Machine Learning For WAFs

Research Paper by - Shivang Kumar

Abstract

This paper aims at creating a model for the modern Web Application Firewalls (WAF) with a very dynamic approch using machine learning. The code used for attack detections does not uses any regex or any pattern matchings thus making it more effective. Model learns from the datasets which is based from past observations with a very high prediction acurracy (over 98%) Model would be trained first with payloads for all very know attacks (xss, sqli,lfi,ssti,shell commands, etc) along with all codes from preprcessing to training and testing. The methodology show in this paper can be changed as per requirements.

For the Readers

Author assumes that readers should at least know basics for python coding. Have a firm grip over web attacks. Have some knowledge regarding machine learning.

Contents

Introduction

Machine Learning is of of the most rapidly growing thing and it's integration with cyber security is just amazing. **WAFs** are the most crucial part for the modern web apps. This paper uses **sklearn** to train a **supervised** backend model for a moder dummy WAF with great precision.

Making datasets

Traning a model is not a very big deal as with view lines of python code any one can do stuffs which would feel like magic for a non coders. But getting a dataset for any training a model is the toughest job. For most of the time if projects are old ..then datasets are easily availabe from past researches. But for this paper we will make our own !.

Getting Payloads

Github is the most lovely thing ever happend for hacker as we can easily get our raw payloads we just need to preprocess it . So we will get all out payloads for various attack types from there and valid quries which waf should allow users to pass through. Make sure that valid quries showed be more than payloads otherwise waf will block everything and also not very big that every payloads get undected !

- Links:

Valids will be prcessed from

ECML

Payloads are collected from

Payload_set_1 Payload_set_2

- Note: Fully processed ready to be trained data set is availabe here

Dataset

Processing Collected Data

Now we have our payloads and valid lists collected let's process it !

Processing Valids

- Code used

```
#Changing json to a csv
import pandas as pd
df_valid_tmp = pd.read_json ('ecml.json')
df_valid_tmp.to_csv ('valid.csv', index = None)
df_valid_tmp.rename(columns = {'pattern':'Payloads', 'type':'Types'}, inplace = True)
df_valid = df_valid_tmp[df_valid_tmp.Types == 'valid']
```

Processing Payloads

- Code used:

```
 import pandas as pd

  #Get all files
  xss = 'xss.txt'
  sqli = 'sql.txt'
  nosql = 'nosql.txt'
  lfi = 'lfi.txt'
  shell = 'shell.txt'
  crlf = 'crlf.txt'
  ssti = 'ssti.txt'
  ssi = 'ssi.txt'


  files_array = [xss, sqli, nosql, lfi, shell, ssti, crlf, ssi]

  #This function will combine all wordlists into a single csv
  def processdata_and_store(files_array):
    #Create empty data frame
    empty_df = pd.DataFrame(columns=['Payloads', 'Types'])
    for files in range(len(files_array)):
        unique_set = set()
        with open(files_array[files], 'r') as f:
          for add in f.read().strip().splitlines():
            unique_set.add(add)

    # This removes empty values
        unique_list = list(unique_set)
        while("" in unique_list) :
          unique_list.remove("")
        for add_payload in unique_list:
          empty_df = empty_df.append({'Payloads': add_payload, 'Types' : files_array[files].split(".")[0]}, ignore_index=True)

    empty_df.to_csv('payload.csv', encoding='utf-8')

  processdata_and_store(files_array)
```

The above function **processdata_and_store** when called will store all raw payload in an organized csv manner. Now time to merge it with **valid.csv**

- Code used:

```
#Prepare payloads's dataframe
df = pd.read_csv('dataset.csv').drop(columns='Unnamed: 0')

#Join both datasets as one
clean_df_tmp = pd.concat([df, df_valid])

#Removing duplicates
clean_df = clean_df_tmp.drop_duplicates(subset=['Payloads', 'Types'])
```

Let's check out how our csv looks

```
clean_df
```

```
 Payloads    Types
0   <select onbeforecut="alert(1)" contenteditable...   xss
1   <th oncontextmenu="alert(1)">test</th>  xss
2   </script> %>/   xss
3   <textarea autofocus onfocusin=alert(1)>test</t...   xss
4   <body onBeforeUnload body onBeforeUnload="java...   xss
... ... ...
35000   beinrm-tet, apw-hc, osnent-wmt4s, e-nsrt;q=0.8...   valid
35001   /f041ihx@/kp/cdxwdk5r96hzmza8r/f2s.shtml    valid
35002   hco-=9858907&xseirwhte=iexecc&0neuwe8=mmi&xgf....   valid
35003   f6het0kvylks=77&rru2xjsock_streamwt4=auaru2c29...   valid
35005   4ncagon66btji=3813&u9ytotln0l4ck=61&flywd2u=19...   valid
45372 rows × 2 columns
```

Pretty cool dataset for machine learning model !

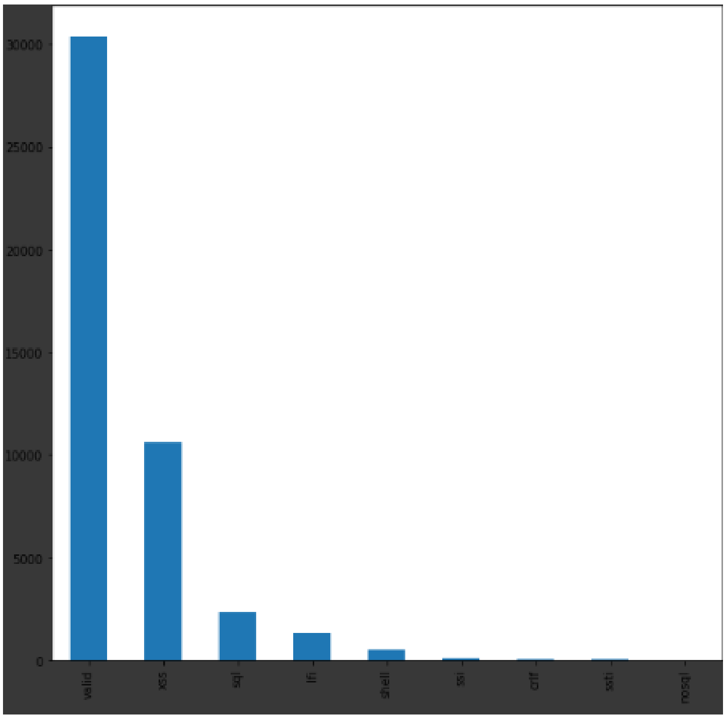
- Save dataset as a csv

```

clean_df.to_csv ('waf_dataset.csv', index = None)
```

Let's check the plot

- Chart



Traning Model

Choosing the best classification

- Note: All classifications are not show in this paper this paper only consists the best classification found with the above the dataset with 98-99 % accuracy and the rest is left for the users to test further work if they learn something form this paper.

Choosing the best parameters

The model is trained with SVM classifier using TF-IDF features and by utilizing GridSearchCV for best params

- Prepare training and testing sets

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd

#Load our dataset
waf_df = pd.read_csv('waf_dataset.csv')
X = waf_df['Payloads'].to_numpy().astype(str)
y = waf_df['Types'].to_numpy().astype(str)
print(len(X))
print(len(y))

#Prepare traning and testing sets
trainX, testX, trainY, testY = train_test_split(X, y, test_size = 0.25, random_state = 42, stratify = y)

np.savez('dataset', trainX=trainX, testX=testX, trainY=trainY, testY=testY)
```

- Train the model

```
pipe = make_pipeline(TfidfVectorizer(input = 'content', lowercase = True, analyzer = 'char', max_features = 1024), SVC())

grids = {'tfidfvectorizer__ngram_range': [(1, 1), (1, 2), (1, 4)], 'svc__C': [1, 10], 'svc__kernel': ['linear', 'rbf']}

grid = GridSearchCV(pipe, grids, cv = 2, verbose = 4)

grid.fit(trainX, trainY)
```

- Model Score

```
grid.score(testX, testY)
```

```
0.9972670369390814
```

We have pretty good accuracy here !

Now: - Best params found are:

```
 {'svc__C': 10, 'svc__kernel': 'rbf', 'tfidfvectorizer__ngram_range': (1, 2)}
```

And Let's Train the model

Traning the model with best params

- Whole code:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd

#Load our dataset
waf_df = pd.read_csv('waf_dataset.csv')
X = waf_df['Payloads'].to_numpy().astype(str)
y = waf_df['Types'].to_numpy().astype(str)
print(len(X))
print(len(y))

#Prepare traning and testing sets
trainX, testX, trainY, testY = train_test_split(X, y, test_size = 0.25, random_state = 42, stratify = y)
np.savez('dataset', trainX=trainX, testX=testX, trainY=trainY, testY=testY)

#Train the model with best params
pipe = make_pipeline(TfidfVectorizer(input = 'content', lowercase = True, analyzer = 'char', max_features = 1024, ngram_range = (1,
pipe.fit(trainX, trainY)
```

Storing the model

- Code used:

```
#Save the model for future use
import pickle

filename = 'waf_model.sav'

pickle.dump(model, open(filename, 'wb'))
```

Testing the model

Let's firt import the model which we stored

- Code used:

```
 import pickel
loaded_model = pickle.load(open('waf_model.sav', 'rb'))
```

Now let's test it !

- Code used:

```
#List of payloads to test waf model
parameters = ["%3f%0dshivang:crlf=injection", "query=home&homeprice=4300","#shivang{{5*7}}","<pre><!--#exec cmd=\"id\"--></pre>","..
              "something;|id|", "{$gt: ''}",
              "<img src=x onerror=\"&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#0000114&#0000105&#0000112&#0000116&#0000
temp_array = []

#Function acts as backend for payload detection

def waf_check(parameters, temp_array):
  for detect in range(len(parameters)):
    temp_array.append(parameters[detect])
    prediction = loaded_model.predict(temp_array)
    if "valid" in prediction:
      print("\n[+] You can access our site!\n")
    else:
      print("[!] Attack detected!...Hold your horses!")
      for result in prediction:
        print(f"[~] Attack type", result)
    temp_array = []

#Call the api

waf_check(parameters, temp_array)
```

- Output:

```
 [!] Attack detected!...Hold your horses!
[~] Attack type crlf

[+] You can access our site!

[!] Attack detected!...Hold your horses!
[~] Attack type ssti
[!] Attack detected!...Hold your horses!
[~] Attack type ssi
[!] Attack detected!...Hold your horses!
[~] Attack type lfi
[!] Attack detected!...Hold your horses!
[~] Attack type sql
[!] Attack detected!...Hold your horses!
[~] Attack type shell
[!] Attack detected!...Hold your horses!
[~] Attack type nosql
[!] Attack detected!...Hold your horses!
[~] Attack type xss
```

Get the above codes as notebook

Code

Conclusion

Machine Learning can be easily intergrated with cyber world to improve security and it's opposite it aslo true...which is what my next research paper will be on !