Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.29, NO.6, Dec. 2019 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2019.29.6.1327

정적 분석과 앙상블 기반의 리눅스 악성코드 분류 연구*

황 준 호,† 이 태 진‡
호서대학교

Study of Static Analysis and Ensemble-Based Linux Malware Classification*

Jun-ho Hwang,† Tae-jin Lee‡
Hoseo University

요 약

IoT 시장의 성장과 더불어 linux 아키텍처를 사용하는 디바이스들에 대해 악성코드 보안 위협이 꾸준히 증가하고 있다. 하지만, Mirai 등의 심각한 보안피해를 야기한 주요 악성코드들을 제외하면 linux 악성코드에 대한 보안 커뮤니티의 관련 기술이나 연구는 전무한 수준이다. 또한, IoT 환경의 디바이스, 벤더, 아키텍처 등의 다양성이 더욱 심화됨에 따라 linux 악성코드 대응 난이도 또한 심화되고 있다. 따라서, 본 논문에서는 linux 아키텍처의 주요 포맷인 ELF를 분석하고 이를 기반으로 한 분석 시스템과, IoT 환경을 고려한 바이너리 기반의 분석 시스템을 제안한다. ELF 기반의 분석 시스템은 상대적으로 고속으로 다수의 악성코드에 대해 전처리 분류 할 수 있으며 상대적으로 저속의 바이너리 기반의 분석 시스템은 전처리 하지 못한 데이터에 대해 모두 분류 가능하다. 이러한 두 개의 프로세스는 서로 상호보완되어 효과적으로 linux 기반의 악성코드를 분류할 수 있을 것이라 기대한다.

ABSTRACT

With the growth of the IoT market, malware security threats are steadily increasing for devices that use the linux architecture. However, except for the major malware causing serious security damage such as Mirai, there is no related technology or research of security community about linux malware. In addition, the diversity of devices, vendors, and architectures in the IoT environment is further intensifying, and the difficulty in handling linux malware is also increasing. Therefore, in this paper, we propose an analysis system based on ELF which is the main format of linux architecture, and a binary based analysis system considering IoT environment. The ELF-based analysis system can be pre-classified for a large number of malicious codes at a relatively high speed and a relatively low-speed binary-based analysis system can classify all the data that are not preprocessed. These two processes are supposed to complement each other and effectively classify linux-based malware.

Keywords: Linux Malware, Machine Learning, Static Analysis

I. 서 론

지난 몇 년 동안 linux 아키텍처를 사용하는 디바이스들에 대한 악성코드 보안 위협이 꾸준히 증가하고 있다. 전통적으로 악성코드는 다양한 아키텍처에서 존재하며 침해사고를 유발시키는 주요 보안 위협원으로 여겨졌지만, 기존 PC 환경의 대부분을 차지하던 windows 운영체제를 주 타겟으로 한 악성코드들이 대부분이였다. 하지만 근래에 들어서 다양한 아키텍처를 대상으로 한 악성코드 공격들이 보고

Received(07. 15. 2019), Modified(09. 10. 2019), Accepted(10. 23. 2019)
* 본 연구는 과학기술정보통신부 및 정보통신기획평가원의 대학ICT연구센터지원사업의 연구결과로 수행되었음 (IITP-2019-2016-0-00304)
† 주저자, hwangso93@gmail.com
‡ 교신저자, kinjecs0@gmail.com(Corresponding author)

가 되었다. Mac OS의 경우에는 2016년도에 370%의 악성코드 증가율을 나타내었고[1], 모바일 악성코드의 경우에도 200%의 악성코드 증가율을 나타내었다[2]. 특히, 이러한 동향에서 모바일 디바이스나 AI 스피커, 자율주행 어시스턴트 등의 IoT 디바이스 시장의 폭발적인 성장과 더불어 linux 운영체제와 임베디드 linux 악성코드에 대응하는 보안 기술 및 연구의 필요성이 대두되고 있다.

반면, 최근들어 여러 보안 커뮤니티가 linux 악성코드에 관심을 나타내고 있지만 관련 기술 및 연구는 거의 전무한 실정이다. 2016년, Mirai Botnet으로 인해 대규모의 DDoS(Distributed Denial of Service) 공격이 발생하였고 북미권의 인터넷이 마비되는 등의 막대한 규모의 피해가 발생하였다[3]. 이는 보안 커뮤니티의 이목을 끌기에 충분했지만 Mirai Botnet의 공격 방식이 단순하였음에도 불구하고 현재까지도 많은 디바이스에서는 적합한 보안 패치가 이루어지지 않고 있으며 유사한 공격 기법에 무방비로 노출된 상황이다.

따라서, 본 논문에서는 linux 악성코드에 대응하기 위하여 linux ELF(Executable and Linking Format)에 존재하는 여러 feature를 밝히고 해당 feature를 머신러닝 알고리즘을 통해 학습 및 분석하는 ELF feature based Analysis System을 평가한 결과를 제시한다. 특히, 별도의 파일/펌웨어 포맷이 존재하지 않거나 ELF를 따르지 않는 여러 linux 아키텍처 환경을 가정하여 바이너리 데이터만을 이용하여 feature를 생성하고 이미지를 생성한 후, 이를 학습 및 분석하는 CNN (Convolutional Neural Network) 기반의 분석 시스템도 제시한다. 해당 시스템은 임베디드 linux 아키텍처를 사용하고 있는 대부분의 IoT 기기를 감안하면 IoT 악성코드 대응에도 활용될 수 있을 것이라 기대한다. 본 논문에서는 linux 악성코드 분석에 활용될 수 있는 여러 feature를 밝히고 해당 feature vector를 다수의 머신러닝 알고리즘으로 평가하고, IoT 환경을 고려한 바이너리 기반의 분석 시스템 또한 평가 및 제시함으로써 향후 linux 악성코드 관련 기반 연구로 활용될 수 있을 것이라 기대한다.

다음으로, 2장에서는 linux 악성코드에 대응하기 위한 보안 커뮤니티들의 관련 연구들을 제시한다. 3장에서는 ELF 파일에서의 feature 후보군을 밝히고, 본 논문에서 가용한 feature vector 및 분석 시스템을 제안한다. 4장에서는 분석 시스템을 평가한 시험 결과를 제시하며 5장에서는 정적 분석과 앙상블 기반의 리눅스 악성코드 분류 연구의 주요 결론을 제시한다.

II. 관련 연구

2.1 Malware Analysis based on Machine Learning

지능화된 악성코드들에 대응하기 위해 지속적이고 꾸준한 연구가 진행되어 왔다. 악성코드의 주요 권한들을 분석하고 권한 행렬을 구축하여 이와 머신러닝 기반의 분석 시스템으로 악성코드를 식별하는 연구[4], 파일 바이트 embedding과 머신러닝 기반의 분석 시스템으로 악성코드를 식별하는 연구[5], 새로운 악성코드들에 대해 기존의 feature들이 제대로 인식되지 않는 문제에 대응하기 위해 모델 재훈련 없이 악성코드를 온전히 탐지하는 기법[6], 악성코드의 API call과 같은 다양한 동적 기능을 분석하고 악성코드 분류 시 각 기능들의 기여도를 측정하는 연구[7], 바이트 n-gram과 Elastic-Net으로 정규화된 학습 모델을 통해 악성코드를 분류하는 연구[8], ARM 기반의 IoT 어플리케이션의 opcode를 분석하고 RNN(Recurrent Neural Network)을 통해 악성코드를 탐지하는 연구[9] 등이 있다. 특히, 이러한 최근 동향에서 공통적으로 관찰되는 주된 분류 방법론으로는 머신러닝 기반의 분석 시스템이다. 이러한 경향은 악성코드들이 zero-day 취약점을 기반으로 침투하면 사전대응이 어렵고, 또한 이러한 악성코드들이 대량으로 유포되기 때문에 전문가가 수동 분석하기 어려운 현실에서 기인한다. 따라서, 현재 악성코드 보안의 주요 과제인 신규 악성코드 대응, 대량의 악성코드 대응을 위해서 관련연구들이 머신러닝 방법론을 지향하는 것은 당연하다. 하지만 전문가를 통한 시그니처 분석 기법 등에 비해 비교적 오탐이 높고 아키텍처나 파일 포맷 등에서 도출되는 구조적 특징에 기반한 시스템의 경우에는 현재의 IoT 환경을 고려하면 폭넓게 활용될 수 없다는 문제점도 존재한다.

2.2 The Challenge of linux Malware Analysis

악성코드 관련 연구는 현재까지 windows 운영체

제 계열에서만 주로 진행되어 linux 악성코드 대응을 위한 관련 연구는 거의 없는 실정이다. 특히나 linux 운영체제는 수십개가 넘는 다양한 아키텍처가 존재하기 때문에 연구에 어려운 점이 있다. 이러한 linux 악성코드 분석 연구 현황에 대해 밝힌 주요 연구로는 Cozzi. E, et al. 의 연구[10]가 있다. 해당 연구에서는 linux 악성코드 연구에서 발생할 수 있는 주요 난제들과 분석 시스템 및 자체적으로 구축한 1만여개의 데이터 셋의 샘플들의 주요 동작 과정을 밝혔다. linux 악성코드 연구에서 발생할 수 있는 주요 난제들은 Table 1.과 같다.

특히, Table 1.과 같이 분석한 linux 악성코드 연구에 대한 챌린지 중, 주요하게 고려해야 하는 난제는 다음과 같다. 첫 번째로, 아키텍처의 다양성으로 인한 난제이다. linux 시스템은 CPU, OS, 라이브러리 등에 따라 동작하는 방식이 상이하며 특히 임베디드 linux 시스템의 경우에는 디바이스 종류, 벤더, 소프트웨어 의존성 등 고려해야할 사항이 더욱 복잡하기 때문에 이러한 시스템을 타겟으로 하는 악성코드를 대응하는데 어려움이 존재한다. 두 번째로, 기존연구의 부재로 인한 난제이다. linux 악성코드에 대한 관련 연구는 거의 전무한 실정이며 이마저도 Mirai Botnet이나 Shellshock 등의 주요 악성코드나 취약점 분석 리포트에 한정되어 있으며 공개된 데이터 셋 수집도 거의 없어서 연구에 어려움이 존재한다.

반면에, Cozzi. E, et, al.의 연구에서는 1만여개의 데이터 셋의 샘플을 분석하여 ELF 악성코드가 동작할 때의 특징들 또한 밝혔는데 이는 Table 2.와 같다. Table 2.에 나타나 있는 특징들은 root 권한에 대한 특징 등 linux 악성코드에서 주요하게 나타나는 특징들도 있지만, 전통적인 windows 악성코드에서 나타나는 특징들과 유사하다. 이러한 특징들은 악성코드 분석을 어렵게 하는 악성코드 패킹이나 샌드박스 분석 우회 등의 주요 기법들이 포함되어 있는데, 악성코드 제작자들이 이러한 기법을 이용할 수 있는데 반해 관련 연구나 대응 시스템이 부족한 실정이기 때문에 linux 악성코드에 대응하는 자동화된 시스템 필요하다.

Table 1. The Challenge of Linux Malware Research

challenge	description
computer architecture	linux is known to support more than 10 different architectures
loaders and libraries	ELF file dynamic analysis may be blocked when there are no suitable loaders and libraries
operating system	linux can have many interoperability issues, dependency problems, etc
static linking	static linking makes it difficult for analysts to analyze files
analysis environment	linux malware analysis is difficult to set environments such as architecture, library, and operating system that are perfectly matched
lack of previous studies	there is no comprehensive analysis of the linux malware area

Table 2. The Challenge of ELF Malware Analysis

category	function
ELF header manipulation	anomalous ELF
	invalid ELF
	impact on userspace tools
persistence	subsystems initialization
	time-based execution
	file infection and replacement
	user files alteration
deception	renaming
required privileges	privileges escalation
	kernel modules
packing & polymorphism	UPX variations
	custom packers
process interaction	multiple processes
	shell commands
	process injection
information gathering	*proc* and *sysfs* file systems
	configuration files
evasion	sandbox detection
	processes enumeration
	anti-debugging
	anti-execution
	stalling code
libraries	common libraries

Table 3. ELF Samples that cannot be Properly Parsed by Known Tools

program	errors on malformed samples
readelf 2.26.1	166 / 211
GDB 7.11.1	157 / 211
pyelftools 0.24	107 / 211
IDA Pro 7	- / 211

이외에도, ELF header 등의 정적 정보를 추출하기 위해 readelf, GDB, pyelftools, IDA Pro 등의 분석 도구들로 정적 정보를 추출하였을 때 파일에서 Table 2.의 invalid ELF 등의 특성으로 인해 정적 정보가 추출되지 않는 경우가 발생 가능하다. Table 3.은 각 도구별로 샘플 파일의 정적 정보가 추출되지 않는 비율을 나타낸 것이다.

고가의 상용 분석도구인 IDA Pro 7를 제외한 나머지 분석 도구들의 경우에 다수의 파일들에서 구조적 특징을 정상적으로 추출하지 못하였으며 pyelftools의 경우에는 절반 가량이나 추출하지 못하였음을 볼 수 있다.

결과적으로, linux 악성코드 보안의 경우에 복잡한 환경과 기반 연구의 부재로 인해 도전적 난제들이 다수 존재하고 있는 상황이라고 볼 수 있다.

2.3 The Challenge of Embedded Linux Malware Analysis

모바일 디바이스나 AI 스피커, 자율주행 어시스턴트 등의 IoT 디바이스 시장이 성장함에 따라 IoT 악성코드에 대응하는 보안 기술 및 연구의 필요성이 대두되고 있다. IoT 디바이스는 보통 물리적 제약조건으로 인해 임베디드 linux 아키텍처를 채택한 경우가 대부분이다. 따라서, 본 논문에서는 임베디드 linux 환경을 고려한 악성코드 분석 시스템을 통해 이러한 동향에 기여하고자 한다.

임베디드 linux 악성코드 환경은 linux와 크게 다르지 않지만 몇 가지 특징이 존재한다. A. Costin, et al. 의 연구[11]에 따르면 크게 5가지의 주요 난제들이 존재하는데 이는 Table 4.와 같다.

예를 들어, Firmware Identification은 부족한 표준화, 펌웨어 의존성 등의 복잡한 펌웨어 생태계 등에 기인한다. Unpacking and Custom Formats은 벤더별 독자적인 펌웨어/파일 포맷에 따른 전문가의 정적분석의 어려움 등에 기인한다. 이와 같이, 기존 linux와 여러 공통점과 차이점이 있지만, 주요하게 2개의 관점으로 요약할 수 있다. 첫 번째로, 기존 linux 시스템에 다양한 디바이스, 벤더, 아키텍쳐 등을 포함하여 환경의 복잡성이 심화되었다. 이러한 사실은 환경별로 수많은 타입의 데이터가 분포할 수 있어 정제된 데이터 셋을 수집하는 것을 어렵게 만든다. 또한, 부족한 표준화로 인해 벤더별로 데이터 포맷을 자체적으로 개발하여 배포하는 등 커스텀 포맷으로 인해 데이터에 대한 분석을 어렵게 만든다. 두 번째로, 복잡한 환경으로 인해 필연적으로 분석 전문가의 수동분석과 같은 사람의 개입의 소요가 대단히 크다. 이로 인해 자동화된 분석 시스템의 필요성은 당연하다.

Table 4. The Challenge of Embedded Firmware/Malware Analysis

challenge	description
building a representative dataset	complex environments such as various devices, vendors, architectures, and commands make it difficult to configure a balanced dataset
firmware identification	difficult to reliably extract data from firmware
unpacking and custom formats	lack of a standard format for files / more file unpacking efforts for static analysis
scalability and computational limits	analysis performance is dependent on the speed of computing
results confirmation	need human intervention to verify results

따라서, 본 논문에서는 기존 연구들에서 기술한 이러한 난제들에 대응하기 위해 2개의 분석 시스템으로 구성된 linux 아키텍처의 악성코드들에 대해 자동화된 분석을 수행하는 시스템을 제안한다. 제안하는 시스템은 기존의 표준화된 ELF 포맷을 따르는 데이터 뿐만 아니라 IoT 환경의 복잡한 상황을 고려하여 커스텀 포맷도 분석 가능하도록 설계되었다.

III. 제안모델

3.1 Proposed System

본 논문에서 제안하는 시스템은 크게 2가지의 분류 시스템으로 구성되어 있다. 첫 번째 시스템은 ELF의 구조적 특징을 활용한 분류 시스템이다. 해당 시스템은 수집된 파일을 ELF parsing 도구들을 이용하여 3.2절에서 서술할 ELF feature vector를 추출 및 정규화하고 이를 SVM, DNN, Naive Bayes 등의 모델을 이용하여 분류한다. 두 번째 시스템은, Table 3.과 같이 ELF 표준을 따르지 않거나 정상적으로 구조적 특징을 얻어낼 수 없는 경우 바이너리 데이터를 기반으로 악성코드를 분류하는 분류 시스템이다. ELF parsing이 불가능한 파일들의 바이너리 데이터를 이미지화하고 이를 CNN 모델을 이용하여 분류한다.

ELF Structure feature based Classification은 바이너리의 ELF 구조에서 feature를 바로 추출할 수 있고 SVM, DNN 등의 알고리즘으로 분류하기 때문에 비교적 고속으로 악성코드를 분류할 수 있지만 ELF 구조를 따르지 않는 바이너리의 경우에는 분류가 불가능하다. 반면에, Binary based Image Classification은 바이너리를 이미지로 치환하고 이를 통해 CNN 알고리즘을 통해 분류하기 때문에 비교적 느리지만 ELF Structure feature based Classification에서 분류하지 못한 바이너리에 대해 분류가 가능하다. 결과적으로, ELF feature를 활용하여 상대적으로 고속의 SVM(Support Vector Machine), DNN(Deep Neural Network), Naive Bayes 모델로 악성코드를 우선적으로 분류하고, ELF feature를 사용할 수 없는 데이터들에 대해 바이너리 기반의 이미지 분류 모델로 분류한다. 이러한 것을 감안하여 설계한 분류 시스템의 구성도는 Fig 1.과 같다.

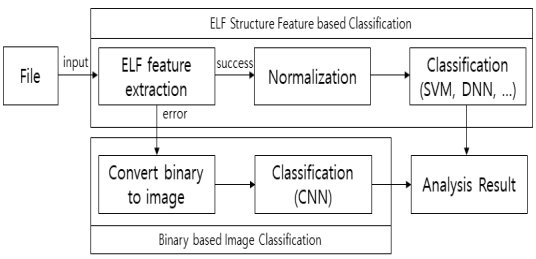

Fig. 1. System Overview

3.2 ELF Feature

기존에 windows PE(Portable Executable) 파일을 활용한 악성코드 분석 도메인의 경우에는 파일 포맷에서 도출할 수 있는 여러 가지 정보들을 분석하고 이를 통계적 해석, 혹은 전처리를 통해 feature로서 가용한 연구[12, 13, 14]들이 다수 존재한다. 이는 파일 포맷이 가지고 있는 본연의 목적과 기능들이 분석에 사용될 수 있는 feature로 드러나기 때문인데, 마찬가지로 linux 도메인의 경우에도 표준화된 ELF가 존재하기 때문에 파일 포맷을 활용한 악성코드 분석을 진행할 수 있다. ELF의 구조는 Fig. 2.와 같다.

ELF 구조에서, ELF 헤더는 파일의 구성을 나타내며 각 섹션은 linking을 위한 오브젝트 파일의 정보 등을 가지고 있다. 섹션 헤더 테이블의 경우에는 섹션 이름 및 크기와 같은 정보를 가지고 있다. 본 논문에서는 이러한 주요 헤더에서 얻을 수 있는 feature들 중에서, 비교적 구조적 역할이 분명하거나 해석이 용이한 feature들을 통계적 해석을 통해 가용하였다. 우선, 가용한 헤더들과 그 역할의 예시는 Table 5.와 같다.

이러한 파일의 구조에서 얻을 수 있는 구조적 정보들은 특정한 값으로 그 기능을 나타낸다. ARM ELF 스펙[15]의 header의 여러 구조체들 중 program header의 p_type 구조체와 section header의 sh_flags 구조체의 세부 값과 기능은 Table 6. 및 Table 7.과 같다. 이러한 구조체들은

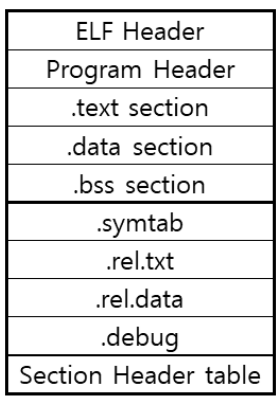

Fig. 2. ELF Structure

Table 5. ELF Header Overview

field	description
e_entry	ELF entry point
e_phentsize	size of program headers
e_phnum	number of program headers
e_shentsize	size of section headers
e_shnum	number of section headers
e_ehsize	size of ELF header
p_type	segment type
p_filez	file of segment size
p_memsz	memory of segment size
sh_name	section name
sh_type	section type
sh_size	size of section

Table 6. Program Header's p_type Structure

field	value	description
PT_NULL	0	unused/undefined program header
PT_LOAD	1	loadable segment
PT_DYNAMIC	2	dynamic linking information
PT_INTERP	3	the location and size of a null-terminated path
PT_NOTE	4	the location and size of auxiliary information
PT_SHLIB	5	unspecified semantics
PT_PHDR	6	the location and size of the program header table

Table 7. Section Header's sh_flags

field	value	description
SHF_WRITE	0x1	the section contains data that should be writable during process execution
SHF_ALLOC	0x2	the section occupies memory during process execution
SHF_EXECINSTR	0x4	the section contains executable machine instructions

지정된 값으로 파일의 기능을 직/간접적으로 나타내기 때문에 통계적 분석을 통해서 정상파일과 악성코드를 일정 수준으로 분류할 수 있다. 다음 Fig. 3., Fig. 4., Fig. 5., Fig. 6.은 본 논문에서 수집한 데이터 셋을 이용하여 도출한 feature 일부의 통계적 분포를 나타낸다. 데이터 셋 구성은 4.1절에 기술하였다.

데이터 셋에서 악성코드의 개수가 정상파일보다 약 4배 많다는 점을 감안하더라도, 위와 같은 feature들의 통계적 분포가 일정 수준 차이가 난다는 것을 확인할 수 있다. 이러한 feature들은 적절

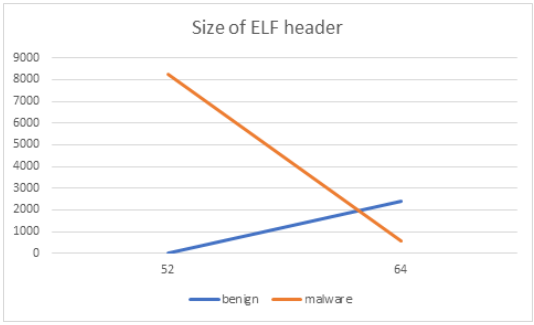

Fig. 3. Size of ELF Header

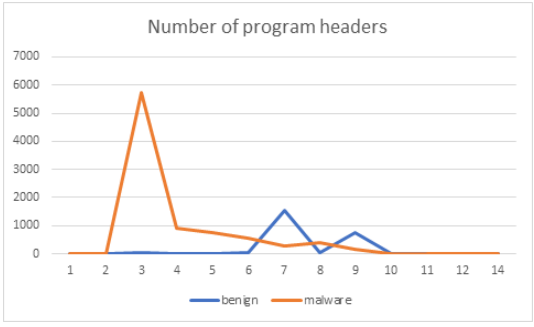

Fig. 4. Number of Program Headers

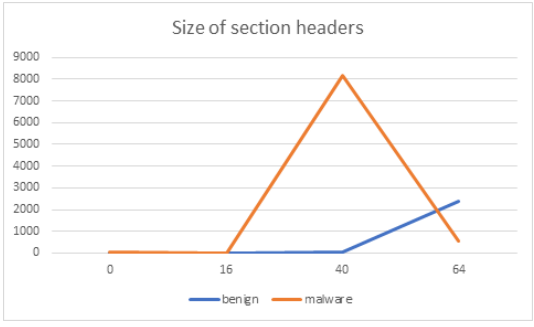

Fig. 5. Size of Section Headers

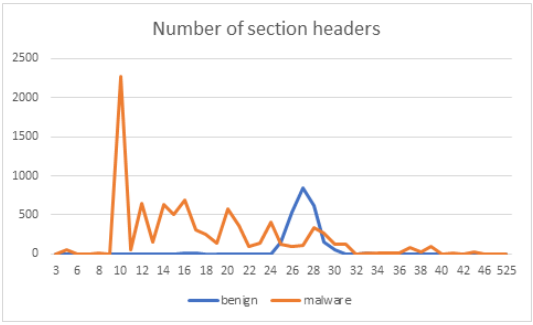

Fig. 6. Number of Section Headers

한 전처리나 정규화 등을 포함하여 전통적인 통계 모델인 Naive Bayes 알고리즘 뿐만 아니라 다양한 머신러닝 알고리즘을 통해 악성코드를 분류하는 데 활용될 수 있다. 특히, 이러한 구조적 특징의 경우에는 위와 같은 수동적인 통계적 분석이나 정규화 등을 통하여 특징 값의 성능을 향상시킬 수 있는 여지가 존재하며 구조적 특징 자체에서 추가적인 해석이나 활용 방안이 존재한다. 예를 들어, Fig. 6.의 Number of Section Headers의 경우에 정상파일보다 악성코드의 섹션 헤더의 개수가 적은 수치를 나타내는 것을 확인할 수 있다. 이러한 경향은 악성코드 분석 전문가가 수동적인 통계적 분석을 수행한다고 하였을 때, 세부적인 분석 이전에 악성코드에 주로 적용되는 패킹 알고리즘으로 인해 섹션 개수가 줄어드는 것이라고 직관적인 추론을 할 수 있다. 결론적으로, 이러한 구조적 특징 분석의 가치는 직관적이며, 사전 지식과 연계한 추가적인 해석이 가능하며, 표준화된 일련의 구조를 나타내므로 다른 분석 방법론 대비 비교적 빠르다는 장점이 존재한다. 반면에, 이러한 구조적인 특징은 Table 3.과 같이 가지고 있는 데이터의 구성이나 도구에 따라 구조적 특징을 얻어낼 수 없는 경우가 존재하며, 특징 값들의 수동적인 분석 및 해석이 동반된다는 단점이 존재한다. 3.3절에서는 이러한 단점을 상호보완 할 수 있는 이미지 기반의 분석 기법을 제안한다.

본 논문에서는 이러한 구조적 특징 총 200개를 사용하여 feature vector를 구성하였고 이를 통해 악성코드와 정상파일을 분류한 결과를 4절에서 제시한다.

3.3 Binary based Image Feature

3.2절의 ELF 구조적 특징을 활용한 악성코드 분석 기법에서, ELF 특징 값들을 추출할 수 없는 데이터의 경우에 분석이 어려운 점이 존재하였다. 임베디드 linux 아키텍처를 채택하고 있는 대부분의 IoT 디바이스 보안 환경에서는 이러한 경향이 더 심화될 것이기 때문에 이에 대한 대응 방안은 상당히 중요하다. 기존의 복잡한 linux 기반의 PC 환경보다 임베디드 linux는 더 다양한 디바이스나 벤더, 아키텍처, 명령어 등으로 인해 극도로 복잡한 환경에 직면해 있다. 또한, 기존의 소프트웨어들과는 다르게 임베디드 소프트웨어의 경우에 표준이 거의 전무한 실정이므로 벤더별로 자체 파일 형식을 개발하는 상황이다. 결론적으로, 이러한 상황에 대응하기 위해 구조적 특징에 의존하지 않는 별도의 데이터 분석 체계가 필요하다. 바이너리 데이터의 경우에는 데이터만 존재한다면 어떤 경우에도 활용 가능하다.

본 논문에서는 바이너리 데이터를 사용하여 악성코드를 분류하는 방법으로 바이너리를 시각화시켜 CNN 알고리즘을 통해 악성코드를 분류하는 기법을 채택하였다. 시각화는 기본적으로 1바이트를 grayscale 이미지 1픽셀로 표현하였고 이미지 크기 및 CNN 알고리즘을 고려해 파일 크기에 따라 이미지를 압축하거나 제로패딩하는 방법론을 적용하였다. 예를 들어, 본 논문에서는 64x64 크기의 이미지로 CNN 모델을 통해 학습 및 분류하게 되는데 4,096 바이트 미만의 파일은 4,096바이트가 되도록 제로패딩하고 4,096바이트 이상의 파일은 4,096바이트가 되도록 그 배수만큼의 바이트를 평균을 내어 압축한다. 해당 프로세스의 동작 과정은 Fig. 7.과 같다.

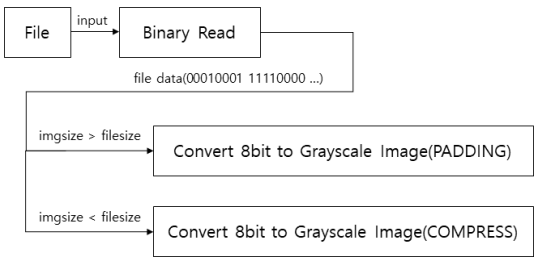

Fig. 7. Binary Visualization Approach

IV. 시험결과

4.1 Dataset

본 논문에서 가용한 데이터는 총 2만개로,

virusshare에서 제공하는 linux 악성코드 1만개[16]와 linux 아키텍쳐에서 자체 수집한 시스템 정상파일 1만개로 구성되어 있다. 이 중, 3.2절에서 제안하는 구조적 특징을 온전히 얻을 수 없는 데이터들을 제외하면 데이터 구성은 Table 8.과 같다. Table 3.과 같이 feature를 추출할 수 없는 경우가 존재하듯이, linux 아키텍쳐의 환경은 매우 복잡하고 정상파일의 경우에는 다양한 linux 아키텍쳐의 시스템 파일을 수집하였기 때문에 이러한 경향이 심화된 것으로 보인다. 3.3절의 바이너리 기반의 분석의 경우에는 바이너리 값 자체를 활용하기 때문에 전체 데이터를 가용 가능하다.

Table 8. Dataset Configuration

category	malware	benign	description
ELF structure based	8,812	2,423	ELF extraction error exists
binary based	10,000	10,000	-

4.2 ELF Feature based Analysis Result

본 절에서는 SVM, Random Forest, Naive Bayes, k-NN, DNN 알고리즘과 ELF feature를 기반으로 도출한 분석 결과를 제시한다. DNN의 경우에는 hidden layer 2개와 각 layer 당 10개의 unit으로 구성되었다. 실험은 4.1절의 데이터 셋을 사용하였고 over-fitting를 완화하기 위해 학습 및 테스트에 데이터 셋을 랜덤하게 8:2로 나누어 실험하였다. 실험으로 도출한 테스트 데이터에 대한 평균 정확도와 커버리지는 Table 9.와 같다. 한정된 데이터로 인해 데이터가 많아질수록 강점을 나타내는 k-NN과 DNN의 경우에는 다른 알고리즘에 비해 성능이 떨어짐을 보이며 SVM은 linear 모델의 단점을 완화하고자 kernel SVM을 적용하였지만 악성코드의 다양성으로 인해 한계점이 존재하는 것으로 판단된다.

Table 9. ELF Feature based Result

category	test accuracy	coverage	remarks
SVM	87.76%	56.17%	kernel SVM
Random Forest	99.68%	56.17%	-
Naive Bayes	97.41%	56.17%	-
k-NN	86.64%	56.17%	k=5
DNN	76.94%	56.17%	step= 10,000

4.3 Binary based Image Analysis Result

본 절에서는 CNN 알고리즘과 바이너리를 기반으로 도출한 분석 결과를 제시한다. CNN의 경우에는 convolution layer, fully connected layer 각각 2개로 구성되었다. 실험 데이터는 4.2절과 동일하지만 바이너리를 기반으로 하므로 2만개의 전체 데이터에 대해 실험이 가능하였다. 실험으로 도출한 테스트 데이터에 대한 정확도와 커버리지는 Table 10.과 같다. 해당 결과를 통해 제안기법이 바이너리 데이터를 대상으로, 아키텍쳐에 무관하게, 효과적으로 동작하는 것을 확인할 수 있다. 이러한 결과를 미루어 보아 제안 시스템과 같이 ELF 기반의 분석기법과 연계하면 linux 아키텍쳐를 타겟으로 하는 악성코드에 대해 효과적으로 동작할 것이라는 것을 확인할 수 있다. Fig. 8은 앞선 두 결과에 대해 보다 객관적인 분류성능평가를 위한 그래프이다. 해당 그래프는 각 알고리즘 별 Accuracy, Precision,

Table 10. Binary based Result

category	test accuracy	coverage	remarks
CNN	95.05%	100.00%	filter = 64, stride = 1x1

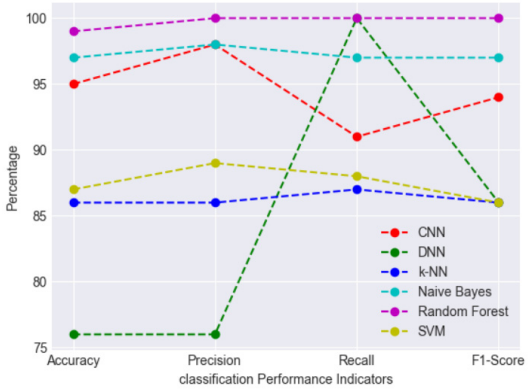

Fig. 8. Binary Visualization Approach

Recall, F1-Score 수치를 나타낸다.

V. 결 론

본 논문에서는 IoT 환경에 더불어 크게 증가하고 있는 linux 아키텍처에 대한 보안 위협에 대응하기 위한 머신러닝 기반의 linux 악성코드 분석 방법론에 대해 주로 다루었다. 기본적으로 ELF 파일에 존재하는 여러 구조적 특징들에서 추출할 수 있는 feature들을 밝히고 이에 대한 통계적 해석을 그래프로 제시하였다. 특히, 기존의 windows 환경에서의 pe 분석 문제와는 다르게 복잡한 linux 아키텍처 환경을 고려하여 ELF 구조를 따르지 않는, 혹은 링킹과 실행 관점에서 해석이 불가능하거나 상이한 악성코드들에 대해서도 대응하기 위하여 바이너리를 기반으로 한 분석 프로세스도 제안하였다. 이는 IoT 환경에서 부족한 표준화 수준으로 인해 디바이스, 벤더, 아키텍처 별로 자체적인 데이터 포맷을 개발하는 상황에서 유용하게 동작할 것으로 기대된다.

제안모델에서는 위의 두 가지 분석 프로세스를 이용하여 효과적으로 linux 악성코드를 분류할 수 있는 시스템을 제안하였다. 여러 연구동향들에서 linux 악성코드 데이터 셋에서 ELF 구조적 특징을 추출 가능한 데이터는 절반 이상이 가능한 것으로 보인다. 따라서, 상대적으로 고속으로 동작할 수 있는 ELF 구조적 특징과 SVM, DNN, Naive Bayes 등의 방법론을 활용한 분류 시스템으로 악성코드를 전처리 함으로써 많은 수의 악성코드들에 대해 전처리 가능하다. 여기에서 ELF 표준을 따르지 않는 데이터들을 상대적으로 느리지만 전체 데이터에 대해 분류가 가능한 CNN 모델을 이용하여 분류할 수 있다. 시험 결과에서는 두 개의 분석 프로세스에 대해 유의미한 악성코드 분류 정확도 결과를 보였다.

결과적으로, 본 논문은 ELF 구조적 특징을 활용하여 feature들을 통계적 분석하고 여러 머신러닝 알고리즘으로 분류한 결과를 제시하였고, 아키텍처나 파일 포맷 등 복잡한 linux 악성코드 환경에 무관한 바이너리 기반의 탐지 결과를 제시하였다. 이는 기존의 pc 환경보다 복잡한 IoT 환경의 임베디드 linux 악성코드 등을 탐지 하는데 확장할 수 있다. 또한, 부족한 linux 악성코드의 기반 연구에 기여할 수 있을 것이라 기대한다. 반면에, 해당 시스템은 머신러닝 방법론을 기반으로 하고 있기 때문에 이러한 방법론에서 필연적으로 발생하는 미탐지된 악성코드를 최소화해야하는 과제와 다양한 데이터 셋을 구성하여 보다 신뢰성 있는 시험결과를 도출해야할 과제가 존재한다.

향후, linux 악성코드 데이터 셋 수집 난제를 해결하기 위한 방안을 연구하여 본 시스템의 결과 정확도를 높일 수 있을 것이며 바이트 n-gram 등 바이너리에 적용할 수 있는 여러 방법론들을 연구하여 기존 시스템에 비해 IoT 환경에 효과적으로 적용할 수 있는 자동화된 분류 시스템을 제안할 수 있을 것이라 기대한다.

References

[1] AV-TEST, “Security Report 2016/17”, https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2016-2017.pdf, accessed Dec 10.

[2] Securelist.com, “Mobile malware evolution 2018”, https://securelist.com/mobile-malware-evolution-2018/89689/, accessed Dec 10.

[3] KISA & KrCERT, “2016 Mirai Malware Trends Report”, https://www.krcert.or.kr/data/reportView.do?bulletin_writing_sequence=24864&queryString=cGFnZT0xJnNvcnRfY29kZT0mc2VhcmNoX3NvcnQ9dGl0bGVfbmFtZSZzZWFyY2hfd29yZD1taXJhaSZ4PTAmeT0w, accessed Dec 10.

[4] Li. Jin, Sun. L, Yan. Q, Li. Z, Srisa-an. W and Ye. H, “Significant permission identification for machine-learning-based android malware detection,” IEEE Transactions on Industrial Informatics, vol. 14, no. 7, pp. 3216-3225, July. 2018.

[5] Kolosnjaji. B, Demontis. A, Biggio. B, Maiorca. D, Giacinto. G, Eckert. C and Roli. F, “Adversarial malware binaries: Evading deep learning for malware detection in executables,” 2018 26th European Signal Processing Conference(EUSIPCO), IEEE, Dec. 2018.

[6] Machiry. A, Redini. N, Gustafson. E, Fratantonio. Y, Choe. Y. R, Kruegel. C, and Vigna. G, "Using Loops For Malware Classification Resilient to Feature-unaware Perturbations," Proceedings of the 34th Annual Computer Security Applications Conference. ACM, pp. 112-123, Dec. 2018.

[7] Kakisim. A. G, Nar. M, Carkaci. N and Sogukpinar. I, "Analysis and Evaluation of Dynamic Feature-Based Malware Detection Methods," International Conference on Security for Information Technology and Communications. Springer, Cham, pp. 247-258, Nov. 2018.

[8] Raff. E, Zak. R, Cox. R, Sylvester. J, Yacci. P, Ward. R and Nicholas. C, "An investigation of byte n-gram features for malware classification," Journal of Computer Virology and Hacking Techniques, vol. 14, no. 1, pp. 1-20, Feb. 2018.

[9] HaddadPajouh. H. Dehghantanha. A, Khayami. R and Choo. K. K. R, "A deep Recurrent Neural Network based approach for Internet of Things malware threat hunting," Future Generation Computer Systems 85, vol. 85, pp. 88-98, Aug. 2018.

[10] Cozzi. E, Graziano. M, Fratantonio and Balzarotti. D, "Understanding Linux Malware," IEEE Symposium on Security and Privacy, pp. 161-175, May. 2018.

[11] Costin. A, Zaddach. J, Francillon. A and Balzarotti. D, "A large-scale analysis of the security of embedded firmwares." 23rd {USENIX} Security Symposium ({USENIX} Security 14), pp. 95-110, Aug. 2014.

[12] U. Baldangombo, N. Jambaljav, SJ. Horng, "A Static Malware Detection System Using Data Mining Methods," Cornell University, Aug. 2013.

[13] K. Iwamoto, K. Wasaki, "Malware Classification based on Extracted API Sequences using Static Analysis," Internet Engineering Conference, pp. 31-38, Nov. 2012.

[14] Younghoon Lee, "A Study on Generic Unpacking using Entropy Variation Analysis," Journal of the Korean Institute of Information Security and Cryptology, vol. 22, no. 2, pp. 179-188, June. 2012.

[15] ARM, "ARM ELF", https://www.uclibc.org/docs/psABI-arm.pdf, accessed Dec 10.

[16] virusshare, "virusshare", https://virusshare.com/, accessed Dec 10.

〈저자소개〉

황 준 호 (Jun-ho Hwang) 학생회원
2012년 3월~현재: 호서대학교 정보보호학과
〈관심분야〉 머신러닝, 정보보호, 통계학

이 태 진 (Tae-jin Lee) 종신회원
2003년 1월~2017년 2월: 한국인터넷진흥원 팀장
2017년 3월~현재: 호서대학교 정보보호학과
〈관심분야〉 시스템 보안, 악성코드 분석, 침해사고 대응