REVERSE PROXY WITH APPLIED MACHINE LEARNING IN WEB APPLICATION FIREWALL

CS6301 Machine Learning Laboratory Project

E. Blessing charles - 2019103012
RR. Kishore - 2019103607
M . Manoj kumar - 2019103033

INTRODUCTION :

Heat Blaster is a reverse proxy which can act as a load balancer and web application firewall , which is powered by the ML model to detect malicious http requests and prevent it by reaching the backend servers . The web application firewall is developed in a layered architecture , each layer has a ML model which has its own functionality to analyse the http request and forward it to the next layer. As the WAF follows layered architecture so we can enable or disable any layer according to our wish . The proxy server also logs both normal and malicious request information in log files for future analysis . The load balancer in the reverse proxy is implemented in round robin fashion and it also keeps a log file for uptime for each backend server .

Reverse Proxy :

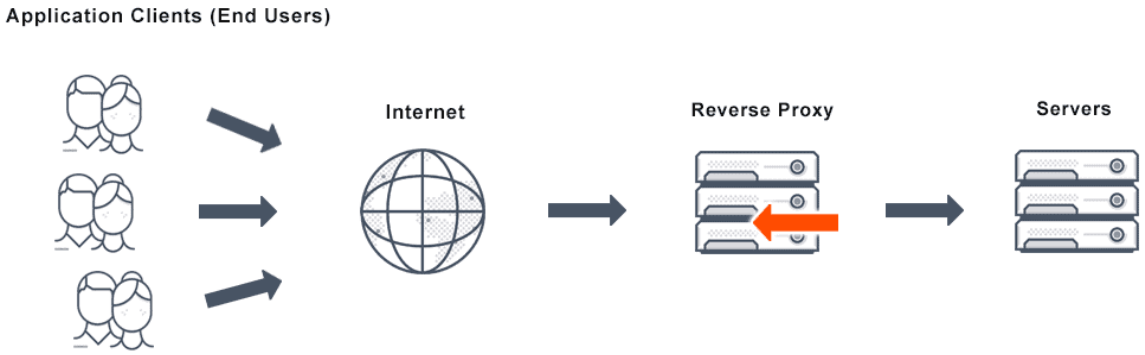
A Reverse Proxy Server, sometimes also called a reverse proxy web server, often a feature of a load balancing solution, stands between web servers and users, similar to a forward proxy. However, unlike the forward proxy which sits in front of users, guarding their privacy, the reverse proxy sits in front of web servers, and intercepts requests. In other words, a reverse proxy acts on behalf of the server, while a proxy acts for the client.

A reverse proxy server acts like a middleman, communicating with the users so the users never interact directly with the origin servers. It also balances client requests based on location and demand, and offers additional security.

Benefits of reverse proxy servers include:

- load balancing

- caching content and web acceleration for improved performance
- more efficient and secure SSL encryption, and
- protection from DDoS attacks and web application firewall

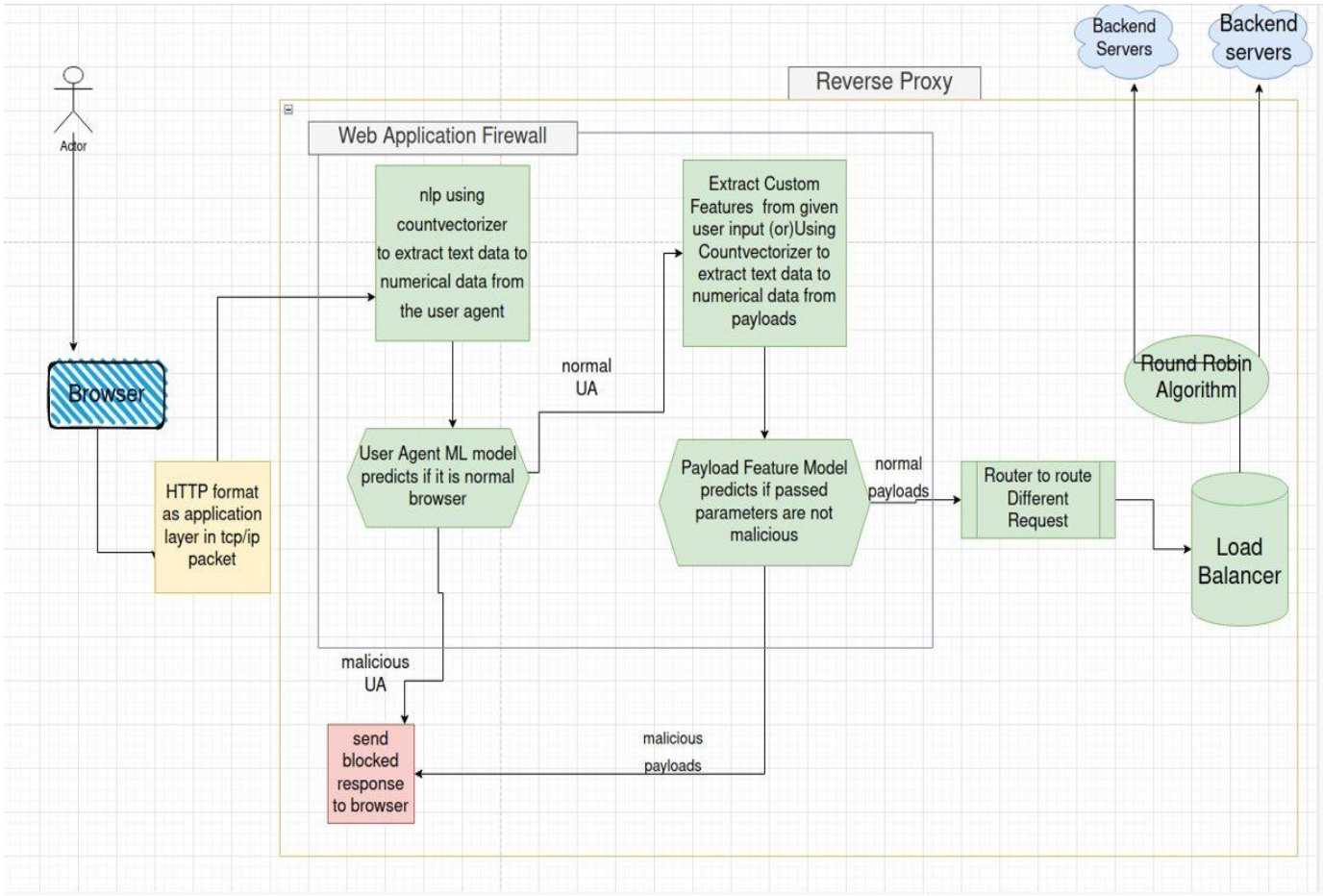


Web application Firewall

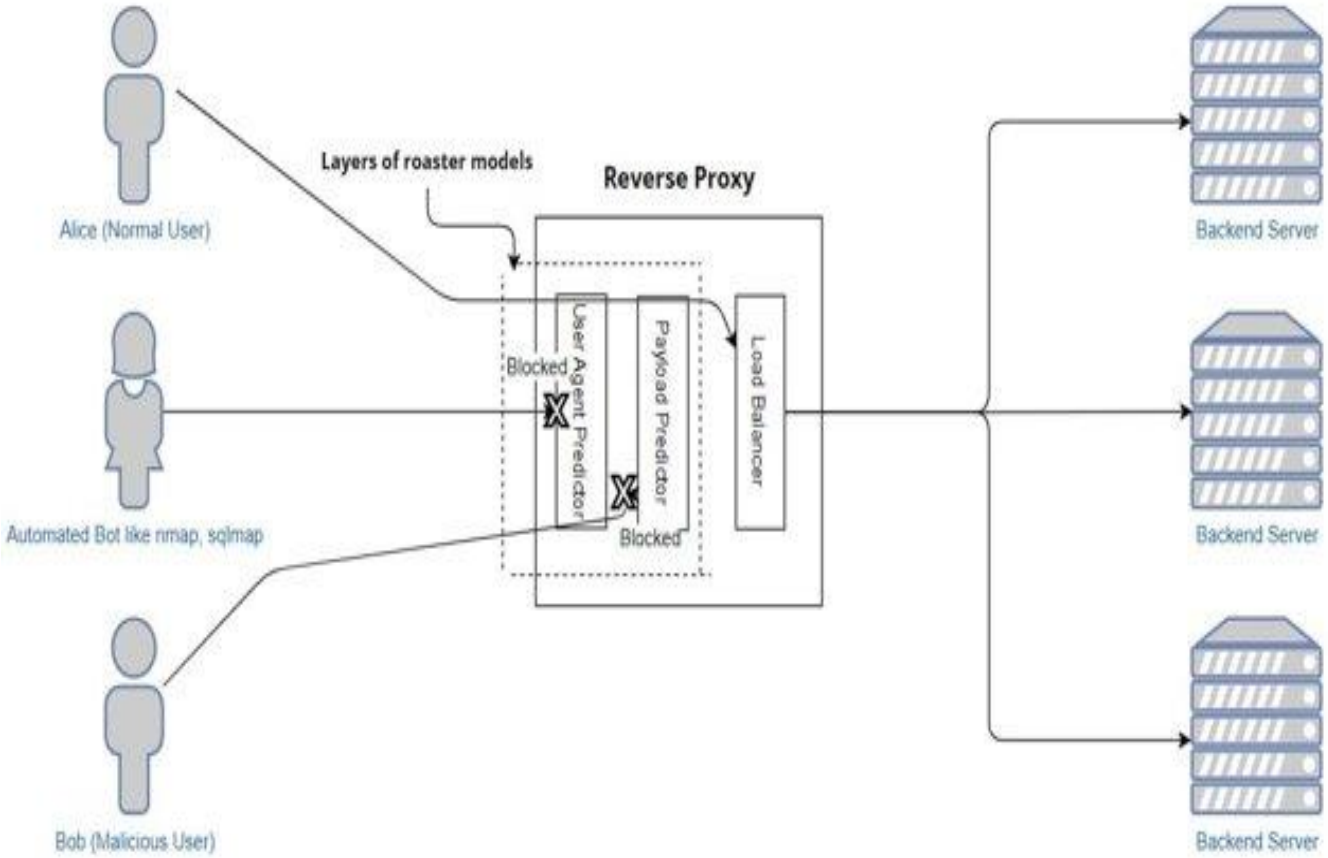
A WAF or web application firewall helps protect web applications by filtering and monitoring HTTP traffic between a web application and the Internet. It typically protects web applications from attacks such as csrf, xss, file inclusion, and sql injection, among others. A WAF is a protocol layer 7 defense (in the osi model), and is not designed to defend against all types of attacks. This method of attack mitigation is usually part of a suite of tools which together create a holistic defense against a range of attack vectors.

By deploying a WAF in front of a web application, a shield is placed between the web application and the Internet. While a proxy server protects a client machine's identity by using an intermediary, a WAF is a type of reverse proxy, protecting the server from exposure by having clients pass through the WAF before reaching the server.

BLOCK DIAGRAM



ARCHITECTURE DIAGRAM



Complete Flow

The http request send by the user is first processed by each layer in the web application firewall . All these models are highly configurable like which algorithm to use , or a layer can be completely disabled as per wish of the developer .

The Two Layers of WAF :

1) User Agent Classifier
2) Payloads Detector

User Agent Classifier :

The first ml model is used to analyze the user agent of the request and classify it , if the request is generated by a normal browser it contains a user agent which follows from standardization , if it is generated by various security tools(sqlmap) or automated bots we can capture the request and block it . So the ml model can classify between if the user agent is malicious or not . Since user agents can be forged by hackers , it is not completely fool proof but it can protect from script kiddies and roast the request in the first layer itself .

Payload Predictor :

The second ML model is used to analyse the inputs given by the users from http parameters , query and http POST request body ,classify them if it is a normal input given by the user or if it is a malicious payload

from the hackers . The ML model is trained with malicious payload vectors for xss , sql , nosql , ssrf , xxe attacks . So the ML model is able to classify normal requests from malicious requests . The malicious requests are roasted in this layer and only normal requests are served to the backend .

Overall Algorithm

Get http data as input to reverse proxy :

```
Bog = CountVectorizer(user-agent);
If user_agent_model(Bog) is malicious:
        Block_request();
Else :
        Payload = feature_extractor(user_input);
        If payload_model(Payload) is malicious:
        Block_request();
        Else:
                Route_request()
                Load_Balance(backend_servers);
```

ML models Algorithms Available

The developer can configure which dumped model should be used in production environment by changing in the environment variable of the reverse proxy application . The classification algorithms supported are

1) Support Vector Classifier

2) Logistic Regression

3) Decision Tree

4) Multinomial Naïve Bayes

5) Random Forest

MODULES OVERVIEW

Tree View of modules

```
.
├── blasterDB
├── blasterModels
│   ├── payloads_models
│   └── user_agents_models
├── env
├── heatblaster
│   ├── build
```

```
|    ├── data_collection
|    ├── datasets
|    ├──ml_notebooks
├── README.md
├── requirements.txt
├── ReverseProxy
|    ├── load_balancer
|    ├── mod_request_handlers
|    ├── mod_roaster
|    ├── templates
|    └── utils
├── run.py
```

Reverse Proxy module :

It’s the Core module which co-operates the load balancing , routing and the web application firewall where the layers of ml model presents .

It further consists of sub modules

1) load_balancer : contains logic of how to load balance among the backend servers

2) mod_request_handlers : contains logic of how to route requests

3) mod_roaster : contains the layers of web application firewall

4) templates : html templates for the reverse proxy

5) utils : utility functions for all sub modules

HEAT BLASTER MODULE

It contains all the Machine Learning Model training functions and preprocessing steps in jupyter notebook . From here only ml models are dumped using joblib library , then they are loaded into reverse proxy according to environmental configurations mentioned by the developer .

BlasterDb :

It contains all the data needed for the reverse proxy , which actually acts similar to database to all the modules .

ENV :

It contains all the environment variables for configuring the reverse proxy , the configs are httpconfig , proxyconfig , mlconfig , serverconfig .

Blaster Models :

It contains all the dumped ml model in two different sub directories of

1) payload models

2) user agents models

The different MI models available for each layer:

User Agent Classifier(Layer 1) :

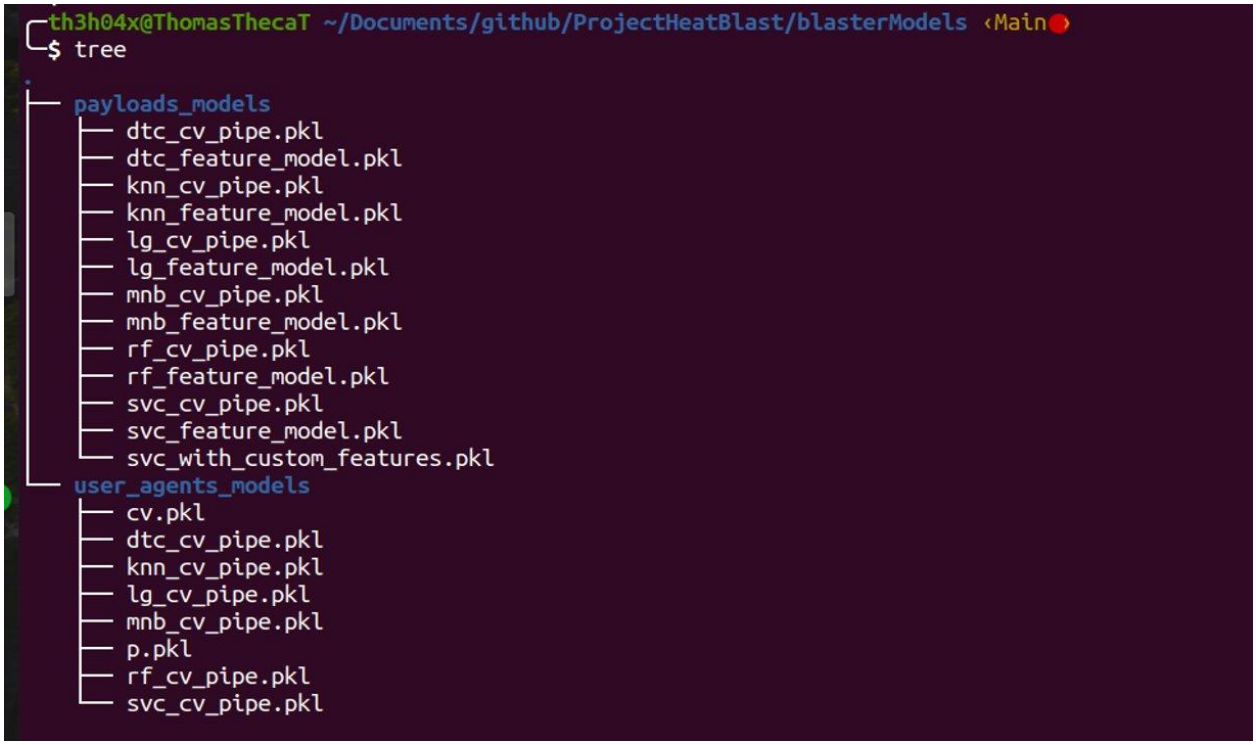
i) Count Vectorizer models

Payload Predictor

i) Custom Feature extraction (like special characters etc)

ii) Count Vectorizer models

LIST OF MODELS



Machine Learning in the Reverse Proxy

The heat blaster modules contains the jupyter notebooks of the machine learning models where the preprocessing steps and training of the model occurs . The trained model are then dumped into blasterModels directory , from where they can be loaded into the reverse proxy , which acts to filter out malicious requests .

Datasets Collection

User Agents Dataset

Common Browsers Dataset:

All common user agents names are collected from various github pages

1) https://gist.github.com/pzb/b4b6f57144aea7827ae4

2)https://raw.githubusercontent.com/yusuzech/top-50-user-agents/master/user_agent.csv

3) https://www.networkinghowtos.com/howto/common-user-agent-list/

4) https://github.com/N0taN3rd/userAgentLists

Malicious Tools Dataset :

1) https://github.com/herrbischoff/user-agents/tree/master/data

Payloads Dataset

Common Payloads :

1) https://www.kaggle.com/dhruvildave/google-trends-dataset

Malicious Payloads :

1) https://github.com/payloadbox/xss-payload-list

2)https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Request%20Forgery

3) https://github.com/foospidy/payloads

All the Combined datasets can be found in the project url

https://github.com/blessingcharles/ProjectHeatBlaster

Preprocessing Steps :

UserAgents Model Preprocessing Steps

The useragent is extracted from the http headers of the http request which is then converted into bag of words using countvectorizer .

CountVectorizer tokenizes (tokenization means dividing the sentences in words) the text along with performing very basic preprocessing. It

removes the punctuation marks and converts all the words to lowercase. The vocabulary of known words is formed which is also used for encoding unseen text later. The vocabulary of known words is formed which is also used for encoding unseen text later.

CountVectorizer is a great tool provided by the scikit-learn library in Python. It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text. This is helpful when we have multiple such texts, and we wish to convert each word in each text into vectors

Payloads Model Preprocessing Steps

The normal payloads inputs are collected from google trends dataset from kaggle and the malicious payloads are collected from different payloads of xss , sql , ssrf , xxe attacks used by security researchers and pentesters in the wild . There are two types of feature extraction is done in the payload data , the first one is similar to the previous user agent model ie. count vectorizer and the second one is getting the characters count which are not normally found in normal user inputs .

The custom Features are

1. `Length`
2. `Nonprintable_keywords`
3. `Special_chars`
4. `Punctuation_chars`
5. `Js_events`
6. `Html_tags`
7. `Sql_keywords`
8. `Percentage_count`
9. `spaces_count`

The developer can configure any models from these nlp preprocessing and custom feature preprocessing models in the reverse proxy , as all the 6 classification algorithms are trained in this two different datasets which leads to 12 different models available for the developer to choose .

The Python Class to extract custom features

```
class PayloadSpecialFeatureExtractor:

   def __init__(self , payload_dataframe : pd.DataFrame , payloads_path : str = "Payloads/"):
       self.payload_dataframe = payload_dataframe
       self.payloads_path = payloads_path
       self.js_event_keywords = []
       self.html_tags = []
       self.sql_keywords = []
       self.get_features_info()

   def get_features_info(self):

       JS_EVENTS_FILE_PATH = "keywords/js_events.txt"
       HTML_TAGS_FILE_PATH = "keywords/html_tags.txt"
       SQL_KEYWORDS_PATH = "keywords/sql_keywords.txt"

       #js events
       with open(self.payloads_path+JS_EVENTS_FILE_PATH) as f:
           for l in f.readlines():
               self.js_event_keywords.append(l.strip().lower())

       #html tags
```

```
        with open(self.payloads_path+HTML_TAGS_FILE_PATH) as f:
            for l in f.readlines():
                self.html_tags.append(l.strip().lower())

        #sql keywords
        with open(self.payloads_path+SQL_KEYWORDS_PATH) as f:
            for l in f.readlines():
                self.sql_keywords.append(l.strip().lower())

    def fit(self):

        self.payload_dataframe["length"] =
self.payload_dataframe["payloads"].apply(lambda payload :
self.extract_payload_length(payload))
        self.payload_dataframe["nonprintable_keywords"] =
self.payload_dataframe["payloads"].apply(lambda payload :
self.extract_nonprintable_keywords(payload))
        self.payload_dataframe["special_chars"] =
self.payload_dataframe["payloads"].apply(lambda payload :
self.extract_special_chars(payload))
        self.payload_dataframe["punctuation_chars"] =
self.payload_dataframe["payloads"].apply(lambda payload :
self.extract_punctuation_chars(payload))
        self.payload_dataframe["js_events"] =
self.payload_dataframe["payloads"].apply(lambda payload :
self.extract_js_events(payload))
        self.payload_dataframe["html_tags"] =
self.payload_dataframe["payloads"].apply(lambda payload :
self.extract_html_tags(payload))
        self.payload_dataframe["sql_keywords"] =
self.payload_dataframe["payloads"].apply(lambda payload :
self.extract_sql_keywords(payload))
```

```
        self.payload_dataframe["percentage_count"] =
self.payload_dataframe["payloads"].apply(lambda payload :
self.percentage_count(payload))
        self.payload_dataframe["spaces_count"] =
self.payload_dataframe["payloads"].apply(lambda payload :
self.spaces_count(payload) )

    def extract_special_chars(self , payload : str) -> int:
        count = 0
        for c in payload:
            if c in string.punctuation:
                count += 1
        return count

    def extract_nonprintable_keywords(self , payload : str) ->
int:
        count = 0
        for c in payload:
            if c not in string.printable:
                count += 1
        return count

    def extract_punctuation_chars(self , payload ):
        count = 0
        for c in payload:
            if c in string.punctuation:
                count += 1
        return count

    def extract_js_events(self , payload : str):
        count = 0
        for word in payload.split():
            for event in self.js_event_keywords:
                if event.lower() in word.lower():
```

```
                    count += 1

        return count

    def extract_html_tags(self , payload : str):
        count = 0

        for word in payload.split():
            word = urllib.parse.unquote(word).lower()
            for tag in self.html_tags:
                a = f""
                if
re.search(f".*(%3C|<).*{tag.lower()}.*(>|%3E).*" , word):
                    count += 1

        return count

    def extract_sql_keywords(self , payload : str):
        count = 0

        for word in payload.split():
            word = urllib.parse.unquote(word).lower()
            for sql_word in self.sql_keywords:
                if word in sql_word:
                    count += 1

        return count

    def extract_payload_length(self,payload : str) -> int:
        return len(payload)

    def percentage_count(self , payload : str) -> int:
        count = 0
```

```
        for c in payload:
            if c == "%":
                count += 1
        return count

    def spaces_count(self , payload : str ) -> str :
        count = 0
        for c in payload:
            if c == " ":
                count += 1
        return count

obj = PayloadSpecialFeatureExtractor(df)
obj.fit()
# df
```

Training the models

As there are 18 models to train , 6 models for user agents , 6 count vectorizer models for payloads and 6 custom feature payload models to train .The algorithm are imported from sklearn library and they are dumped using joblib library . The classification algorithms used in the project are

1. Support vector classifier
2. Logistic Regression
3. K nearest neighbours
4. Decision Tree
5. Random Forest
6. Multinomial Naive Bayes

Hyper Parameter Tuning

In machine learning, hyperparameter optimization or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are learned.

Since it will takes lot of slides , the results can be seen in the project github
https://github.com/blessingcharles/ProjectHeatBlaster

Model Evaluation Metrics :

When it comes to evaluating a Binary Classifier, Accuracy is a well-known performance metric that is used to tell a strong classification model from one that is weak. Accuracy is, simply put, the total proportion of observations that have been correctly predicted. There are four (4) main components that comprise the mathematical formula for calculating Accuracy, viz. TP, TN, FP, FN, and these components grant us the ability to explore other ML Model Evaluation Metrics. The formula for calculating accuracy is as follows:

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

- TP represents the number of True Positives. This refers to the total number of observations that belong to the positive class and have been predicted correctly.
- TN represents the number of True Negatives. This is the total number of observations that belong to the negative class and have been predicted correctly.
- FP is the number of False Positives. It is also known as a Type 1 Error. This is the total number of observations that have been predicted to belong to the positive class, but instead, actually, belong to the negative class.
- FN is the number of False Negatives. It may be
  referred to as a Type 2 Error. This is the total number of observations that
  have been predicted to be a part of the negative class but instead belong to
  the positive class.

The main reason for individuals to utilize the Accuracy Evaluation Metric is for ease of use. This Evaluation Metric has a simple approach and explanation. It is, as discussed before, simply the total proportion (total number) of observations that have been predicted correctly. Accuracy, however, is an Evaluation Metric that does not perform well when the presence of imbalanced classes-when in the presence of imbalanced classes, Accuracy suffers from a paradox; i.e., where the Accuracy value is high but the model lacks predictive power and most, if not all, predictions are going to be incorrect.

For the above reason, when we are unable to use the Accuracy Evaluation Metric, we are compelled to turn to other evaluation metrics in the scikit-learn arsenal. These include, but are not limited to, the following Evaluation Metrics:

Precision

This refers to the proportion (total number) of all observations that have been predicted to belong to the positive class and are actually positive. The formula for Precision Evaluation Metric is as follows:

$$Precision = \frac{TP}{TP + FP}$$

Recall

This is the proportion of observation predicted to belong to the positive class, that truly belongs to the positive class. It indirectly tells us the model's ability to randomly identify an observation that belongs to the positive class. The formula for Recall is as follows:

$$Recall = \frac{TP}{TP + FN}$$

F1 Score.

This is an averaging Evaluation Metric that is used to generate a ratio. The F1 Score is also known as the Harmonic Mean of the precision and recall Evaluation Metrics. This Evaluation Metric is a measure of overall correctness that our model has achieved in a positive prediction environment-

i.e., Of all observations that our model has labeled as positive, how many of these observations are actually positive. The formula for the F1 Score

Evaluation Metric is as follows:

$$F1 - score = \frac{2 * Precision * Recall}{Precision + Recall}$$

User Agents Model Evaluation Metrics :

```
[+] Classification report

K nearest neighbours :
              precision    recall  f1-score   support

           0       1.00      0.92      0.96       215
           1       0.87      1.00      0.93       110

    accuracy                           0.95       325
   macro avg       0.93      0.96      0.94       325
weighted avg       0.95      0.95      0.95       325
```

```
Multinomial Naive Bayes :
              precision    recall  f1-score   support

           0       0.68      1.00      0.81       215
           1       1.00      0.09      0.17       110

    accuracy                           0.69       325
   macro avg       0.84      0.55      0.49       325
weighted avg       0.79      0.69      0.59       325

Support Vector Classifier :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       215
           1       0.99      1.00      1.00       110

    accuracy                           1.00       325
   macro avg       1.00      1.00      1.00       325
weighted avg       1.00      1.00      1.00       325


Decision Tree Classifier :
              precision    recall  f1-score   support

           0       1.00      0.99      1.00       215
           1       0.98      1.00      0.99       110

    accuracy                           0.99       325
   macro avg       0.99      1.00      0.99       325
weighted avg       0.99      0.99      0.99       325

Random Forest Classifier :
              precision    recall  f1-score   support
```

```
              0       1.00      1.00      1.00       215
              1       0.99      1.00      1.00       110

       accuracy                           1.00       325
      macro avg       1.00      1.00      1.00       325
   weighted avg        1.00      1.00      1.00       325

Logistic Regression :
                  precision    recall  f1-score   support

              0       1.00      1.00      1.00       215
              1       0.99      1.00      1.00       110

       accuracy                           1.00       325
      macro avg       1.00      1.00      1.00       325
   weighted avg        1.00      1.00      1.00       325
```

Payloads Custom Feature Models Evaluation Metrics

```
[+] Classification Report

K nearest neighbours :

                precision    recall  f1-score   support

              0       1.00      1.00      1.00      5786
              1       1.00      1.00      1.00      6688
```

```
   accuracy                           1.00     12474
  macro avg       1.00      1.00      1.00     12474
weighted avg       1.00      1.00      1.00     12474


SVC

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5786
           1       1.00      1.00      1.00      6688

   accuracy                           1.00     12474
  macro avg       1.00      1.00      1.00     12474
weighted avg       1.00      1.00      1.00     12474


multinomial naive bayes

              precision    recall  f1-score   support

           0       1.00      0.98      0.99      5786
           1       0.98      1.00      0.99      6688

   accuracy                           0.99     12474
  macro avg       0.99      0.99      0.99     12474
weighted avg       0.99      0.99      0.99     12474


Decision Tree

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5786
```

```
           1       1.00      1.00      1.00      6688

    accuracy                           1.00     12474
   macro avg       1.00      1.00      1.00     12474
weighted avg       1.00      1.00      1.00     12474


Random Forest

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5786
           1       1.00      1.00      1.00      6688

    accuracy                           1.00     12474
   macro avg       1.00      1.00      1.00     12474
weighted avg       1.00      1.00      1.00     12474
```

Payloads Count Vectorizer Models evaluation Metrics

```
Logistic Regression

              precision    recall  f1-score   support

           0       1.00      0.71      0.83      4854
           1       0.80      1.00      0.89      5541

    accuracy                           0.86     10395
```

macro avg	0.90	0.85	0.86	10395
weighted avg	0.89	0.86	0.86	10395

Support Vector Classifier

	precision	recall	f1-score	support
0	0.67	0.98	0.80	3312
1	0.99	0.78	0.87	7083
accuracy			0.84	10395
macro avg	0.83	0.88	0.83	10395
weighted avg	0.89	0.84	0.85	10395

Decision Tree Classifier

	precision	recall	f1-score	support
0	1.00	0.53	0.69	9152
1	0.22	0.99	0.36	1243
accuracy			0.59	10395
macro avg	0.61	0.76	0.53	10395
weighted avg	0.91	0.59	0.65	10395

Random Forest Classifier

	precision	recall	f1-score	support
0	1.00	0.53	0.69	9158
1	0.22	1.00	0.36	1237
accuracy			0.59	10395
macro avg	0.61	0.76	0.53	10395
weighted avg	0.91	0.59	0.65	10395

Multinomial Naive Bayes

	precision	recall	f1-score	support
0	1.00	0.85	0.92	4854
1	0.89	1.00	0.94	5541
accuracy			0.93	10395
macro avg	0.94	0.93	0.93	10395
weighted avg	0.94	0.93	0.93	10395

K nearest Neighbours

	precision	recall	f1-score	support
0	0.48	1.00	0.65	4854
1	1.00	0.06	0.12	5541
accuracy			0.50	10395
macro avg	0.74	0.53	0.39	10395
weighted avg	0.76	0.50	0.37	10395

PROJECT DEMO WORKING

Reverse Proxy starts at port 3000

```
th3h04x@ThomasThecaT ~/Documents/github/ProjectHeatBlast ‹Main●›
$ python3 run.py
[INFO] 2022-01-05 13:30:23,111 : Server started at 0.0.0.0 port : 30
 * Serving Flask app 'ReverseProxy' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a product
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a product
 * Running on http://10.10.79.103:3000/ (Press CTRL+C to quit)
 * Restarting with stat
[INFO] 2022-01-05 13:30:24,097 : Server started at 0.0.0.0 port : 30
 * Debugger is active!
 * Debugger PIN: 121-907-728
```

Demo backend server starts at port 5000

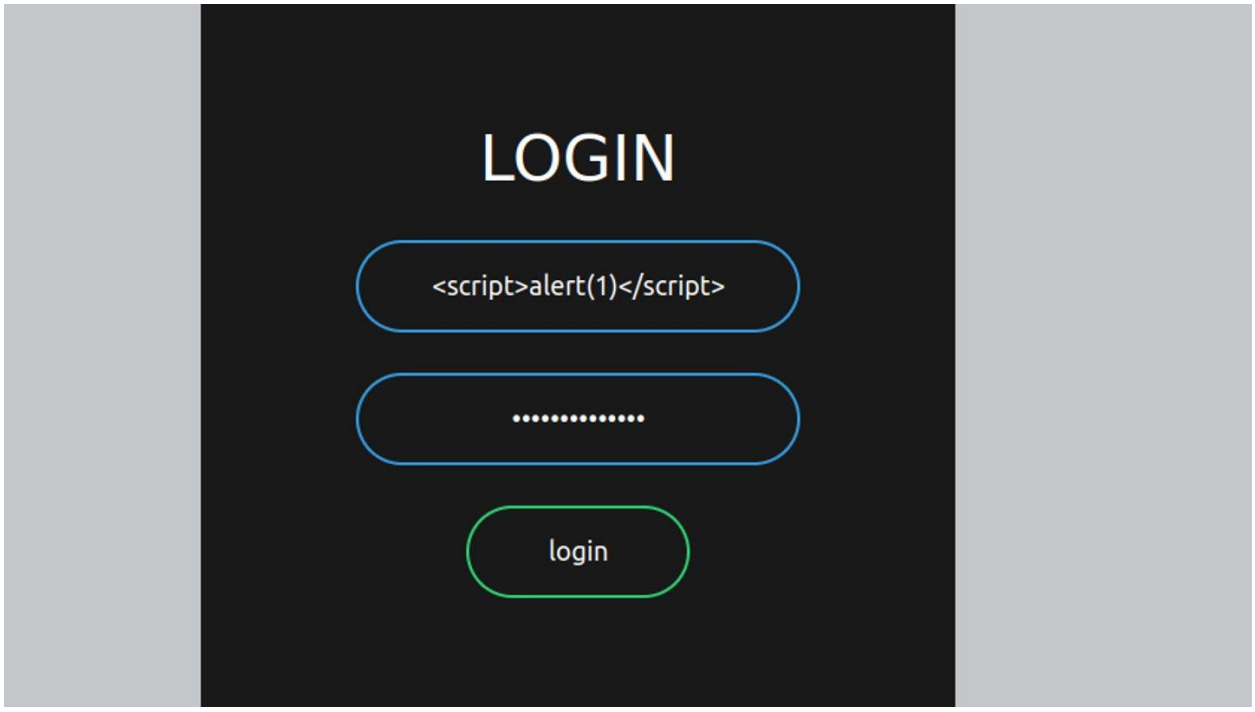
```
th3h04x@ThomasThecaT ~/Documents/github/ProjectHeatBlast/testApp ‹Main●›
$ nodemon app.js
[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
[+]Server Started at 5000
```

Demo Home Page

The client access the backend server via the reverse proxy at port 5000 which analyze the http request and forwards it to the backend server .

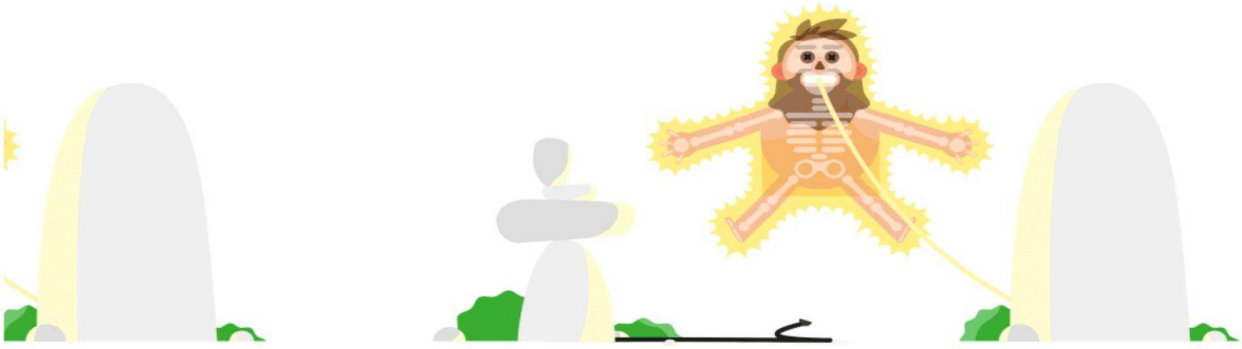
Login Page

Attacker injecting xss payloads in the user name field



The user got blocked by the web application firewall

403 FORBIDDEN



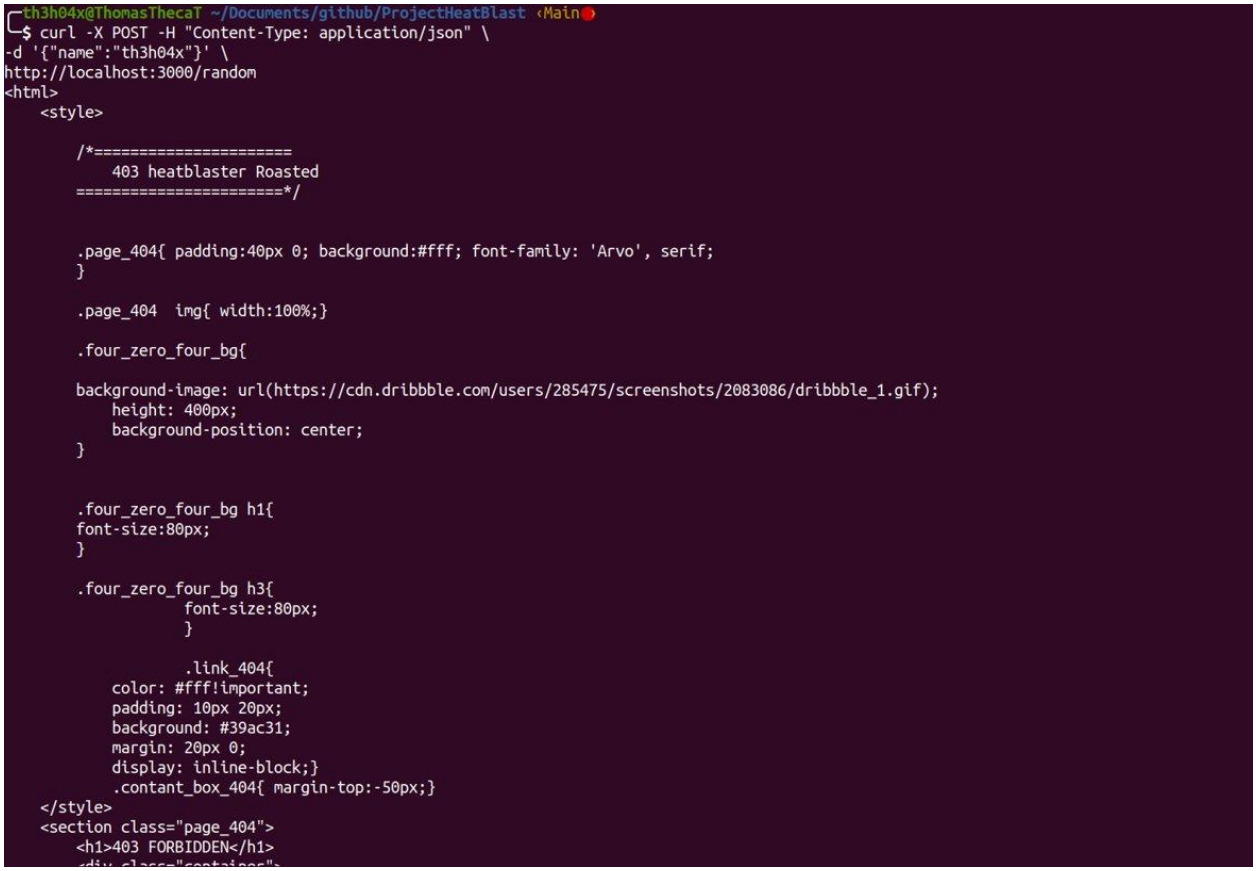
You are Request seems not to be legitimate

You are blocked by Web Application Firewall

’

Security tools and automated scanners also be blocked by the reverse proxy

Curl request got blocked



Comparison With Base Paper

The Base paper uses nlp technique of Count Vectorizer ie . bag of words to extract features from the Payloads , but we use custom feature extractor from the payloads to get the datasets , this increases increases the performances of our model .

The Web application firewall in the heat blaster reverse proxy is more robust than the base paper by the following :

1) It consists of layered architecture protection (user agent model and payload model) , which is lacked in the base paper
2) The reverse proxy is highly configurable , it can be used in real world

3) The feature extraction for payloads model is done in both bag of words model as of in base paper and also custom feature extraction which performs very well than the bag of words technique
4) The models are trained in relatively new attack vectors released by security researchers , but the base paper uses very old datasets in 2007 .
5) The base paper is just a web application firewall but heat blaster act as a reverse proxy of routing , load balancing and also as a web application firewall

Table II
SUMMARIZED RESULTS OF THE MODSECURITY BASELINE AND THE THREE SCENARIOS PRESENTED

Scenario	Dataset	Algorithm	Precision	Recall	TPR	FPR
Baseline	DRUPAL	MODSECURITY W/OWASP CRS	0.11	0.76	76.22%	38.89%
	PKDD2007	MODSECURITY W/OWASP CRS	0.70	0.93	92.97%	57.21%
	CSIC2010	MODSECURITY W/OWASP CRS	0.50	0.34	34.32%	23.93%
*sc*1	DRUPAL	Random Forest	0.97	0.91	91.22%	0.05%
		KNN-3	0.97	0.89	88.97%	0.05%
		SVM	0.98	0.90	89.67%	0.04%
	PKDD2007	Random Forest	0.96	0.85	85.10%	1.67%
		KNN-3	0.96	0.70	70.39%	1.41%
		SVM	0.95	0.81	81.10%	1.91%
	CSIC2010	Random Forest	0.97	0.72	72.00%	1.47%
		KNN-3	0.95	0.65	65.03%	2.38%
		SVM	0.97	0.70	70.34%	1.26%
*sc*2	DRUPAL	Random Forest	0.95	0.48	47.57%	0.05%
		K-NN 3	0.90	0.07	6.99%	0.01%
	PKDD2007	Random Forest	0.96	0.23	22.56%	0.45%
		K-NN 3	1.00	0.12	11.70%	0.02%
	CSIC2010	Random Forest	0.91	0.32	32.06%	2.27%
		K-NN 3	0.66	0.50	50.43%	17.85%
*sc*3	DRUPAL	One-class w/λ=0.156 (test)	0.22	0.95	95.34%	22.15%
		One-class w/λ=0.156 (validation)	0.15	0.89	88.90%	26.83%
	PKDD2007	One-class w/λ=0.156 (test)	0.70	0.93	93.12%	58.30%
	CSIC2010	One-class w/λ=0.156 (test)	0.75	0.48	47.66%	11.15%

Evaluation metrics in custom feature payload models

```
[+] Classification Report

K nearest neighbours :

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5786
           1       1.00      1.00      1.00      6688
```

```
   accuracy                           1.00     12474
  macro avg       1.00      1.00      1.00     12474
weighted avg       1.00      1.00      1.00     12474


SVC

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5786
           1       1.00      1.00      1.00      6688

   accuracy                           1.00     12474
  macro avg       1.00      1.00      1.00     12474
weighted avg       1.00      1.00      1.00     12474


multinomial naive bayes

              precision    recall  f1-score   support

           0       1.00      0.98      0.99      5786
           1       0.98      1.00      0.99      6688

   accuracy                           0.99     12474
  macro avg       0.99      0.99      0.99     12474
weighted avg       0.99      0.99      0.99     12474


Decision Tree

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5786
```

```
           1       1.00      1.00      1.00      6688

    accuracy                           1.00     12474
   macro avg       1.00      1.00      1.00     12474
weighted avg       1.00      1.00      1.00     12474


Random Forest

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5786
           1       1.00      1.00      1.00      6688

    accuracy                           1.00     12474
   macro avg       1.00      1.00      1.00     12474
weighted avg       1.00      1.00      1.00     12474
```

REFERENCE PAPERS

1) Web Application Attacks Detection Using Machine Learning Techniques
https://ieeexplore.ieee.org/document/8614199

2) Development of a hybrid web application firewall to prevent web based attacks
https://ieeexplore.ieee.org/document/7035910