Crafting and Detecting Adversarial Web Requests

Xinyu Gong*, Huidi Zhu†, Ruofan Deng‡, Fu Wang§, and Jialiang Lu¶
SPEIT, Shanghai Jiao Tong University
Shanghai, China
Email: {*zachary, †huidi16, ‡henri16, §wangf98, ¶jialiang.lu}@sjtu.edu.cn

***Abstract*—Deep learning (DL) techniques have provided state-of-the-art results for many machine learning tasks. In response to the increasing demand for web security, many researches have been focusing on applying DL to detect web attacks. However, these works just pay attention to the detection accuracy, not the robustness of the detection model itself. In this paper, we proved that it is possible to generate adversarial web requests by modifying only a few characters of them, which can lead the existing DL based detection model to wrong predictions. The attackers may take this vulnerability to trigger false positive alarms and even disable the whole detection model. As the defensive measure, we propose to use a combined method of density estimation and model uncertainty estimation to detect these adversaries. We report a ROC-AUC of over 95% of detecting these adversarial web requests.**

***Index Terms*—Web Attack, Adversarial Attack, Deep Neural Network, Model Uncertainty, Density Estimation**

I. Introduction

Machine learning (ML) techniques have been widely exploited in the cyber security domain. Comparing to the traditional way of detecting cyber attacks, using ML models saves the manual effort of building the corresponding attack signatures and achieves a higher detection accuracy. In the past decade, with the rise of Deep Learning (DL), many researches took advantage of Deep Neural Networks (DNN) to detect cyber attacks [1], [2]. Comparing to the conventional ML techniques like SVM, decision tree, random forest, etc..., DL has the following advantages: 1). DL models can take unstructured data as its input, which saves the effort of data pre-processing [3]. 2) Benefited from more non-linear transformations, DL models can reach an even higher detection accuracy [4], [5].

Nevertheless, DNNs have been criticized for the lack of interpretability. For conventional ML models, e.g. decision tree, the decision boundary is quite clear, the reason why an access is classified as an attack is humanly understandable. While for the DL models, the reason for classification is difficult for human to understand. This lack of interpretability becomes a more severe problem in the security domain because the system maintainer needs to take full control of the deployed detection algorithm.

Furthermore, in recent researches, the DL models are shown to be vulnerable to adversarial attacks [7]. The adversarial attacks are triggered by injecting some manually crafted data samples that can cause the DL model to make wrong predictions. It has been proved that these adversarial samples exist in both image and text classification domain [8], [9].

Original URL:
http://localhost:8080/tienda1/publico/registro.jsp?modo=registro&login=ive&password=g%21arb%F3n&nombre=Virginia&apellidos=Quitral+Blasi&email=rodis-ryan%40colorprint.info&dni=05301323F&direccion=Carrer+Sant+Joan+De+Deu+186+&ciudad=Navares+de+Ayuso&cp=31253&provincia=Granada&ntc=5239525099748134&B1=Registrar

Adversarial URL:
http://localhost:8080/tienda1/publico/registro.jsp?modo=registro&login=ive&password=g%21arb%F3n&nombre=Virginia&apellidos=Quitral+Blasi&email=rodis-ryan%40colorprint.info&dni=05301323F&direccion=Carrer+Sant+Joan+De+Deu+186+&ciudad=Navares+de+Ayuso&cp=31253&provincia=Granada&ntc=Q23v32A099748O34&B1=Registrar

Fig. 1: Example of adversarial web request in the CSIC [6] dataset. The original URL is a normal access. The adversarial request is generated by our gradient based method. The red part is the input of the user. By slightly tuning the user's input can let the DL detection model classify this request as attack.

In the previous researches of applying DL techniques to detect web attacks, their only concern is to improve the detection accuracy without considering the vulnerability of the detection model. Whether the adversarial samples also exists in DL-based web attack detection is still an open question.

In this paper, we propose to use gradient-based methods to generate adversarial web requests against the current CNN fashioned web attack detection model. We proved that by changing only a few (less than 10) characters of the URL can make the CNN-based detection model give out wrong predictions. Because CNN model only captures the local features, the location of the changed characters can be manually adjusted. As shown in Fig. 1, changing the user's input part of the URL can mislead the detection model. The attackers may take advantage of these adversarial samples to cause a large amount of false-positive alarms.

In the application scenario, many attack detection systems have an online-learning algorithm which includes the encountered traffic data with high classification confidence to enrich its training dataset. In our experiment, we present that by injecting the adversarial requests with high classification confidence in the training dataset and re-launch the training will decrease the precision of the model to zero in just a few epochs.

Following the research in [10] which analyzes the distribution of the adversarial samples in the data manifold, we use a combined method of kernel density estimation and model

uncertainty estimation to detect adversaries in CSIC 2010 [6] dataset. The combined method reaches an ROC-AUC of over 92% on detecting the generated adversarial samples.

The main contribution of this paper is concluded in the following two aspects:

- We demonstrate that the current DL-based web attack detection system can be easily broke through by adversarial samples.
- We build an effective method combining kernel density estimation and model uncertainty estimation to detect these adversarial samples.

The rest of the paper is organized as follows. In Section II, we provide some background information which are related to our research. Our approach of triggering and detecting adversarial attacks are introduced in Section III. In Section IV, we list the experiment setup and evaluate the experimental results. Finally, we conclude our presented work and propose some future research directions in Section V.

II. Related Work

A. Deep Learning Based Web Attack Detection

Many researches have been conducted to apply conventional ML methods to detect attacks in the network layer. However, these methods faced difficulties to detect web attacks, because web requests contain unstructured parts like URL and request entity.

As Convolutional Neural Network (CNN) has shown its ability to copy with text data [11], it is also applied to analyze web requests and detect attacks. In the work of [3], [12], [13], the authors use CNN model to detect web attacks through URL. [3] applies a word splitting technique to separate URL into words. The proposed method is trained and tested on the CSIC 2010 HTTP dataset where it achieves a high detection rate and a low false alarm rate. In [12], the author saves the work of splitting URL into words. In their work, the raw URL is taken as the input. Each character in the URL is embedding into a high dimension space. Their model can reach a higher detection comparing to their n-gram model baseline and security experts. [13] applies the same character-level embedding technique on the whole piece of web log. The author proves that a CNN model trained on a new classification task can be used as the feature extractor for web attack detection task.

B. Adversarial Attacks Methods

Adversarial samples in image classification task are widely known. The first attack method to craft adversarial samples is proposed in [7]. Building these samples is equivalent to solving the following optimizing problem:

$$\arg\min_{r} \|\mathbf{r}\| \text{ s.t. } \mathbf{F}(\mathbf{X}+r) = \mathbf{Y}^* \tag{1}$$

where $\mathbf{F} : \mathbf{X} \mapsto \mathbf{Y}$ is the input and output mapping of the given DNN model, $\mathbf{X}^* = \mathbf{X} + r$ is the adversarial sample, $\mathbf{Y}^*$ is the desired model output, and $\|\cdot\|$ is the chosen to evaluate the norm of the inserted adversarial perturbation r. This optimizing problem can be solved by stochastic gradient descent or L-BFGS algorithm.

Based on the research of [7], [8] argues that the existence of adversarial samples is not caused by nonlinearity or overfitting of the DNN, but the vulnerability to adversarial perturbation is DNNs' linear nature. The author propose a more efficient way of generating adversarial sample which is referred as **Fast Gradient Sign Method (FGSM)**. This method is described in the following equation:

$$X^* = X + \epsilon \operatorname{sign}\left(\nabla_X J(\Theta, X, Y)\right) \tag{2}$$

where $J(\Theta, X, Y)$ is the cost function and ϵ is a coefficient manually defined to adjust the distortion of the adversarial sample comparing to the original one.

The trade-off between adversarial success rate and the portion of the input feature being modified is discussed in [14]. The author propose that twisting only a small part of the input feature is sufficient to mislead the DNN model. By exploiting the forward derivative of a DNN, the attacker can find an adversarial perturbation that forces the model to classify the input data into a target class.

For generating adversaries in the text classification task, [9] introduces an iterative method to replace the words with their synonyms or typos. The importance of the words are ranked by their feedback gradient, and only the more important ones are replaced or removed. However, the candidate words for replacement is not ranked, the process of generating these samples is more like an attempting process.

C. Defensive Measures to Adversarial Attacks

As adversaries might threaten the system security, many researchers have proposed defensive measures against them. These measures can be mainly classified as prevention and detection. One straightforward prevention method is to actively generate adversaries, tag them with the real label, include them as a part of the training dataset, and then re-train the model to increase the its robustness [7]. [8] proposes that adding an extra item $\epsilon \operatorname{sign}\left(\nabla_X J(\Theta, X, Y)\right)$ to the cost function can increase the robustness of the DNN model.

To detect these adversarial samples, [15] proposes to add an additional *detector* network with the original DNN. The *detector* network is trained on normal and adversarial samples. The idea is similar to the first idea of prevention. They both *vaccinate* the DNN to resist potential adversaries. The problems lies in the following two aspects: The *detector* network can also be bypassed by adversarial samples against it; This method proposes that the *detector* can find the hidden pattern of adversarial samples, while does these samples share a same pattern remains an open question.

By analyzing the distribution of the adversarial samples in data manifold, [10] proposes to apply kernel density estimation and model uncertainty estimation on the prediction output. The estimation results are combined in a logistic regression model to detect adversaries. Enlightened by [10], we tried to apply this combined method in detecting adversaries in web requests. In the following sections, we will present that in the more

discontinuous feature space – web requests, this combined method is highly accurate and can reach a ROC-AUC of over 92%.

III. Triggering and Defending Adversarial Attacks in Web Security

A. Triggering Adversarial Attacks

The principle of generating adversarial samples is to manually tune the real data sample to cause classification error without changing the **real** label of the data. In the image and text classification domain, the real label is determined by human: A cat picture with adversarial perturbation seems almost unchanged to human; A sentence with some adversarial typos is still humanly understandable.

While for the web attack detection, the real label is not decided by a human, instead, it is defined by a more direct rule: web requests which cause dis-functionality or information leakage are defined as attacks, the others are labelled as normal accesses. Therefore, it is clear that there are two directions for generating adversarial web requests: 1) Tuning the normal access to let it be classified as an attack. 2) Tuning the attack to let it be classified as normal access.

However, the latter one is not feasible, because web attacks are targeting some special vulnerabilities of the web application, changing even only one letter of the attack query will disable it. Therefore, in the following sections, we focus on tuning the normal requests and let it be classified as attacks. These false positive alarms can harm the normal operation of the website. Moreover, in Section IV-C, we will show that if an online-learning algorithm is deployed, training on these adversaries can reduce the model precision to zero in just a few epochs.

We propose two rules to generate adversarial web requests:

- The adversarial perturbation can not be added to each input feature (character). The modification on the original request needs to be constrained to as few characters as possible.
- The real label should remain unchanged. This means that the tuned normal request does not harm the web site.

Based on the above two rules, also the prime iterative method and the FGSM method mentioned in Section II, we propose two methods to generate adversarial web requests. They are respectively named as **masked iterative method** and **masked FGSM method**. The steps of the algorithms are explained in Algo.1 and 2.

In Algo. 1, the added perturbation r is optimized by gradient descent method to maximize the probability of misclassification. Because we only want to change certain part of the web request, like the user input part in the URL as shown in Fig. 1, we generate a list of index I to indicate the characters that can be changed. A mask M_I is generated according to I, and for each element $M(i)$ in M_I:

$$M(i) = \begin{cases} 1, & \text{if } X_i \text{ can be changed} \\ 0, & \text{if } X_i \text{ cannot be changed} \end{cases} \tag{3}$$

Algorithm 1 Convert a normal web request X into an adversarial request with masked iterative method

Input: Web request X, the transformation of the DNN model F, the cost function J, the trainable perturbation r, the learning rate η for r, the desired output Y', and N the number of training iteration.
Output: Adversarial request X'

Get the classification result for the input $X : Y = F(X)$
if X is classified as normal request ($Y = 0$) **and** the real label of X is normal request ($Y' = 0$) **then**
  Get the index list I where the adversarial perturbation is inserted
  Generate the mask for perturbation M_I
  $i \leftarrow 1$
  Initialize r with all zeros
  for $i < N$ **do**
    Update $r \leftarrow r - \eta \frac{\partial J}{\partial r}$
    if $F(X + M_I \times r) = Y'$ **then**
      return $X + M_I \times r$
    end if
    $i \leftarrow i + 1$
  end for
end if

Algorithm 2 Convert a normal web request x into an adversarial request with masked FGSM method

Input: Web request X, the transformation of the DNN model F, the cost function J, the desired output Y', and s the step size.
Output: Adversarial request X'

Get the classification result for the input $X : Y = F(X)$
if X is classified as normal request ($Y = 0$) **and** the real label of X is normal request ($Y' = 0$) **then**
  Get the index list I where the adversarial perturbation is inserted
  Generate the mask for perturbation M_I
  while $\epsilon < 1$ **do**
    $\epsilon = \frac{i}{s}$
    if $F(X + M_I \epsilon \operatorname{sign}(\nabla_X J(\Theta)) = Y'$ **then**
      return $X + M_I \epsilon \operatorname{sign}(\nabla_X J(\Theta))$
    end if
  end while
end if

where X_i indicates the $i-th$ character in the web request X.

In Algo. 2, the perturbation is defined as $\epsilon \operatorname{sign}(\nabla_X J(\Theta))$. For web requests, it is more difficult to find their adversarial counterparts. Therefore, we set the parameter s to generate different ϵ. Moreover, different combinations of M_I and ϵ are tested to generate the corresponding adversarial counterparts for each web request as much as possible.

B. Defending Adversarial Attacks

1) Kernel Density Estimation: Kernel density estimation is a common non-parametric method for estimating the probabil-

ity density function given a set of data samples. As illustrated in [10], it is viewed as a measure of how far x is from the submanifold for sblings of x in the same class. Its definition is given in the following equation:

$$\hat{f}_h(x) = \frac{1}{N}\sum_{i=1}^{N} K_h\left(x, x_i\right) \tag{4}$$

where x is the desired data point, and $x_{i\in N}$ is the known data point having the same label as x. K_h is the kernel function. In our work, we choose the Gaussian kernel function:

$$K_h(x, x_i) \sim \exp\left(-\|x - x_i\|^2/h^2\right) \tag{5}$$

where h is a parameter for smoothing, which is often called *bandwidth*. It needs manual tuning, while in section IV-D we will demonstrate that different h can affect detection performance on adversaries, but influence is not essential.

2) Model Uncertainty: The density estimation analyze the prediction confidence based on the geometric distance between the data point and the data manifold. While the uncertainty output of Bayesian models interpret the prediction confidence in a different way. In [16], the author has proved that by using dropout techniques, it is also possible to obtain the uncertainty estimation for a given DNN model. Dropout [17] is firstly introduced by as an technique to prevent overfitting. [16] has proved that inserting dropout can be regarded as a Bayesian approximation for estimating model uncertainty. Therefore the model uncertainty can be obtained without changing the DNN architecture.

To get the model uncertainty in DNN model, the dropout mask $r^{(l)}$ for layer l is sampled for a Bernoulli distribution where the dropout rate is manually set to p.

$$r^{(l)} \sim \text{ Bernoulli } (p) \tag{6}$$

$$\widetilde{\mathbf{y}}^{(l)} = \mathbf{r}^{(l)} * \mathbf{y}^{(l)} \tag{7}$$

$$z^{(l+1)} = \mathbf{w}^{(l+1)}\widetilde{\mathbf{y}}^{l} \tag{8}$$

$$y^{(l+1)} = f\left(z^{(l+1)}\right) \tag{9}$$

$y^{(l)}$ is the original output of layer l, $w^{(l+1)}$ the weight matrix of layer $l+1$, and f the activation function for layer $l+1$. The model uncertainty is approximated by sampling $r^{(l)}$ for T times as shown in the following equation:

$$\sigma^2 = \frac{1}{T}\sum_{i=1}^{T}(\hat{y}_i - \frac{1}{T}\sum_{i=1}^{T}\hat{y}_i)^2 \tag{10}$$

where $\hat{y}_i$ is the output of the model, and σ^2 is the variance of the model output which is interpreted as model uncertainty.

IV. Experiments and Evaluation

A. Web Attack Detection Model Setup

CNN fashioned models are widely used in web attck detection, therefore we also built our detection model based on this idea. We implement our web attack detection model based on the design in [13]. The detection performance of our model is compared to the one in [3] on the public CSIC 2010 dataset. In our implementation, each character in the web request is embedding into a 5-D space, the kernel width is set to {3, 5, 7, 9}, and the number of each type of kernel is set to 100. The produced latent features is passed to three fully connected layers with dropout. The numbers of neurons in each fully connected layer is set to {256, 64, 2}, and the dropout rate is set to 0.5. Cross-entropy loss function is used to optimize the model. The model is trained using PyTorch API with Adam optimizer [18].

We follow the dataset setup in [3]: 36000 normal requests and 24668 abnormal requests are splited into three parts: training, validation, and test dataset. The portion of the training set is 70%, 5% for validation set, and 25% for test set. The data distribution of each part is shown in Table I.

TABLE I: Data distribution.

	Training	Validation	Test
Normal	25200	1800	9000
Attack	17268	1233	6167
Total	42468	3033	15167

In the work of [3], the author split the URL into words by special characters like /, &, =, +, etc. While in our implementation, we save this effort of word splitting by directly embedding characters into high dimension space. With this simpler data pre-processing, our detection model is still able to outperform the previous one. Three metrics are used to evaluate the detection model: *detection rate*, *false alarm rate*, and *accuracy*. The formulas of the three metrics are shown in Eqn. 11, 12, and 13.

$$Detection\ Rate = \frac{TP}{TP + FN} \tag{11}$$

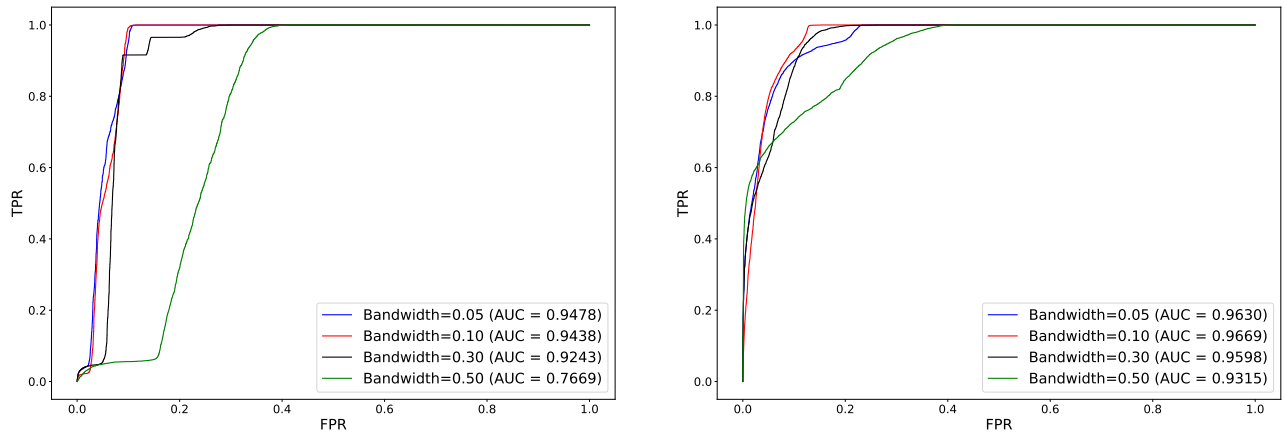
$$False\ Alarm\ Rate = \frac{FP}{FP + TN} \tag{12}$$

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{13}$$

As shown in Table II, our web attack detection model is highly accurate on the CSIC dataset. The built model surpasses the model in [3] in all three metrics.

TABLE II: Evaluating results.

	Detection rate	False alarm rate	Test accuracy
Model in [3]	95.35%	1.37%	96.49%
Our model	**95.38%**	**0.56%**	**97.79%**



(a) Masked iterative attack.

(b) Masked FGSM attack.

Fig. 2: ROC result for the kernel density estimation method with different kernel bandwidth setup.

Original URL #1:
http://localhost:8080/tienda1/miembros/editar.jsp?modo=registro&login=juan&password=Le%3FGioNario&nombre=Leonelo&apellidos=Bennasar&email=bojanic-gidra-tolan%40elcarritodelacompra.ma&dni=30503437D&direccion=Cami%F1o+Do+Monte+147%2C+10%3FH&ciudad=Santib%E1%F1ez+de+B%E9jar&cp=29450&provincia=Palencia&ntc=0462514258960948&B1=Registrar

Adversarial URL #1:
http://localhost:8080/tienda1/miembros/editar.jsp?modo=registro&login=juan&password=L.%FJBikNIrio&nombre=Leonelo&apellidos=Bennasar&email=bojanic-gidra-tolan%40elcarritodelacompra.ma&dni=30503437D&direccion=Cami%F1o+Do+Monte+147%2C+10%3FH&ciudad=Santib%E1%F1ez+de+B%E9jar&cp=29450&provincia=Palencia&ntc=0462514258960948&B1=Registrar

Original URL #2:
http://localhost:8080 /tienda1/publico/autenticar.jsp?modo=entrar&login=ilana&pwd=opinable&remember=off&B1=Entrar

Adversarial URL #2:
http://localhost:8080 /tienda1/publico/autenticar.jsp?modo=entrar&login=ilana&pwd=SpA~a~*E&remember=off&B1=Entrar

Fig. 3: Generated adversarial URLs. The red part is the password domain where the adversarial perturbation is inserted.

B. Adversary Generation

Experiments are conducted on the built web attack detection model and CSIC dataset. To evaluate our method, we only perturb those normal requests in the test dataset which are correctly classified by the detection model. Because there's no reason to perturb these requests which are already misclassified. For the masked iterative method, the learning rate is set to 0.05 and the training iteration N is set to 100 to reduce training time. For the masked FGSM method, the step size s is set to 10. We generate 9000 adversarial web requests using each of these two methods. As shown in Fig. 3, the perturbation can be inserted in the *password* domain and let these normal request be classified as attack. This relieves the drawback of the CNN fashioned detection model: convolutional operations can only capture local features and they treat each sub-part of the web request equally, therefore the perturbation in the user input domain can affect the model.

C. Model Degradation

Forcing the detection model to make wrong predictions is one of the hazards of the adversarial attacks. The other one is ruining the model's detection performance by injecting adversaries in the training dataset. Many learning-based web attack detection system may include these traffic data with high prediction confidence in the training dataset to expand its volume. In our experiments, we include those adversarial requests with prediction confidence over 90% in our training dataset. As we have mentioned in the previous section, we generate 9000 adversarial attacks with each of the attack method. Among the masked iterative attacks, 8126 of them are predicted as attack with confidence over 90%, while for the masked FGSM attacks, 6191 of them are the highly confident ones.

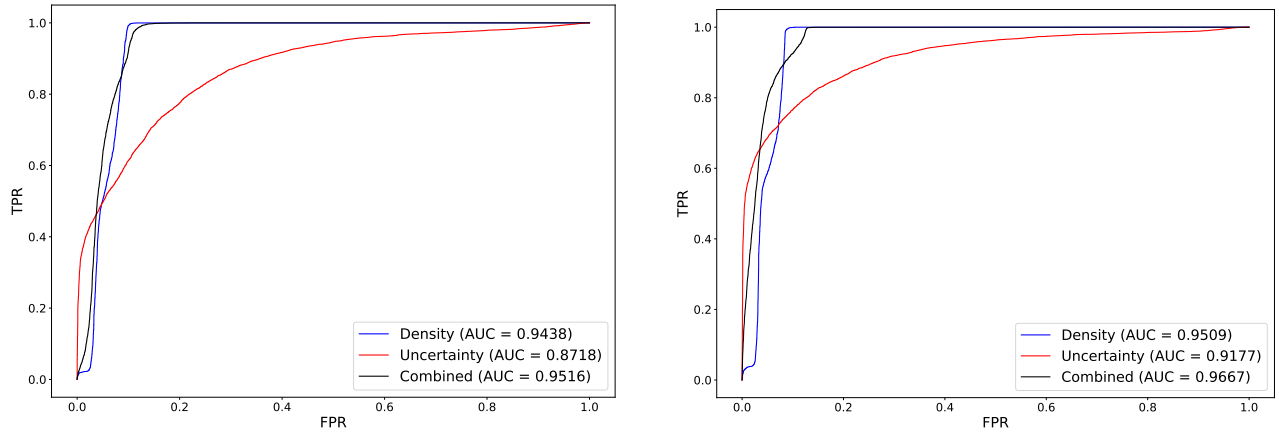
As shown in Table III, we include each of the two attacks which have high prediction confidence in the training set, and re-train the detection model. The false alarm rate will rise to 100% after only one training epoch.

TABLE III: Evaluating results.

	Detection rate	False alarm rate	Accuracy
Original model	95.38%	0.56%	97.79%
With masked-iterative	100.00%	100.00%	40.22%
With masked-FGSM	100.00%	100.00%	40.22%

D. Adversary Detection

We have generated adversarial web requests using two different methods on the CSIC dataset. We try to detect these attacks using kernel density estimation and model uncertainty estimation. The kernel density estimation is applied on the output of the last layer in our detection model. This attack detection problem is a binary classification task, therefore the output dimension of the last layer is 2. In our experiments, we firstly use the kernel density estimation to detect adversaries and exploit the influence of kernel bandwidth. The logistic regression model is chosen as the classifier. The detection



(a) Masked iterative attack

(b) Masked FGSM attack.

Fig. 4: ROC results for different classifiers and different attacks.

performance corresponding to different bandwidth is shown in Fig. 2. We report the detection performance using the AUC metrics. It can be observed that when setting bandwidth between 0.05 and 0.3, the detection performance remains almost the same. While setting bandwidth to a much larger value has a negative impact on the performance.

Then the kernel density and model uncertainty is combined in a single logistic regression model. The detection performance is compared in Fig. 4. We selected the best kernel bandwidth according to the above experiments. It can be observed that combining the two detection methods can improve AUC. For both attacks, the combined methods can reach AUC score of over 95%. For the separate applications, the kernel density estimation performs better than the model uncertainty. Nevertheless, the separate application of model uncertainty is still an effective method to detect adversaries through the AUC score.

V. Conclusion

In our presented work, we presented the existence of adversarial web requests that causes mis-classification of the current CNN fashioned attack detection model. To detect these adversarial attacks, a combined method of kernel density estimation and model uncertainty estimation is exploited.

In the future research, we may extend our work to other neural network architectures like RNN, which is highly efficient to copy with sequential data like web requests. Moreover, as we have presented methods to detect adversarial attacks, it is also a good solution to increase the robustness of the DL model and find the preventive measures against these attacks.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

[7] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

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

[9] S. Samanta and S. Mehta, "Towards crafting text adversarial samples," *arXiv preprint arXiv:1707.02812*, 2017.

[10] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner, "Detecting adversarial samples from artifacts," *arXiv preprint arXiv:1703.00410*, 2017.

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

[12] 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.

[13] 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*.

[14] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 372–387.

[15] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, "On detecting adversarial perturbations," *arXiv preprint arXiv:1702.04267*, 2017.

[16] 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.

[17] 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.

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