Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.30, NO.1, Feb. 2020 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2020.30.1.51

API Call Time Interval을 활용한 머신러닝 기반의 악성코드 탐지

조 영 민,[1†] 권 헌 영[2‡]
[1]삼성SDS(프로), [2]고려대학교(교수)

Machine Learning Based Malware Detection Using API Call Time Interval

Young Min Cho,[1†] Hun Yeong Kwon[2‡]
[1]SamsungSDS (Senior Engineer), [2]Korea University (Professor)

요 약

사이버 위협에 있어서 악성코드를 활용하는 것은 시대를 불문하고 지속적으로 활용되고 있고, 앞으로 IT기술이 발전하여도 여전히 주요한 공격 방법이 될 것이다. 따라서 이러한 악성코드를 탐지하기 위한 연구는 끊임없이 다양한 방법으로 시도되고 있다. 최근에는 AI 관련 기술이 발전하면서 악성코드 탐지에도 이와 관련한 연구를 많이 진행하고 있다. 본 연구에서는 동적분석 데이터 중 API Call이 발생하는 각각의 호출간격, 즉 시간차이(Time Interval)을 중심으로 특징값(Feature)을 생성하고, 이를 머신러닝 기법에 적용하여 악성코드를 탐지하는 방안을 제시하고자 한다.

ABSTRACT

The use of malware in cyber threats continues to be used in all ages, and will continue to be a major attack method even if IT technology advances. Therefore, researches for detecting such malicious codes are constantly tried in various ways. Recently, with the development of AI-related technology, many researches related to machine learning have been conducted to detect malware. In this paper, we propose a method to detect malware using machine learning. For machine learning detection, we create a feature around each call interval, ie Time Interval, in which API calls occur among dynamic analysis data, and then apply the result to machine learning techniques.

Keywords: Machine Learning, Malware Detection, Time Interval, AI

I. 서 론

악성코드는 외부 해커가 기업 내부에 침입할 때 활용하는 오래된 방법 중의 하나로, 지금도 가장 유효하고 확실한 방법이다. 따라서 악성코드를 이용한 보안 위협은 지속적으로 지능화되고 증가하고 있다. 글로벌 보안업체 카스퍼스키에 따르면 2018년 한해 동안 신종 악성코드를 일평균 34만6천개를 탐지하였으며, 이는 2017년 7만개 수준에 비해 5배 증가하였다고 밝혔다[1].

악성코드의 증가와 제작 기술의 발전으로 이를 탐지하기 위한 연구 또한 다양하게 시도되고 있다. 악성코드를 탐지하는 방법은 특정한 문자열이나 함수사용과 같은 시그니처를 기반으로 하는 방법(Signature Based Technique)과 가상의 환경에서 파일을 실행시키면서 악의적인 메모리 호출이나 추가 파일 다운로드와 같은 행동을 분석하는 행위기반(Behavior Based Technique)로 크게 구분할

Received(12. 02. 2019), Modified(02. 11. 2020), Accepted(02. 12. 2020)
† 주저자, ymin_cho@naver.com
‡ 교신저자, cabkwon@gmail.com(Corresponding author)

수 있으며, 이를 다시 조합하면 정적분석(Static Analysis), 동적분석(Dynamic Analysis), 복합분석(Hybrid Analysis)로 구분할 수 있다[3].

그러나 다양한 탐지 연구가 진행되고 있음에도 여전히 악성코드 탐지 장비를 우회하는 새로운 기술들이 등장하므로 모든 악성코드를 100% 탐지하는 기술은 존재하기 어렵다. 이번 연구에서는 기존 연구에서 사용한 다양한 동적분석 내용들을 쓰지 않고, API Call의 시간간격 정보만을 활용하여서도 의미있는 탐지 결과를 낼 수 있음을 검증하고자 한다.

II. 관련 연구

악성코드 분석방법은 Fig.1에서 보는바와 같이 주어진 프로그램 코드 자체를 그대로 분석하는 정적분석 방법과, 파일을 직접 실행시키며 메모리, 레지스트리 사용 등의 활동을 분석하는 동적분석 방법이 대표적이다. 동적분석 방법 중 가장 대표적인 방법이 API Call을 기반으로 하는 연구이며, API Call 정보를 기반으로 하는 다양한 연구가 진행되어 왔다.

API Call Sequence 기반 연구는 호출되는 API의 순서에 의미를 부여하는 것이다. 악성코드가 동작하는 방식의 특성이 유사할 수 있고, API를 호출하는 순서를 통해 비슷한 악성코드를 탐지할 수 있다[3][4]. API Call 관련 연구의 또 다른 유형으로 API Call Name 기반 연구가 있다. 악성코드가 호출한 API 자체에 무게를 두고 특성을 파악하는 연구다[5][6]. 또 하나의 케이스는 API 호출 관계를 통한 분석방법이다. 전체 호출된 API Call의 관계도 기반 분석 방법이다[7][8]. 이렇게 다양한 시도를 통해 기존에 탐지하기 어려웠던 악성코드를 새롭게 탐지할 수 있는 계기들이 마련되고 있다. 본 연구에서는 이러한 방법과 동일하게 API Call 정보를 토대로 하되, 시간 특성이라는 정보, 특히 시간 간격(Time Interval) 정보를 토대로 악성코드를 분류하는 방법을 제시하고자 한다.

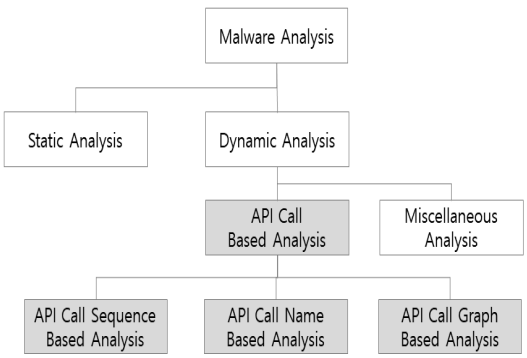

Fig. 1. Related Research Field

III. Time Interval 기반 탐지 방안

3.1 가정 및 전제 사항

유사한 악성코드의 군집을 지칭하는 악성코드 패밀리(Malware Family)는 시간의 흐름에 따라 유사한 행위를 할 것이라는 가정을 통해 진행하였으며, 실제로 Fig.2에서 보이는 바와 같이 악성코드 일부 샘플을 토대로 API호출 순서별 시간간격을 그래프로 도식하였을 때 대체로 비슷한 유형을 보이는 것을 발견하였다. X축은 API Call을 호출한 횟수이므로 실행파일마다 X축의 길이가 달라진다. Y축은 각각의 API Call 호출과 다음 호출 사이의 시간간격을 의미한다.

또한 본 논문은 API Call 마다의 시간 정보를 핵심 정보로 활용하므로, 기본적으로 API Call 데이터가 필요하다. API Call의 시간정보를 뽑아내기 위해서는 실제 파일을 실행해야 알 수 있으므로 동적분석 장비인 Sandbox를 필수적으로 사용해야 한다. 요즘은 악성코드에 대한 관심이 높아지면서 많은 기업들이 자체적으로 상용 Sandbox 장비를 도입하여 활용하고 있다. Sandbox 상용장비가 없는 경우에는 쿠쿠(Cuckoo)와 같은 오픈소스 소프트웨어 기반의 무료 기능을 활용하여 데이터를 확보할 수 있다.

본 논문에서는 저자가 속한 기업에서 이미 보유하고 있는 Sandbox 장비를 활용하여 데이터를 추출하였다. 따라서 데이터의 형식이 특정 기업 제품이 생성한 데이터 포맷(Format)에 맞춰져있고, 이에 따라 최초 분석이 진행되는 원천 데이터 자체가 특정 회사 제품의 종속성을 가지고 있다. 글로벌 보안업체 F사의 제품이 생성하는 json파일을 활용하였다. 향후에는 보다 범용적인 활용을 위해 오픈소스 Sandbox 장비 기반 연구도 진행할 예정이다.

3.2 탐지 방안

3.2.1 탐지 방안 개요

전체 구성은 Fig.3처럼, 실행파일(PE File)에서 API Call 정보와 시간 정보 데이터를 파싱하고 시간 간격을 계산한다. 전체 API Call 횟수를 기준으로 초기, 중기, 말기 3단계(3-Steps)로 구분하여 특징 값을 도출하며, 이렇게 도출된 특징 정보를 토대로 머신러닝을 실시하고 만들어진 모델을 통해 파일을 정상과 악성으로 분류하는 구조다.

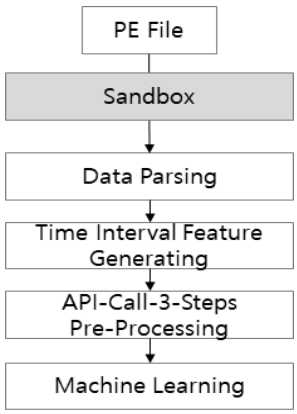

Fig. 3. Overall Structure

3.2.2 Data Parsing

상용 Sandbox 제품에서 분석한 동적분석 결과 데이터(json)에서 API Call 호출과 관련한 정보와 각각의 시간정보(timestamp)를 가져온다.

JSON 파일은 Fig.4.에서와 같이 API Call의 횟수와 각 호출 시 apiname, dllname 등 부가 정보를 보여준다. 또한 진단명의 유무에 따라 정상과 악성으로 구분되어 생성된다. 각각의 JSON 파일을 읽어서 API Call 부분을 추출하는 작업을 진행하며, 추출한 결과를 파이프구분자(|)로 재결합하여 텍스트 파일로 저장하였다.

3.2.3 Time Interval Feature Generating

API 호출 정보에서 시간간격을 계산하여 특징 값으로 만든다. 예를 들어 〔CreateProcess〕를 호출하고 이후에 〔Sleep〕을 호출하였다면, 두 호출의 Timestamp를 기준으로 시간 간격을 계산한다.

즉, 두 호출정보의 Time Interval은 〔Sleep

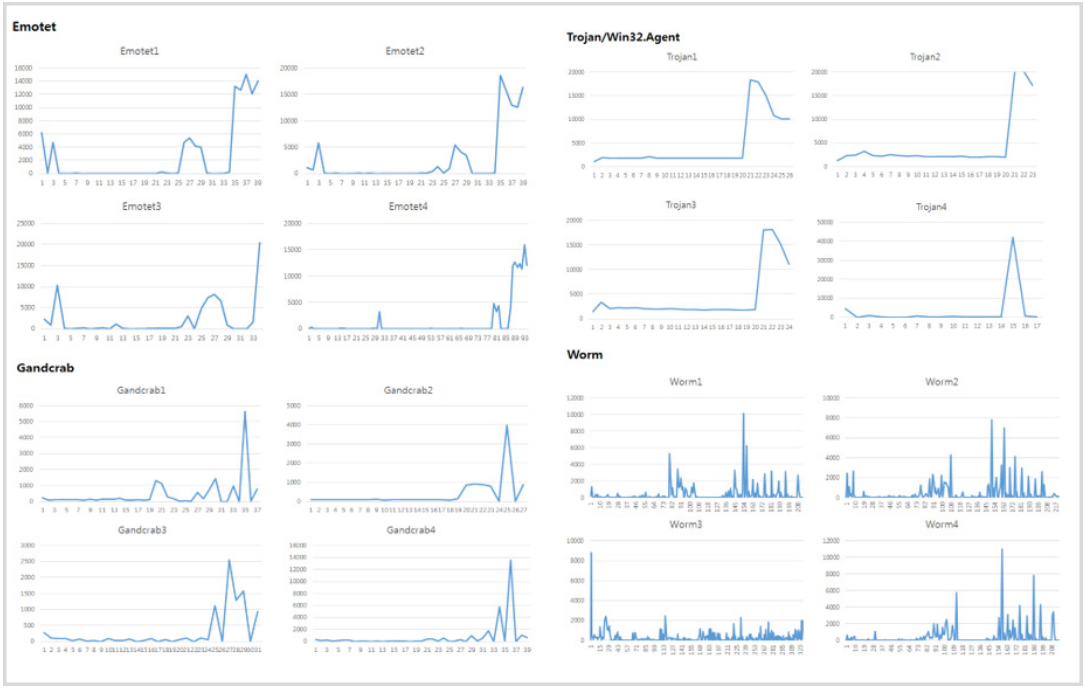

Fig. 2. Pre-Sampling Test Results (X axis: Number of Api call sequence, Y axis: Time Interval)

Fig. 4. JSON Sample

timestamp〕 - 〔CreateProcess timestamp〕로 계산한다.

그밖에 평균 소요시간, Time Interval 급감, 급증 기울기 값, 전체 소요시간 등 시간간격 관련 특징 값을 도출하였다. 급증과 급감은 시간간격 차이가 갑자기 증가한 최대치와 감소한 최대치를 의미하며 다음 시간간격과 이전 시간간격의 차이가 가장 큰 경우와 가장 작은 경우가 해당된다. 즉, 그래프로 표현할 경우 기울기가 가장 급격히 증가하는 경우와 기울기가 급격히 감소하는 부분에 해당한다.

Time Interval 최대치(max)는 각각의 Call 사이의 시간차가 가장 큰 것을 의미하고, 최대 기울기(max-slope)는 이전 시간차와 이후 시간차를 비교했을 때 가장 경사도가 급격한 것을 의미한다. 추가로, 급감과 급증의 경우는 어떠한 API Call일 때 발생하는지를 참고하기 위해 해당 API name을 따로 기록하였다.

Time Interval of N
= Timestamp[n+1] – Timestamp[n]

Fig. 5. Calculation of Time Interval

3.2.4 API-Call-3-Steps Pre-Processing

전체 호출된 API를 단순히 시간간격을 구하는 것 이외에 단계를 초기, 중기, 후기의 3단계로 구분하여 특징을 추출하였다. 이는 API를 호출한 횟수가 많아질수록 변화량이 작은 시간간격이 증가 할 수 있고, 결국 평균 시간차를 계산할 경우 변화량이 큰 호출 정보의 의미를 축소시킬 수 있으므로 이를 방지하기 위해 단계별 구분을 시도하였다.

전체 호출 횟수가 악성코드에 따라 무작위의 횟수가 발생하므로 발생 횟수를 그룹으로 묶어 단계를 적용하였다. 많은 악성코드가 실행초기 탐지 회피를 위해 동작하는 초기 부분과 실제 주요한 파일의 내용이 동작하는 중간부분, 그리고 파일의 행위를 마무리하는 마지막 부분으로 구분하여 초기, 중기, 후기로 나누는 방법을 적용하였다.

API 전체 call을 3단계로 나누는 방법은 3으로 나눈 나머지를 고려하여 경우의 수를 모두 실험에 적용해보고 성능 결과가 가장 우수한 방법을 채택하는 방식으로 구분하였다. 예를 들면, 전체 호출 건수가 10개일 경우, (3,3,4), (3,4,3), (4,3,3)의 세가지 경우를 도출하고 이를 모두 적용하여 최적의 방법을 선정하였다.

3개의 단계를 나눈 후, 각 단계별 특징 값을 도출하였다. 각 단계별 time interval의 표준편차, 평균, 최대값 등을 계산하였고, 부가적으로 첫 번째 API name, 마지막 API name 등을 찾아내어 One-hot-encoding을 실시하였다.

Table 1. How to divide API Call into 3 Steps

Total Call Count	Early	Middle	Late
3n	3m	3m	3m
3n+1	3m	3m	3m+1
3n+2	3m	3m+1	3m+1

3.2.5 Machine Learning

본 논문에서는 특징 값을 도출한 다음 머신러닝을 적용하는 단계에서 랜덤포레스트(Random Forest)를 사용하였다. 랜덤포레스트는 여러 개의 결정트리(Decision Tree)를 이용하여 다수의 결과를 최종으로 결정하는 방법이다〔10〕. 랜덤포레스트는 앙상블(ensemble) 기법을 이용하여 단일한 결정트리를 사

용하는 것보다 우수한 성능을 나타내는 장점을 가지고 있으며, 결과를 해석하는데 유용하다는 장점도 가지고 있다. 실무에 적용한 여러 가지 머신러닝 알고리즘 사용 경험에 비추어 볼 때, 분류 목적을 달성하는데 적합하여 본 논문에서는 랜덤포레스트를 사용하였다. 세부 방법과 사용한 데이터 등 실험 관련 내용은 뒷부분의 IV.실험 결과 및 분석에서 다룬다.

3.3 탐지 한계

본 논문에서는 API Call 항목 중 시간간격이라는 특징에 의미를 두고 탐지를 진행하였다. 이는 기존에 인식한 악성코드와 신규로 유입되는 악성코드는 제작 과정과 실행과정에서 유사성을 가지고 있다는 가정을 바탕으로 하므로, 특수한 목적을 가지고 새롭게 제작된 악성코드의 경우 전혀 새로운 시간특징을 나타내어 탐지가 어려울 수 있다. 또한, 시작이 되는 원천데이터를 특정 sandbox 장비에 의존하게 되므로 해당 장비가 오작동하거나 분석 결과가 미흡할 경우, 또는 sandbox를 우회하는 악성코드인 경우에 탐지가 어렵다. 본 논문에서는 완전히 새로운 환경에서 신규 악성코드를 탐지하겠다는 목표가 아닌, 기존에 동적 분석 장비를 보유하고 있는 기업 및 기관에서 이에 대한 추가 보완수단 목적으로 제안되었다.

IV. 실험 결과 및 분석

4.1 실험 환경 및 데이터

본 논문의 탐지 방안을 실험하기 위해 전용 서버에 환경을 구축하였다. 서버는 CPU 8core, 메모리 192GB 사양이며, 리눅스 계열의 CentOS 7.4와 개발언어 파이썬(python) 3.7버전을 사용하였고, 머신러닝을 위해 파이썬 기반 머신러닝 라이브러리인 사이킷런(Sci-kit Learn)을 사용하여 실험을 진행하였다.

실험 데이터는 한국인터넷진흥원(KISA)의 2018년 데이터(2018 KISA 정보보호 챌린지)와 저자의 소속기업내 유입된 파일을 합쳐 총 66,628개의 파일을 대상으로 실시하였다. 악성코드 데이터를 인터넷에서 무작위로 수집할 경우, 특정 악성코드 패밀리에 속하는 파일이 다수 수집되어 데이터 편향을 나타낼 수 있다고 판단되어, 데이터의 균형을 갖추고 있는 KISA 제공 데이터를 사용하게 되었다. 데이터의 정상파일과 악성파일의 비율은 1:1로 균등하게 하였으며 실험 시 학습데이터(train data)와 검증데이터(test data)는 실무에서 활용하면서 최적의 결과를 보여주었던 8:2의 비율로 구성하였다.

실험에는 시간간격 정보를 토대로 도출한 총 32개의 특징정보(feature)를 선정하여 사용하였다.

Table 2. Feature List

No.	Feature	Remarks
1	Total API Call Type	Total number of API Call types
2	Total API Call Count	Total number of API Call
3	First TimeStamp	First call time
4	Last TimeStamp	Last call time
5	Total Time Interval	Total call duration
6	Max Time Interval	Maximum time interval of all
7	Max Time Interval Index	Maximum time interval point of all
8	Early-Std	Early stage time interval standard deviation
9	Middle-Std	Middle stage time interval standard deviation
10	Late-Std	Late stage time interval standard deviation
11	Early-Mean	Early stage time interval mean
12	Middle-Mean	Middle stage time interval mean
13	Late-Mean	Late stage time interval mean
14	Early-Max	Early stage maximum time interval
15	Middle-Max	Middle stage maximum time interval
16	Late-Max	Late stage maximum time interval
17	Max API Name	API name at maximum time

No.	Feature	Remarks
		interval
18	First API Name	First call API name
19	Last API Name	Last call API name
20	Average Time	API call average time
21	Early-Min -Slope	Minimum time interval slope in early stage
22	Middle-Min -Slope	Minimum time interval slope in middle stage
23	Late-Min -Slope	Minimum time interval slope in late stage
24	Early-Max -Slope	Maximum time interval slope in early stage
25	Middle-Max -Slope	Maximum time interval slope in middle stage
26	Late-Max -Slope	Maximum time interval slope in late stage
27	Early-Min -API Name	Minimum time interval API name in early stage
28	Middle-Min -API Name	Minimum time interval API name in middle stage
29	Late-Min -API Name	Minimum time interval API name in late stage
30	Early-Max -API Name	Maximum time interval API name in early stage
31	Middle-Max -API Name	Maximum time interval API name in middle stage
32	Late-Max -API Name	Maximum time interval API name in late stage

4.2 실험 결과 및 분석

실험은 우연에 의한 테스트 결과를 방지하기 위해 학습과 테스트를 포함하여 총 3회를 실시하였다. 8:2로 나눈 데이터를 활용하여, 비율 8의 데이터를 모델을 학습시키는 Train에 적용하고, 나머지 2의 데이터를 모델 성능을 검증하는 Test에 적용하였다. 정확도(Accuracy) 뿐만 아니라 민감도(Precision), 재현률(Recall) 모두 3회씩 측정하였으며, 3회의 평균 정확도(accuracy)는 91%로 의미 있는 결과가 도출되었다. 세부 결과는 Table 3.의 내용과 같다.

Table 3. Result of Verification

	Train		Test	
1st	Accuracy	98.7%	Accuracy	91.4%
	Precision	98.9%	Precision	91.5%
	Recall	98.5%	Recall	91.0%
2nd	Accuracy	98.6%	Accuracy	91.3%
	Precision	99.0%	Precision	91.5%
	Recall	98.2%	Recall	91.1%
3rd	Accuracy	98.6%	Accuracy	91.4$
	Precision	98.9%	Precision	91.5%
	Recall	98.3%	Recall	91.4%

각 탐지 내역을 분석한 결과 Table 4.에서 보는 바와 같이, 상용으로 기존에 쓰고 있는 Sandbox 장비에서 놓친 5개의 파일을 머신러닝으로 탐지하는 결과를 나타냄으로써, 본 실험의 목적인 상용 장비를 추가 보완하는 수단으로 활용이 가능한 것으로 확인되었다. 추가 탐지된 5개의 파일 중 4개는 트로이목마(Trojan/Win32.Agent)였고, 1개는 PUP(PUP/Win32.DealPly.C2054160)로 진단되었다. Table 4.의 내용은 동적분석 장비(Sandbox)에서 돌린 결과와 머신러닝 결과(ML)을 비교한 값이며, 실제 악성 여부를 라벨(Label)로 표시하였다.

Table 4. Result of ML Detection

MD5	Sand box	ML	Label
01b6d419530539e53db924a13e76fb23	B	M	M
00751f6a37e29ec7dd78d1ddfb6428e2	B	M	M
01cb2729f7d73edd68721f6ec94cf1d8	B	M	M
17d895537c4eafa43ef4d4654f504365	B	M	M
e1fea5c9db3aadc3217591e9a2ec8472	B	M	M

(B : Benign, M : Malignant)

V. 결 론

본 제안의 실험을 수행한 결과 악성코드 동적분석 장비에서 생성된 API Call의 Time Interval 정보를 활용하여 의미 있는 탐지 결과를 낼 수 있음을 확인하였다. 본 논문의 탐지 방안은 동적분석 장비의 결과를 기반으로 하기 때문에 장비가 필수적으로 요구되고 이를 통해 제공되는 데이터에 제한적일 수 밖에 없는 한계를 가지고 있다. 하지만 이러한 한계에도 불구하고 비교적 간단한 실험환경을 구성하여 시간간격 정보만으로도 기존 장비를 보완하는 용도로 활용이 가능함을 확인하였다.

향후에는 분석 단계 또한 현재의 3단계(초기, 중기, 후기)를 넘어 여러 가지 단계별로(2단계, 5단계, 10단계 등) 자동으로 구분하여 실험하고 최적의 결과를 도출하는 형식으로 탐지 성능을 높이는 연구를 수행할 계획이다.

References

〔1〕 "Average 346,000 new malicious codes detected in 2018", ITWorld, 12.13.2018, http://itworld.co.kr/news/113070

〔2〕 Kirti Mathur and Saroj Hiranwal, "A Survey on Techniques in Detection and Analyzing Malware Executables", International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 3, Issue 4, 2013.

〔3〕 Shankarapani, M.K, Mukkamala, S, "Malware detection using assembly and API call sequences.", Journal in Computer Virology. 7(2):107-119, May. 2011.

〔4〕 Youngjoon Ki, Eunjin Kim, "A Novel Approach to Detect Malware Based on API Call Sequence Analysis.", International Journal of Distributed Sensor Networks, Vol. 11, 2015.

〔5〕 Alazab, M, Venkatraman, S, "Zero-day malware detection based on supervised learning algorithms of API call signatures," Conferences in Research and Practice in Information Technology Series, 121:171-182, 2010.

〔6〕 Fujino, A, Mori, T, "Discovering similar malware samples using API call topics," IEEE, CCNC, :140-147, July. 2015.

〔7〕 Ammar Ahmed E. Elhadi, Mohd Aizaini Maarof, "Improving the Detection of Malware Behaviour Using Simplified Data Dependent API Call Graph," International Journal of Security and Its Applications, Vol. 7, Issue 5, pp.29-42, 2013.

〔8〕 Im, E.G, Han, K.-S, "Detection methods for malware variant using API call related graphs," LNEE, :607-611, 120. 2012.

〔9〕 Wikipedia, "Randomforest" https://en.wikipedia.org/wiki/Random_forest, 02.11.2020.

〈저자소개〉

조 영 민 (Young Min Cho) 정회원
2002년 2월: 고려대학교 물리학과 졸업
2019년 2월: 고려대학교 정보보호대학원 석사
2019년 3월~현재: 고려대학교 정보보호대학원 박사 과정
2010년 6월~현재: 삼성SDS 연구원
〈관심분야〉 보안인식, 사회공학, 머신러닝, 인공지능, 빅데이터, 위협정보

권 헌 영 (Hun Yeong Kwon) 종신회원
1992년 2월: 연세대학교 법학과 졸업
1998년 2월: 연세대학교 법학과 석사
2005년 2월: 연세대학교 법학과 박사
2015년 9월~현재: 고려대학교 정보보호대학원 교수
〈관심분야〉 정보보호법 및 정책, 정보통신법 및 정책, 사이버법률, 인터넷규제, 전자정부