IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received May 9, 2021, accepted June 1, 2021, date of publication June 7, 2021, date of current version June 16, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3087316

Intelligent Behavior-Based Malware Detection System on Cloud Computing Environment

ÖMER ASLAN[1], MERVE OZKAN-OKAY[2], AND DEEPTI GUPTA[3]
[1]Department of Computer Engineering, University of Siirt, 56100 Siirt, Turkey
[2]Department of Computer Engineering, Ankara University, 06830 Ankara, Turkey
[3]Department of Computer Science, The University of Texas at San Antonio, San Antonio, TX 78249, USA

Corresponding author: Ömer Aslan (omer.aslan@siirt.edu.tr)

ABSTRACT These days, cloud computing is one of the most promising technologies to store information and provide services online efficiently. Using this rapidly developing technology to protect computer-based systems from cyber-related attacks can bring many advantages over traditional protection schemes. The protected assets can be any computer-based systems such as cyber-physical systems (CPS), critical systems, desktop and laptop computers, mobile devices, and Internet of Things (IoT). Malicious software (malware) is any software which targets the computer-based system to launch cyber-attacks to threaten the integrity, confidentiality and availability of the data. To detect the massively growing malware attacks surface, we propose an intelligent behavior-based detection system in the cloud environment. The proposed system first creates a malware dataset on different virtual machines which identify distinctive features efficiently. Then, selected features are given to the learning-based and rule-based detection agents to separate malware from benign samples. Totally, 10,000 program samples have been analyzed to evaluate the performance of the proposed system. The proposed system can detect both known and unknown malware efficiently with high detection and accuracy rate. Besides, the proposed method results have outperformed the leading methods' results in the literature. Our evaluation results show that the proposed algorithms along with machine learning (ML) classifiers achieve 99.8% detection rate, 0.4% false positive rate, and 99.7% accuracy. Our proposed system and algorithms may assist those who would like to develop a novel malware detection system in the cloud environment.

INDEX TERMS Cloud computing, virtualization, malware detection, behavioral detection, rule-based detection.

I. INTRODUCTION

Nowadays, there is a tremendous increase in both the amount and severity of cyber-related attacks. In general, different malware variants are the main reason for cyber-attacks. Malware is any kind of software which is designed to exploit computer and network systems' vulnerabilities to perform malicious activities and gain financial benefits. Virus, worm, Trojan, backdoor, rootkits, and ransomware are well-known examples of malware. Each malicious code variant and its family are designed for different purposes. While some malware variants steal sensitive data, others initiate distributed denial of service (DDoS) attacks and allow remote code execution [1]. During sophisticated attacks, more than one malware type and family are used.

The associate editor coordinating the review of this manuscript and approving it for publication was Tyson Brooks.

Over the years, the number of malware samples have been increasing rapidly. According to business and scientific reports, around 1 million malicious software variants are generated every day. Most of these malware variants are evolving versions of existing malware. Adding new devices to the computer networks every day such as IoT devices, the amount of applications created in a short period of time, and the amount of data created every day in social media also increase the malware-related attacks in the virtual world. On the other hand, new malware variants are using sophisticated concealing techniques such as obfuscation and packing to hide from detection systems. This makes it almost impossible to identify and classify complex malware with a conventional detection method.

The sophistication of malware attacks, spread methods and economic damage to the world economy have hit the peak recently. According to the researchers, cyber-attacks

TABLE 1. The evolution of malware-related attacks over the years.

Malware Related Attack	Year	Attack Spread Method	Result
Melissa Virus	1999	It used social engineering techniques to persuade users to click on the email attachment.	It caused billions of dollars in losses across many countries.
ILOVEYOU Worm	2000	It used social engineering to entice users to open the attachment.	It stole users' credentials and infected more than 45 million computer users.
MyDoom worm	2004	It spread by email using attention-grabbing subjects, such as errors, tests, etc.	It launched DDos attacks and allowed remote control.
Zeus Trojan	2007	Malicious emails in the form of spam and drive-by downloads.	It stole login details for social networks, bank and email accounts.
Stuxnet Worm	2010	Attack on the programmable logic unit by stealing source codes.	It took control of industrial processes.
Mirai Malware	2016	It exploited the vulnerability of IoT devices.	It launched DDos attacks.
WannaCry Ransomware	2017	It exploited Windows vulnerability.	It encrypted computer hard drives and affected 150 countries.
Emotet Trojan	2018	Malicious emails in the form of spam and phishing campaigns.	It stole information from individuals, like credit card details on banking systems.
MyFitnessPal	2018	By exploiting software vulnerability.	It affected 150 million users.
LockerGoga Ransomware	2019	Malicious emails, phishing scams and credentials theft.	It completely blocked victims' access to the system and caused millions of dollars in damage.
CovidLock Ransomware	2020	It exploited users' trust by providing statistical information about COVID-19.	It encrypted data on Android devices and denied data access.

cause trillion dollars damage to the world economy globally. The evolution of malware related attacks over the years is given in Table 1. It can be seen that back in the early days, viruses and worms were used to launch attacks, but over the years Trojans and ransomwares are mostly used. Attack spread methods that have been evolved, and damages that have been inflicted are changing over the years as well. Social engineering techniques which exploit user trust, software vulnerabilities, malicious emails, and phishing scams are used for attacks' spread methods. Most of the recent attacks steal information from individuals like credit card details on banking systems, encrypt computer data on hard drives to block victims' access to the system, and cause damage to millions of users around the globe.

Malware detection is the process of specifying whether a given program is malware or benign. There are a lot of different methods presented to detect malware which can be categorized as traditional and new detection approaches. Traditional approaches include signature-, heuristic-, behavior-, and model checking-based while new approaches include cloud-, deep learning-, and mobile devices-based detection [2].

As it is known, the signature-based detection approach performs well for known and different versions of the same malware, but it fails to detect unknown malware which has a completely different signature. Behavior-, heuristic-, and model checking-based approaches may detect a significant portion of the zero-day malware. However, they cannot detect new malware which uses advanced packing techniques. Although deep learning- and mobile devices-based detection approaches improve the detection rate (*DR*) for mobile devices to a certain degree, they fail to detect malware which seems completely different from the previous version [2].

Cloud-based malware detection approach brings several benefits over the other approaches. The cloud computing environment provides easy access, on-request storage, more computational power and considerably bigger databases while decreasing the cost. Multiple execution traces of the same malware have been collected by using different virtual machines (VMs) and servers [3]. Cloud environment improves the *DR* for personal computers, mobile and IoT devices. In addition, various detection algorithms can be implemented on different servers. Using several algorithms improves the detection performance while decreasing the false positive and negative rates.

In this paper, an intelligent behavior-based malware detection schema is proposed in the cloud computing environment. The cloud-based detection schema consists of two parts including feature extraction and detection phases. A client submits a suspicious file over the computer network and receives the analysis result from the server which shows whether the given suspicious file is malware or not. The suggested cloud-based system provides the following contributions:

- Suggested model creates a malware dataset with fewer features than known models do.
  - ➢ First, several system calls are mapped into relevant behaviors.
  - ➢ Second, relationships are determined among the behaviors.
  - ➢ Finally, features are extracted from behaviors which have semantic relationships between them.
- Learning-based detection engine is used to separate malware from benign.
- Rule-based detection engine is used to determine malware from benign as well.
- The proposed schema can detect both previously known and unknown malware.
- Proposed model detection and accuracy rate are measured higher than known models.

The rest of the paper is organized as follows: Section II explains the cloud computing environment. Related work

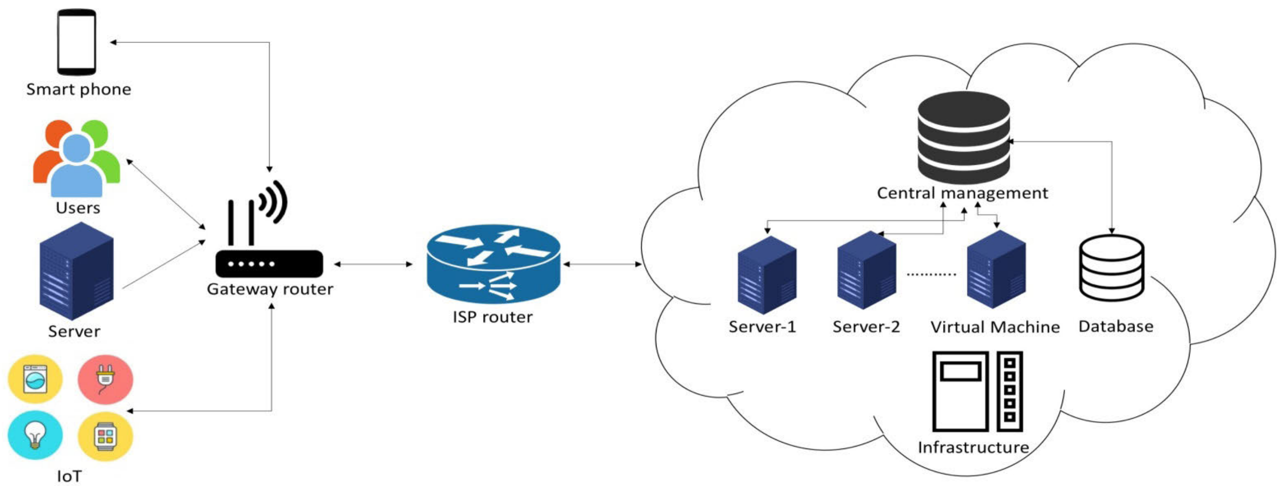


FIGURE 1. Cloud computing environment.

is summarized in section III. Proposed methodology and case study are defined in section IV and V. The results and discussion are presented in section VI. The limitations and future works of the proposed model are given in section VII. Finally, the conclusion is given in section VIII.

II. CLOUD COMPUTING

Cloud computing provides various computing services over the Internet [4]. By using the cloud, different users and businesses are storing their data remotely in the data centers instead of using their own local storage. This makes the data available from anywhere, anytime, and from any devices. Cloud environment provides data storage, servers, VMs, databases, networking, and software. Typical cloud computing environment and its infrastructure can be seen in Figure 1.

These days, cloud computing is used almost everywhere. Cloud computing brings several advantages over traditional kinds of information storage such as easy access, pay as you go, increase in speed, efficiency and performance, and decreasing cost. There are many cloud service providers including Amazon, Microsoft, Google, IBM, Rackspace, and Verizon.

Virtualization is the foundation of cloud computing. Without virtualization, cloud computing will be incomplete in many ways and will not be used as much as it is used today. Virtualization uses a hypervisor (Virtual machine monitor) to separate the operating system from the computer hardware. This allows us to use multiple operating systems that run on the same physical machine. Server virtualization on the cloud site brings many advantages including: less equipment, lower energy consumption, increase in server uptime, faster server provisioning, redundancy, and improvement in disaster recovery.

Cloud environment provides different types of services including infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS) [5]. For businesses' needs, one cloud service can have more advantages than others. There are also different kinds of cloud deployment models including public, private, community and hybrid clouds [6]. Public clouds are owned by third-party cloud providers and services, and available to the general public. Users can access the cloud services by using a web browser. In the public cloud, different organizations share the same infrastructure which may disclose sensitive data. Private clouds provide physical infrastructure and services for specific organizations. Infrastructure can be physically located in the company's data center or third-party's data center. Community clouds are created for exclusive use by a specific community. Hybrid clouds combine public and private clouds together. Hybrid clouds provide more deployment options, security and flexibility.

Cloud computing brings several advantages over traditional storing schema:

1. Users and organizations can store and back up their data in an efficient manner.
2. Regular users and organizations can access their data from any device, anywhere, and any time via browser or application.
3. Organizations can subscribe only for needed services.
4. Cloud environment provides cost-savings from small businesses to big organizations.
5. Cloud environment provides more storage space, computational power and considerably bigger databases.
6. It eliminates the need for onsite equipment, maintenance, and management issues.
7. It enables rapid response when increasing data volume requirements.
8. It reduces cost for physical resources, energy, and personnel training needs.

There are some issues which need to be addressed in the cloud computing environment:

1. Users lose control over their data.
2. Sensitive and top-secret data can be disclosed.

3. On public clouds the same physical resources are used for different organizations which also raises the security issues.
4. Data can be lost because of internal bugs, natural disasters and other reasons.
5. If the Internet is slow, it takes a lot of time to access the data.
6. Real time monitoring is not possible for all locations.

Using the cloud computing environment for malware detection brings many advantages. Cloud environment presents more computational power and much bigger databases [6]. Different methods and algorithms can be implemented in the cloud such as machine learning (ML), data mining, and deep learning. Multiple execution traces of the same malicious software can be collected. It enhances the detection performance for personal computers, mobile and IoT devices.

III. RELATED WORK

Cloud computing technology has been rapidly developing recently owing to some advantages, including simple accessibility, lower costs and scalability. Due to the innovation and convenience of cloud technology, the interest and use of cloud computing have increased among users as well as researchers. Cloud computing plays an important role in the protection of computer systems such as smart cyber-physical systems (CPSs) [7], IoT devices [8] and personal computers from several cyber-attacks, especially from malware attacks. In literature review, several studies are presented to detect malware in the cloud environment. Different malware detection approaches were analyzed based on the main idea, algorithms that are used, and feature extraction methods. Well-known cloud-based malware detection methods are summarized in Table 2. The common goal of all these studies is to identify malware by increasing *DR* while decreasing misclassification rates. When these studies are examined, it is seen that although each detection method has its own superiorities and performs better for particular datasets in the cloud, none of them could detect all malware.

Martignoni *et al.* [9] introduced a new framework to support dynamic behavior-based malware analysis based on cloud computing. The proposed framework is based on two assumptions. First, the security lab has no limit on available computing resources and can take advantage of hardware features. Second, end users' environments are more literal and nonhomogeneous than synthetic environments and are therefore more suitable for analyzing malware. They performed an empirical prototype to approve their ideas and integrated it into their existing behavior-based malware detection system. The evaluation results showed that the proposed framework enables security labs to advance the integrity of the analysis while performing a detailed analysis of the program's behavior without computational costs for end-users. On the other hand, the proposed framework increased the security issues and inclined to several detection and hijacking attacks. Solving security-related problems and applying a framework resistant to evasion attacks will improve framework performance.

Cha *et al.* proposed a new malware detection system named SplitScreen [10]. It is a distributed malware detection system that uses a supplemental screening step before the signature matching stage. SplitScreen's two-stage screening step is separated into client-server processes. The suggested method was implemented as an extension of ClamAV, which increases scanning throughput with more than 2x the signature set using half of the memory. As the authors mentioned that the acceleration and memory savings of SplitScreen improves when the number of signatures increases. The proposed method is scalable with a wide range of low end consumer and handheld devices. Since only one server is used on the cloud side, it would be better to optimize server efficiency and load some work on the client side.

Win *et al.* studied cyber-attacks targeting the virtualization infrastructure underlying cloud computing services [11]. They proposed a malware and rootkit detection system that defends guests from several attacks. The system was combined with Support Vector Machines (SVM) based external monitoring on the host, with system call monitoring and system call hashing in the guest kernel. The design of the proposed approach is to perform a system that detects the entity of attacks against guests in real time without the demand for a signature database. They indicated the efficiency of the proposed approach by appreciating it against well-known user-level malware and kernel-level rootkit attacks. According to the authors, the implemented solution eliminated the demand to use a signature database for malware classification.

Gupta *et al.* proposed a novel model for malware detection in the cloud [12]. The aim of this study is to detect malicious activities with some techniques and warn guest VMs about it. In this paper, DNA sequence detection process, the symbolic detection process and the behavioral detection process are combined. During the DNA sequence detection process, they extracted the DNA sequence from a file to detect malware. In the symbolic detection process, they clustered files according to file formats and used symbols to detect malware files. During the behavioral detection operation, they observed the behavior of the file and determined whether it was a malicious program using the Anubis sandbox. A prototype of the proposed approach (PMDM) is partially implemented on the Eucalyptus. According to the authors, PMDM is inexpensive, needs less runtime, and ensures well performance for large numbers of files compared to other known systems. However, this study can be improved further by using a bigger dataset.

Rakotondravony *et al.* categorized attacks in the IaaS cloud that can be analyzed using VMI-based mechanisms [13]. They focused on attacks that directly scramble VMs deployed in the IaaS cloud. The classification methodology takes into account the target, source and direction of attacks. They provided an overview of attacks where each actor could be threatened in the environment. They defined a

TABLE 2. Summary of cloud-based malware detection methods.

Paper	Proposed Method	Goal/Success	Year
Martignoni *et al.* [9]	Presented a new framework based on cloud computing for dynamic behavior-based analysis.	It provides security labs to enhance the accuracy of the analysis.	2009
Cha *et al.* [10]	Anti-malware system called SplitScreen.	It increases detection while decreasing memory usages.	2011
Win *et al.* [11]	A malware and rootkit detection system.	It removes the necessity of using a signature database.	2015
Gupta *et al.* [12]	A novel malware detection model on cloud architecture.	PMDM is inexpensive, takes less working time and presents well performance for large numbers of files.	2016
Rakotondravony *et al.* [13]	Attack classification in the IaaS cloud that can be examined using VMI-based mechanisms.	It lets distinct actors in a cloud scenario evaluate different malware attacks and design sufficient detection and mitigation mechanisms based on VMI.	2017
Sun *et al.* [14]	CloudEyes, which presents effective and confident security services for limited resource devices.	It is effective, practical, and saves time and data storage when detecting malware.	2017
Babu and Murali [15]	Improved and designed an intermediary malware protection in cloud environments.	It protects the cloud from malware transportations, and decreases time and cost.	2017
Xiao *et al.* [16]	Malware detection scheme with Q-learning.	It increases the accuracy, while reducing the latency.	2017
Abdelsalam *et al.* [17]	Malware detection approach in cloud infrastructure.	The 2-D CNN model achieves the 79% accuracy rate, and 3-D model notably enhances to 90% the accuracy rate.	2018
Mirza *et al.* [18]	An energy effective hosting model in the cloud environment.	It shows important energy efficiency with regard to CPU usage by the hosting model.	2018
Mirza *et al.* [19]	Cloud-based energy effective hosting model for an intelligent malware detection	It performs better than the conventional antiviruses.	2018
Shen *et al.* [20]	Malware detection system implemented by an intrusion detection system with cloud and fog computing.	It decreases delay of data traffic as well as data transfer overhead.	2018
Zhou and Yu [21]	A cloud-assisted model for malware detection and the dynamic system against malware propagation.	It can prevent the spreading of malicious codes obviously and efficiently and is convenient to the resource limited WMS.	2018
Yadav [22]	Consolidated WFCM-AANN malware detection technique.	It successfully determines the malicious software with high detection precision thereby outperforming existing classifiers.	2019
Indirapriyadarsini *et al.* [23]	Random and some other modeling like KNN, Logistic Regression (LR), etc.	It has come up with the unique solution by working with ML and cloud computing simultaneously to determine the legitimacy of the file.	2020
Deyannis *et al.* [24]	Cloud-based malware detection solution called TrustAV.	It can protect the transmission and processing of user data even in distrusted networks.	2020

common IaaS cloud scenario as a range of three different elements: cloud provider, external entity, and VMs. First, they summarized the distinct properties of attacks classified in the literature in respect to attack complexity, security effect, and suggested defense metrics. They then analyzed statistics on virtualization vulnerabilities misused by attacks, noticed them in public databases, and highlighted their evolution over time. Finally, they presented the economic impact of attacks on business processes. This study allowed several actors in a cloud scenario to evaluate different malware attacks and as a result design sufficient detection and mitigation mechanisms based on virtual machine introspection. Paper can be further enhanced by focusing not only on attacks involving direct VMs, but also on other types of attacks.

Sun *et al.* [14] explained a cloud-based malware detection system called CloudEyes. CloudEyes ensures effective security and data privacy for limited resource devices. Suspect bucket cross-filtering, a novel signature-based detection system for the cloud server, has been proposed based on reversible structure. It can provide retroactive and correct processing of malicious signature fragments. A scanning tool is applied to quickly define the file content suspicion with respect to the summary of the reversible sketch for the client. An interaction mechanism has been designed to protect the data privacy and decrease consumption of communication. The client transmits the coordinates of the suspicious file segments rather than the entire file content. They evaluated the performance of CloudEyes using both suspicious and normal traffic. According to the authors, the test results showed that CloudEyes is effective, practical and outperforms other existing systems in terms of time usage and consumption of communication. However, *DR* and accuracy can be further improved. In addition, some methods can be applied to reduce the data size to optimize storage and matching performances.

Babu and Murali designed a protection system against malware spreading in cloud environments [15]. This investigation presents several layered protections to address the problem and creates a two-layered epidemic model for preventing spread of malware from network-to-network. In the proposed system, they designed the malware detection system for various cloud servers using a middle monitoring server, allowing scanning, detection and removal of malware before transferring to cloud servers. According to the authors, this study secures malware transfers to the clouds and saves time and cost.

Xiao *et al.* analyzed the malware detection game based on cloud in which mobile devices upload the traces of their application to security servers over access points or base stations in dynamic networks [16]. Q-learned malware detection system was designed for a mobile device. The aim of this study is achieving the optimal payload transfer ratio without knowing the trace creation and radio bandwidth model of different mobile devices. They used the Dyna architecture to enhance performance and a post-decision learning method to speed up the reinforcement learning phase.

Abdelselam *et al.* [17] presented a malware detection method based upon Convolutional Neural Network in cloud computing environments. They used a standard 2-D CNN, training on data existing for each of the processes in a VM acquired through the hypervisor. They improved CNN classifier accuracy rate by using a new 3-D CNN, which considerably helps decrease mislabeled samples while training and data collecting. They performed experiments on collected data by working varied malware on VMs. The 2-D CNN model achieves the 79% accuracy rate, and 3-D model notably enhances to 90% the accuracy rate. This study could be improved with increasing the experiments scale by examining more malware binaries.

Mirza *et al.* [18] proposed a combination of ML techniques applied on large dataset. The paper mainly focused on two important goals including higher *DR* and low resource consumption. They extracted a group of features from the dataset including malicious and normal files, and implemented a SVM, boosting, and decision tree on the decision tree to obtain the highest possible detection rate. Boosting of the decision tree classifier showed a better performance in the assessment of CloudIntell. They also introduced a scalable cloud based architecture hosted on Amazon Web Services (AWS). They tested proposed methodology on different scenarios. According to the authors, their methodology produced high results with lowest energy consumption. Besides, implementing the boosting algorithms on a real-time platform is difficult and training the classifier with large amounts of data takes a lot of time and computation. In another study, Mirza *et al.* [19] suggested an energy efficient hosting model which consists of distinct components of Amazon's cloud services to improve a unique and scalable model. This research examined the set benchmarking numbers and known antiviruses for the cloud based hosting model. According to the paper, the proposed approach not only was successful for the hosted detection framework, but also performed optimally better than traditional antiviruses. However, the malware detection framework and hosting model can be improved further by integrating the intrusion detection mechanism to be assisted by the cloud based engine.

Shen *et al.* [20] explained a malware detection structure implemented by a cloud and fog computational intrusion detection system (IDS) to accomplish the IDS spreading problem in smart objects. There are three main contributions of the proposed study. First, they suggested an intrusion detection approach to detect malicious software in fog cloud based IoT networks. Second, they introduced a multistage privacy-preserved game which is based on confidentially leakage evaluation of smart objects to detect malware in IoT networks. Finally, they explained a framework to integrate the presented game into fog cloud based IoT networks using the right detection strategies. According to the authors, the proposed model fulfilled the large data processing requirement caused by the greatly increasing number of smart objects and the reduced data traffic latency as well as the data transfer overhead.

Zhou and Yu suggested a cloud assisted model for the dynamic differential game against malware spread and malware detection [21]. In the suggested model, first, a malware detection model based on SVM is created by sharing data on the security platform in the cloud. Second, the number of malware infected nodes that physically infect sensitive nodes is calculated according to attributes of wireless multimedia system (WMS). Finally, the transition of states between WMS devices is described by the changed epidemic model and Hamilton function has been presented to simplify the saddle point solution. Also, a target cost function and dynamic differential game has been sequentially derived for the Nash equilibrium between the WMS system and malware. According to the paper, obtained results demonstrated that the proposed algorithm is capable of suppressing the spread of malicious code clearly and efficiently and is suitable for resource-constrained WMS.

Yadav explained a unified WFCM-AANN malware detection approach to identify malware on the system [22]. The presented study consists of 2 modules, including classification and clustering. In the clustering module, the input data set is obtained in clusters by applying the WFCM (Weighted Fuzzy C-mean) algorithm. In the classification module, the centroid from the clusters is given to the discontinuous Auto-Associative Neural Network, which is applied to characterize whether information is intruded or not. The author claims that the proposed classifier successfully determines malware with high detection rate and therefore outperforms the existing classifiers.

Indirapriyadarshini *et al.* [23] proposed a machine learning-based detection technique on the cloud environment. They first used random modeling to get the worst log loss and then used some modelling such as KNN, LR etc. They then looked at the log loss of each algorithm and determined whether it was a perfect model. Finally, they deployed the ML model with the user interface on the cloud AWS. According to the authors, they had found a unique solution by working simultaneously with ML and cloud computing to determine the legitimacy of the file. However, this study can be enhanced by applying different data mining techniques for feature selection or by implementing new learning models.

Deyannis *et al.* [24] presented a cloud based malware detection solution named TrustAV. This solution is based on a pattern matching technique to determine contaminated data. TrustAV transmits the processing of malware analysis to a remote server and it is proposed as a cloud based solution. According to the paper, TrustAV can protect the transmission and processing of user data even in distrusted environments. In addition, TrustAV also uses a variety of techniques offered by Intel SGX technology to overcome general performance

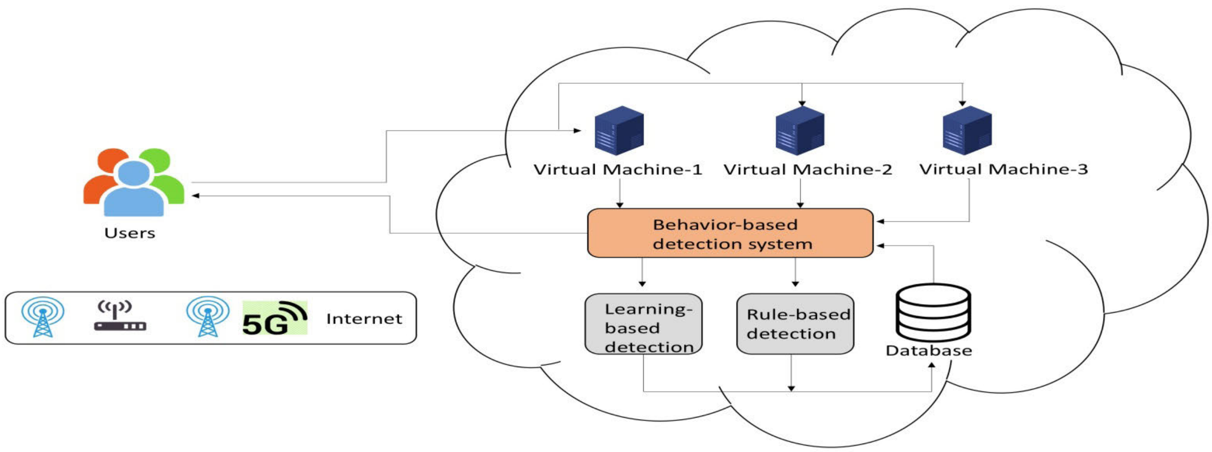


FIGURE 2. Proposed cloud-based malware detection architecture.

loads and limit the risk. However, there is no real data to evaluate the proposed cloud-based TrustAV solution.

When the existing studies are examined for the cloud environment, it is shown that various techniques such as preprocessing, feature reduction and extraction, and ML algorithms have been applied on the dataset to detect malware with high accuracy. When the proposed malware detection methods in these studies are evaluated, it is seen that preprocessing and feature selection stages before implementing ML algorithms improve the performance. In addition, some ML algorithms may perform better than other algorithms according to the size, distribution, and number of features used in the dataset. It can be concluded that the cloud based malware detection approach and its methods improve the detection performance for computers, mobile, and IoT devices with bigger malware databases, and heavy computing resources. Other benefits of cloud based detection are configurations, installations and regular updates. However, some portions of malware could not be detected by using a cloud based detection approach and its methods. To build a more effective detector on the cloud site, hybrid-based detection approach, which combines behavior-, model checking-, and using deep learning altogether on the cloud environment can be a promising method. We believe that cloud-based malware detection approach is still at the early stage, and there needs to be more studies in this area to see effectiveness of the cloud at detecting malware.

IV. PROPOSED SYSTEM

This section explains the proposed system including model architecture, dataset, features, and detection methods in detail.

According to our proposed system, the user submits a suspicious file to the cloud environment by using a computer network. Then, the submitted file is executed in different VMs and execution traces are gathered by using relevant dynamic tools. Generated execution traces are collected on behavior-based detection agent and behaviors

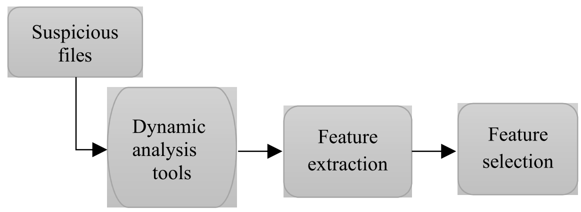


FIGURE 3. Malware analysis process.

are generated. Related behaviors are grouped according to the predefined rules in order to create features. When features are being created, a proposed cloud-based behavior centric model (CBCM) is used. After that, most discriminative features are selected by suggested algorithms and selected features are sent to the detection agents including learning-based detection and rule-based detection. In learning-based detection agent, selected features are trained by using machine learning algorithm such as logistic model trees (LMT), C4.5 (J48), random forest (RF), simple logistic regression (SLR), sequential minimal optimization (SMO), and k-nearest neighbor (KNN). On the other hand, in rule-based detection agent, features are evaluated based upon predefined features sets. Based on learning- and rule-based detection agents, each sample is marked as malware or benign and stored in the database. The analysis result is sent back to the user which shows whether the suspicious file is malware or not.

A. PROPOSED SYSTEM ARCHITECTURE

The system architecture of the cloud-based malware detection model is presented in Figure 2.

B. BEHAVIOR CREATION, FEATURE EXTRACTION AND SELECTION

Analyzing malware manually and extracting features require a lot of time and manpower. Therefore, there is an urgent need to build a system which can automatically analyze the

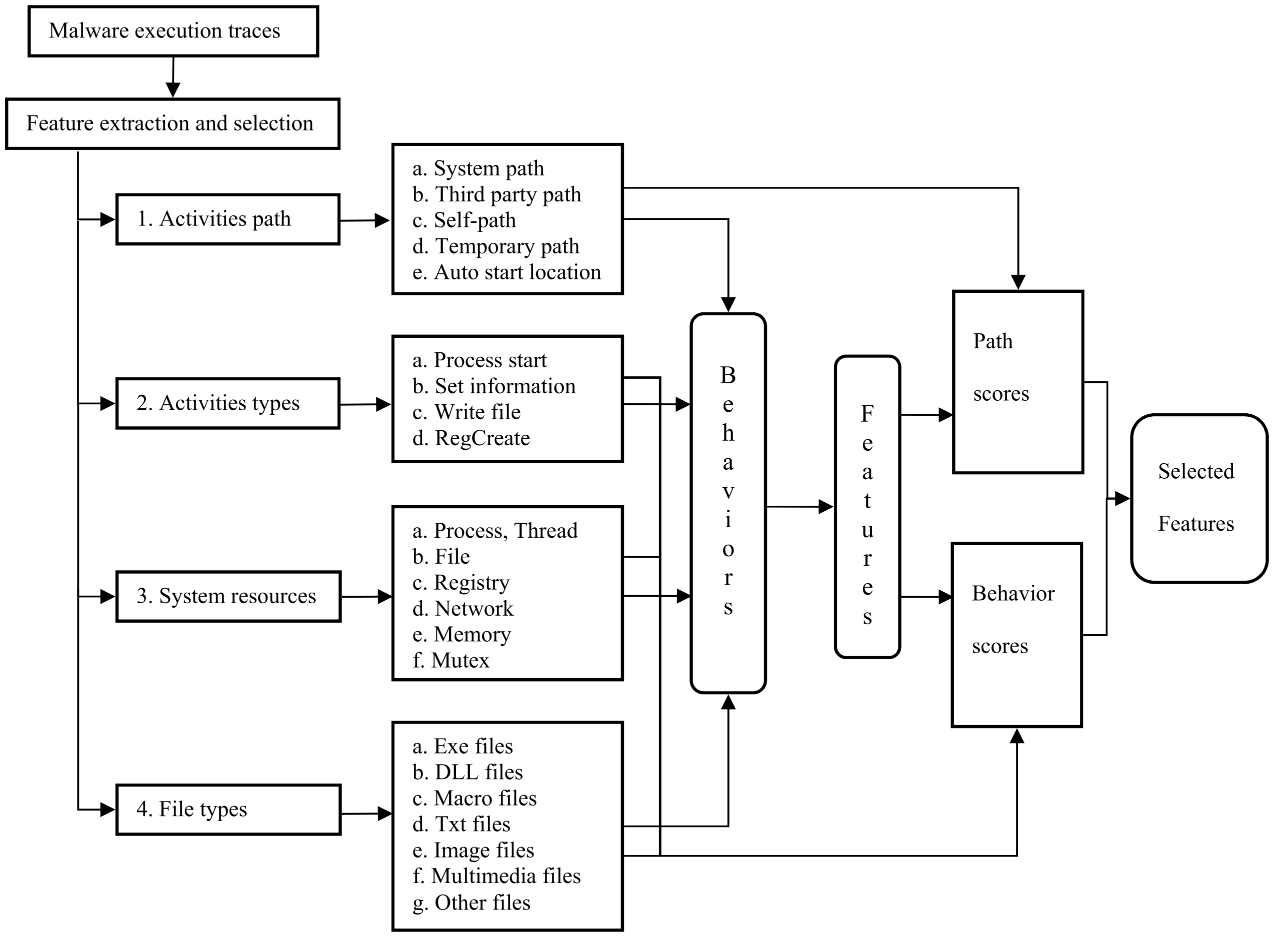


FIGURE 4. Malware feature creation and selection process.

malware and extract features. Although malware performs actions which are related to one another, it also carries out unrelated actions to hide its real behaviors. Because of that, it is vital to determine the interrelated actions and extract real features while creating a dataset. Automatic dataset creation models such as leading methods in the literature and the n-gram are lacking in this regard because those methods generate too many features as well as unrelated features. These deficiencies increase the detection time while decreasing the *DR*. For these reasons, the CBCM model is proposed in this study which creates features and selects features effectively.

Overview of malware analysis process can be seen in Figure 3 and feature creation and selection process can be seen in Figure 4. To create features for each suspicious file, an executable file is analyzed by using dynamic analysis tools such as Process Monitor, API Monitor, Process Explorer, Autoruns, and Debuggers in different VMs. Then, execution traces are collected and sent to the behavior-detection agent. In detection agent, behaviors and features are being created by using the CBCM model. Behavior creation, feature extraction and feature selection are intertwined in the proposed model.

The CBCM model is a modification of the subtractive center behavior model which was proposed in our previous work [25]. While creating features, malicious behavior patterns are determined. The malicious properties are those which can be frequently seen in malicious codes but rarely seen in non-malicious program samples. The goal is to gather the most important properties. To identify malicious properties: system calls, system call paths, system resource types, and file types are taken into consideration (Figure 4). In this way properties that can distinguish malware from benign samples are obtained. To create behaviors, necessary relationships are established among the system calls. One or more system calls, which can represent meaningful activity, create a behavior. The behavior creation algorithm can be seen in Algorithm 1. It takes the list of activities (system calls) D_1 as an input and generates a list of behaviors D_2. During the behavior creation, first based on the activities that are performed, action states (ψ : *active* $(A) - passive(P)$) are established. Then, action paths (μ: self (SF), system (S), third party (TP), temporary (T), and auto start (AS)), where the system calls perform, are determined. After that, consecutive system calls, which can represent behaviors, are determined. The same consecutive system calls and ending system calls,

TABLE 3. Malware execution trace system calls path (The list is abbreviated).

Action System Path
' c:\windows', system folder
' hklm\system ', system folder
' c:\windows\system32', system folder
'c:\program files', third party folder
'c:\program files (x86)', third party folder
'\....\suspicious file', self-folder
' c:\users\...\startmenu\...\startup', auto start location
' hklm\software \microsoft\active setup\installed components', auto start location
'hkcu\software\Microsoft\windows\currentversion\runonce\setup', auto start location
'hklm\software\microsoft\windows\currentversion\run', auto start location
'c:\documents and settings\ user name\local settings\temp', temporary folder
'c:\ users\user name \appdata\local\temp', temporary folder

which represent the actions, are excluded when behaviors are generated. Finally, obtained behaviors are written to do D_2.

Unlike n-gram, the CBCM uses twenty consecutive behaviors when creating properties. If there is a relationship between first and tenth or twentieth behaviors, it can create a property. The importance of the properties is determined as follows:

1. Paths that system calls are performed on.
2. Resources that system calls are performed on.
3. File types that are created.

1) PATHS THAT SYSTEM CALLS ARE PERFORMED ON

We divided performed system calls locations into five categories, which is shown in Figure 4 including system, third party, self, temporary, and auto start locations. Each path is further divided into subfolders and path scores are established. These paths are used during the behavior and feature creation. Example list of system activities path can be seen in Table 3 and path score calculation for behaviors can be seen in Algorithm 2.

When the system call is performed in system folder, following criteria are taken into consideration (Algorithm 2):

- (i) If an analyzed program sample interacts with the operating system files and directories in order to work properly, these interactions are evaluated normal. Most of the time, these interactions are provided by system DLLs, background processes, and system services. These interactions are considered to be normal, so the risk level of these interactions will be low or moderate depending upon the other information
- (ii) If analyzed program tries to inject some codes to the system DLLs and exes including kernel32.dll, advapi.dll, svchost.exe, winlogon.exe, etc., those actions are considered to be malicious and the risk level of these interactions will be high.

When the system call is performed in third party folder, following criteria are taken into consideration:

- (i) Most programs need third-party software to run properly. If an analyzed program sample needs other programs to run properly, the risk level of these interactions will be low or moderate depending on the other information.
- (ii) However, if actions on the third party files and directories that are not related to the performed sample, those actions are considered malicious and the risk level of these interactions will be high.

Algorithm 1 Behavior Creation

Input (D_1 file): List of activities/system calls
Output (D_2 file): List of behaviors

for each system call in D_1i **do**
  if $D_{1[i][state]}$ == 'active' **then**
    ψ = 'A'
  else
    ψ = 'P'
  end if
  if process.$_{name}$ ==d_1.$_{filename}$ **then**
    μ = 'SF'
  elif path == 'system' **then**
    μ = 'S'
  elif path == 'thirdParty' **then**
    μ = 'TP'
  elif path == 'temporary' **then**
    μ = 'T'
  elif path == 'autostart' **then**
    μ = 'AS'
  end if
  if $D_{1[i][systemcall]}$!= $D_{1[i+1][systemcall]}$ **then**
    if $D_{1[i][systemcall]}$!= 'ending system call' **then**
      write. $D_2(D_{1[i]\,[systemcall]})$
    end if
  end if
  if $D_{1[i][systemcall]}$ == $D_{1[i+1][systemcall]}$ **then**
    if $D_{1[i][path]}$!= $D_{1[i+1]\,[path]}$ **then**
      if $D_{1[i][systemcall]}$!= 'ending system call' **then**
        write. $D_2(D_{1[i]\,[systemcall]})$
      end if
    end if
  end if
end for

When the system call is performed in its own folder, following criteria are taken into consideration:

- (i) If an analyzed program needs some data from its own directory or file in order to run properly, it generates normal actions that cannot be categorized as malicious. For those actions, the risk level will be low.
- (ii) However, if an analyzed program sample performs registry and network-related actions within some files or copies its own file content to other files, it is considered to be malicious and the risk level of these interactions will be high.

Temporary folder and auto start locations are other paths which need to be considered. This is because most of the malware types use temporary folders when performing malicious actions, and use auto start file-registry locations to become persistent in the system.

(i) If an analyzed program sample is using temporary folder or auto start locations, these interactions are considered to be malicious and the risk level of these interactions will be fairly high.

2) RESOURCES THAT SYSTEM CALLS ARE PERFORMED ON

In order to create behaviors and related properties, system resources are split into following categories: process, thread, file, registry, network, memory and mutex. During the determining behaviors and properties, usually the same types of resources are considered. When malware first runs, it creates some processes and threads to perform malicious actions. These processes and threads can make some changes on files, registry entries, memory and mutexes, or can connect other networks to exchange some sensitive data. Because of that each action which is carried out on those system resources is analyzed deeply during the feature creation.

3) FILE TYPES THAT ARE CREATED

Created file types are also taken into consideration during feature creation. We considered portable executable (exe, DLL) and macro files slightly more dangerous than other files including txt, image, multimedia files, etc. This is because several malware variants create exe extension files or inject malformed program codes into DLL files to launch attacks.

The feature extraction algorithm is presented in Algorithm 3. When features are generated from behaviors, twenty consecutive behaviors are considered. In this phase, features, feature action types, and path scores are calculated. The same types of system resources (file, registry, mutex, network, etc.) are considered when determining property relationships. In addition, different resources create features if relationships can be established among them. Path scores and action states (AA, AP/PA, PP) are used during the feature selection.

The feature selection algorithm is presented in Algorithm 4. First, the frequency of each property is calculated. During the feature frequency calculation, we try to reduce the number of different features as many as we can. The features of the same name, which occur on the same resource type and have the same path score but different locations, are combined with the same property and the frequency is increased. For instance, even though ReadFileWriteFile (‘\...\path1\’, pathScore = ‘x’) and ReadFileWriteFile (‘\...\path2\’, pathScore = ‘x’) have been performed in different locations and instances, they set to the same feature and frequency is increased.

After frequency calculation is finished, features are selected based upon path scores and action states. If the path score is moderate, high or very high, related property is chosen.

Algorithm 2 Behavior Path Score Calculation

Input (D_2 file): List of behaviors
Output: Behavior's path score
for each behavior i **do**
  if μ == 'SF' **then**
    if process. $_{name}$ == D_2. $file_{name}$ **then**
      pathScore = 'low'
    elif process. $_{name}$! = D_2. $file_{name}$ and D_2. $file_{name}$ =='system $*$. exe' **then**
      pathScore = 'high'
    else
      pathScore = 'moderate'
    end if
  elif μ == 'TP' **then**
    if $D_{2[i][path]}$ == 'program files' **then**
      pathScore = 'moderate'
    elif $D_{2[i][path]}$ == 'AS' **then**
      pathScore = 'high'
    else
      pathScore = 'low'
    end if
  elif μ == 'S' **then**
    if process. $_{name}$ == D_2. $file_{name}$ **then**
      pathScore = 'low'
    elif D_2. $File_{extension}$ == 'exe' or DLL **then**
      pathScore = 'moderate'
    elif $D_{2[i][path]}$ =='registry AS similar locations' **then**
      pathScore = 'high'
    elif $D_{2[i][path]}$ == 'system *. exe' **then**
      pathScore = 'very high'
    else
      pathScore = 'low'
    end if
  elif μ == 'T' or μ == 'AS' **then**
    pathScore = 'very high'
  end if
end for

Furthermore, even if the path score is low but the action state is AA or AP/PA, this property is also chosen. These properties are considered because we try to choose only malicious related patterns which differentiate malware from benign. That way, normal features which can be performed by malware and benign samples are removed from the dataset. Thus, our proposed algorithms create far fewer features than well-known algorithms and n-gram.

C. LEARNING-BASED DETECTION

After features are selected from the previous section, each program sample is represented by a row vector. For each property, frequency value is written. If property is repeated x times, x is written as a property value, if property is not repeated, 0 is written as a value. After the dataset is built based on feature vectors, learning algorithms are applied. In learning-based detection agent, selected features

Algorithm 3 Feature Extraction From Behaviors

Input (D_2 file): List of behaviors
Output (D_3 file): List of Features

propertyName [] = ' '
for each i in D_2 **do**
  for j = i+1 to i+20 **do**
    if $D_{2[i][behaviorStatus]}$ == $D_{2[j][behaviorStatus]}$ and $D_{2[i][behaviorStatus]}$ == 'A' **then**
      ψ_i = 'AA'
    elif $D_{2[i][behaviorStatus]}$ == $D_{2[j][behaviorStatus]}$ and $D_{2[i][behaviorStatus]}$ == 'P' **then**
      ψ_i = 'PP'
    else
      ψ_i = 'AP' = 'PA'
    end if
    if $D_{2[i][behaviorType]}$ ==$D_{2[j][behaviorType]}$ and $D_{2[i][behaviorName]}$!= $D_{2[j][behaviorName]}$ and ($D_{2[i][path]}$ == $D_{2[j][path]}$ or $D_{2[i][behaviorRead]}$ before $D_{2[j][behaviorWrite]}$) **then**
      Algorithm 2 BehaviorPathScoreCalculation ($D_{2[i][behavior]}$, $D_{2[j][behavior]}$)
      propertyName$_{[k]}$ = $D_{2[i][behaviorName]}$+' '+ $D_{2[j][behaviorName]}$
      k = k +1
    end if
    write. D_3(propertyName$_{[k]}$, ψ_i, $D_{2[i][path]}$, $D_{2[j][path]}$, $D_{2[i][pathScore]}$, $D_{2[j][pathScore]}$)
    if $D_{2[i][behaviorType]}$!=$D_{2[j][behaviorType]}$ and $D_{2[i][behaviorRead]}$ before $D_{2[j][behaviorWrite]}$ **then**
      propertyName$_{[k]}$ = $D_{2[i][behaviorName]}$+' '+ $D_{2[j][behaviorName]}$
      k = k +1
      Algorithm 2 BehaviorPathScoreCalculation ($D_{2[i][behvior]}$, $D_{2[j][behavior]}$)
    end if
    write. D_3(propertyName$_{[k]}$, ψ_i, $D_{2[i][path]}$, $D_{2[j][path]}$, $D_{2[i][pathScore]}$, $D_{2[j][pathScore]}$)
  end for
  write. D_3($D_{2[i][behaviorName]}$, $D_{2[i][\psi]}$, $D_{2[i][path]}$, $D_{2[i][pathScore]}$)
end for

Algorithm 4 Frequency Calculation and Feature Selection

Input (D_3 file): List of Features
Output (D_5 file): Selected Features with Frequency

for each feature i in D_3 **do**
  calculateFeatureFrequncy ()
  write. D_4 (propertyName, frequency)
end for
for each feature i in D_4 **do**
  if (pathScore == 'moderate' or == 'high' or == 'very high') **then**
    write. D_5 (propertyName, frequency)
  end if
  if (pathScore == 'low') and (ψ == 'AA' or == 'AP/PA') **then**
    write. D_5 (propertyName, frequency)
  end if
end for

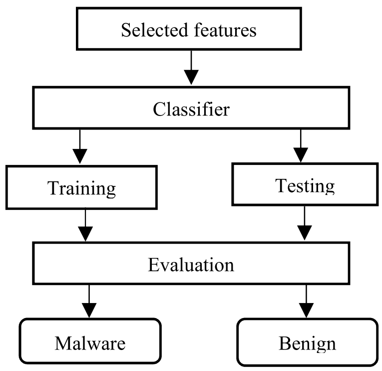


FIGURE 5. Learning-based malware detection agent.

are trained by using machine learning algorithms (classifiers) including C4.5, LMT, RF, KNN, SLR and SMO. Several classifiers are used for classification to measure the proposed method efficiency. Learning-based detection agent in the cloud can be seen in Figure 2, training and testing phase can be seen in Figure 5.

During the training phase, cross-validation and holdout methods are used to measure the performance. Decision trees such as C4.5, LMT and RF are used for training and testing as a classifier because they return scalable and highly accurate results in the cloud environment. Besides, decision trees are suitable classifiers for our dataset features distribution to separate malware from benign. C4.5 uses gain ratio for feature placement. In gain ratio, the feature with the maximum gain is selected recursively for splitting criteria when features are placed on the tree. The gain ratio is prone to unbalanced partitioning and hence can create uneven trees. The information gain ratio is measured as follows.

$$Gain\ Ratio\,(A) = Gain\,(A)\,/Split\ Information_A\,(D) \tag{1}$$

$$Gain\,(A) = Information\,(D) - Information_A\,(D) \tag{2}$$

$$Split\ Information_A\,(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} log_2(\frac{|D_j|}{|D|}) \tag{3}$$

$Gain\,(A)$ shows how much information will be gained when branching using the property A and $SplitInformation_A(D)$ shows the intrinsic information which measures the entropy of the sub-dataset. C4.5 is appropriate for our dataset because it works with continuous data, eliminating data with noise, and prunes decision trees effectively.

RF is a combination of many trees which classifies using attributes that each tree is sampled independently. This classifier produces satisfying results in a dataset with low variance and many interrelated features. It uses the CART

(Classification and regression tree) algorithm to generate RF trees. On the other hand, LMT is a classifier which uses a supervised learning algorithm that combines LR and decision tree learning, and produces results with high accuracy. LMT classifier creates LR functions on each node using the LogitBoost algorithm [26] and prunes the tree using the CART algorithm. The CART algorithm works according to the depth priority search and uses the Gini index as a criterion for splitting features. The Gini index is used to measure the differences between the probability distributions of target feature values. The feature with the minimum Gini index is selected as the splitting attribute. The Gini index does not work well when the number of classes and the value of properties are very large. The Gini index is calculated as follows.

$$Gini\,(A) = Gini\,(D) - Gini_A\,(D) \tag{4}$$

$$Gini\,(D) = 1 - \sum\nolimits_{i=1}^{m} (P_i)^2 \tag{5}$$

$$Gini_A\,(D) = \frac{|D_1|}{|D|} Gini\,(D_1) + \frac{|D_2|}{|D|} Gini\,(D_2) \tag{6}$$

KNN is a statistical model classifier which uses example-based learning. It is a classifier that produces good results when there is no prior knowledge about data distribution. Even though KNN classifier needs a lot of storage space during the learning phase, it performs well in the cloud environment for our dataset. Even if the SLR algorithm is not adequate to solve non-linear problems and comprise high bias which reduces the efficiency of the classifier, it is suitable and fast when combined with the proposed model. Since SMO works well for non-linear boundary situations and performs well on high-dimensional data, it performs well on our dataset on the cloud. After training and testing phases are performed by using C4.5, LMT, RF, KNN, SLR and SMO, the results are sent to the behavior-based detection agent.

D. RULE-BASED DETECTION

Rule-based behavior malware detection agent is running on different machines in the cloud. For detection there is no training or learning phase, instead detection is performed based on the predefined property list (Figure 6). We use malware behaviors when creating predefined properties. This list consists of features which differentiate malware from benign based on malicious behavior patterns. The malicious behavior patterns are the features which can be frequently performed by malware while rarely performed by benign samples. The malicious behavior pattern list is dynamically updated when new malware features are determined. After features are created and selected in section IV.B, the feature values are categorized based on repeated frequencies into four categories. These categories are {few}, {average}, {many}, and {excessive}. If the analyzed program features are substantially similar to the features in the list, the program is marked as malware (Algorithm 5). Otherwise, the program is marked as benign. For instance, analyzed program features are somehow in the predefined properties list but without enough repeated frequency, the analyzed program is marked as benign. Feature paths are also used during the detection. According to our findings we determined fifty features which are frequently used by malware such as CreateService, CreateRemoteThread, FindFirstFile,FindNextFile, MapviewofFile, CreateFileMapping, QueryDirectoryWriteFile, ReadFileWriteFile, RegDeleteValue, RegQueryKeyRegSetInfoKey, etc.

Algorithm 5 Rule-Based Detection

Input: List of Features
Output: List of Marked Program Samples
pathScore {'very low', 'low', 'moderate', 'high', 'very high'}
list {…} ← predefinedList {…}, fC ← frequencyCategory {'few', 'average', 'many', 'excessive'}
for each feature i **do**
  if $i_{pathScore}$ == 'low' and i ∈ list { } and (fC == 'many' or 'excessive') **then**
    suspiciousFile = 'malware'
  elif $i_{pathScore}$ == 'moderate' and i ∈ list { } and (fC == 'average' or 'many' or 'excessive') **then**
    suspiciousFile = 'malware'
  elif $i_{pathScore}$ == 'high' and i ∈ list { } and (fC == 'average' or 'many' or 'excessive') **then**
    suspiciousFile = 'malware'
  elif $i_{pathScore}$ == 'very hig' and i ∈ list { } and (fC == 'few' or 'average' or 'many' or 'excessive') **then**
    suspiciousFile = 'malware'
  elif $i_{pathScore}$ == 'hig' and (fC == 'excessive') **then**
    suspiciousFile = 'malware'
    list = list + '$i_{feature}$'
  elif $i_{pathScore}$ == 'very hig' and (fC == 'many' or 'excessive') **then**
    suspiciousFile = 'malware'
    list = list + '$i_{feature}$'
  else
    suspiciousFile = 'benign'
  end if
end for

The proposed rule-based detection agent detects various forms of unknown and known malware efficiently. It is also quite fast when compared with a learning-based detection agent. After the rule-based detection agent finishes its task, the results are stored in the database and sent back to the behavior-based detection agent. Since the rule-based detection agent is quite fast when compared with a learning-based agent, the detection result is first sent to the client while learning-based detection is still performing. After the learning-based detection process is finished, its results are also sent to the client as well. For future study, we aim to combine learning-based and rule-based detection results. Behavior-based detection agent will compare the results coming from learning-based detection and rule-based detection agents. If there are some differences, the detection

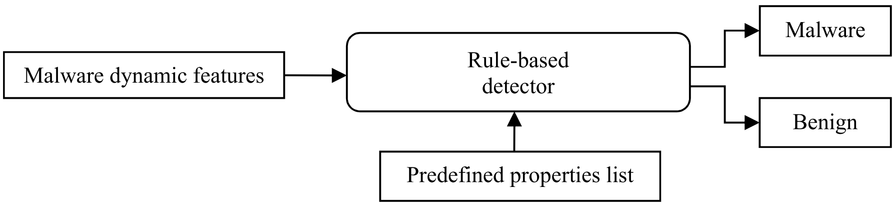


FIGURE 6. Rule-based malware detection agent.

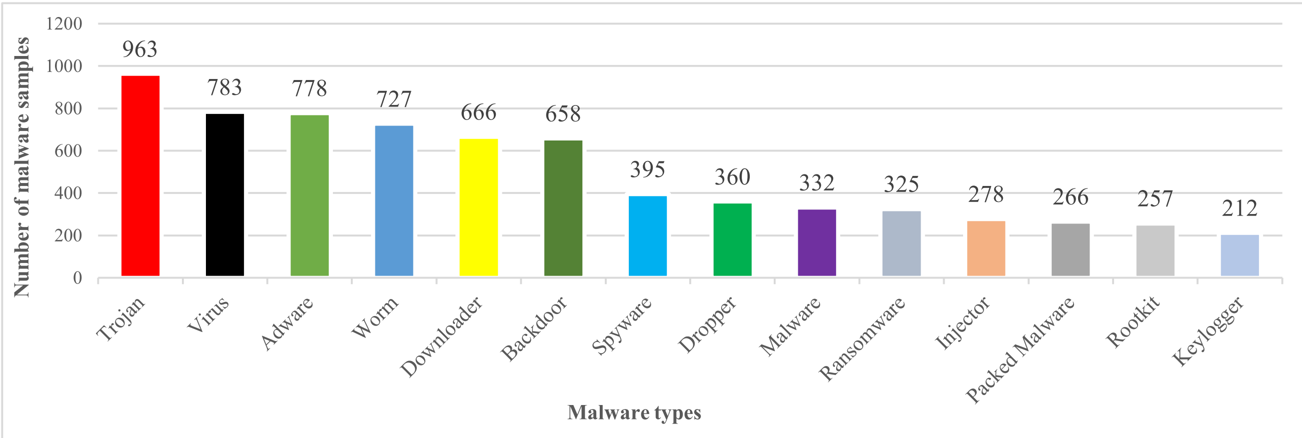


FIGURE 7. Analyzed malware distribution.

process will be repeated for those samples to decrease the misclassification rate. When the same classification results are gathered from both detection agents, the results will be sent to the client.

V. CASE STUDY

This section presents case study and experiments. In order to simulate the cloud environments, we used different computers, VMs, switches and routers in the campus network. Different versions of Windows machines are used for test cases including Windows 7, 8, 10, VMs 7, 8 and 10. Proposed dataset creation model is implemented by using Python scripting language. For learning-based detection, Weka and some python libraries are used, and for rule-based detection the proposed algorithm is implemented in Python language as well. Totally, 7000 malware and 3000 benign portable executables are analyzed. Data collection and representation, model performance and evaluation are explained in the following subsections.

A. DATA COLLECTION AND REPRESENTATION

Malware samples were collected from various sources including Das Malwerk, MalwareBazaar, Malware DB, Malware Benchmark, Malshare, Tekdefense, ViruSign, VirusShare, KernelMode [27]–[35]. Benign samples were collected from different legitimate websites which include various categories such as office documents, games, system tools, and other third party's software. Totally, 7000 malware and 3000 benign samples were collected and analyzed on different Windows VMs as well as real machines. Different malware types include virus, worm, trojan, rootkit, backdoor, ransomware, spyware, etc. and families include Generic, Agent, Win32, Emotet, Ramnit, Sinowal, Sality, Snoopy, Cryptolocker, Ransomlocker, etc. were collected. Analyzed malware types and their families can be seen in Table 4. Collected malware samples were labeled by using Virustotal [36]. 7000 malware samples were chosen among 20,000 malware. The number of collected and analyzed malware samples for each category can be seen in Figure 7.

Collected malware samples were performed in different VMs and execution traces were sent to the detection agent in the cloud. To get execution traces Process Monitor, Process Explorer, and Autoruns were used. Data collection, analysis and representation process can be seen in Figure 8. Each sample was executed between 5 to 15 seconds depending on the number of activities generated by malware. Execution traces were analyzed by using our proposed algorithms to generate behaviors and features (Figure 9-10). Features are selected based on risk scores (Figure 11). The proposed algorithms were implemented by using Python scripting language. Each sample is represented as a row feature vector for learning-based detection (Figure 12). If the feature occurs for a related sample, the frequency of the repetition is written. If the feature is not repeated for a related sample, 0 is written. Since 0 is written for not repeated features, the size of the feature set is increasing over time. For 10,000 samples we gathered 751 features. On the other hand, for rule-based detection feature vectors, only features that are performed by related samples are taken into consideration. The repetition of each feature is written. For 10,000 samples, we got an average

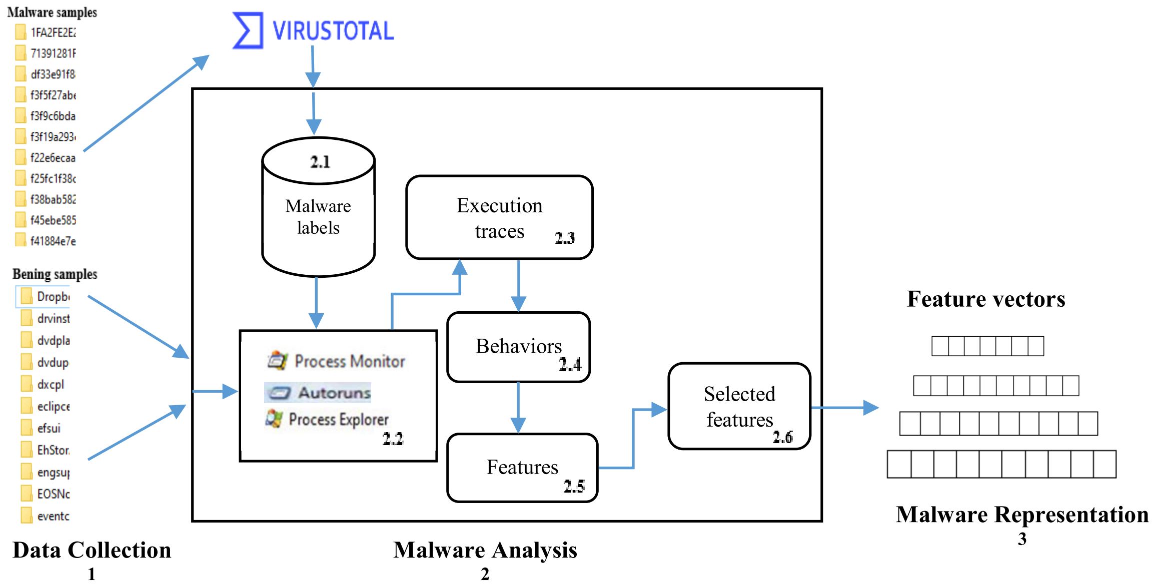


FIGURE 8. Data collection, analysis and representation process.

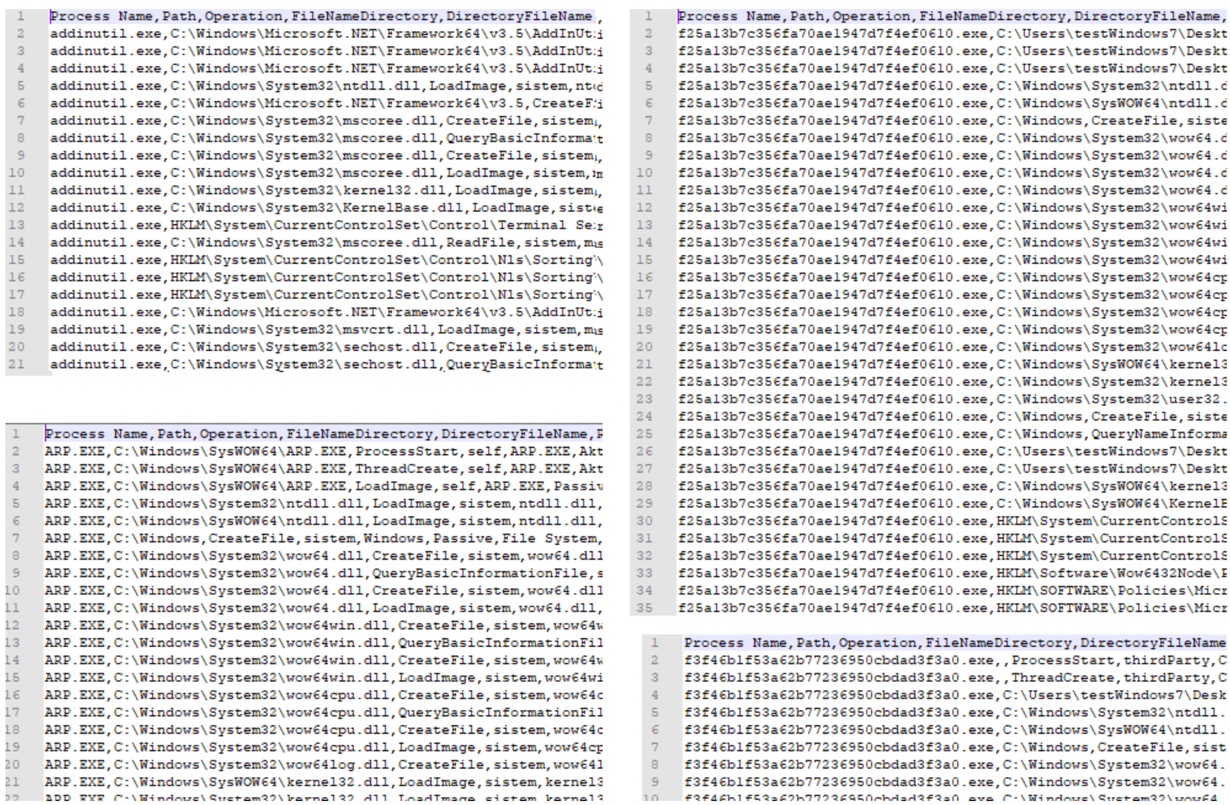


FIGURE 9. Extracted behaviors for 4 samples (Figure 8. 2.4) (The list is shortened).

of 60 features, which is quite fewer when compared to the other datasets.

B. MODEL PERFORMANC AND EVALUATION

After the feature selection process is completed, learning-based and rule-based detection processes start. To measure the performance of proposed model holdout and cross-validation methods, as well as detection rate (*DR*), false positive rate (*FPR*), f-measure, and accuracy metrics were used. At first, when the generated dataset was small, the cross-validation method returned more feasible results than holdout. However, when the dataset has grown timely, the holdout method returned favorable results as well. Best performances are obtained when k is chosen 10 for cross validation, and the data is divided into 80% training and 20% test for holdout method. TP represents the number of

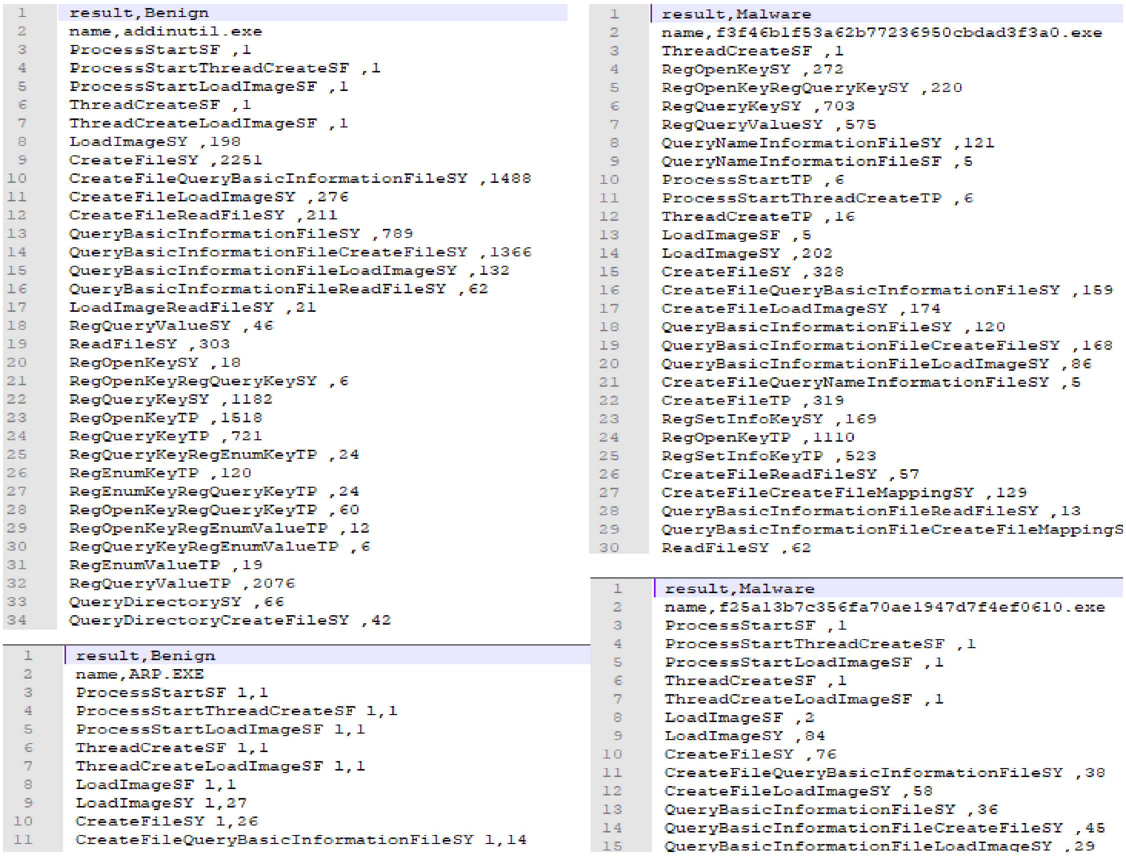


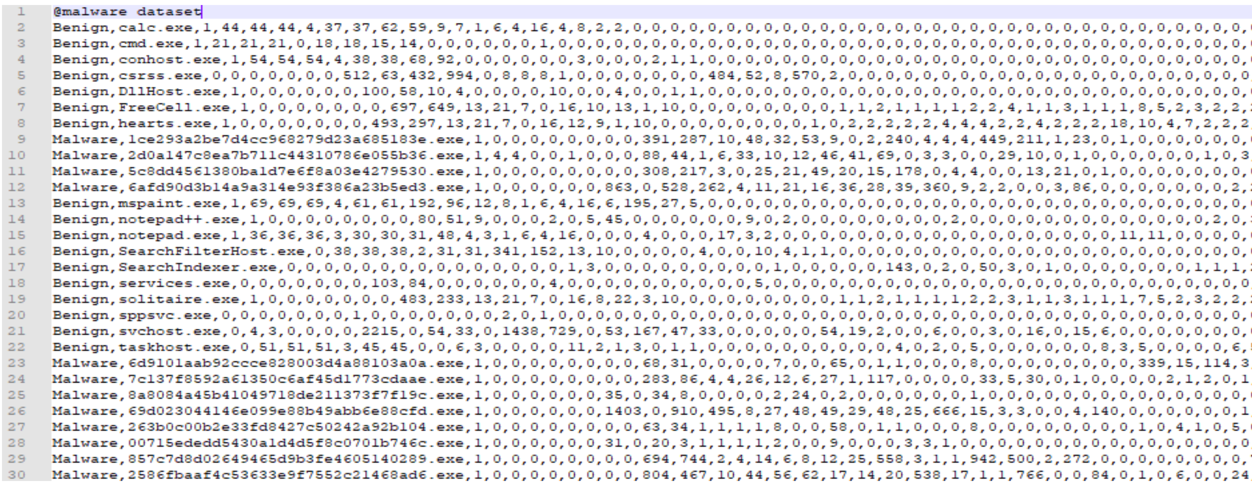
FIGURE 10. Generated features for 4 samples (Figure 8. 2.5) (The list is shortened).

FIGURE 11. Selected features 4 samples (Figure 8. 2.6) (The list is shortened).

malware samples correctly classified as malware, TN the number of benign samples correctly classified as benign, FP the number of benign samples being mistakenly classified as malware, and FN the number of malware samples being mistakenly classified as benign. *DR*, *FPR*, f-measure, and accuracy metrics are calculated by using confusion matrix (Table 5) as follows:

$$DR = \text{Recall} = TP/(TP + FN) \tag{7}$$

$$FPR = FP/(FP + TN) \tag{8}$$



```
@malware dataset
Benign,calc.exe,1,44,44,44,4,37,37,62,59,9,7,1,6,4,16,4,8,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
Benign,cmd.exe,1,21,21,21,0,18,18,15,14,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
Benign,conhost.exe,1,54,54,54,4,38,38,68,92,0,0,0,0,0,0,3,0,0,0,2,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
Benign,csrss.exe,0,0,0,0,0,0,0,512,63,432,994,0,8,8,8,1,0,0,0,0,0,0,0,484,52,8,570,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
Benign,DllHost.exe,1,0,0,0,0,0,0,100,58,10,4,0,0,0,0,10,0,0,4,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
Benign,FreeCell.exe,1,0,0,0,0,0,0,0,697,649,13,21,7,0,16,10,13,1,10,0,0,0,0,0,0,0,0,0,1,1,2,1,1,1,1,2,2,4,1,1,3,1,1,1,8,5,2,3,2,2,2
Benign,hearts.exe,1,0,0,0,0,0,0,0,493,297,13,21,7,0,16,12,9,1,10,0,0,0,0,0,0,0,0,0,1,0,2,2,2,2,2,4,4,4,2,2,4,2,2,2,18,10,4,7,2,2,2,
Malware,1ce293a2be7d4cc968279d23a685183e.exe,1,0,0,0,0,0,0,0,391,287,10,48,32,53,9,0,2,240,4,4,4,449,211,1,23,0,1,0,0,0,0,0,0,0
Malware,2d0a147c8ea7b711c44310786e055b36.exe,1,4,4,0,0,1,0,0,0,88,44,1,6,33,10,12,46,41,69,0,3,3,0,0,29,10,0,1,0,0,0,0,0,0,1,0,3,
Malware,5c8dd4561380ba1d7e6f8a03e4279530.exe,1,0,0,0,0,0,0,0,0,308,217,3,0,25,21,49,20,15,178,0,4,4,0,0,13,21,0,1,0,0,0,0,0,0,0,0
Malware,6afd90d3b14a9a314e93f386a23b5ed3.exe,1,0,0,0,0,0,0,863,0,528,262,4,11,21,16,36,28,39,360,9,2,2,0,0,3,86,0,0,0,0,0,0,0,0,2,2
Benign,mspaint.exe,1,69,69,69,4,61,61,192,96,12,8,1,6,4,16,6,195,27,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
Benign,notepad++.exe,1,0,0,0,0,0,0,0,0,80,51,9,0,0,0,2,0,5,45,0,0,0,0,0,0,9,0,2,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,3
Benign,notepad.exe,1,36,36,36,3,30,30,31,48,4,3,1,6,4,16,0,0,0,4,0,0,0,17,3,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,11,11,0,0,0,0,0
Benign,SearchFilterHost.exe,0,38,38,38,2,31,31,341,152,13,10,0,0,0,0,4,0,0,10,4,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
Benign,SearchIndexer.exe,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,3,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,143,0,2,0,50,3,0,1,0,0,0,0,0,0,0,1,1,1,2
Benign,services.exe,0,0,0,0,0,0,0,103,84,0,0,0,0,0,0,4,0,0,0,0,0,0,0,0,0,0,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
Benign,solitaire.exe,1,0,0,0,0,0,0,0,483,233,13,21,7,0,16,8,22,3,10,0,0,0,0,0,0,0,0,0,1,1,2,1,1,1,1,2,2,3,1,1,3,1,1,1,7,5,2,3,2,2,2
Benign,sppsvc.exe,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,2,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
Benign,svchost.exe,0,4,3,0,0,0,0,2215,0,54,33,0,1438,729,0,53,167,47,33,0,0,0,0,0,54,19,2,0,0,6,0,0,3,0,16,0,15,6,0,0,0,0,0,0,0,0
Benign,taskhost.exe,0,51,51,51,3,45,45,0,0,6,3,0,0,0,0,11,2,1,3,0,1,1,0,0,0,0,0,0,0,0,0,0,4,0,2,0,5,0,0,0,0,0,0,8,3,5,0,0,0,0,6,5
Malware,6d9101aab92ccce828003d4a88103a0a.exe,1,0,0,0,0,0,0,0,0,68,31,0,0,0,0,7,0,0,65,0,1,1,0,0,0,8,0,0,0,0,0,0,0,0,339,15,114,3,
Malware,7c137f8592a61350c6af45d1773cdaae.exe,1,0,0,0,0,0,0,0,0,0,283,86,4,4,26,12,6,27,1,117,0,0,0,0,33,5,30,0,1,0,0,0,0,2,1,2,0,1,
Malware,8a8084a45b41049718de211373f7f19c.exe,1,0,0,0,0,0,0,0,35,0,34,8,0,0,0,0,2,24,0,2,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
Malware,69d023044146e099e88b49abb6e88cfd.exe,1,0,0,0,0,0,0,1403,0,910,495,8,27,48,49,29,48,25,666,15,3,3,0,0,4,140,0,0,0,0,0,0,1,
Malware,263b0c00b2e33fd8427c50242a92b104.exe,1,0,0,0,0,0,0,0,0,63,34,1,1,1,1,8,0,0,58,0,1,1,0,0,0,8,0,0,0,0,0,0,0,1,0,4,1,0,5,0
Malware,00715ededd5430a1d4d5f8c0701b746c.exe,1,0,0,0,0,0,0,0,31,0,20,3,1,1,1,1,2,0,0,9,0,0,0,3,3,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
Malware,857c7d8d02649465d9b3fe4605140289.exe,1,0,0,0,0,0,0,0,0,694,744,2,4,14,6,8,12,25,558,3,1,1,942,500,2,272,0,0,0,0,0,0,0,0,7
Malware,2586fbaaf4c53633e9f7552c21468ad6.exe,1,0,0,0,0,0,0,0,804,467,10,44,56,62,17,14,20,538,17,1,1,766,0,0,84,0,1,0,6,0,0,24,
```


FIGURE 12. Feature vectors (Figure 8. 3) (The list is shortened).

TABLE 4. List of malware types that are analyzed (The list is shortened).

Malware MD5 Signature	Malware Type	Malware Family/Malware Specific Name
f2c6a6541976bab117d03f7a8c2ccbf7	Trojan	Trojan.Win32.Generic, Trojan.Injector, Trojan.Symmi
f3a6ab31986c928d312699dc7208a211	Ransomware	TR/Crypt.Zpack, Ransom:Win32
28cb0c8083f6a41e7b04137ab166c580	Packed Malware	Gen:Packer.PESpin, Trojan.Win32.Crypt
f448a906cc9906b8f7589b117a079280	Backdoor	Backdoor.BDS, Trojan [Backdoor]/Win32.Hlux
b64f34137982332156e058cd63cf480b	Dropper	Win32/TrojanDropper.VB, Trojan.TR/Drop.VB
996f29ba29a14fc0ebf46ce38675f8cd	Ransomware	Ransom: Win32/Blocker
4a7e35d8c111e213a051462e66e73a3e	Packed Malware	Win32.Packed.VMProtect, HackTool.GameHack!8
f3aa059c23a2080bc0b219eebf5577e0	Virus	Virus:Win32, Win32.Parite.B
9085a7dff20d6a5c287d3056d3ed1cc4	Rootkit	Dropper.Generic_r.AC, Win32:Rootkit-gen
48cd89827939b3a8976d9bb0993bc338	Spyware	Win.Spyware.Zbot, Gen:Variant.Razy
f3b03c25e1a53168a606732fc96707e3	Keylogger	Gen:Application.Keylog.dm0, TrojanSpy.Vwealer
f3c6372f9c95ba38d72a5c2219ce9f8b	Worm	Worm/Win32.Mabezat

TABLE 5. Confusion matrix.

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN

$$\text{Precision} = TP/(TP + FP) \tag{9}$$

$$\text{F} - \text{Measure} = (2 * \text{precision} * \text{recall})/(\text{precision} + \text{recall}) \tag{10}$$

$$\text{Accuracy} = TP + TN/(TP + TN + FP + FN) \tag{11}$$

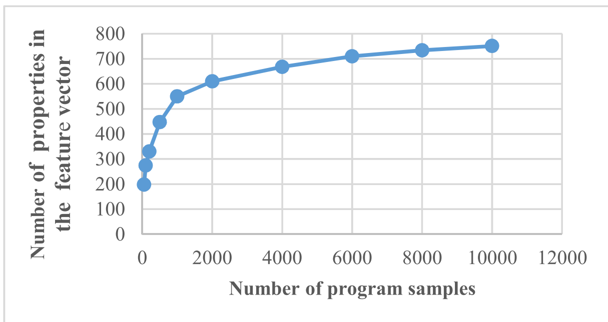


FIGURE 13. Number of analyzed program samples versus properties in the feature vectors for learning-based detection.

VI. RESULTS AND DISCUSSION

This section summarizes the test results and discusses the proposed system performance. When performance is evaluated, various learning algorithms are used. During the training and testing, best performances are gathered by using cross validation k = 10 and holdout method which is using 80% training and 20% testing sets. *DR*, *FPR*, f-measure, and accuracies are used as metrics to compare test results. For rule-based detection, we did not use any training and testing phases. The performance is obtained in real time by using predefined features. The cloud environment has provided a fast and scalable environment for our learning- and rule-based system. The test results can be seen in Figure 13, Table 6, Table 7, Table 8, Table 9, and Table 10.

Table 6 demonstrates the proposed model performance when 10,000 program samples are analyzed. For learning-based detection, both cross validation and holdout methods are performed fairly well. Cross validation results are slightly higher than holdout results. The best results are obtained when ML classifiers such as decision trees (J48, RF, LMT) and KNN are used. For instance, in J48 *DR*, *FPR*,

TABLE 6. Proposed model results on various classifiers and rule-based detection.

Method	Classifier	DR (%)	FPR (%)	F-measure (%)	Accuracy
Learning-based detection using Cross-validation	**J48**	99.8	0.4	99.8	99.75
	RF	100	0.6	99.6	99.83
	LMT	99.3	0.5	99.6	99.38
	KNN	100	1.2	99.7	99.64
	SLR	95.4	1.6	97.3	96.29
	SMO	93.4	6.7	95.2	93.37
Learning-based detection using Holdout	**J48**	99.3	0.3	99.6	99.45
	RF	99.9	1	99.7	99.6
	LMT	99.4	0	99.7	99.6
	KNN	100	1.5	99.7	99.55
	SLR	93.4	3.6	95.8	94.35
	SMO	92.5	7.6	94.4	92.45
Rule-based detection	**No training**	97.8	6.6	97.4	96.5

TABLE 7. Performance comparison of *n*-gram, ClaMP and our dataset based on selected classifiers.

Classifier	*n*-gram Dataset(1386 samples, 14,520 features)			ClaMP Dataset (5210 samples, 69 features)			Our Dataset (10,000 samples, 751 features)		
	DR	FPR	Accuracy	DR	FPR	Accuracy	DR	FPR	Accuracy
J48	98.1	2	98.05	98.2	2.6	97.8	**99.8**	**0.4**	**99.75**
RF	99	0.4	99.2	99.7	2.1	98.8	**100**	**0.6**	**99.83**
KNN	94.9	0	97.47	98	2	97.9	**100**	**1.2**	**99.64**

TABLE 8. Number of features for rule-based detection.

	Min	Max	Average
Each sample	5	150	60

f-measure and accuracy are measured as 99.8%, 0.4%, 99.8% and 99.75%, respectively. In the same way, RF algorithm achieved 100% for *DR*, 0.6% for FPR, 99.6% for f-measure, and 99.83% for accuracy; LMT achieved 99.3% for *DR*, 0.5% for *FPR*, 99.6% for f-measure, and 99.38% for accuracy; and KNN achieved 100% for *DR*, 1.2% for *FPR*, 99.7% for f-measure, and 99.64% for accuracy. The obtained results on SLR and SMO classifiers are slightly lower than J48, RF, LMT and KNN. Using appropriate kernels for SMO and reducing bias for SLR can increase the performance. On the other hand, the obtained test results are satisfactory for rule-based detection which do not use any learning algorithm and do not require any training phase. rule-based detection achieved 97.8%, 6.6%, 97.4%, and 96.5% for *DR*, *FPR*, f-measure and accuracy, respectively.

Table 7 shows performance on *n*-gram, ClaMP and our dataset based on selected classifiers. It can be clearly seen that J48, RF, and KNN classifiers perform better on our dataset. For example, the J48 algorithm performance on n-gram is measured as 98.1% for DR, 2% for FPR, and 98.05% for accuracy; on ClaMP dataset [37] it is measured as 98.2% for DR, 2.6% for FPR, and 97.8% for accuracy; and on our dataset it is measured as 99.8% for DR, 0.4% for FPR, and 99.75% for accuracy. Similar results are obtained by using different ML classifiers as well.

The number of features is reasonable on our dataset when it is compared with the *n*-gram dataset (Table 7). Figure 13 and Table 8 indicate the number of properties in the feature vectors for learning-based detection and rule-based detection, respectively. Until a certain number, the numbers of properties are increased while analyzed program samples are increasing (Figure 13). This is because some program samples exhibit different features. Furthermore, when properties are combined, each sample's features are added to the feature vector. For repeated frequency, frequency number is written for feature value. If property is not presented in the feature vector, 0 is written for that property. This conversion also raises the number of features in our dataset. However, for rule-based detection we have not created a feature vector. Thus, the number of features is reasonable when compared to other feature extraction methods (Table 8). For rule-based detection, our dataset consists of an average of 60 features with a minimum of 5 and a maximum of 150 (Table 8). Most of the time, the extracted malware features are more than the benign features for each sample for our dataset.

To evaluate the efficiency of the proposed system more accurately, model performance for different feature extraction methods is compared in Table 9. In addition, *DR*, *FPR* and accuracies are compared on the same classifiers for various cloud-based and other studies in the literature (Table 10). The proposed feature extraction method has generated considerably better results than other methods (Table 9) including studies in [38], [39], [40], [41], [42]. The 99.8% performance is measured for the proposed feature extraction method versus 96.4%, 99.6%, 97.6%, 89.92%, and 99.28% in [38], [39], [40], [41], [42] studies, respectively. Even though the performance of some feature extraction methods is quite good in the literature, it cannot be confirmed that they are as successful as the proposed method due to the other deficiencies such as the low number of analyzed samples and higher number of extracted features. The performance of various ML classifiers such as J48, RF, KNN, SVM, SLR, and neural networks on different studies are measured (Table 10). It can be clearly seen that combining the proposed

TABLE 9. Comparison of different malware dataset creation methods.

Paper	Feature Extraction Method	Performance (%)	Year
Anderson et al. [38]	Markov chain weighted directed graph	96.40	2011
Chandramohan et al. [39]	Bounded feature space behavior modeling	99.60	2013
Das et al. [40]	Semantics of malicious behaviors	97.60	2016
Narayanan et al. [41]	Context sensitive, adaptable and scalable rules	89.92	2017
Jeon et al. [42]	Dynamic analysis with CNN	99.28	2020
Proposed Method	**Cloud-based behavior centric model**	**99.80**	**2021**

TABLE 10. Performance of ML classifiers on different studies.

Study	Classifier	DR (%)	FPR (%)	Accuracy (%)	Year
Santos et al. [43]	DT: J48	92	9	91.25	2013
	KNN K = 1	93	7	92.8	
	KNN K = 3	91	8	91.7	
	SVM: Polynomial	88	9	89.65	
Yousefi-Azar et al. [44]	SVM	-	5.07	93.44	2018
	RF	-	6.82	90.05	
	KNN	-	10	91.28	
Yadav [22]	Neural network	86.4	-	-	2019
Kumar et al. [45]	Decision tree	-	12.5	95.7	2020
	RF	-	6.7	97.9	
	LR	-	4.2	94.3	
Azeez et al. [46]	NB	-	-	32.52	2021
	Decision tree	95	5.36	98.29	
	RF	98	2.13	99.24	
Proposed Method	J48	**99.8**	**0.4**	**99.75**	2021
	RF	**100**	**0.6**	**99.83**	
	KNN	**100**	**1.2**	**99.64**	
	SMO	**93.4**	**6.7**	**93.37**	

feature extraction method with an appropriate ML algorithm produces more satisfactory results in terms of *DR*, *FPR* and accuracies when compared with other studies in the literature.

In this section, the performance of the proposed cloud-based malware detection system and the leading methods in the literature are compared. The proposed system successfully detects both different types and families of malware, as well as the new generation and previously unknown malware. The measured performance values increase as the number of programs analyzed increase. In addition, the number of features does not increase after a certain number of programs being analyzed in the proposed system. The results obtained were higher than the pioneering method results in the literature. In addition, testing the proposed system in the cloud environment and using 2 different detection mechanisms provide a distinct advantage. On the other hand, current studies in the literature face some insufficiency for malware detection:

1. Behaviors are not clearly determined.
2. The number of extracted features is high.
3. Perform well for only certain types and families of malicious software.
4. Inadequate for detection of new generation malware.
5. Not resistant to evasion and stealth techniques which leads to decreasing performance.

In the proposed system, these deficiencies were identified and necessary contributions were made to increase the performance.

In addition, a number of key findings were obtained during the malware analysis. These findings should be taken into account while creating a fast and effective detection method. The main findings identified can be listed as follows:

1. Malware creates random files with meaningless file names.
2. Several malware types use newly created processes and existing processes for malicious purposes.
3. Some of the malware injects itself into operating system exe files including svchost.exe, conhost.exe, winlogon.exe, etc. and system DLLs on Windows operating systems.
4. Some malware variants hide themselves by creating similar systems' and third-parties' file names.
5. Some malware variants disable the existing security software such as firewall, IDS, antivirus scanner, etc. whenever they are performed.
6. Some malware variants perform malicious activities in the temporary files.
7. Malware becomes persistent in the computer system by locating itself in the automatic startup file and registry locations.

VII. LIMITATIONS AND FUTURE WORKS

Although CBCM is quite effective in detecting different kinds of malware, there are some limitations that need to be addressed. Malware samples were selected randomly among several malicious software variants, but malware types were not equally distributed. For example, most of the malware samples analyzed were Trojan, virus, adware, worm and downloader. More ransomware, spyware, rootkit, and packed malware need to be analyzed. In total, 10,000 program samples were analyzed, in the future the number of program samples will be increased. Even though our feature extraction and selection algorithms work quite well, there are still some benign samples misclassified as malware. This is because there are some features which are frequently seen in malware but rarely seen in benign. The numbers of these features increased when more program samples are analyzed. In the future, we will improve our feature selection algorithm and also use well-known algorithms mentioned in the literature to decrease the features that lead to misclassification. Even though the proposed model can detect some portion of the obfuscated and packed malware samples, it cannot detect all of them. Thus, the proposed model will be improved more in order to detect those malware samples. We analyze malware only on various versions of Windows machines. We will extend our system to other operating systems including different Linux distributions and macOS.

In this study, we classify the analyzed program samples into two categories including malware and benign classes. In the future, we will also classify the malware types into different classes such as virus, worm, Trojan, rootkit, ransomware, etc. In the cloud environment, a limited number of servers and VMs are used during the analysis, these numbers can be increased in the feature. Rule-based detection agent works in real-time, but learning-based detection agent does not work in real time. In the future, we are planning to combine learning-based and rule-based detection agents to work together in real time. In addition, we aim to build a deep learning-based detection agent on different servers on the cloud as well.

VIII. CONCLUSION

In this paper, a malware detection system which works in the cloud computing environment is presented. There are two parts including client and cloud environment. A client sends suspicious file samples to the cloud, and receives the analyzed results which show whether the suspicious samples are malware or benign. In the cloud, our system consists of three phases. In the first phase, file samples are analyzed by using relevant tools to gather execution traces on different VMs and sent to the behavior-detection agent. In behavior-based detection agent, behaviors and features are generated by using proposed CBCM. In this phase, system calls, system call types, system call paths, system resources and different file types are considered. By this way, malicious features patterns are segregated from benign ones. In the third phase, selected features are sent to the learning-based and rule-based agents to classify file samples as malware or benign. The results are sent back to the behavior-based detection agent, evaluated and sent back to the client.

Our test results confirm that combining proposed feature extraction and selection phases with appropriate learning- and rule-based detection agents increase the performance. The proposed system can effectively detect both known and unknown malware for different data samples. When the proposed system is compared to other systems in the literature, the obtained *DR* and accuracies are quite higher while *FPR* and *FNR* are lower. On the other hand, some portion of the malware samples are remaining undetected due to the use of advanced code obfuscation techniques. Increasing the analysis time, as well as determining more specific features may increase the *DR*. We also aim to extend our system to work on different cloud provider premises such as AWS, IBM Cloud Foundry, Salesforce Platform as well. We hope that our proposed system and its algorithms will assist those who would like to develop an influential detection system on the cloud for daily evolving malware.

REFERENCES

[1] O. Aslan and R. Samet, "Investigation of possibilities to detect malware using existing tools," in *Proc. IEEE/ACS 14th Int. Conf. Comput. Syst. Appl. (AICCSA)*, Oct. 2017, pp. 1277–1284.

[2] O. Aslan and R. Samet, "A comprehensive review on malware detection approaches," *IEEE Access*, vol. 8, pp. 6249–6271, Jan. 2020.

[3] M. R. Watson, N.-U.-H. Shirazi, A. K. Marnerides, A. Mauthe, and D. Hutchison, "Malware detection in cloud computing infrastructures," *IEEE Trans. Dependable Secure Comput.*, vol. 13, no. 2, pp. 192–205, Mar. 2016.

[4] *Website Explaining Cloud Computing*. Accessed: Jan. 12, 2021. [Online]. Available:https://www.investopedia.com/terms/c/cloud-computing.asp

[5] *Website Explaining Cloud Computing*. Accessed: Jan. 14, 2021. [Online]. Available: https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/#benefits

[6] Ö. Aslan, M. Ozkan-Okay, and D. Gupta, "A review of cloud-based malware detection system: Opportunities, advances and challenges," *Eur. J. Eng. Technol. Res.*, vol. 6, no. 3, pp. 1–8, Mar. 2021.

[7] D. Gupta, S. Bhatt, M. Gupta, and A. S. Tosun, "Future smart connected communities to fight COVID-19 outbreak," *Internet Things*, vol. 13, Mar. 2021, Art. no. 100342.

[8] D. Gupta, S. Bhatt, M. Gupta, O. Kayode, and A. S. Tosun, "Access control model for Google cloud IoT," in *Proc. IEEE IEEE 6th Int. Conf. Big Data Secur. Cloud (BigDataSecurity) Int. Conf. High Perform. Smart Comput., (HPSC) IEEE Int. Conf. Intell. Data Secur. (IDS)*, May 2020, pp. 198–208.

[9] L. Martignoni, R. Paleari, and D. Bruschi, "A framework for behavior-based malware analysis in the cloud," in *Proc. Int. Conf. Inf. Syst. Secur.* Berlin, Germany: Springer, 2009, pp. 178–192.

[10] S. K. Cha, I. Moraru, J. Jang, J. Truelove, D. Brumley, and D. G. Andersen, "SplitScreen: Enabling efficient, distributed malware detection," *J. Commun. Netw.*, vol. 13, no. 2, pp. 187–200, Apr. 2011.

[11] T. Y. Win, H. Tianfield, and Q. Mair, "Detection of malware and kernel-level rootkits in cloud computing environments," in *Proc. IEEE 2nd Int. Conf. Cyber Secur. Cloud Comput.*, Nov. 2015, pp. 295–300.

[12] M. K. Gupta, S. Shaw, and S. Chakraborty, "Pattern based malware detection technique in cloud architecture," Inst. Eng. Manage., Kolkata, India, 2016.

[13] N. Rakotondravony, B. Taubmann, W. Mandarawi, E. Weishäupl, P. Xu, B. Kolosnjaji, M. Protsenko, H. de Meer, and H. P. Reiser, "Classifying malware attacks in IaaS cloud environments," *J. Cloud Comput.*, vol. 6, no. 1, pp. 1–12, Dec. 2017.

[14] H. Sun, X. Wang, R. Buyya, and J. Su, "CloudEyes: Cloud-based malware detection with reversible sketch for resource-constrained Internet of Things (IoT) devices," *Softw., Pract. Exper.*, vol. 47, no. 3, pp. 421–441, Mar. 2017.

[15] N. M. Babu and G. Murali, "Malware detection for multi cloud servers using intermediate monitoring server," in *Proc. Int. Conf. Energy, Commun., Data Analytics Soft Comput. (ICECDS)*, Aug. 2017, pp. 3609–3612.

[16] L. Xiao, Y. Li, X. Huang, and X. Du, "Cloud-based malware detection game for mobile devices with offloading," *IEEE Trans. Mobile Comput.*, vol. 16, no. 10, pp. 2742–2750, Oct. 2017.

[17] M. Abdelsalam, R. Krishnan, Y. Huang, and R. Sandhu, "Malware detection in cloud infrastructures using convolutional neural networks," in *Proc. IEEE 11th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2018, pp. 162–169.

[18] Q. K. A. Mirza, I. Awan, and M. Younas, "CloudIntell: An intelligent malware detection system," *Future Gener. Comput. Syst.*, vol. 86, pp. 1042–1053, Sep. 2018.

[19] Q. K. A. Mirza, I. Awan, and M. Younas, "A cloud-based energy efficient hosting model for malware detection framework," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–6.
[20] S. Shen, L. Huang, H. Zhou, S. Yu, E. Fan, and Q. Cao, "Multistage signaling game-based optimal detection strategies for suppressing malware diffusion in fog-cloud-based IoT networks," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1043–1054, Apr. 2018.
[21] W. Zhou and B. Yu, "A cloud-assisted malware detection and suppression framework for wireless multimedia system in IoT based on dynamic differential game," *China Commun.*, vol. 15, no. 2, pp. 209–223, Feb. 2018.
[22] R. M. Yadav, "Effective analysis of malware detection in cloud computing," *Comput. Secur.*, vol. 83, pp. 14–21, Jun. 2019.
[23] P. Indirapriyadarsini, M. U. Mohiuddin, M. Taqueeuddin, C. S. Reddy, and T. Koushik, "Malware detection using machine learning and cloud computing," *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 8, no. 6, pp. 101–104, Jun. 2020.
[24] D. Deyannis, E. Papadogiannaki, G. Kalivianakis, G. Vasiliadis, and S. Ioannidis, "TrustAV: Practical and privacy preserving malware analysis in the cloud," in *Proc. 10th ACM Conf. Data Appl. Secur. Privacy*, Mar. 2020, pp. 39–48.
[25] Ö. Aslan, R. Samet, and Ö. Ö. Tanriöver, "Using a subtractive center behavioral model to detect malware," *Secur. Commun. Netw.*, vol. 2020, pp. 1–17, Feb. 2020.
[26] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: A statistical view of boosting (With discussion and a rejoinder by the authors)," *Ann. Statist.*, vol. 28, no. 2, pp. 337–407, Apr. 2000.
[27] Das Malwerk. *Malware Downloading Website*. Accessed: Jan. 15, 2021. [Online]. Available: https://dasmalwerk.eu/
[28] MalwareBazaar. *Malware Downloading Website*. Accessed: Jan. 15, 2021. [Online]. Available: https://bazaar.abuse.ch/
[29] TheZoo Aka. *Malware Downloading Website*. Accessed: Jan. 15, 2021. [Online]. Available: https://thezoo.morirt.com/
[30] Malwarebenchmark. *Malware Downloading Website*. Accessed: Jan. 15, 2021. [Online]. Available: http://malwarebenchmark.org/
[31] Malshare. *Malware Downloading Website*. Accessed: Jan. 15, 2021. [Online]. Available: https://malshare.com/
[32] Tekdefense. *Malware Downloading Website*. Accessed: Jan. 15, 2021. [Online]. Available: http://www.tekdefense.com/downloads/
[33] Virussign. *Malware Downloading Website*. Accessed: Jan. 15, 2021. [Online]. Available: https://virussign.com/
[34] Virusshare. *Malware Downloading Website*. Accessed: Jan. 15, 2021. [Online]. Available: https://virusshare.com/
[35] Kernelmode. *Malware Downloading Website*. Accessed: Jan. 15, 2021. [Online]. Available: https://www.kernelmode.info/forum/
[36] VirusTotal. *Online Malware Detection Scanner*. Accessed: Jan. 19, 2021. [Online]. Available: https://www.virustotal.com/
[37] *Classification of Malware With PE Headers (ClaMP)*. Malware Dataset. Accessed: Feb. 23, 2021. [Online]. Available: https://github.com/urwithajit9/ClaMP
[38] B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, "Graph-based malware detection using dynamic analysis," *J. Comput. Virol.*, vol. 7, no. 4, pp. 247–258, Nov. 2011.
[39] M. Chandramohan, H. B. K. Tan, L. C. Briand, L. K. Shar, and B. M. Padmanabhuni, "A scalable approach for malware detection through bounded feature space behavior modeling," in *Proc. 28th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2013, pp. 312–322.
[40] S. Das, Y. Liu, W. Zhang, and M. Chandramohan, "Semantics-based online malware detection: Towards efficient real-time protection against malware," *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 2, pp. 289–302, Feb. 2016.
[41] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu, "Context-aware, adaptive, and scalable Android malware detection through online learning," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 1, no. 3, pp. 157–175, Jun. 2017.
[42] J. Jeon, J. H. Park, and Y.-S. Jeong, "Dynamic analysis for IoT malware detection with convolution neural network model," *IEEE Access*, vol. 8, pp. 96899–96911, Jun. 2020.
[43] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection," *Inf. Sci.*, vol. 231, pp. 64–82, May 2013.
[44] M. Yousefi-Azar, L. G. C. Hamey, V. Varadharajan, and S. Chen, "Malytics: A malware detection scheme," *IEEE Access*, vol. 6, pp. 49418–49431, 2018.
[45] R. Kumar, K. Sethi, N. Prajapati, R. R. Rout, and P. Bera, "Machine learning based malware detection in cloud environment using clustering approach," in *Proc. 11th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT)*, Jul. 2020, pp. 1–7.
[46] N. A. Azeez, O. E. Odufuwa, S. Misra, J. Oluranti, and R. Damaševičius, "Windows PE malware detection using ensemble learning," *Inform., Multidisciplinary Digit. Publishing Inst.*, vol. 8, no. 1, pp. 1–10, Feb. 2021.

ÖMER ASLAN received the B.Sc. degree from the Computer Engineering Department, University of Trakya, Turkey, in 2009, the M.Sc. degree in information security from The University of Texas at San Antonio, USA, in 2014, and the Ph.D. degree in cyber security field from the University of Ankara, Turkey, in 2020. He is currently a Ph.D. Researcher with the Computer Engineering Department, University of Siirt, Turkey. He is working on computer systems, information security, cyber security, malware analysis, cloud computing, and the IoT device security. He has published several articles on international journals and conferences. He has been also serving as a reviewer in some prestigious journals.

MERVE OZKAN-OKAY received the B.S. and M.S. degrees in computer engineering from Ankara University, in 2014 and 2016, respectively, where she is currently pursuing the Ph.D. degree with the Department of Computer Engineering. She is also a Research Assistant with the Department of Computer Engineering, Ankara University. She has published several articles on international journals and conferences. Her current research interests include cyber security, cloud-based systems, machine learning, and image processing.

DEEPTI GUPTA received the B.S. and M.S. degrees in mathematics from India, the M.Tech. degree in computer engineering from India, and the M.S. degree in computer science from The University of Texas at San Antonio (UTSA), where she is currently pursuing the Ph.D. degree in computer science. She has worked as an Adjunct Faculty with the Department of Computer Science, St. Edward University, Austin, TX, USA. Her primary area of research interests include security and privacy in cloud computing and the Internet of Things (IoT), anomaly detection models, access control models, game theory, and deep learning assisted security solutions. She is also interested in machine learning-based malware analysis. She has also served as a reviewer and a committee member in conferences.