Locate-Then-Detect: Real-time Web Attack Detection via Attention-based Deep Neural Networks

Tianlong Liu [1] , **Yu Qi** [2,*] , **Liang Shi** [3] and **Jianan Yan** [1]
[1]Alibaba Cloud Intelligence Business Group, Alibaba Group, China
[2]College of Computer Science and Technology, Zhejiang University, China
[3]AI&Data Department, Dingxiang Tech.Inc, China
tim.ltl@alibaba-inc.com, qiyu@zju.edu.cn,
liang_shi@hotmail.com, jianan.yjn@alibaba-inc.com

Abstract

Web attacks such as Cross-Site Scripting and SQL Injection are serious Web threats that lead to catastrophic data leaking and loss. Because attack payloads are often short segments hidden in URL requests/posts that can be very long, classical machine learning approaches have difficulties in learning useful patterns from them. In this study, we propose a novel Locate-Then-Detect (LTD) system that can precisely detect Web threats in real-time by using attention-based deep neural networks. Firstly, an efficient Payload Locating Network (PLN) is employed to propose most suspicious regions from large URL requests/posts. Then a Payload Classification Network (PCN) is adopted to accurately classify malicious regions from suspicious candidates. In this way, PCN can focus more on learning malicious segments and highly increase detection accuracy. The noise induced by irrelevant background strings can be largely eliminated. Besides, LTD can greatly reduce computational costs (82.6% less) by ignoring large irrelevant URL content. Experiments are carried out on both benchmarks and real Web traffic. The LTD outperforms an HMM-based approach, the Libinjection system, and a leading commercial rule-based Web Application Firewall. Our method can be efficiently implemented on GPUs with an average detection time of about 5ms and well qualified for real-time applications.

1 Introduction

Most enterprises provide Web services open to the public and thus are prone to Web attacks. A large portion of these attacks are Cross-Site Scripting (XSS) and SQL Injection (SQLi), which are two of the most frequently seen threats toward Web applications according to the OWASP report [1]. Through successful attacks, attackers can obtain more-than-need access to Web resources such as databases and major sensitive data leaking/loss would happen.

Researchers have developed various approaches to deal with the Web attack problem. Since XSS and SQLi vulnerabilities are basically caused by insufficient input validation or type checking, defensive coding practices have been widely applied to mitigate XSS and SQLi attempts [Johari and Sharma, 2012; Gupta *et al.*, 2014]. However, it is usually difficult for the programmers to keep every input check completely, rigorously, and correctly. Black and white box security testing can help discover Web risks [Gupta *et al.*, 2014; Kindy and Pathan, 2011], however, they still have limitations. The white-box testing statically checks code structure and logic safety. It works poorly when the code contains dynamic contents. The black-box testing checks possible XSS and SQLi points with attempting payloads of all known types at a website [Bau *et al.*, 2010]. This may slow down the website responses and can have a side effect of real data damage caused by payload replaying. A popular option for XSS and SQLi detection is Web Application Firewall (WAF), which monitors real-time Web traffic. Traditional WAF usually works in a knowledge-driven way, and typically integrates a set of heuristic rules. Despite that rules are usually effective, they usually have difficulties in detecting unseen attacks. Besides, the development and maintenance of the rules highly depends on specialized human knowledge and experience. Recent studies trend to build WAF in a data-driven way, which learns the rules directly from Web data. One pioneer study to this end is the Libinjection system [Galbreath, 2012] for SQLi detection. Different from regular expression based rules, the input strings are firstly tokenized with semantic syntax, and then the system learns the 'malicious signatures' of SQLi from labeled data. The Libinjection method obtains good performance upon many types of SQLi attacks and has been used in several industry products such as the Google Chrome.

Machine learning algorithms especially deep learning models [LeCun *et al.*, 2015] are powerful in data-driven learning, and many efforts have been made to improve generalization ability in Web attack detection by using them. Advanced machine learning models such as Neural Networks, Support Vector Machines [Pinzón *et al.*, 2010], and probability based models [Kar *et al.*, 2016] have shown effectiveness for XSS or SQLi classification. To capture unknown attacks, anomaly detection approaches such as the HMM-Web [Corona *et al.*, 2009] have been proposed. Compared with

*Corresponding author.
[1]https://www.owasp.org/index.php/Top_10-2017_Top_10

knowledge-driven methods, data-driven approaches usually have a better generalization ability in detecting unseen attacks. Besides, they are more friendly in quick deployment and capable of rapid updating.

Although machine learning methods are promising to provide superior performance in Web attack detection, they are still not practical in real-time applications. The first problem in Web attack detection lies in that, the attacks are usually short text segments called payloads hidden in background strings that could be very long, which make it difficult for model training. Specifically, URL requests/posts are often large in size, sometimes can be several-thousand-character long, while a payload can be as short as 6 characters. The second problem is real-time detection. Keeping a website's response fast is critical to its Web service, which is often challenging for machine learning based detections such as deep neural networks. Thirdly, traditional machine learning methods are mostly so called black-box methods. The black-box property of machine learning models makes it difficult for WAF operations and incident analysis, which is also a serious barrier to practical applications.

In this paper, we propose a novel Locate-Then-Detect (LTD) system that can precisely detect Web threats such as SQLi and XSS in real-time by using attention-based deep neural networks. In our approach, an efficient Payload Locating Network (PLN) [Ren *et al.*, 2015] is employed to propose the most suspicious regions from large URL requests/posts. Then the proposed regions are fed to Payload Classification Network (PCN) to classify the malicious regions. With the Locate-Then-Detect process, the attack classifier can focus more on classifying malicious segments from suspicious ones thus higher accuracy can be reached. Besides, with payload locating, 95% of the background contents can be ignored, which highly reduces computational costs. Further, the LTD method is interpretable and is able to point out the exact attack payloads and their types, which is beneficial for security investigation.

The LTD system is evaluated on both a benchmark and a real-world Web traffic dataset. Our approach outperforms other methods such as an HMM-based method, the Libinjection system, and a leading commercial rule-based WAF, and can be efficiently implemented on GPUs with an average detection time of about 5ms and well qualified for real-time applications.

2 Method

The Locate-Then-Detect (LTD) system consists of two main modules: a Payload Locating Network (PLN) to propose suspicious regions from large requests/posts, and a Payload Classification Network (PCN) to accurately recognize attacks from the suspicious regions. The framework of the LTD system is illustrated in Figure 1. Training of LTD require large amount of annotated data, which can be highly strenuous and time-consuming. Therefore, we specially propose a Hidden Markov Model (HMM)-based anomaly detection system to facilitate attack annotating process.

2.1 Payload Locating Network (PLN)

PLN scans a Web request and extracts the suspicious fragments from long texts. The idea of region proposal has been successfully applied in image processing [Ren *et al.*, 2015]. It can facilitate Web attack detection with higher efficiency and accuracy. PLN is a multilayer neural network which takes Web request text with fixed length as its input, and outputs the location and suspicious confidence of the regions.

Web Request Embedding

The Web request text is first embedded in character level [Kim *et al.*, 2016; Xiao and Cho, 2016]. A *embedding layer* projects one-hot presentations into a k-dimensional continuous vector space $\mathbb{R}^k$ by multiplying the vectors with a weight matrix $\mathbf{W} \in \mathbb{R}^{k\times|\mathbf{V}|}$, where $|\mathbf{V}|$ is the number of unique characters in vocabulary. Suppose the length of an input text is L. Then the input is an $L \times k$ matrix. Before text embedding, the characters that are not included in ASCII set have been removed. To guarantee that all inputs share the same size, we set a maximum text length L_{max}, and pad all embedding matrices to $L_{max} \times k$ with zeros.

Feature Extraction

We make embedding matrices go through a *feature extraction layer* to learn useful and compact features. A variant of the Xception model[Chollet, 2016] is used for feature extraction. To accelerate computing, we use thin feature maps with small channels, which can highly improve the speed without loss much accuracy [Li *et al.*, 2017].

Region Proposal

After feature extraction, Web requests are presented in the feature space. Then we slide several mini-networks along the feature maps to detect suspicious segments. This network takes an $n \times m$ spatial window of the input feature map. Two sibling $1 \times m$ convolutional layers following the mini network–a region regression layer (*reg*) and a region classification layer (*cls*). In order to preserve the semantic integrity of those vectors in the embedding tensor, we let m equal the embedding size of the character vector. Simultaneously, we predict p proposals at each sliding-window location. The p proposals, also called *anchors*, have the same width and are centered at the sliding window with different scale ratios (the *heights* of proposals are different). So the *reg* layer outputs $2p$ coordinates of p proposals, which are the start and the end positions of the payload in the sequence. The *cls* layer has $2p$ scores that predict probabilities of attack payloads for each proposal. For the feature map of a size $W \times H$, there are $H \times p$ anchors in total.

However, not all anchors are valid. Denote the start and the end positions of an anchor as $start$ and end. The filter conditions are defined as follows:

1. Anchor size should be greater than 3, that is $end - start > 3$;
2. Anchor should be in the range of $[0, L_{max}]$, so we set $start = max(start, 0)$ and $end = min(end, L_{max})$;
3. Anchor should not contain too many zero padding parts, that is $start < L$ and $end - m < L$, where m is a hyper parameter.

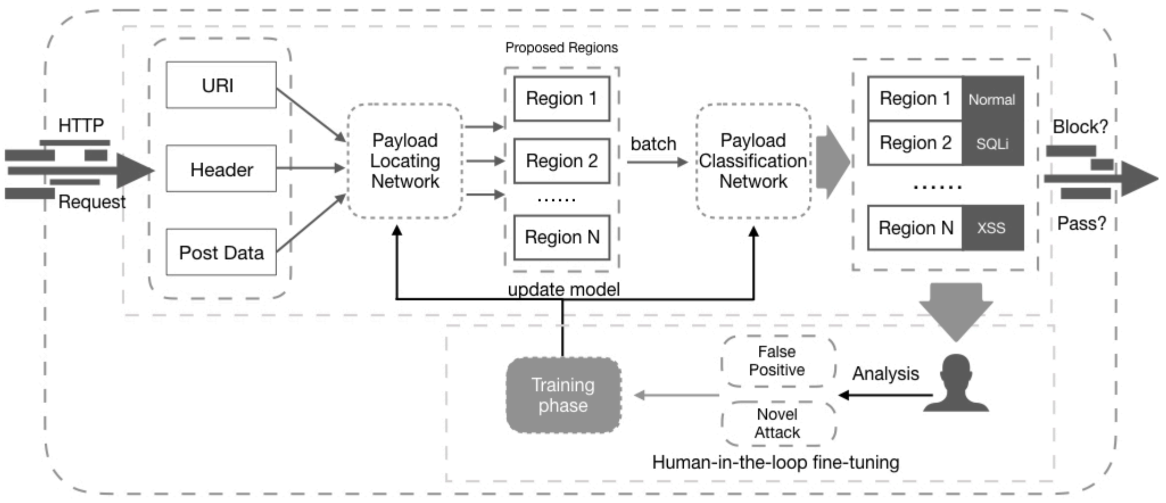

Figure 1: Detection phase of the Locate-Then-Detect framework.

For training the model, each anchor needs to be classified as a payload or not. We assign a positive label to two types of anchors:

1. An anchor with the highest IoS (Intersection over Sequence) overlap with a ground-truth box.
2. An anchor that has an IoS greater than 0.5 with any ground-truth box.

A negative label is assigned to an anchor if its IoS is less than 0.2 for all ground-truth boxes. Anchors that neither have a positive label nor a negative label are ignored in the training stage. Usually the negative anchors are much more in quantity than the positive anchors, and the ratio of negative anchors to positive ones is adjusted to 3:1 when $N_{neg} : N_{pos} > 3 : 1$. The loss function of PLN contains two parts:

$$\begin{aligned} L(l_i, pos_i, p_i, pos_i^*) &= \frac{1}{N_{cls}} \sum_i L_{cls}(l_i, p_i) \\ &+ \lambda \frac{1}{N_{reg}} \sum_i l_i L_{reg}(pos_i, pos_i^*), \end{aligned} \tag{1}$$

where i denotes the index of an anchor in a training batch, the ground-truth label l_i is 1 if the anchor is positive, otherwise 0. pos_i and pos_i^* here are ground-truth bounding positions associated with the positive anchor and a predicted bounding position, respectively. L_{cls} is a log loss for the anchor classification. The second part of the loss function, $l_i L_{reg}$ means the regression loss is only for positive anchors ($l_i = 1$), otherwise the $l_i L_{reg}$ is 0 ($l_i = 0$). For the regression loss L_{reg}, we choose the smooth–L_1[Girshick, 2015]

$$smooth_{L_1}(x) = \begin{cases} 0.5x^2 & if\ |x| < 1, \\ |x| - 0.5 & otherwise. \end{cases} \tag{2}$$

x denotes the loss between the ground-truth coordinates and predictions. There is a hyper-parameter λ in Eq.(1). It controls the importance between the losses of two parts. For training the PLN, we set $\lambda = 1.0$, set N_{cls} as the mini-batch size, and set N_{reg} as the number of anchor locations.

2.2 Payload Classification Network (PCN)

After PLN, the suspicious regions of a request are located. Then the PCN is applied to further analyze these regions and recognize malicious attacks from the candidates. The PCN is a text classification neural network [Kim, 2014] sharing the request embedding layer with the PLN, and outputting the label (attack types or benign) of the given segment.

The model architecture of the PCN use 5-layer convolution networks with different filter windows h, and each convolution network is followed by a max-overtime pooling operation [Collobert *et al.*, 2011]. The diverse filters guarantee that the PCN is able to identify attack precisely with multiple features. These features are concatenated and passed into three fully connected layers with rectified linear units. The output layer is a soft-max layer whose output is probabilities of different attack types.

The loss function of PCN is composed of a multi-class cross entropy loss and an L_2 regular term:

$$L_{PCN} = -\frac{1}{N} \sum_i^N \sum_j^K y_{ij} log(p_{ij}) + \lambda \|\mathbf{W}\|, \tag{3}$$

where y_{ij} denotes the true label of the ith sample in a training batch belonging to the jth class, and p_{ij} denotes the corresponding prediction probability. The first part is the multi-class cross entropy loss, and the second part is the L_2 regular term.

2.3 Web Attack Annotation

Training of the LTD system requires large amount of labeled attack data, which is a very resource-consuming work for security experts. Therefore, we propose using an anomaly detection system, the HMM-Web [Corona *et al.*, 2009], to collect attack samples from Web traffic. The HMM-Web anomaly detection system consists of a large number of HMM models. Each HMM model is trained for a specific parameter value in a particular URI of a targeted host. The

HMM-Web can annotate payload locations. Then all the detected anomalies will be gathered and fed into a rule-based attack detection system. For those anomalies that are able to be classified into specific attack types (e.g. SQLi, XSS, etc.), start and end positions are marked in their original requests.

For example, for uri_1 = /a.php?id=1&name=1' and 1=1, each parameter value is extracted first. Among the two values ($\{val_1 : 1, val_2 : 1' \; and \; 1 = 1\}$), val_2 is an anomaly detected by an HMM model, and it is correctly classified as a SQLi attack. We then find the location of val_2 in uri_1. The annotated data is as: [Start (17), End (27), Label (1)].

2.4 Locate-Then-Detect (LTD) System

The LTD system merges the PLN and PCN models that are trained separately. A feedback work-flow is used to fine-tune the whole network in a human-in-the-loop way.

Attack Detection

The framework of the LTD system is shown in Figure 1. A given request is parsed and split into several different parts (e.g., URI, User-Agent, and Cookie). Each part then is fed into the PLN, and the model would predict whether a region is a suspicious payload or not and the model would give payload positions as well. All the suspicious fragments proposed by the PLN are processed by the attack classification model. If all fragments are classified as benign, the request will be passed as benign. Otherwise the request will be marked as malicious with the attack type given by the classifier. In the right of the Figure 1, *Region 2* is classified as SQLi attack and *Region N* as XSS attack, thus the request will be mitigated because of its malicious contents.

Human-In-The-Loop Tuning

In order to make the system more robust, and to hopefully be able to discover zero-day attacks, we integrate our system with a work-flow for diagnosis and incrementally updated weights for the PLN and the PCN, which could help the models to detect never-seen attacks. In the real-world network environment, the number of Web attacks is far less than the number of benign Web requests. Security experts are able to do analysis on the small amount of the detected attacks. As shown in Figure 1, the system's false positives will be further used to improve the PCN. For the fine-tuning PLN, those samples should be annotated with payload positions and attack types, then they would be fed back into the training samples for the next iteration.

3 Experiments

Experiments are carried out to evaluate our LTD system. Two datasets of one CSIC 2010 benchmark dataset and one real-world Web traffic dataset are used. We compare our methods with three leading Web attack detection methods. 1) A rule-based commercial Web Application Firewall (RWAF), in which different rules are assigned for parameters in the GET and the POST data. 2) Libinjection [Galbreath, 2012], which is a leading SQL Injection detection system via lexical syntax-aware analysis. 3) HMM-Web, an anomaly detection based Intrusion Detection System (IDS) implemented with Hidden Markov Model (HMM) methods [Corona *et al.*, 2009]. The HMM-Web can only be trained with a great amount of data, so we only introduce it in the experiment of the large real traffic dataset.

3.1 Implementation Details

Both PLN and PCN are optimized using Adam optimizer [Kingma and Ba, 2014]. For the PLN, the learning rate and weight decay are $1e-6$ and 0.99 respectively. For the PCN, the parameters are set to $1e-5$ and 0.995, respectively. The number of candidate regions is set to 3.

For PLN, an input sequence is projected into a tensor $\mathbf{T}_s$ with shape of $1000 \times 8 \times 1$. The size of the feature map is 32×8, so there will be $32 \times 75 = 2400$ anchors in total. We also use a Non-Maximum Suppression method to reduce the redundant anchors, and the IoS (Intersection over Sequence) threshold is set to 0.7. Moreover, in the training phase, we set a limit of the ratio of negative anchors to positive anchors as 3:1 when $N_{neg} : N_{pos} > 3 : 1$. For PCN, the maximum length is set to 512 and the embedding size is set to 64. In detection phase, we select the top-3 ranked proposals per sequence from the PLN and pass them to the PCN for classification.

3.2 Performance on the CSCI Benchmark

The CSCI 2010 dataset contains a generated traffic targeted to an e-Commerce Web application[Nguyen *et al.*, 2011]. This dataset contains more than 25,000 anomalous requests and 36,000 benign requests. Human experts manually review all the samples to label attack types. In the labeled dataset, we have 2,072 SQLi and 1,502 XSS samples. We treat SQLi and XSS detection as a two-class classification problem. Only SQLi/XSS samples are regarded as attacks while others are benign, even they would be other types of attacks.

As shown in Table 1, LTD method outperforms RWAF on both precision and recall. The LTD method has the same perfect 100% precision as the Libinjection, while the LTD has a much higher recall than the Libinjection. In term of the F1-score, the LTD method outperforms the other two methods.

3.3 Performance on Real-World Traffic

Datasets

To evaluate the strength of the LTD system, we construct a *RealDataset* from real-world Web traffic. In the *RealDataset*, we collect 3 millions Web traffic logs including 38,600 attack samples from eight Web servers of a busy cloud environment. The attack samples are labeled with SQLi or XSS by security experts.

Results

The performance of LTD attack detection on the real traffic is shown in Table 3. LTD obtains the highest precision. HMM-Web achieves the highest recall, slightly better than LTD, while it suffers from high FPR, which is unacceptable for real-world WAF applications where the FPR is usually required to be less than 0.01%. For Web attack detection, there is a trade-off between the FPR and recall, while a low FPR is a prerequisite in a production environment. A system with high FPR may block normal user requests which is unacceptable for real-time detection. In this experiment,

Method	#Detect	TP	TN	FP	FN	Precison	Recall	Acc	FPR	F1-score
RWAF	3325	3323	35998	2	251	99.93%	92.97%	99.361%	0.005%	0.963
Libinjection	2002	2002	36000	0	1572	**100%**	43.980%	96.02%	**0.0000%**	0.611
LTD (ours)	3513	3513	36000	0	61	**100%**	**98.29%**	**99.845%**	**0.0000%**	**0.991**

Table 1: Performance on the CSCI Dataset. #Detect denotes number of total detections; TP, TN, FP, FN are true positive, true negative, false positive and false negative, respectively.

Dataset	Benign URLs	Malicious URLs	Attack type(s)
RealDataSet	3000000	38600	SQLi,XSS

Table 2: Description of the real dataset.

the FPR of HMM-Web is significantly higher than LTD's. It is because as an anomaly-based approach, HMM-Web would classify anything as malicious if it is unseen before. Besides, when Web applications are updated, the HMM-Web has to be re-trained with new samples to comply with new user behavior patterns, otherwise the FPR will increase significantly. According to the reasons stated above, the HMM method is not a good candidate for real-time detection, it's otherwise a good choice for offline ad-hoc analysis.

The Libinjection method is widely adopted in many real-world applications such as Google Chrome. It obtains a perfect 100% precision (zero FPR) the same as what LTD has. However, its recall is only slightly larger than 70%, while LTD detects more than 99% of the attacks. Compared with Libinjection, the LTD system achieves much higher detection rate. In Table 4, we list several samples that Libinjection mis-classifies, while LTD detects them correctly. The token sequence of the 1st and the 2nd payload **ooUEn**, **s&1o.** are not in the fingerprint database. Other token sequences, such as **s&1o(**, **s&1of**, **s&1os**, appear as attacks in the fingerprint list. However, the only difference among them is their last character. Obviously, the last two payloads in Table 4 are benign requests, but their token sequences **1c** and **nc** are listed in the attack fingerprint list of Libinjection. From the above examples we could see that even though the Libinjection system has analyzed the lexical and syntax contents of requests, it still could not avoid certain mis-classifications. Compared with Libinjection, the LTD system based on deep learning methods shows better flexibility, stronger generalization and higher adaptability to avoid those mis-classifications.

Compared to the RWAF, LTD outperforms it in both recall and precision. In term of the F1-score, the LTD method outperforms all other three methods.

3.4 Effectiveness of Payload Proposal

In order to evaluate the importance of payload proposal, experiments are conducted to explore the influence of payload proposal on both efficiency and accuracy.

Efficiency

In this experiment, we compare LTD with or without payload proposal. In the system without payload proposal (VPCN), the parameters of a URL are split into a Key-Value format, and the values are regarded as the regions to input to PCN for attack classification. We compare VPCN and LTD with 10,000 randomly sampled real requests. The experiments are conducted on both CPU and a GPU environment. The setting and results are shown in Table 5.

As shown in Table 5, with the CPU environment, the LTD method's RT_{avg} (detection time in average) is 8 times faster than those of the VPCN. While on the GPU environment, the RT_{avg} of the LTD method is only 5.58 ms, which is nearly 6 times faster than VPCN.

With payload proposal, the LTD system is computationally much more efficient because there is 27.5% URLs having too many parameters to be split (13 parameters or more), and the split function consumes lots of time. In real-word Web environment, an typical URL may contain as many as dozens of parameters, which requires lots of split operations. Moreover, many Web developers tend to rewrite original URLs with URL-rewrite module to hide the parameters for several reasons. Under this circumstances, rule-based split method would take complicated computation to extract values. Compared with traditional methods, LTD reduces the computational time by restricting the detection scope in a controllable way in case of too many parameters presenting in URLs. Additionally, LTD also avoids the problem that the parameters are resolved inaccurately or very slowly in URL rewriting.

Accuracy

The second experiment is carried out to evaluate accuracy improvement of the payload proposal. We compare LTD with a typical char-level CNN approach [Kim *et al.*, 2016] that classifies attacks from raw requests instead of proposed regions. For fair comparison, we re-train both LTD models and CNN on a new-constructed training data. We randomly pick 3,200,000 normal web traffic logs from a real-world cloud platform and 800,000 attack samples from our attack database by random sampling to construct a training dataset. The testing dataset is constructed by randomly extracting 100,000 normal Web traffic logs at different days from a popular cloud platform to form the benign testing samples. We randomly replace one parameter value by an arbitrary SQLi/XSS attack payload for each URL of the previously extracted benign 100,00 logs to form the malicious samples.

As illustrated in Table 6, The FPR and FNR of the CNN model are significantly higher than those of LTD. It mostly results from that some payloads are short in length while the URL request can be very long. Certain attack payloads could be as short as 6-char-long. These short payloads can hide themselves in a relatively long background string, and these background strings reduce the performance of the CNN model due to the poor generalization. Therefore, it would be difficult for the char-level CNN model to learn and detect such a short malicious value hiding in a long URL. The locating mechanism of LTD enables the detection system to discover short attack payloads that hide themselves in long

Method	#Detect	TP	TN	FP	FN	Precison	Recall	Acc	FPR	F1-score
RWAF	37856	37809	2999953	47	744	99.87%	97.95%	99.972%	0.0015%	0.989
HMM-Web	38752	38600	2999848	152	0	99.61%	**100%**	99.994%	0.0051%	0.998
Libinjection	27638	27638	3000000	0	10962	**100%**	71.60%	99.639%	**0.0000%**	0.834
LTD (ours)	38548	38548	3000000	0	52	**100%**	**99.86%**	**99.998%**	**0.0000%**	**0.999**

Table 3: Performance on the real-world traffic. #Detect denotes number of total detections; TP, TN, FP, FN are true positive, true negative, false positive and false negative, respectively.

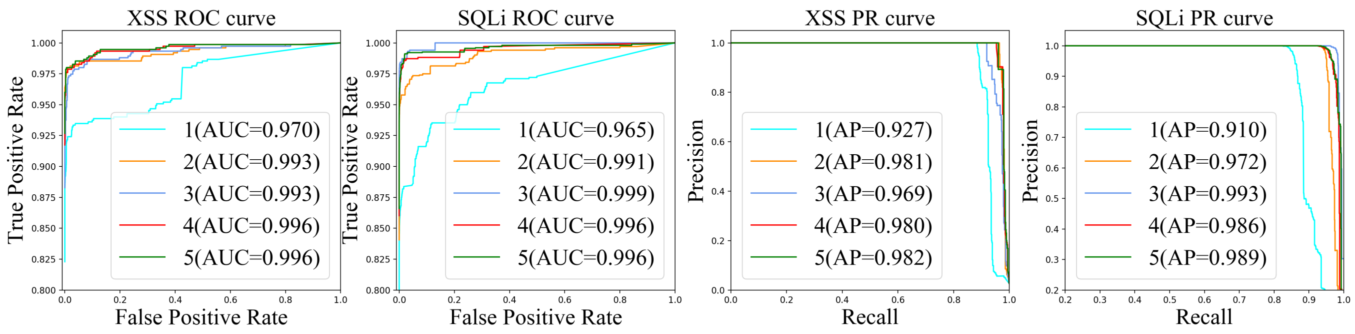

Figure 2: The performance of LTD with different number of proposals,the numbers "1,2,3,4,5" in the legends represent the number of proposals

Value	Libin	LTD	Label
*/UNION SELECT password FROM users–	×	√	√
' or 1<@. union select @@version,version()#	×	√	√
78–cfb85327-9f81-43e8-9206-32341b8be733	√	×	×
image/*	√	×	×

Table 4: Misclassified examples of Libinjection.

Test environment	Method	$RT_{avg}(ms)$
MacBook Pro Intel Core i7, 16GB RAM	VPCN	274.67
	LTD	34.36
64 $Intel^{(R)}$ $Xeon^{(R)}$ CPU@2.50GHz, Nvidia Tesla P100 GPU *2, 32GB RAM	VPCN	31.98
	LTD	**5.58**

Table 5: Comparison of average response time (RT_{avg}).

URLs and by filtering out those irrelevant parts. Therefore, LTD can more accurately classify between the actual attack payloads and those malicious-like benign URL fragments.

Influence of Proposal Number

To explore the influence of different numbers of candidate proposals (i.e., the numbers of payload proposals) on LTD's performance, we test the PLN's performance by choosing different N proposed regions. The experiment is carried out with the CSCI dataset. In Figure 2, we plot the curves (PR and ROC) of SQLi and XSS prediction results, respectively. For both the SQLi and the XSS detection, the system achieves best or close-to-best performance (For XSS detection the 5-region is slightly better than the 3-region) in PR curve when $\#R = 3$. Extracting more payload proposals may lead to a better recall, however, extracting 4 or more proposal regions would trigger more false positives. In production environment, a low FRP is more important than other metrics. In order to have a relatively good recall with an as-low-as-possible FPR, we choose $\#R = 3$ in the PLN.

Method	Precision	Recall	FPR	FNR
char-level CNN	75.85%	59.37%	7.66%	40.63%
LTD	98.17%	96.45%	0.11%	3.55%

Table 6: Comparison between LTD and char-level CNN.

3.5 Interpretability

Deep learning methods release us from the burden of complicated and error-prone feature engineering. However, deep learning models are so-called black-boxes, which have little interpretability. This black-box property brings great challenges to researchers, developers and especially the engineers who are responsible for interpreting model results usually in a daily or oftener basis. Through a locate-then-detect approach, we can clearly know where the attack payload is and which type the attack payload is. In case that there are false predictions, security experts can easily locate problems and optimize the LTD model.

4 Conclusion

In this paper, we introduce a novel Locate-Then-Detect (LTD) detection system that can precisely detect Web threats in real-time by using attention-based deep neural networks. LTD firstly extracts suspicious segments from long requests, then the proposed regions are further inspected by a classifier. By effectively filtering out irrelevant background strings, LTD can increase accuracy by 22.3% and reduce 82.6% of computational costs. Experiments on real Web traffic show that, LTD outperforms several leading methods and can efficiently detect attacks with an average response time of 5ms and thus be well qualified for real-time applications. Despite of many advantages of LTD, it can only deal with SQLi and XSS attacks currently. The ongoing LTD work covers detection of more Web hacks such as File Inclusion and Code Execution.

References

[Bau *et al.*, 2010] Jason Bau, Elie Bursztein, Divij Gupta, and John Mitchell. State of the art: Automated black-box web application vulnerability testing. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 332–345. IEEE, 2010.

[Chollet, 2016] François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint*, 2016.

[Collobert *et al.*, 2011] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[Corona *et al.*, 2009] Igino Corona, Davide Ariu, and Giorgio Giacinto. Hmm-web: A framework for the detection of attacks against web applications. In *Communications, 2009. ICC'09. IEEE International Conference on*, pages 1–6. IEEE, 2009.

[Galbreath, 2012] Nick Galbreath. Libinjection. *Blackhat*, 2012.

[Girshick, 2015] Ross Girshick. Fast r-cnn. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, pages 1440–1448. IEEE, 2015.

[Gupta *et al.*, 2014] Mukesh Kumar Gupta, MC Govil, and Girdhari Singh. Static analysis approaches to detect sql injection and cross site scripting vulnerabilities in web applications: A survey. In *Recent Advances and Innovations in Engineering (ICRAIE), 2014*, pages 1–5. IEEE, 2014.

[Johari and Sharma, 2012] Rahul Johari and Pankaj Sharma. A survey on web application vulnerabilities (sqlia, xss) exploitation and security engine for sql injection. In *Communication Systems and Network Technologies (CSNT), 2012 International Conference on*, pages 453–458. IEEE, 2012.

[Kar *et al.*, 2016] Debabrata Kar, Khushboo Agarwal, Ajit Kumar Sahoo, and Suvasini Panigrahi. Detection of sql injection attacks using hidden markov model. In *Engineering and Technology (ICETECH), 2016 IEEE International Conference on*, pages 1–6. IEEE, 2016.

[Kim *et al.*, 2016] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *AAAI*, pages 2741–2749, 2016.

[Kim, 2014] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[Kindy and Pathan, 2011] Diallo Abdoulaye Kindy and Al-Sakib Khan Pathan. A survey on sql injection: Vulnerabilities, attacks, and prevention techniques. In *Consumer Electronics (ISCE), 2011 IEEE 15th International Symposium on*, pages 468–471. IEEE, 2011.

[Kingma and Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[LeCun *et al.*, 2015] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[Li *et al.*, 2017] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Light-head r-cnn: In defense of two-stage object detector. *arXiv preprint arXiv:1711.07264*, 2017.

[Nguyen *et al.*, 2011] Hai Thanh Nguyen, Carmen Torrano-Gimenez, Gonzalo Alvarez, Slobodan Petrović, and Katrin Franke. Application of the generic feature selection measure in detection of web attacks. In *Computational Intelligence in Security for Information Systems*, pages 25–32. Springer, 2011.

[Pinzón *et al.*, 2010] Cristian Pinzón, Juan F De Paz, Javier Bajo, Álvaro Herrero, and Emilio Corchado. Aiida-sql: An adaptive intelligent intrusion detector agent for detecting sql injection attacks. In *Hybrid Intelligent Systems (HIS), 2010 10th International Conference on*, pages 73–78. IEEE, 2010.

[Ren *et al.*, 2015] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[Xiao and Cho, 2016] Yijun Xiao and Kyunghyun Cho. Efficient character-level document classification by combining convolution and recurrent layers. *arXiv preprint arXiv:1602.00367*, 2016.