Classifying Malicious Thread Behavior in PaaS Web Services

Cemile Diler Özdemir[1,2], Mehmet Tahir Sandıkkaya[2] and Yusuf Yaslan[2]
[1]*Corporate Technology, Development Center, Siemens AS, Istanbul, Turkey*
[2]*Computer Engineering Department, Istanbul Technical University, Istanbul, Turkey*

Keywords: Cloud Security, PaaS, Malicious Behavior, Machine Learning.

Abstract: Multitenant structure of PaaS cloud delivery model allows customers to share the platform resources in the cloud. However, this structure requires a strong security mechanism that isolates customer applications to prevent interference between different applications. In this paper, a malicious thread behavior detection framework using machine learning algorithms is proposed to classify whether user requests are malicious. The framework uses thread metrics of worker threads and N-Gram frequencies of operations as its features. Test results are evaluated on a real-life scenario using Random Forest, Adaboost and Bagging ensemble learning algorithms and evaluated using different accuracy metrics. It is found that the malicious request detection accuracy of the proposed system is 87.6%.

1 INTRODUCTION

Cloud computing is a popular concept for companies to be enabled on-demand network access for outsourcing their resources as infrastructure, platform or software with the minimum effort. Cisco reports that cloud data center workloads will tripled from 2015 to 2020 (Networking, 2017). Considering this growth, cloud computing vendors focus on security research to adapt rapid development of this technology (Banerjee et al., 2013).

Most of the PaaS providers offer web application platforms to their customers because PaaS development is web oriented. This strategy is beneficial both for the PaaS providers and for the PaaS customers. Since scripting languages (Ruby and Python) and virtualized platforms (Java and .NET) are commonly used in web development in recent years, providers build their servers on these popular technologies. In addition, cloud customers quickly customize their existing web applications and deliver them to the providers to be served in the cloud. Thus deploying many customers' applications turns into an easy and cheap process for PaaS providers. Thereby, the common benefit is rapid adoption to the cloud.

These advantages, however, come with a major flaw. Different cloud customers share PaaS platform resources (hardware, software, services, configuration, etc.) and it requires isolation between customer applications to prevent interference between different applications. This interference can occur unconsciously or maliciously. For instance a faulty application can consume most of memory or CPU on the provided platform for many customers. Other customers are influenced; even there is no conscious attack to platform. In addition it is possible that maliciously acting customers can execute code to attack other customers or platform. Availability, confidentiality and integrity of PaaS are threaten for these reasons (Modi et al., 2013). PaaS providers need a strong security mechanism to protect and isolate their customer applications and the platform.

Currently, PaaS customers are limited to web applications due to leading PaaS providers Google[1], Heroku[2] and Amazon[3]. The providers may limit customer applications' access to trivial resources such as files or sockets via carefully set up permissions. However, memory and CPU are shared among multiple threads in web applications as well as per-request user behavior cannot be traced (Sandıkkaya et al., 2014).

Several systems have already been proposed for cloud security, such as host based intrusion detection systems (Arshad et al., 2012), network based intrusion detection systems (Hamad and Al-Hoby, 2012), distributed intrusion detection systems (Sanjay Ram, 2012) and hypervisor based intrusion detection systems (Garfinkel et al., 2003). However, these men-

[1]https://cloud.google.com/appengine/
[2]https://www.heroku.com/
[3]https://aws.amazon.com/elasticbeanstalk/

Özdemir, C., Sandıkkaya, M. and Yaslan, Y.
Classifying Malicious Thread Behavior in PaaS Web Services.
DOI: 10.5220/0006688204180425
In *Proceedings of the 8th International Conference on Cloud Computing and Services Science (CLOSER 2018)*, pages 418-425
ISBN: 978-989-758-295-0

tioned systems were designed to run on operation system, system virtual machine or hypervisor level. They do not consider isolation of several different customer applications hosted in the same process virtual machine. In addition, only application and data layers are manageable by customer in PaaS service model. Base layers are managed by cloud provider. Figure 1 presents PaaS service model and its layers. The customers cannot manage the underlying cloud infrastructure including network, operating systems or servers in PaaS model.

Hardware and software levels isolation mechanisms have also been utilized in PaaS. Software level mechanisms isolate threads, processes or virtual machines of different tenants (Bazm et al., 2017). Heroku uses container-based isolation which groups operating system processes by kernel name-spaces and resource allocations to isolate from other groups. Docker[4] is one of the most popular open-source container platform provider which has been adopted by many PaaS providers. In addition, Cloud Foundry[5] isolates its tenants using user-based isolation mechanism. It is a traditional and widely used technique that each application runs as a different user on the operating system. However, sharing the same process virtual machine environment by multiple tenants needs runtime-based isolation mechanisms (Zhang et al., 2014).

In this paper a runtime-based security framework is proposed for multitenant PaaS providers to detect malicious behavior of threads using machine learning. The main contributions of this paper are summarized as follows:

- *Thread behavior detection framework* : Thread behavior is detected with the proposed framework and this framework could be integrated into cloud customer web application with the minimum effort. The framework is designed for PaaS providers which have many customer web applications in the same application server. In this deployment scenario, customers' web applications reside on the same operating system process. Within this process, worker threads serve web applications. Proposed framework measures, classifies and detects malicious behavior using worker threads' resource usage metrics.
- *Well selected metrics*: All necessary metrics are measurable on web application level. Measurements of the features are independent from operation system, programming language or cloud provider. Therefore, features can be collected without any dependency. Also features are privacy friendly, so that they do not contain any sensitive data of the cloud providers or the cloud customers.

[4]https://www.docker.com/

[5]https://www.cloudfoundry.org/

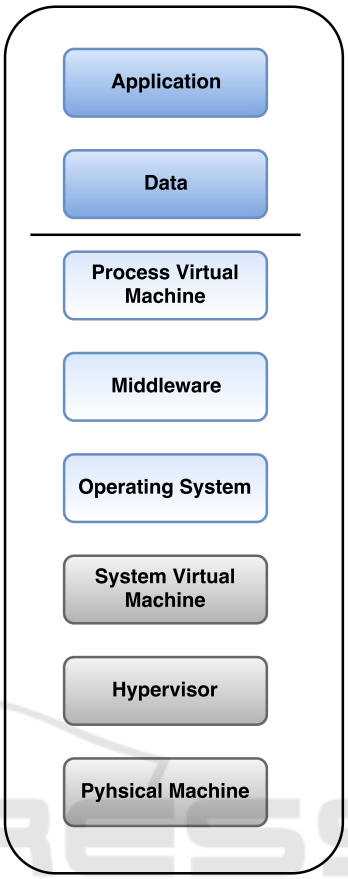

Figure 1: PaaS deployment model: The physical computer resides a hypervisor to monitor the operating systems through system virtual machines. This level of abstraction is mostly known as IaaS. On top of operating systems, many process virtual machine instances could be run. Each of these process virtual machines (E.g. JVM) may isolate an application, but not the threads within the same application. This level of abstraction is mostly known as PaaS. The process virtual machine may be configured as a web application server and the threads may belong to different web applications.

The main difference of the proposed framework from intrusion detection systems is, it does not monitor the network activity or any feature from the network connection; but directly focuses on the running thread. On the other hand; the framework's main difference from a virus (malware) scanner is, it detects malicious threads rather than files. Moreover, this detection is done throughout considering access to the critical OS resources without inspecting the whole code sequence of the thread.

The rest of this paper is organized as follows: Sec-

tion 2 gives the background of the subject. In Section 3, the proposed security mechanism is described. Section 4 presents how the experimental setup is organized. Section 5 describes feature set and classification algorithms applied to the data. The results of the experiments are presented in Section 6. Finally, section 7 concludes the paper and presents a discussion on the results.

2 RELATED WORK

In the literature there is a vast amount of research on malicious behavior detection systems that use machine learning algorithms (Fan et al., 2016), (Modi et al., 2013). Most of the existing works deal with intrusion detection systems and they try to classify the malicious communications to avoid the external attacks on perimeterized computer networks of an organization. These models generally extract the feature vector from data, packets user input command sequences, log files, low-level system information and CPU/memory usage (Wu and Banzhaf, 2010). However, these intrusion detection systems differ from the proposed mechanism as they always assume a trusted perimeter to be protected. As a result, they hardly detect malicious activity that originates from the insiders. Note that, in PaaS deployments, the providers are willingly to accept the customers as tenants. Then, malicious activities may originate from these tenants or tenants' users. Therefore, an intrusion detection system cannot detect malicious user behavior after the request is once accepted to access internal resources. In practice, an intrusion detection system may exist independent of the proposed mechanism and controlled by the PaaS provider to protect the whole PaaS deployment against third party attackers; e.g. denial-of-service attackers.

Application program interface (API) calls and machine instructions are mostly used features of intelligent malware detection systems (Bazrafshan et al., 2013). Pirscoveanu et al. (Pirscoveanu et al., 2015) used sequence, frequency and count of the windows platform system calls as main features to classify with Random Forest algorithm. Several malwares can be dynamically classified in parallel using this approach. Fan et al. (Fan et al., 2016) also used API calls as features of their Malicious Sequential Pattern Malware Detection (MSPMD) framework. They applied modified Generalized Sequential Pattern algorithm for sequence mining with All-Nearest-Neighbor classifier to Windows Portable Executable (PE) samples. Uppal et al. (Uppal et al., 2014) applied N-Gram algorithm to extract features from API sequences. Shabtai et al. (Shabtai et al., 2012) utilized machine instruction data and extracted features with N-Gram pattern from opcodes. Several classification algorithms are applied to extracted feature vector to detect unknown malicious codes.

N-Gram algorithm is also used in web prediction models. Su et al. (Su et al., 2000) utilized N-Gram model to predict future request of users. This probabilistic prediction model aims to make best guesses on the users' next actions based on previous actions. Although they do not focus on user's malicious behavior, it is foreseen that N-Gram prediction model is adaptable to predict malicious behavior of users. In addition, N-Gram is applied to API calls and machine instructions data in previous security studies (Uppal et al., 2014), (Shabtai et al., 2012). However, they can provide only system level cloud detection and prevention mechanisms. In addition, collection of the mentioned data is a time and resource consuming process. Moreover, one may argue that, privacy of PaaS user is not considered during data collection. Proposed mechanism observes only thread behavior and collects processor usage and requested operation sequences on runtime. Finally framework classifies malicious behavior of PaaS user using collected feature vector.

Despite the need of malicious thread execution detection in PaaS cloud and mentioned malware detection techniques' favorable results; a malicious thread behavior detection framework is proposed in PaaS cloud using machine learning algorithms.

3 PROPOSED MECHANISM

The proposed mechanism covers a wide range of scenarios offered in the current PaaS ecosystem. A PaaS system provides its computational power to its customers through threads, and these customers can reside in the same process virtual machine. In that case, there is a risk of inference between different customers in addition to the PaaS provider resources. In such an adversarial model, the aim of the proposed method is not completely isolate cloud customers' access to the platform resources. The aim of the proposed model is to determine if the cloud customer is acting maliciously. This malicious act can occur consciously or unconsciously. The maliciously acting threads can be stopped or at least kept away from accessing more resources right after they are classified as malicious by the proposed security framework.

It is assumed that, a PaaS customer's cloud application reside in a JVM and deployed the proposed framework. The framework focuses on web applica-

tion users' request based analysis. Fundamentally, the mechanism distinctively analyses each worker thread per request of each user, so even capable of classify one-time unconscious malicious activity of a trustworthy user.

The framework runs in the PaaS provider side, together with PaaS customers' cloud applications. This framework monitors thread behavior and collected information is analyzed offline to train a classifier that is used for real-time decision making.

Java Management Extensions (JMX) is utilized to measure processor shares, memory and average time consumption for a user request. In addition, access to each PaaS resource is wrapped in a separate method which are entitled as checkpoints. Checkpoints are defined in enter and exit of resource consuming system methods and are identified using aspects (Kiczales et al., 1997) because of its numerous advantages. First, time consumption of each resource access can be collected per-request without affecting privacy of the customers. The only leaked data is the sequence of the resource access and the time consumption in each resource. Second, the checkpoints are programmed to disrupt the execution of a thread if its behavior after it is classified as malicious by the framework. This is beneficial because the threads cannot request any more resources if they are classified as malicious.

Proposed mechanism detects maliciously acting threads on the cloud platform using machine learning techniques. First, classification features are selected to be measured by proposed framework. Instant CPU usage and cumulative CPU usage per request are two attributes of the feature vector and measured by JMX. Moreover, feature vector contains three more attributes per request. These attributes are resource access duration, resource access type and resource access sequence. Access type feature is mapped to CRUD (create, read, update, delete) functions and contains information about requested function. Sequence feature holds order of requested functions. It is so informative to have sequence of these operations to obtain frequency of the operation and transaction between each operation.

Proposed mechanism processes feature vector using N-Gram algorithm. N-Gram represents a contiguous sequence of N items from a given list of items and predicts the next item. In natural language processing these items can be letters or words, in speech recognition items can be phonemes and in malware analysis they can be system calls or machine instruction sequence. Operation sequence is one the feature in proposed mechanism and set of operations is represented as $O = \{Add, Delete, Read, Update\}$. Table 1 represents sample tokenized operation sequences data and their types to visualize structure of the train data. This sequence data is represented as string and these tokens are converted into a set of new attributes using Weka NGramTokenizer API with the values $N_{min} = 1$ and $N_{max} = 5$. After this filtering process, new feature vector has 132 new attributes according to occurrence frequency of the transactions between each operation from the sequence text data. Operation transactions are visualized in Figure 3.

Classification algorithms are applied to training data after feature vector measured and tokenized with the N-Gram. In order to detect malicious thread, different classification algorithms have been evaluated. After the comparison of test results, Random Forest classifier is integrated into proposed framework as the most accurate classifier.

Proposed framework uses runtime observation to classify a request. Figure 2 shows classification process of runtime observations. Proposed framework is integrated into an experimental cloud web application which is explained in the next section. This behavior based system does not require a signature database and framework can be integrated into a cloud application with the minimum effort.

4 EXPERIMENTAL SETUP

The proposed mechanism is tested on a demo cloud system that contains an event ticketing application connected to a relational database. Conventional paths of usage are recorded and repeated with Apache JMeter to reproduce a set of regular requests. Realistic attack scenarios are considered and also added to the query set of JMeter. Experimental ticketing cloud application is composed of basic user operations that may be mapped to CRUD (create, read, update, delete) operations on a database. These four main operations are: add, delete, read, and update. Depending on the payload of the request an event, a user, a ticket may be added, read, updated or deleted from the application. The application also includes many meta elements such as text, graphics, audio, and video. Training set contains add, delete, read and update functions either as a regular or a malicious operation. Regular requests are defined as common user behavior depending on cloud application's scope and goal. On the other hand, malicious operations are selected from a wide set of possibilities. An unexpected content may be added to the database, whole table may be dropped, a large set of bogus data may be inserted, etc. The attack scenarios are produced by considering cross-site scripting, SQL-injection, database modification and file system access scenarios.

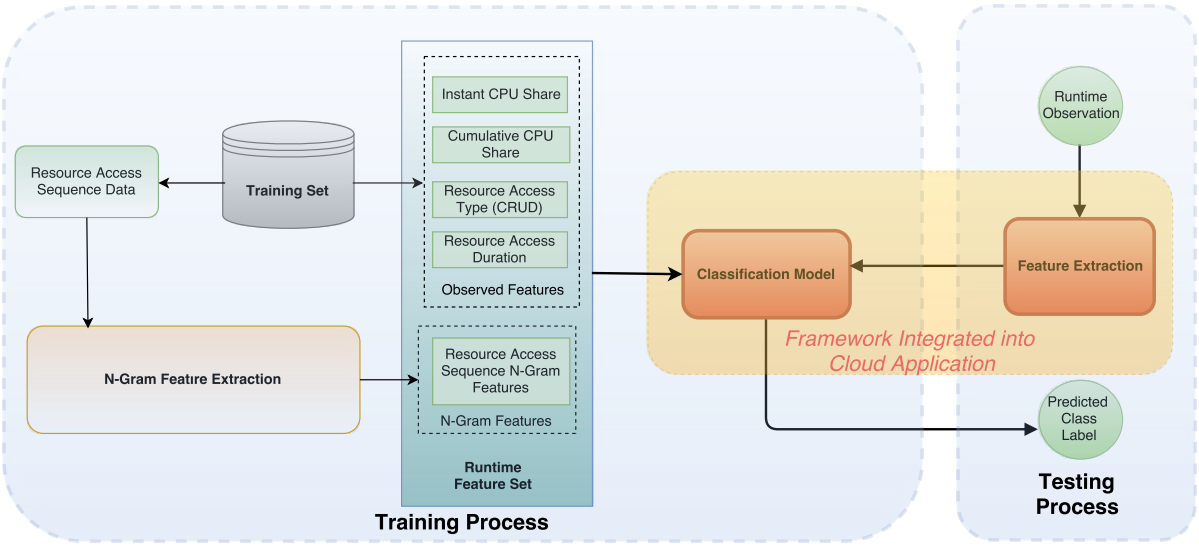

Figure 2: The overall architecture of the proposed framework for malicious thread behavior detection. The framework extracts four features from any PaaS web service deployment in a cost-effective and privacy-friendly way; instant CPU share of a thread in the web service, CPU share of a thread over a period of time, a thread's access type (create, read, update, delete) to critical resources (databases, files, sockets, etc.) and finally the duration of this access. These features are enriched with N-Gram based on access sequence during training phase. During testing, a thread's behavior is classified based on previous knowledge.

The final set contains ten sets that include nearly 1% malicious requests and the total number of queries is 100 000. Each experiment is conducted with 10 000 requests of which nearly 100 of the requests are malicious. Each experiment is repeated 10 times. In the final set, there are 1000 malicious requests and 99 000 regular requests. The results are presented as the average of 10 independent experiments.

5 USED FEATURE SET AND CLASSIFICATION ALGORITHMS

In this paper, N-Gram feature extraction algorithm is applied to feature set and its brief description is given in Subsection 5.1. After the feature extraction, ensemble learning algorithms, Random Forest, Bagging and AdaBoost, are run on the data to evaluate their accuracy in proposed framework. These classification algorithms are described respectively in Subsections 5.2, 5.3 and 5.4.

5.1 N-Gram Features

N-Gram models sequence of n elements that can be letters, words or phonemes. In this paper, N-Gram probabilistic model predicts type of next operation X_i based on previous operation sequence $X_{i-(n-1)}, X_{i-(n-2)}, \ldots X_{i-1}$. Likelihood of next element in the sequence is symbolized as $P(X_i \mid X_{i-(n-1)}, X_{i-(n-2)}, \ldots X_{i-1})$ and it bases on $(n-1)$ order Markov model.

Proposed framework applies Weka API NGramTokenizer filter to collected operation sequence data. An example data is shown in Table 1. N-Gram splits this sequence data with the minimum and maximum grams. It calculates frequencies and transitions of each operation that illustrated in Figure 3. N-Gram model can store more context with larger N. Storing more context provides better prediction but it requires more memory usage and time consumption. However, prediction gets worse with the small N while memory usage and time consumption decrease. Eventually, N is given with the interval of $[1,5]$ that provides best efficiency both prediction accuracy, time consumption and memory usage in proposed framework. After the calculation, NGramTokenizer generates new features vector that contains probability of each grams.

5.2 Random Forest Classifier

Random Forest is an ensemble learning method that grows many random classification or regression trees. Trees vote for the most popular class and the result is the combination of these tree predictors. It runs efficiently on large datasets. Moreover, Random Forest algorithm has randomness in tree construction which

Table 1: Sample data is shown below for N-Gram classification. N-Gram classifies regular or malicious threads only by the sequence of their access types. It does not check which resource threads access or for how long. It also does not check the parameters of the operations. Such information is fed into the classification model by other features.

Operation Sequence	Request Type
Read → Read	Regular
Read → Read → Add → Update	Regular
Delete → Delete → Delete	Malicious
Update	Regular

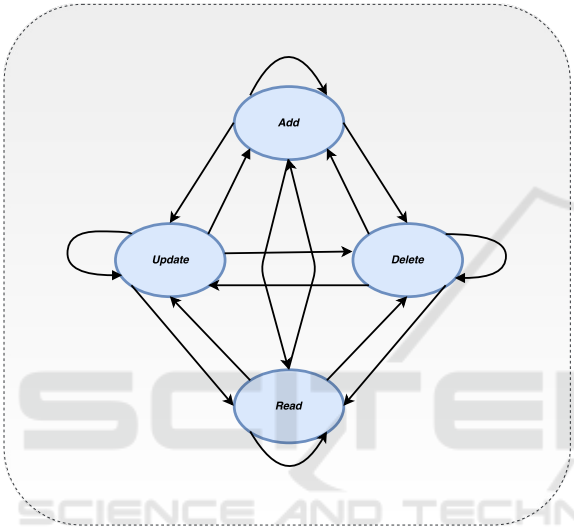

Figure 3: Transitions, $S_t \rightarrow S_{t+1}$, between states. States represent resource access types and transitions can be evaluated with this state machine.

minimizes the correlation. In addition, Random Forest algorithm does not overfit to data (Breiman, 2001). Rapidly increasing cloud data traffic requires memory and time efficient algorithms to run with big data. These characteristics make random forest algorithm applicable to cloud security. The pseudo code of the random forest algorithm is given in Algorithm 1.

5.3 Bagging

Bagging (Mamitsuka et al., 1998), also called Bootstrap Aggregating, is an ensemble technique that uses classifiers trained on instances generated by randomly drawn examples, with replacement. Therefore, each classifier in the ensemble is obtained with a different random sampling of the training dataset. The final decision is given by majority vote over individual classifiers' outputs.

Algorithm 1: Random Forest generates ensemble of trees using randomly selected instances and features.

```
1: procedure RANDOMFOREST
2:     f ← features
3:     N ← number of trees
4:     H ← ∅
5:     for i = 1 to T do
6:         n_i ← bootstrap samples from original data
7:         g_i ← GROWTREE(n_i, f)
8:         H ←H∪g_i
9: procedure GROWTREE(n,f)
10:    f_i ← subset of f
11:    best split among f_i
12:    return tree
```

5.4 AdaBoost

AdaBoost, short for Adaptive Boosting, is a successful boosting algorithm that constructs a strong classifier $H(x)$, as linear combination of weak classifiers $h_t(x)$ shown in Equation 1. Prediction of the class label $H(x)$ is made by calculating the weighted average of the weak predictions $h_t(x)$. The weight, α_t, is based on the classifier's error rate which infers the number of misclassified instances over the training set divided by the training set size.

Previously Adaboost algorithm was used in network intrusion detection because of its low computational complexity, a high detection rate, and a low false-alarm rate (Hu et al., 2008). This algorithm is one of the selected classifier to measure its accuracy in the proposed framework because of these advantages.

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \quad (1)$$

6 EXPERIMENTAL RESULTS

The experimental results are obtained using ten folds cross validation. Results are evaluated using given metrics: incorrectly classified instance percentage, precision, recall and F-measure. These measurements are computed according to True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN) rates given in confusion matrix in Table 2. Incorrectly classified percentage calculation is given in Equation 2, respectively precision, recall and F-Measure computations are given in Equation 3, Equation 4 and Equation 5.

Table 2: Confusion matrix legend.

		Predicted	
		Malicious	Regular
Actual	Malicious	TP	FN
	Regular	FP	TN

Table 3 shows different classifiers' result without resource access sequence data. Operation sequence feature is not evaluated in this experiment. This feature set has just processor usage, resource usage duration and base operation type metrics. Random Forest classification results on this dataset have a large number of incorrectly classified instances. In addition Bagging and AdaBoost algorithms with J48 decision tree base classifier do not obtain better accuracy than Random Forest algorithm.

Table 4 shows classification results with N-Gram feature extraction. Operation sequence features are filtered with the values $N_{min} = 1$ and $N_{max} = 5$ and feature extraction process in framework is shown in Figure 2. After the extraction, new feature vector has attributes processor usage, resource usage, base operation type and generated N-Gram features.

Table 4 has much more accurate results than Table 3 for all classifiers, although the same classification algorithms are run. The number of misclassified instances drops from 888 to 162 after the operation sequence feature is added to framework to be collected and to be processed with N-Gram module.

Random Forest algorithm gets the most accurate and promising result in Table 4. Only 162 instances are labeled incorrectly out of 100000. In addition, only 38 regular instances out of 99000 are classified as malicious. It shows that proposed framework has low false-alarm rate. 124 malicious instances out of 1000 are classified as regular request. Table 5 shows confusion matrix of Random Forest algorithm with N-Gram feature extraction. Precision result given in Table 4 indicates that, a malicious predicted instance is classified correctly with the probability of 0.95 by the proposed framework. In addition, recall result shows a malicious request is detected by framework with the probability of 0.87. Since classes are unbalanced F-Measure is evaluated as success criteria of the framework to inspect if it is close to its best value at 1. As a result, 0.91 F-Measure result indicates that proposed framework is accurate on both malicious and regular classes.

$$Misclassified = \frac{FP+FN}{FP+TP+FN+TN} \quad (2)$$

$$Precision = \frac{TP}{TP+FP} \quad (3)$$

Table 3: Classifiers' result without resource access sequence data.

	Random Forest	Bagging	Ada-Boost
Misclassified %	0.888	0.9111	0.986
Precision	0.6204	0.6164	0.5133
Recall	0.299	0.242	0.291
F-Measure	0.4011	0.3457	0.3690

Table 4: Classifiers' results enhanced with N-Gram feature extraction.

	Random Forest	Bagging	Ada-Boost
Misclassified %	0.162	0.192	0.2
Precision	0.9584	0.9646	0.9381
Recall	0.876	0.839	0.857
F-Measure	0.9153	0.8968	0.8954

$$Recall = \frac{TP}{TP+FN} \quad (4)$$

$$F\text{–}Measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (5)$$

7 DISCUSSION AND CONCLUSION

This paper proposes a malicious thread behavior detection framework for the use of PaaS providers. This approach utilizes machine learning techniques and especially beneficial in the current multitenant PaaS ecosystem as cloud customers may share the resources within the same OS process. The multitenancy requires strong isolation between customer applications. The proposed framework obtains such isolation by monitoring thread behavior and purposes to satisfy cloud customers' need for security and isolation. The proposed framework is deployed in the application level and it can be integrated into any web service application in the PaaS cloud with a standard JVM requirement.

Table 5: Confusion matrix of the proposed framework. The proposed framework runs Random Forest classifier with N-Gram feature extraction into feature set.

		Predicted	
		Malicious	Regular
Actual	Malicious	876	124
	Regular	38	98962

The proposed mechanism investigates several machine learning techniques and combines them. First, N-Gram module filters the operation sequence. The filtered sequence is combined with resource usage metrics. Then, the proposed framework classifies the requests as regular or malicious using the combined measured metrics. This classification module is built on the training data. The Random Forest classifier is able to detect a malicious request with the probability of 0.87 in the proposed framework.

It is obvious that better results can be obtained using proposed framework with the more precise measurement and different metrics in the future. Improved frameworks would be applicable to malicious behavior detection into the PaaS clouds domain in the near future. As a future work, proposed framework scenarios will be extended using different cloud applications and different metrics.

REFERENCES

Arshad, J., Townend, P., and Xu, J. (2012). An abstract model for integrated intrusion detection and severity analysis for clouds. *Cloud Computing Advancements in Design, Implementation, and Technologies*, 1.

Banerjee, C., Kundu, A., Basu, M., Deb, P., Nag, D., and Dattagupta, R. (2013). A service based trust management classifier approach for cloud security. In *Advanced Computing Technologies (ICACT), 2013 15th International Conference on*, pages 1–5. IEEE.

Bazm, M.-M., Lacoste, M., Südholt, M., and Menaud, J.-M. (2017). Side Channels in the Cloud: Isolation Challenges, Attacks, and Countermeasures. working paper or preprint.

Bazrafshan, Z., Hashemi, H., Fard, S. M. H., and Hamzeh, A. (2013). A survey on heuristic malware detection techniques. In *Information and Knowledge Technology (IKT), 2013 5th Conference on*, pages 113–120. IEEE.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Fan, Y., Ye, Y., and Chen, L. (2016). Malicious sequential pattern mining for automatic malware detection. *Expert Systems with Applications*, 52:16–25.

Garfinkel, T., Rosenblum, M., et al. (2003). A virtual machine introspection based architecture for intrusion detection. In *Ndss*, volume 3, pages 191–206.

Hamad, H. and Al-Hoby, M. (2012). Managing intrusion detection as a service in cloud networks. *International Journal of Computer Applications*, 41(1).

Hu, W., Hu, W., and Maybank, S. (2008). Adaboost-based algorithm for network intrusion detection. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(2):577–583.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., and Irwin, J. (1997). Aspect-oriented programming. *ECOOP'97Object-oriented programming*, pages 220–242.

Mamitsuka, N. A. H. et al. (1998). Query learning strategies using boosting and bagging. In *Machine learning: proceedings of the fifteenth international conference (ICML98)*, volume 1.

Modi, C., Patel, D., Borisaniya, B., Patel, H., Patel, A., and Rajarajan, M. (2013). A survey of intrusion detection techniques in cloud. *Journal of Network and Computer Applications*, 36(1):42–57.

Networking, C. V. (2017). Ciscoglobal cloud index: forecast and methodology, 2015-2020. white paper.

Pirscoveanu, R. S., Hansen, S. S., Larsen, T. M., Stevanovic, M., Pedersen, J. M., and Czech, A. (2015). Analysis of malware behavior: Type classification using machine learning. In *Cyber Situational Awareness, Data Analytics and Assessment (CyberSA), 2015 International Conference on*, pages 1–7. IEEE.

Sandıkkaya, M. T., Ödevci, B., and Ovatman, T. (2014). Practical runtime security mechanisms for an apaas cloud. In *Globecom Workshops (GC Wkshps), 2014*, pages 53–58. IEEE.

Sanjay Ram, M. (2012). Secure cloud computing based on mutual intrusion detection system. *International Journal of Computer application*, 1(2):57–67.

Shabtai, A., Moskovitch, R., Feher, C., Dolev, S., and Elovici, Y. (2012). Detecting unknown malicious code by applying classification techniques on opcode patterns. *Security Informatics*, 1(1):1.

Su, Z., Yang, Q., Lu, Y., and Zhang, H. (2000). Whatnext: A prediction system for web requests using n-gram sequence models. In *Web Information Systems Engineering, 2000. Proceedings of the First International Conference on*, volume 1, pages 214–221. IEEE.

Uppal, D., Sinha, R., Mehra, V., and Jain, V. (2014). Malware detection and classification based on extraction of api sequences. In *Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on*, pages 2337–2342. IEEE.

Wu, S. X. and Banzhaf, W. (2010). The use of computational intelligence in intrusion detection systems: A review. *Applied Soft Computing*, 10(1):1–35.

Zhang, Y., Juels, A., Reiter, M. K., and Ristenpart, T. (2014). Cross-tenant side-channel attacks in paas clouds. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 990–1003. ACM.