electronics

MDPI

Article

Platform-Independent Malware Analysis Applicable to Windows and Linux Environments

Chanwoong Hwang [1], Junho Hwang [1], Jin Kwak [2] and Taejin Lee [1,*]

[1] Department of Information Security, Hoseo University, Asan 31499, Korea; 20205251@vision.hoseo.edu (C.H.); 20185184@vision.hoseo.edu (J.H.)
[2] Department of Cyber Security, Ajou University, Suwon 16499, Korea; security@ajou.ac.kr
* Correspondence: kinjecs@hoseo.edu

Received: 13 April 2020; Accepted: 6 May 2020; Published: 12 May 2020

Abstract: Most cyberattacks use malicious codes, and according to AV-TEST, more than 1 billion malicious codes are expected to emerge in 2020. Although such malicious codes have been widely seen around the PC environment, they have been on the rise recently, focusing on IoT devices such as smartphones, refrigerators, irons, and various sensors. As is known, Linux/embedded environments support various architectures, so it is difficult to identify the architecture in which malware operates when analyzing malware. This paper proposes an AI-based malware analysis technology that is not affected by the operating system or architecture platform. The proposed technology works intuitively. It uses platform-independent binary data rather than features based on the structured format of the executable files. We analyzed the strings from binary data to classify malware. The experimental results achieved 94% accuracy on Windows and Linux datasets. Based on this, we expect the proposed technology to work effectively on other platforms and improve through continuous operation/verification.

Keywords: malware analysis; binary analysis; strings analysis; deep neural network; feature importance

1. Introduction

1.1. Background

Recently, security threats from malware have been increasing every year. As new and unknown malware appears, there is a limit to responding with signature-based antivirus. Previously distributed malware was mostly for the purpose of stealing information or remote control of devices, but recently ransomware that requires money after encrypting files in electronic devices has surged. Ransomware is mostly distributed using phishing emails or access to malware-infected pages and file sharing methods such as torrent. To counter such attacks, it is necessary to regularly patch (update) antivirus software and applications. This type of security patching is vulnerable to zero-day attacks, i.e., before the patch was applied, and may result in defenselessness. To respond to this, AI technology that can predict and detect new and malware variants based on machine learning has been developed. This method however often produces false positives and many studies have been conducted to reduce the number of these errors [1–4].

With the advent of smart homes, embedded systems have been developed that use systems mounted in various electronic products such as TVs, radios, and air purifiers. In addition, the development of IoT devices that connect sensors to all objects to communicate and interact with each other has provided convenience to users. IoT and embedded technology are services for various interactions, not just development techniques, and various services are configured according

to the provider and user location. There are various analysis environments that support many services available. Table 1 shows the analysis environment used for IoT/embedded environments. However, malware is appearing on various platforms due to the increase of IoT/ embedded systems as well as existing PCs. This is a malicious operation that leaks device information or account information and controls the device remotely. In order to respond to these various malware platforms in real time, there are limitations in existing security methods, and there is a disadvantage in that security policies must be implemented for each platform. This is why we started research that can analyze malware independently on various platforms.

Table 1. Examples of analysis environments that support IoT/embedded devices.

Name	Tizen	Brillo	Fuchsia	LiteOS	YunOS
Developer	Linux Foundation, Tizen Association, Samsung, Intel	Google	Google	Huawei	Alibaba
OS family	Unix-like	Android	Android	Linux based	Android
Platform	ARM and x86	ARM, Intel x86, and MIPS-based hardware	ARM(32/64 bit) and PC(64 bit)	ARM (M0/3/4/7, A7/17/53, ARM9/11), x86, and RISC-V	-

The latest research shows that on average people own three Internet-connected smart devices such as smartphones and tablets [5]. Various IoT devices, such as smart devices, constitute an endpoint. Most cyberattacks use malware to control the endpoint. Based on this, cyberattacks lead to the flow of internal network scans, main server access and other large-scale security incidents. Most of the existing endpoints used to be PC environments, but as IoT devices have become widely used, there are a large number of IoT endpoint devices. Many of these IoT endpoints can increase the number of targets to attack and, in conjunction with the current weak IoT device security environment, can lead to serious security incidents. As the information accessible by IoT devices extends beyond personal information to very sensitive information such as financial services and autonomous driving information, the security threats are becoming more serious. Security experts, on the other hand, need to establish malware analysis and security policies for multiple IoT devices. Manual analysis of malicious codes on these various architectures is difficult. Therefore, there is a need for a method that can statically analyze malware in various architectures and automatically run as code.

The problems with existing malware analysis technologies can be summarized as follows: First, a lack of complex Linux-based malware analysis. According to Eclipse's Key Trends for IoT Developers 2018 [6], Linux accounts for 71.8% of the operating systems used for IoT devices, gateways, and cloud back-end devices, and the application of Linux to industrial devices is also expanding. However, pattern-based and AI-based malware analysis techniques are mostly limited to Windows malware, and there is no Linux-based anti-malware technology which use is expected to increase significantly in the IoT/embedded environment. In addition, the development of malicious code response technology that can operate in various architecture environments is complicated because it is mainly based on network logs. Second, an endpoint environment that is not affected by the platform should be considered. The proposed model can classify malware that penetrates endpoints based on AI-based malware analysis technology that is not affected by the platform. The existing AI-based malware analysis technology consisted mainly of analysis and research on Windows, Android, etc., and the analysis technology of many architectural/operating system combinations in Linux or IoT environments was not fully studied. Thus, the platform-independent malware analysis proposed in this paper is a malware analysis technology that can be commonly applied to any binary data regardless of the architecture or type of operating system. It is applied to a 5G/IoT environment and its performance and results have been verified using open data sets and self-collected data sets. The system is an effective and sustainable model that can apply separate security policies, such as expert analysis, to complement each other's inherent technology shortcomings.

1.2. Challenges with Linux/Embedded/IoT Environments

Statistics show that Microsoft's Windows operating system has a 83% of the PC market [7], so malware writers have also targeted the Windows operating system. Malware-related research is also mainly conducted only in the Windows operating system environment, so there is a lack of research on Linux malware. Cozzi et al. [8] is the main study that revealed the current status of Linux malware analysis research. The research revealed the major challenges that can arise from Linux malware research and major operational processes of samples of more than 10,000 datasets built on its own. The main challenges that can arise from Linux malware research can be represented as:

1. Diversity of computer architectures: Linux is known to support more than 10 different architectures.
2. Diversity of loaders and libraries: If you do not have the appropriate loader and library for your analysis environment, you can prevent the sample from starting execution.
3. Diversity of operation systems: Linux can have many interoperability issues, dependency problems, etc.
4. The challenge of static links: Static linking makes the resulting binary code more portable, but it is difficult for analysts to analyze the files.
5. The challenge of the analysis environment: Linux malware analysis is difficult to perform in environments such as architecture, libraries, and operating systems that are perfectly matched.
6. Lack of previous studies: It is not clear how to design and implement an analysis pipeline specifically tailored for Linux malware and there is no comprehensive analysis.

First, it is related to various target environments. Linux systems are known to support dozens of architectures, which requires analysts to prepare different sandboxes and port different architecture-specific analysis components to support each one. In addition, a copy of the requested loader to use the ELF file format might not exist in the analysis environment, preventing the sample from starting execution. With the recent increase in the number of IoT devices, considerations such as devices, considerations such as device type, vendor, and software dependencies become more complex, making it difficult to deal with malware targeting these systems. Second, there is a lack of existing research. It is not clear how to design and implement an analytics pipeline specifically designed for Linux malware, and existing studies build and use a representative dataset using honeypots that focus solely on botnets.

Recently, as the industrial market is growing around the Internet of Things (IoT), the number of various embedded devices is overflowing. In addition, the need for security technology and research to IoT malware is emerging. The embedded Linux malware environment is not very different from Linux, but there are some distinctive features. According to Costin et al. [9], there are five major challenges that can be summarized as follows:

1. Difficulty to build a representative dataset: In complex environments with various devices, vendors, architectures, and commands, it is difficult to construct scale datasets.
2. Difficulty extracting data by identifying firmware: One challenge often encountered in firmware analysis and reverse engineering is the difficulty of reliably extracting metadata from a firmware image.
3. Unpacking and custom formats: While this task would be easy to address for traditional software components, where standardized formats for the distribution of machine code, resources and groups of files exist, embedded software distribution lacks standards.
4. Scalability and computational limits: One of the major advantages of performing extensive analysis is the ability to correlate information across devices. Thus, analysis speed is crucial to computing speed.

5. Direct results check: Confirming the results of the static analysis on firmware devices is a tedious task requiring manual intervention from an expert. Scaling this effort to thousands of firmware images is even harder.

Typically, collecting refined datasets is difficult because the environment is complex due to a variety of devices, vendors, and architectures in existing Linux systems. Lack of standardization also makes it difficult to analyze data due to vendor-specific data formats. In addition, due to the complexity of the environment, human intervention, such as manual analysis by analysis experts, is very much required. Thus, for Linux-based malware security, many problems arise due to complex environments and a lack of basic research and requires a natural automated analysis system.

With the development of IoT, new security problems are emerging. The main challenge for IoT security are a consequence of the heterogeneity and the large scale of objects. Zhang et al. [10] described the ongoing challenges of security and research opportunities. The main challenges related to IoT-related malware can be summarized as follows:

1. Linux-based IoT malware: The first IoT malware discovered was Linux-based malware.
2. Limited resources: Unlike in x86-architectured PCs, the computing power of IoT devices is relatively small.
3. System vulnerability easily exposed: Most of the IoT is occupied by the mobile operating system Android, and unlike iOS, Android is open-sourced.
4. Lack of previous studies: To our best knowledge, at present there is little research work dedicated to countermeasures against IoT-targeted malware.

As aforementioned, the threat of IoT-targeted malware is serious due to the limited resources of IoT devices. Moreover, conventional security mechanisms against malware can be infeasible while shifted directly from the common x86 architecture platforms to the IoT platform. There are also security issues for Android, which accounts for the largest portion of IoT devices. Unlike iOS, Android is open-sourced. Therefore, it is easy to detect the vulnerability of the system. Once malware compromises front end devices, the IoT network is exposed to threats. The main concern is sensitive data leakage. The current permission protection only provides course-grain management, namely all-or-nothing choice, to restrict the type of connected devices and disable the runtime control. The malware threats and countermeasures in IoT will become critical and should addressed. Therefore, without a generic abstraction of the IoT malware, current solutions can be ad-hoc and even inapplicable.

1.3. Contribution

1.3.1. Malware Analysis Support in Various Linux/IoT Environments

Linux/IoT supports various environments and services, so it is difficult to understand the architecture or operating system used. Table 2 shows the various platforms that support IoT. In addition, with the increase in IoT devices, various endpoint environments are being constructed. There are also security issues. Most IoTs use the Linux operating system called Ubuntu Core [11]. However, the IoT configuration has so far faced an incompleteness that is difficult to analyze due to the lack of a standard architecture definition worldwide. As a result, there are difficulties in analyzing Linux, and security becomes more difficult when utilizing the vulnerable IoT to take control of internal systems [12]. The reason Linux malware analysis is difficult is that files are not in the same structural format as Windows. The executable file on Linux follows the Executable and Linkable Format (ELF) file format, but each executable has a different file format. Table 3 shows the results of extracting the file-structured formats Portable Executable (PE) and ELF using 20,000 datapoints each in Windows and Linux environments.

Table 2. Various platforms supporting IoT.

Service Platform	Product
Operation System	Linux, Android, Apple, Windows etc.
Hardware Platform	Arduino, Raspberry Pi, MOBIUS etc.
Connection Platform	Modacom, Cisco, KT, LG U+, SK etc.
Data Platform	NAVER, DAUM, Google etc.

Table 3. Number of parsed data based on ELF and PE file formats in Linux and Windows.

	Linux (ELF)			Windows (PE)		
	Total Count	**Analysis Count**	**Ratio**	**Total Count**	**Analysis Count**	**Ratio**
Malware	10,000	7904	79.04%	14,300	14,298	99.98%
Benign	10,000	2520	25.2%	5700	5699	99.98%

The interpretation of Table 3 is as follows. In most Windows environments, the file structure format is judged to be the same, and in Linux environments, the ELF file structural format is followed, but it is different. Therefore, it is difficult to analyze malware statically in Linux environments. To contribute to the solving of these problems, the method proposed in this paper is to extract a binary format and analyze malware. We propose an approach to analyze malware on endpoints even if the file has a different structural format without being affected by the operating system. Later, through the proposed technology, new and variant malware can be detected through K-nearest neighbor (KNN), a distance-based similarity comparison algorithm with existing data.

1.3.2. Automatic Linux/IoT Malware Analysis Technology with One Code

There is a lack of previous research on Linux/IoT systems compared to Windows systems, and dozens of architectures are supported. To support dozens of architectures, it was necessary to create malware analysis codes for each architecture. It is expensive and complicated from an operational point of view because it requires a lot of code to be managed. IoT analysis is the same. As the number of IoT devices increases, considerations such as dependency with software become more complicated because each device has a different type. Therefore, this paper proposes an approach to solve the considerations of Windows and Linux/IoT environment at the same time. The proposed technology operates automatically using a single code for complex malware analysis in a Windows, Linux environment, or an IoT environment supporting various architectures. In the future, it can be used as a base technology for automatic analysis considering the operating environment of Linux/IoT malware. It is possible to analyze Windows, Linux, or IoT with a single code of complex analysis technology supporting various architectures.

2. Related Work

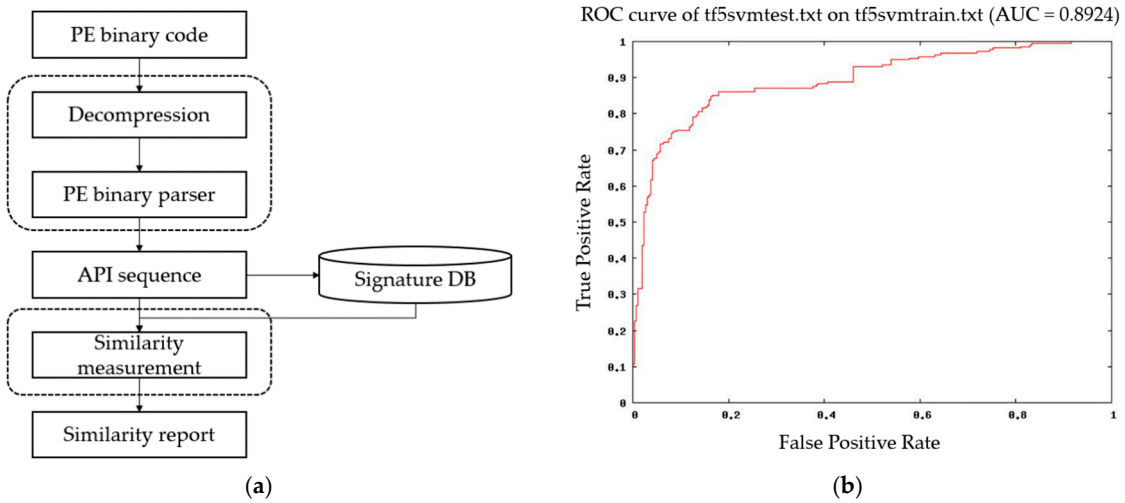
2.1. PE Format-Aware Analysis

Pattern-based and AI-based malware analysis techniques have been actively studied to cope with malware that penetrates into endpoints. In particular, in the field of static analysis of malware, studies were mainly conducted to analyze and classify PE files executable for major architectures in commercial environments such as Windows. Markel et al. [13] compared the experimental results using naïve Bayes, decision tree, and logistic regression, which are machine learning techniques using Windows Portable Executable 32-bit (PE32) file format. The experimental data used 22,500 training samples and was tested on 2500 test samples. Table 4 show the experimental results of naïve Bayes, decision tree, and logistic regression according to malprev, which is the percent of records in the sample that correspond to malicious files. The results shows a 0.97 F-score when using the decision tree technique.

Table 4. Experimental results according to naïve Bayes, decision tree, logistic regression using PE file format.

Malprev (Train, Test)	Naïve Bayes	Decision Tree (CART)	Logistic Regression
(0.5, 0.5)	0.5127	0.9792	0.9456
(0.1, 0.1)	0.4640	0.9270	0.7941
(0.5, 0.1)	0.4804	0.9750	0.7905
(0.01, 0.01)	0.4250	0.7581	0.3023
(0.5, 0.01)	0.3342	0.5247	0.2873
(0.001, 0.001)	0.0493	0.4157	0.03124
(0.5, 0.001)	0.0697	0.1193	0.04857

There is also a window API analysis study called by the PE file [14–16]. Shankarapani et al. [17] proposed an effort to detect malware based on Windows API calls. Figure 1a shows the overall operation. The API calling sequence is extracted using a PE parser and malware classified by measuring similarity with known malware sequences or a signature DB. The TF-IDF technique was used to remove commonly appearing API sequences. After that, the executable files were classified using SVM. Figure 1b shows the ROC curve obtained from the malware detection using SVM.



(**a**) (**b**)

Figure 1. It shows the results of malicious code detection based on Windows API calls: (**a**) Operating process; (**b**) ROC curve obtained from malware detection using SVM.

Islam et al. [18] suggested that the extraction of printable string information from static features and malware samples and classification into a single test is better than the result obtained by using single features individually. Features merge extracted static features using both function length Frequency (FLF) and printable string information (PSI) methods. FLF uses the function name frequency of the extracted string. In addition, PSI is used to create a list of all strings that occur in the database. The experimental data used feature information from over 1,500 samples, including unpacked trojans and viruses from the CA Zoo along with clean files from several different Windows environments. In addition, they used five well-known classification algorithms: naïve Bayes, SVM, random forest, decision table, and IB_1. It includes a tree-based classifier, the nearest neighbor algorithm, and a statistical learning algorithm, each tested with the booster algorithm Adaboost. Table 5 show the experimental results. As a result, a classification accuracy of 98.86% or more was achieved.

Table 5. Performance comparison when using the basic classifier and boosting technology: (**a**) Results of analysis through the basic classifier; (**b**) Analysis result using AdaBoost.

(**a**)					
Base Classifier					
	NB	SVM	IB_1	DT	RF
W.Avg	94.31	97.77	97.69	98.15	96.69
Cleanfiles	81.51	92.4	85.64	91.04	94.4
(**b**)					
Meta Classifier-AdaBoost					
	NB	SVM	IB_1	DT	RF
W.Avg	96.51	98.16	98.2	98.86	97.73
Cleanfiles	95.2	92.81	92.74	96.13	95.8

2.2. Linux Format Aware Analysis

As such, most endpoints have many PCs in the Windows environment, and many Windows-based malware studies have continued. Recently, in the field of static analysis of malicious code, research has been conducted to analyze and classify executable and linkable format (ELF) files in Linux [19–21] as well as Windows. Bai et al. [22] proposed a new malware detection method through mining system calls in the symbol table of Linux executables. The experiment collected 756 positive ELF executables and 763 malware data, and developed an ELF parser, feature extraction, classifier training and malware detector. This study employed four classification algorithms (J48, random forest, AdboostM1 (J48) and IBk) to train the classifier. The experimental results are presented in Table 6. The ROC curves for these methods are shown in Figure 2.

Table 6. Experimental results for four classification algorithms.

Classification Algorithms	TPR (%)	FPR (%)	Accuracy (%)	AUC
J48	99.7	4.5	97.7	0.987
Random Forest	99.4	3.1	98.2	0.985
AdboostM1(J48)	99.2	2.2	98.5	0.993
IBK	99.7	2.5	98.6	0.993

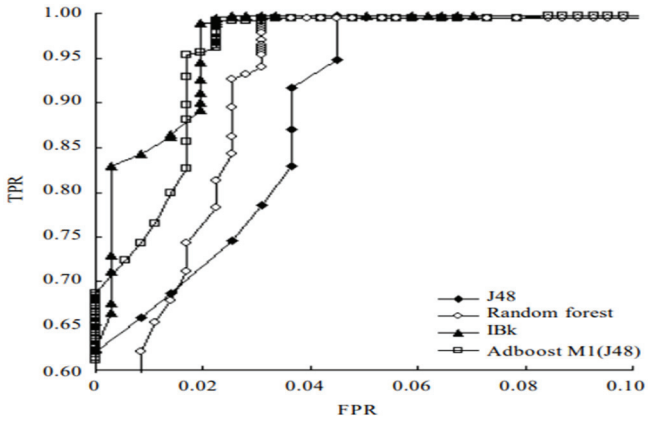


Figure 2. ROC curves for four classification algorithms.

Jeon et al. [23] extracted the structural information based on the same file format as the ELF file without using binary raw data and calculate the information gain to reduce the number of structural data features of the ELF file. The experiment was conducted by selecting the top 38% 147 feature sets. Table 7 shows some of the top 147 features, about 38% through information gain.

However, an analysis methodology based on structural features of a file is inherently vulnerable to obfuscation techniques such as packing. In addition, there is a disadvantage that the analysis system cannot be classified if the file format of the analysis and the file format of the malicious code does not match. This can be a very critical issue when looking at current security research trends. For example, in order to cope with spear phishing attacks that distribute malicious code by disguising official documents, the document file format should be further researched by a malware analysis expert. Document file formats vary widely, including pdf, doc, and xlsx, and it is practically difficult for a malware analysis expert to analyze all these document formats. In addition, considering the nature of the Linux operation system in various architectures, the technology based on ELF file formats is insufficient to current endpoint security issues.

Table 7. Selected ELF features after calculating information gain.

Feature Selection Sources	Selected Features by Calculating Information Gain (Top 38%)
ELF header	Identification, MachineType, ProgramHeaderOffset, SectionHeaderOffset, Flags, HeaderSize, SizeProgramHeader, EntriesProgram, SizeSectionHeader EntriesSection, StringTableIndex
Section headers	.text_type .text_size .text_alignment .bss_type .bss_size .bss_alignment .comment_size .comment_entsize .dynamic_table_index_link .dynamic_alignment etc.
Program headers	-
Symbols section	STB_LOCAL STT_OBJECT_STB_GLOBAL STT_OBJECT_STB_WEAK STT_NOTYPE_STB_WEAK STT_FUNC_STB_GLOBAL STT_NOTYPE_STB_GLOBAL etc.
Dynamic section & dynamic symbol section	dynamic_s_c s_STT_FUNC_STB_LOCAL s_STT_OBJECT_STB_GLOBAL DYNRELAENT DYNFINI DYNFINI_ARRAY DYNNULL etc.
Relocation sections	R_386_GLOB_DAT R_386_JUMP_SLOT etc.
Global offset table	GOT_SIZE
Hash table	HASH_SIZE

2.3. Binary Format Aware Analysis

On the other hand, due to these problems, studies using binary data are being conducted [24–27]. However, binary data is more difficult to analyze than structural methodology because it is difficult for humans to interpret it intuitively. Jain's [28] and Kwon's [29] studies are typical of research conducted on this problem. Jain has proposed a methodology for classifying malware based on algorithms such as naïve Bayes, instance-based learner, and extracting features into N-grams of binary units. In particular, this methodology is very heavy from the system point of view because substrings are divided by bytes. In order to reduce the exponentially increasing feature space, the method of reducing feature space with classwise document frequency is adopted. Kwon sees the drawback of the N-gram methodology in using the existing full binary data in this research trend. Therefore, in connection with the structural features of the file formats of the previous studies, as a methodology of applying N-gram only to data in a specific area, the analysis speed of the system is greatly improved, as shown in Table 8.

Most malicious codes are produced by packing or encryption to prevent reverse engineering analysis. Accordingly, a methodology for detecting malware through machine learning by extracting all character strings in a PE file and applying policies for each feature has been studied [30–32]. Figures 3 and 4 show the results of character string distribution analysis in normal files and malware [33]. In the case of normal files, the count is mostly concentrated below 200, whereas in the case of malicious files, it is concentrated above 200. In addition, there is a study that analyzes malware through binary images. Su's research [34] proposes a new lightweight approach to detecting DDos malware in IoT environments. The main course of action is to convert to binary images to extract one-channel gray scale images to classify malware and use lightweight neural networks to classify families. Some examples

of malware and benign-ware images are shown in Figures 5 and 6. By comparison, the structural difference between malware and goodware images can be identified. The experimental results show that the proposed system shows 94% accuracy for the classification of goodware and DdoS malware and 81.8% accuracy for the major malware classification. However, there are still many problems that need to be corrected for these research trends. To solve these problems in this paper we propose a platform-independent malware analysis technology.

Table 8. Kwon's Section N-gram and KNN-Ngram experiments.

Category		Filename	Ngram		.text Section Ngram	
			Similarity	Execution Time	Similarity	Execution Time
Trojan Malware	Boxed Family	Boxed.a	0.882	19.89	0.824	6.04
		Boxed.b	0.824	11.81	0.941	5.73
		Boxed.i	0.706	45.56	0.706	13.22
		Boxed.m	0.706	46.69	0.824	15.41
		Boxed.u	0.882	45.03	1	29.82
	Delf Family	Delf.af	0.471	12.43	0.353	9.33
		Delf.b	0.293	2.44	0.118	2.08
		Delf.c	0.353	1667	0.294	1280
Benign		notepad.exe	0.353	558.26	0.294	7.49
		ipconfig	0.471	3.37	0.294	2.70
		calc.exe	0.412	2116.95	0.294	688.66
		iexplorer.exe	0.176	3349.36	0.412	2018

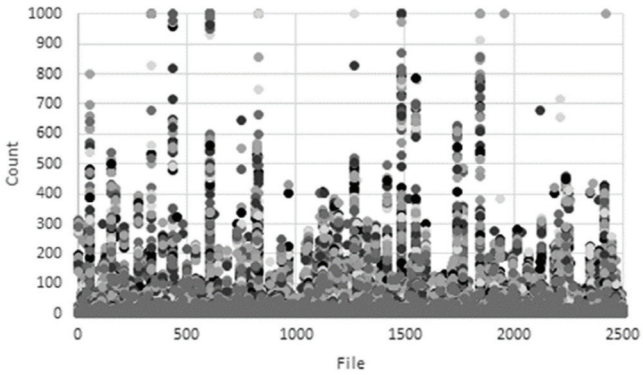


Figure 3. String-based distribution analysis (benign).

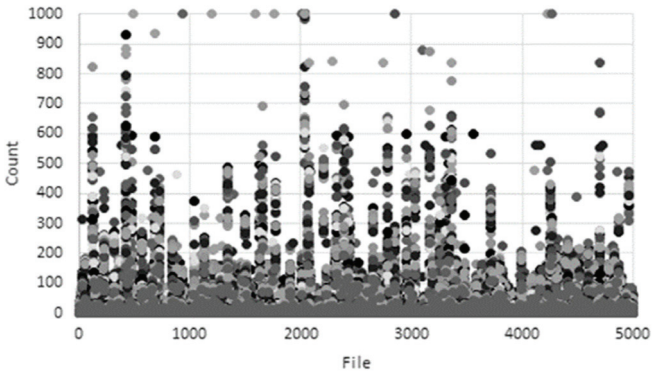


Figure 4. String-based distribution analysis (malware).

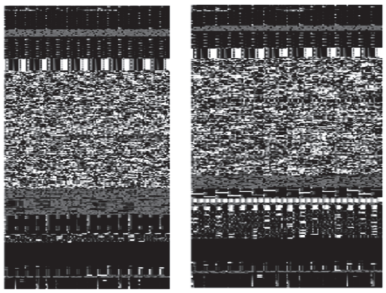


Figure 5. Image of goodware.

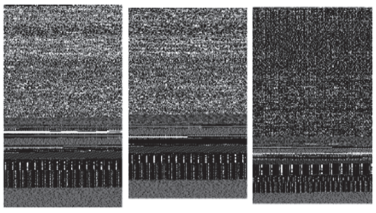


Figure 6. Linux malware image examples.

3. Proposed Model

3.1. System Overview

We propose a platform-independent malware analysis technology to detect malware that is entering the endpoint. Platform-independent malware analysis technology is a malware analysis technology that can be commonly applied to binary type data regardless of architecture/operating system type. This technology analyzes strings from binary data and classifies malware based on the results of the analysis. On the other hand, undetected malware, which inevitably occurs in AI-based malware analysis technology, can be responded to by security policies such as manual analysis or monitoring by experts. The overall system configuration is shown in Figure 7. Step 1 extracts strings from unknown binary data and classifies the malware by expressing it as a vector. Step 2 can improve the model by constructing a continuous security policy such as manual analysis or monitoring based on the results from Step 1. Finally, it is allowed to enter the endpoint only when the binary to be analyzed is determined to be benign.

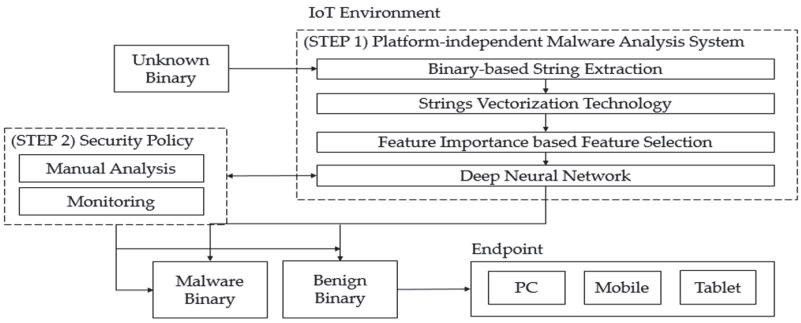


Figure 7. Platform-independent malware analysis overall system.

3.2. Platform-Independent Malware Analysis

3.2.1. Binary-Based Strings Analysis

All computer files are in binary format. Binary files contain data encoded in binary format for computer storage and processing purposes. Many binary file formats contain parts that can be interpreted as strings. However, most of them are obfuscated or packed in order to make static

analysis difficult when manufacturing malware. Obfuscation and packing are tasks that make it difficult to read code and file structured formats written in programming languages. Encryption or compressing the executable file makes it difficult to analyze the source code. Malware anaylsts say that it is difficult to analyze malware due to obfuscation and packing, and many studies on this topic have been conducted [35–40]. Based on this, since malware has a binary format and includes a part that can be interpreted, strings from binary format can be used to analyze the malware. However, compared to benign files, the malware has a lot of noise such as special characters and unnecessary characters. To be platfrom independent, we have to do static analysis and extract features from binary files, so we are not free from obfuscation issues. Strings to be extracted from the binary files are typically DLL and API names, library function names supported by programming languages, and PE or ELF file formats. To reduce this effect, we analyzed by setting the length of strings to be extracted. As a result of analysis, the extracted strings contained a lot of noise, and most of them did not exceed 5 in length. Therefore, in this paper, only strings with a string length of 5 or more were extracted. Figure 8 show strings sample extracted from binary data.

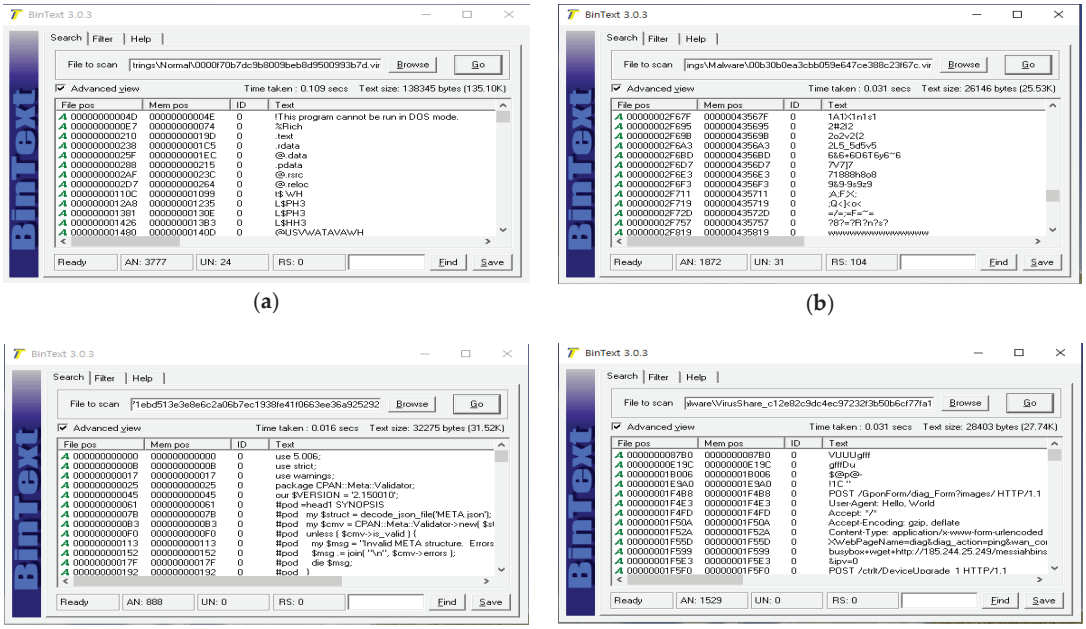


Figure 8. String analysis example in binary data: (**a**) Windows benign file analysis; (**b**) Windows malware file analysis; (**c**) Linux benign file analysis; (**d**) Linux malware file analysis.

3.2.2. Count-Based Strings Vectorization Technology

Since the number and size of extracted strings are different for each file, it should be expressed as a vector of fixed size. In order to classify the malware by applying DNN, a vector value of a fixed size is required, therefore, this paper proposes a count-based strings vectorization technique that can convert each string into a fixed-size vector. The advantage of this technique is that we can set the number of features we want to create. In other words, if you set V, the vector size, you can create as many features as V. If the V value is large, the amount of computation increases, which degrades model performance. If the V value is small, model training does not work properly. When extracting strings from Linux binary files, an average of 1800 were exteacted. Therefore, in this paper, experiments were conducted with various V values below 1800. As a result of the experiment, when the vector size was set to 1000, the proposed model learned Linux/Windows binary files well. The function used is shown in Figure 9.

The extracted strings basically have a byte format. Hash algorithms are used to reflect the unique characteristics of strings and to represent them as numbers. The string is expressed as a number and is counted through a modular operation with the vector size you want to generate.

Function1 – Convert strings to vectors Function

Description. This function converts the strings of a file to fixed size, V.

SET V : Fixed Vector Size
SET X : Array of Extracted Byte String

```
1. Array Vector[V]=0
2. FOR string In X :
3.     Point=HASH(string) Mod V
4.     Vector[Point]=Vector[Point]+1      # Count Operation
5. Return Vector
```

Figure 9. Strings vectorization pseudo-code.

3.2.3. Feature Importance Based Feature Selection

We have tried to increase the accuracy of the model by identifying related features in the dataset and removing unrelated or less important features. Using the feature selection method reduces overfitting and reduces the training time. We tried to use the wrapper method to select features, but we found that the more datasets, the longer it took to select features. In this paper, features were selected using feature importance. We calculated the importance using the extratreesclassifier. Since the extratreesclassifier operates based on a tree, information gain obtained from nodes can be obtained, so it is possible to compare which independent variables are important while comparing the average of the information gains obtained by each independent variable. Table 9 shows the number of selected features when the importance of the feature is greater than the set threshold using the features used in the experiment. It is 94.13% when verified with 1000 features, and 94% when verified with 2000 features. Therefore, in this paper, as a result of verification using each selected feature, good results were obtained when using the first 1000 features.

Table 9. Number of selected features using feature importance.

Threshold	Features Count	Features Count	Features Count
Default	1000	1500	2000
Feature_Importance > 0	730	877	955
Feature_Importance > 1	473	536	512
Feature_Importance > 2	340	328	321

3.2.4. Deep Neural Network

We classify malware using a deep neural network (DNN) [41,42]. The learning model consists of two hidden layers, excluding the input layer and the output layer. The number of nodes in the hidden layer is 10. To improve the model, we used the gradient-based optimization algorithm Adam. This method is simple to implement, highly computationally efficient, has little memory requirements, is not affected by the diagonal sizing of the gradient, and is suitable for large issues in terms of data and parameter. Also, a function called ReLU was used instead of sigmoid to activate the hidden layer. ReLU is a function that returns 0 if a value less than 0 is found, returns the value if it is greater than 0. It is different from sigmoid, which returns 1 if it is greater than 0. Therefore, ReLU was applied to the inner hidden layer, and the sigmoid function was used only for the last output layer.

4. Experiments

4.1. Dataset

Proposed techniques verified the performance and results by using public datasets and self-collected datasets. Platform-independent malware analysis was conducted on the major architectures of Windows and Linux binaries. The dataset used in the Windows binary experiments were malicious files and benign files published by the KISA Data Challenge in 2019 [43], and consist of a total of 40,000 data. The Windows binary learning dataset used 18,000 benign and 12,000 malware examples,

and the test data used 5000 malware and 5000 benign ones. On the other hand, the dataset used for the Linux binary experiments consists of 10,000 malware from Virus-Share [44] and 10,000 system benign files collected by the Linux architecture. The Linux binary learning dataset used 8000 malware and 8000 benign examples. The test data were tested using 2000 malware and 2000 benign samples.

4.2. Platform-Independent Malware Analysis Experiment Results

4.2.1. Windows Executable (PE) File Analysis

We analyzed binary strings in Windows datasets and set the vector size to 1000 to apply the count-based strings vectorization technology. In addition, malware was classified using the DNN algorithm. Figure 10 shows the distribution of two most important features in the malware strings and benign strings by reducing the vector size from 1000 to 100 with principal component analysis (PCA) for visualization. PCA is the most representative dimension reduction algorithm. PCA works by first obtaining hyperplane closest to the data and then projecting the data onto the hyperplane. Figure 11 shows the top 10 features in feature importance. The results for binary classification using the DNN model are shown in Table 10. Figure 12 shows the ROC curve to evaluate the performance of the model.

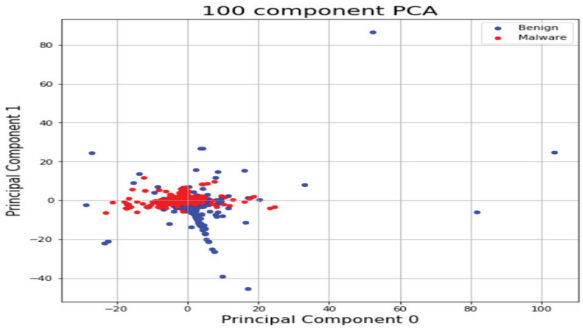


Figure 10. Feature vector distribution of Windows binary.

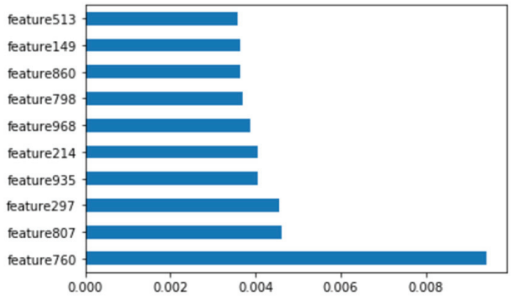


Figure 11. Top 10 features of Windows binary.

Table 10. Classification Performance of Windows Binary.

Classification Performance Metrics	Value
Accuracy	91.77%
Precision	89.93%
Recall	94.8%
F1-score	92.01%

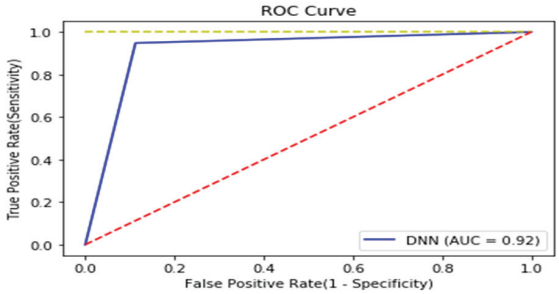


Figure 12. ROC curve of Windows binary.

4.2.2. Linux Executable (ELF) File Analysis

As with the Windows analysis, we analyzed binary strings in Linux datasets and set the vector size to 1000 to apply the count-based strings vectorization technology. In addition, malware was classified using the DNN algorithm. Figure 13 shows the distribution of two most important features in the malware strings and benign strings by reducing the vector size from 1000 to 100 with PCA for visualization. PCA is the most representative dimension reduction algorithm. PCA works by first obtaining hyperplane closest to the data and then projecting the data onto the hyperplane. Figure 14 shows the top 10 features in feature importance. The results for binary classification using the DNN model are shown in Table 11. Figure 15 shows the ROC curve to evaluate the performance of the model.

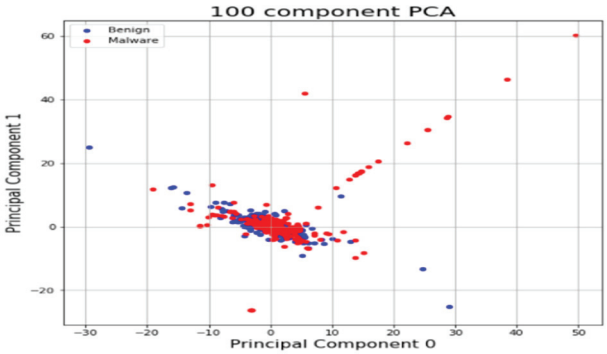


Figure 13. Feature vector distribution of Linux binary.

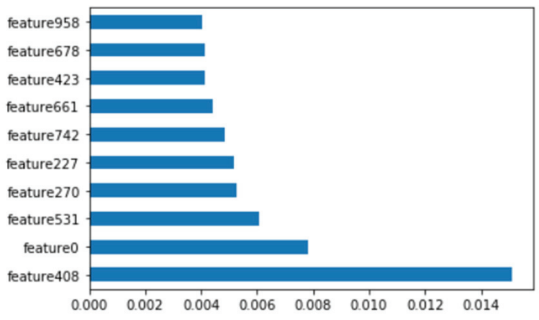


Figure 14. Top 10 features of Linux binary.

Table 11. Classification Performance of Linux Binary.

Classification Performance Metrics	Value
Accuracy	97.65%
Precision	97.46%
Recall	97.85%
F1-score	97.65%

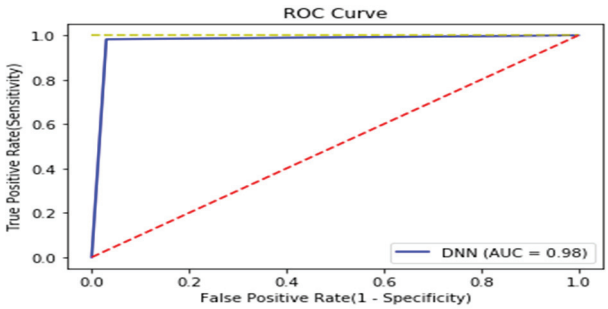


Figure 15. ROC curve of Linux binary.

4.2.3. Windows and Linux Executable File Analysis

This section shows the results of platform-independent endpoint malware analysis mentioned in Section 1.3. One code can analyze malware regardless of where the operating system is Windows and Linux. The dataset is a combination of the Windows and Linux datasets mentioned in Section 4.1. Figure 16 shows the distribution of two most important features in the malware strings and benign strings by reducing the vector size from 1000 to 100 with PCA for visualization. Figure 17 shows the top 10 features in feature importance. The results for binary classification using the DNN model are shown in Table 12. Figure 18 shows a confusion matrix of examples classified as malware and benign on Windows and malware and benign on Linux.

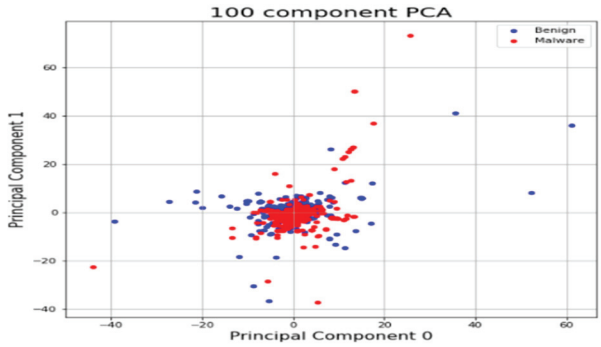


Figure 16. Feature vector distribution of Windows and Linux binary.

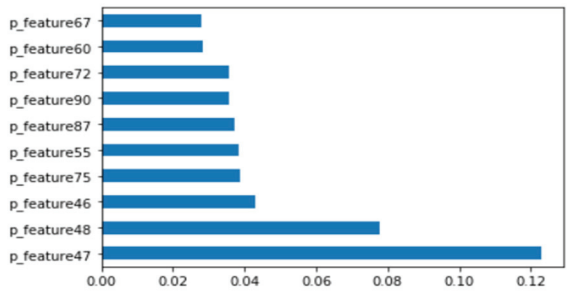


Figure 17. Top 10 features of Windows and Linux Binary.

Table 12. Classification Performance of Windows and Linux binary.

Classification Performance Metrics	Value
Accuracy	94.13%
Precision	94.89%
Recall	93.28%
F1-score	94.08%

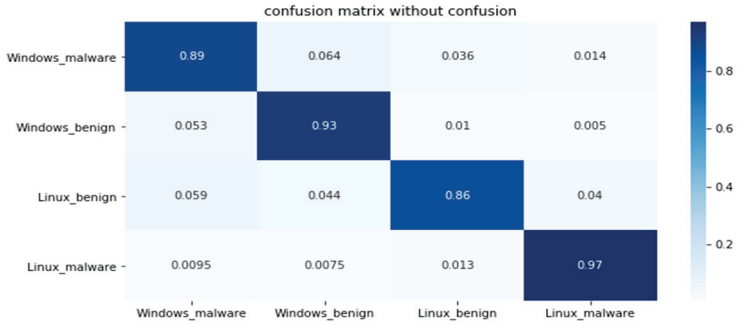


Figure 18. Confusion matrix of Windows and Linux binary.

5. Conclusions

Various IoT devices are emerging due to the influence of smartphones and tablets as well as PCs, and most of them are based on Linux/embedded environments. However, threats from IoT malware are increasing recently, and malwares perform a wide range of attacks against users using various platforms. Research into Linux-based malware analysis continues, but there is a lack of technology to identify Linux malware variants used in IoT/embedded environments. In addition, since Linux malware supports various architectures, it is difficult to identify the architecture in which the Linux malware is used during malware analysis. If the architecture environment used in the malware is identified, the automatic analysis technology for each architecture can operate. Therefore, in order to counter malware threats, platform-independent security technologies such as operating systems and architectures are required. The platform-independent malware analysis proposed in this paper is a malware analysis technique that can be commonly applied to binary data regardless of the architecture/operating system type. It is applicable to the 5G/IoT environment. The proposed technique used public datasets and self-collected datasets to validate the performance and results. As a result of evaluating the proposed technology with public and self-collected datasets, the accuracy was 94% for Windows and Linux malware. The system is an effective and sustainable model that can apply separate security policies, such as expert analysis, to complement each other's inherent technology shortcomings. The system is an effective and sustainable model that can cover one of the inherent weaknesses of technology by applying separate security policies, such as expert analysis, and also works for malware on other platforms. In the future, we will continue to verify and improve the proposed system in numerous malware environments to ensure the continuous operation and practicality of platform-independent malware analysis systems. In addition, we will study technologies that can identify the architecture when analyzing Linux malware. If the architecture used by malware is identifiable, we expect that automated analysis technology will advance significantly.

Author Contributions: Conceptualization, J.H., J.K. and T.L.; Data curation, C.H.; Formal analysis, J.K.; Funding acquisition, J.K.; Investigation, J.H., J.K. and T.L.; Methodology, C.H.; Project administration, C.H.; Software, C.H.; Supervision, T.L.; Validation, C.H.; Visualization, C.H.; Writing—original draft, J.H.; Writing—review & editing, C.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2017R1E1A1A01075110).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Boivin, A. *Defense Against the Real Threat of AI-Based Malware*; Utica College: Utica, NY, USA, 2018.
2. Zhang, L. Automated Feature Extraction and Artificial Intelligence (ai) Based Detection and Classification of Malware. U.S. Patent Application No 16/051,138, 6 February 2020.
3. Tirumala, S.S.; Valluri, M.R.; Nanadigam, D. Evaluation of Feature and Signature based Training Approaches for Malware Classification using Autoencoders. In Proceedings of the 2020 International Conference on COMmunication Systems & NETworkS (COMSNETS), Bengaluru, India, 7–11 January 2020.

4. Kaloudi, N.; Li, J. The AI-Based Cyber Threat Landscape: A Survey. *ACM Comput. Surveys (CSUR)* **2020**, *53*, 1–34. [CrossRef]
5. Juniper. Trusted Mobility Index, Juniper. 2012. Available online: http://www.juniper.net/us/en/local/pdf/additional-resources/7100155-en.pdf (accessed on 27 February 2013).
6. Eclipse, Key Trends from the IOT Developer Survey 2018. Available online: https://blog.benjamin-cabe.com/2018/04/17/key-trends-iot-developer-survey-2018 (accessed on 3 April 2019).
7. Stats, StatCounter Global. Destop Operating System Market Share Worldwide. 2017. Available online: http://gs.statcounter.com/os-market-share/desktop/worldwide (accessed on 24 May 2018).
8. Cozzi, E.; Graziano, M.; Fratantonio, Y.; Balzarotti, D. Understanding Linux Malware. In Proceedings of the IEEE Symposium on Security and Privacy, San Francisco, CA, USA, 20–24 May 2018; pp. 161–175.
9. Costin, A.; Zaddach, J.; Francillon, A.; Balzarotti, D. A large-scale analysis of the security of embedded firmwares. In Proceedings of the 23rd {USENIX} Security Symposium ({USENIX} Security 14), San Diego, CA, USA, 20–22 August 2014; pp. 95–110.
10. Zhang, Z.K.; Cho, M.C.Y.; Wang, C.W.; Hsu, C.W.; Chen, C.K.; Shieh, S. IoT security: Ongoing challenges and research opportunities. In Proceedings of the IEEE 7th International Conference on Service-Oriented Computing and Applications, Matsue, Japan, 17–19 November 2014.
11. Fossmint. Available online: https://www.fossmint.com/operating-systems-for-the-internet-of-things/ (accessed on 5 April 2020).
12. Vyas, D.A.; Bhatt, D.; Jha, D. IoT: Trends, challenges and future scope. *Int. J. Comput. Sci. Commun.* **2016**, *7*, 186–197.
13. Markel, Z.; Bilzor, M. Building a machine learning classifier for malware detection. In Proceedings of the 2014 Second Workshop on Anti-malware Testing Research (WATeR), Canterbury, UK, 23 October 2014; pp. 1–4.
14. Bayer, U.; Moser, A.; Kruegel, C.; Kirda, E. Dynamic analysis of malicious code. *J. Comput. Virol.* **2006**, *2*, 67–77. [CrossRef]
15. Xu, J.Y.; Sung, A.H.; Chavez, P.; Mukkamala, S. Polymorphic malicious executable scanner by API sequence analysis. In Proceedings of the Fourth International Conference on Hybrid Intelligent Systems (HIS'04), Kitakyushu, Japan, 5–8 December 2004; pp. 378–383.
16. Ye, Y.; Wang, D.; Li, T.; Ye, D.; Jiang, Q. An intelligent PE-malware detection system based on association mining. *J. Comput. Virol.* **2018**, *4*, 323–334. [CrossRef]
17. Shankarapani, M.; Kancherla, K.; Ramammoorthy, S.; Movva, R.; Mukkamala, S. Kernel machines for malware classification and similarity analysis. In Proceedings of the 2010 International Joint Conference on Neural Networks (IJCNN), Barcelona, Spain, 18–23 July 2010; pp. 1–6.
18. Islam, R.; Tian, R.; Batten, L.; Versteeg, S. Classification of Malware Based on String and Function Feature Selection. In Proceedings of the Second Cybercrime and Trustworthy Computing Workshop (CTC), Ballarat, VIC, Australia, 19–20 July 2010; pp. 9–17.
19. Shahzad, F.; Farooq, M. Elf-miner: Using structural knowledge and data mining methods to detect new (Linux) malicious executables. *Knowle. Inf. Syst.* **2012**, *30*, 589–612. [CrossRef]
20. Hernández, A. *Elf Parsing Bugs by Example with Melkor Fuzzer*; IOActive Inc.: Seattle, WA, USA, 2014.
21. Ermakov, M.K.; Vartanov, S.P. Dynamic analysis of ARM ELF shared libraries using static binary instrumentation. *Proc. Institute Syst. Program. RAS* **2015**, *27*, 5–24. [CrossRef]
22. Bai, J.; Yang, Y.; Mu, S.; Ma, Y. Malware Detection Through Mining Symbol Table of Linux Executables. *Inf. Technol. J.* **2013**, *12*, 380–384. [CrossRef]
23. Deok-Jo, J.; Dong-Gue, P. Real-time Linux Malware Detection Using Machine Learning. *J. Korean Institute Inf. Technol.* **2019**, *17*, 111–122.
24. Emary, E.; Zawbaa, H.M.; Hassanien, A.E. Binary grey wolf optimization approaches for feature selection. *Neurocomputing* **2016**, *172*, 371–381. [CrossRef]
25. Huang, Y.; Lin, Z. Binary multidimensional scaling for hashing. *IEEE Trans. Image Process.* **2017**, *27*, 406–418. [CrossRef] [PubMed]
26. Kolosnjaji, B.; Demontis, A.; Biggio, B.; Maiorca, D.; Giacinto, G.; Eckert, C.; Roli, F. Adversarial malware binaries: Evading deep learning for malware detection in executables. In Proceedings of the 2018 26th European Signal Processing Conference (EUSIPCO), Rome, Italy, 3–7 September 2018; pp. 533–537.
27. Raff, E.; Zak, R.; Cox, R.; Sylvester, J.; Yacci, P.; Ward, R.; Nicholas, C. An investigation of byte n-gram features for malware classification. *J. Comput. Virol. Hacking Tech.* **2018**, *14*, 1–20. [CrossRef]

28. Jain, S.; Meena, Y.K. Byte level n-gram analysis for malware detection. In Proceedings of the International Conference on Information Processing, Bangalore, India, 5–7 August 2011; pp. 51–59.
29. Kwon, H.J.; Kim, S.W.; Im, E.G. An Malware classification system using multi n-gram. *J. Secur. Eng.* **2012**, *9*, 231–512.
30. Shrestha, P.; Maharjan, S.; de la Rosa, G.R.; Sprague, A.; Solorio, T.; Warner, G. Using String Information for Malware Family Identification. In *Ibero-American Conference on Artificial Intelligence*; Springer: Cham, Germany, 2014; pp. 686–697.
31. Yu, F.; Bultan, T.; Ibarra, O.H. Symbolic string verification: Combining string analysis and size analysis. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 322–336.
32. Yu, F.; Alkhalaf, M.; Bultan, T. Stranger: An automata-based string analysis tool for php. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 154–157.
33. Sunbin, H.; Hogyeong, K.; Junho, H.; Taejin, L. A Study on Two-dimensional Array-based Technology to Identify Obfuscated Malware. *J. Inf. Sci.* **2018**, *45*, 769–777.
34. Su, J.; Vasconcellos, D.V.; Prasad, S.; Sgandurra, D.; Feng, Y.; Sakurai, K. Lightweight Classification of IoT Malware Based on Image Recognition. In Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Tokyo, Japan, 23–27 July 2018; pp. 664–669.
35. Brunton, F.; Nissenbaum, H. Vernacular resistance to data collection and analysis: A political theory of obfuscation. *First Monday* **2011**, *16*. [CrossRef]
36. Gaudesi, M.; Marcelli, A.; Sanchez, E.; Squillero, G.; Tonda, A. Malware obfuscation through evolutionary packers. In Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation, New York, NY, USA, 11–15 July 2015; pp. 757–758. [CrossRef]
37. Schrittwieser, S.; Katzenbeisser, S.; Kinder, J.; Merzdovnik, G.; Weippl, E. Protecting software through obfuscation: Can it keep pace with progress in code analysis? *ACM Computing Surveys (CSUR)* **2016**, *49*, 1–37. [CrossRef]
38. Hoffmann, J.; Rytilahti, T.; Maiorca, D.; Winandy, M.; Giacinto, G.; Holz, T. Evaluating analysis tools for android apps: Status quo and robustness against obfuscation. In Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, New York, NY, USA, 9–11 March 2016; pp. 139–141. [CrossRef]
39. Sharif, M.I.; Lanzi, A.; Giffin, J.T.; Lee, W. Impeding Malware Analysis Using Conditional Code Obfuscation. In Proceedings of the 15th Annual Network and Distributed System Security Symposium, Atlanta, GA, USA, 8 February 2008.
40. Xian, G.-M.; Zeng, B.-Q. An intelligent fault diagnosis method based on wavelet packer analysis and hybrid support vector machines. *Expert Syst. Appl.* **2009**, *36*, 12131–12136. [CrossRef]
41. KISA, Security R&D Dataset. Available online: https://www.kisis.or.kr/kisis/subIndex/283.do (accessed on 28 August 2019).
42. VirusShare, VirusShare_ELF_20190212. Available online: https://virusshare.com/ (accessed on 28 August 2019).
43. Tobiyama, S.; Yamaguchi, Y.; Shimada, H.; Ikuse, T.; Yagi, T. Malware detection with deep neural network using process behavior. In Proceedings of the 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), Atlanta, GA, USA, 10–14 June 2016; pp. 577–582.
44. Xu, L.; Zhang, D.; Jayasena, N.; Cavazos, J. Hadm: Hybrid analysis for detection of malware. In *Proceedings of SAI Intelligent Systems Conference*; Springer: Cham, Germany, 2016; pp. 702–724.