VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

INFORMATION SECURITY MANAGEMENT

Final Report

Project Title:

Detection of Cross-Site Scripting (XSS) attacks with CNN Technology

Team Members:

20BDS0003 - Sai Ruthvik Athota
20BCE2443 - Sarthak Khandelwal

Submitted to

Prof. Rajarajan G

Slot: F1

Demo video Link-

https://drive.google.com/file/d/1dJ7dM0kFxn1xmuiOqSyy-40D9JK_12dL/view?usp=share_link

Abstract

In the contemporary world, even after advancements in technology and the boost in the number of ethical hackers, there have been a lot of cases of websites being hacked using Cross-Site Scripting XSS attacks. The severity of these hacks ranges from small pranks to bringing huge losses to both the company and the end users. Cross-site scripting attacks, like any other network security threat attack, use a browser function which it uses to segregate different websites and allow websites to load up scripts in the browser if they have the same origin as the website. The problem here is that the browsers do not look into what kind of data it is loading. This leads the hacker to circumvent the script loading part of the process and load up his malicious script to the browser masking it to the same origin as the original website.

Introduction

One of the common platforms for service updates and presenting data and information over the Internet are web applications. Security flaws led to numerous kinds of attacks on online applications as a result. Cross-site scripting, generally known as XSS, is one of them. One of the most common threats identified by web security software is XSS. Malicious scripts are injected into online applications to cause XSS, which can result in serious violations for the site or the user. Therefore, our project is related to detecting Cross Site Scripting with CNN.

Objectives

1. To train a CNN model to detect XSS Attacks
2. To test the trained model on new and latest XSS attacks and check for validity of the model
3. To design a simple UI and frontend for user to easily enter URL to check for XSS attacks
4. To integrate the model with the frontend in order to detect if the entered URL contains an xss attack or not

Tools used

- CNN model
- Sklearn
- Tensorflow
- Keras
- Flask
- Kaggle

Literature review

Author and Date	Title	Methodology	Conclusion
Zarul Fitri Zaaba Abdalla Wasef Marashdih Universiti Sains Malaysia October 2016	Cross Site Scripting: Detection Approaches in Web Application	The following approaches were used to analyze Cross Site XSS attacks:- Static (High positive rate) Dynamic (Result not accurate) Hybrid (Results not accurate) Genetic (Can handle only reflected XSS)	All the methods aren't good to come up with a foolproof solution to cross-site scripting attacks. A new approach is needed, something which could increase its effectiveness and accuracy as time passes by, while not aging with time.
Keiron Teilo O'Shea Ryan Nash Aberystwyth University November 2015	An Introduction to Convolutional Neural Networks	Talks about the implementation of CNN, and where it's better than the classical ANN algorithms.	CNN is better for unstructured data like images, videos, etc.
Sepp Hochreiter	Long Short-term	Presents the Long	Long Short-Term

Jürgen Schmidhuber Johannes Kepler University Linz December 1997	Memory	Short Term Memory (LSTM) which is implemented by using a gradient-based algorithm to solve existing problems of RNN (Recurrent Neural Network) Algorithms.	Memory is, if not the best, one of the best algorithms to work with time series data.

Existing systems/models

There are several existing models for the detection of XSS (Cross-Site Scripting) attacks, which include:

- Signature-based models: These models use predefined signatures or patterns to detect known XSS attacks. The drawback of this approach is that it cannot detect new or unknown XSS attacks.

- Anomaly-based models: These models use statistical and machine learning techniques to detect unusual or anomalous behavior in web traffic. This approach can detect new or unknown XSS attacks, but it may also generate false positives.

- Hybrid models: These models combine signature-based and anomaly-based approaches to improve the accuracy of detection.

- Rule-based models: These models use predefined rules or heuristics to identify potential XSS attack vectors. This approach can be effective, but it requires manual rule creation and maintenance

Our Proposed Model

Our proposed model is a Machine Learning based model -

Machine learning models: These models use various machine learning algorithms such as decision trees, neural networks, and support vector machines to detect XSS attacks. They can learn from historical data and adapt to new and evolving attack patterns.

Our model uses a type of neural network called Convolutional Neural Network to identify features in links/text and classify them to predict whether it is an XSS attack or not.

Outcomes of the proposed model

The outcomes of the CNN model for detecting XSS attacks are:

1. High accuracy: The model has a high accuracy (99.03%) in detecting XSS attacks, reducing the risk of false positives or false negatives.

2. Robustness: The model can be robust to variations in the types of XSS attacks, and can detect previously unseen or unknown attacks.

3. Speed: The model can be efficient and fast at detecting XSS attacks, making it suitable for real-time detection in web applications.

4. Overall, using a CNN model for detecting XSS attacks can improve the security and reliability of web applications, reducing the risk of malicious attacks and protecting sensitive user data.

Merits of the proposed model

Using a CNN (Convolutional Neural Network) model to detect XSS (Cross-Site Scripting) threats is better than using other models in several ways:

- Automatic feature extraction: CNN models can automatically extract relevant features from the data they are given. This makes them very good at finding complex patterns and relationships between the features they are given. In the case of XSS detection, the model can learn to recognise the signs of an XSS attack, such as malicious code and patterns on web pages.

- Scalability: CNN models can be easily scaled up to process large amounts of data, which is important for finding XSS attacks in real time. Because the model is scalable, it can also be used to find XSS attacks in different web applications at the same time.

- Generalizability: CNN models are very generalizable and can be learned on a wide range of data sources. This means that the model can be taught on a wide range of web pages and is very good at finding XSS attacks on web pages it has never seen before.

- Robustness: CNN models are very resistant to noise and can handle changes in the input data. This makes them good at finding XSS attacks that are complicated and change over time. The model can adapt to new and emerging types of attacks and can continue to find XSS attacks even when attackers try to avoid being found by using new methods.

- Explainability: CNN models can show how they made a certain choice by pointing out the most important parts of the data they were given. This lets security pros understand how the model is detecting XSS attacks and make smart choices about how to improve the model's ability to detect attacks.

Demerits of the proposed model

There are some good things about a model that uses CNN to find XSS threats, but there are also a few drawbacks to consider regarding the following:

- Need for a lot of training data: For CNN models to learn well, they need a lot of data that has been labelled. For XSS identification, this means a big collection of web pages with both good and bad content. Collecting and labelling this kind of information can take a lot of time and money.
- Heavy on computation: CNN models are heavy on computation, especially when they have to deal with a lot of data. This can make it hard to train and run the model without a lot of time and money.
- Vulnerable to adversarial attacks: CNN models are susceptible to being affected by adversarial attacks, in which attackers change the data that goes into the model on purpose to avoid being caught by it. For XSS detection, attackers can change the text of a web page to keep the model from finding them.
- CNN models can help you understand the most important parts of the data you give them, but it can be hard to figure out how they work, especially if you are not an expert. This can make it hard to find mistakes or biases in the model and fix them.
- Problems with unstructured data: CNN models work best with organised data like images and text, but they might have trouble with unstructured data or data with a lot of variation.

Implementation

Code

1. app.py

```
from flask import Flask, redirect, url_for, request, render_template

import numpy as np

from tensorflow.keras.models import load_model

import cv2



# Define a flask app

app = Flask(__name__)



def convert_to_ascii(sentence):

    sentence_ascii=[]



    for i in sentence:



        """Some characters have values very big e.d 8221 and some
are chinese letters

        I am removing letters having values greater than 8222 and
for rest greater

        than 128 and smaller than 8222 assigning them values so they
can easily be normalized"""
```

```
    if(ord(i)<8222):      # ” has ASCII of 8221

        if(ord(i)==8217): # ’  :  8217
            sentence_ascii.append(134)



        elif(ord(i)==8221): # ”  :  8221
            sentence_ascii.append(129)


        elif(ord(i)==8220): # “  :  8220
            sentence_ascii.append(130)



        elif(ord(i)==8216): # ‘  :  8216
            sentence_ascii.append(131)


        elif(ord(i)==8217): # ’  :  8217
            sentence_ascii.append(132)


        elif(ord(i)==8211): # –  :  8211
            sentence_ascii.append(133)
```

```
            #If values less than 128 store them else discard them
            elif(ord(i)<=128):
                    sentence_ascii.append(ord(i))

            else:
                    pass


    zer=np.zeros((10000))

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

    zer.shape=(100, 100)
    return zer

def prepro(sentence):

    model = load_model('model.h5')

    image=convert_to_ascii(sentence)
```

```
    x=np.asarray(image,dtype='float')

    image =  cv2.resize(x, dsize=(100,100),
interpolation=cv2.INTER_CUBIC)

    image/=128

    image=image.reshape(1,100,100,1)



    result = model.predict(image);

    if(result>=0.5):

        ans = 1



    else:

        ans = 0



    return ans




@app.route('/', methods=['GET'])

def index():

    # Main page

    return render_template('index.html')



@app.route('/predict', methods=['GET','POST'])

def upload():

    errors = []
```

```
    results = {}

    if request.method == "POST":

        # get url that the user has entered

        try:

            url = request.form['url']

            result = prepro(url)

            print(result);

            if result==0:

                return render_template('notxss.html')

            else:

                return render_template('xss.html')



        except:

            errors.append(

                "Unable to get URL. Please make sure it's valid and
try again."

            )




if __name__ == '__main__':

    app.run(debug=True)
```

2. Model_training.py

```
import pandas as pd

import cv2

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras.models import load_model

df=pd.read_csv("XSS_dataset.csv")

df.head()

df=df[df.columns[1:]]

df.head()

sentences=df['Sentence'].values

sentences[0]

len(sentences)

def convert_to_ascii(sentence):

    sentence_ascii=[]



    for i in sentence:




        """Some characters have values very big e.d 8221 and some
are chinese letters

        I am removing letters having values greater than 8222 and
for rest greater
```

```
        than 128 and smaller than 8222 assigning them values so they
can easily be normalized"""


        if(ord(i)<8222):      # ” has ASCII of 8221


            if(ord(i)==8217): # ’  :  8217

                sentence_ascii.append(134)



            elif(ord(i)==8221): # ”  :  8221

                sentence_ascii.append(129)


            elif(ord(i)==8220): # “  :  8220

                sentence_ascii.append(130)



            elif(ord(i)==8216): # ‘  :  8216

                sentence_ascii.append(131)


            elif(ord(i)==8217): # ’  :  8217

                sentence_ascii.append(132)


            elif(ord(i)==8211): # –  :  8211

                sentence_ascii.append(133)
```

```
            #If values less than 128 store them else discard them
            elif(ord(i)<=128):
                    sentence_ascii.append(ord(i))

            else:
                    pass


    zer=np.zeros((10000))

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

    zer.shape=(100, 100)
    return zer
#Applying this function to all our sentences
arr=np.zeros((len(sentences),100,100))

for i in range(len(sentences)):

    image=convert_to_ascii(sentences[i])
```

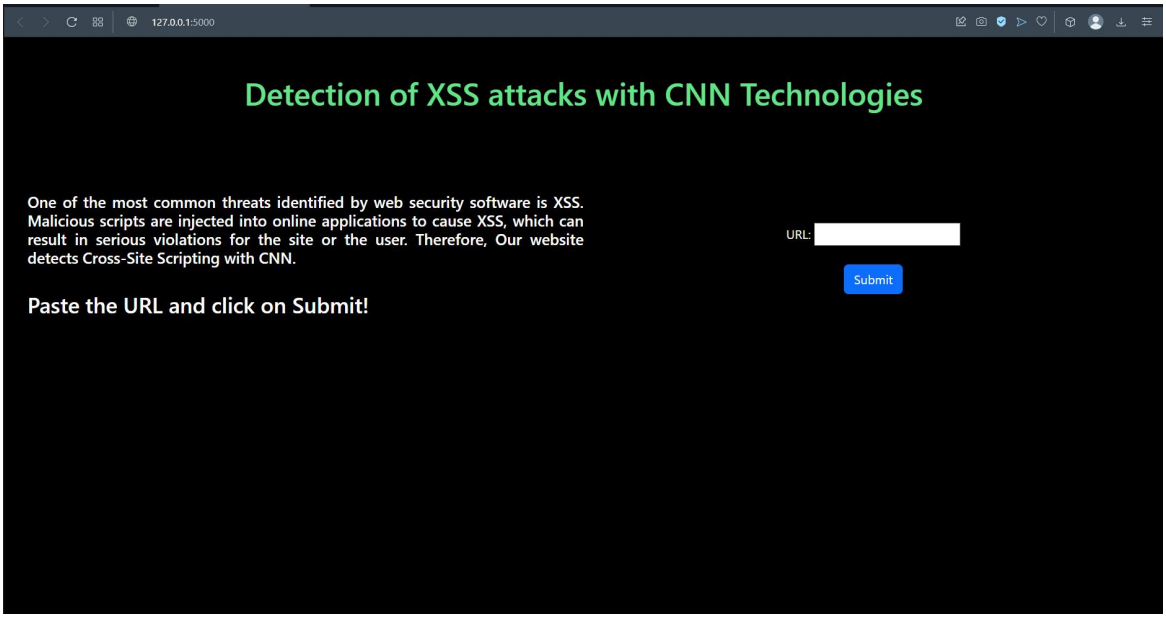
```
    x=np.asarray(image,dtype='float')

    image =  cv2.resize(x, dsize=(100,100),
interpolation=cv2.INTER_CUBIC)

    image/=128

    arr[i]=image

x=arr.reshape(arr.shape[0],100,100,1)

y=df['Label'].values

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.33,ra
ndom_state=42)

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import
Conv2D,Dense,Activation,MaxPooling2D,Flatten,Dropout,MaxPool2D,Batch
Normalization

#Creating the model

model=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')


])
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.summary()
#Training the model
batch_size = 128
num_epoch = 10


model_log = model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=num_epoch,
          verbose=1,
```

```
          validation_data=( x_test, y_test))

# model = load_model('model.h5')


def prepro(sentence):


    model = load_model('model.h5')


    image=convert_to_ascii(sentence)


    x=np.asarray(image,dtype='float')

    image =  cv2.resize(x, dsize=(100,100),
interpolation=cv2.INTER_CUBIC)

    image/=128

    image=image.reshape(1,100,100,1)


    result = model.predict(image);

    if(result>=0.5):

        ans = "XSS ATTACK"


    else:

        ans = "NOT AN XSS ATTACK"


    return ans
```

- **Implementation screenshots**

Main page

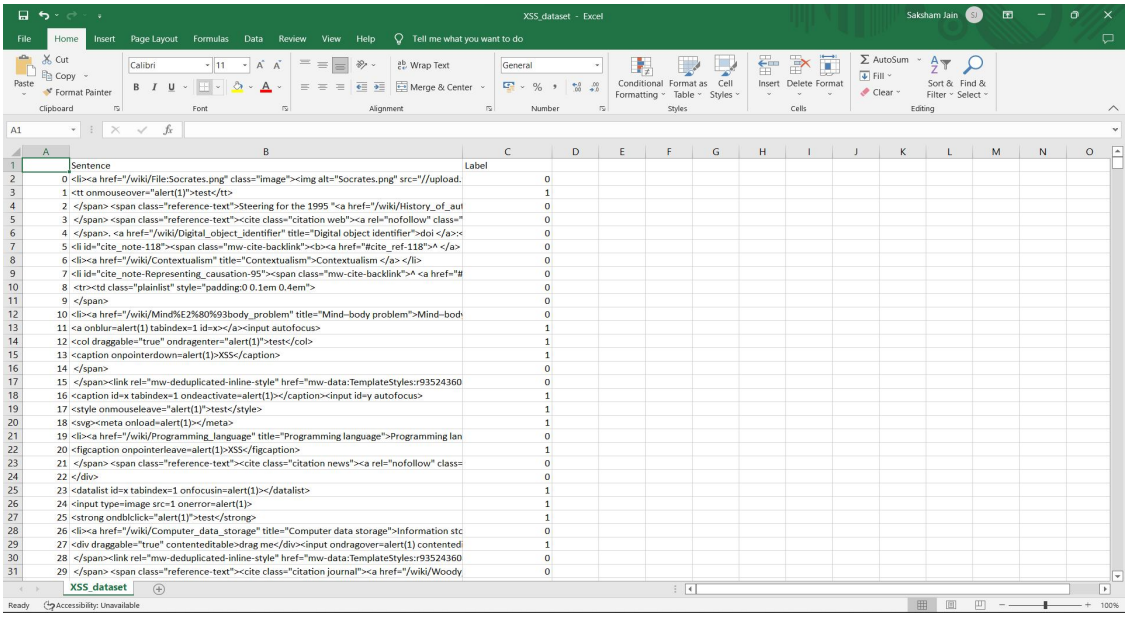


XSS attack detected in URL

XSS attack not detected

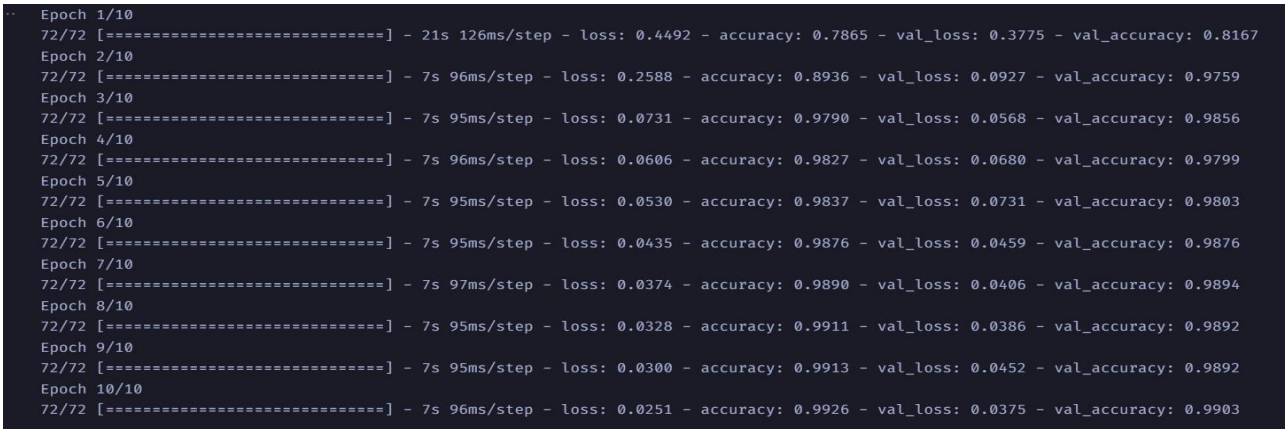
Experimental Evaluation

- Dataset used: [Cross site scripting attack detection using CNN | Kaggle]



Results:

Accuracy	99.03%



Improvements from existing systems

The technologies already used for XSS detection are not reliable, inaccurate, or performance intensive for example Static Approach shows a high rate of false positives whereas Dynamic and Hybrid Methods give inaccurate results and Genetic Approaches can handle only reflected XSS. Our solution uses the CNN model which trains a classification to differentiate between host samples and XSS. CNN is used here instead of ANN as CNN is better at processing unstructured data. This significantly increases detection accuracy, does not impact the code structure of the webpage, and gives a good performance.

Conclusion

Web applications are widely used, putting them at significant risk of assaults from numerous sources, varying degrees of severity and sophistication. Cross-Site Scripting (XSS) is a type of code injection attack that utilizes web browsers on the client side. The attacker's primary tactic is to use the victim's browser to run the malicious scripts that have been injected into a simple web application. Other security attacks like the distribution of malware, credential theft, credential phishing, social network worms, and website defacement can all be caused by XSS vulnerabilities. Therefore, there is a need for the detection of XSS attacks. Our project is able to detect this with the help of a CNN model with an accuracy of 99.03%

References

1. Introduction to Cross-Site Scripting (XSS) Attacks: https://owasp.org/www-community/attacks/xss/

2. Overview of Convolutional Neural Networks: https://towardsdatascience.com/a-beginners-guide-to-convolutional-neural-networks-cnns-14649dbddce8

3. Using CNNs for XSS Detection: https://ieeexplore.ieee.org/document/8587451

4. Advantages of CNNs for Malware Detection: https://ieeexplore.ieee.org/abstract/document/8406972

5. Adversarial Attacks on CNNs: https://arxiv.org/abs/1712.07107

6. The Importance of Data Quality in Machine Learning: https://towardsdatascience.com/the-importance-of-data-quality-in-machine-learning-5242e00d1082

7. Understanding the Limitations of Machine Learning: https://towardsdatascience.com/understanding-the-limitations-of-machine-learning-506c8d30706a

8. Practical Tips for Interpreting CNNs: https://towardsdatascience.com/practical-tips-for-interpreting-cnns-8393ce8fbeae