Identification of Cross-Site Scripting attacks using Convolutional Neural Networks

Luka Cvetko
Dept. of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN, USA
lcvetko@nd.edu

ABSTRACT

This paper explores the use of Convolutional Neural Networks (CNN) to aid in detection of Cross-Site Scripting(XSS) attacks in web applications. This serves as a shift from traditional methods employed by static analysis tools. The machine-learning based approach aims to adapt to a rapidly changing environment in web technology and aims to classify code as either malicious or benign. The CNN model demonstrated promising results with an accuracy of 99.01%, recall rate of 99.66% and precision rate of 98.53%. These metrics indicate the model's potential in identifying XSS vulnerabilities. However, the research faces limitations due to a small and biased dataset. Future work includes expanding the dataset for better generalization and exploration of hybrid models that can integrate machine learning with traditional static analysis methods.

I. INTRODUCTION

Today web-based applications have become the main forms of conducting modern businesses and services. With this increase in usage, the reliance on web applications has also made them the target of many cyber-attacks. [6] Many unidentified vulnerabilities in these applications can serve as points of entry for potential attackers. This can in turn lead to potential data breaches and financial losses for the business. Many approaches that attempt to mitigate this type of vulnerability are very labor intensive because they require manual revision by technical experts. These are very time-consuming, resource–heavy and can be outdated fast because of rapid development in web applications. The vulnerabilities discovered range from low level coding errors to more complex flaws in the design architecture. There do exist scanners that can account for this, but they often rely on manual configurations or the search of already discovered vulnerabilities using CWE IDs [5] such as Pythia [3] and the Django framework. [4] In this paper the aim is to tackle a specific type of injection attack called Cross-Site-Scripting(XSS) attacks. These are characterized as one of the most rapidly growing types of attacks and is currently ranked third with the injection category in the OWASP Top 10. [17] Cross-Site Scripting attacks are a type of injection attack, in which scripts of malicious nature are inserted into a trusted website or web-source. [16] This can happen in the following:

1. Malicious data can enter a web application through an untrusted source in a form of a web request and is then stored into the data source content of the web application
2. The data is included in dynamic content that is sent to a web user without being validated for malicious content.

The reason these attacks are important to prevent is to keep the integrity of a website and to safeguard user data from being accessed without proper authentication from malicious parties. In order to address these issues a solution that can adapt to the rapid changes in web applications and streamline the detection process. With these constraints in mind an application that can use machine learning to detect possible code which is vulnerable to XSS attacks is proposed. Machine learning can be used to autonomously analyze data, search for patterns and use this information to make a decision. [7] This capability is pivotal for a security-based approach to detect web applications vulnerabilities by allowing for solutions which can learn and adapt as the application evolves.

The project aims to answer these research questions:

1. How can the application discern malicious code from benign code and classify it?
2. How can machine-learning be implemented to identify Cross-Site Scripting (XSS) attacks [8] and enhance overall security?
3. How can machine learning-based identify Cross-Site Scripting (XSS) detection systems be integrated with existing web application security frameworks?

By attempting to answer these research questions the paper aims to give a solution that can automate the vulnerability detection process and can adapt quickly to changes in web application technology with the end goal of making applications safer and more secure for the end user.

II. RELATED WORK

There are many static analyzers that do not use machine learning in order to detect vulnerabilities. These include the following:

- Bandit static code analysis tool to identify common security issues in Python. It relies on predefined rules and signatures to detect vulnerabilities. [1]
- PyLint – static analysis tool that checks the source code against a large set of coding standards which can help identify potential issues. [2]
- Pythia – Python based tool for analyzing and identifying vulnerabilities by performing static analysis of source code. [3]
- Django framework – Django is a web framework for Python based Web applications. Essentially a “middleman” that can help protect against common vulnerabilities in web applications. [4]

Unlike these traditional scanners, the unique selling point of the proposed solution is the ability to automate the detection process based on machine learning and will be able to detect vulnerabilities no matter if the dependencies issue has been patched. By analyzing various amounts of code a classifier will be created in order to discern between malicious and benign code in order to aid in enhancing security practices.

A. Work Novelty

This project extends well beyond the literature taught in class which covered static and dynamic analyzers.

It attempts to do the following:

1.This method aims to use machine-learning approach to vulnerability detection and not a manual or library-based approach which identifies vulnerabilities in dependencies

2. The goal is to give feedback about the code which can later be used to perform various tests to determine if the code is vulnerable or not

III. METHODOLOGY

A. Convolutional Neural Network (CNN)

The method which will be used to approach this project is to use a Convolutional Neural Network[18] in order to train a model to classify code as either being or malicious. A CNN is a type of deep-learning algorithm which has shown exceptional results in the field of image recognition and classification, video analysis and segmentation of objects and natural language processing. In the secure software engineering context particularly in XSS detection, a CNN can be utilized to analyze and detect patterns in code and find any nuances that may signify a security threat is present. Figure 1. Shows the general structure of a CNN model.

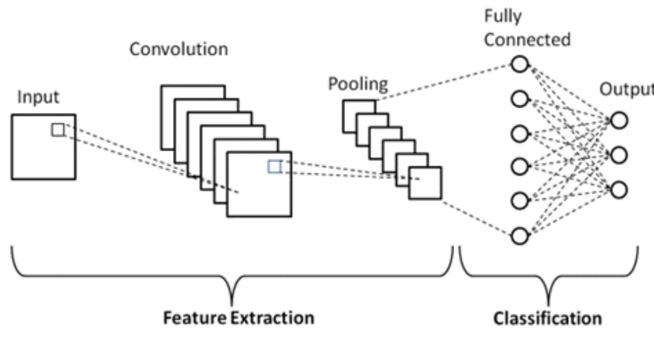


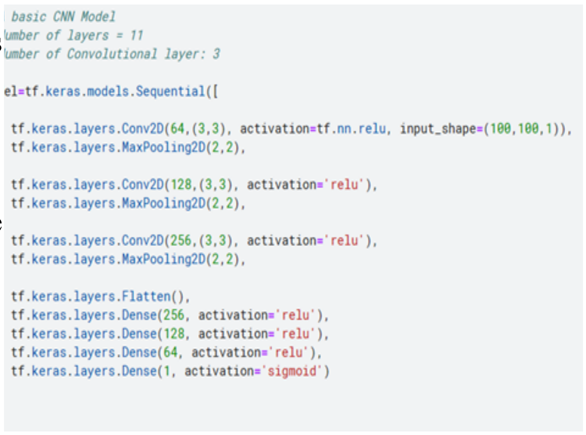
Figure 1. Basic CNN architecture

As can be observed from Figure 1. There are many layers in a CNN model. The first layer is the input layer, where data gets inputted. In this case these will be snippets of code that will be acquired from a dataset. Next, the input data will go through a convolution layer. These layers will apply a number of filters on our input data to create feature maps, in the case of our data these will be patterns that the network will learn within the provided code. Following each convolutional layer an activation function will be used such as ReLU[19] The purpose of this function is to introduce non-linearity to the model, allowing it to learn more complex patterns and prevent the vanishing gradient problem. The vanishing gradient problem occurs in neural networks when gradients of the loss function become increasingly small as they are propagated back through the layers during training, leading to very slow or stagnant learning in the earlier layers. Towards the end, CNNs have one or more fully connected layers. Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their role is to use the features extracted by the convolutional layers to classify the input into various categories. In XSS detection, this could translate to classifying code snippets as either safe or potentially malicious. Finally we reach the output layer which uses SoftMax function to determine if the code is benign or safe. During training a CNN attempts to learn the weights of their filters through a process known as backpropagation.[20]

The training process is the following:

1. The CNN takes in the input data
2. It processes the data through the layers, creating the feature maps and reducing dimensionality
3. Finally the output layer will classify input based on the extracted features.
4. Output is then compared against the actual label(malicious or benign) from our ground truth dataset using a loss function
5. The error of the loss function is back propagated through the entire network, adjusting weights in each epoch to reduce the error rate.

In the specific model proposed for XSS detection the following structure which can be seen in Figure 2. was used.



```
 basic CNN Model
umber of layers = 11
umber of Convolutional layer: 3

el=tf.keras.models.Sequential([

 tf.keras.layers.Conv2D(64,(3,3), activation=tf.nn.relu, input_shape=(100,100,1)),
 tf.keras.layers.MaxPooling2D(2,2),

 tf.keras.layers.Conv2D(128,(3,3), activation='relu'),
 tf.keras.layers.MaxPooling2D(2,2),

 tf.keras.layers.Conv2D(256,(3,3), activation='relu'),
 tf.keras.layers.MaxPooling2D(2,2),

 tf.keras.layers.Flatten(),
 tf.keras.layers.Dense(256, activation='relu'),
 tf.keras.layers.Dense(128, activation='relu'),
 tf.keras.layers.Dense(64, activation='relu'),
 tf.keras.layers.Dense(1, activation='sigmoid')
```

Figure 2. CNN model used in XSS detection

The model consists of 11 different layers, 3 of them being convolutional layers with reLU activation function. The first convolution layer has 64 filters of (3,3) size and accepts the input shape (100,100,1), meaning that our data must go through a preprocessing stage before it can be accepted and discussed in the data section. The second convolutional layer has 128 filters with a (3,3) kernel size and reLu activation function. The third layer has 256 filters with the same kernel size and reLu activation function. After each convolutional layer comes a MaxPooling layer with the pool size (2,2) which reduces the dimensions of output from the convolutional layers. After these a Flatten layer is used to convert all the 2D feature maps into a 1D array which serves as input for the Dense layer. Following this layer there are 3 fully connected Dense layers, with 256,128 and 64 neurons. These are all used for classification based on features extracted from convolutional layers in our case these would be the code patterns. The last layer is again a dense layer which uses a sigmoid function to classify the code as either benign or malicious. The network is then trained on 20 epochs.

B. Dataset

The dataset which was used to accomplish this classification task was acquired from Kaggle.com.[21] The dataset features 3 columns. First one is the index column, second isa column which contains the actual code which will be analyzed. In this column there is a plethora of malicious and benign code, which is then classified with the third column as either malicious with 1 or benign with a 0. As mentioned in the section above we must preprocess this data in order for it to fit the proper shape of our input layer. First all the data is converted to a 2D array of ASCII encoding and only using values up to 128 in ASCII and discarding the rest. Then the data is turned into a 2D array of shape size (100,100) standardizing the input size. After this preprocessing the dataset will be split into training and testing dataset with a 80/20 split where 80% of the data will be used for training and 20% of the data will be used for testing. After these steps are completed the data can now finally be used to train and test the efficiency of the model.

With the data and model in place we can return to our research questions and provide a method for answering them.

C. Research Questions

1. How can the application discern malicious code from benign code and classify it?

   From the method provided a machine learning CNN network is used to classify code from the given dataset as either benign or malicious using binary classification.

2. How can machine-learning be implemented to identify Cross-Site Scripting (XSS) attacks and enhance overall security?

   Determine the efficiency of the model using different model evaluation techniques such as accuracy, precision and recall in order to determine the performance of the model.

3. How can machine learning-based identify Cross-Site Scripting (XSS) detection systems be integrated with existing web application security frameworks?

   If model efficacy and performance demonstrated satisfactory results, discuss how the approach could be implemented into existing web application security frameworks.

IV. RESULTS

After training for 20 epochs there were some promising results by using the CNN model for XSS detection. As can be seen in Figure 3. The confusion matrix shows the number of true and false positives for each category.

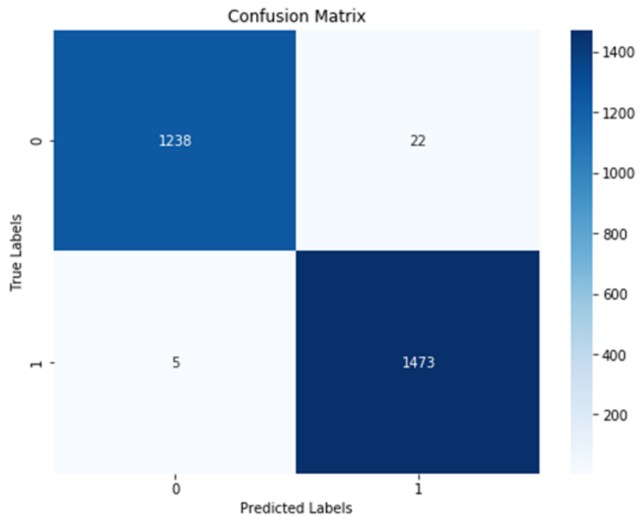


Figure 3. Confusion matrix of predictions

There were 2711 correct predictions, and 27 false predictions. Breaking down the predictions there were 5 incorrect predictions for benign data which the model thought was malicious and 1238 correct predictions, 22 false predictions for malicious data which the model thought was benign and 1473 correct predictions. This gave us an accuracy of 99.01%, a recall rate of 99.66% and precision rate of 98.53%. This means that the model is very effective at classifying the data correctly. The high recall rate proves the model is able to identify true positives and is highly effective at catching actual XSS vulnerabilities. Finally, the high precision rate indicates the model is very effective. This means that if the model predicts a code as positive for XSS attacks there is a high chance it correctly identifies that source as a potential attack.

With these results we can attempt to answer the provided research questions.

1. How can the application discern malicious code from benign code and classify it?

From the results and the provided metrics of accuracy, recall and precision the model can successfully identify and classify code as either malicious or benign with a great deal of accuracy.

2. How can machine-learning be implemented to identify Cross-Site Scripting (XSS) attacks and enhance overall security?

The high recall rate of 99.66% indicates the model can identify true positives and effectively catch actual XSS vulnerabilities. The precision rate of 98.53% shows that when the model predicts a piece of code as positive for an XSS attack , there is a great likelihood the model is correct. High precision is vital to maintain trust in the system and ensure that resources are not wasted on investigating false positives. The combination of high recall and precision indicated that the model can be used as a powerful tool to detect XSS vulnerabilities without overwhelming the system with false positives.

3. How can machine learning-based identify Cross-Site Scripting (XSS) detection systems be integrated with existing web application security frameworks?

The model can be deployed as a complement to existing security tools that use static analysis to find vulnerable sinks and could be used in parallel with those to ensure a source identified is a source of threat. The proposed application is to ensure the model is able to receive input from and send output to existing system components. For example, the model might analyze incoming traffic and web requests and alert existing tools when a potential attack is detected, then a static analyzer can check the source and a decision can be determined.

V. DISCUSSION

The results from the model are very promising however there are some implications that can hurt the model when exposed to actual real-life practices and applications. One of the drawbacks of the model is the reliance on its dataset, only relying on one dataset which has been used to learn the patterns of only that specific dataset. When presenting information, which is not demonstrated as in the provided dataset such as just plain text, the model automatically characterizes it as a malicious source because it doesn't see any HTML formatting such as brackets, head, body which it was learned on to detect potential risks. Next, issue is that the dataset itself was small it had around 14,000 samples, which were already filtered out, a dataset with more than 100,000 code samples should be used to train the model and see how the model would perform on data that is not curated and filtered to match the training data precisely. This dataset would ensure a plethora of potential XSS attack scenarios and not hinder the models ability to generalize to new unseen data. The dataset also has a preference for benign code, a new dataset should be developed that does not favor any type of and has an even 50/50 split for number of benign and malicious code samples. For future projects one should look to use a Recurrent Neural Network(RNN) which is used for sequence analysis. This network is designed to compare the syntax of words and can be used to predict if a code sample is malicious or not dependent on the entire syntax of the sample not just detected patterns, this can in turn lead to further improvement in detection results and potentially increase the range of detectable threats since the network would not be learning patterns but would take the entire syntax into account before making a decision. This approach can be done by using Transformer architecture which is based on attention mechanisms that can predict words based on different parts of an input sequence. One could use this approach to better understand the context and relationship within the data which could make for a highly effective detector of complex attack patterns, these could also be very scalable in nature and flexible to many inputs since their ability to learn from unstructured data. The optimal solution would be to incorporate a hybrid model that can use both static analysis and the Transformer to detect and prevent XSS attacks, however to accomplish this first a larger dataset needs to be collected and higher computational power is needed.

VI. CONCLUSION

This research project attempted to explore the feasibility of using a machine-learning approach in order to detect potential Cross-Site Scripting (XSS) attacks within web-applications. The approach taken was using the fact that Convolutional Neural Network (CNN) models could offer a significant advantage over traditional static analysis tools. The results show promise. The CNN model achieved an impressive accuracy of 99.01%, a recall rate of 99.66% and a precision rate of 98.53%. These metrics indicate the model's robust capability to accurately classify code samples as either malicious or benign. This highlights its potential as a powerful tool that can be used with existing methods. Specifically, its high recall rate signifies that the model can effectively detect actual XSS vulnerabilities while its high precision rate assures that its predictions are accurate and reliable. One major limitation faced was the small and limited dataset. This raised concerns about the model ability to generalize on new and unseen data. Another area of concern was the dataset bias towards benign code which could skew the model's learning process. Despite these, one cannot disregard the potential a machine-learning approach has in detecting potential XSS attacks. Future research should aim to solve the current studies limitations by employing a larger and more diverse dataset, which ensures a balanced representation of malicious and benign code samples. It is also recommended to explore diverse hybrid models which would be able to utilize the strengths of static analysis tools and Transformers architecture to open new research opportunities for more robust and secure methods. In conclusion this study provides an innovative method to existing methods of detection and attempts to provide an adaptive approach. By integrating machine-learning into web-application security a promising future development can occur where identification and mitigation of XSS attack become more efficient and reliable. The entire project can be accessed on the following GitHub link:Windtwist/SSE-final: Repo for final of SSE project (github.com)

VII. REFERENCES

[1] Welcome to Bandit — Bandit documentation. (n.d.). https://bandit.readthedocs.io/en/latest/

[2] Pylint 2.17.7 documentation. (n.d.). https://pylint.readthedocs.io/en/stable/

[3] About PYTHIA - PYTHIA 8.3. (n.d.). https://pythia.org/about/

[4] Django. (n.d.). Django Project.

https://docs.djangoproject.com/en/4.2/topics/security/

[5] CWE - Common Weakness Enumeration. (n.d.). https://cwe.mitre.org/

[6] Liu X, Ahmad SF, Anser MK, Ke J, Irshad M, Ul-Haq J, Abbas S. Cyber

security threats: A never-ending challenge for e-commerce. Front Psychol.

2022 Oct 19;13:927398. doi: 10.3389/fpsyg.2022.927398. PMID: 36337532;

PMCID: PMC9629147.

[7] Machine Learning: What it is and why it matters. (n.d.). SAS.

https://www.sas.com/en_us/insights/analytics/machine-learning.html

[8] Kaur, Jasleen & Garg, Urvashi & Bathla, Gourav. (2023). Detection of

cross-site scripting (XSS) attacks using machine learning techniques: a

review. Artificial Intelligence Review. 56. 1-45. 10.1007/s10462-023-10433-3.

[9] OWASP. OWASP Broken Web Applications Project.

https://www.owasp.org/index.php/OWASP_Broken_Web_

Applications_Project

[10] What are Naive Bayes classifiers? | IBM. (n.d.). What Are Naive Bayes

Classifiers? | IBM. https://www.ibm.com/topics/naive-bayes

[11] Understanding LSTM Networks -- colah's blog. (n.d.). Understanding

LSTM Networks -- Colah's Blog. https://colah.github.io/posts/2015-08-

Understanding-LSTMs/

[12] https://www.callrail.com/blog/what-is-crawling-and-indexing#crawling.

(n.d.). https://www.callrail.com/blog/what-is-craw ling-and-indexing#crawling

[13] B. Peng, X. Xiao and J. Wang, "Cross-Site Scripting Attack Detection Method Based on Transformer," 2022 IEEE 8th International Conference on Computer and Communications (ICCC), Chengdu, China, 2022, pp. 1651-1655, doi: 10.1109/ICCC56324.2022.10065892.

[14] S. Saleem, M. Sheeraz, M. Hanif and U. Farooq, "Web Server Attack Detection using Machine Learning," 2020 International Conference on Cyber Warfare and Security (ICCWS), Islamabad, Pakistan, 2020, pp. 1-7, doi: 10.1109/ICCWS48432.2020.9292393.

[15] A. Sivasangari, J. Jyotsna and K. Pravalika, "SQL Injection Attack Detection using Machine Learning Algorithm," 2021 5th International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 2021, pp. 1166-1169, doi: 10.1109/ICOEI51242.2021.9452914.

[16] Cross Site Scripting (XSS) | OWASP Foundation. owasp.org/www-community/attacks/xss.

[17] OWASP Top Ten | OWASP Foundation. owasp.org/www-project-top-ten.

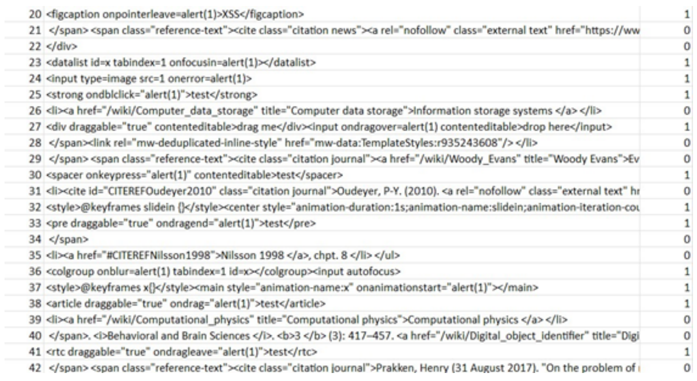
[18] Alzubaidi, L., Zhang, J., Humaidi, A.J. et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. J Big Data 8, 53 (2021). https://doi.org/10.1186/s40537-021-00444-8

[19] Brownlee, Jason. "A Gentle Introduction to the Rectified Linear Unit (ReLU)." MachineLearningMastery.com, 20 Aug. 2020, machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks.

[20] Kostadinov, Simeon. "Understanding Backpropagation Algorithm - Towards Data Science." Medium, 11 Dec. 2021, towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd.

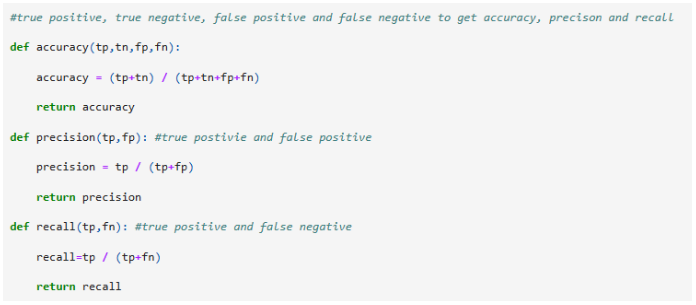
[21] "Cross Site Scripting XSS Dataset for Deep Learning." Kaggle, 17 Mar. 2020, www.kaggle.com/datasets/syedsaqlainhussain/cross-site-scripting-xss-dataset-for-deep-learning/data.

APPENDIX



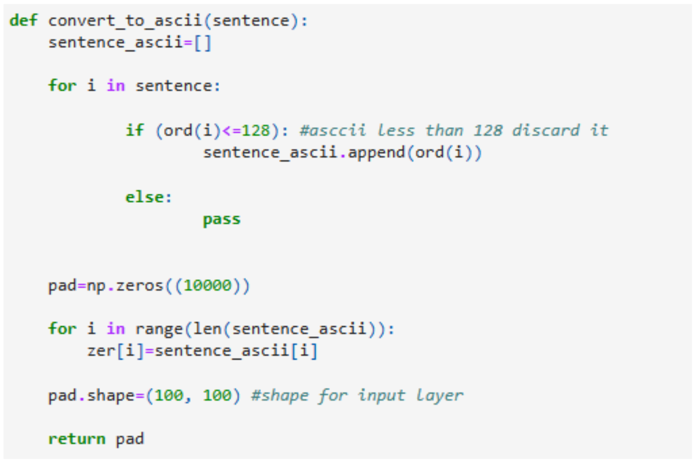
20	<figcaption onpointerleave=alert(1)>XSS</figcaption>	1
21	</span> <span class="reference-text"><cite class="citation news"><a rel="nofollow" class="external text" href="https://ww	0
22	</div>	0
23	<datalist id=x tabindex=1 onfocusin=alert(1)></datalist>	1
24	<input type=image src=1 onerror=alert(1)>	1
25	<strong ondblclick="alert(1)">test</strong>	1
26	<li><a href="/wiki/Computer_data_storage" title="Computer data storage">Information storage systems </a> </li>	0
27	<div draggable="true" contenteditable>drag me</div><input ondragover=alert(1) contenteditable>drop here</input>	1
28	</span><link rel="mw-deduplicated-inline-style" href="mw-data:TemplateStyles:r935243608"/> </li>	0
29	</span> <span class="reference-text"><cite class="citation journal"><a href="/wiki/Woody_Evans" title="Woody Evans">Ev	0
30	<spacer onkeypress="alert(1)" contenteditable>test</spacer>	1
31	<li><cite id="CITEREFOudeyer2010" class="citation journal">Oudeyer, P-Y. (2010). <a rel="nofollow" class="external text" hr	0
32	<style>@keyframes slidein {}</style><center style="animation-duration:1s;animation-name:slidein;animation-iteration-cou	1
33	<pre draggable="true" ondragend="alert(1)">test</pre>	1
34	</span>	0
35	<li><a href="#CITEREFNilsson1998">Nilsson 1998 </a>, chpt. 8 </li> </ul>	0
36	<colgroup onblur=alert(1) tabindex=1 id=x></colgroup><input autofocus>	1
37	<style>@keyframes x{}</style><main style="animation-name:x" onanimationstart="alert(1)"></main>	1
38	<article draggable="true" ondrag="alert(1)">test</article>	1
39	<li><a href="/wiki/Computational_physics" title="Computational physics">Computational physics </a> </li>	0
40	</span>. <i>Behavioral and Brain Sciences </i>. <b>3 </b> (3): 417–457. <a href="/wiki/Digital_object_identifier" title="Digi	0
41	<rtc draggable="true" ondragleave="alert(1)">test</rtc>	1
42	</span> <span class="reference-text"><cite class="citation journal">Prakken, Henry (31 August 2017). "On the problem of	0

Code snippet from used dataset.



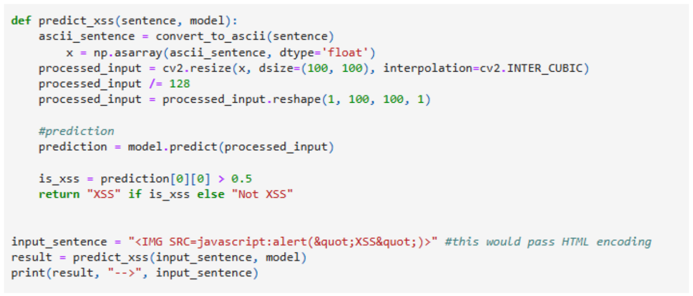
```
#true positive, true negative, false positive and false negative to get accuracy, precison and recall
def accuracy(tp,tn,fp,fn):
    accuracy = (tp+tn) / (tp+tn+fp+fn)
    return accuracy
def precision(tp,fp): #true postivie and false positive
    precision = tp / (tp+fp)
    return precision
def recall(tp,fn): #true positive and false negative
    recall=tp / (tp+fn)
    return recall
```

Accuracy,precision, recall functions



```
def convert_to_ascii(sentence):
    sentence_ascii=[]

    for i in sentence:

            if (ord(i)<=128): #asccii less than 128 discard it
                sentence_ascii.append(ord(i))

            else:
                pass


    pad=np.zeros((10000))

    for i in range(len(sentence_ascii)):
        zer[i]=sentence_ascii[i]

    pad.shape=(100, 100) #shape for input layer

    return pad
```

Convert to ascii function



```python
def predict_xss(sentence, model):
    ascii_sentence = convert_to_ascii(sentence)
        x = np.asarray(ascii_sentence, dtype='float')
    processed_input = cv2.resize(x, dsize=(100, 100), interpolation=cv2.INTER_CUBIC)
    processed_input /= 128
    processed_input = processed_input.reshape(1, 100, 100, 1)

    #prediction
    prediction = model.predict(processed_input)

    is_xss = prediction[0][0] > 0.5
    return "XSS" if is_xss else "Not XSS"


input_sentence = "<IMG SRC=javascript:alert(&quot;XSS&quot;)>" #this would pass HTML encoding
result = predict_xss(input_sentence, model)
print(result, "-->", input_sentence)
```

Predict XSS function - takes in user input to recognize input as either benign or malicious.