인스트럭션 데이터를 추출한다. 그 후에 이 인스트럭션 데이터에서 오퍼코드의 앞 세 글자만을 추출하여 블록단위별로 개행을 함으로써, 이미지매트릭스의 입력 파일로 파싱한다. 이 파싱된 데이터는 sha256을 통해 해쉬값으로 변환되며, 이 해쉬값이 다음 [그림 25]의 과정을 통해 bmp로 변환된다.

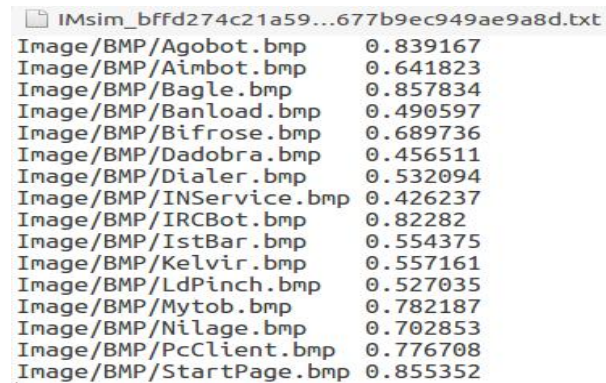


[그림 26] 인스트럭션 데이터 해쉬값 변환 및 이미지 변환

각 해쉬값의 앞에서 두개의 값씩은 256x256 bmp를 만드는 정보가 되는데, 순서대로 이미지에서의 x좌표, y좌표, 색상의 R, G, B값을 띄게 된다. 그 후 각 이미지를 악성코드 패밀리별 대표 이미지와 비교한다. 이 대표 이미지는 이미 패밀리별로 나누어진 악성코드들의 이미지를 패밀리별로 오버랩하여 70%이상 겹치는 점으로 구성된 이미지로 대표 이미지들과 비교함으로써 입력된 악성코드와 기존의 패밀리



간의 유사도를 산출한다. 마지막으로 [그림 26]과 같은 결과물을 통해 특정 Threshold값(0.7)을 기준으로 패밀리 명을 추출하여 분석 단계에서 사용한다. 악성 코드의 분석 환경은 Windows 7이다.



IMsim_bffd274c21a59...677b9ec949ae9a8d.txt	
Image/BMP/Agobot.bmp	0.839167
Image/BMP/Aimbot.bmp	0.641823
Image/BMP/Bagle.bmp	0.857834
Image/BMP/Banload.bmp	0.490597
Image/BMP/Bifrose.bmp	0.689736
Image/BMP/Dadobra.bmp	0.456511
Image/BMP/Dialer.bmp	0.532094
Image/BMP/INService.bmp	0.426237
Image/BMP/IRCBot.bmp	0.82282
Image/BMP/IstBar.bmp	0.554375
Image/BMP/Kelvir.bmp	0.557161
Image/BMP/LdPinch.bmp	0.527035
Image/BMP/Mytob.bmp	0.782187
Image/BMP/Nilage.bmp	0.702853
Image/BMP/PcClient.bmp	0.776708
Image/BMP/StartPage.bmp	0.855352

[그림 27] 패밀리별 대표 이미지와 입력한 악성코드 이미지간의 유사도 결과

2. Parsing 단계

파싱 단계에서는 악성코드를 Cuckoo sandbox로 분석하여 아래 [그림 27]과 같은 Json파일을 추출한다. 이 Json 파일은 바이너리파일의 API 정보와 parameter 정보 등을 포함하고 있다. 이렇게 추출된 Json 파일은 분석 단계에서 행위분석과 파라미터 분석, N-gram을 통한 API sequence 분석의 재료가 된다. 본 실험에서는 Cuckoo sandbox 버전 1.2-dev, 환경은 Windows XP를 통해 악성코드를 분석하였다.



```

report.json
1 {
2   "info": {
3     "category": "file",
4     "package": "",
5     "started": "2015-04-03 15:44:44",
6     "custom": "",
7     "machine": {
8       "shutdown_on": "2015-04-03 15:46:31",
9       "label": "cuckoo2",
10      "manager": "VirtualBox",
11      "started_on": "2015-04-03 15:44:52",
12      "id": 291,
13      "name": "cuckoo2"
14    },
15    "ended": "2015-04-03 15:46:31",
16    "version": "1.2-dev",
17    "duration": 107,
18    "id": 292
19  },
20  "signatures": [],
21  "procmemory": [],
22  "static": {
23    "pe_imports": [
24      {
25        "imports": [
26          {
27            "name": "MapViewOfFile",
28            "address": "0x44e018"
29          }
30        ]
31      }
32    ]
33  }
34 }

```

[그림 28] Json 파일 예시

3. Analysis 단계

분석 단계에서는 크게 행위 분석 및 위험도 산출, 파라미터 분석, 유사도 분석 3가지로 나뉜다.

가. 행위 분석

Json파일을 분석하여 정해진 행위 척도를 기준으로 분석된 악성코드가 다음과 같은 행위를 하는지를 측정하는 방식이다. 행위 척도는 Access to Physical Disks, Startup, Self Copy 등 총 15가지가 있다. 아래의 [그림 22]는 행위 분석 결과 예시이며, 아래의 [그림 28]는 위험도 산출 결과 예시이다.



```

Access to Physical Disks: false
local DNS modification: false
Remote Threads: false
Access to Other Process` Memory: false
Startup: false
Disable Security: false
Browser Extensions: false
Incorrect File Format: false
Self Copy: true
Driver Load: false
Similar Process Name: false
Self Remove: false
Environment Detection: true
Access to System Path: true
Mutex: true

```

[그림 29] 행위 분석 결과 예시

나. 파라미터 분석

패밀리별로 구분되어있던 악성코드들의 모든 파라미터 정보들을 저장해 놓고 이를 Json파일에 있는 파라미터 정보와 비교하여 패밀리간 유사도를 측정하는 방식이다. 여기서 파라미터의 빈도수는 결과에 영향을 주지 않고 해당 파라미터의 유무만을 판단하여 분석된 악성코드의 파라미터와 각 패밀리별 파라미터간의 유사도를 측정한다. 아래 [그림 29]는 파라미터 분석 결과 예시이다.

```

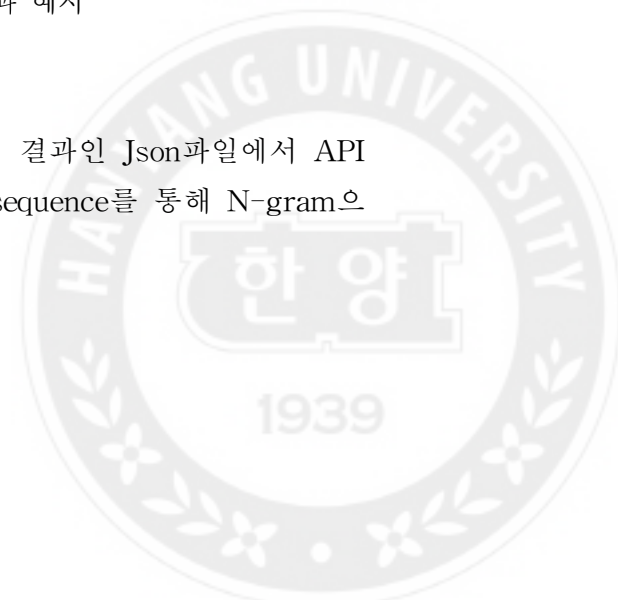
0a2986ef824a3f2ef393a0a9dc4a1d4c.txt.txt x
Banload : 50.00%
Bagle : 64.29%
Dadobra : 42.86%
StartPage : 50.00%
Dialer : 50.00%
Nilage : 64.29%
LdPinch : 42.86%
IstBar : 42.86%
INService : 21.43%
Mytob : 71.43%
Kelvir : 42.86%
PcClient : 57.14%
IRCBot : 57.14%
Bifrose : 50.00%
Aimbot : 50.00%
Agobot : 50.00%

```

[그림 30] 파라미터 분석 결과 예시

다. 악성코드간의 유사도 분석

파싱 단계에서 추출한 분석 대상인 악성코드의 결과인 Json파일에서 API를 추출하여 sequence를 제작한다. 그 후, API sequence를 통해 N-gram으



로 빈도수를 추출하고, 추출한 빈도수를 벡터로 표현한다. 다음으로 모든 패밀리에 대한 API sequence 비교 분석이 아닌, 필터링 단계에서 필터링된 패밀리 정보를 이용하여 해당 패밀리 내의 악성코드와의 유사도를 계산한다. 아래 [그림 30]은 유사도 분석 결과의 예시이다.

NG_sim_009718454c90...8af39962c1997816.txt x		
Agobot	cosine similarity: 0.0438049	overlap similarity: 0.155135
Aimbot	cosine similarity: 0.0410394	overlap similarity: 0.304419
IRCBot	cosine similarity: 0.0398207	overlap similarity: 0.160965
Bagle	cosine similarity: 0.037654	overlap similarity: 0.412504
Kelvir	cosine similarity: 0.0244833	overlap similarity: 0.134296
Mytob	cosine similarity: 0.0303108	overlap similarity: 0.267579
Dialer	cosine similarity: 0.056528	overlap similarity: 0.399957
StartPage	cosine similarity: 0.0244811	overlap similarity: 0.166071
Banload	cosine similarity: 0.0330548	overlap similarity: 0.332715
Dadobra	cosine similarity: 0.0314177	overlap similarity: 0.275711
INService	cosine similarity: 0.0759681	overlap similarity: 0.124414
IstBar	cosine similarity: 0.0335535	overlap similarity: 0.303027
LdPinch	cosine similarity: 0.045289	overlap similarity: 0.356487
Nilage	cosine similarity: 0.0238068	overlap similarity: 0.115375
PcClient	cosine similarity: 0.0623168	overlap similarity: 0.124649
Bifrose	cosine similarity: 0.0608109	overlap similarity: 0.335581

[그림 31] 유사도 분석 결과 예시



제 7 장 결론 및 향후 연구

본 논문에서는 악성코드 동적 분석하여 획득 가능한 악성코드의 API 정보를 이용하였다. 데이터 정보 분석을 위한 실험과 N-gram 기법에 대한 최적의 N 값을 찾는 실험을 진행하였다. 그리고 빈도수 기반인 N-gram 기법과 서열 정렬 기반인 Smith-Waterman 기법을 이용하여 유사도 측정하는 실험을 진행하였으며, 이를 비교하였다. 또한, N-gram 기법의 결과인 빈도수를 이용한 4가지 유사도 계산식(Cosine Similarity, Overlap coefficient, Jaccard Index, Sorensen-Dice)비교 실험을 진행하였다. 이 실험을 기반으로 Cosine Similarity와 Overlap coefficient를 2가지 계산식을 이용하여 악성코드 분류 실험을 진행하였다.

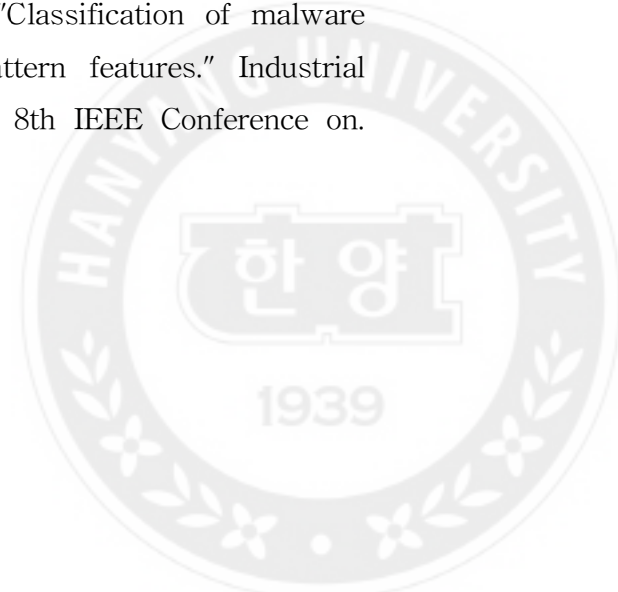
마지막으로 이러한 연구를 기반으로 한 악성코드 분석 시스템을 최종적으로 개발하여 실험 및 실험 결과를 분석하였다.

향후 연구에서는 패밀리 정보가 존재하는 다양한 샘플을 이용하여 악성코드 패밀리 분류 정확도를 측정하고, 이를 기반으로 악성코드 패밀리 분류 시스템을 개발할 예정이다.

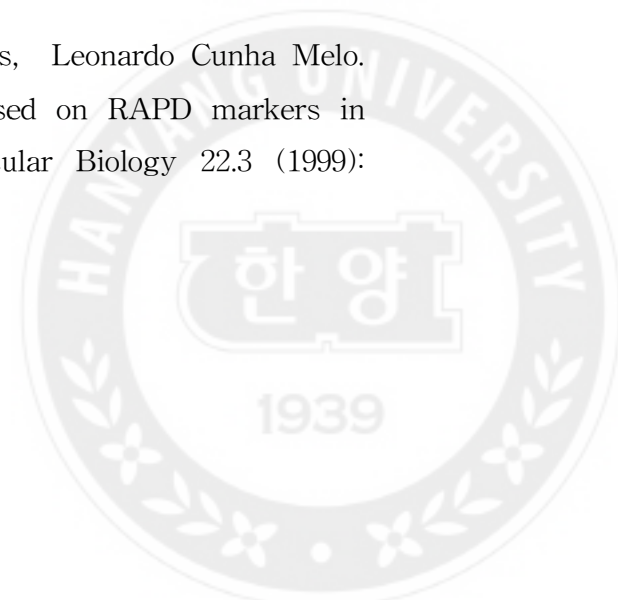


제 8 장 참고 문헌

- [1] AV-TEST, <http://www.av-test.org/en/>
- [2] Cuckoo sandbox, <http://www.cuckoosandbox.org/>
- [3] Pearson, William R. "Searching protein sequence libraries: comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms." *Genomics* 11.3 (1991): 635-650.
- [4] Rieck, K., Holz, T., Willems, C., Dussel, P., Laskov, P. "Learning and Classification of Malware Behavior." In *Detection of Intrusions and Malware, and Vulnerability Assessment*, 108-125: Springer, 2008.
- [5] Wu, Yilun, Bofeng Zhang, Zhiquan Lai, Jinshu Su "Malware network behavior extraction based on dynamic binary analysis" *Software Engineering and Service Science (ICSESS)*, 2012 IEEE 3rd International Conference on. IEEE, 2012.
- [6] Wu, L., Ping, R., Ke, L., Hai-xin, D. "Behavior-based malware analysis and detection", *Complexity and Data Mining (IWCDM)*, 2011 First International Workshop on. 39-42, 2011.
- [7] Alazab, M., Venkataraman, S., Watters, P. "Towards understanding malware behaviour by the extraction of API calls", *Cybercrime and Trustworthy Computing Workshop (CTC)*, 2010 Second. 52-59, 2010.
- [8] Liangboonprakong, Chatchai, Ohm Sornil. "Classification of malware families based on n-grams sequential pattern features." *Industrial Electronics and Applications (ICIEA)*, 2013 8th IEEE Conference on. IEEE, 2013.



- [9] Zhong, Yang, Hirofumi Yamaki,, Hiroki Takakura. "A malware classification method based on similarity of function structure." Applications and the Internet (SAINT), 2012 IEEE/IPSJ 12th International Symposium on. IEEE, 2012.
- [10] Kang, B., Kim, T., Kwon, H., Choi, Y., Im, E. G. "Malware classification method via binary content comparison." Proceedings of the 2012 ACM Research in Applied Computation Symposium. ACM, 2012.
- [11] Cavnar, William B., John M. Trenkle. "N-gram-based text categorization." Ann Arbor MI 48113.2 (1994): 161-175.
- [12] Nguyen, Hieu V., Li Bai. "Cosine similarity metric learning for face verification." Computer Vision-ACCV 2010. Springer Berlin Heidelberg, 2011. 709-720.
- [13] Manders, E. M. M., F. J. Verbeek, J. A. Aten. "Measurement of co-localization of objects in dual-colour confocal images." Journal of microscopy 169.3 (1993): 375-382.
- [14] Lieve Hamers, Yves Hemeryck, Guido Herweyers, Marc Janssen, Hans Keters, Ronald Rousseau, Andre Vanhoutte "Similarity measures in scientometric research: the Jaccard index versus Salton's cosine formula." Information Processing & Management 25.3 (1989): 315-318.
- [15] Duarte, Jair Moura, Joao Bosco dos Santos, Leonardo Cunha Melo. "Comparison of similarity coefficients based on RAPD markers in the common bean." Genetics and Molecular Biology 22.3 (1999):



427-432.



Appendix

저자 이력

- 수행 프로젝트

○ 프로젝트 1

- 연구과제명 : 행위 기반 악성코드 유사도 측정 및 분류 방법에 관한 연구
- 연구기간 : 2014.08 ~ 2015.01 · 위탁기관 : 한국인터넷진흥원

○ 프로젝트 2

- 연구과제명 : 악성코드 호출 API 및 파라미터 기반 위험도 산출 방안 연구
- 연구기간 : 2015.07 ~ 2015.12 · 위탁기관 : 한국인터넷진흥원

- 발표 논문

○ 국내 학술대회 논문

- API 호출 시퀀스를 이용한 악성코드 유사도 계산 기법 비교, CISC 동계학술대회, 2014. 12
- N-gram 기반의 악성코드 유사도 계산식 연구, WDCS, 2015.6
- API 호출 시퀀스를 이용한 악성코드 유사도 측정 기법 연구, CISC 동계학술대회, 2015. 12
- API 호출 시퀀스 기반 악성코드 유사도 계산 기법 비교, 융합·스마트미디어시스템 워크샵, 2016. 5

○ 국외 학술대회 논문

- A study on similarity calculation method for API invocation sequences (IJCRS), 2015.10



Abstract

Malware Detection And Classification System based on API Call Sequence

Yu Jin Shim

Dept. of Computer and Software

The Graduate School of

Hanyang University

Malware variants have been developed and spread in the Internet, and the number of new malware variants is increases every year. Recently, malware is applied with obfuscation and mutation techniques to hide its existence, and malware variants are developed with various automatic tools that transform the properties of existing malware to avoid static analysis based malware detection systems. It is difficult to detect such obfuscated malware with static-based signatures, so we have designed a detection system based on dynamic analysis. In this paper, we propose a dynamic analysis based system that uses the API invocation sequences to compare behaviors of suspicious software with behaviors of existing malware.



감사의 글

2016년 8월

심유진

대학원 2년이 벌써 지나 졸업을 앞두고 있습니다.
그 동안 다양한 경험을 할 수 있는 많은 일들이 있었습니다.
때론 즐거워서 웃기도 하고, 때론 힘들어 울기도 하였습니다.
2년 동안 연구를 하는 동안 도움을 주신 많은 분들에게 감사를 표합니다.
그 분들의 이름을 여기에 하나하나 적는 것이 제가 할 수 있는 감사의
표시인 것 같습니다.

아버지 심재봉, 어머니 원영희, 형 심세진, ,외할머니 손옥화, 친척 분들
지도교수이신 임을규 교수님, 강수용 교수님, 차재혁 교수님

한경수 박사님, 김태근 박사님, 김준형 박사님

성명재 선배님, 강병호 선배님, 박정빈 선배님, 이현수 선배님, 조인겸
선배님, 권일택 후배님, 이태경 후배님, 이용찬 후배님, 박준규 후배님,
최두섭 후배님, 왕러 후배님, 이청 후배님, 김용혁 후배님, 지환태 후배님,
정윤희, 박소영

그 외 다른 분들까지

2년 동안 많은 도움을 받아 정말로 감사드립니다.

앞으로도 더욱 발전한 모습을 보여드리는 것이 보답이라 생각하며,
사회에 나가 열심히 정진하도록 하겠습니다.

다시 한번 모두에게 감사하다는 말씀을 드리며
이만, 마치도록 하겠습니다



연구 윤리 서약서

본인은 한양대학교 대학원생으로서 이 학위논문 작성 과정에서 다음과 같이 연구 윤리의 기본 원칙을 준수하였음을 서약합니다.

첫째, 지도교수의 지도를 받아 정직하고 엄정한 연구를 수행하여 학위논문을 작성한다.

둘째, 논문 작성시 위조, 변조, 표절 등 학문적 진실성을 훼손하는 어떤 연구 부정행위도 하지 않는다.

셋째, 논문 작성시 논문유사도 검증시스템 "카피킬러"등을 거쳐야 한다.

2016년06월20일

학위명 : 석사

학과 : 컴퓨터·소프트웨어학과

지도교수 : 임을규

성명 : 심유진

(서명)

한 양 대 학 교 대 학 원 장 귀 하



Declaration of Ethical Conduct in Research

I, as a graduate student of Hanyang University, hereby declare that I have abided by the following Code of Research Ethics while writing this dissertation thesis, during my degree program.

"First, I have strived to be honest in my conduct, to produce valid and reliable research conforming with the guidance of my thesis supervisor, and I affirm that my thesis contains honest, fair and reasonable conclusions based on my own careful research under the guidance of my thesis supervisor.

Second, I have not committed any acts that may discredit or damage the credibility of my research. These include, but are not limited to : falsification, distortion of research findings or plagiarism.

Third, I need to go through with Coppykiller Program(Internet-based Plagiarism-prevention service) before submitting a thesis."

JUNE 20, 2016

Degree : Master

Department : DEPARTMENT OF COMPUTER AND SOFTWARE

Thesis Supervisor : Eul Gyu Im

Name : SHIM YUJIN

(Signature)