Estimating Web Attack Detection via Model Uncertainty from Inaccurate Annotation

Xinyu Gong*, Yuefu Zhou†, Yue Bi‡, Mingcheng He§, Shiying Sheng¶, Han Qiu‖, Ruan He** and Jialiang Lu††
SJTU-ParisTech Elite Institute of Technology
Shanghai Jiao Tong University
Shanghai, China 200240
Email: {*zachary, †remicongee, ‡biyue111, §mingcheng.he, ¶ssy_1995, ††jialiang.lu}@sjtu.edu.cn;
‖Telecom-ParisTech
Paris, France 75013
Email: han.qiu@telecom-paristech.fr
**Tencent
Shenzhen, China 518000
Email: ruanhe@tencent.com

***Abstract*—In the past decades, Machine Learning (ML) techniques have become a hot topic in the web security field. Deep learning (DL), as a sub-field of machine learning, has proved its effectiveness in concluding various attack patterns via raw input data. To reach high accuracy, DL models are usually trained with labelled data. However, in the security field, annotation error can have a significant impact on model training. Under such a premise, we introduced the model uncertainty to the DL-based web attack detection. The model uncertainty is used to estimate the credibility of the prediction made by the model. As far as we know, we are the first to introduce this concept to web security. In our work, the model uncertainty is provided in the form as the variance of a Bayesian model. By training our attack detection model on real web logs with annotation errors, we proved that the wrongly tagged web logs tended to gain a higher variance. Therefore, by analyzing the variance result, the security operators can easily locate these mistagged web logs. This helps to find unknown attacks neglected by data annotation and to refine the existing attack detection methods.**

***Index Terms*—Cyber security, Web attack, Model uncertainty, Data annotation, Deep learning**

I. Introduction

The traditional Intrusion Detection System (IDS) and Intrusion Prevention System (IPS) apply a series of rules to detect and prevent cyber attacks. For example, an IDS using SNORT [1] as its back-end is sensitive to data packages sent from or to certain ports. It also captures data packages which contain certain keywords. These kinds of systems depend on hand-designed attack signatures or normal behaviour signatures, and they cannot be adapted to unknown attacks.

To cope with the above problem, researchers have proposed many systems that apply Machine Learning (ML) algorithms to detect attacks. Learning algorithms like decision tree, support vector machine, and neural networks are investigated in [2], [3] for attack detection. Applying these detection methods does not rely on attack signatures. To detect unknown attacks, some clustering methods [4], [5] are investigated. The clustering method does not rely on data annotation, but it highly depends on human intervention to identify each cluster. These machine learning methods have the problem that they highly depend on the provided features. For example, the KDD 1999 dataset [6] contains 41 features of network traffic. Selecting useful features from them is the main difficulty when applying these learning algorithms.

As the sub-field of traditional machine learning, Deep Learning (DL) overcomes the above drawback. It analyzes raw input data and generates latent features from it. The feature selection stage is no longer needed. The latent features can be considered as the generated attack signatures. Therefore, the training of a deep learning model is equivalent to the process of creating attack signatures automatically. Researchers also proved that deep learning techniques achieve a better accuracy [7]–[10] comparing with the previous works [11], [12].

Nevertheless, no matter the feature-based machine learning methods or the deep learning methods, they all suffer from the following shortcoming: learning algorithms require a large volume of training data. The data labels are also required to reach a high classification accuracy. Correctly tagging these traffic data manually is a difficult task. Using a clean dataset like KDD 1999 for training and testing is just the ideal case.

In this paper, we introduce model uncertainty to the web security field. Model uncertainty is used to evaluate the detection result. A higher model uncertainty shows the ML model has less confidence in its detection result. We propose a two-step ML model to yield model uncertainty via web logs. It consists of a Convolutional Neural Network (CNN) feature extractor and a Bayesian classifier. The CNN part creates latent features from raw web logs, while the Bayesian classifier detects attacks based on the latent features and also provides model uncertainty in the form of variance. We trained this ML model on datasets with annotation errors. In this case, some attacks were labelled as normal accesses, or some normal accesses were labelled as attacks. After training, the ML model obtained a good classification performance by following the tags in the training set. For an attack tagged as normal access, the model also classified this attack as normal access, while it marked these attacks with high variance. Therefore, these wrongly tagged attacks can still be detected by our model.

To measure the effectiveness of the proposed method, we conducted experiments on two real web log datasets. The first one is tagged manually, and we assumed that the manual annotation result is the ground truth. We repeatedly changed the tags of a part of the malicious logs to simulate the common scenario that the IDS neglects some certain type of attacks. We proved that these wrongly tagged logs are very likely to show a higher variance, and they count a large percentage in the logs with the highest variance. For the second dataset, which is too large for manual inspection, we roughly tagged it with existing attack signatures. From the variance result, we found the annotation errors caused by some attack signatures. We proved that the proposed method is not only capable of finding missing attacks but also capable of finding the false alarms caused by annotation errors.

The paper is structured as follows: In Section II, we introduce some previous researches related to our work. In Section III, we describe the ML techniques that we used to build this attack detection model. We present the datasets used, experiment setup, and experiments result in Section IV. Finally, we conclude our work and propose the future research direction in Section V.

II. Related Works

A. Machine learning based attack detection:

The cyber attack detection is commonly classified as misuse detection and anomaly detection [13]. In many cases, they are respectively implemented by supervised and unsupervised learning [14] algorithms. [15] builds and compares different learning algorithms including random forest, decision tree, and adaboost classifier for web log attack detection. It has conducted experiments on the CSIC 2010 HTTP [16] dataset to prove its effectiveness.

[17] proposes an anomaly detection method for web applications using unsupervised learning techniques. It defines users' behaviors by the feature of web resources, and uses K-means model to divide different behavior clusters.

[18] introduces a hybrid model combining misuse detection and anomaly detection. It uses existing attack signatures to match the web logs. The attack signatures is used to capture obvious attacks. It also builds a normal-behavior library based on the k-means algorithm. The incoming web requests which do not belong to this library will be delivered for manual inspection.

B. Deep learning based attack detection:

Yoon Kim has proved the capability of CNN for a sentence classification task in [19]. In his work, he applies word embedding on sentences which transfers the word vectors into matrices. Detecting attacks through web logs is similar to the sentence classification in the aspect that: 1) They are both classification tasks. 2) They both take the chain of characters as their input.

Built on the work of Yoon Kim, [20]–[22] uses CNN models to detect web attacks. Among them, the first two works analyze the URL in the HTTP requests. [20] splits URL into words by special characters like /, &, =, +, etc. A word-level embedding is then applied to these separated words. It uses CSIC 2010 HTTP as its training and testing dataset. It achieves satisfactory result with high accuracy and a low false alarm rate. In [21], instead of separating the URL into words, the raw URL is taken as the input. Each character in the URL is embedded in a high dimension space. The author also changed the max-pooling layer in the model of [19] to a sum pooling layer. The researchers proved their model has a higher detection rate than an n-gram model and security expert-derived features. In [22], the author has trained a CNN model for a text classification task,and he proved that this trained model can also extract useful features from web logs.

C. Dropout and model uncertainty:

Dropout is introduced by [23] for improving the capacity of generalization of deep neural network. A variational variant of this technique is then proposed in [24], showing better performance supported by the Bayesian framework. Based on these works, [25] proves that dropout can be viewed as a Bayesian approximation for estimating model uncertainty [26], which measures the confidence of model prediction.

III. Machine Learning Models

Our attack detection model is consist of two parts: 1) A CNN model extracts latent features from raw web logs. 2) A Bayesian model analyzes these latent features and gives out the detection result and the variance result.

A. Convolutional Neural Network as Feature Extractor

We leverage the previous work of [19]–[21], and build a CNN model to extract useful features from web logs. As it is shown in Fig. 1, the proposed CNN model has a simple architecture: it is consist of a convolution layer, a max pooling layer, and a fully connected layer.

Differ from [20], [21], the raw web log is taken as the input of our model. Using the whole piece of web log, not only the URL part, helps to detect attacks from more aspects. Response code, source address, and user agent can also provide useful information for the CNN model.

To mitigate the pre-processing, we transferred each character in the web log into an ASCII code. Each character is then embedded into a high dimension vector as it is shown in Fig. 2. In the experiment part, we also proved that without the word splitting, our model is still able to reach a high accuracy in attack detection.

B. Bayesian Model as Classifier

Dropout [23], first introduced for better generalization, is a technique that randomizes the model parameters via multiplicative stochastic noise. [25] further proves, under the Bayesian framework, that the model output can be randomized as well so that the model uncertainty is available in form as

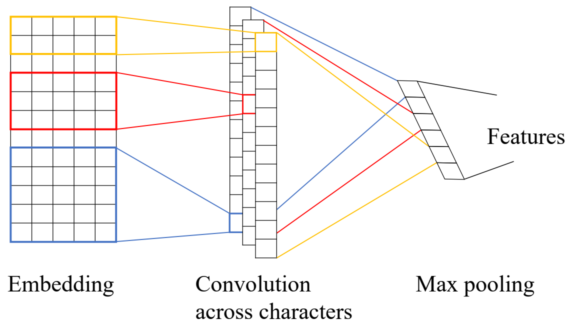
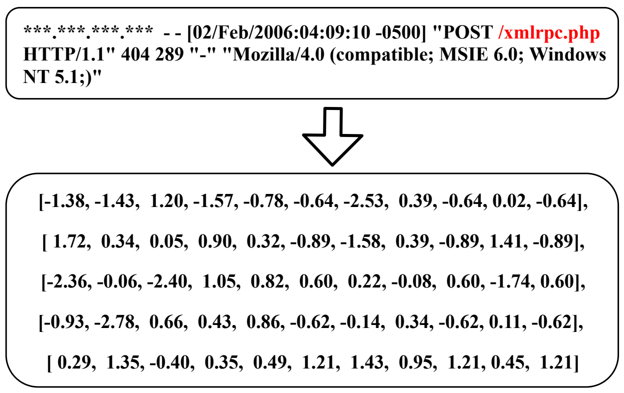


Fig. 1: The CNN architecture adopted, which follows the normal convolution process but with a character across pattern and max pooling over the sequence.



```
***.***.***.***  - - [02/Feb/2006:04:09:10 -0500] "POST /xmlrpc.php
HTTP/1.1" 404 289 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows
NT 5.1;)"
```

```
[-1.38, -1.43,  1.20, -1.57, -0.78, -0.64, -2.53,  0.39, -0.64, 0.02, -0.64],
[ 1.72,  0.34,  0.05,  0.90,  0.32, -0.89, -1.58,  0.39, -0.89, 1.41, -0.89],
[-2.36, -0.06, -2.40,  1.05,  0.82,  0.60,  0.22, -0.08,  0.60, -1.74, 0.60],
[-0.93, -2.78,  0.66,  0.43,  0.86, -0.62, -0.14,  0.34, -0.62, 0.11, -0.62],
[ 0.29,  1.35, -0.40,  0.35,  0.49,  1.21,  1.43,  0.95,  1.21, 0.45,  1.21]
```

Fig. 2: An example of character-level embedding, the red part which is a vector containing 11 characters, it is transferred into a 5×11 matrix. It should be remarked that our CNN feature extractor analyzes the whole piece of web log, not only the URL part.

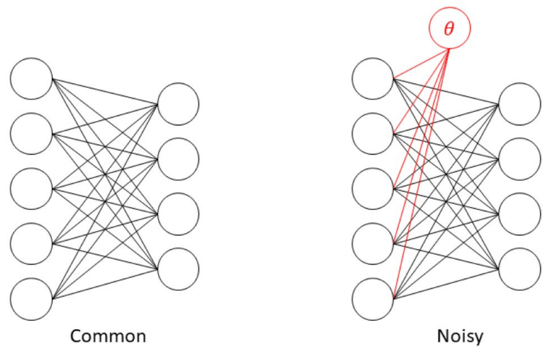


Fig. 3: Illustration for dropout noise insertion. Compared with a common case (left), a multiplicative random noise (right) is sampled independently for each input dimension when forwarding.

the variance of prediction. Inspired from this point, we insert a dropout noise multiplied on input in each layer, as shown in Fig. 3. To achieve better fitness on data, we adopt Gaussian noise [23] and tune it via re-parameterization trick [24] along with model parameters in back-propagation fashion [27]. In terms of the prediction at inference time, we provide a variance estimator for computing model uncertainty.

Formally, note x_j^l the jth input in lth layer, $\theta_j^l \sim_q \mathcal{N}(1, \alpha_j^l)$ the corresponding dropout noise. Then the output of this layer is computed as

$$y_i^l = \sigma \left(\sum_{j=1}^{J_l} W_{ij}^l \cdot \tilde{x}_j^l \right), \quad \tilde{x}_j^l = x_j^l \cdot \theta_j^l \tag{1}$$

where J_l is the input dimension, W^l the weight matrix, $\sigma(.)$ the activation function. Under Bayesian framework, as illustrated in [24], to optimize $\alpha = \{\alpha_j^l\}$ and $W = \{W^l\}$ is equivalent to maximizing the variational lower bound (Eqn. 2), which compromises between data likelihood and prior knowledge.

$$\begin{aligned}\mathcal{L}(\alpha, W) = \sum_{(x,y) \in D} & p\left(y^L = y | x, \alpha, W\right) \\ & - KL\left(q(\Theta) || p(\Theta)\right)\end{aligned} \tag{2}$$

where y^L is model output, x the input, D the dataset, $\Theta = \{\theta_j^l\}$ and $KL(q(\Theta)||p(\Theta))$ the Kullback-Leibler divergence from q to the prior p.

In the field of attack detection, there is no precedent for the prior of dropout noise, thus we simply adopt the uniform distribution on real value $\theta_j^l \sim_p \mathcal{U}(\mathbb{R})$. Hence,

$$KL(q(\Theta)||p(\Theta)) = \sum_{l,j} \frac{1}{2} \log\left(\alpha_j^l\right) \tag{3}$$

With the optimums obtained as:

$$\alpha^*, W^* = \arg\max \mathcal{L}(\alpha, W) \tag{4}$$

The prediction of the trained model is a random variable fitting the dataset. To compute the variance as model uncertainty, we provide an estimator for any input x as follow:

$$\hat{V}(x) = \frac{\sum_{i=1}^N f(y^L(\Theta_i))}{N-1} \cdot \left(1 - \frac{\sum_{i=1}^N f(y^L(\Theta_i))}{N}\right) \tag{5}$$

where $y^L(\Theta_i)$ is the output based on i-th sampling for the dropout noise Θ, the prediction function $f(.)$ returns the maximum's index (0 or 1). For ones concerning the prediction itself, $f(\bar{y}^L)$ (Eqn. 6) is a nice choice. Besides, we provide a fast alternative computed as $f(y^L(\mathbb{E}[\Theta]))$, where the model forwards by fixing the dropout noise as its mean value.

$$f(\bar{y}^L) = f\left(\frac{1}{N} \sum_{i=1}^N y^L(\Theta_i)\right) \tag{6}$$

In this study, we adopt two stacked fully-connected layers as our Bayesian model's architecture. The input is the CNN features extracted and restored in advance, as seen in Fig. 4.

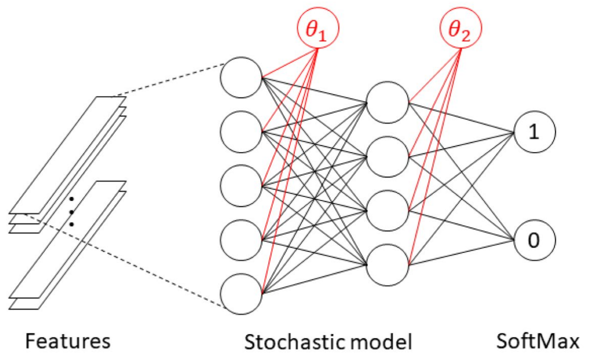


Fig. 4: The complete Bayesian model. The features forward through the stochastic model which outputs SoftMax vectors.

IV. Experiments and Evaluation

The experimental evaluation presented in the following section demonstrates the behaviour of our proposed detection model on two real web log datasets. The first one has a relatively small volume. It is collected in 2006, so we refer it as *Apache-2006 dataset* for the sake of convenience. We tagged it manually and then use a signature-based web attack detection program *scalp*[1] to detail the attack types for each piece of attack in it. The second dataset is named as *Apache-2017 dataset*. It has a larger volume which makes it impossible to be tagged manually. Therefore, Scalp is directly used to tag the Apache-2017 dataset. The Apache-2006 dataset contains relatively less complicated logs. Therefore, in the aspect of classifying attack/no-attack web logs, Scalp works quite well and gives out a coherent result as manual work. While the other dataset contains more sophisticated cases, so Scalp introduces some annotation errors.

A. Evaluation Metrics

For better illustration, we introduce the notices for this attack/no-attack classification problem in Table I, e.g. an attack which is correctly classified is defined to be *TP*.

TABLE I: Confusion matrix.

Tag \ Predicted	Normal	Attack
Normal	TN	FP
Attack	FN	TP

Though the main target of this attack detection model is to provide variance for each web log, we still evaluate the performance of the Bayesian classifier by accuracy, precision, recall, and F1-score. It demonstrates that the classifier is not giving a random detection result, and in addition, although the tags contain errors, it performs quite well on this classification task.

$$Precision = \frac{TP}{TP+FP} \tag{8}$$

[1]https://code.google.com/archive/p/apache-scalp/

$$Recall = \frac{TP}{TP+FN} \tag{9}$$

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \tag{10}$$

$$F1\ Score = 2 \cdot \frac{Recall \cdot Precision}{Recall + Precision} \tag{11}$$

B. Automated Web Attack Detection Program

Scalp is an Apache log analyzer, it uses regular expressions from the PHP-IDS project[2] to match the attack signatures in the Apache server logs. Its default filter profile contains 76 filters which detect various attack types including: Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Spam, Local File Inclusion (LFI), SQL Injection (SQLI), Remote File Execution (RFE), Denial Of Service (DOS), Directory Traversal (DT), and Information Disclosure (ID). It is the widely used tool to detect attacks through web logs, but it is not perfect. Here we provide an example in Fig. 5 to show why these attack signatures are causing annotation errors for a piece of real web log.

```
[ATTACKED] ***.***.***.*** - - [13/Nov/2016:23:10:11 +0800]
"GET/scrape.phppasskey=2138171641cea812369207dbd4b3fa15&
info_hash=%abs%9a%96%d8%19%d%1f7%23%d5%91%27%
20%0b5m2%b7_ HTTP/1.1" 301 185 "-" "Transmission/2.84"
```

Fig. 5: A sample from the Apache-2017 dataset, which is actually normal access, but it is tagged as an attack for the reason: Detects nullbytes and other dangerous characters.

C. Apache-2006 Dataset

Description: A piece of real server log recorded in 2006 is used in this experiment. It contains 14432 log records. We manually tagged it with 2 tags: SAFE/ATTACKED. Among all the log records, 4260 are tagged as ATTACKED. An example of attack is given in Fig. 6. We then used Scalp to tag this dataset. It should be noticed that in this piece of log, Scalp gives out the same result as the manual work, therefore, we regard the annotation result as our test baseline. Scalp also details the reason why it recognizes these logs as attacks. The amount of each type of attacks is shown in Table II.

```
[ATTACKED] ***.***.***.*** - - [22/Jan/2006:08:28:17 -0500] "GET
/awstats/awstats.pl?configdir=|echo;echo%20YYY;cd%20%2ftmp
%3bwget%20209%2e136%2e48%2e69%2fmirela%3bchmod%20%2bx
%20mirela%3b%2e%2fmirela;echo%20YYY;echo| HTTP/1.1" 404 296
"-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;)"
```

Fig. 6: A sample of web attack in the Apache-2006 dataset.

Training Setup: To simulate the case that some of the attacks are neglected, we have conducted three sets of experiments. Each time, one of the first three attack types in Table II was manually tagged as SAFE, e.g. in the first experiment, all the attacks of type No.1 were tagged as normal

[2]https://github.com/PHPIDS/PHPIDS/blob/master/lib/IDS/default_filter.xml

TABLE II: Different attack types in Apache-2006 dataset.

Type	Description	Amount
1	JavaScript with(), ternary operators and XML predicate attacks	839
2	JavaScript DOM/miscellaneous properties and methods	1352
3	Code injection attempts	1896
4	Basic obfuscated JavaScript script injections	39
5	Common comment types	71
6	Halfwidth/fullwidth encoded unicode HTML breaking attempts	49
7	Specific directory and path traversal	14

logs. For comparison, an additional experiment was conducted on the original dataset. In each experiment, the dataset was divided into training and testing set with a portion of 5:5. The CNN feature extractor was trained for 5 epochs and the Bayesian classifier was trained for 2 epochs with the Adam [28] optimizer. It should be noticed that the mistagged attacks exist in both the training and testing set.

Results: The classification result is firstly shown in Table III. The result follows the changed tags for the first three experiments.

TABLE III: Classification result in the repeated experiments.

	Precision	Recall	Accuracy	F1 Score
Type 1	90.44%	99.47%	97.38%	94.74%
Type 2	99.93%	98.72%	99.72%	99.32%
Type 3	96.21%	93.51%	98.37%	94.84%
Clean dataset	99.76%	99.34%	99.73%	99.55%

TABLE IV: Variance result in the repeated experiments.

	TP	TN	FP	FN	Total
Type 1	0.0641	0.0155	0.2357	0.2212	0.0327
Type 2	0.0750	0.0131	0.1258	0.2374	0.0264
Type 3	0.0327	0.0063	0.2099	0.2379	0.0139
Clean dataset	0.0123	0.0057	0.1987	0.2277	0.0082

As it is shown in Table III, this two-step model reached a satisfactory result on classification. The model performed best when the dataset had no data annotation error. This is a quite reasonable result because a clean dataset does not mislead the training process. The average variances are respectively shown in Table IV, the average variance on the clean dataset is the lowest. It shows that the detection model is more confident when detecting attacks on the clean dataset.

Shown in Fig. 7, we demonstrate that the wrongly tagged logs count a large portion in the logs which have the highest variance. It is a practical feature in the real application scenario that the security experts just need to view a small part of the web log records so as to fix the data annotation errors and to refine the existing attack signatures or ML models.

Through Fig. 8, it can be observed that in the first three sets of experiments, the wrongly tagged attacks which are classified as normal accesses claim a much higher variance comparing to the average variance of all the web logs in the whole TN class. From Fig. 7 and 8, it can be proved that: the wrongly

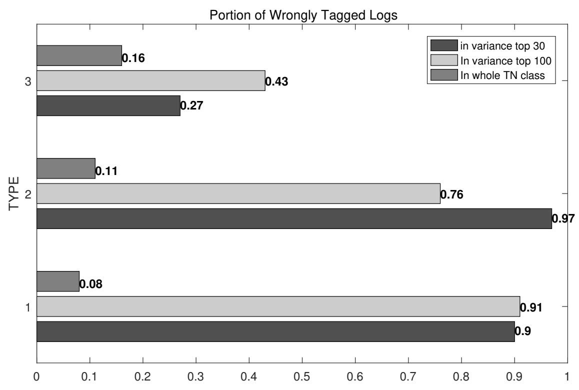


Fig. 7: Wrongly tagged logs with high variance, e.g. in the second experiment, the wrongly tagged logs count 11% in *TN* class. In the 30 logs that have the highest variance in *TN* class, 29 are the wrongly tagged logs. In the top 100 logs, 73 are the wrongly tagged ones.

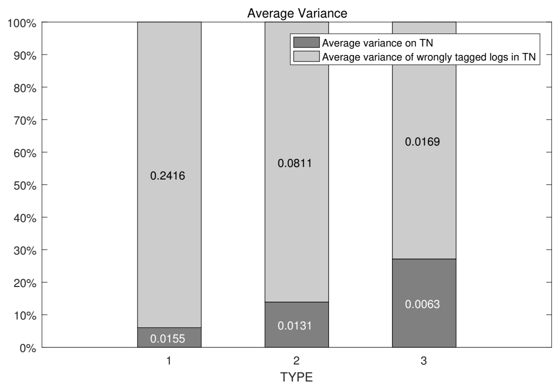


Fig. 8: The average variance of the wrongly tagged logs in the TN class and the average variance of all the logs in the TN class, e.g. in the second experiment, the average variance of the wrongly tagged logs in the TN class is 0.0811, while the average variance of all the logs in TN class is 0.0131.

tagged logs tend to possess high variance, and the logs with high variance are very likely to be the wrongly tagged ones.

D. Apache-2017 Dataset

Description: A piece of real web log collected from a student forum is used in this experiment. It contains 7966387 web logs from November 2016 to August 2018. In this piece of web log, over 50000 logs are tagged as ATTACKED by Scalp. As we have provided an example of the annotation errors made by Scalp in Fig. 5, this tagging result cannot be regarded as the ground truth. Among these web logs, we have randomly selected 70000 SAFE logs and 30000 ATTACKED logs to compose our dataset.

Training Setup: The subset of 100000 web logs are divided into a training set and a testing set. The proportion between the training set and the testing set is set to 7:3. The CNN

TABLE V: Evaluating results on the Apache-2017 dataset.

	Precision	Recall	Accuracy	F1 Score
On the test set	99.84%	94.77%	98.38%	97.24%

feature extractor was trained for 2 epochs and the Bayesian classifier was trained for 2 epochs.

Results: The classification result on the testing set is shown in Table V. Although this dataset contains more sophisticated logs, our model still highly performed on classification. The precision, recall rate, F1 score, and accuracy are all over 94%.

In Section IV-C, when conducting experiments on the Apache-2006 dataset, we know the real tag of each log, therefore we have calculated the average variance and the portion of wrongly tagged logs. While for this dataset, we know little about each log's true tag. Therefore we only investigate respectively the 100 web logs with the highest variance in the TP and TN class. With the aid of variance provided by the Bayesian classifier, many wrongly tagged logs can be found:

1) Among the 100 web logs with the highest variance in the TP class, there are in total 81 logs which are tagged as the attack but are actually normal logs.

2) Among the 100 web logs with the highest variance in the TN class, there are in total 17 logs which are tagged as normal logs but are actually attacks. They are all in a different format, which shows that our model is also capable of finding missing attacks in the testing set.

V. Conclusion

In our present work, we introduced model uncertainty to web attack detection. We proved its value of estimating the detection result when the training dataset contains annotation errors. In our research, we used the variance provided by a Bayesian model to represent model uncertainty. A high variance shows that the ML model has less confidence in the prediction made. We proved that this lack of confidence is usually caused by data annotation errors. The security experts can check the web logs with the highest variance to correct data annotation and to refine the current attack signatures or ML-based detection model.

In the future research, we may extend our research to detecting attacks by analyzing the traffic data in other layers other than web logs in the application layer.

References

[1] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks." in *Lisa*, vol. 99, no. 1, 1999, pp. 229–238.

[2] G. Stein, B. Chen, A. S. Wu, and K. A. Hua, "Decision tree classifier for network intrusion detection with ga-based feature selection," in *Southeast Regional Conference*, 2005, pp. 136–141.

[3] S. Mukkamala, G. Janoski, and A. Sung, "Intrusion detection using neural networks and support vector machines," in *International Joint Conference on Neural Networks*, 2002, pp. 1702–1707.

[4] A. M. Vartouni, S. S. Kashi, and M. Teshnehlab, "An anomaly detection method to detect web attacks using stacked auto-encoder," in *2018 6th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS)*, Feb 2018, pp. 131–134.

[5] G. R. Hendry and S. J. Yang, "Intrusion signature creation via clustering anomalies," *Proc Spie*, vol. 6973, pp. 69–730, 2008.

[6] W. Lee, S. J. Stolfo, and K. W. Mok, "A data mining framework for building intrusion detection models," in *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*. IEEE, 1999, pp. 120–132.

[7] T. T. H. Le, J. Kim, and H. Kim, "An effective intrusion detection classifier using long short-term memory with gradient descent optimization," in *2017 International Conference on Platform Technology and Service (PlatCon)*, Feb 2017, pp. 1–6.

[8] J. Kim, J. Kim, H. L. T. Thu, and H. Kim, "Long short term memory recurrent neural network classifier for intrusion detection," in *2016 International Conference on Platform Technology and Service (PlatCon)*, Feb 2016, pp. 1–5.

[9] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access*, vol. 5, pp. 21 954–21 961, 2017.

[10] Q. Han, Q. Meikang, L. Zhihui, and G. MEMMI, "An efficient key distribution system for data fusion in v2x heterogeneous networks," *Information Fusion*, vol. 50, pp. 212–220, 2019.

[11] A. H. Sung and S. Mukkamala, "Identifying important features for intrusion detection using support vector machines and neural networks," in *null*. IEEE, 2003, p. 209.

[12] N. Farnaaz and M. Jabbar, "Random forest modeling for network intrusion detection system," *Procedia Computer Science*, vol. 89, pp. 213–217, 2016.

[13] S. Kumar and E. H. Spafford, "A pattern matching model for misuse intrusion detection," 1994.

[14] H. B. Barlow, "Unsupervised learning," *Neural computation*, vol. 1, no. 3, pp. 295–311, 1989.

[15] T. S. Pham, T. H. Hoang, and V. C. Vu, "Machine learning techniques for web intrusion detection x2014; a comparison," in *2016 Eighth International Conference on Knowledge and Systems Engineering (KSE)*, Oct 2016, pp. 291–297.

[16] "Csic 2010 http dataset," http://www.isi.csic.es/dataset/.

[17] Y. Gao, Y. Ma, and D. Li, "Anomaly detection of malicious users' behaviors for web applications based on web logs," in *2017 IEEE 17th International Conference on Communication Technology (ICCT)*, Oct 2017, pp. 1352–1355.

[18] J. Yu, D. Tao, and Z. Lin, "A hybrid web log based intrusion detection model," in *2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS)*, Aug 2016, pp. 356–360.

[19] Y. Kim, "Convolutional neural networks for sentence classification," *Eprint Arxiv*, 2014.

[20] M. Zhang, B. Xu, S. Bai, S. Lu, and Z. Lin, "A deep learning method to detect web attacks using a specially designed cnn," in *Neural Information Processing*, D. Liu, S. Xie, Y. Li, D. Zhao, and E.-S. M. El-Alfy, Eds. Cham: Springer International Publishing, 2017, pp. 828–836.

[21] J. Saxe and K. Berlin, "expose: A character-level convolutional neural network with embeddings for detecting malicious urls, file paths and registry keys," *arXiv preprint arXiv:1702.08568*, 2017.

[22] H. Zheng, Y. Wang, C. Han, F. Le, R. He, and J. Lu, "Learning and applying ontology for machine learning in cyber attack detection," in *17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications / 12th IEEE International Conference On Big Data Science And Engineering, TrustCom/BigDataSE 2018, New York, NY, USA, August 1-3, 2018*.

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[24] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," in *Advances in Neural Information Processing Systems*, 2015, pp. 2575–2583.

[25] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: representing model uncertainty in deep learning," in *International Conference on International Conference on Machine Learning*, 2016.

[26] Y. Gal, "Uncertainty in deep learning," Ph.D. dissertation, PhD thesis, University of Cambridge, 2016.

[27] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in neural information processing systems*, 1990, pp. 396–404.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.