Log-based intrusion detection for cloud web applications using machine learning

Jaron Fontaine, Chris Kappler, Adnan Shahid, and Eli De Poorter

Abstract With the ongoing rise and ease-of-use of services provided by major cloud providers, many enterprises migrate their infrastructure and applications to cloud platforms. However, the success of using many and diverse services leads to more attack vectors for potential attackers to exploit, that again leads to more difficult, complex and platform-specific security architectures. This paper is an attempt to remedy this problem by proposing simplified cloud security using machine learning techniques. This leads to a more general architecture that uses classifiers such as decision trees and neural networks, trained with data logged by cloud applications. More specifically, we collected easy-to-interpret access logs from multiple web applications which are often the only kind of security information available on various services on such platforms. The results show a more flexible approach, that was experimentally validated on cloud platforms and improve detection speed, using neural networks and J48 decision trees, up to 26-47 times while still maintaining an accuracy of 98.47% and 97.71% respectively.

1 Introduction

Typical enterprises use multiple cloud solution providers for hosting services. Cloud platforms come with advantages such as scalability, faster development, higher cost efficiency with minimal management effort or service provider interaction [3]. Cloud platforms offer a wide range of applications. These include a range of simple web-based services such as web applications, storage, databases, data analytics, etc. They also include platforms required by many computationally intensive software

Jaron Fontaine · Adnan Shahid · Eli De Poorter
Ghent University - imec, IDLab, Department of Information Technology, Belgium, e-mail: {jaron.fontaine,adnan.shahid,eli.depoorter}@ugent.be

Chris Kappler
PwC, e-mail: chris.kappler@pwc.com

such as virtual machines, container services, load balancers, etc. With the increasing popularity of cloud solutions, security becomes more important than ever to ensure data integrity, availability and confidentiality. Users can access private data on such services from everywhere, at any time. This makes it possible for intruders to try accessing or modifying the same resources [18]. Common intrusion detection systems (IDS) can detect very specific well-defined attacks that intrude certain platforms or networks. As more cloud service public endpoints are provided to customers, more attack vectors arise from potential intruders to attack these systems. Deploying multiple platform- and application-specific intrusion detection systems is a challenging problem, however, it is not very flexible and demands domain expertise in how vulnerabilities are exploited and attacks are executed. Complex detection methods on many layers of the system will also degrade system performance while scanning each kind of packet, even safe ones. This paper provides a remedy for this problem by aiming for a broader system to perform attack detection in cloud services using web access logs and machine learning (ML) techniques. In this paper we do not try to implement a full stack detection system, but rather explore the possibilities of ML that can accelerate and increase the flexibility of attack classification. We target detection on web applications, using data available in cloud security logs.

Our main contribution is an introduction of a flexible ML approach to perform attack detection using logs from web applications. This contribution includes information-rich features of these logs and proposes configurations of ML algorithms with outstanding performance and minimal time overhead. In contrast to previous detection methods, our solution does not require domain expertise in a vast amount of exploits, making deployment almost effortless for web applications in cloud environments. More specifically, feature selection methods on web application logs are used to prove useful features needed for such detection. Accuracy and computation time trade-offs of ML techniques are demonstrated and compared to more traditional ruled-based systems. The particular techniques include both simple and state-of-the-art models such as decision trees, neural networks and ensemble meta-algorithms.

The remainder of this paper is structured as follows. Section 2 briefly describes related work. In Section 3 a brief overview lists the available cloud security logs, followed by datasets used for the training of a log-based intrusion detection model. In Section 4, focusing on web application logs, feature extraction is performed, followed by an extensive search for the optimal feature selection method. ML algorithms we used, are examined in Section 5 for the use in attack classification problems followed by their associated results in Section 6. Next, future research is presented in 7 by proposing a cloud security dashboard architecture and log aggregation system, together with a hybrid approach for attack and anomaly detection. Finally, Section 8 concludes the paper.

Table 1 Overview of related work in the field of attack detection on web or cloud applications.

Paper	Supported web attacks	Input Data	Classification approach	Cloud services scaling
[19]	DoS & probe attacks	Network-layer header data	DTree, FNN & SVM	
[15]	Local access, dos & probe	Network-layer header data	Autoencoder & FNN	
[17]	SQL injection & XSS	Web application requests	SVM, Naïve-Bayes & kNN	
[7]	Android data theft	Network & application data	SVM, FNN, DTree & kNN	
This	10+ high risk web attacks	Cloud-based web app logs	DTree, meta-learning & FNN	✓

2 Related work

This section describes related work on web and network attack detection using ML techniques. Table 1 compares the related works in terms of their supported web attacks, input data, approach and support for multiple types of cloud services such as web applications, databases, cloud storage, etc. The first two papers focus on the detection of network-related attacks, while the other papers focus on attacks set in the application layer. Compared to the works explained in more detail below, this paper focuses on a much broader set of web attack types and extensively compares accuracy and complexity of multiple state-of-the-art types of ML-based classifiers. Moreover, our paper focuses on the ability to scale towards multiple cloud services and supports multiple cloud platforms.

The authors of [19] perform real time detection of attacks using ML techniques and stream processing. Although their method is robust against new attacks and achieves high accuracy, they only validate their model's performance on two kinds of network-based threats: Denial of Service (DoS) attacks and a probe. Similarly, the authors of [15] focus on the detection of three kinds of network-based threats: DoS, probe and remote to local access attacks. They use state-of-the-art ML techniques such as neural networks and autoencoders. Whereas these solutions can intend host security from the cloud provider perspective, our approach focuses on a broad set of attacks, with a big impact on security, targeting cloud customers running many cloud services and hosting web applications.

The authors of [17] used ML-based detection methods (SVM, Naïve-Bayes, kNN) for cross-site scripting and SQL injection attacks. Although this paper shows similar interest in web-based attacks, their focus is smaller and does not cover all types of attacks that can be targeted towards web cloud services. The paper focuses on similar ML techniques, except neural networks and achieves comparable accuracy, however they used smaller datasets which cannot cover most important attacks and their solution cannot be directly applied to the datasets that we considered in this work. [7] combines both network and application (GET/POST request) features to detect malicious malware on Android devices causing data theft, premium SMS features and downloading extra malicious code and backdoors. They also used machine learning approaches such as support vector machines (SVM), neural networks, decision trees and k-nearest neighbours. More specifically, their model indicates transmissions to attackers of user sensitive information coming from client-side devices. Our approach detects attacks that attempt to retrieve such information and also detects attacks that try to modify server-side information.

3 Cloud web application security logs

This paper focuses on web application logs produced by a Infrastructure as a Service (IaaS) or Platform as a Service (PaaS). Software as a Service (SaaS) applications are expected to be monitored and secured by the cloud provider. In addition, SaaS applications do not often provide the needed details for attack detection.

3.1 Azure, Amazon Web Services and Google Cloud

Microsoft Azure, Amazon Web Services (AWS) and Google Cloud Platform are the most used cloud platforms by enterprises today [6]. These cloud platforms offer services to collect and analyse cloud logs such as Azure Diagnostics, Azure Monitor, Amazon CloudWatch and Google Stackdriver.

The structure of many of these logs is similar to web HyperText Transfer Protocol Secure (HTTPS) request logs. Section 3.2 describes web application logs more in detail, as this paper uses such logs to analyse machine learning performance and accuracy. Other cloud service specific logs are not tested since there is a need for cross-cloud platform log aggregation system to connect all cloud service logs. However, the training and testing phases of such systems is similar to the methods proposed in this paper.

3.2 Web application logs

Web application logs, offered by many PAAS and IAAS, are a good way to demonstrate the possibility of log aggregation and conversion to a single format. This is required when using attack detection using machine learning on multiple platforms. To train the classifiers, various web application logs were collected and converted to the World Wide Web Consortium (W3C) format [23] that is common in Apache web servers. Section 3.3 describes the structure of W3C logs as well as attack labels.

3.3 Dataset and labeling

We constructed a dataset with open logs originating from honeypots running web servers among other services. These logs are available at [2] [1]. Honeypots are heavily monitored systems that are intentionally vulnerable and try to collect information upon exploitation in order to detect new attack trends [20].

These datasets contain raw Apache access log files in the W3C format. A common problem in ML applications is the absence of labeled datasets. To allow supervised learning, a rule-based labeling approach was utilized. While this restricts

```
217.160.XXX.XXX - - [12/Mar/2004:22:37:18 -0500]
"GET .?=' or 'a'='a&M=A&N=D&S=A& HTTP/1.1" 400 373 "-" "-"
Reason: "SQL injection"

68.48.XXX.XXX - - [10/Mar/2004:09:56:01 -0500]
"GET /c/winnt/system32.cmd.exe?/c+dir HTTP/1.0" 200 566 "-" "-"
Reason: "Detects basic directory traversal"
```

Fig. 1 Labeled logs used for training attack detection models.

the absolute accuracy that can be achieved, it allows us to continue the research on the accuracy and detection speed trade-off relative to rule-based intrusion detection systems and improve attack detection in complex cloud systems. Fig. 1 shows the structure of W3C logs together with attack classes as labels. These labels were generated using Apache scalp [10], and detection rules from the PHP-IDS project [5].

4 Feature selection

We applied feature selection methods to optimize the attack detection speed and accuracy of ML algorithms. First, we discuss features extracted from the logs. Next, methods of analysing detection accuracy and speed of features are discussed, followed by results of feature selection.

4.1 Web security features

The following attack classes are detected by the proposed classifiers:
"Basic directory traversal", "SQL injection", "Cross-site scripting (XSS)", "Common comment types", "Half-/fullwidth encoded unicode HTML breaking", "JS with(), ternary operators and XML predicates", "Nullbytes and other dangerous characters", "Obfuscated JavaScript script injections", "Includes and typical script methods", "Find attribute, HTML injection and whitespace."

A script was developed to extract features in the well-known Comma Separated Value (CSV) format. Fig. 2 shows the extracted the 28 features from web application logs. Some features are based on the value of parameters in the HTTPS protocol. Most features are constructed using a series or combination of special characters.

```
"method",path_length,file_and_query_length,"protocol","code",bytes,
special_chars_point,special_chars_doublepoint,underscores,
double_underscores,special_chars,special_double_chars,
parentheses_left,parentheses_right,squareparentheses_left,
squareparentheses_right,ampersand,questionmark,colon,percent_sign,
slash,backslach,quote,equalsign,dash,pipe,space,nullbytes,"class"
```

Fig. 2 28 web log features used for attack detection.

4.2 Selection methods

Feature selection is the process of determining which features are relevant for ML algorithms. It can provide detection speed and accuracy improvements by selecting a subset of the total amount of features [11]. We investigated these claims using Weka [24]. Weka contains a collection of machine learning algorithms and allows ranking features (by importance according to the feature selection method) to select a relevant subset, as described in Section 4.3. We selected the top n ranked features to compare results from different selection methods, where n is a number of selected features. Training and testing performances are measured using a J48 classifier. 10-fold cross validation is used to have a representative accuracy result. The following feature selection methods were utilized to score features:

No selection: using all the features and collect their performance and accuracy.

Correlation evaluation: this method evaluates Pearson correlation between the values of features and their corresponding classes [24]. A feature is considered 'good', resulting in a high score when there is a high correlation between two variables (X,Y) with feature X and class Y. The correlation coefficient r is calculated using (1). Values of r will lie between -1 and 1 with 0 having no correlation and 1 high positive correlation [12].

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x}_i)(y_i - \bar{y}_i)}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x}_i)^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}}. \tag{1}$$

where x_i, $\bar{x}_i$, y_i and $\bar{y}_i$ correspond to the ith feature value, the mean feature value, the ith class value and the mean class value, respectively.

Gain ratio: gain ratio uses entropy to evaluate the value of features [24]. Gain ratio between feature X and class Y is defined by (2).

$$GainR(Y,X) = \frac{H(Y) - H(Y|X)}{H(X)}, \tag{2}$$

with

$$H(X) = -\sum_{i=1}^{n} P(x_i) \ln P(x_i), \tag{3}$$

and

$$H(Y|X) = H(Y,X) - H(X). \tag{4}$$

where $P(x_i)$ is the probability of feature X having a value x_i out of all possible values, $H(Y,X)$ is the joint entropy and $H(Y|X)$ is the conditional entropy between class Y and feature X.

Info gain: info gain is similar to gain ratio, but does not divide in (5) by the entropy of feature X.

$$InfoG(Y,X) = H(Y) - H(Y|X). \tag{5}$$

OneR evaluation: this method uses a simple classifier with only one feature to train on. The score is determined by the classification accuracy [14].

A desirable metric that indicates the correlation between features was not found in Weka and could not be tested. Section 4.3 presents results using the discussed selection methods.

4.3 Results

Table 2 shows the measured accuracy and training time with a selection of the 18 top ranked features (instead of all the 28 features). Correlation evaluation and gain ratio have a much lower accuracy than info gain and OneR, but their training timing is lower. This behaviour occurs because the classifier uses different, sometimes more valuable, features selected by the selection algorithms. When the model trains using all the 28 features, results show a higher accuracy of 97.71%, but a slower training time of 40 seconds. The time to classify instances remained the same. In addition, we confirmed this observation as the trained decision trees did not change enough in size. These (deeper) trees contained less features but had more nodes. According to info gain and OneR, the features which are most informative for ML algorithms are: *the length of requested file and query, the amount of special chars, the amount of dots and the amount of percentage signs.*

We conclude that info gain and OneR are the optimal feature selection methods, with a preference for info gain because of its faster training time and minimal accuracy loss. Further results presented in the paper will use all the 28 features and are preferred in the security context because of their high accuracy and equal classification time compared to the selected subsets of features.

Table 2 Detection accuracy and training time for 18 selected features. OneR achieves the highest accuracy and correlation evaluation offers a shorter training time.

Method	**Accuracy**	**Training time**
Correlation evaluation	82.88%	20 seconds
Gain ratio	78.01%	20 seconds
Info gain	97.24%	29 seconds
OneR evaluation	97.25%	32 seconds

5 Machine learning methods

When enterprises make the decision to use a ML-based intrusion detection system over a rule-based system, they have to know what their advantages and trade-offs are. When constructing a trained model to detect attacks on multiple platforms, it is important to focus on generalization because it prevents overfitting. This leads to bad results on unseen data. ML algorithms also promise execution time advantages which, if large enough, could be considered over a rule-based system. Additionally, ML techniques, with deep learning in particular, offer online learning [22]. These techniques enable the support of new attacks without the need to retrain with the full dataset and thus allows fast detection of new attack types. Section 5.1 draws a short overview of the tested algorithms. Next, the test method is explained in Section 5.2, followed by the results in Section 6.

5.1 Tested algorithms

Reduced Error Pruning (REP) tree is a fast decision tree learner [24]. The tree is trained using information gain at each leaf. The feature having the most information gain is used to make two new arcs and nodes. The result is a tree, which is simplified and shown in Fig. 3. After the training process pruning takes place, where leaves are removed to mitigate overfitting. textbfRandom trees are similar to REP trees, but do not apply pruning. At each leaf, a random feature is used to split the tree further into new leaves. This results in a much faster training process, but is usually sensitive to overfitting. As a result, the decision tree is much larger.

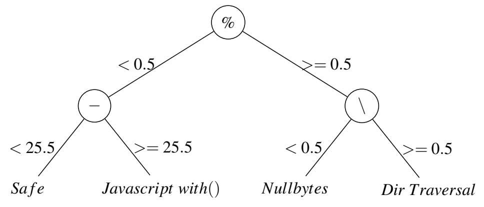


Fig. 3 Simplified representation of a REP tree decision tree.

Table 3 Fully-connected neural network architecture for attack detection

Layer	**Activation function**	**Output dimensions**
Input	-	28 neurons
Dense layer	ReLu	1024 neurons
Dropout layer	-	1024 neurons
Dense layer	ReLU	256 neurons
Dropout layer	-	256 neurons
Dense layer	ReLU	64 neurons
Output layer	Softmax	11 neurons

J48 is Weka's implementation of the C4.5 algorithm. This algorithm uses information gain or entropy to build the J48 decision tree [21]. A confidence factor of 0.25 was used to ensure pruning operations.

Random trees are similar to REP trees, but do not apply pruning. At each leaf, a random feature is used to split the tree further into new leaves. This results in a much faster trainings process, but is usually sensitive to overfitting. As a result, the decision tree is much larger.

Random forest consists of multiple trees, constructed with random subsets of the total features [13]. When a trained forest is used to classify, the feature vector is given to each tree. The class with the greatest occurrence is then chosen by the classifier. In this paper 100 trees were trained.

Boosting is a ML ensemble meta-algorithm that combines multiple weak classifier into one strong classifier [8]. Weka's implementation of boosting is AdaBoost [24]. Multiple iterations modify the weak classifiers to reduce the amount of misclassified instances. The final boosted classifier uses a weighted sum of the accuracy from all previous weak classifiers to determine the class. To test this algorithm, a J48 classifier with 15 iterations was used.

Bagging, like boosting, is a ML ensemble meta-algorithm. Multiple random subsets of the total training set are used to train new classifiers. The bagging classifier chooses the class that is predicted the most, given a feature vector, by all the generated classifiers [4]. Again, a J48 classifier with 15 iterations was used.

Neural networks such as a multilayer perceptron exist out of connected neurons, having a non-linear mapping between input and output vectors [9]. The output of each neuron is the output of the activation function in a neuron, with the sum of all inputs as parameter. We implemented a Fully-connected Neural Network (FNN) in Tensorflow, using rectified linear units (ReLU) as activation functions. Table 3 presents the proposed FNN architecture. It contains five dense layers and includes dropout to improve generalisation. For simplicity and equal comparison to the other proposed methods, in this paper we did not consider more complex architectures e.g. Convolutional Neural Networks (CNN). CNN has the advantage of automatically extracting features, a step which we already performed before providing input data. The proposed neural network was configured to perform 1000 iterations (epochs) and uses a batch size of 256. In each iteration, the weights of each arc in the network are adapted to better fit the training data. This can be achieved using back propagation and gradient descent [9].

5.2 Testing method

Accuracy of the proposed ML algorithms is measured with 10-fold cross validation, that will detect overfitting by testing 10 times with a 10% random subset and training with the other 90%. A pareto-efficiency chart is used to illustrate these results. This chart gives a quick overview if a algorithm is not pareto-efficient, meaning that there is another algorithm with a higher accuracy and a lower classification time. Classification time is calculated based on the total time needed to classify all 73165 logs. Numerical results are presented in terms of accuracy, F-score, training and classification time.

6 Performance and accuracy results

Fig. 4 shows the pareto-efficiency of the random tree, REP tree, J48, bagging, boosting, random forest and neural networks. The rule-based measurement is not shown on the graph, because the classification time is too extended. However, we can see which algorithm achieves the most optimal results. Random tree is the fastest in classifying instances, while the neural network is the most accurate. We can also see that there should be no reason to choose random forest in this case. Random forest is pareto-inefficient, because it is slower and less accurate than boosting and the neural network. Table 4 shows the accuracy and F-score together with training and testing times of the tested ML algorithms. Based on Table 4 and Fig. 4, J48 should be the choice for cases where classification time is important. On the other hand, if accuracy is critical, neural networks should be used. The (relative) 100% rule-based method completes classification in 115.74 seconds. This is 47 times slower than J48. The neural network is 26 times faster than a rule-based method, while only loosing 1.69% accuracy. Taking into account the training time, decision trees show good scaling when using large datasets. The authors of [25] show that training decision

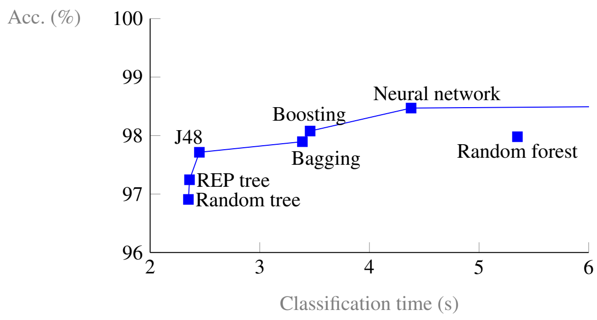


Fig. 4 Accuracy (Acc.) and classification time on our dataset using the proposed machine learning techniques.

Table 4 Performance of rule-based versus machine learning attack detection systems.

	NN	**Rand. tree**	**REP tree**	**J48**	**Bagging**	**Boosting**	**Rand. forest**	**Rule-based**
Accuracy	**98.47%**	96.91%	97.24%	97.71%	97.90%	98.08%	97.98%	100%
F-score	**98.47%**	96.90%	96.55%	97.30%	97.90%	98.10%	98.00%	100%
Train time	110.05s	**2.69s**	4.64s	11.56s	31.59s	98.14s	12.71s	n/a
Classification time	4.38s	**2.35s**	2.36s	2.45s	3.39s	3.46s	5.43s	115.74s

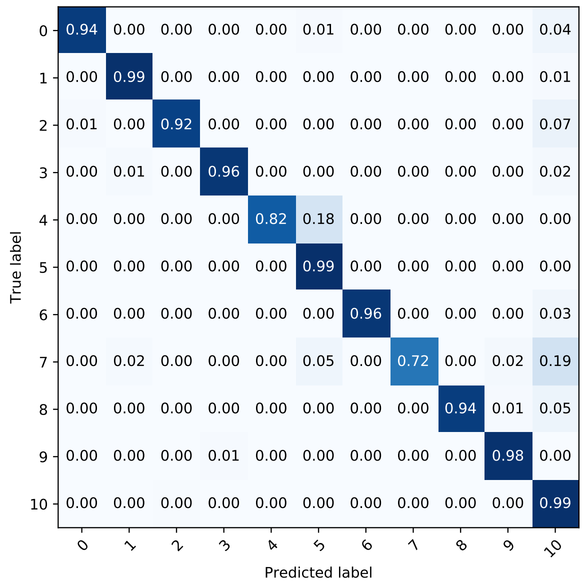
trees has a complexity of $O(mnlog(n)) + O(n(log(n))^2)$, where m and n correspond to the amount of features and training instances respectively. Decision trees such as J48 and REP tree will introduce pruning, that will increase training time, but decrease the total height of the tree. In addition, a less complex model is produced. Boosting and bagging are more complex because of their nature of building multiple decision trees. However, using weak classifiers, that have an accuracy just above 50%, these algorithms will produce less complex models. Unfortunately, we were not able to achieve good results with the weak classifiers.

To briefly go into more detail, the confusion matrix in Fig. 5 shows the classification score for each of the 11 classified classes (10 attack classes + safe class). These results are derived from the best classifier, in our case the fully-connected neural network. For better readability, the categorical attack names are not displayed. The 'safe' attack class (nr. 10) is classified 99% correctly. Except class nr. 4 'Half-/fullwidth encoded unicode HTML breaking' and class nr. 7 'Includes and typical script methods', all other attack classes are classified near perfectly. However, class nr. 4 and nr. 7 only contain 34 and 130 training examples respectively, making generalisation harder for these classes. Wrongly detected attack classes are only infrequently classified as a 'safe' class, which is unwanted behaviour in the security context. This accuracy could be strengthened further by training the classifier only on attack / safe class and increasing the generalisation for attack detection.

As a conclusion, we can verify that neural networks in combination with the proposed selected features enable near-perfect accuracy and is able to differentiate all the attacks from the safe logs. Moreover, J48 scores better when considering classification time. This classifier achieves very good accuracy, while having 44% less the classification time compared to neural networks. Using ML techniques can be a viable option if web applications are constrained by the performance of an intrusion prevention system (IPS). We see 26-47 times higher detection speed, that can lead to the same improvement when used in an IPS.

7 Future research

Future work should depend on more real-world data, coming from many deployed cloud services. At the time of writing, hardly any open data is available concerning this. We urge companies to commence publishing data in an open format, preferably anonymized, to allow and help future research to collect more results and design up-to-date systems. Extending this research, consists of implementing ML-trained



	0	1	2	3	4	5	6	7	8	9	10
0	0.94	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.04
1	0.00	0.99	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01
2	0.01	0.00	0.92	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.07
3	0.00	0.01	0.00	0.96	0.00	0.00	0.00	0.00	0.00	0.00	0.02
4	0.00	0.00	0.00	0.00	0.82	0.18	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.00	0.00
6	0.00	0.00	0.00	0.00	0.00	0.00	0.96	0.00	0.00	0.00	0.03
7	0.00	0.02	0.00	0.00	0.00	0.05	0.00	0.72	0.00	0.02	0.19
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.94	0.01	0.05
9	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.98	0.00
10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99

Fig. 5 Confusion matrix of the fully-connected neural network.

models in the real world and preprocessing combined logs of multiple cloud services, that generate logs. Next these logs can be transferred to a central system that processes these logs e.g. an Azure WebJob. An Azure WebJob runs in an infinite loop, while constantly checking if new data is available to be processed. We used Weka for .NET in a similar configuration to confirm successful deployment of the model proposed in this paper. The classified output of such model is then saved to a central database. These logs can then be visualized on a security cloud dashboard. This future work requires building a platform that combines logs from multiple cloud services (other than web applications) while simulating attacks on the platform in order to extend available training data. Another addition can be made by introducing an online self-learning system. Such system receives input from experts on the proposed cloud security platform. Experts can e.g. indicate which attacks are misclassified. Periodically the model is updated, keeping these corrected labels or new attacks into account. A major advantage, equivalent to the techniques proposed in this paper, is that we only need to know which attack occurred, without knowing how the attack is constructed.

Another contribution can be made investigating a hybrid approach by combining anomaly detection, based on the ML techniques in the proposed model, with a rule-based system for classifying specific attacks. As shown in [16] attack detection by only classifying attack or no-attack can perform better than signature-based methods. More complex rule-based systems can investigate which kind of attack occurred while the anomaly detection method would have the same performance advantages as the proposed model in this paper, allowing more real-time detection and mitigation of attacks on cloud-based applications. Such contribution can also investigate if the anomaly detection is able to detect zero-day attacks, that the rule-based system cannot classify yet.

8 Conclusion

The increasing complexity in the landscape of cloud applications leads to the need of a more flexible security system. Such system should allow less complex and easier-to-update detection models. In this work, contributions are made by presenting web cloud security logs that show similarities with traditional on-premises security logs. Extra metrics can allow attacks to be classified that influence the performance of a system, e.g. a distributed denial-of-service attack. Machine learning techniques are proposed, with the advantage of having more simplicity when training detection models by quickly adding newly discovered attacks without the knowledge of their details. These models are applied on web application logs, derived from multiple servers. Such logs are converted into a single format, that helps easier feature extraction. Feature selection test scenarios show that information gain offers the highest accuracy compared to other feature selection methods, while also being the fastest method. However, using all the features still delivers the highest accuracy. The J48 decision tree offers the overall best results, considering both accuracy and performance. This classifier achieves a 47x performance improvement over the traditional rule-based systems, while only loosing 2.29% accuracy. The neural network offers the highest accuracy of 98.47% and enables efficient and accurate attack detection, possible on multiple web services in cloud environments.

Acknowledgements This work was funded by the Fund for Scientific Research Flanders (Belgium, FWO-Vlaanderen, FWO-SB grant number 1SB7619N).

References

1. Honeypot project. http://old.honeynet.org/. (Accessed on 03/22/2018)
2. Public security log sharing site. http://log-sharing.dreamhosters.com/ (2006). (Accessed on 03/22/2018)
3. Almorsy, M., Grundy, J.C., Müller, I.: An analysis of the cloud computing security problem. CoRR **abs/1609.01107** (2016). URL http://arxiv.org/abs/1609.01107
4. Breiman, L.: Bagging predictors. Machine Learning **24**(2), 123–140 (1996). DOI 10.1023/A:1018054314350. URL http://dx.doi.org/10.1023/A:1018054314350
5. Christian Matthies, M.H.: Phpids. https://github.com/PHPIDS/PHPIDS (2014). (Accessed on 05/11/2017)
6. Clutch: 2016 enterprise cloud computing survey. https://clutch.co/cloud#survey (2016). Accessed on 29/05/2017
7. Feizollah, A., Anuar, N.B., Salleh, R., Amalina, F., Maarof, R.R., Shamshirband, S.: A study of machine learning classifiers for anomaly-based mobile botnet detection. Malaysian Journal

of Computer Science **26**(4), 251–265 (2013). URL https://mjes.um.edu.my/index.php/MJCS/article/view/6785
8. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences **55**(1), 119 – 139 (1997). DOI http://dx.doi.org/10.1006/jcss.1997.1504. URL http://www.sciencedirect.com/science/article/pii/S002200009791504X
9. Gardner, M., Dorling, S.: Artificial neural networks (the multilayer perceptron) - a review of applications in the atmospheric sciences. Atmospheric Environment **32**(14-15), 2627 – 2636 (1998). DOI https://doi.org/10.1016/S1352-2310(97)00447-0. URL http://www.sciencedirect.com/science/article/pii/S1352231097004470
10. Gaucher, R.: Apache-scalp. https://code.google.com/archive/p/apache-scalp/ (2008). (Accessed on 05/11/2017)
11. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. J. Mach. Learn. Res. **3**, 1157–1182 (2003). URL http://dl.acm.org/citation.cfm?id=944919.944968
12. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update. SIGKDD Explor. Newsl. **11**(1), 10–18 (2009). DOI 10.1145/1656274.1656278. URL http://doi.acm.org/10.1145/1656274.1656278
13. Ho, T.K.: The random subspace method for constructing decision forests. IEEE Transactions on Pattern Analysis and Machine Intelligence **20**(8), 832–844 (1998). DOI 10.1109/34.709601
14. Holte, R.: Very simple classification rules perform well on most commonly used datasets. Machine Learning **11**, 63–91 (1993)
15. Ieracitano, C., Adeel, A., Gogate, M., Dashtipour, K., Morabito, F., Larijani, H., Raza, A., Hussain, A.: Statistical analysis driven optimized deep learning system for intrusion detection (2018)
16. Kaur, J.: Wired lan and wireless lan attack detection using signature based and machine learning tools. In: G.M. Perez, K.K. Mishra, S. Tiwari, M.C. Trivedi (eds.) Networking Communication and Data Knowledge Engineering, pp. 15–24. Springer Singapore, Singapore (2018)
17. Komiya, R., Paik, I., Hisada, M.: Classification of malicious web code by machine learning. In: 2011 3rd International Conference on Awareness Science and Technology (iCAST), pp. 406–411 (2011). DOI 10.1109/ICAwST.2011.6163109
18. Liu, W.: Research on cloud computing security problem and strategy. In: 2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet), pp. 1216–1219 (2012). DOI 10.1109/CECNet.2012.6202020
19. Lobato, A., Lopez, M.A., Duarte, O.: An accurate threat detection system through real-time stream processing
20. Provos, N., et al.: A virtual honeypot framework. In: USENIX Security Symposium, vol. 173, pp. 1–14 (2004)
21. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1993)
22. Sahoo, D., Pham, Q., Lu, J., Hoi, S.C.: Online deep learning: Learning deep neural networks on the fly. arXiv preprint arXiv:1711.03705 (2017)
23. W3: Logging in w3c httpd. https://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format (1995). (Accessed on 05/27/2017)
24. Weka: Weka documentation. http://weka.sourceforge.net/doc.stable/ (2016). (Accessed on 05/15/2017)
25. Witten, I.H., Frank, E., Hall, M.A., Pal, C.J.: Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann (2016)