*Journal of **AI** and **D**ata **M**ining*
Published online:

Assessment Methodology for Anomaly-Based Intrusion Detection in Cloud Computing

Mohsen Rezvani*

Faculty of Computer Engineering, Shahrood University of Technology, Shahrood, Iran

Received 24 April 2017; Revised 17 June 2017; Accepted 05 November 2017
*Corresponding author: mrezvani@shahroodut.ac.ir (M. Rezvani)

Abstract

Cloud computing has become an attractive target for attackers as the mainstream technologies in the cloud, such as the virtualization and multitenancy, permitting multiple users to utilize the same physical resource, and thereby posing the so-called problem of internal facing security. Moreover, the traditional network-based intrusion detection systems (IDSs) are ineffective to be deployed in cloud environments. This is because such IDSs employ only the network information in their detection engines, and this, therefore, makes them ineffective for the cloud-specific vulnerabilities. In this paper, we propose a novel assessment methodology for anomaly-based IDSs in cloud computing that takes into account both the network and system-level information for generating the evaluation dataset. Our approach deploys the IDS sensors in each virtual machine to create a cooperative environment for our anomaly detection engine. The proposed assessment methodology is then deployed in a testbed cloud environment to generate an IDS dataset, which includes both network and system-level features. Finally, we evaluate the performance of several machine learning algorithms over the generated dataset. Our experimental results demonstrate that the proposed IDS assessment approach is effective for attack detection in the cloud, as most of the algorithms are able to identify the attacks with a high level of accuracy.

Key words. *Intrusion detection system, cloud computing, classification algorithm, anomaly detection, dataset generation, IDS assessment, machine learning.*

1. Introduction

The advances in cloud computing have provided significant benefits and computing power based on the idea of pay as you go, which far exceeds that contained in their physical worlds. The cloud computing can provide three models of services: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS); and four deployments: private, community, public, and hybrid clouds [2]. In SaaS, systems offer complete online applications that can be directly executed by their users; in IaaS, providers allow their customers to have access to entire virtual machines; and in SaaS, it offers development and deployment tools, languages, and APIs used to build, deploy, and run applications in the cloud. The security of cloud services is a major obstacle to the growth of this such technology that requires some new considerations regarding the new security issues [4]. Moving the organizations' sensitive information into a public cloud entails additional security risks such as availability, confidentiality, and integrity of those organizations [5].

Although the recent developments have brought new technologies in network security, such as firewalls and security gateways, to prevent unpermitted traffic, IDSs still have a significant role in network security solutions [1, 9]. The main objective of an IDS is to detect all intrusions regarding the deployment location of the sensor component of such security system. IDSs can be categorized from different aspects. Depending on the detection engine, IDSs can be divided into three categories: misuse, anomaly, and hybrid detection systems. The misuse detection systems can only detect known attacks using some pattern matching algorithms and a list of predefined attack signatures. However, an anomaly-based system employs a machine learning technique to model the normal behavior of the system, and then detects the attacks based on their difference from the normal behavior model. Machine learning and data mining algorithms are frequently used in anomaly IDSs [9]. A hybrid IDS employs both the misuse and anomaly detection engines.

The evaluation of an IDS in misuse engine is straightforward based on the attack patterns in a signature database. However, the assessment of accuracy in an anomaly detection engine depends on the environment and methods of generating training and validating datasets. Moreover, the datasets are of great importance in such IDSs. This is because the accuracy of a learning technique directly depends on the training and validation datasets that are generated in a specific environment. Therefore, having datasets specifically generated for an environment can significantly improve the performance of its anomaly IDS. This is due to a better modeling of the normal behavior of the environment.

One of the most widely used datasets is the KDD 1999 dataset [17], which was generated from the DARPA 1998 TCP/IP data [18], and has been used for the KDD Cup

challenge since 1999. Following the issues provided by two main critics of DARPA program [19, 21] and the requirement presented by NIST [22], there are a few research works aimed to either clean or improve the KDD dataset [32] or create new datasets for evaluating IDSs [27]. None of these works were specifically conducted for cloud computing environments, while cloud users face security threats both from outside and inside the cloud. The virtualization and multi-tenancy in cloud permits various users to utilize the same physical resource and poses the new problem of internal facing security [33]. In other words, cloud users must be protected from other users, and also cloud providers need to be protected against data leakage and denial-of-service (DoS) attacks from users. Thus the traditional anomaly IDSs generated based on the above datasets (which were obtained from attacks over general-purpose networks) are ineffective for cloud computing. Furthermore, it is crucial to generate cloud-specific datasets for both training and testing of machine learning algorithms that improve the performance of IDSs for the cloud.

In order to protect the cloud environment from the above attacks, we propose a novel assessment methodology based on a novel anomaly IDS model for cloud computing. In the proposed methodology, both training and testing datasets are generated within a cloud computing environment, and several machine learning algorithms will be applied to the datasets in order to choose the most effective learning model for our proposed anomaly IDS in the cloud. In contrast to the traditional anomaly IDSs, the proposed approach links network and system-level data for generating the datasets. Moreover, deploying IDS sensors in each virtual machine helps us to develop a cooperative anomaly detection engine for the cloud environment. We summarize our contributions as follows:

- We propose a novel methodology for generating benign and attack traffic in the cloud, which can be used as a basis to develop an anomaly IDS in the cloud.
- We deploy the proposed methodology in a testbed cloud environment to generate an IDS dataset that includes both network and system-level features.
- We provide an extensive and detailed analysis of the generated traffic in a testbed cloud.
- We perform some experiments to assess the performance of several machine learning algorithms over the generated datasets.

The rest of this paper is organized as what follows. In Section 2, the related works are discussed. Section 3 presents the details of our proposed methodology for IDS assessment in the cloud. Performance analysis and experimental results are presented in Section 4. Finally, concluding remarks are made in Section 5.

2. Related works

Due to the existing several different machine learning and data mining techniques applicable in IDSs, there are many research activities proposed to contribute an anomaly-based IDS [1, 8, 9, 10, 15, 16]. Kumar et al. in [16] have divided the artificial intelligent based IDSs into four categories: decision tree, data mining, machine learning, and ensemble classifiers. In the taxonomy presented in [8], authors have classified anomaly detection techniques in six categories including statistical, classification-based, clustering and outlier-based, soft computing, knowledge-based, and combination learners. Two recent published surveys of IDSs based on ensemble and hybrid techniques are presented in [1, 10].

In 1998, DARPA (Defence Advanced Research Project Agency) sponsored the first Intrusion Detection Evaluation (IDEVAL) project in the MIT Lincoln Labs [18]. The traffic was collected by the tcpdump network sniffer on a simulated network similar to a military topology. The training data was generated from seven weeks of the simulated traffic including 24 attack types, while the test dataset was obtained from another two-week traffic collection containing additional 14 attack types. The result of the DARAPA project was reviewed in 1999 for creating a widely used benchmark tool for IDSs. Another famous dataset derived from the project entitled KDD Cup 1999 was generated originally by Stolfo et al. [17] by extracting the flows from the DARPA traffic. In addition, DARPA initiated the LARIAT (Lincoln Adaptable Real-time Assurance Test-bed) project in 2001 whose results were restricted to the military USA environment [26].

Two main contributions provided several questions about the accuracy of the DARPA simulations including research works of McHugh in [21] and Mahoney and Chan in [19], while NIST provided the major requirements of evaluation of IDSs in [22]. Following the issues provided by the critics of DARPA program and the requirements presented by NIST, several research works were conducted to create new datasets for evaluating IDSs [6]. Authors in [7] also presented an assessment methodology based on real traffic, while they performed several signature-based IDSs for labelling generated datasets. Massicotte et al. [20] have proposed a framework for the automatic evaluation of IDSs based on virtual infrastructure and vulnerability exploitation programs for generating attack traffic, while they have no support for the benign traffic generation. Shiravi et al. [27] have proposed a systematic approach to generate the datasets required for IDSs. This method is based upon the concept of profiles that contain detailed descriptions of intrusions and abstract distribution models for applications, protocols, or lower level network entities.

Privacy is a significant issue for using real traffic in an IDS assessment method. In order to resolve the challenge, some researchers proposed a synthetic traffic generation based on modelling real behaviour and network services. Authors in [14] have provided a benchmark framework for IDSs based on modelling the normal behaviour from observation of the last traffics and generating a synthetic data based on the model. Sommers et al. [30] have proposed a framework named Trident for IDS evaluation based on both benign and malicious traffic generation. For generating the benign traffic and with respect the parameters of mix and volume in traffic, the authors created a protocol-aware emulation based on payload interleaving. This mechanism is based upon a collection of automata with states that describe classes of packet observed in a specific service. Moreover, the idea was implemented as a plug-in for Harpoon traffic generator [28]. For generat-

ing the attack traffic, the authors use MACE [29] as an attack composition framework with some extensions on its attack profiles. To the best of our knowledge, no existing work considers the IDS assessment methodology for cloud computing environments.

3. Cloud-IDS assessment methodology

3.1. IDS assessment framework

The DARPA project [18] and requirements presented by NIST [22] provide some general steps for an assessment framework in anomaly IDSs. Having these recommendations, our IDS assessment framework is also inspired by the research work conducted in [27]. We use similar steps for generating datasets, while our framework resolves a few shortcomings in those studies for packet grooming and labelling. Moreover, our experiment environment is adapted for cloud computing environments. Thus our approach needs to collect the system-level data, and links them to the network traffic in order to consider the cloud specific attacks lead to cloud resource exhausting and originated from the virtualization and multi-tenancy in the cloud computing. Finally, we generate the dataset in a collaborative context, where several virtual machines along with the hypervisor in the cloud environment are involved in the traffic generation. This helps to capture traffic required to detect the attacks originated within the cloud from one tenant to another one.
Figure 1 illustrates the main steps in our framework for creating IDS assessment datasets. The following describes the main steps in our framework.

- **Benign traffic generation:** To collect the benign traffic, we select the real world traffic for several sample applications we developed and deployed on our cloud computing environment.
- **Attack traffic generation:** The attack traffic is generated using several IDS stimulator tools, where their attack signatures are also updated for covering the recent attacks reported in the cloud computing.
- **Traffic sniffer:** This is a network sniffer that collects and stores all the packets related to the machines involved in the experiments.
- **SysPerf collector:** This module collects and stores some system-level data related to the performance of the machines involved in the experiments.
- **Feature extraction:** The features extracted in this module are divided into two categories: network and system-level features. For the network-level features, we first extract the network sessions and aggregate the packets within the sessions. After that, we extract the several features from the network sessions related to both header and payload of the packets within each session. For the system-level features, we focus on the features related to the system performance such as the memory and CPU usage. We explain the details of the feature extraction process in Section 3.4.
- **Labeling:** A significant step in every IDS assessment methodology is to label the generated dataset. It is to be noted that we may have both controlled and uncontrolled traffic as we employ real cloud environment for running our assessment methodology. The former contains the traffic we know its exact label as the traffic is synthetically generated in a controlled environment. The uncontrolled traffic is generated during the live operations of the cloud computing environment, and we need a mechanism to label such a traffic. In this step, we employ more than one signature-based IDSs to label uncontrolled traffic, and then the label is obtained by performing a majority voting algorithm over the reports of IDSs. Obviously, the controlled traffic is manually labeled.
- **Feature selection:** An important step for an anomaly IDS is to decide which features should be included in the final datasets considered as an input for learning algorithms. It is clear that there are some features extracted in the previous step and have no contribution to distinguishing the records within the datasets. We explain the details of the feature selection process in Section 3.5.
- **Learning experiments:** For evaluating the performance measures of our generated datasets, we apply several machine learning algorithms on the dataset. The performance results of these algorithms are reported in Section 4.

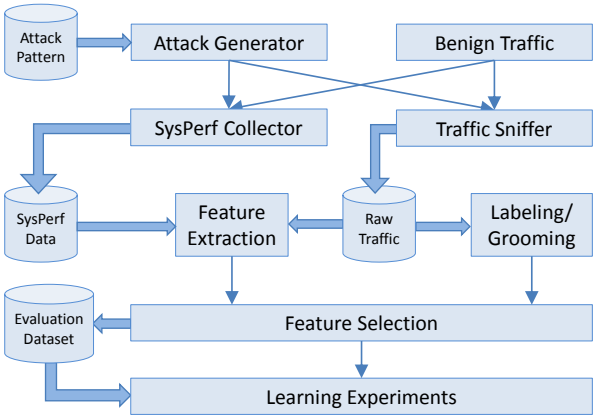


Figure 1 . Our framework for generating datasets.

3.2. Deployment in cloud

In order to determine the deployment plan for an IDS in the cloud computing, we need to understand the major points of the cloud-specific vulnerabilities. This is due to the fact that the deployment points of the IDS sensors have significant impacts on the coverage of the data collection using the sensors as well as the detection performance of IDS. Regarding the attack model and attack scenarios presented in the literature for the cloud computing [4, 33], there are three significant roles in attacks of the cloud: User, Service, and Cloud. In other words, we need to consider all the above three roles to propose a comprehensive IDS in the cloud. Thus, we deploy the IDS sensors in both virtual machines and the hypervisor of the cloud that helps to collect both environmental and network data among these three roles.
Figure 2 shows the logical topology of our test environment in the cloud computing for generating IDS dataset.

In this topology, two virtual machines run Linux as the hosted operating systems and were deployed on a hypervisor Xen as the guest operating system. Two sample web applications are installed on the virtual machines, while the data collection tools are installed on sensor modules in virtual machines and hypervisor. The first virtual machine hosts a sample web-based survey application developed using J2EE standard, Tomcat as the web application server, and MySQL as DBMS. Also the second virtual machine hosts a sample car reservation web application developed with Apache, PHP, and MySQL platform.

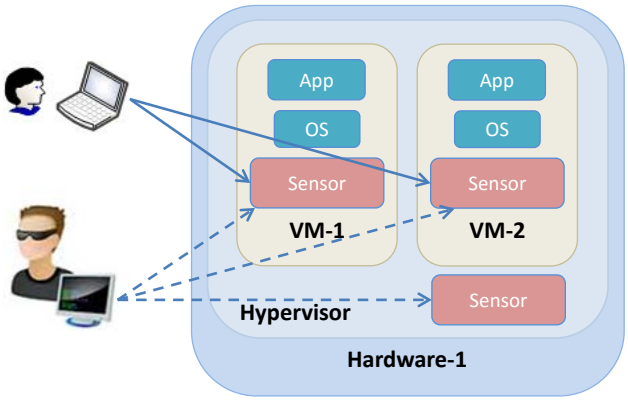


Figure 2 . Logical topology of experiments for collecting IDS datasets.

3.3. Implementation

In order to generate the dataset using the methodology presented in Figure 1 and deployment plan presented in Figure 2, we perform the methodology in two phases after the installation of the modules in the plan. In the first phase, our objective is to generate the benign dataset. To this end, we make the topology in a controlled environment and request users to work with the services deployed in the topology. At the same time, we collect both traffic and system level data inside each sensor as well as the hypervisor. To this end, we use some tools such as *tcpdump* for packet sniffer, *ntop* for network usage, *sysstat* and *vmstat* for CPU usage, free and *vmstat* for memory usage, and *df* for disk usage.

In the second phase of our implementation plan, we aim to generate the attack dataset. For generating the attack traffic, we use three famous attack generator tools including Mucus [23] and MACE [29], which generate the attack traffic based on the Snort [25] signatures and *hping* [31] that is a network tool widely used for packet injection.

As a result of the above two phases, we have two datasets, generated from the attack and benign traffic, separately. Although we generate this two traffic in separate time slots, there is a possibility of overlapping between them. For example, some benign traffic may be generated by some legitimate users during the time we perform the second phase. Moreover, there is a low chance of having attack traffic by some attackers at the time we perform the first phase. Both of these two scenarios may happen as the cloud computing environment we use is a public cloud and it is publicly available on the Internet. Thus there is no guarantee that the traffic is generated in a fully controlled environment.

In order to obtain the true label for both benign and attack traffic generated in the above two phases, we replay the traffic as an input for three IDSs including Snort, Bro [24], and a commercial IDS. Then, the true label of each session within the traffic is obtained by performing a majority voting algorithm over the labels obtained from these three IDSs.

3.4. Feature Extraction

As described earlier, an important contribution of our assessment methodology is to link the network features to the performance features such as system-level attributes in each virtual machine. Thus we divide the features extracted in our methodology into two categories: network and system performance features.

The network features are extracted based on network sessions such as the TCP/UDP/ICMP session. Thus we first separate the network sessions from the generated traffic and store the packets of each session within a separate *pcap* file. Then we develop a feature extraction tool for extracting the feature values from each pcap file. We divide the network features into the following categories:

- Intrinsic features of a session that are related to the header data of the packets in a network session. We extract 22 features in this category, reported in Table 1.
- Content-based features that are extracted from the data part of the packets in a network session. We extract 18 features in this category, reported in Table 2.
- Time-based features include some statistical features computed in a specific time frame. Note that all features in this category are obtained from the packet headers. Table 3 reports 5 features extracted in this category.
- Host-based features include some statistical features computed for each host address involved in the traffic. Note that all features in this category are obtained from the packet headers. Table 4 reports 6 features extracted in this category.

It is worth noting that the last two categories in the network based features are mostly related to the throughput of the network which obtained based on both the time and various hosts. The main contribution of these two categories of features is to improve the detection performance in the case of resource exhausting attacks that are very prevalent in the cloud computing environments [4, 11, 33]. The system performance features are extracted from some system-calls in the virtual machines. These features are divided into three categories, including CPU, memory, and IO related features, which contains totally 15 features, listed in Table 5. Note that we need to link the network features to the system performance features. Therefore, we capture the system performance features at the time we collect the benign and attack traffic. This helps us to use the exact capturing time to link these two types of features in order to create the final dataset.

3.5. Feature selection

In this work, we studied the contribution of the extracted features in the previous step using Principal Component

Table 1 . Intrinsic features of network connections.

Feature	Description
duration	connection duration
src_port	source port
dst_port	destination port
protocol	protocol type: tcp, udp and icmp
icmp_type	icmp type
icmp_code	icmp code
frames_ctos	# of frames, source to destination
bytes_ctos	# of data bytes, source to destination
frames_stoc	# of frames, destination to source
bytes_stoc	# of data bytes, destination to source
pkts_no	# of packets in the connection
bytes_no	# of bytes in the connection
flow	incoming or outgoing traffic
data_bit_rate	data bit rate
avg_pkt_size	mean packet size
avg_pkt_rate	mean packet rate
str_time_order	stream time order (true;false)
land	true for a land attack; false otherwise
wrong_fragment	# of wrong fragments
urgent	# of urgent packets
tcp_errors	# of tcp errors
icmp_checks_bad	# of icmp bad checksum

Table 2 . Content-based features of network connections.

Feature	Description
textbinfiletype	text; binary; unknown
execfiletype	exec; noexec; unknown
lines	# of lines
words	# of whole words
maxwordlen	len of the largest word
hotwords	# of hot words
characters	# of characters
alpha	% of alpha letters
lalpha	% of lower case letters
ualpha	% of upper case letters
digits	% of digits
xdigits	% of hexadecimal digits
visibles	% of visible characters
printables	% of printable characters
puncts	% of punctuation and symbols
blanks	% of blank characters
spaces	% of space characters
controls	% of control characters (0x00-0x7F)

Table 3 . Time features of network connections.

Feature	Description
count	# of sessions to the same dst address
srv_count	# of sessions to the same dst port
same_srv_rate	% of sessions to the same service
diff_srv_rate	% of sessions to different services
srv_diff_hrate	% of sessions to different hosts

Table 4 . Host features of network connections.

Feature	Description
dh_count	# of sessions to the same dst address
dh_srv_count	# of sessions to the same dst port
dh_same_srv	% of sessions to the same service
dh_diff_srv	% of sessions to different services
dh_same_sport	% of sessions to the same source port
dh_srv_diff_hst	% of sessions to different dst machines

Table 5 . Features related to system performance.

Feature	Description
cpu_user	CPU %user
cpu_system	CPU %system
cpu_iowait	CPU %iowait
cpu_idle	CPU %idle
memory_used	memory used
mem_frmpg_psec	memory frmpg_psec
mem_bufpg_psec	memory bufpg_psec
mem_campg_psec	memory campg_psec
mem_swpused	memory swpused
mem_swpcad	memory swpcad
io_tps	IO tps
io_rtps	IO rtps
io_wtps	IO wtps
io_bread	IO bread
io_bwrtn	IO bwrtn

Analysis (PCA), which determines the most important features. PCA is a simple yet popular and useful linear transformation technique that ranks the components by importance [3]. To this end, it takes dataset from one coordinate system into another such that the greatest variance by any projection of the data comes to lie on the first coordinate (called the first principal), the second greatest on the second coordinate, and so on. We employed the PCA technique to determine the most important features extracted in the previous step of our methodology. The features selected in this step contribute to the most variance for all records in the datasets and those features that contribute to the least variance for correlated records. In this section, we briefly discuss how PCA is performed on our generated dataset to reduce its dimensions. More details about PCA can be found in [13]. Note that some of the features in our dataset is nominal, and we need to transform the nominal features into a numerical format before running the PCA algorithm.

Let us assume that we have a dataset with n records and m features (dimensions). In this dataset, the ith record stores the m values as a vector x_i points to $\mathbb{R}^m$. We can store the mean of all m features as a single vector in $\mathbb{R}^m$ as follows:

$$\mu = \frac{1}{n}(x_1 + \cdots + x_n). \quad (1)$$

Now by subtracting the mean μ from each sample vector x_i, the data is transformed so that the mean becomes zero. To this end, let B indicate the $m \times n$ matrix whose ith column is $x_i - \mu$:

$$B = [x_1 - \mu | \ldots | x_n - \mu]. \quad (2)$$

Now we define the covariance matrix S, an $m \times m$ matrix, as:

$$S = \frac{1}{n-1} BB^T. \quad (3)$$

It is worth noting that the ith entry on the diagonal of S is the variance of the ith feature in the dataset. The matrix is symmetric, and it thus can be orthogonally diagonalized. Consequently, we can compute the eigenvalues of S, denoted as $\lambda_1, \ldots, \lambda_m$ with corresponding orthogonal eigenvectors $u_1, \ldots, u_m$, which are called *principle components* of the dataset. Now we define the *variance fraction* of a principal component u_i as $\frac{\lambda_i}{\lambda_1 + \cdots + \lambda_m}$. Such variance fraction indicates the significant level of the ith feature in the dataset. Now we select the principal components from the largest variance fraction until achieving an acceptable level of significance. Using the above equations, Algorithm 1 shows a brief description of the PCA algorithm used for the feature selection over our generated dataset.

Algorithm 1 Feature selection using PCA.

```
procedure PCA
    Compute μ using Equation(1).
    Compute B using Equation(2).
    Compute S using Equation(3).
    Compute the eigenvalues λ_1, ..., λ_m of S.
    Arrange the eigenvalues in decreasing order.
    Compute the eigenvectors u_1, ..., u_m of S.
    repeat
        Select the largest principle component.
        Update the selected variance fractions.
    until the selected variance fraction is accepted
end procedure
```

4. Experimental evaluation

In this section, we present the results of our experiments in order to evaluate the effectiveness and efficiency of our approach.

4.1. Implementation and experimental environment

We performed our dataset generation phases on a cloud environment running OpenStack[1]. We used the SplitCap[2] tool to separate the network sessions from the collected traffic. We also implemented our feature extraction tools on a Linux platform, mostly written in the Linux shell and Python scripts.

The experiments were conducted on our generated dataset at two parts. In the first part, we employed the PCA technique to select the most significant features in the dataset. This helped us to generate a new dataset including only the features selected using PCA. In the experiments, for PCA, we used ratio α as 99.9%, as it was most desirable based on our experimental results.

In the second part, we conducted an experiment for evaluating various machine learning algorithms and reporting the performance and effectiveness of these algorithms over the dataset generated using our assessment methodology. To this end, we used Weka that has been widely used in machine learning performance studies [12].

All experiments were conducted on an off-the-shelf computer with 2.00GHz Intel Core 2 Duo processor and 4GB RAM running.

4.2. Traffic generation

As shown in Figure 1, the first and second phases of our assessment methodology are to generate the benign and attack traffic, respectively. In this section, we describe the infrastructure for generation of these two real-life network intrusion datasets using our testbed cloud computing. This testbed is an IaaS cloud platform we established using the OpenStack[3] architecture. OpenStack is an open-source cloud operating system that controls large pools of computing, storage, and networking resources, all managed through a dashboard that gives administrators control, while empowering their users to provision resources through a web interface.

Figure 3 shows the virtual network topology in our cloud testbed. All network elements including servers, workstations, and switches were deployed virtually on the top of the OpenStak system. As one can see in this figure, we deployed the servers and workstations into three separate zones. We also allocated separate and invalid IP address ranges for each zone in order to effectively decrease the broadcast domain for each element. The network address translation (NAT) server acts as an access provider to the Internet for the local and server zones. The NAT server also provides access to the public servers in the DMZ server for all users on the Internet. To this end, the NAT server connects to the Internet through multiple valid IP addresses that are used for connecting the workstations to the Internet as well as making the DMZ servers visible for the Internet users. It also provides firewall facilities to block unauthorized access from the Internet to local and server zones.

The capturing server in our testbed cloud provides means to non-disruptively collect both system-level and traffic on the servers deployed in the server zone. Since all the servers are virtually deployed on the cloud, we installed several agents on each virtual server to collect the required information. Thus all the capturing activities are performed locally on the servers, and the collected information will be manually merged on the capturing server.

We configured 6 servers within the server zone of the testbed cloud. On each server, we installed Ubuntu Server 14.04 with different network services including FTP server, Samba service, Apache server, SQL server, PostgreSQL server, WebLogic server with Java 2 platform, enterprise edition (J2EE) technologies, PHP development

[1] http://www.openstack.org

[2] https://www.netresec.com/?page=SplitCap

[3] https://www.openstack.org/

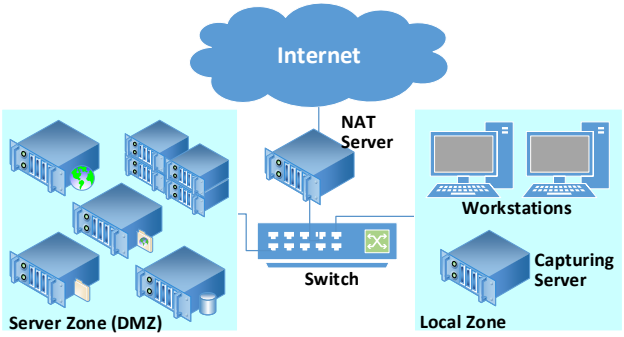


Figure 3 . Virtual network topology for our testbed cloud.

tool, Telnet, and SSH servers. Table 6 summarizes the servers and their services employed in our testbed. For all servers in the server zone, we used an *m.xlarge* instance type, and for the remaining machines (NAT and capturing servers and two workstations), we used *m.large* instance type in our cloud testbed with specification shown in Table 7.

Table 6 . Servers and their services installed on our testbed. Installed operating system for all servers is Ubuntu Server 14.04.

Row	Server	Services
1	Web server1	Apache2, PHP5
2	Web server2	WebLogic 12cR1
3	DBMS server1	MySQL 5.5
4	DBMS server2	PostgreSQL 9.2
5	FTP server	VsFTPD 3.0.2
6	TelnetSSH server	telnet-server-0.17 openSSH 7.2p

Table 7 . Instance types in our testbed cloud.

Instance	Processor	vCPU	Memory	Storage
m.large	64-bit	2	8GB	40GB SSD
m.xlarge	64-bit	4	16GB	80GB SSD

In order to generate the real time normal traffic, we employed around 10 persons on the Internet to normally use our services provided on the servers in DMZ. The capturing process was conducted during three separate time slots, each of which lasted for half an hour while services were accessible for the Internet users. Finally, we merged all the captured data in order to generate the benign dataset. The attack traffic was generated by launching attacks through the workstations within the local zone of the testbed cloud at controlled time slots. During the time slots, we disconnected the server zone from the Internet in order to ensure that there was no benign traffic arrived in the zone. This helped us to increase the accuracy of our labeling procedure, described in Section 3. We employed three famous attack generator tools including Mucus [23] and MACE [29] *hping* [31], which automatically generate network traffic using the signatures of Snort.

4.3. Dataset description

The raw data contains traffic generated in our assessment methodology in two phases: benign and attack traffic generation. After generating the traffic, we extracted the features from the packet traces, and the final dataset contained one record for each network session specified with values for each feature.

In the dataset, each network session was labeled as either normal or as an attack. The network connection data contained 67 features, among which 59 were numeric and 8 were symbolic. Only 58 features were used in the experiments, which were selected in our feature selection phase. Each connection in the dataset was thus transformed into a 58-dimensional vector as data input for detection engine. There were 7,827 session records in the dataset, of which, 6,444 were benign and 1,383 were attacks. The traffic was collected from 62 different hosts (IP addresses), and the whole size of the captured traffic was around 450MB.

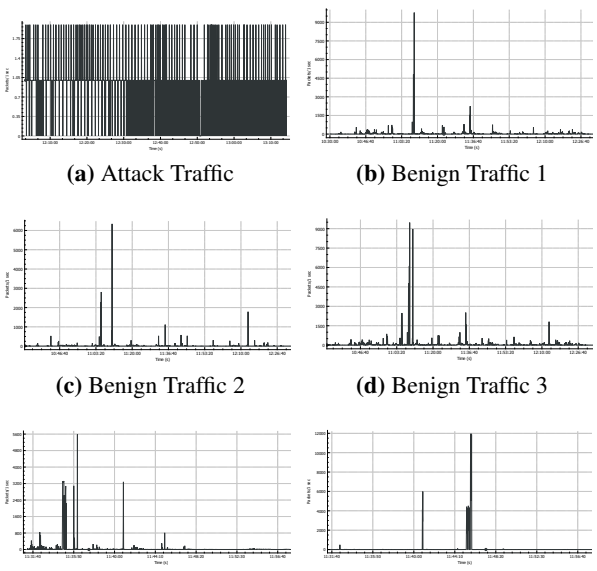
Figure 4 depicts the trend of packet capturing in different time slots. As one can see in this figure, we captured the attack traffic in one-time slot and the benign traffic in five different time slots. Figure 5 illustrates the arrangement of protocols in various layers in the network stack protocol observed in the captured traffic. Clearly, we employed a wide range of protocols for collecting the benign and attack traffic. As mentioned in the previous Section, several protocols were chosen to be served in our testbed cloud including HTTP, HTTPS, SSH, TELNET, and FTP. The remaining protocols shown in Figure 5, such as PING, DNS, and ARP, were generated as an indirect consequence of deploying the above main protocols in the cloud. After extracting the features from the generated traffic, we employed the Weka tool to analyze the distribution of the class values based on some other features in the dataset, as shown in Figure 6.

4.4. Feature selection

In this section, we studied the impact of the feature selection on anomaly detection. To this end, we used the PCA technique implemented in the Weka tool. Consequently, we found nine features with very low rank, close to zero, including icmp_type, icmp_code, wrong_fragment, exec-filetype, memory_swpcad, urgent, icmp_checksum_bad, memory_swpused, and land. More details about these features are presented in Section 3.4. Clearly, these features were eliminated from our base dataset for evaluating the anomaly detection algorithms.

4.5. Evaluation of anomaly detection

In this section, we evaluated both the effectiveness and efficiency of different machine learning algorithms for attack detection in the cloud using the dataset generated through our assessment methodology. To this end, we selected six machine learning algorithms, one in each category implemented in Weka including: *NaiveBayes*, *SVM*,

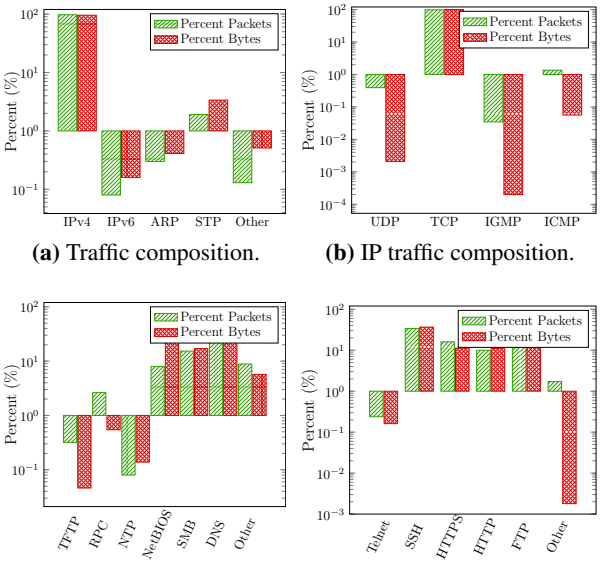


(a) Attack Traffic **(b)** Benign Traffic 1

(c) Benign Traffic 2 **(d)** Benign Traffic 3

(e) Benign Traffic 4 **(f)** Benign Traffic 5

Figure 4 . Throughput of packets seen within capture periods.



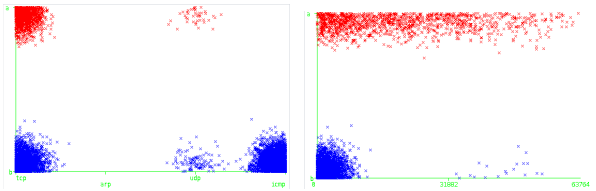
(a) Traffic composition. **(b)** IP traffic composition.

(c) UDP traffic composition. **(d)** TCP traffic composition.

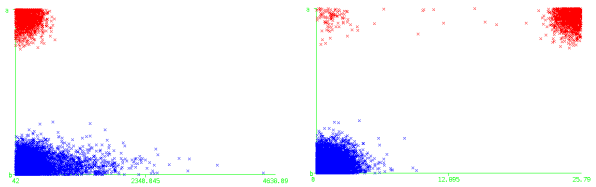
Figure 5 . Composition of protocols within generated traffic.

IBk, *DecisionTable*, *J48* and *RandomForest*, which are widely employed in the intrusion detection literature.

The classification experiments were performed on the generated dataset, employing the k-Fold cross-validation. Since there was only one dataset generated in our methodology, we needed an idea of splitting the data into two datasets: the training dataset was used for training the classification algorithm, and the remaining data (the validation dataset) were used for evaluating the performance of the algorithm. In k-fold cross-validation, the dataset was first partitioned into k equally (or nearly equally) sized segments (folds). Subsequently, k iterations of training and validation were performed such that within each iteration a different fold of the data was held-out for validation, while the remaining $k-1$ folds were used for learning. Figure 7 shows an example with $k = 4$. The lighter parts of the dataset were used for training while the darker parts were used for validation. In all the experiments, we set k to 10 for the cross-validation technique.



(a) Protocol and class. **(b)** Destination port and class.



(c) Mean packet size and class. **(d)** CPU usage in user level and class.

Figure 6 . distribution of some features and class feature.

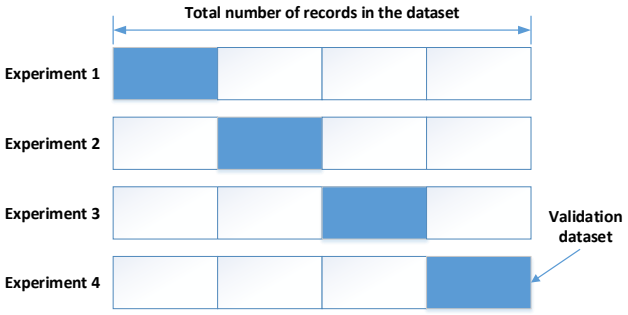


Figure 7 . Procedure of 4-fold cross-validation.

The classification performance of the machine learning algorithms over our generated dataset was evaluated by their relative absolute error, root relative squared error, true positive rate (TPR), false positive rate (FPR), true negative rate (TNR), false negative rate (FNR), and the area under ROC (Receiver Operating Characteristic) for each experimental scenario. These measurements were calculated based on a confusion matrix shown in Table 8. This matrix offers a detailed picture on the actual and predicted classification task done by any classification approach. The detailed computations of the performance measurements are as follows:

$$TPR = \frac{TP}{TP+FN} \times 100 \quad (4)$$

$$TPR = \frac{FP}{FP+TN} \times 100 \quad (5)$$

$$TPR = \frac{TN}{FP+TN} \times 100 \quad (6)$$

$$TPR = \frac{FN}{FN+TP} \times 100 \quad (7)$$

Table 8 . Confusion matrix for detecting risky flows. (TP: True Positive; FN: False Negative; FP: False Positive; TN: True Negative)

Actual Class	Predicted Class	
	Attack	Benign
Attack	TP	FN
Benign	FP	TN

The relative absolute error (RAE) and root relative squared error (RRSE) were computed using equations (8) and (9), respectively. In these two equations, $\hat{\theta}_i$ is the predicted value of the i-th sample and θ_i is the corresponding true value.

$$RAE = \frac{\sum_{i=1}^{N} |\hat{\theta}_i - \theta_i|}{\sum_{i=1}^{N} |\bar{\theta} - \theta_i|} \tag{8}$$

$$RRSE = \sqrt{\frac{\sum_{i=1}^{N} \left(\hat{\theta}_i - \theta_i\right)^2}{\sum_{i=1}^{N} \left(\bar{\theta} - \theta_i\right)^2}} \tag{9}$$

In order to evaluate the classification algorithms, we first obtained the confusion matrix, and then computed the above metrics using the matrix. Table 9 reports the classification performance of the different learning algorithms for attack detection over our dataset when we used the cross-validation technique by setting $k = 10$. One can see in this table that all algorithms perform a perfect classification with accuracy close to 100%. Such results lead us to conclude that the system-level features have a significant effect on detection performance of the cloud-based attacks.

Figure 8 illustrates the training and testing time required by different machine learning algorithms. The elapsed times reported in this figure were collected from the output reports of Weka when it performed various machine learning algorithms with the cross-validation technique over the generated dataset. The results reported in this figure along with the performance of the classification algorithms showed in Table 9 give us a better understanding of the capabilities of the machine learning algorithms to accurately and timely classify the malicious traffic using our generated dataset for cloud computing environments.

4.6. Evaluation by splitting dataset

In this section, we evaluated the effectiveness of different machine learning algorithms for attack detection in the cloud by splitting our dataset into two parts: training and validating datasets. To this end, we randomly split the dataset into training and validating sets by considering 66% for the training set. After that, the experiments described in the previous section were repeated to obtain the performance metrics. we run these steps 10 times and then averaged the results.

Table 10 reports the classification performance of the different learning algorithms for attack detection over our dataset when we split the dataset into training and validating sets. One can see in this table that most of the algorithms show a slightly higher error comparing to the previous experiment over the cross-validation. This can be explained by the fact that splitting the dataset by considering 66% training leads to generate a less accurate model in most of the learning algorithm in general. Figure 9 illustrates the elapsed time of different machine learning algorithms using splitting the dataset into training and validating sets. The results reported in this figure show that the algorithms present an efficiency similar to the previous experiments over the cross-validation.

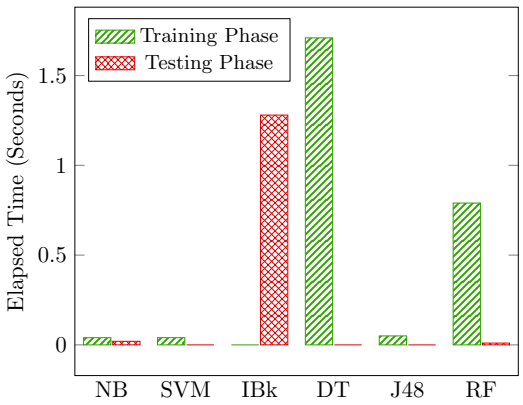


Figure 8 . Elapsed time for training and testing phases in different machine learning algorithms. (NB: NaiveBayes, DT: DecisionTable, RF: RandomForest.)

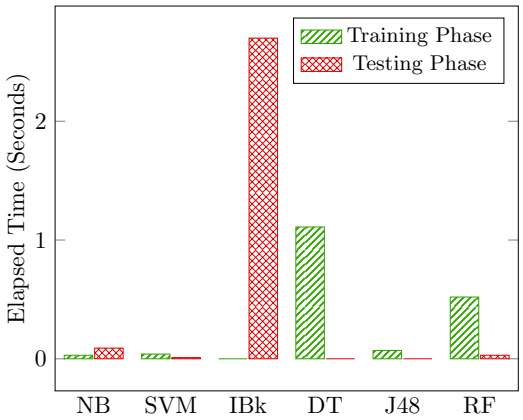


Figure 9 . Elapsed time of different machine learning algorithms by splitting dataset to training and validating sets. (NB: NaiveBayes, DT: DecisionTable, RF: RandomForest.)

5. Conclusions

In this paper, we proposed a new methodology for assessing the anomaly-based IDSs for cloud computing. To this end, we combined the network and system-level information for feature extraction in the dataset generation phase of our methodology. We then deployed the IDS sensors in several virtual machines, to capture the network flows as well as the system information for all modules in the cloud. Accordingly, we generated a dataset including such features for a lab-scale cloud environment, and then evaluated the performance of several machine learning algorithms using the generated dataset. The experimental

Table 9 . Classification performance for attack detection using cross validation and different learning algorithms over our dataset.

	NaiveBayes	SVM	IBk	DecisionTable	J48	RandomForest
Relative absolute error	0.1	0	0.07	0.1	0	0.2
Root relative squared error	1.98	0	0.87	0.1	0	1.61
True positive rate	1	1	1	1	1	1
False positive rate	0.001	0	0	0	0	0
True negative rate	1	1	1	1	1	1
False negative rate	0	0	0	0	0	0
Area under ROC	0.999	1	0.999	1	1	1

Table 10 . Classification performance for attack detection using splitting our dataset and different learning algorithms.

	NaiveBayes	SVM	IBk	DecisionTable	J48	RandomForest
Relative absolute error	0.00	0.14	0.13	0.00	0.00	0.30
Root relative squared error	0.00	2.98	0.13	0.00	0.00	2.41
True positive rate	1.00	1.00	1.00	1.00	1.00	1.00
False positive rate	0.00	0.00	0.00	0.00	0.00	0.00
True negative rate	1.00	1.00	1.00	1.00	1.00	1.00
False negative rate	0.00	0.00	0.00	0.00	0.00	0.00
Area under ROC	1.00	1.00	1.00	1.00	1.00	1.00

results show that our assessment methodology is highly effective for attack detection in the cloud. In the future, we plan to deploy our assessment methodology in a real-world cloud environment and generate a reference dataset that can be used by researchers in the intrusion detection context.

References

[1] Aburomman, A. A., & Reaz, M. B. I. (2017). A survey of intrusion detection systems based on ensemble and hybrid classifiers. *Computers & Security*, *65*, 135–152.

[2] Aceto, G., Botta, A., de Donato, W., & Pescapè, A. (2013). Cloud monitoring: A survey. *Computer Networks*, *57*(9), 2093 - 2115.

[3] Ahmadi Livani, M., Alikhany, M., Yadollahzadeh Tabari, M., et al. (2013). Outlier detection in wireless sensor networks using distributed principal component analysis. *Journal of AI and Data Mining*, *1*(1), 1–11.

[4] Aikat, J., Akella, A., Chase, J. S., Juels, A., Reiter, M. K., Ristenpart, T., ... Swift, M. (2017). Rethinking security in the era of cloud computing. *IEEE Security Privacy*, *15*(3), 60-69.

[5] Ali, M., Khan, S. U., & Vasilakos, A. V. (2015). Security in cloud computing: Opportunities and challenges. *Information Sciences*, *305*, 357–383.

[6] Athanasiades, N., Abler, R., Levine, J., Owen, H., & Riley, G. (2003). Intrusion detection testing and benchmarking methodologies..

[7] Bermúdez-Edo, M., Salazar-Hernández, R., Díaz-Verdejo, J. E., & García-Teodoro, P. (2006). Proposals on assessment environments for anomaly-based network intrusion detection systems. In *Proceedings of the first international conference on critical information infrastructures security* (Vol. 4347, pp. 210–221).

[8] Bhuyan, M. H., Bhattacharyya, D. K., & Kalita, J. K. (2014). Network anomaly detection: methods, systems and tools. *IEEE communications surveys & tutorials*, *16*(1), 303–336.

[9] Buczak, A. L., & Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, *18*(2), 1153–1176.

[10] Folino, G., & Sabatino, P. (2016). Ensemble based collaborative and distributed intrusion detection systems: A survey. *Journal of Network and Computer Applications*, *66*, 1–16.

[11] Grobauer, B., Walloschek, T., & Stocker, E. (2011, March). Understanding cloud computing vulnerabilities. *IEEE Security and Privacy*, *9*(2), 50–57.

[12] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009, November). The WEKA data mining software: An update. *SIGKDD Explor. Newsl.*, *11*(1), 10–18.

[13] Jauregui, J. (2012, August). *Principal component analysis with linear algebra.*

[14] Kayacik, G. H., & Zincir-Heywood, N. A. (2005). Analysis of three intrusion detection system benchmark datasets using machine learning algorithms. In *Proceedings of the 2005 ieee international conference on intelligence and security informatics* (pp. 362–367).

[15] Keshtgary, M., Rikhtegar, N., et al. (2018). Intrusion detection based on a novel hybrid learning approach. *Journal of AI and Data Mining*, *6*(1), 157–162.

[16] Kumar, G., Kumar, K., & Sachdeva, M. (2010). The use of artificial intelligence based techniques for intrusion detection: a review. *Artificial Intelligence Review*, *34*, 369-387.

[17] Lichman, M. (2013). *UCI machine learning repository.* Retrieved from `http://archive.ics.uci.edu/ml`

[18] Lippmann, R., Haines, J. W., Fried, D. J., Korba, J., & Das, K. (2000, October). The 1999 DARPA off-line intrusion detection evaluation. *Comput. Netw.*, *34*(4), 579–595.

[19] Mahoney, M. V., & Chan, P. K. (2003). An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection. In *In proceedings of the sixth international symposium on recent advances in intrusion detection* (pp. 220–237). Springer-Verlag.

[20] Massicotte, F., Gagnon, F., Labiche, Y., Briand, L., & Couture, M. (2006, dec.). Automatic evaluation of intrusion detection systems. In *Computer security applications conference, 2006. acsac '06. 22nd annual* (p. 361 -370). doi:

[21] McHugh, J. (2000). The 1998 lincoln laboratory IDS evaluation. In *Recent advances in intrusion detection* (Vol. 1907, p. 145-161). Springer Berlin Heidelberg.

[22] Mell, P., Hu, V., Lippmann, R., Haines, J., & Zissman, M. (2003, June). *An overview of issues in testing intrusion detection systems.*

[23] Mutz, D., Vigna, G., & Kemmerer, R. (2003). An Experience Developing an IDS Stimulator for the Black-Box Testing of Network Intrusion Detection Systems. In *Proceedings of the annual computer security applications conference.*

[24] Paxson, V. (1999, December). Bro: a system for detecting network intruders in real-time. *Comput. Netw.*, *31*(23-24), 2435–2463.

[25] Roesch, M. (1999). Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th usenix conference on system administration* (pp. 229–238).

[26] Rossey, L., Cunningham, R., Fried, D., Rabek, J., Lippmann, R., Haines, J., & Zissman, M. (2002). LARIAT: Lincoln adaptable real-time information assurance testbed. In *Aerospace conference proceedings, 2002. ieee* (Vol. 6, p. 2671-2676).

[27] Shiravi, A., Shiravi, H., Tavallaee, M., & Ghorbani, A. A. (2012, May). Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers and Security*, *31*(3), 357–374.

[28] Sommers, J., Kim, H., & Barford, P. (2004, June). Harpoon: a flow-level traffic generator for router and network tests. *SIGMETRICS Perform. Eval. Rev.*, *32*(1), 392–392.

[29] Sommers, J., Yegneswaran, V., & Barford, P. (2004). A framework for malicious workload generation. In *Proceedings of the 4th acm sigcomm conference on internet measurement* (pp. 82–87).

[30] Sommers, J., Yegneswaran, V., & Barford, P. (2005). *Toward comprehensive traffic generation for online IDS evaluation* (Tech. Rep.). University of Wisconsin-Madison.

[31] Stanger, J. (2009). Security testing with hping. *Linux Magazine*, 38–41. `http://www.linux-magazine.com/content/download/62147/484014/file/038-041_hping.pdf`.

[32] Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. In *Proceedings of the second ieee international conference on computational intelligence for security and defense applications* (pp. 53–58).

[33] Zhai, Y., Yin, L., Chase, J., Ristenpart, T., & Swift, M. (2016). CQSTR: Securing cross-tenant applications with cloud containers. In *Proceedings of the seventh acm symposium on cloud computing* (pp. 223–236).