Journal of KIIT. Vol. 17, No. 7, pp. 111-122, Jul. 31, 2019. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2019.17.7.111

머신러닝을 활용한 실시간 리눅스 악성파일 탐지

전덕조*, 박동규**

Real-time Linux Malware Detection Using Machine Learning

Deok-Jo Jeon*, Dong-Gue Park**

이 논문은 순천향대학교 연구비에 의하여 연구하였음.

요 약

최근 IoT 혁명과 임베디드 장치 채택의 급증으로 악성코드 환경이 빠르게 변화하고 있다. 개인용 컴퓨터는 주로 x86 기반 아키텍처에서 실행되지만 임베디드 시스템은 다양한 아키텍처에 의존하고 있다. 이러한 측면에서 많은 시스템이 Linux 운영 체제 및 변종을 실행하고 있어 공격자는 “리눅스 악성코드“를 만들어내고 있다. 더욱이 리눅스 시스템에서 사용할 수 있는 안티 바이러스 및 샌드박스 솔루션은 매우 드문 현실이다. 따라서 리눅스 악성코드 탐지 방안을 개발하는 것이 필수 불가결한 상황하다. 리눅스 악성코드 문제를 해결하기 위하여, 본 논문에서는 리눅스 악성코드를 실시간에 탐지가 가능한 시그니처를 사용하지 않는 머신 러닝 기법을 제안한다. 본 논문에서는 보안 분석가들에게 악성 분류 이유에 대한 직관을 제시하기 위해서 실행파일로부터 추출한 각 특징의 정보 이득에 결정 트리(DT) 기법을 적용한다. 실험을 통하여 제안한 방안이 높은 탐지율과 낮은 오탐율을 갖고 있음을 증명한다

Abstract

Recently, the IoT revolution and surge in adoption of embedded devices are rapidly changing the malware environment. While personal computers run predominantly on x86-flavored architectures, embedded systems rely on a variety of different architecture. In turn, this aspect causes a large number of these systems to run some variants of the Linux operating system, pushing malicious actors to give birth to "Linux malware". Also, there are not many Anti-Virus and sandboxing solutions for Linux systems. So, it is indispensable to develop Linux malware detection methods. To solve them, we propose a real-time, non‐signature-based machine learning technique for detecting Linux malware. To provide human-comprehensible reasons to network security analysts, our method uses a decision tree technique on the Information Gain of each feature extracted from executable files. And we verify that our method has high accuracy at a low false positive rate by implementing it.

Keywords

linux malware detection, data mining, machine learning, decision tree, elf ransomware detection

* ㈜ 시큐비스타 대표이사
- ORCID: http://orcid.org/0000-0002-0343-3384
** 순천향대학교 정보통신공학과 교수(교신저자)
- ORCID: http://orcid.org/0000-0002-5864-8825

· Received: Jun. 07, 2019, Revised: Jul. 19, 2019, Accepted: Jul. 22, 2019

· Corresponding Author: Dong-Gue Park
Dept. of Information and Communication Engineering Soonchunhyang Univ.
Tel.: +82-41-530-1347, Email: dgpark@sch.ac.kr:

Ⅰ. 서 론

최근 IoT 혁명과 임베디드 장치 채택의 급증으로 악성코드 환경이 빠르게 변화하고 있다. 특히 임베디드 시스템은 많은 시스템이 리눅스 운영 체제에서 실행하고 있어 공격자는 "리눅스 악성코드"를 만들어내고 있는 상황이다. 2018년 iotall에 따르면, IoT 디바이스가 보다 완벽한 기능을 갖추게 되면서, 이를 구동하는 운영 체제가 RTOS(Real Time Operating Systems)에서 리눅스로 전환되고 있음을 알 수 있다[1]. 또한 Eclipse Foundation의 "IoT 개발자를 위한 주요 경향 2018"에 의하면, IoT 디바이스, 게이트웨이 및 클라우드 백 엔드 디바이스 등에 사용되는 운영체제 중 리눅스 점유율은 71.8% 에 달하고 있으며, 산업 분야 디바이스에 리눅스 적용 역시 확대되고 있음을 알 수 있다[2]. 그리고 오늘날 가장 성공적인 비즈니스를 진행중인 구글 또한 IBM/레드햇/노벨/인텔/오라클/HP/노키아/SGI/소니 등 다수 업체들과 마찬가지로 리눅스 커널 팀을 운영하고 있다[3][4].

2019년에 클라우드 컴퓨팅에서 가장 널리 쓰이고 있는 클라우드 리눅스 OS는 클라우드 컴퓨팅을 위해 특별히 제작 된 배포 판으로, 전용 및 공유 서버를 위한 최고의 보안 기능을 갖추고 있으며, 최대 2,000 만 개의 웹 사이트를 지원하고 있다고 밝히고 있다. 또한 아마존의 아마존 리눅스는 아마존 웹 서비스에서 사용되는 리눅스로서 클라우드 컴퓨팅에서 2번째로 많이 쓰이고 있는 운영체제로 꼽히고 있다[5].

결과적으로 IoT 및 클라우드 컴퓨팅분야에서 리눅스가 널리 쓰이고 있기 때문에 리눅스는 해커에게 가장 선호되는 대상이 되고 있으며, 이러한 시장 점유율은 리눅스 호스트에 대한 공격이 매력적인 포인트가 될 것이다. 아직까지는 리눅스 악성 코드가 윈도우에 비해서는 적은 경향이 있으나, 향후에는 심각한 위협으로 대두될 것으로 예상된다.

본 논문에서는 이러한 문제를 해결하기 위한 방안으로 리눅스 악성코드의 ELF(Executable and Linking Format) 파일 헤더의 구조 정보를 분석하여 악성 코드를 악의적인 파일과 정상 파일을 구별하는 데 사용할 수 있는 일련의 구조적 특징을 식별하여 추출하고, 정보 이득(Information gain) 계산 기법을 이용하여 이들 중에서 적합한 특징을 선별하여 머신러닝 기법 중의 하나인 결정 트리(DT, Decision Tee) 기법을 적용함으로써 높은 탐지율과 낮은 오탐율을 가지는 리눅스 악성코드 탐지 방안을 제시한다. 이러한 방안은 정적인 특징만을 사용하므로 실시간 탐지가 가능하며 결정 트리 기법은 악성코드 분류 과정에 대한 정보를 제공함으로써 보안 분석가가 전체 분류 과정을 이해할 수 있도록 하는 장점을 제공할 수 있다.

Ⅱ. 관련 연구

ELF는 바이너리 파일로서, Unix System Laboratory에서 ABI(Application Binary Interface)의 일부로 개발되어 발전하고 있다[6]. TIS(Tool Interface Standards committee)에서 32 비트 인텔 아키텍처 환경에서 동작하는 이식 가능 객체 파일 포맷으로 ELF 표준을 정의하였다. ELF표준은 프로그래머에게 여러 운영체제 환경으로 확장될 수 있는 바이너리 인터페이스 정의들의 집합을 제공한다[6]. 따라서, 프로그래머들은 바이너리 파일의 인터페이스를 중심으로 프로그램을 할 수 있는 방법을 통해서 다른 리눅스 운영체제 적용을 위해서 새롭게 코드를 작성하고 재 컴파일해서 적용할 필요가 없다. 객체 파일은 크게 3가지가 존재하며, 각각은 아래와 같이 정의될 수 있다[7].

* 재배치 가능 파일(Relocatable file) - 링크를 쉽게 할 수 있는 코드 및 데이터와 함께, 실행 파일을 만들거나 공유 객체 파일을 생성하기 위한 다른 객체 파일을 보유하고 있는 형태의 파일이다.

* 실행 가능 파일(Executable file) - 실행을 위해서 적합한 프로그램을 보유하는 파일로서, exec시스템 콜을 통해서 프로그램이 어떻게 프로세스의 이미지를 만들지를 기술하는 형태의 파일이다.

* 공유 객체 파일(Shared object file) - 2개의 환경상에서 링크에 적합한 형태의 코드와 데이터를 보유한 파일로서, 먼저 LD와 같은 링크 에디터가 이 파일과 함께, 다른 재배치 가능 파일과 공유 객체

파일을 처리해서 또 다른 하나의 객체 파일을 생성하고, 이를 다시 동적 링커(Dynamic linker)가 생성한 객체 파일을 실행 가능 파일과 객체 파일을 묶어서 프로세스의 이미지를 생성해 주는 두 단계를 거치는 파일이다.

어셈블러나 링크 에디터에 의해서 생성된 객체 파일은 프로세서에서 실행될 프로그램의 바이너리 형식이다. ELF 파일의 형식을 보면 그림 1과 같다. 즉, 객체 파일은 크게 두 가지의 관점에서 볼 수 있는데, 프로그램의 링킹 관점과 실행 관점이다[7].

Linking View	Execution View
ELF Header	ELF Header
Program Header Table(optional)	Program Header Table(optional)
.Text Section	.Text Section
.Data Section	.Data Section
…	.Data Section
Section Header Table	Section Header Table(optional)

그림 1. 실행 및 링킹 가능 형식(ELF) 구조
Fig. 1. Executable and linkable format structural view

그림 1에서 중요한 부분만 살펴보면, ELF 헤더는 파일의 구성을 나타내는 로드 맵과 같은 역할을 하며, 파일의 맨 앞부분을 차지한다. 섹션(section)은 링킹을 위한 객체 파일 정보를 다수 포함하고 있는데, 명령(Instruction), 데이터(Data), 심볼 테이블(Symbol table), 재배치 정보(Relocation information) 등이 포함된다. 프로그램 헤더 테이블(Program header table)은 옵션이며, 시스템에 어떻게 프로세스 이미지를 생성할 것인가를 지시한다. 프로세스의 이미지를 만들기 위해서 사용되는 파일은 반드시 프로그램 헤더 테이블을 포함하여야 하며, 재배치 가능 파일의 경우에는 프로그램 헤더 테이블이 포함되지 않을 수 있다. 섹션 헤더 테이블은 파일의 섹션에 대해서 기술하는 정보를 포함하고 있는데, 모든 섹션은 섹션 헤더 테이블에 하나의 항목 이상을 포함하여야 한다. 각각의 항목은 섹션 이름이나, 섹션의 크기와 같은 정보를 제공한다. 만약 파일이 링킹 된다면, 반드시 섹션 헤더 테이블을 포함하여야 하며, 실행 객체 파일은 섹션 헤더 테이블을 포함하지 않을 수 있다[7].

비 시그니처 방식의 악성코드 탐지 기법은 분석 유형에 따라 정적 분석 및 동적 분석 기법으로 분류되는데, 동적 분석 방식은 가상환경에서의 실행을 전제로 하기 때문에 일반적으로 약 15분의 시간이 소요된다. 본 논문에서는 실시간 탐지를 위하여 정적 분석을 기반으로 하는 비 시그니처 기반 악성코드 탐지 방식을 제안하고자 하며, 특히 악성코드 분류 과정을 보안 관리자가 이해할 수 있는 방안을 제시하고자 한다. 이와 관련된 이전의 주요 연구는 다음과 같다.

Shafiq[8]-[10]은 악성 PE 파일을 탐지하는 두 가지의 연관된 기법을 제안하였다. 첫 번째 방법은 PE 파일로부터 구분되는 특징을 추출하여 주성분 분석(PCA) 등의 전처리 기법 적용하여 여러 분류 알고리즘의 입력 값으로 사용하였다. 그러나 이러한 방법은 패킹되지 않은 실행파일로 학습하고 패킹된 실행파일을 테스트하는 경우, 성능이 저하되는 단점이 있었다. 이를 개선하기 위하여 패킹 또는 패킹되지 않은 파일을 식별하고, 패킹된 파일에 특화된 분류기와 패킹되지 않은 파일에 특화된 분류기를 활용하는 방안을 제안하여 실시간 탐지 성능 및 높은 정확성을 제시하였다. Yan[11]은 이미지 자원 NameID 필드 및 특성(Characteristic)을 제외한 모든 PE 헤더 필드를 숫자 특징으로 변환하여 사용하였다. Bai[12]는 PE 헤더 필드에서 추출한 197 개의 특성으로 구성된 초기 특징 세트를 구성하고 필터 및 래퍼 기능 선택 방법을 사용하여 특징을 19 개 및 20개로 축약한 특징을 이용하여 결정 트리, 랜덤 포레스트(Random forest), 배깅된(Bagged) 결정 트리(DT) 및 부스트된(Boosted) 결정 트리(DT)를 이용하였다. Belaoued[13]은 PE 옵션 헤더 필드에 저장된 정보를 분석하여 카이 제곱 및 파이 계수를 사용하여 특징을 선택하고 선택된 특징에 대해 로테이션 포레스트(Rotation forest) 분류기를 적용한 PE 악성코드 탐지 시스템을 발표하였다.

Shafiq[8]-[10], Yan[11], Bai[12], Belaoued[13] 등이 파일의 구조적 특징을 이용하여 악성 PE 파일을 탐지하는 방안을 제안하였으나, 대부분의 연구는 특징을 축약하기 위한 다양한 방법을 사용하는 과정과 머신러닝 알고리즘의 특성상 악성코드 분류 과정에

대한 상세한 정보를 보안 분석가에게 제공할 수 없다는 한계가 있거나 실시간 탐지에 부적합한 특징을 가지고 있다.

이러한 문제점을 해결하기 위하여 Anselm[14]는 결정 트리 알고리즘을 사용하였으며, PE(Portable Executable)특징을 추출하여 전처리 기법을 사용하지 않고 원시 특성을 그대로 이용하는 방안을 제안하였다.

본 논문의 저자들은 이전 연구[15]에서 Anselm [14] 연구의 문제점 중의 하나인 결정 트리 규모를 줄이기 위한 방안으로 PE헤더 필드의 구조적 특징 280 개를 구성된 초기 특징 세트로 구성하고, 정보 이득을 계산하여 상위 30%인 84개를 사용함으로써 Anselm[14]이 사용한 특징보다도 적은 수를 사용하여 정탐율 및 오탐율을 각각 1.77%, 2.08% 개선할 수 있는 방안을 제시한 바 있다.

그러나 대다수의 이전 연구들은 윈도우즈 실행 파일인 PE파일의 경우에만 적용할 수 있으며, 리눅스 실행 파일인 ELF에는 적용할 수 없는 단점을 가지고 있다. 또한 리눅스 악성코드 탐지 방안에 대한 연구는 윈도우즈에 비해 활발히 이루어지지 않은 상황이다.

리눅스 악성코드에 관한 기존 연구로는 Bai[16]이 리눅스 심볼 테이블을 분석하여 모든 시스템 콜을 수집하고, ELF 파일을 역 어셈블하여 각 시스템 콜의 빈도를 계산하여 300번 이상 사용된 시스템 콜들을 추출하여 690개의 시스템 콜을 특징으로 선정하여, J48, IBk, 랜덤포레스트를 적용한 결과로서 97.7% 이상의 높은 탐지 율을 제시하였다. 그러나 이러한 접근 방식은 실시간 탐지 및 보안 관리자에게 악성코드 분류 과정에 대한 정보 제공이라는 본 연구의 목표와는 거리가 있다.

Shahzad[17]은 리눅스 악성코드 탐지를 위한 구조적 특징을 이용한 데이터 마이닝 방법을 제안하였는데, ELF파일 형식에서 343개의 특징을 선택한 후, 포렌식 분석 과정을 거쳐 RA(Resistor Average Distance)와 Frequency Histogram Analysis를 통해서 중복성을 배제하여 약 200개 특징을 이용하였다. 다양한 악성코드 유형을 11개로 분류하고 그룹화하여 JRip, J48, PART, 랜덤 포레스트 등을 실험하였으며, 99.8%이상의 탐지 성능을 제시하였다.

본 논문에서는 이전 연구[15]에서 수행하였던 파일의 구조적 정보를 사용하여 PE 악성코드의 실시간 탐지 및 악성코드 분류 과정을 보안 관리자가 이해할 수 있도록 하는 방안을 리눅스 환경의 ELF 파일에 적용해 봄으로써 이러한 접근 방법이 일반화 될 수 있는지를 실험하여 제시하고자 한다.

III. 연구 방법

본 논문에서는 이전 연구에서 윈도우 PE 파일의 구조적 정보를 사용하여 실시간 탐지 및 악성코드 분류 과정을 보안 관리자가 이해할 수 있도록 하는 동일한 접근 방법을 리눅스 ELF 파일에 적용하고자 한다. 따라서 ELF 파일의 원시 특징을 그대로 이용하여 결정 트리 알고리즘을 적용함으로써 보안 분석가에게 악성코드 분류 과정에 대한 통찰을 제공하는 실시간 탐지가 가능한 비 시그니처 기반 악성코드 탐지 방안을 제시하고자 한다.

본 논문에서는 그림 2의 실험 환경과 그림 3의 실험 방안을 설계하여 리눅스 환경에서의 악성코드 및 신종 악성코드 탐지 능력을 평가하고자 한다.

그림 2의 실험 환경에서 먼저 악성 코드 및 정상 파일을 수집하여 검증한 후, 검증된 데이터 샘플로부터 ELF 파일 구조 속성에서 추출 가능한 특징을 추출한다. 추출된 각 특징에 대해 정보 이득을 계산하여 정보 이득이 높은 특징을 선택하여 서브 세트를 정의한다. 서브 세트를 이용하여 결정 트리를 학습 시키고 나온 결과를 10중 교차 검증으로 검증하여 최종 결과를 도출한다.

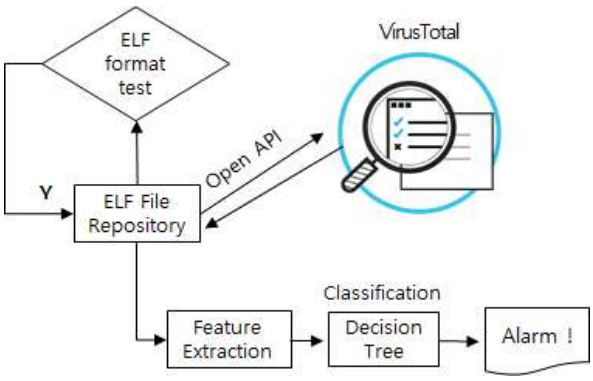

그림 2. 실험 환경
Fig. 2. Experiment environment

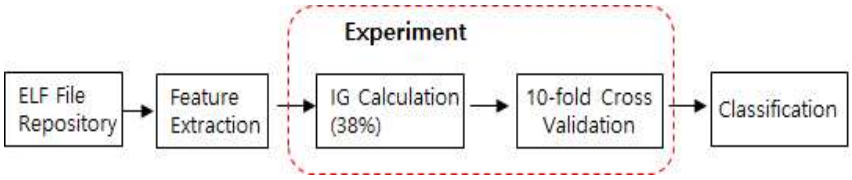

그림 3 실험 아키텍처
Fig. 3. Experiment architecture

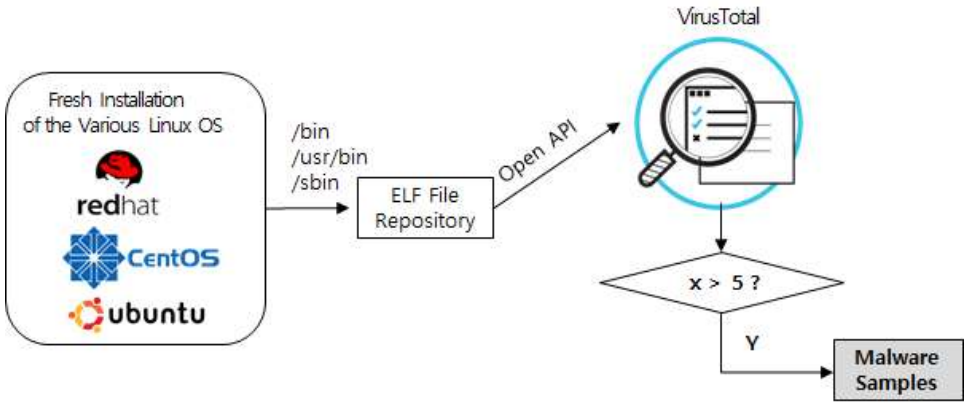

그림 4. 정상 파일 수집 및 검증
Fig. 4. Benign file collection and verification

그림 3 실험에서는 정보 이득 계산 기법에 기반한 특징 선택 및 결정 트리 기반 분류 기법을 실험한다. 본 논문에서는 정보 이득 계산 기법을 적용하여 원시 특징을 상위 38%만을 사용함으로써 이전 연구[15]와 유사한 접근 방법을 리눅스에서도 동일하게 적용하고자 한다.

3.1 정상 파일 수집 및 검증

본 논문에서는 그림 4와 같이 정상 ELF 파일을 수집하여 실험에 사용된 데이터 세트의 개요를 제시한다. ELF 파일 (악성 및 양성 모두)는 20KB에서 4MB 크기의 파일을 이용한다. 정상 ELF 파일은 다양한 리눅스 운영 체제를 설치한 후, / bin, / sbin 및 / usr / bin 디렉토리에서 수집한다. 이러한 데이터 세트 파일은 바이러스토탈(Virustotal)의 오픈 API를 사용하여 검증하였는데, 검증 기준은 전체 약 69개의 안티바이러스 엔진 중에서 1개라도 악성으로 판정되는 경우에는 해당 파일의 무결성을 확보하기 위해 배제하였다. 바이러스토탈은 구글의 자회사로 편입된 회사이며, 약 69개 안티바이러스 회사들과 제휴하여 악성코드 정보를 공유하고 있으며, 특히 전 세계 일반 사용자들이 업 로드한 악성코드, URL 및 PCAP등을 약 69개 안티바이러스 회사들의 엔진으로 검사하고 해당 결과를 제공하는 클라우드 서비스이다[18]. 바이러스토탈에서는 동일한 과정을 자동화된 서비스로 제공하기 위하여 API를 제공하는 데, 오픈 API와 유료 API로 구분된다. 바이러스토탈 사이트는 사용자가 수동 또는 오픈 API를 통해 업로드 한 악성 파일에 대하여 여러 개의 백신 엔진으로 검사한 결과를 투명하게 제공한다.

3.2 악성 파일 수집 및 검증

양질의 최신 악성코드를 수집하는 가장 좋은 방법은 바이러스토탈을 이용하는 것이지만, 현실적으로 유료 API를 구매하여야 한다는 문제점이 있다. 대안으로는 VirusShare에서 최신 악성코드를 수집하는 것이 가능하다. 그러나 레이블이 되어 있지 않기 때문에 해당 악성코드를 그대로 이용할 수 없다는 문제점이 있다[19].

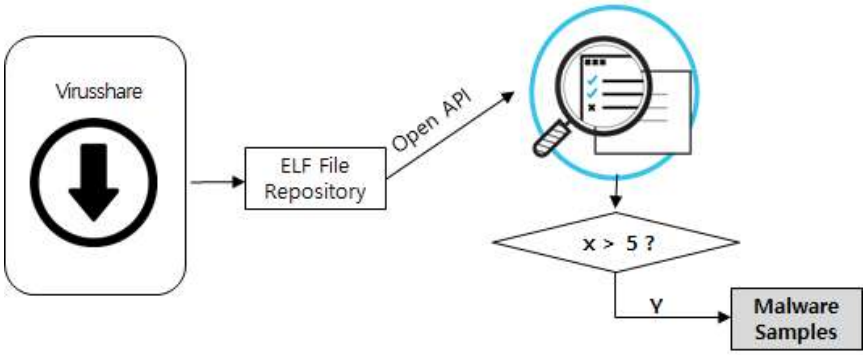

그림 5. 악성파일 수집 및 검증
Fig. 5. Malware collection and verification

본 논문에서는 이러한 문제점들을 해결하기 위하여 그림 5와 같이 악성코드 샘플들을 VirusShare로부터 수집하고 이들을 바이러스토탈의 오픈API를 이용하여 레이블 하는 방법을 사용하였다. 검증 기준은 전체 약 69개의 안티바이러스 엔진 중에서 5개 초과 AV엔진이 악성으로 판정되는 경우에는 해당 파일을 악성 파일로 판단하여 선택하였다.

표 1. 악성 및 정상 데이터 샘플의 비교
Table 1. Comparison of benign file and malware samples

Samples	ELF-miner	Our research
Benign ELF samples	734	2,180
Malicious ELF samples	709 (Collected from VxHeaven & offensive computing)	6,950 (Collected from VirusShare and verified by Virustotal with open API)
Total	1,443	9,130

표 1에서 기존 연구[17]과 본 논문에서 사용되는 악성 및 정상 ELF 파일을 비교하였다. 그림 4와 같은 과정을 통해서 수집한 정상파일 데이터 세트 및 그림 5와 같은 과정을 통하여 수집한 악성 파일 데이터 세트와 이전의 유사 연구[17]에서 사용한 데이터 세트를 비교하여 나타내고 있으며, 이전의 유사 연구에 비해서 다소 많은 양의 데이터 세트를 사용하고 있음을 알 수 있다.

3.3 ELF 파일의 특징 추출

본 논문에서는 ELF 파일에서 추출 가능한 모든 특징을 추출하고, 이 특징을 기반으로 정보 이득을 계산하여 상위 38%를 선택하여 사용함으로써 결정 트리의 규모를 줄이는 동시에 실시간으로 보안 분석가에게 악성코드 분류 과정에 대한 상세한 정보를 제공할 수 있는 방안을 제시하고자 한다.

본 논문에서 사용한 원시 특징은 표 2와 같다. ELF 헤더는 ELF 파일의 구성을 설명하는 데이터 구조로 구성된다. 헤더 구조는 ELF 파일을 파싱하는데 도움이 되는 정보를 포함한다. ELF 헤더 구조는 ID(매직 바이트), 머신 유형, ELF 파일 버전, 바이트 단위 프로그램 헤더 테이블 오프셋, 바이트 단위 섹션 헤더 테이블 오프셋, 프로세서 관련 플래그, ELF 헤더 크기, 프로그램 헤더 테이블의 엔트리 개별 항목 크기 및 수, 섹션 헤더 크기 (바이트) 및 프로그램 헤더 크기, 섹션 헤더 테이블의 엔트리 크기 및 수 등의 여러 가지 일반 필드로 구성된다. 이러한 필드 중에서 프로그램과 섹션 헤더 오프셋을 제외한 ELF 헤더의 16개 필드를 사용한다.

표 2. ELF 원시 특징 선택
Table 2. Selection of original ELF structural features

Feature selection sources	Selected raw features for our research
ELF header	16
Section headers	238
Program headers	40
Symbols section	17
Dynamic section	27
Dynamic symbol section	17
Relocation sections	26
Global offset table	1
Hash table	1
Total	383

섹션 헤더 구조는 섹션 이름, 섹션 유형, 기타 속성을 나타내는 섹션 플래그 필드, 섹션의 첫 번째 바이트 주소 필드, ELF 파일의 시작점부터 섹션의 첫 번째 바이트 섹션 오프셋, 바이트 단위의 섹션 크기, 섹션 헤더 테이블 인덱스 링크, 섹션 유형에 따라 달라지는 섹션 정보 필드, 조정 제약 조건을 나타내는 섹션 주소 조정 필드를 포함한다. 섹션 이름, 주소 및 오프셋 필드를 제외한 34개 섹션 헤더 구조 중의 238개 필드를 사용한다.

프로그램 헤더는 모든 실행 가능 및 링크 가능 파일은 프로그램 세그먼트 및 프로그램 실행에 필수적인 추가 정보를 나타내는 구조의 배열이다. 각 실행 파일 세그먼트는 하나 이상의 섹션이 포함될 수 있다. 세그먼트 헤더 구조는 세그먼트 유형, 파일 시작 부분으로부터 오프셋인 세그먼트 오프셋, 메모리에 첫 번째 바이트가 존재하는 가상 주소, 세그먼트의 물리적 주소, 세그먼트 파일 이미지의 바이트 수를 제공하는 파일 사이즈 필드, 메모리 이미지 내의 바이트 수를 나타내는 메모리 크기 필드, 세그먼트의 플래그 관련 정보를 나타내는 플래그의 필드, 하고, 파일 및 메모리의 세그먼트의 조정 정보를 포함하는 p_align 멤버로 구성된다. 이 중에서 첫 8개 세그먼트의 모든 주소 및 오프셋 필드를 제외하고 40개의 특징을 이용한다.

심볼 섹션은 ELF 파일내의 심볼과 관련된 정보를 포함하지만 모든 유형의 객체 파일에서 필수 사항은 아니다. 심볼 테이블 구조에는 이름, 값, 크기, 정보, 기타 및 섹션 헤더 인덱스와 같은 5개의 주요 필드가 있지만 항목 수는 각 실행 가능 파일의 심볼 테이블마다 다르다. 따라서 st_info 필드를 기반으로 카테고리를 생성하며 심볼의 바인딩과 속성에 대한 정보를 포함한다. 결과 카테고리는 [심볼, 지역 심볼, 전역 심볼, 약한 심볼 및 stb_lo_proc 심볼]의 전체 수이다. 객체와 함수 심볼은 그 범위, 즉 [로컬 객체, 전역 객체, 약한 객체, 로컬 함수, 전역 함수, 약한 함수, 섹션, 파일, stt_lo_proc 및 stt_hi_proc]의 전체 수를 기준으로 추가로 카테고리화 된다. 결과적으로 심볼 테이블에서 17개의 범주 필드를 특징으로 이용한다.

동적 섹션은 ELF 파일에 동적 링크 정보가 있으면 PT DYNAMIC (동적 섹션의 중요한 부분)이 프로그램의 헤더 테이블에 포함된다. 동적 섹션의 구조는 4 바이트의 필드 d_tag 와 union d_un을 포함한다. 모든 필드는 매개 변수 d_tag에서 가용한 정보에 따라 분류된다. 심볼 테이블과 마찬가지로 동적 섹션에도 고정 된 수의 항목이 포함되지 않을 수 있다. 따라서 모든 항목은 27개의 범주로 분류되는 데, 이것을 특징으로 이용한다.

동적 심볼 섹션(DSS, Dynamic Symbols Section) 객체 파일에 동적으로 링크된 객체가 포함된 경우, 동적 심볼 섹션도 포함될 수 있다. 동적 심볼 섹션의 세부 사항은 앞에서 언급한 심볼 섹션의 세부 사항과 유사하다. 17개의 특징을 DSS 섹션에서 추출하여 이용한다.

재배치 섹션(Relocation section)은 ELF에서 재배치는 참조된 심볼과 그 자체의 정의와 연결하는 과정이다. 프로그램이 함수를 호출 할 때 통제 또한 적절한 주소에 있는 함수 정의로 통제가 이관 되어야 한다. rel의 구조에는 offset과 info의 두 멤버가 포함된다. 오프셋 필드는 재배치 행위가 수행될 위치를 나타낸다. 파일이 실행파일 또는 공유 객체인 경우, 오프셋은 저장 단위의 가상 주소를 나타낸다. info 필드는 재배치 유형 및 재배치에 사용되는 심볼 테이블 인덱스에 대한 정보를 유지한다. 정상 파일과 악성 파일을 구분하기 위해 섹션. rel과 .rela의 내용에서 26개의 특징을 추출하여 이용한다.

글로벌 오프셋 테이블(GOT, Global Offset Table)에는 사설 데이터를 나타내는 객체의 절대 주소를 포함한다. 프로그램 텍스트의 위치 독립성 및 공유 기능에 영향을 주지 않는 주소를 제공한다. 프로그램은 위치 독립적 주소 지정을 사용하는 절대 주소 값을 위해 GOT를 참조한다. 특징으로 GOT의 크기를 나타내는 필드 1개를 사용한다.

해시 테이블(Hash table)은 심볼 테이블에 대한 액세스를 용이하게 하는 32 비트 객체 테이블이다. 해시 테이블 크기 1개 필드를 특징으로 이용한다.

본 논문에서는 위에서 선택한 전체 ELF파일의 구조적 데이터 특징 수를 줄이기 위하여 정보 이론적인 측정 - 정보 이득-을 사용하여 데이터 세트의 특징을 줄임으로써 결과적으로 결정 트리의 크기를

축소하고자 한다. 정보 이득은 속성 값이 알려진 경우 불확실성 감소에 대해 측정한다[20][21]. 속성 X와 속성 Y의 정상, 악성에 대한 불확실성은 각각의 엔트로피 H (X)와 H (Y)에 의해 부여된다. 그러면 Y에 대한 X의 정보 이득은 Information Gain (Y; X)에 의해 부여된다.

$$\text{Information Gain } (Y; X) = H(Y) - H(Y \mid X) \quad (1)$$

높은 정보 이득 가치는 높은 분류 잠재력을 나타내고 그 반대도 마찬가지이다. 정보 이득은 1에서 0까지 다양하다. 정보 이득을 계산하여 상위 약 38%인 147개의 특징 세트를 선택하였다.

표 3. 정보 이득 계산 후 선택한 ELF 특징
Table 3. Selected ELF features for this research using information fain calculation

Feature selection sources	No. of raw features	Selected features by calculating Information Gain (Top 38%)	No. of top 38% features
ELF header	16	Identification, MachineType, ProgramHeaderOffset, SectionHeaderOffset, Flags, HeaderSize, SizeProgramHeader, EntriesProgram, SizeSectionHeader EntriesSection, StringTableIndex	11
Section headers	238	.text_type .text_size .text_alignment .bss_type .bss_size .bss_alignment .comment_size .comment_entsize .data_type .data_size .data_alignment .dynamic_type .dynamic_size .dynamic_entsize .dynamic_table_index_link .dynamic_alignment .dynstr_type .dynstr_size .dynsym_type .dynsym_size .dynsym_entsize .dynsym_table_index_link .dynsym_info .dynsym_alignment .fini_type .fini_size .fini_alignment .hash_type .hash_table_index_link .hash_alignment .gnu.hash_type .gnu.hash_size .gnu.hash_entsize .gnu.hash_table_index_link .gnu.hash_alignment .init_type .init_size .init_alignment .got_type .got_flags .got_size .got_entsize .got_alignment .interp_type .interp_size .plt_type .plt_flags .plt_size .plt_entsize .plt_alignment .rodata_type .rodata_flags .rodata_size .rodata_entsize .rodata_alignment .shstrtab_size .strtab_size .symtab_size .symtab_entsize .symtab_table_index_link .symtab_info .symtab_alignment .sbss_type .sbss_flags .rel.dyn_type .rel.plt_type .rel.plt_flags .got.plt_type .got.plt_size .got.plt_entsize .got.plt_alignment	71
Program headers	40		0
Symbols section	17	STB_LOCAL STT_OBJECT_STB_GLOBAL STT_OBJECT_STB_WEAK STT_NOTYPE_STB_WEAK STB_WEAK STT_NOTYPE_STB_LOCAL STT_FUNC STT_FUNC_STB_GLOBAL STT_NOTYPE_STB_GLOBAL STB_GLOBAL STT_NOTYPE STT_OBJECT STT_FUNC_STB_WEAK symbol_tab	14
Dynamic section & dynamic symbol section	44	dynamic_s_c s_STT_FUNC_STB_LOCAL s_STT_FUNC_STB_WEAK s_STT_FUNC s_STB_LOCAL s_STT_FUNC_STB_GLOBAL s_STB_GLOBAL s_STB_WEAK s_STT_OBJECT_STB_WEAK s_STT_OBJECT s_STT_OBJECT_STB_LOCAL s_STT_NOTYPE s_STT_NOTYPE_STB_GLOBAL s_STT_NOTYPE_STB_WEAK s_STT_NOTYPE_STB_LOCAL s_STT_OBJECT_STB_GLOBAL DYNRELAENT DYNFINI DYNPLTRELSZ DYNVERNEEDNUM DYNINIT_ARRAY DYNSTRSZ DYNSTRTAB DYNPLTREL DYNRELENT DYN DYNJMPREL DYNSYMTAB DYNRPATH DYNFINI_ARRAYSZ DYNNEEDED DYNSYMENT DYNCOUNT DYNINIT DYNRELSZ DYNINIT_ARRAYSZ DYNVERNEED DYNRELASZ DYNREL DYNRELA DYNPLTGOT DYNHASH DYNDEBUG DYNFINI_ARRAY DYNNULL	45
Relocation sections	26	R_386_GLOB_DAT R_386_JUMP_SLOT R_386_COPY	3
Global offset table	1	GOT_SIZE	1
Hash table	1	HASH_SIZE	1
Total	**383**		**146**

3.4 리눅스 악성코드 분류기(Classifier)

파일 샘플을 대상으로 특징 추출 프로그램을 실행하면 각 파일에 대한 원시 특징 목록이 생성된다. 원시 특징 목록은 일반적으로 정확성을 향상시키기 위해 사전 처리한다. 그러나 사전 처리하여 선택된 특징은 보안 분석가에게는 어떤 특징이 악성코드 분류에 이용하였는지 파악하기 어려운 단점이 있다. 그 이유는 원시 특징을 사전 처리하는 경우, 원시 특징 속성을 조합한 형태의 합성 속성이 생성되거나 속성이 다른 값으로 변화되어 보안 분석가에게 분류과정에서 어떤 특징이 어떻게 분류에 이용되는지에 대한 정보를 제공할 수 없기 때문이다. 머신러닝 알고리즘은 분류과정에서 어떤 속성을 어떻게 이용하였는지 표출되지 않는 경우가 대부분이다. 이러한 문제가 존재하지 않는 거의 유일한 분류 알고리즘은 결정 트리 알고리즘이다.

따라서 본 논문에서도 결정 트리 알고리즘을 사용한 후, 10 중 교차 검증 기법을 사용하여 결과를 검증하고자 한다[22]. 10 중 교차 검증 기법은 데이터 세트 내에 총 N개의 샘플이 존재하는 경우, 샘플을 10개 집단으로 나누고, 이때, 각 집단의 평균이 비슷한 정도로 나눈다. 분류기를 10-1 (9)개의 집단으로 학습을 시키고 나머지 1개의 집단으로 테스트하여 분류기의 성능을 측정한다. 이 과정을 서로 겹치지 않도록 10번 수행한 후, 10번의 정확도를 평균하여 그 값을 분류기의 성능으로 정의할 수 있는데, 장점은 보유하고 있는 데이터 샘플의 대부분을 학습에 재사용할 수 있다는 점이다.

본 논문에서는 결정 트리 알고리즘과 랜덤 포레스트를 분류기로 사용하여 실험을 수행하였다. 결정 트리 분석은 의사결정규칙(Decision rule)을 도표화 하여 관심의 대상이 되는 집단을 몇 개의 소집단으로 분류 또는 예측하는 분석방법으로서 보안 분석가들에게 분류 과정에 대한 직관을 제공할 수 있다. 그 이유는 분석하고자 하는 자료의 여러 변수들 중 각 변수들 사이의 연관성 정도에 따라 중요 변수를 선별한 후 선별된 변수를 통하여 트리 구조를 생성하기 때문에 다른 분석 방법들에 비해 분류 과정을 쉽게 이해하고 설명할 수 있는 장점이 있기 때문이다. 랜덤 포레스트는 결정 트리의 앙상블 학습 방법의 일종으로, 훈련 과정에서 구성한 다수의 결정 트리로부터 분류 또는 평균 예측치를 출력함으로써 동작하는 데, 여러 개의 의사결정트리를 만들고, 투표하여 다수결로 결과를 결정하는 배깅(Begging) 과정 때문에 보안 분석가들에게 분류 과정에 대한 직관을 제공할 수는 없다.

머신러닝 계열에서 널리 사용하는 결정 트리(DT) 알고리즘은 C4.5, C5.0 등이며, J48은 C4.5와 동일한 결정 트리 알고리즘이다. C5.0은 C4.5 (J48)를 개선하여 개발되었으나, 라이선스를 지불하여야 사용할 수 있으므로, C4.5 (J48)가 널리 이용되고 있으며, 본 논문에서도 J48을 결정 트리 기반 데이터 분류 알고리즘으로 사용하였다.

IV. 실험 결과 및 분석

4.1 분류 속도

본 논문에서 사용한 특징 추출 및 분류 기법의 경우에는 파일 당 약 평균 17 밀리 초 이내의 시간이 소요되어 실시간 악성코드 탐지에 적용할 수 있음을 확인하였다. 샘플당 처리 시간 비교는 표 4와 같다.

표 4 샘플 당 처리 시간 비교
Table 4. Scan time per sample

	Average classification time per sample
Shahzad[17]	16 ~ 17 milliseconds.
Our research	17 milliseconds.

4.2 결과 분석

Shahzad[17]의 실험결과 중에서 J48 분류기를 사용했을 때의 결과는 Resistor Average(RA) divergence 및 Frequency Histogram analysis를 이용하여 불필요한 특징을 제거하는 기법을 사용하였으나 본 논문에서는 정보 이득을 적용하여 상위 38%를 선택하여 실험하였다. 두 연구의 악성코드 분류 결과는 표 5에 비교되어 있다.

Shahzad[17]의 경우, 악성코드를 11가지 유형별로

분류하고 이를 그룹으로 묶어 다양한 실험을 하였으나, 본 논문에서는 악성코드 수집 분량의 한계로 별도로 분류하지는 않았다.

표 5. 악성 코드 탐지 성능 비교
Table 5. Malware classification performance

	ML algorithm used	Feature selection method(s)	No. of selected features	Accuracy (%)
Shahzad [17]	J48	Resistor Average (RA) divergence & frequency histogram analysis	200	99.98
	Random fores			99.80
Our research	J48	Information gain	146	99.68
	Random forest			99.59

그러나 표 5에서 알 수 있듯이 본 논문 결과도 J48을 사용했을 경우에 99.68%의 높은 탐지 정확성을 제시하고 있다. 미세한 정확성의 차이는 세 가지 원인에서 기인되었을 것으로 추정된다. 첫째로 본 논문에서는 샘플을 9,130개를 사용한 반면, Shahzad [17]는 1,443개의 샘플을 사용하였는데, 이는 본 연구에서 사용한 샘플의 수가 약 6배 이상으로서 샘플의 크기가 클수록 탐지 모델이 일반화되는 경향을 의심해 볼 수 있다. 두 번째로는 RA divergence 및 Frequency Histogram analysis의 두 가지 포렌식 분석을 통하여 불필요한 특징을 제거한 후에 정보 이득을 계산하여 특징을 선택한 반면, 본 논문에서는 정보 이득만을 계산하여 상위 38%의 특징을 선택하였기 때문에 미세한 차이가 발생할 수 있을 것으로 추정된다. 마지막으로 기존 연구[17]에서는 약 200개의 특징을 이용한 반면, 본 논문에서는 146개의 특징을 이용하였는데, 이전 연구[15]에서 정보 이득을 계산하여 상위 30%를 선택하였을 때, 상위 10% 또는 20%를 선택하여 사용하였을 경우보다 약간의 정확성이 개선되는 것을 확인하였고 바로 이러한 부분에서 미세한 차이가 발생할 수 있을 것으로 추정된다. 표 5에서 보이는 바와 같이 본 논문의 분류기에 따른 탐지 성능을 비교해 보면 J48의 실험 결과가 랜덤 포레스트 실험 결과 보다 약간 높은 것으로 나타났는데, 이는 Shahzad[17]의 연구와 동일한 현상임을 알 수 있다.

V. 결 론

이전 연구[15]에서는 윈도우 PE 파일의 구조적 특징을 기반으로 높은 정확성을 가지는 실시간 윈도우 악성코드 탐지 방안을 제시하였다. 본 논문에서는 이전 연구와 동일한 탐지 방안을 리눅스 악성코드에 적용하여 실시간 리눅스 악성 코드 탐지 방안을 제시하였다. 즉, 리눅스 ELF 헤더의 여러 섹션에서 추출한 구조적 특징을 이용하여 실시간 앞어 코드 탐지가 가능하였고 보안 분석가들이 악성코드 분류 과정을 이해 할 수 있도록 하였다. 본 논문의 실험결과, 이전 연구[15]에서 수행하였던 윈도우즈 파일의 구조적 정보를 사용하여 실시간 탐지 및 악성코드 분류 과정을 보안 관리자가 이해할 수 있도록 하는 동일한 접근 방법이 리눅스와 같은 다른 운영 체제의 실행 파일에도 적용하여 일반화 될 수 있음을 확인할 수 있었다, 향후 동일한 방안을 Mac OS X, android 등 다른 운영 체제의 실행 파일에 적용하는 연구가 계속 수행되어야 할 것으로 사료된다.

References

[1] Christian Daudt, "The Shift to Linux Operating Systems for IoT", https://www.iotforall.com/linux-operating-system-iot-devices/2018. [accessed : Apr. 02, 2019]

[2] ECLIPSE, "KEY TRENDS FROM THE IOT DEVELOPER SURVEY 2018", https://blog.benjamin-cabe.com/2018/04/17/key-trends-iot-developer-survey-2018 2018. [accessed : Apr. 03, 2019]

[3] "Interview with Google's Chris DiBona", https://lwn.net/Articles/253718/2007. [accessed : Apr. 02, 2019]

[4] STEPHEN SHANKLAND, "Q&A: Google's open-source balancing act", https://www. cnet.com/news/q-a-googles-open-source-balancing-act/2008. [accessed : Apr. 12, 2019]

[5] crouchtech, "Top Linux Distributions for Cloud Computing" https://crouchtech.io/top-linux-distributions-

for-cloud-computing 2019. [accessed : Apr. 02, 2019]

[6] SYSTEM V APPLICATION BINARY INTERFACE, http://www.sco.com/developers/devspecs/gabi41.pdf. [accessed : Apr. 02, 2019]

[7] Executable and Linkable Format (ELF), http://www.cs.cmu.edu/afs/cs/academic/class/15213-s01/s00/doc/elf.pdf. [accessed : Apr. 02, 2019]

[8] M. Shafiq, S. Tabish, F. Mirza, and M. Farooq, "PE-Miner: Mining structural information to detect malicious executables in realtime", Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection (RAID '09), Berlin, Heidelberg. Springer-Verlag, pp. 121-141, Sep. 2009.

[9] M. Shafiq., S. Tabish, and M. Farooq, "PE-Probe: Mining Structural Information to Detect Malicious Executables in Realtime", Proceedings of Virus Bulletin Conference, Oct. 2009 (VB2009).

[10] M. Shafiq, S. Tabish, F. Mirza, and M. Farooq, "A framework for efficient mining of structural information to detect zero-day malicious portable executables", Technical Report TR-nexGINRC - 2009-21, Next Generation Intelligent Networks Research Center, Islamabad, Pakistan. Jan. 2009.

[11] G. Yan, N. Brown, and D. Kong, "Exploring discriminatory features for automated malware classification", International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, DIMVA, LNCS 7967, pp. 41-61. Jul. 2013.

[12] J. Bai, J. Wang, and G. Zou, "A Malware Detection Scheme Based on Mining Format Information", The Scientific World Journal, Vol. 2014, Article ID 260905, pp. 1-11, Jun. 2014.

[13] Belaoued and Mazouzi., "A Real-Time PE-Malware Detection System Based on CHI-Square Test and PE-File Features", International Conference on Computer Science and its Applications, IFIPAICT, Vol. 456, pp. 416-425, May 2015.

[14] T. Anselm and S. Arran, "Human-Readable Real-Time Classifications of Malicious Executables", Australian Information Security Management Conference, pp. 9-18, Dec. 2012.

[15] Deok-Jo Jeon and Dong-Gue Park, "Real-time Malware Detection Method Using Machine Learning", Korean Institute of Information Technology, Vol. 16, No. 3, pp. 101-113, Mar. 2018.

[16] J. Bai, Y. Yang, S. Mu, and Y. Ma. "Malware Detection Through Mining Symbol Table of Linux Executables", Information Technology Journal, Vol. 12, No. 2, pp. 380-384, Feb. 2013.

[17] M. Shahzad and M. Farooq, "ELF-Miner: using structural knowledge and data mining methods to detect new (Linux) malicious executables", Knowledge and Information Systems, Vol. 30, No. 3, pp. 589-612, Mar. 2012.

[18] https://www.virustotal.com/gui/home/upload. [accessed : Jan. 01, 2018]

[19] https://virusshare.com/ [accessed : Jan. 01, 2018]

[20] Richong Zhang and Thomas Tran, "An information gain-based approach for recommending useful product reviews", Knowledge and Information Systems, Vol. 26, No. 3, pp. 419-434, Mar. 2011.

[21] T. M. Cover and J. A. Thomas, "Elements of information theory", John Wiley & Sons, 2006.

[22] J. Han and M. Kamber, "Data Mining: Concepts and Techniques", Morgan Kaufmann, 2nd edition. 2006.

저자소개

전 덕 조 (Deok-Jo Jeon)

1989년 2월 : 아주대학교
전자계산학과(공학사)
1995년 5월 : 뉴맥시코대학교
전자계산학과(이학석사)
2005년 10월 ~ 현재 :
(주)씨큐비스타 대표이사
관심분야 : 지능형 위협 대응, 지능형 보안관제, 네트워크 보안

박 동 규 (Dong-Gue Park)

1992년 2월 : 한양대학교
전자공학과(공학박사)
1992년 3월 ~ 현재 : 순천향대학교
정보통신공학과 교수
관심분야 : 제어 시스템 보안, 네트워크보안, 시스템 보안, 모바일 보안