Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.27, NO.1, Feb. 2017 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2017.27.1.91

API 콜 시퀀스와 Locality Sensitive Hashing을 이용한 악성코드 클러스터링 기법에 관한 연구*

고 동 우,† 김 휘 강‡
고려대학교 정보보호대학원

A Study on Malware Clustering Technique Using API Call Sequence and Locality Sensitive Hashing*

Dong Woo Goh,† Huy Kang Kim‡
Korea University

요 약

API(Application Program Interface) 콜 시퀀스 분석은 분석 대상 프로그램에서 API 콜 정보를 추출한 후 분석하는 기법으로 다른 기법들에 비해 대상의 행위를 특징할 수 있는 장점이 있다. 하지만 기존의 API 콜 시퀀스 분석 기법은 동일한 기능을 수행하는 함수를 상이한 함수로 잘못 식별하여 분석을 수행하는 문제점이 존재한다. 본 연구에서는 API 각각을 추상화시키는 방식을 추가하여 기존의 식별 문제를 해결하고 분석 성능을 향상시키고자 한다. 그 후 분석 대상들에서 획득한 추상화된 API 콜 시퀀스에 LSH(Locality Sensitive Hashing) 기법을 적용하여 각 분석 대상들 간의 유사도를 계산하고 유사한 유형끼리 클러스터를 형성하는 과정을 수행하였다. 본 연구는 악성코드 분석 시 악성코드의 유형을 파악하는 데 요긴하게 사용할 수 있으며, 최종적으로는 해당 유형 정보를 기반으로 악성코드 분석의 정확도를 향상시키는 데 기여할 수 있다.

ABSTRACT

API call sequence analysis is a kind of analysis using API call information extracted in target program. Compared to other techniques, this is advantageous as it can characterize the behavior of the target. However, existing API call sequence analysis has an issue of identifying same characteristics to different function during the analysis. To resolve the identification issue and improve performance of analysis, this study includes the method of API abstraction technique in addition to existing analysis. From there on, similarity between target programs is computed and clustered into similar types by applying LSH to abstracted API call sequence from analyzed target. Thus, this study can attribute in improving the accuracy of the malware analysis based on discovered information on the types of malware identified.

Keywords: API call sequence, malware analysis, clustering, dynamic analysis

I. 서 론

카스퍼스키(kaspersky)의 "Kaspersky Security Bulletin 2015" 보고서에 따르면 1년간 카스퍼스키 백신에 탐지된 악성 프로그램은 총 121,262,075개에 달한다. 이는 사람이 하나하나 분석할 수 없을

Received(10. 27. 2016), Modified(12. 19. 2016), Accepted(12. 19. 2016)
* This work was supported by the ICT R&D Program of MSIP/IITP.[14-912-06-002, The Development of Script-based Cyber Attack Protection Technology]
† 주저자, lylian@korea.ac.kr
‡ 교신저자, cenda@korea.ac.kr(Corresponding author)

정도로 많은 악성코드들이 유포되고 있다는 것을 의미한다[1]. 좀 더 상세하게 살펴보자면, 2015년 한 해 동안 랜섬웨어 프로그램은 753,684개의 컴퓨터에서 탐지되었고, 179,209개의 컴퓨터가 랜섬웨어의 타겟으로 지정되었다. 또한 해당 업체의 백신에서는 4,000,000개의 악성 객체 및 잠재적으로 불필요한 객체를 발견하였다.

리포트 결과에서 나타나듯 수많은 악성코드들이 유포되고 있고, 이를 분석하기 위해서는 시간과 자원의 큰 소모가 발생한다. 또한 악성코드 분석은 여전히 숙련된 분석가의 능력에 따라 분석 시간 및 분석 효율이 좌우되는 상황이다. 이를 극복하기 위해 API 콜 그래프 분석, API 호출 빈도 분석, 정렬(alignment) 기법을 이용한 API 콜 시퀀스 분석 등 여러 가지 악성코드 자동 분석 기법들이 제안되고 있다. 악성코드 자동 분석 기법을 개발하기 위해서는 우선 악성코드를 분석하기 위해 정적/동적 분석을 수행할 엔진이나 프레임워크가 필요하다. 여기서 수집한 정보를 이용하여 시그니처 기반 분석, 데이터 마이닝, 머신 러닝, 통계적 분석 등 여러 가지 방식을 적용하여 악성코드 자동 분석을 수행하게 된다.

본 연구에서는 이러한 악성코드 자동화 분석 기법 중 하나로 API의 콜 시퀀스를 동적 분석 기법을 이용하여 추출한 후 LSH를 이용하여 다른 악성코드와의 유사도를 계산하고, 유사 악성코드를 클러스터링해 내는 기법을 제안한다. 본 기법은 가상 분석 환경을 이용한 동적 분석을 수행하여 API 콜 시퀀스를 추출하고, LSH 기법을 이용하여 유사도를 계산한 후 최종적으로 유사한 유형의 악성코드를 클러스터링해 내는 기법이다. 기존의 API 콜 그래프 분석이나 정렬 기법을 이용한 분석 기법들은 연산의 복잡도가 높아 시간 및 컴퓨팅 자원을 많이 소모하기 때문에 LSH를 이용한 API 콜 시퀀스 분석 기법을 이용하여 자원 소모를 최소화하고 신속한 분석을 수행하기 위해서 해당 기법을 제안한다.

본 기법을 이용하여 악성코드 분석 시 악성코드의 유형을 파악하는 데 요긴하게 사용할 수 있으며, 최종적으로는 해당 유형 정보를 기반으로 악성코드 분석의 정확도를 향상시키는 데 기여할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 악성코드 분석 관련 연구 및 LSH에 관련된 문헌 연구를 수행한다. 3장에서는 제안하는 자동화된 악성코드 분석 기법에 대해 상세히 정의한다. 4장에서는 제안하는 기법의 분석 정확도를 검증하고, 마지막 5장에서는 결론을 맺고 향후 연구에 관해 언급한다.

II. 관련 연구

2.1 API 콜 시퀀스 관련 악성코드 분석 연구

Hofmeyr 외 2인이 API 콜 시퀀스를 이용한 악성코드 분석에 관한 연구를 최초로 제안하였다[2]. 해당 연구에서는 정상 동작을 프로그램에 의해 실행되는 시스템 콜의 짧은 시퀀스로 정의하고, 정상 동작과 비정상 동작의 편차를 검출하기 위해 세 개의 지표를 사용하였다.

Sami 외 5인은 API 콜 마이닝 기반의 악성코드 탐지 기법을 개선하고, 악성코드 연구 촉진을 위한 최초의 공공 데이터 집합을 만든 연구를 제안하였다[3]. 해당 기법은 PE 파일의 IAT(Import Address Table)에서 API 콜 목록을 수집하고, 3가지 종류의 데이터를 생성한 후 랜덤 포레스트, 나이브 베이지안, 의사 결정 트리 분류기를 이용하여 분류를 수행하였다.

한경수 외 3인은 문자열 및 API를 이용한 악성코드 자동 분류 시스템은 같은 집합에 속한 악성코드는 포함된 문자열과 호출되는 API의 빈도가 유사할 것이라는 가정에서 출발하여 정적 분석과 동적 분석 모두를 사용한 악성코드 분류 방식에 관한 연구를 제안하였다[4]. 해당 연구에서는 정적 분석 단계에서는 악성코드 실행 파일에 포함된 문자열을 이용하여 악성코드와의 유사도를 비교하고, 동적 분석 단계에서는 악성코드의 API 호출 빈도를 이용하여 유사도를 비교한다. 최종적으로 문자열의 유사도와 API 유사도 결과 모두를 이용하여 가장 높은 유사도를 가지는 집합으로 분류를 수행한다.

박재우 외 6인은 API 호출 빈도를 이용한 악성코드 분류에 관한 연구를 제안하였다[5]. 해당 연구에서는 기존의 악성코드 샘플에서 호출 빈도수 API 리스트를 추출하여 데이터베이스로 생성한 후 분류할 프로그램에서 추출한 API 리스트와 비교하여 유사도를 측정한다. API 리스트 추출에는 동적 커널 후킹이 적용된 Strace NT 도구를 사용하였다[6].

Shankarapani 외 3인은 데드 코드 인젝션, 제어 흐름 난독화, 레지스터 재할당, 데이터 난독화 등의 난독화 기술이 적용된 변종 악성코드를 탐지하기 위한 연구를 제안하였다[7]. 해당 연구는 정적 분석기와 코드 검사기의 2단계 구조로 구성되어 있는 특

징이 있다. 정적 분석기에서는 PE 파싱을 통해 획득한 정적 API 콜 시퀀스와 정적 API 콜 셋에 시퀀스 정렬 기법을 적용하여 콜 시퀀스의 순서를 맞추고 코사인 유사도와 확장 자카드 상관계수, 피어슨 상관계수를 이용하여 유사도를 측정한다. 코드 검사기에서는 디스어셈블된 어셈블리 콜을 이용하여 3단계에 걸쳐 악성코드의 명령어와 유사한지 확인하는 과정을 걸쳐 악성 행위를 식별한다.

Alazab 외 3인은 제로데이 악성코드 탐지를 위한 분류 기법에 관한 연구를 제안하였다[8]. 해당 연구에서는 IDA Pro 도구의 IDA2SQLite와 디스어셈블러 도구를 이용하여 API 시퀀스를 획득하고, 나이브 베이지안, kNN(k-Nearest Neighbors), 순차적 최소 최적화 알고리즘, 역전파 신경망, 의사결정 트리 등의 여러 가지 분류기를 이용하여 악성코드를 분류하였다.

Erbiai Elhadi 외 2인은 그래프 매칭 알고리즘을 이용하여 API 콜 그래프 기반의 악성코드 탐지 시스템을 제안하였다[9]. 해당 연구에서는 기존의 그래프 매칭 알고리즘을 사용하여 API 콜 그래프를 매칭시키는 것은 NP-complete 문제로 많은 시간이 소요되고, 이를 개선하기 위해 입력 샘플을 단순한 데이터 의존성 그래프로 변환하고, 그래프 매칭 알고리즘을 적용하여 연산 복잡도를 낮추어 빠른 연산을 수행할 수 있도록 하였다. 탐지율, 오탐율, 정확도, ROC 등 네 가지 측정치를 이용하여 성능을 평가하였고, 높은 탐지율과 낮은 오탐율을 보였다.

기영준 외 2인은 MSA(Multiple Sequence Alignment) 알고리즘과 LCS(Longest Common Subsequence) 계산 방식을 이용한 악성코드 탐지 시스템을 제안하였다[10]. 해당 시스템에서는 동적 분석을 이용하여 API 콜 시퀀스를 수집하고, 콜 시퀀스를 알파벳 문자열로 매칭시켜 추상화를 수행하였다. 그 후 생물학에서 DNA 시퀀스 정렬을 위해 사용하는 MSA 알고리즘을 사용하여 여러 가지 종류의 악성코드 내에서 일반적인 악성 기능에 대한 API 콜 시퀀스 패턴을 추출하였다. 최종적으로 추출한 API 콜 시퀀스에 LCS 계산 방식을 적용하여 악성코드의 API 콜 시퀀스 시그니처를 생성하였다. 새로운 프로그램이 입력되면 API 콜 시퀀스를 구하고 API 콜 시퀀스 시그니처 데이터베이스와 비교하여 최종 탐지 결과를 출력한다. 해당 연구는 API 콜 시퀀스 시그니처를 데이터베이스로 생성할 수 있다는 장점이 있지만, MSA 알고리즘이나 LCS 계산 방식이 각각 NP-complete, NP-hard 문제이기 때문에 큰 연산 비용이 발생한다는 단점이 있다.

장재욱 외 4인은 시스템 콜 그래프에 소셜 네트워크 분석을 적용하여 악성 코드의 유형을 분류하는 시스템을 제안하였다[11]. 해당 시스템은 난독화 및 패킹 등의 탐지 우회 기술이 적용되어 있는 악성코드를 탐지하기 위하여 시스템 콜 그래프를 생성하고 소셜 네트워크 분석을 이용하여 악성코드 탐지를 수행하는 구조를 가지고 지고 있다. 이 때 악성코드의 분류를 위해 소셜 네트워크의 특성, degree 분포, degree 중심성, 네트워크 밀도 등의 여러 가지 특성들을 이용하였다. 실험 결과는 영향력 기반 그래프를 이용하여 표현하였고 나이브 베이지안, 의사 결정 트리, kNN 등의 여러 분류기를 이용하여 성능 평가를 수행하였다. 결과적으로 악성코드 분류에는 degree 중심성이 효과적인 특성임을 보였다.

기타 데이터 마이닝 알고리즘을 이용한 악성코드 탐지 기법들도 제안되었다. Ye 외 5인[12]은 연관관계 마이닝 알고리즘을 악성코드 탐지에 적용하여 기존의 시그니처 기반 탐지나 데이터 마이닝 기반 탐지 방법론을 능가하는 성능을 보인 연구를 제안하였다. Ahmadi 외 3인[13]은 동적 악성코드 탐지 시스템에서 실행 파일의 API 콜에서 반복적인 패턴을 추출하기 위해 패턴 마이닝 기법을 적용한 연구를 제안하였고, Sundarkumar 외 3인[14]은 텍스트 마이닝과 텍스트 문서 내의 단어들이 어떤 특정 토픽에 포함될 확률을 모델링하는 LDA(Latent Dirichlet Allocation) 토픽 모델링을 이용한 분석 모델을 제안하였다.

2.2 Locality Sensitive Hashing (LSH)

LSH 기법은 1bit의 내용만 달라져도 전체 해시 값이 확연히 달라지는 일반적인 해시와 달리 비슷한 데이터의 충돌(collision) 확률을 극대화하여 유사한 데이터들 끼리 유사한 해시 값을 가지도록 하여 유사도를 비교하는 해시 알고리즘이다. ssdeep, simhash, TLSH 등이 대표적인 알고리즘이다.

Li 외 6인은 기존 LSH 기반의 악성코드 클러스터링 기법의 문제점을 해결한 새로운 알고리즘을 제안하였다[15]. 해당 논문에서 언급하고 있는 기존 LSH의 문제점은 크게 두 가지이다.

첫 번째로는 입력 값의 입력 순서에 따라 결과가

차이가 날 수 있는 문제이다. 예를 들어, sdhash 알고리즘은 길이가 짧은 입력 값을 기준 데이터로 선정하여 유사도를 비교하는 작업을 수행하는데 길이가 같은 두 입력 값의 유사도를 비교하는 경우, 두 값의 입력 순서에 따라 기준이 되는 데이터가 변하기 때문에 상이한 유사도 결과를 획득하는 문제가 발생한다.

두 번째로는 다른 크기의 데이터에 대한 유사도를 측정할 때 해석에 따라 다른 결과 값을 가질 수 있는 문제이다. 두 개의 문자열 "ABC"와 "ABCDE"의 유사도를 측정할 경우를 예로 들면, 문자열 "ABC"가 문자열 "ABCDE"의 내부에 포함되기 때문에 100% 일치한다고 해석할 수도 있고, "DE"라는 문자열을 문자열 "ABC"가 포함하고 있지 않기 때문에 100% 일치하지 않는다고 해석할 수도 있다는 문제점이다.

해당 연구에서는 이 문제를 해결하기 위해 TLSH 알고리즘을 사용하였다.

2.2.1 ssdeep

Kornblum은 2006년에 LSH 알고리즘을 이용한 유사도 비교 도구 ssdeep을 제안하였다[16].

ssdeep에는 CTPH(Context-Triggered Piecewise Hashing)라는 기법이 적용되어 있는데, 이는 파일을 일정한 크기의 구역으로 나눈 후 각각의 구역에 대한 해시 값을 계산하여 하나의 문자열로 합치는 방식이다. 이 방식을 통해 구한 해시는 동일한 데이터를 가진 구역의 해시 값은 동일하고, 상이한 데이터를 가진 구역의 해시 값은 상이하게 나타나는 형태로 나타난다. 이를 이용하여 두 파일의 유사도를 계산하게 된다.

해당 알고리즘은 파일의 유사도를 쉽게 계산할 수 있지만, 상이한 길이의 데이터의 유사도를 비교할 때 차이나는 길이만큼 유사도가 떨어지는 단점이 있다.

2.2.2 simhash

Sadowski와 Levin이 제안한 simhash는 미리 정렬된 해시 키 값을 비교하는 방식으로 유사도를 비교하는 알고리즘이다[17].

우선 데이터에서 feature와 weight를 추출한 후 일반적인 해시를 적용한다. 이 해시 결과 에 대해 비트 단위로 0 또는 1의 값을 갖는 리스트를 생성하고, 1은 양의 weight로, 0은 음의 weight로 간주하여 리스트의 같은 위치 값들을 합산한다. 마지막으로 합산 된 값들을 다시 1과 0의 비트 벡터로 변환하여 simhash의 해시 값으로 사용한다.

2.2.3 TLSH

Oliver 외 2인이 제안한 TLSH는 트렌드마이크

Table 1. False positive and detection rate for ssdeep, TLSH and nilsimsa hash

ssdeep			TLSH			Nilsimsa		
score	FP rate (%)	detect rate (%)	score	FP rate (%)	detect rate (%)	score	FP rate (%)	detect rate (%)
〉0	0.04711	37.1	〈 300	79.30	98.8	〉120	99.86	100.0
〉5	0.02718	36.6	〈 250	69.06	98.8	〉130	99.20	100.0
〉10	0.02174	36.1	〈 200	50.10	98.8	〉140	98.11	100.0
〉20	0.01812	35.4	〈 150	24.33	98.1	〉150	96.98	100.0
〉30	0.01268	34.4	〈 100	6.43	94.5	〉160	94.26	100.0
〉40	0.00544	32.7	〈 90	4.49	92.3	〉170	89.52	100.0
〉50	0.00362	29.7	〈 80	2.93	89.0	〉180	81.38	100.0
〉60	0.00362	26.0	〈 70	1.84	83.6	〉190	69.69	99.7
〉70	0.00181	18.8	〈 60	1.09	76.0	〉200	54.45	98.8
〉80	0.00181	12.4	〈 50	0.52	65.3	〉210	36.73	96.4
〉90	0.00181	4.6	〈 40	0.07	49.6	〉220	18.29	91.9
〉99	0.00000	1.0	〈 30	0.00181	32.2	〉230	5.52	72.0
			〈 20	0.00181	17.3	〉240	1.26	35.2
			〈 10	0.00181	6.4	〉250	0.49	9.5

로 사에서 개발하여 제공하는 LSH 알고리즘이다[18]. 크기 5의 윈도우를 슬라이딩하면서 데이터를 분할하고, 분위 점을 계산하여 해시 값의 헤더와 바디를 구성하는 방식으로 구성되어 있다. 심플한 알고리즘 구조를 가져 ssdeep 알고리즘과 거의 비슷한 해시 계산 속도를 가진다. 또한 상이한 길이의 데이터를 유사도를 측정할 때 길이 차만큼 유사도가 감소하는 ssdeep의 단점을 해결하였다.

Table 1.에서 보듯이, ssdeep, nilsimsa 등의 기존 LSH 알고리즘과 비교 해 보았을 때 낮은 오탐율(false positive)과 높은 탐지율을 보이는 등 우수한 성능을 가지고 있는 LSH 알고리즘이기 때문에 본 논문의 클러스터링 기법에서 사용하였다.

III. API 콜 시퀀스를 이용한 악성코드 클러스터링 기법

3.1 개요

Fig.1.은 본 논문에서 제안하는 악성코드 클러스터링 기법의 동작 흐름도이다.

첫 단계는 아래 그림의 1~2번에 해당되는 단계로서 분석 대상 악성코드를 가상 분석 환경을 이용하여 동적 분석하고 API 콜 시퀀스를 획득하기까지의 과정이다. 이 때 분석 결과는 JSON(JavaScript Object Notation) 포맷으로 전송받게 되는데, 이 파일을 파싱하여 API 콜 시퀀스를 획득할 수 있다.

두 번째 단계는 획득한 API 콜 시퀀스에서 distance matrix를 계산하는 과정이다. 이 때 LSH 해시 값과 악성코드 간의 거리를 구할 때는 TLSH 파이썬 라이브러리의 함수를 사용한다.

마지막 단계에서는 distance matrix를 이용하여 클러스터링을 수행한다.

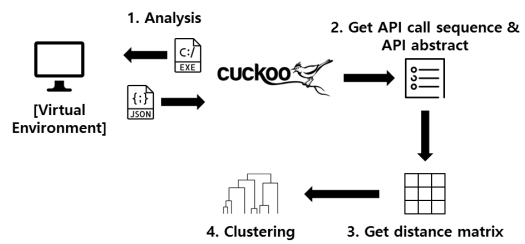

Fig. 1. Architecture of malware clustering

3.2 본 연구와 기존 연구의 차이점

본 연구에서 제안하는 악성코드 클러스터링 기법은 다음과 같은 기존 연구의 문제점을 극복하는 데 주안점을 두었다.

첫째로 기존 API 콜 시퀀스 분석에서는 같은 기능을 수행하는 다른 명칭의 함수에 대한 구분이 어려운 문제점이 존재하였는데, 이를 해결하기 위해 API 추상화 방식을 적용하였다. 같은 기능을 수행하는 API를 같은 의미로 처리하지 않을 경우에는 유사도 측정에 영향을 끼칠 수 있다. 따라서 같은 기능을 수행하는 API는 같은 의미로 인식할 수 있도록 추상화 해 주는 방식을 채용하여 유사도 측정 시 발생할 수 있는 문제를 줄이고자 한다. 기영준 외 2인이 연구한 논문에서도 추상화 기법이 적용되어 있는데[10], 해당 논문에서는 MSDN 정보를 이용하여 API 추상화를 수행하였지만 본 논문에서는 cuckoo sandbox에서 정의한 API 카테고리를 이용하여 추상화 테이블을 생성하여 추상화를 수행하였다.

둘째로 데이터의 길이에 영향을 받지 않는 LSH 알고리즘을 사용하여 유사도 비교를 수행하는 데 발생할 수 있는 노이즈를 줄일 수 있도록 하였다. ssdeep 등의 기존의 일부 LSH 알고리즘은 상이한 길이를 가진 데이터의 유사도를 측정할 때 데이터 길이의 영향을 받아 유사도가 비정상적으로 측정되는 경우가 존재한다. 이 문제를 해결하기 위해 유사도 측정 시 데이터의 길이 차에 영향을 받지 않는 TLSH 알고리즘을 사용하여 문제를 해결하였다.

3.3 API 콜 시퀀스 추출 및 전처리

API 콜 시퀀스를 추출하기 위해 샌드박스 기반의 동적 분석 도구인 cuckoo sandbox를 이용한다. cuckoo sandbox는 2010년 google summer of code project의 the honeynet project에서 시작된 프로젝트로 악성코드의 동적 분석을 위한 샌드박스를 구축하고 분석 결과를 사용자에게 GUI 및 파일로 제공하는 도구이다[19]. cuckoo sandbox는 수많은 API 중 323개 함수의 호출을 기록한다.

cuckoo sandbox를 이용하여 실행 파일을 분석하게 되면, JSON 형식으로 결과가 저장된다. 이를 파싱하여 API 콜 시퀀스를 획득한다.

이 때 추출한 상태 그대로의 API 콜 시퀀스는 상이한 함수 명을 가진 API가 동일한 행위를 하는 경

Table 2. API abstract table

class	description	example	# of APIs
A	file/directory	CopyFile, CreateDirectory, GetFileType, ···	47
B	registry	RegCreateKeyEx, NtCreateKey, RegDeleteValue, ···	38
C	internet explorer	CDocument_write, CScriptElement_put_src, ···	7
D	user interface	DrawText, FindWindow, LoadString, ···	11
E	net API	NetGetJoinInformation, NetShareEnum, ···	6
F	network	DnsQuery_A, GetAdaptersInfo, HttpOpenRequestA, ···	62
G	OLE	CoCreateInstance, CoInitialize, ···	3
H	process	CreateProcess, CreateThread, Module32First, ···	41
I	synchronization	GetLocalTime, GetSystemTime, ···	8
J	resource	FindResource, LoadResource, ···	6
K	services	ControlService, CreateService, ···	12
L	system	GetNativeSystemInfo, LdrLoadDll, NtClose, ···	26
M	certificate	CertControlStore, CertOpenStore, ···	5
N	encryption	CryptCreateHash, CryptGenKey, ···	19
O	exception	SetUnhandledExceptionFilter, RtlDispatchException, ···	6
P	misc	GetUserName, GetDiskFreeSpace, WriteConsole, ···	20
Q	notification	__anomaly__, __exception__, ···	4

우, 유사도 비교에 영향을 줄 수 있는 문제가 발생할 수 있다. 이 문제를 해결하기 위해 함수의 특성에 따라 특정 기호로 추상화하는 전처리 과정을 수행한다.

추출한 API 콜 시퀀스의 각 API의 특성에 따라 1바이트의 알파벳으로 추상화하는 과정을 수행하였다. cuckoo sandbox에서 자체적으로 정의한 기준에 따라 323개 함수를 17가지 카테고리로 Table 2.와 같이 구분하였다.

이와 같은 전처리 과정을 통해 다른 명칭을 가진 함수라도 같은 동작을 수행하는 경우 같은 문자열로 표현될 수 있도록 처리한다.

3.4 TLSH를 이용한 distance matrix 생성

추상화된 API 콜 시퀀스들을 TLSH를 이용하여 해시 계산 과정을 수행한다. 이 단계를 수행하고 나면 추상화된 API 콜 시퀀스들 각각마다 TLSH 해시 값이 하나씩 맵핑된다. 그 후 맵핑된 모든 TLSH 해시 값 간의 유사도를 계산하여 Table 3. 과 같은 형태의 distance matrix를 생성한다.

Table 3. Distance matrix

dist	1	2	3	4	5
1	0.0000	0.3385	0.2945	0.4002	0.2506
2	0.3385	0.0000	0.3480	0.2969	0.3171
3	0.2945	0.3480	0.0000	0.3741	0.2553
4	0.4002	0.2969	0.3741	0.0000	0.4454
5	0.2506	0.3171	0.2553	0.4454	0.0000

3.5 클러스터링

전 단계에서 구한 distance matrix를 이용하여 클러스터링 단계를 수행한다. distance matrix를 사용하는 클러스터링 알고리즘은 계층적 클러스터링, k-medoids 알고리즘 등이 있다.

본 연구에서는 두 가지 클러스터링 알고리즘 모두를 사용하여 분석을 수행하였다.

3.5.1 계층적 클러스터링 (hierarchical clustering)

계층적 클러스터링은 가장 유사한 두 그룹을 계속 병합하는 방식으로 클러스터링을 수행하는 알고리즘이다. 동작 방식을 자세히 살펴보면, 그룹 각각은 한 개 항목을 가지고 군집화를 시작하게 되고 거리를 기반으로 가장 가까운 그룹들을 병합하는 방식을 반복하며 최종적으로 하나의 그룹이 생성될 때 까지 반복하는 방식이다. 이 때 Fig.2.와 같이 덴드로그램 형

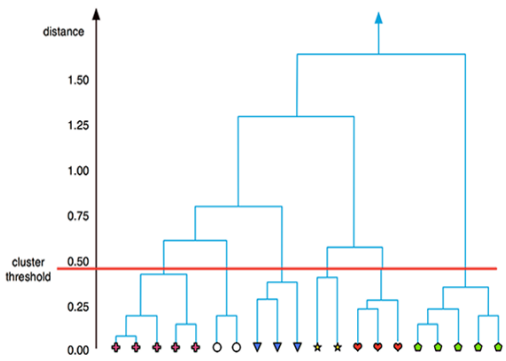

Fig. 2. Hierarchical clustering (height ≈ 0.5)

태로 결과를 표현할 수 있는데, 높이(세로축)나 클러스터 개수를 기반으로 분할하여 클러스터링을 수행할 수 있다. 계층적 클러스터링은 데이터를 직관적으로 확인할 수 있는 장점이 있다.

본 연구에서는 덴드로그램을 구성한 후 유사도 값을 기반으로 클러스터링을 수행하였다. 이 때 클러스터 구분의 기준을 유사도 값 100으로 설정하였는데, 그 이유는 해당 값 정도의 구간이 TLSH 알고리즘을 통한 분석 시에 정확도는 크게 감소하지 않으면서 오탐율을 감소시킬 수 있기 때문이다. 해당 유사도보다 높은 값을 선정하였을 때는 오탐율이 너무 높고, 100보다 작은 값을 선정하였을 때는 정확도 감소가 발생하기 때문이다.

3.5.2 k-medoids 알고리즘

k-medoids 알고리즘은 평균값을 기준으로 하여 가장 가까운 객체들을 군집에 할당하는 k-means 알고리즘과 유사한 알고리즘으로, 평균값을 기준으로

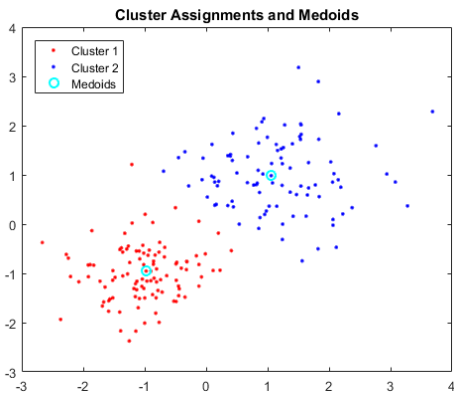

Fig. 3. k-medoids algorithm (k=2)

하여 가장 가까운 객체들을 군집에 할당하는 k-means 알고리즘과는 다르게 Fig.3.과 같이 군집을 대표하는 중앙 객체(medoid)를 선택하고 이 객체를 중심으로 나머지 객체들을 군집에 할당하는 방식을 사용한다.

노이즈나 이상치에 민감한 k-means 알고리즘의 단점을 극복하기 위해 제안되었고, distance matrix를 사용하여 군집화를 수행 할 수도 있다.

IV. 실험 및 결과

4.1 개요

실험에 사용한 데이터 셋은 windows 7 32bit 초기 설치 환경에서 수집한 정상 파일과 malwares.com에서 2016년 8월 1일부터 8월 31일까지 수집한 악성코드 중 일부를 사용하였다[20].

Table 4.에 나타난 악성코드 유형 정보는 카스퍼스키 사의 백신 탐지 결과를 기반으로 획득하였다.

Table 4. Experiment dataset

category	subcategory	# of files
normal	Normal	502
abnormal	AdWare	27
	Backdoor	76
	DangerousObject	55
	Downloader	20
	Trojan-Downloader	27
	Trojan-Dropper	49
	Trojan-PSW	18
	Trojan-Ransom	12
	Trojan-Spy	58
	Virus	146
	Worm	25
	Total	513
total		1,015

4.2 클러스터링 알고리즘에 따른 클러스터링 정확도 비교

우선 cuckoo sandbox를 이용하여 정상 파일과 악성코드의 동적 분석을 수행하고, API 콜 시퀀스를 추출한다. 그 이후 API의 명칭을 기반으로 API의 특성을 나타내는 문자열로 API를 추상화 시켜 주는 전처리 과정을 수행한다.

TLSH를 이용한 distance matrix 생성 과정에

서는 추상화된 모든 API 콜 시퀀스를 입력으로 하여 해시 값을 계산하고, 모든 악성코드 사이의 유사도를 계산하여 distance matrix를 생성한다.

마지막으로 R 언어를 이용하여 k 값을 증가시키며 클러스터링을 수행하였다. Fig.4.는 계층적 클러스터링을 수행한 결과 덴드로그램이다. 해당 덴드로그램에서 y축은 TLSH 해시 값의 유사도 수치를 나타내는 값이다.

Table 5.에 나타난 계층적 클러스터링과 k-medoids 클러스터링을 적용했을 때의 클러스터링 정확도 실험 결과를 살펴보면, Worm을 제외한 모든 카테고리에서 k-medoids 클러스터링이 더 높은 성능을 보임을 확인할 수 있다.

정상 파일을 k-medoids 클러스터링을 이용하여

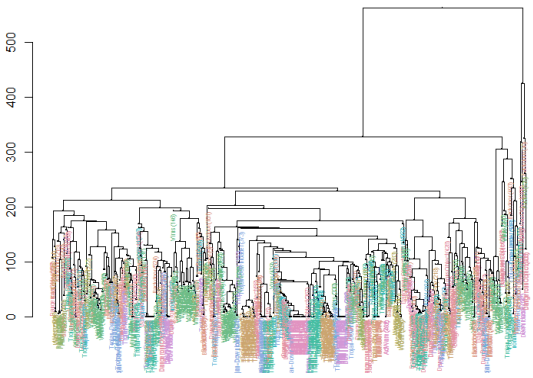

Fig. 4. Hierarchical clustering result

Table 5. Clustering results according to clustering algorithm

subcategory	total	hierarchical		k-medoids	
		#	%	#	%
Normal	502	315	62.75	362	72.11
AdWare	27	17	62.96	18	66.67
Backdoor	76	43	56.58	52	68.42
Dangerous Object	55	28	50.91	38	69.09
Downloader	20	11	55.00	18	90.00
Trojan-Downloader	27	15	55.56	19	70.37
Trojan-Dropper	49	28	57.14	35	71.43
Trojan-PSW	18	9	50.00	12	66.67
Trojan-Ransom	12	7	58.33	10	83.33
Trojan-Spy	58	35	60.34	43	74.14
Virus	146	111	76.03	122	83.56
Worm	25	17	68.00	16	64.00

클러스터링을 수행하였을 때에는 약 72.11%의 정확도로 클러스터링을 수행 해 내었고, Downloader, Virus, Trojan-Ransom 카테고리의 악성코드들은 80% 이상의 정확도로 클러스터링 하였다. 반면 계층적 클러스터링에서는 정상 파일의 클러스터링 정확도가 k-medoids 클러스터링에 비해 10% 정도 낮은 성능을 보이며, 악성코드들 또한 Virus를 제외하면 70% 이상의 정확도로 클러스터링 해 낸 카테고리가 존재하지 않았다. 이를 통해 k-medoids 클러스터링이 악성코드를 클러스터링 하는 데 더 효과적인 것으로 판단할 수 있다.

또한 API 콜 시퀀스의 관점에서 보았을 때 Downloader, Trojan-Ransom, Virus 카테고리의 악성코드들은 유사한 행위적 특징을 가지는 경우가 다른 카테고리에 비해 많다고 판단할 수 있다. 해당 실험에서 특정 클러스터로 잘 클러스터링 된다는 것은 해당 악성코드들이 특성의 유사성을 충분히 가진다는 의미이기 때문이다.

4.3 API 추상화 여부에 따른 클러스터링 정확도 비교

API 추상화로 인한 성능 증가가 얼마나 이루어지는지 확인하고자 API 추상화를 적용한 데이터와 적용하지 않은 두 가지 경우의 클러스터링 결과를 비교하는 실험을 수행하였다.

Table 8.에 나타난 API 추상화 여부에 따른 클

Table 6. Clustering results according to API abstract

subcategory	total	API Abstract			
		X		O	
		#	%	#	%
Normal	502	341	67.93	362	72.11
AdWare	27	16	59.26	18	66.67
Backdoor	76	47	61.84	52	68.42
Dangerous Object	55	32	58.18	38	69.09
Downloader	20	14	70.00	18	90.00
Trojan-Downloader	27	13	48.15	19	70.37
Trojan-Dropper	49	28	57.14	35	71.43
Trojan-PSW	18	12	66.67	12	66.67
Trojan-Ransom	12	8	66.67	10	83.33
Trojan-Spy	58	37	63.79	43	74.14
Virus	146	97	66.44	122	83.56
Worm	25	13	52.00	16	64.00

Table 7. Clustering results according to LSH algorithm

subcategory	total	ssdeep		simhash		TLSH	
		#	%	#	%	#	%
Normal	502	127	25.30	241	48.01	362	72.11
AdWare	27	7	25.93	12	44.44	18	66.67
Backdoor	76	22	28.95	40	52.63	52	68.42
DangerousObject	55	17	30.91	28	50.91	38	69.09
Downloader	20	7	35.00	8	40.00	18	90.00
Trojan-Downloader	27	6	22.22	13	48.15	19	70.37
Trojan-Dropper	49	17	34.69	28	57.14	35	71.43
Trojan-PSW	18	6	33.33	9	50.00	12	66.67
Trojan-Ransom	12	5	41.67	5	41.67	10	83.33
Trojan-Spy	58	19	32.76	18	31.03	43	74.14
Virus	146	43	29.45	60	41.10	122	83.56
Worm	25	9	36.00	15	60.00	16	64.00

러스터링 정확도 실험 결과를 살펴보면, API 추상화를 적용한 경우가 적용하지 않은 경우보다 전체적으로 클러스터링 정확도가 높은 것을 확인할 수 있다. 특히 Downloader, Trojan-Downloader 악성코드의 경우 20% 이상 정확도가 향상되었다. 해당 결과를 분석하였을 때, API 추상화 과정은 클러스터링 정확도 향상에 영향을 끼침을 확인할 수 있다.

4.4 LSH 알고리즘에 따른 클러스터링 정확도 비교

본 논문에서 채택한 TLSH 알고리즘이 클러스터링 성능에 미치는 영향에 대해 분석하고자 ssdeep, simhash 알고리즘을 적용했을 때의 클러스터링 결과를 비교하는 실험을 수행하였다.

Table 7.의 LSH 알고리즘에 따른 클러스터링 정확도 비교 실험 결과를 살펴보면 ssdeep 알고리즘을 적용한 경우 클러스터링 정확도가 50%를 넘는 경우가 없는 것을 확인할 수 있고, simhash 알고리즘 또한 TLSH에 비해 낮은 정확도를 보이는 것을 알 수 있다. 이 결과를 통해 TLSH 알고리즘이 ssdeep이나 simhash 알고리즘에 비해 클러스터링 정확도 향상에 영향을 끼침을 확인할 수 있다.

4. 결 론

본 논문에서는 다량의 악성코드 유포에 따른 기존 분석 방식의 어려움을 극복할 수 있는 악성코드 자동화 분석 방식 중의 하나인 API 콜 시퀀스를 이용한 악성코드 클러스터링 기법을 제안하였다. 본 연구는 LSH 기법을 이용하여 유사한 행위를 수행하는 악성코드를 클러스터링 하는 데 목적을 두고 있고, API 추상화를 통해 기존 API 콜 시퀀스 기반의 악성코드 탐지 기법의 문제점을 극복하려 하였다.

API 콜 시퀀스 획득을 위해 샌드박스 기반의 동적 분석 도구인 cuckoo sandbox를 사용하였고, 해당 도구를 통해 악성코드를 실행시켜 API 콜 시퀀스를 획득할 수 있었다. 다음으로 API 콜 시퀀스 분석의 문제점인 API명에 따라 유사도 계산에 문제가 발생한다는 점을 극복하기 위해 API 추상화 기법을 적용하였다. 또 다른 문제점인 길이가 다른 데이터에 대한 유사도 분석 문제도 길이에 의존적이지 않은 TLSH를 사용하여 극복하였다. TLSH를 이용하여 악성코드의 유사도 해시를 획득하고, 모든 악성코드별로 각 악성코드에 대한 유사도를 구하여 distance matrix로 생성하였다. 이 matrix 값을 이용하여 k 값을 증가시키며 계층적 클러스터링과 k-medoids 클러스터링 기법을 적용하였다.

본 연구의 검증을 위해 1,015개의 정상 및 악성 파일을 이용하여 실험을 수행하였다. 본 실험에서는 k-medoids 클러스터링 알고리즘을 이용한 클러스터링이 더 높은 정확도를 보였다. k-medoids 클러스터링 알고리즘은 72.11%의 정확도로 정상 파일을 클러스터링 하였고, 악성코드들을 전체적으로 66.67% 이상의 정확도로 클러스터링 하였다. 특히 Downloader, Trojan-Ransom, Virus 카테고리의 악성코드는 80% 이상의 클러스터링 정확도를 보였다. Virus 유형의 악성코드는 두 클러스터링 알고리즘 모두 높은 성능을 보였다.

본 연구는 기존 API 콜 시퀀스 기반의 분석에서 API명에 대한 의존성과 데이터 길이 문제들을 API

추상화를 통해 해결하고자 하였고 API 콜 시퀀스 데이터만을 가지고도 악성코드의 유형을 밝혀내는 분석이 충분히 가능함을 보였다는 점에서 의의가 있다.

향후, 본 연구에서 클러스터링 정확도가 낮게 확인된 유형의 악성코드를 대상으로 연구를 지속적으로 진행하여 성능을 개선 할 계획이다. 또한 가상머신 탐지 등의 안티 디버깅 기법이 적용된 악성코드 분석을 위한 우회 기법을 적용하여 연구를 확장할 계획이다.

References

[1] Kaspersky Lab, "Kaspersky Security Bulletin 2015," Kaspersky Lab, Dec. 2015.

[2] S.A. Hofmeyr, S. Forrest and A. Somayaji, "Intrusion detection using sequences of system calls," Journal of computer security, vol. 6, no. 3, pp. 151-180, Jul. 1998.

[3] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze, "Malware detection based on mining API calls," Proceedings of the 2010 ACM symposium on applied computing, pp. 1020-1025, Mar. 2010.

[4] Kyoung-Soo Han, In-Kyoung Kim and Eul-Gyu Im, "Malware Family Classification Method using API Sequential Characteristic," Journal of Security Engineering, 8(2), pp. 319-335, Apr. 2011.

[5] Jae-woo Park, Sung-tae Moon, Gi-Wook Son, In-Kyoung Kim, Kyoung-Soo Han, Eul-Gyu Im and Il-Gon Kim, "An Automatic Malware Classification System using String List and API," Journal of Security Engineering, 8(5), pp. 611-626, Sep. 2011.

[6] Strace Nt, IntellectualHeaven, http://www.strace-nt.com-about.com/

[7] M.K. Shankarapani, S. Ramamoorthy, R.S. Movva and S. Mukkamala, "Malware detection using assembly and API call sequences," Journal in computer virology, vol. 7, no. 2, pp. 107-119, May. 2011.

[8] M. Alazab, S. Venkatraman, P. Watters and M. Alazab, "Zero-day malware detection based on supervised learning algorithms of API call signatures," Proceedings of the 9-th Australasian Data Mining Conference, vol. 121, pp. 171-182, Dec. 2011.

[9] E. Elhadi, M.A. Maarof and B. Barry, "Improving the Detection of Malware Behavior Using Simplified Data Dependent API Call Graph." International Journal of Security and Its Applications, vol. 7, no. 5, pp. 29-42, Sep. 2013.

[10] Youngjoon Ki, Eunjin Kim, and Huy Kang Kim. "A novel approach to detect malware based on API call sequence analysis," International Journal of Distributed Sensor Networks, vol. 11, no. 6, Jan. 2015.

[11] Jae-wook Jang, Jiyoung Woo, Aziz Mohaisen, Jaesung Yun, and Huy Kang Kim, "Mal-Netminer: Malware Classification Approach Based on Social Network Analysis of System Call Graph," Mathematical Problems in Engineering, vol. 2015, Aug. 2015.

[12] Y. Ye, D. Wang, T. Li, D. Ye and Q. Jiang, "An intelligent PE-malware detection system based on association mining," Journal in computer virology, vol. 4, no. 4, pp. 323-334, Nov. 2008.

[13] M. Ahmadi, A. Sami, H. Rahimi and B. Yadegari, "Malware detection by behavioural sequential patterns," Computer Fraud & Security, vol. 2013, no. 8, pp. 11-19, Aug. 2013.

[14] G.G. Sundarkumar, V. Ravi, I. Nwogu and V. Govindaraju, "Malware detection via API calls, topic models and machine learning," 2015 IEEE International Conference on Automation Science and Engineering (CASE), pp. 1212-1217, Aug. 2015.

[15] Y. Li, S.C. Sundaramurthy, A.G. Bardas, X. Ou, D. Caragea, X. Hu and Jiyong Jang, "Experimental study of fuzzy hashing in malware clustering analysis," 8th Workshop on Cyber Security Experimentation and Test (CSET 15), pp. 52-59, Aug. 2015.
[16] J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing," Digital investigation, vol. 3, pp. 91-97, Sep. 2006.
[17] C. Sadowski and G. Levin, "Simhash: Hash-based Similarity Detection," UCSC-SOE-11-07, University of California, Feb. 2011.
[18] J. Oliver, C. Cheng and Y. Chen, "TLSH - A Locality Sensitive Hash," Cybercrime and Trustworthy Computing Workshop (CTC), pp. 7-13, Nov. 2013.
[19] Automated Malware Analysis - Cuckoo Sandbox, Cuckoo Foundation, https://www.cuckoosandbox.org/
[20] malwares.com, SAINT SECURITY, https://www.malwares.com/

〈저자소개〉

고 동 우 (Dong Woo Goh) 학생회원
2015년 2월: 금오공과대학교 컴퓨터공학과 졸업
2015년 3월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 악성코드 분석, 데이터 마이닝

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